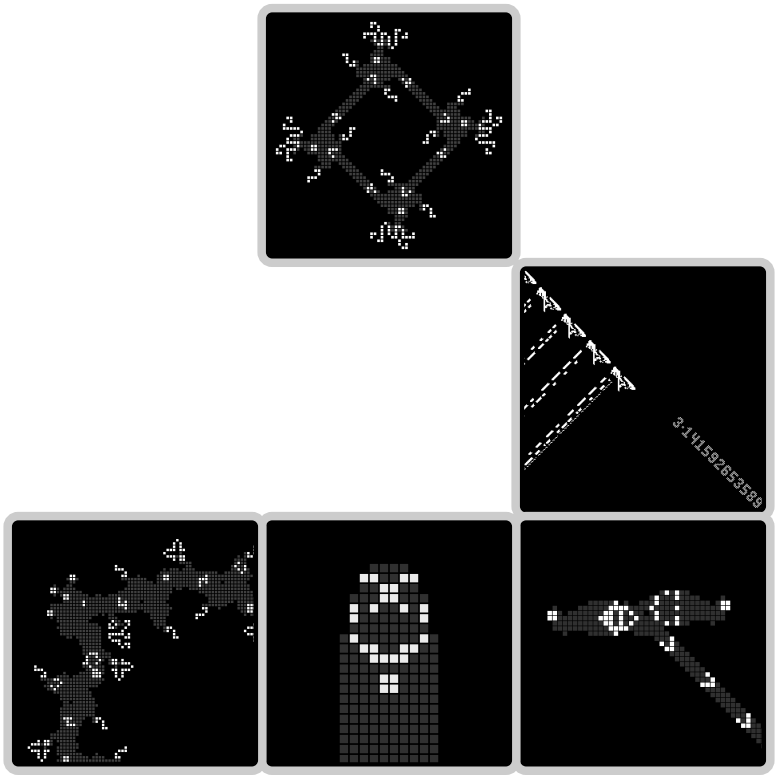
Conway's Game of Life

Mathematics and Construction



Nathaniel Johnston and **Dave Greene**

Free PDF. See conwaylife.com/book for pattern files and print version.

Copyright © 2022 Nathaniel Johnston and Dave Greene

CONWAYLIFE.COM/BOOK

9 781794 816961

Free PDF. See conwaylife.com/book for pattern files and print version.

To John Horton Conway
For giving us 50 years of Life.

Free PDF. See conwaylife.com/book for pattern files and print version.

Free PDF. See conwaylife.com/book for pattern files and print version.



Contents

Preface xi
The Goal xi
Intended Audience xi
How to Use xii
Acknowledgments xiv

I **Classical Topics**

1 **Early Life** 3
1.1 Our First Technique: Random Fumbling 5
1.2 Common Evolutionary Sequences 8
1.3 The Queen Bee 10
1.4 The B-Heptomino and Twin Bees 11
1.5 The Switch Engine 13
1.6 Methuselahs and Stability 16
1.7 Gardens of Eden 20
1.8 Notes and Historical Remarks 26
Exercises 29

2 **Still Lifes** 33
2.1 Strict and Pseudo Still Lifes 34

Free PDF. See conwaylife.com/book for pattern files and print versio

2.2 Still Life Grammar 37
2.3 Eaters 39
2.4 Welded and Constrained Still Lifes 42
2.5 Still Life Density 44
2.6 Notes and Historical Remarks 49
Exercises 50

3 Oscillators 53
3.1 Billiard Tables 54
3.2 Stabilizing Corners 55
3.3 Composite Periods and Sparks 57
3.4 Hasslers and Shuttles 61
3.5 Glider Loops and Reflectors 65
3.6 Herschel Tracks 68
3.7 Omniperiodicity 73
3.8 Phoenices 75
3.9 Notes and Historical Remarks 77
Exercises 79

4 Spaceships and Moving Objects 83
4.1 The Glider 84
4.2 The Light, Middle, and Heavyweight Spaceships 87
4.3 Corderships 90
4.4 Puffers and Rakes 94
4.5 Speed Limits 98
4.6 Speed and Period Status 106
4.7 Notes and Historical Remarks 110
Exercises 113

II Circuitry and Logic

5 Glider Synthesis 121
5.1 Two-Glider Syntheses 122
5.2 Syntheses Involving Three or More Gliders 124
5.3 Incremental Syntheses 126
5.4 Synthesis of Moving Objects 129
5.5 Developing New Syntheses 131
5.6 A Gosper Glider Gun Breeder 134
5.7 Slow Salvo Synthesis 136

Free PDF. See conwaylife.com/book for pattern files and print version.

5.8 Notes and Historical Remarks 145
Exercises 148

6 Periodic Circuitry 153
6.1 Period 30 Circuitry 154
6.2 Primer 159
6.3 Period 46 Circuitry 163
6.4 Bumpers and Bouncers 171
6.5 Glider Timing and Regulators 172
6.6 Notes and Historical Remarks 175
Exercises 179

7 Stable Circuitry 183
7.1 Herschel Conduits 183
7.2 From Herschels to Gliders 188
7.3 From Gliders to Herschels 190
7.4 From Gliders to Gliders 190
7.5 Synthesizing Objects via Conduits 192
7.6 Period Multipliers and Small High-Period Guns 196
7.7 Converters for Other Objects 202
7.8 Factories 205
7.9 Notes and Historical Remarks 211
Exercises 214

8 Guns and Glider Streams 221
8.1 Glider Deletion 221
8.2 Glider Insertion 225
8.3 Streams of Other Spaceships 226
8.4 Glider Guns of Any Period 230
8.5 True-Period Guns 231
8.6 Slide Guns 245
8.7 Armless Guns 247
8.8 Slow and Irregular Guns 254
8.9 Notes and Historical Remarks 261
Exercises 264

III Constructions

9 Universal Computation 271
9.1 A Computer in Life 271

Free PDF. See conwaylife.com/book for pattern files and print version.

9.2 A Compiled APGsembly Pattern: Adding Registers 278
9.3 Multiplying and Reusing Registers 282
9.4 A Binary Register 284
9.5 A Character Printer 292
9.6 A π Calculator 296
9.7 A 2D Printer 301
9.8 Notes and Historical Remarks 306
Exercises 308

10 Self-Supporting Spaceships 311
10.1 The Silverfish 311
10.2 The Caterpillar 321
10.3 Oblique Helices and the Waterbear 327
10.4 Caterloopillars 330
10.5 Notes and Historical Remarks 337
Exercises 340

11 Universal Construction 345
11.1 Gemini and Geminoids 346
11.2 Single-Channel Glider Synthesis with a 90-Degree Elbow 353
11.3 Single-Channel Glider Synthesis with a Zero-Degree Elbow 359
11.4 Duplicating and Reflecting Single-Channel Recipes 362
11.5 A Slow Demonoid 367
11.6 A Middling Demonoid 369
11.7 A Fast Demonoid 371
11.8 Notes and Historical Remarks 377
Exercises 381

12 The 0E0P Metacell 385
12.1 Other 2D Cellular Automata 387
12.2 Rule Emulation 391
12.3 Overview of the Metacell 394
12.4 The Shell and Lane-Switching Circuitry 398
12.5 The Kernel 398
12.6 The Nucleus 402
12.7 Construction and Self-Destruction 402
12.8 A Complete Metageneration 404
12.9 Notes and Historical Remarks 422
Exercises 425

Free PDF. See conwaylife.com/book for pattern files and print version.

Appendices and Supplements

A Mathematical Miscellany 431

A.1 Modular Congruence 431

A.2 Greatest Common Divisor and Least Common Multiple 432

A.3 Big-Θ Notation 434

B Extra Details 437

B.1 Universality of the Clock Inserter 437

B.2 Snarkmaker Timings 440

B.3 Scorbie Splitter Maker Timings 441

B.4 Scorbie Splitter Destroyer Timings 442

B.5 Cordership Maker Timings 443

B.6 Isotropic Rulestrings 444

B.7 Non-Isotropic Rulestrings 446

B.8 π Calculator APGsembly Code 448

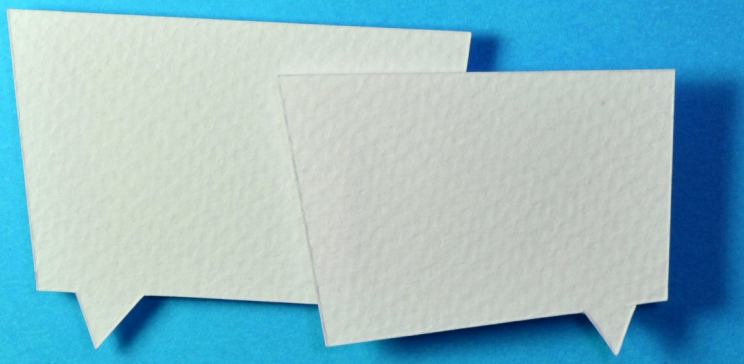
C Solutions to Selected Exercises 451

Bibliography 469

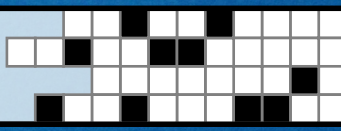
Index 473

Free PDF. See conwaylife.com/book for pattern files and print version.

Free PDF. See conwaylife.com/book for pattern files and print version.



Preface



The Goal

In this book, we provide an introduction to Conway's Game of Life, the mathematics behind it, and the methods used to construct many of its most interesting patterns. Lots of small "building block"-style patterns (especially in the first four or so chapters of this book) were found via brute-force or other computer searches, and we do not go into the details of how these searches were implemented. However, from that point on we try to guide the reader through the thought processes and ideas that are needed to combine those patterns into more interesting composite ones.

While we largely follow the history of the Game of Life as we go through the book, we emphasize that this is *not* its primary purpose. Rather, it is a by-product of the fact that most recently discovered patterns build upon patterns and techniques that were developed earlier. The goal of this book is to demystify the Game of Life by breaking down the complex patterns that have been developed in it into bite-size chunks that can be understood individually.

This book is up to date with regards to Life technology and results that were known as of January 15, 2022. However, new tools and techniques are discovered so frequently in the Game of Life that it will become somewhat out of date rather quickly.[1] A list of errata and notable discoveries since publication of the book can be found at `conwaylife.com/book`.

Intended Audience

While this book does not have any formal mathematical or computer science prerequisites, it is aimed at the level of a first-year undergraduate university student. Some high-school-level topics like logarithms, the "floor" function $f(x) = \lfloor x \rfloor$ for rounding numbers down, the binary representation of a positive integer, and summation notation like $\Sigma_{k=1}^{n} k^2$ for adding numbers, are used frequently and without much explanation. Perhaps the most sophisticated mathematical machinery we use is in Section 9.6, where we multiply some 2×2 matrices (but only upper triangular 2×2 matrices, and we

[1] There are at least three major discoveries mentioned in this book that were found between January 1–14, 2022.

Free PDF. See conwaylife.com/book for pattern files and print versio

provide the matrix multiplication formula).

A basic understanding of how mathematical proofs and computer programming work is also expected. That is, we expect a certain level of mathematical and computer science maturity from the reader, but no specialized knowledge of either topic. Throughout the book, we prove some simple theorems about the Game of Life. These proofs do not use any specialized proof techniques or expect the reader to have any specific university-level mathematical knowledge, but rather just expect the reader to be able to follow a logical argument. Similarly, we introduce a programming language for building computer programs out of Life circuits in Chapter 9, so that material will be easier to master if the reader has had prior exposure to computer programming.

Somewhat more advanced mathematical topics that we make use of are summarized in Appendix A, though they are typically introduced very gently in the main text as well, and we only require a very surface-level understanding of them. We make use of the greatest common divisor, the least common multiple, and Bézout's identity when discussing oscillator periods in Chapter 3, so we introduce these tools in Appendix A.2. Infinite series make brief appearances at the end of Chapter 6 and in Section 9.6, though the reader is not expected to really have any familiarity with them or to understand convergence issues. Finally, big-Θ notation is used to discuss the size and growth rate of patterns in Sections 7.6, 8.8, and 9.7.2, so we introduce this concept in Appendix A.3.

How to Use

Conway's Game of Life is an extremely visual game, so this book makes very liberal use of figures throughout, particularly when new patterns or techniques for creating patterns are introduced. However, the Game of Life is best observed in motion, which makes static images in a textbook less than ideal as learning tools. In order to help present the motion of patterns a bit better, we do five things:

- Figures are presented with alive cells in black and dead cells in white, but furthermore we use a gradient from blue to orange to denote cells that were alive in past generations of the pattern. Bright blue cells were just alive, whereas cells that are orange were alive in the more distant past (roughly 75 generations or more—see Figure 1).

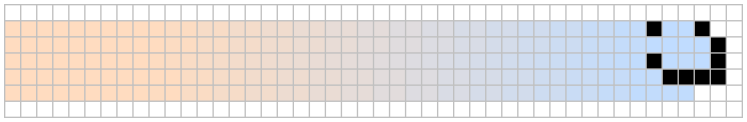


Figure 1: An object moving to the right with a gradient behind it indicating how long ago it was in each location. White cells were never alive, black cells are currently alive, blue cells were alive recently, and orange cells were alive long ago.

- If we really wish to emphasize what a pattern looks like in different generations, all generations of interest will be displayed, along with arrows that specify how many generations have passed. For example, if we want to clarify exactly what the pattern in Figure 1 does as it moves from left to right, we might display it as in Figure 2.

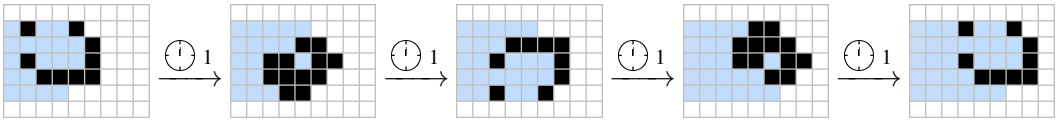


Figure 2: The same object as in Figure 1, but displayed in a more explicit fashion. This image shows how the pattern changes every time 1 generation elapses.

Free PDF. See conwaylife.com/book for pattern files and print version.

- We use colors to highlight various pieces of patterns, and we maintain a consistent coloring scheme throughout the book. Light pastel colors like aqua, magenta, light green, yellow, and light orange are used to highlight *around* objects—cells in these colors are dead, and they indicate a region in the Life plane consisting of cells that all serve some common purpose or logically make up one "object" (see Figure 3). On the other hand, darker colors like dark green, dark orange, and dark red are used to highlight certain live cells. Dark green is typically used to highlight the input to some reaction, while dark orange is typically used for the output of the reaction (see Figure 4).

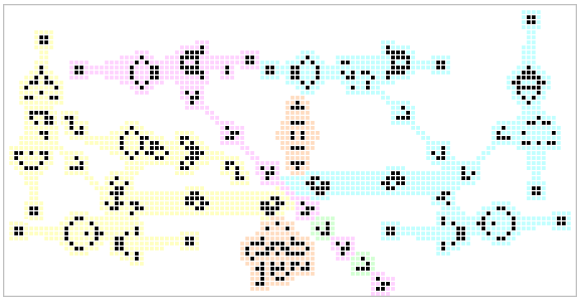


Figure 3: A glider gun made up of several different component reactions that are highlighted in different colors. In particular, gliders are created by a gun highlighted in magenta, lightweight spaceships are created by guns highlighted in aqua and yellow, and those lightweight spaceships are merged into the original glider stream by oscillators highlighted in light orange.

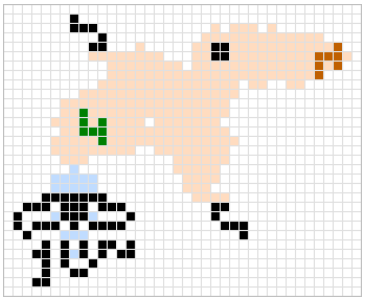


Figure 4: A dark green Herschel comes in from the left and creates the dark orange Herschel on the right. Cells highlighted in blue are constantly changing due to being part of an oscillator, whereas cells that are light orange are usually dead, except when the Herschel passes through.

- In the electronic version of this book, almost every figure is actually a clickable link that will open a text file containing RLE or Macrocell code for the displayed pattern (go ahead, click on any of the figures above, or even one of the five images on the cover page). This code can be copied and pasted into Life simulation software like Golly (`golly.sourceforge.net`) so that it can be explored and manipulated.[2] Note that clicking on figures may not work in certain PDF viewers (such as the viewers built into web browsers), so we recommend using Adobe Acrobat Reader (`get.adobe.com/reader`) to read this book digitally.
- RLE, LifeHistory, or Macrocell codes for all of the patterns displayed in the book are also available at the book's website (`conwaylife.com/book`). Furthermore, the patterns can be viewed and manipulated right on that website as well via any modern web browser, without downloading any additional software.

Exercises

We strongly encourage the reader to work through this book's many exercises that can be found at the end of each chapter. For this reason, it is extremely important to either download Life software or use the book's website to view and edit patterns while making your way through the book—Life is meant to be played, not just watched, and many of the exercises simply cannot be solved without the assistance of Life simulation software.

Roughly half of the exercises are marked with an asterisk ($*$), which means that they have a solution provided in Appendix C. Exercises also begin with a numeric difficulty estimate on a scale

[2] For an explanation of how RLE or Macrocell code works, see `conwaylife.com/wiki/Run_Length_Encoded` or `conwaylife.com/wiki/Macrocell`.

Free PDF. See conwaylife.com/book for pattern files and print version.

of 1 to 5, displayed within square brackets like [3/5] or [5/5]. Difficulty-1 exercises follow fairly immediately from the relevant definitions and results in the text, up to difficulty-4 exercises that involve either an extensive and delicate calculation, or a complicated multi-step construction. Difficulty-5 exercises are more like projects that could take a day or so of work to complete.

Acknowledgments

The authors are indebted to dozens of people who opened the world of Conway's Game of Life to them. Rather than acknowledging the people who discovered the reactions and patterns that we discuss here, they are credited in footnotes and figures throughout the book.

We extend thanks to Adam P. Goucher for numerous contributions to the writing of this book, especially in Chapter 12. Thanks to Nicolay Beluchenko, Oscar Cunningham, Steven Eker, Bill Gosper, Hartmut Holzwart, Tanner Jacobi, Matthias Merzenich, Daniel Mouscher, Mark Niemiec, David Raucci, Chris Rowett, Ville Salo, Rich Schroeppel, Michael Simkin, Connor Steppie, Satoshi Tanaka, Justin Tang, and Kalan Warusa, as well as "AlephAlpha", "Chris857", "dani", "Ian07", and many other ConwayLife.com forum users, for helpful conversations and corrections to early versions of this book.

Thanks to Andrew Trevorrow and Tomas Rokicki for creating the open-source cross-platform cellular automaton editor and simulator Golly (golly.sourceforge.net), without which many of the patterns discussed in this book would not have been discovered. Thanks to Chris Rowett for creating LifeViewer (lazyslug.com/lifeviewer), which is used on this book's website (conwaylife.com/book) to display patterns and make them interactive. Thanks to Velimir Gayevskiy and Mathias Legrand for the *Legrand Orange Book* LaTeX template from LaTeXTemplates.com.

Finally, the authors would like to thank their wives Kathryn and Melanie for tolerating them during their years of mental absence glued to this book, and their parents for encouraging them to care about both learning and teaching. Many thanks also to Mount Allison University for giving the first author the academic freedom to pursue a project like this one.

Free PDF. See conwaylife.com/book for pattern files and print version.

Classical Topics

1 Early Life 3
1.1 Our First Technique: Random Fumbling
1.2 Common Evolutionary Sequences
1.3 The Queen Bee
1.4 The B-Heptomino and Twin Bees
1.5 The Switch Engine
1.6 Methuselahs and Stability
1.7 Gardens of Eden
1.8 Notes and Historical Remarks
Exercises

2 Still Lifes 33
2.1 Strict and Pseudo Still Lifes
2.2 Still Life Grammar
2.3 Eaters
2.4 Welded and Constrained Still Lifes
2.5 Still Life Density
2.6 Notes and Historical Remarks
Exercises

3 Oscillators 53
3.1 Billiard Tables
3.2 Stabilizing Corners
3.3 Composite Periods and Sparks
3.4 Hasslers and Shuttles
3.5 Glider Loops and Reflectors
3.6 Herschel Tracks
3.7 Omniperiodicity
3.8 Phoenices
3.9 Notes and Historical Remarks
Exercises

4 Spaceships and Moving Objects 83
4.1 The Glider
4.2 The Light, Middle, and Heavyweight Spaceships
4.3 Corderships
4.4 Puffers and Rakes
4.5 Speed Limits
4.6 Speed and Period Status
4.7 Notes and Historical Remarks
Exercises

Free PDF. See conwaylife.com/book for pattern files and print version.



1. Early Life

I used to feel guilty in Cambridge that I spent all day playing games, while I was supposed to be doing mathematics. Then . . . I realized that playing games *is* math.

John H. Conway

Conway's Game of Life[1] is a process that takes place on an infinite square grid, where each square can be in one of two states: **alive** or **dead** (depicted via black and white squares, respectively, in this book). The squares (which we typically call **cells**) then evolve in discrete timesteps (called **generations** or **ticks**) according to the following two rules:

- If a cell is alive, it survives to the next generation if it has 2 or 3 live neighbors; otherwise it dies.
- If a cell is dead, it comes to life in the next generation if it has exactly 3 live neighbors; otherwise it stays dead.

We note that these rules are applied to every square in the grid simultaneously, and a "neighbor" in these rules refers to any of the 8 cells that it touches either along a side or at a corner, as in Figure 1.1. As an example of how these rules work, consider what happens to the straight line of 4 alive cells depicted in Figure 1.2. The leftmost and rightmost cells in the line both only have one live neighbor, so they die, while the two central cells above and below the line each have exactly 3 live neighbors, so they come to life. This leaves us with a 3×2 rectangle of live cells, so we say that the line of 4 live cells is a **parent** of the 3×2 rectangle of live cells, or equivalently that the 3×2 rectangle is a **child** of the line of 4 cells. We then apply the evolution rules again: this time, the two central cells in the rectangle are overpopulated and die, while the dead cells to their immediate left and right have exactly 3 live neighbors and thus come to life.

After this point, if we apply the evolution rule again, nothing changes; each of the live cells have

[1] Named for its inventor, John H. Conway.

Free PDF. See conwaylife.com/book for pattern files and print versio

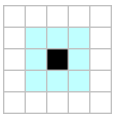


Figure 1.1: A live cell (in black) in the middle and its 8 neighbors (in aqua).

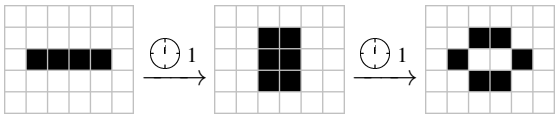


Figure 1.2: A line of four alive cells takes two generations to evolve into a stable object called a **beehive**. The 3×2 object in the middle is sometimes called a **pre-beehive**.

2 live neighbors and thus live to the next generation, and no dead cell has exactly 3 live neighbors, so they all stay dead. A pattern like this that remains unchanged from one generation to the next is called a **still life**.

As we saw in this example, there was no input on our part once the pattern started evolving: we just applied the game's rules and passively watched what happened from one generation to the next. Instead, the "game" in Life is the challenge of finding new and interesting patterns that evolve in unique and unexpected ways, and that is what this book is about. We will see patterns that move back and forth periodically between a finite number of different configurations, patterns that move through the Life grid over time, and patterns that create an infinitely growing family of other patterns. We will collide patterns with each other to create more complicated patterns, which we will then erase with even more patterns. We will construct logical circuitry with Life objects that allow us to simulate arbitrary computation within the Life universe, and we will construct patterns that do remarkable things like list the prime numbers or print out the decimal digits of π. And all of this will work simply based on the two simple life and death rules we described earlier.

This all leads to a very natural question: why those rules? Why not have a dead cell come to life only in the case that it has exactly 4 live neighbors? Why not have a live cell stay alive if it has exactly 1, 2, 4, or 7 live neighbors? Indeed, there are $2^{18} = 262\,144$ distinct **Life-like cellular automata**: rules that can be applied to a 2D square grid of alive and dead cells that simply depend on the numbers of live and dead neighbors that lead to a cell staying alive or coming to life. However, the following three properties make Life special (but by no means unique):

- Its rules are *simple*. For example, having a live cell stay alive if it has exactly 2 or 3 live neighbors is a more "natural" rule than having a live cell stay alive if it has exactly 1, 2, 4, or 7 live neighbors. This can be justified a bit by arguing that, to model something like a biological system, a cell should die of overcrowding if it has "too many" neighbors (e.g., since there won't be enough resources to support all of the live cells), and it should die of isolation if it has "too few" neighbors. The exact threshold for "too many" and "too few" is debatable, but pinned down at least somewhat by the next point.
- It strikes a balance between being *chaotic* and *stable*. Many rules (e.g., almost any rule in which a cell is born when it has 2 live neighbors) cause far too many births for anything to stabilize. These rules are thus "too chaotic", making it almost impossible for us to construct interesting objects. On the other hand, most rules in which cells are *not* born when they have 3 (or fewer) live neighbors typically lead to patterns dying off extremely quickly and thus being "too stable" to be interesting. Life strikes a good balance of patterns typically staying alive but not overtaking the entire grid.
- It is *historical*. This is perhaps a bit of an unsatisfying reason, but part of the interest in Life simply comes from the fact that historically it is the most well-studied rule, and it is fun to see how far we can push one very well-studied rule, rather than dividing our attention and making moderate progress on multiple rules.

Nonetheless, other birth and death rules are sometimes studied as well, and for brevity they are

Free PDF. See conwaylife.com/book for pattern files and print version.

typically described using a **rulestring** of the form `Bx/Sy`, where we replace "`x`" by all numbers of live neighbors that lead to the **b**irth of a dead cell, and we replace "`y`" by all numbers of live neighbors that lead to the **s**urvival of a live cell. For example, the Game of Life is described by the rulestring `B3/S23`. We will discuss other rules much later, in Chapter 12, when we develop machinery that lets us run other cellular automata within Life.

There are also many more exotic ways to change the Game of Life beyond just changing the numbers of neighbors that cause cells to be alive. For example, we could have considered a **neighbor** of a cell to only be one of the 4 cells that shares one of its sides as in Figure 1.3, not just a corner (this 4-cell neighborhood is called the **von Neumann neighborhood**, whereas the 8-cell neighborhood used by Life is called the **Moore neighborhood**). We could have considered a hexagonal or triangular grid instead of a square one, or a 1D or 3D grid instead of a 2D one. We could have constructed the game in such a way that not only the number of live neighbors matters, but also their relative positions—such rules are known as **isotropic rules**, or **INT rules** (short for "isotropic non-totalistic").

These are all potentially interesting modifications to make, and they fall under the general umbrella of **cellular automata (CA)**, but we will not consider them any further until Chapter 12. Instead, as we emphasize one final time, our goal is to take the Game of Life cellular automaton itself and push it to its farthest limits.

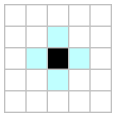


Figure 1.3: A live cell (in black) in the middle and the 4 cells in its von Neumann neighborhood (in aqua).

1.1 Our First Technique: Random Fumbling

There are three main techniques used to construct objects in Life that behave in interesting and unusual ways:

1) We can write a computer program that searches for patterns with particular properties;
2) we can combine different already-known objects in such a way as to create new composite objects; or
3) we can put some random garbage on the Life board and evolve it, with the hope that something interesting pops out.

Option (1) typically requires a fair bit of effort, as well as some knowledge of what exactly it is that we're searching for. Similarly, option (2) is not yet possible for us since we are just starting out with Life and do not yet know of any objects that can be combined in an interesting way. We thus start with the extremely not clever option (3): we try evolving a bunch of random patterns and see what is left of them after their chaos dies down.[2] Our first example is presented in Figure 1.4, which is a random assortment of live cells that takes 116 generations before stabilizing into several distinct objects. Random starting configurations like this one are sometimes called **soup**, and the objects that they leave behind are called **ash**.[3]

In the ash, we see the beehive that we introduced earlier, as well as three other still lifes (i.e., patterns that do not change from one generation to the next). We also see an object called the **blinker**

[2]This is not only *our* approach to getting started with Life, but was also the historical approach: many of the patterns found in the first year or two of Life were found just by evolving many small configurations and looking at the results.

[3]The term "ash" refers to the fact that it is what is left after a pattern stops "burning".

Free PDF. See conwaylife.com/book for pattern files and print version.

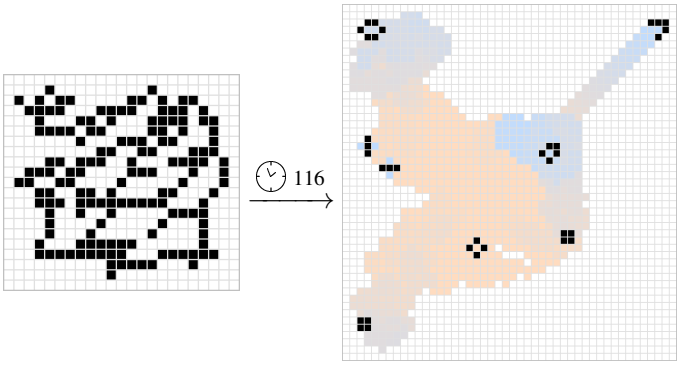


Figure 1.4: Evolving random junk is a decent way to find some small patterns to get us started in Life. The debris on the right introduces us to several still lifes, our first **oscillator**, and the **glider**.

that rotates itself by 90 degrees every generation. Patterns like this one, which cycle between finitely many different configurations, are called **oscillators**, and the configurations that they take on in individual generations are called **phases**. The most exotic object that we see in the ash is the one that is traveling by itself toward the top-right corner of the Life plane. An object that moves through the plane on its own is called a **spaceship**, and this particular one is called the **glider**.

Of course, there is nothing remotely special about the particular random configuration that we started with; we could generate other random starting configurations and see what types of ash pop out of them as well. In fact, Life is so chaotic and unpredictable that we could just change a single cell of the first configuration that we used, and we will likely get a completely different set of ash. Indeed, Figure 1.5 demonstrates the drastic change that can often be made by altering just one cell in a pattern: by just changing one cell from alive to dead, we have made a new pattern that takes almost 3 000 generations to stabilize and results in ash that has over 20 times as many live cells. This new ash also has three additional still lifes in it that were not present in the previous ash. A summary of the ash objects we have seen so far is given in Figure 1.6.

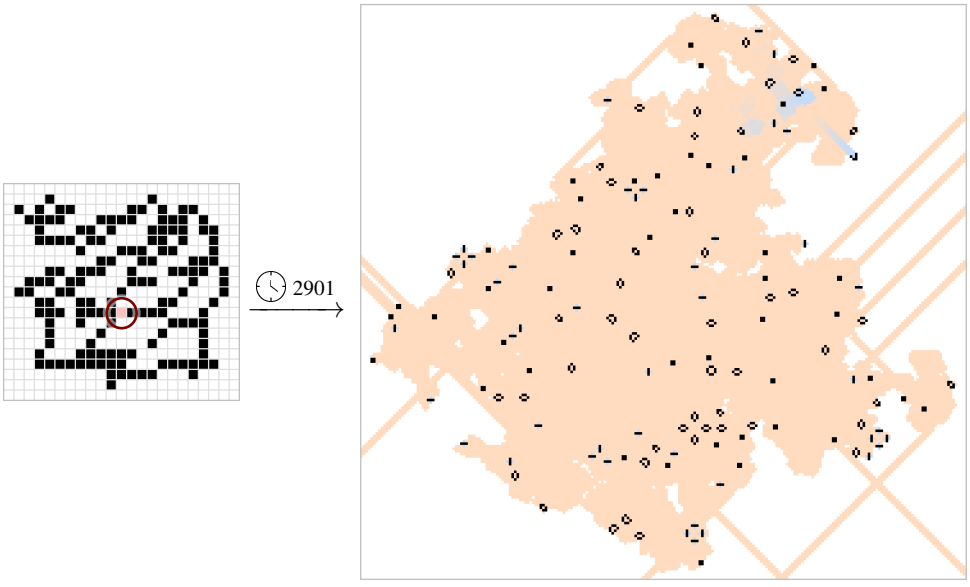


Figure 1.5: Changing just a single cell (circled on the left in red) from alive to dead in our random starting configuration causes its evolution to become completely different. It now takes over 25 times as long to stabilize and its ash contains three new types of still lifes (ponds, ships, and loaves).

Free PDF. See conwaylife.com/book for pattern files and print version.

Still lifes, oscillators, and spaceships are the three most basic types of objects that we will study in Life, and they form the building blocks of all of the more complicated patterns that we will construct. At this point though, there are some natural questions that we can ask about them. For example, we already saw the blinker, which oscillates back and forth between 2 phases every 2 generations. Do there exist oscillators that take longer to return to their original configuration? In other words, do there exist oscillators with **period** larger than 2?

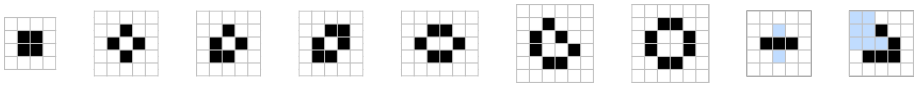


Figure 1.6: From left to right: seven still lifes, called the **block**, **tub**, **boat**, **ship**, **beehive**, **loaf** and **pond**, an oscillator called the **blinker**, and a spaceship called the **glider**. All of these objects frequently appear in the ash left behind by chaotic patterns.

To answer this question, we simply continue not being clever: we evolve more and more random configurations of junk and record new objects that appear in the ash as we find them. Most of the new objects that we will find by doing this are additional still lifes, but new oscillators crop up from time to time as well. For example, the oscillator depicted in Figure 1.7, which is called the **pulsar** and has period 3, appears rather frequently in random ash for its size.

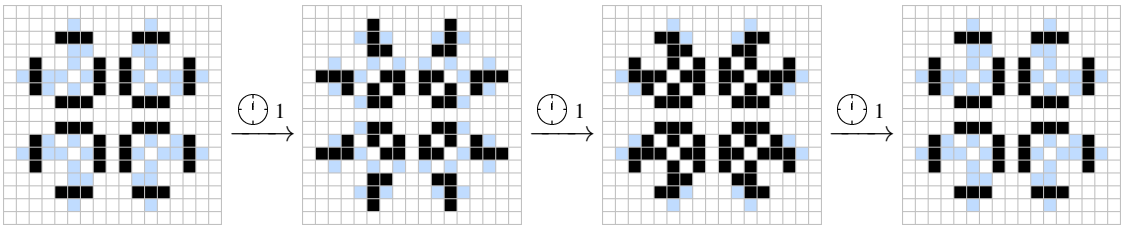


Figure 1.7: A period 3 oscillator called the **pulsar**.

Some other oscillators that appear reasonably often in random ash are depicted in Figure 1.8. Of particular note is the **pentadecathlon**, which has a surprisingly large period of 15. This oscillator behaves somewhat differently than the other ones we have seen, as it pulsates and changes in size as it moves through its period—a property that will be very useful for us later on.

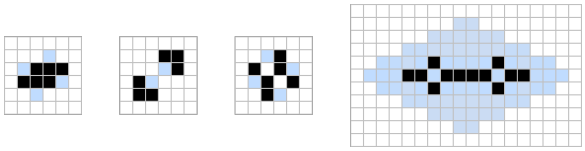


Figure 1.8: From left to right: three oscillators with period 2, called **toad**, **beacon** and **clock**, and a period 15 oscillator called **pentadecathlon**.

In the process of finding these oscillators, it is extremely likely that we will have also come across some other commonly occurring objects, such as the **lightweight spaceship** (or **LWSS** for short) depicted in Figure 1.9. This is another spaceship, but this one moves orthogonally (i.e., directly north, south, east, or west), unlike the glider, which moves diagonally.

Slightly more rare than the lightweight spaceship are the **middleweight spaceship** (or **MWSS**) and the **heavyweight spaceship** (or **HWSS**),[4] displayed in Figure 1.10. There are a handful of other

[4] In keeping with the "____weight spaceship" naming convention, the glider was sometimes called the **featherweight spaceship** in the early days of Life, though this name is very rarely used now.

Free PDF. See conwaylife.com/book for pattern files and print version.

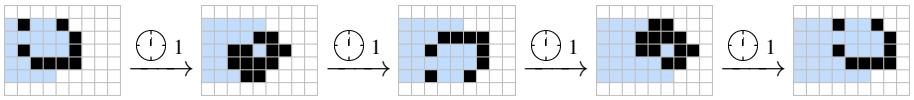


Figure 1.9: The **lightweight spaceship**, which has period 4 and moves orthogonally to the right by 2 cells every 4 generations.

objects that a lucky Life enthusiast might see while evolving random junk, but for the most part these are the "standard" naturally occurring objects. Some examples of random soups that generate more exotic objects are presented in Exercises 1.1 and 1.7.

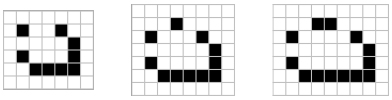


Figure 1.10: From left to right: a lightweight, middleweight, and heavyweight spaceship. They all have period 4 and travel to the right 2 cells every 4 generations.

1.2 Common Evolutionary Sequences

We have seen plenty of interesting patterns so far simply by looking at the results of running random soups until they stabilize. However, we can also learn a lot by carefully watching the evolution of soups as they happen, as not only are there commonly occurring stable ash objects, but there are also commonly occurring unstable objects that explode and drive the evolution of those patterns. For example, one such object is the **T-tetromino**,[5] which is a 4-cell object that appears (for example) in generations 206, 395, and 568 of the evolution of the soup from Figure 1.5. This tiny object (displayed in Figure 1.11) explodes into an arrangement of 4 blinkers that is called a **traffic light**. In fact, three traffic lights (and part of a fourth) can be seen in the ash of Figure 1.5, and the common T-tetromino explosion is exactly why blinkers appear in this formation so frequently.

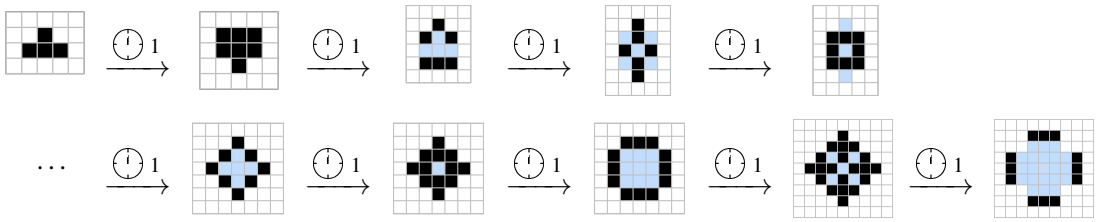


Figure 1.11: A **T-tetromino** (top-left) takes 9 generations to evolve into an arrangement of four blinkers called a **traffic light** (bottom-right).

The T-tetromino also illustrates how symmetry spontaneously forms within Life patterns. While it starts off with only left-right symmetry, after three generations it develops top-bottom symmetry, and then after one more generation it also develops diagonal symmetry. Since Life's rules do not care about the orientation of patterns, symmetry can never be broken once it has formed, which explains why all of the later generations of the T-tetromino also have 8-way symmetry, and why so many of the interesting and/or stable patterns that have been found in Life are symmetric.

[5] A **polyomino** is a pattern made up of orthogonally connected live cells, and a **tetromino** is a polyomino with 4 live cells. More generally, polyominoes with $2, 3, 4, \ldots, 8$ live cells are called **dominoes**, **triominoes**, **tetrominoes**, **pentominoes**, **hexominoes**, **heptominoes**, and **octominoes**.

Free PDF. See conwaylife.com/book for pattern files and print version.

Another unstable object that behaves similarly is the one displayed in Figure 1.12, which explodes over the course of 17 generations in order to create a commonly occurring arrangement of 4 beehives called a **honey farm**. Appropriately enough, the 7-cell object that starts this evolution is called the **pre-honey farm** (as is any other small pattern that follows the same evolution). While only one honey farm appears in the ash in Figure 1.5, many other pre-honey farms appeared earlier (for example, at generations 749 and 1 159) and were subsequently destroyed.

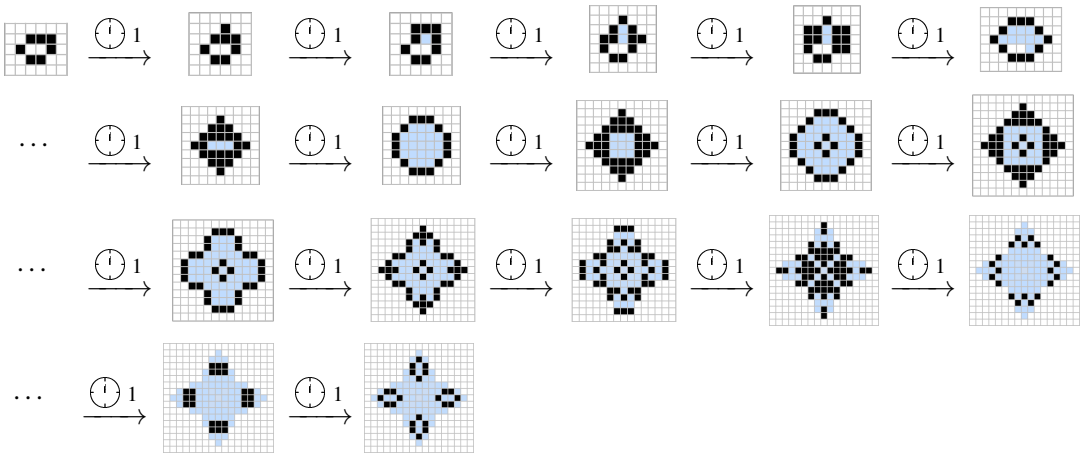


Figure 1.12: A **pre-honey farm** (top-left) takes 17 generations to evolve into an arrangement of four beehives called a **honey farm** (bottom-right).

A slightly messier explosion that is sometimes useful is the one that begins with the **stairstep hexomino** displayed in Figure 1.13. This object takes 63 generations to stabilize into an arrangement of four blocks called the **blockade**. However, since the blockade is somewhat larger than the traffic light and the honey farm, and it also takes much longer to form, it is less commonly seen in ash. For example, the stairstep hexomino can be seen at generation 1 005 of the evolution of the soup in Figure 1.5, but it is interfered with before the blockade can form.

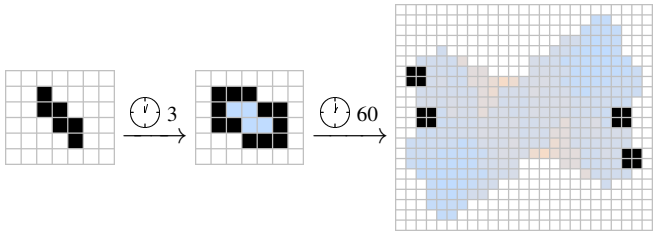


Figure 1.13: The **stairstep hexomino** (left) evolves into an arrangement of four blocks called the **blockade** in 63 generations. This evolution, and any of the intermediate patterns, are often called **lumps of muck**.

It is also often the case that this evolution begins with another small pattern, such as the third generation of the stairstep hexomino, rather than the stairstep hexomino itself. For this reason, this evolutionary sequence, and any of the patterns that appear during its 63-generation explosion, are given a common name: **lumps of muck**.

As one final example, consider the extremely common and extremely messy **pi-heptomino** displayed in Figure 1.14, which appears at generation 20 in the evolution of the soup in Figure 1.4 and numerous times in the evolution of the soup in Figure 1.5 (e.g., in generations 46, 164, 206, 208, 233, ...). While it does not evolve into any particularly interesting stable objects (a rather unremarkable

Free PDF. See conwaylife.com/book for pattern files and print version.

collection of 6 blocks, 5 blinkers, and 2 ponds), it has the interesting property that it moves forward by 9 cells after 30 generations, while leaving behind some messy debris. While this is not quite useful by itself, it can be made useful by combining the pi-heptomino with other objects that destroy the debris before it gets in the way. By doing so, we can use the pi-heptomino to transmit a signal from one place in the Life plane to another or act like a spaceship. We will explore these ideas in Chapter 7 and Section 10.2.

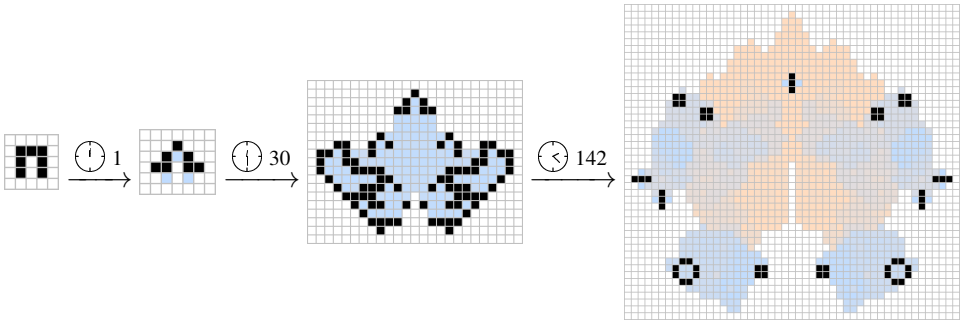


Figure 1.14: The **pi-heptomino** is a commonly occurring unstable object that moves forward by 9 cells after 30 generations (compare the middle two patterns), but leaves behind debris that prevents it from moving ahead another 9 cells.

1.3 The Queen Bee

Sometimes, it is possible to "fix" unstable evolutionary patterns by tweaking them in some simple way, thus creating interesting objects that do not often appear naturally by themselves. Perhaps the most useful example of such a pattern is the **queen bee**, which is a commonly occurring object[6] that reflects itself after 15 generations, but leaves behind a beehive in the process. It follows that after 30 generations, the queen bee is back where it started, but with an additional beehive on either side of it (see Figure 1.15). Shortly after 30 generations, the queen bee collides with the first beehive that it created, resulting in its self-destruction.

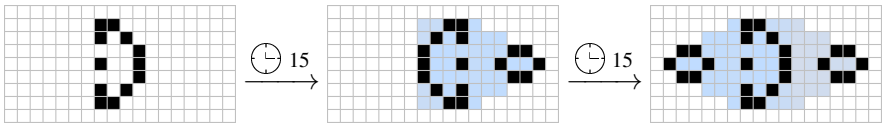


Figure 1.15: A **queen bee** (left) reflects itself and leaves a beehive behind every 15 generations. After doing this twice, the beehives start interfering with the queen bee, causing it to explode.

In order to prevent this self-destruction and turn the queen bee into something useful (an oscillator), we need a way to delete the beehives as they are created. Possibly the simplest way to do this is to use the reaction displayed in Figure 1.16, in which a block being placed next to a beehive results in the beehive being destroyed and the block surviving unharmed (we thus say that the block **eats** the beehive[7]).

We can now simply paste these two reactions together to create an oscillator in which a queen bee (initially facing right) creates a beehive and is reflected to the left, then a block destroys the beehive while the queen bee creates another beehive and is reflected back to the right, then a block destroys

[6] For example, it appears in generation 1 187 of the soup in Figure 1.5.
[7] We will look at eaters in more depth in Section 2.3.

Free PDF. See conwaylife.com/book for pattern files and print version.

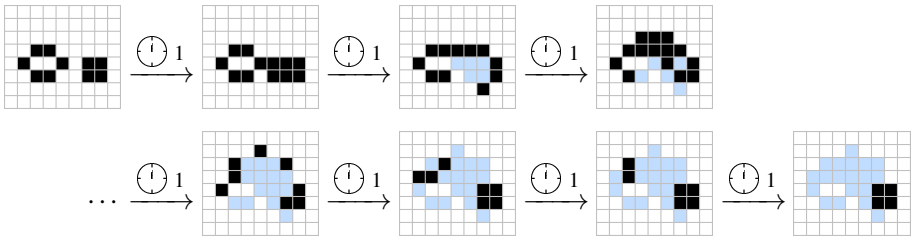


Figure 1.16: Placing a block next to a beehive destroys the beehive in 7 generations, without permanently damaging the block.

the second beehive and the whole process repeats. This period 30 oscillator is called the **queen bee shuttle**[8] and is displayed in Figure 1.17.

The queen bee shuttle is our first example of an engineered object—a pattern that we didn't discover "naturally", but rather one that we specifically constructed by piecing together simpler reactions that we had observed. This is how most recent discoveries in the Game of Life have been made, and it is also the route that we will take through most of the later chapters in this book: we will combine simple patterns and reactions that we have already seen into more complicated patterns that serve a new purpose, and then we will combine those objects into even more sophisticated patterns, and so on.

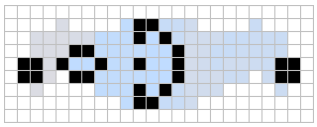


Figure 1.17: The **queen bee shuttle** is a period 30 oscillator constructed by combining the reactions in Figures 1.15 and 1.16.

The queen bee can also be stabilized by some objects other than blocks (see Exercise 1.10). Perhaps the most interesting and useful way to stabilize a queen bee is to use another (very carefully positioned) queen bee. If lined up and timed just right, the queen bees bounce off of each other, and instead of each producing a beehive, their collision produces a glider. This is remarkable, since we can then place stabilizing blocks on the other side of each queen bee so as to create a pattern that oscillates at period 30, but also creates an endless stream of gliders (see Figure 1.18). Patterns that create glider streams are called **glider guns**, and this particular one is called the **Gosper glider gun**.[9] This is the first pattern that we have seen that grows indefinitely (and indeed, it was the first such pattern ever to be discovered), and we will make extensive use of it when constructing more complex objects in later chapters.[10]

1.4 The B-Heptomino and Twin Bees

Another commonly occurring[11] unstable pattern that is very similar in flavor to the queen bee and the pi-heptomino is the **B-heptomino**. This object, like the pi-heptomino, has the interesting property that it behaves somewhat like a spaceship—after 10 generations it evolves into a copy of itself 5 cells

[8] The term "shuttle" typically refers to oscillators in which an unstable object moves back and forth between stabilizing objects (e.g., in this case, the unstable queen bee moves back and forth between the stabilizing blocks on either side of it).

[9] Named after Bill Gosper, who found it in November 1970.

[10] Whether or not there are finite patterns that grow arbitrarily large was one of the major open questions in the early days of Life, and this glider gun's discovery earned Gosper a $50 reward from Conway himself.

[11] For example, it occurs at generation 65 of the soup from Figure 1.4 and generation 274 of the soup from Figure 1.5.

Free PDF. See conwaylife.com/book for pattern files and print version.

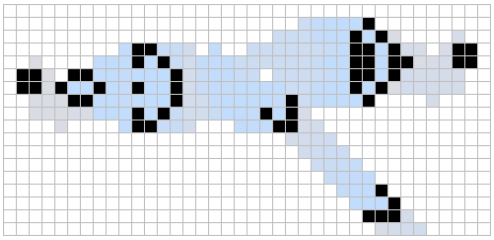


Figure 1.18: The **Gosper glider gun** creates a never-ending stream of gliders by bouncing two queen bees back and forth between two blocks (the object at the top-right is indeed a queen bee, just in a different phase than in the other figures).

forward, plus some extra junk behind it (see Figure 1.19). However, it is then quickly killed by its trailing debris before it has a chance to repeat this process.

Stabilizing the B-heptomino in order to turn it into a useful object is slightly more involved than it was for the queen bee, but fortunately nature does some of the hard work for us: B-heptominoes occasionally occur in pairs, in a configuration that is called the **twin bees**,[12] which are displayed in Figure 1.20. When in pairs like this, the B-heptominoes survive longer than when on their own, and in fact they now reflect themselves, just like the queen bee did (albeit after 23 generations instead of 15), while leaving some junk behind in the process. Unlike the queen bee, the twin bees do not survive long enough to reflect themselves a second time, since the mess they leave behind explodes and destroys them very shortly after the first reflection.

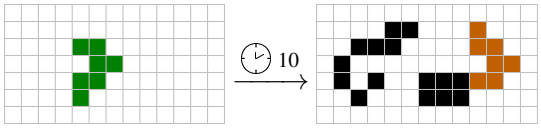


Figure 1.19: A **B-heptomino** (depicted in green on the left) takes 10 generations to move forward by 5 cells (in orange on the right) and create a bunch of junk behind it. The trailing junk subsequently destroys the B-heptomino, preventing it from traveling any farther.

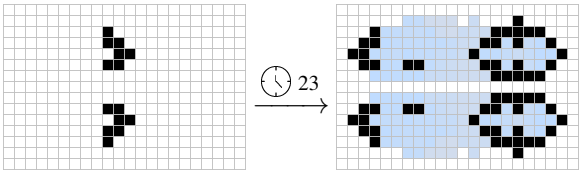


Figure 1.20: Twin bees reflect themselves and leave behind a chaotic mess every 23 generations (the two-cell objects in the middle of the image on the right die off in the next generation and thus have no effect).

In order to stabilize the twin bees, we take a cue from the queen bee and try placing blocks in such a way as to eat the mess that is left behind. This works very well, and the stabilization displayed in Figure 1.21 was already known in the very early days of Life.[13]

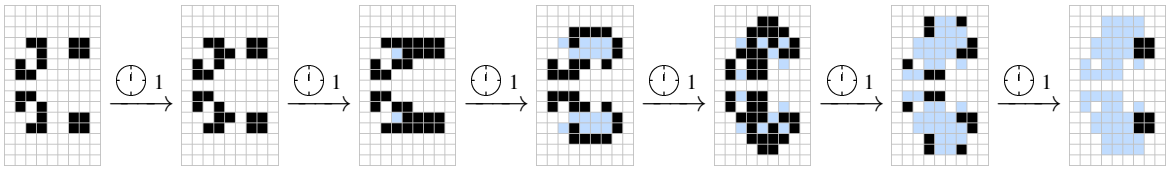


Figure 1.21: Placing two blocks next to the debris left behind by twin bees destroys that debris in 5 generations, while leaving the blocks intact.

[12]The name "twin bees" refers both to the fact that many of its properties and uses are completely analogous to the queen bee, and also to the fact that it is made up of two B-heptominoes, so its name can be read as "twin Bs".

[13]This stabilization was found by Bill Gosper in 1971.

Free PDF. See conwaylife.com/book for pattern files and print version.

We now do exactly what we did with the queen bee: we place blocks on *both* sides of the twin bees so that they reflect from the right to the left, their debris is destroyed, then they reflect from the left to the right, their debris is destroyed again, and then they repeat. This whole process takes $23 + 23 = 46$ generations to complete, and results in the period 46 oscillator known as the **twin bees shuttle** displayed in Figure 1.22.

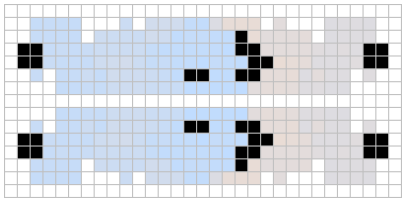


Figure 1.22: The **twin bees shuttle** is a period 46 oscillator constructed by combining the reactions in Figures 1.20 and 1.21.

As with the queen bee, there are many other ways to stabilize the twin bees shuttle besides using two blocks on each side as in Figure 1.21, and some of these alternate stabilizations are investigated in Exercise 1.4. In fact, we can even collide two pairs of twin bees with each other to create a period 46 glider gun in almost the exact same way that we constructed the Gosper glider gun by colliding two queen bees. This new glider gun,[14] which we call the **twin bees gun**,[15] is shown in Figure 1.23.

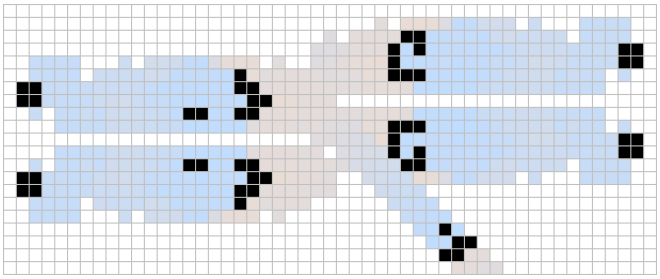


Figure 1.23: The **twin bees gun** is a period 46 glider gun that was constructed by bouncing two twin bees back and forth between each other and two stabilizing pairs of blocks. Note that the pair of objects in the middle-right are indeed twin bees, just in a different phase than the other figures.

1.5 The Switch Engine

One final commonly occurring unstable pattern that is worth introducing at this point is the **switch engine**, which is the configuration of 8 live cells displayed in Figure 1.24. Just like the queen bee and twin bees, the switch engine leaves behind some debris and reappears later on in its evolution (this time after 48 generations), but in a different location and orientation. However, unlike the queen bee and twin bees, the switch engine does not return to its original position after repeating this reaction. Instead, the switch engine continually moves farther and farther away from where it started, just like a spaceship.

As with several other patterns that we have now explored, the switch engine will be destroyed by its debris if we do not stabilize it somehow. Because of the continued movement of the switch engine,

[14] This gun was also found by Bill Gosper, and was the smallest p46 gun until ConwayLife.com forums user "iNoMed" found a smaller version in December 2021: conwaylife.com/forums/viewtopic.php?p=139147#p139147.

[15] A slight variant of this gun was originally found in 1971 and called the **new gun**, since it was the first gun to be found after the Gosper glider gun. However, this name is woefully out of date at this point.

Free PDF. See conwaylife.com/book for pattern files and print version.

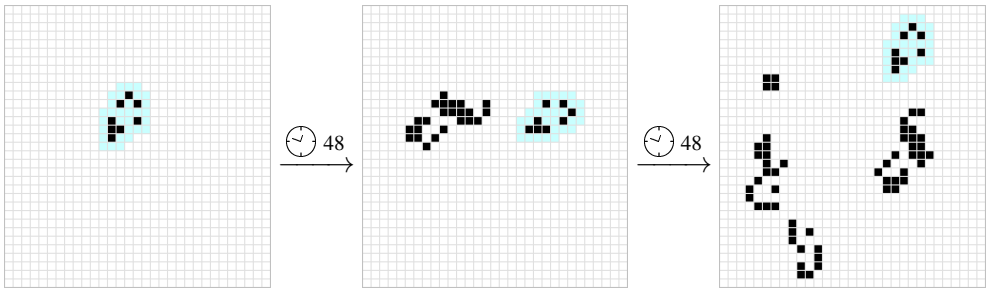


Figure 1.24: A **switch engine** (outlined in aqua) moves and reflects itself across the diagonal every 48 generations, but leaves behind some chaotic junk in the process. After 96 generations, the switch engine is back in its original orientation, but 8 cells northeast of where it started. The switch engine appears for the last time in generation 1152, after repeating this entire process 12 times. The chaotic junk then finally collides with the switch engine and destroys it.

we don't expect to be able to convert it into a stationary object like an oscillator or glider gun, like we did with the queen bee or twin bees from the previous sections, but rather we hope to turn it into some sort of "useful" moving object like a spaceship.

It turns out that turning it into a spaceship is actually somewhat tricky, so we defer that discussion to Section 4.3. However, there are simple ways to stabilize it into a moving object that is *not* a spaceship. For example, if we place a block next to a switch engine as in Figure 1.25, then it evolves into an object called the **block-laying switch engine**—a switch engine whose debris settles down into an ever-lengthening stream of blocks (8 blocks every 288 generations).

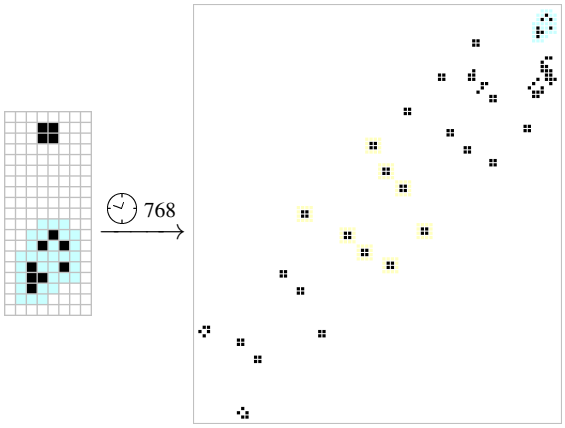


Figure 1.25: A switch engine (outlined in aqua) with a single block strategically placed next to it evolves into a **block-laying switch engine**—a pattern that repeatedly lays a configuration of 8 blocks (outlined in yellow) every 288 generations as it moves northeast.

An object like this one, which moves but leaves periodic junk behind it, is called a **puffer**. Another puffer can be constructed from a switch engine by instead placing a block next to it as in Figure 1.26. In this case, the pattern evolves into something called the **glider-producing switch engine**, which is a switch engine that leaves behind four blinkers, three blocks, two beehives, a boat, a ship, a loaf, and a glider every 384 generations. The glider travels in the same direction as the switch engine, but because the glider travels faster (at a pace of 3 cells every 12 generations, versus the switch engine's 1 cell every 12 generations), it does not take long for it to pass the switch engine.

The block-laying and glider-producing switch engines are remarkable for the fact that, not only do they grow infinitely, but they are also "natural" (as opposed to engineered, like the Gosper glider gun).

Free PDF. See conwaylife.com/book for pattern files and print version.

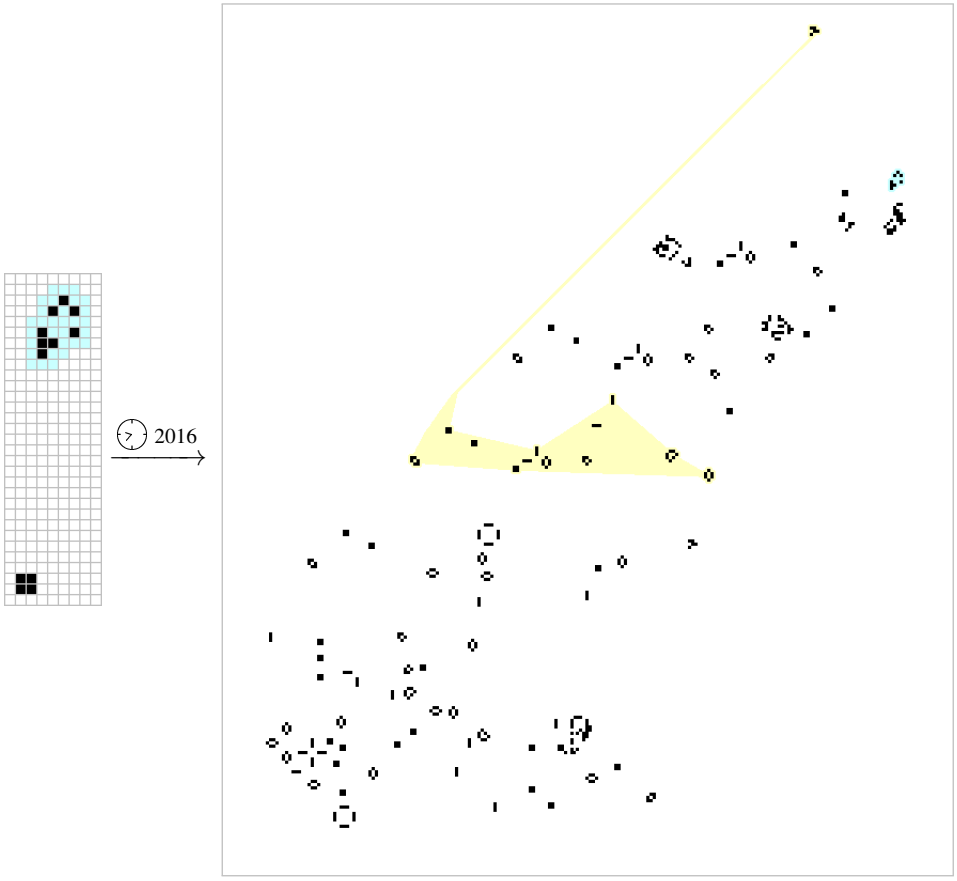


Figure 1.26: A switch engine (outlined in aqua) with a single block strategically placed next to it can also evolve into a **glider-producing switch engine**, which repeatedly lays a messy pattern of 4 blinkers, 8 still lifes, and one glider (outlined in yellow) every 384 generations, as it moves northeast.

In fact, puffers based on switch engines are the only infinitely growing patterns that have ever formed as a result of randomly filling some portion of the Life plane and then evolving it (see Exercise 1.1). The ease with which these switch engine puffers appear (compared to other infinitely growing patterns like the two glider guns we saw earlier) can be explained by noting that they have some very small predecessors, like the 12-cell objects shown in Figures 1.25 and 1.26,[16] whereas glider guns require a comparatively large number of "coordinated" live cells to form.

1.5.1 Arks

Perhaps an even more natural way to stabilize a switch engine is to use additional switch engines; after all, using a stationary object (like a block) to stabilize a moving object seems somewhat nonsensical, and only worked for us because the switch engine was able to turn the block into a moving stabilization (this is why the bottom-left corners of Figures 1.25 and 1.26 are irregular: the block itself wasn't the stabilization, but rather it just reacted chaotically with the switch engine's debris in such a way that it eventually created a moving stabilization).

On the other hand, using a moving object to stabilize another moving object (or using two moving objects to stabilize each other) is a common technique that we will use repeatedly. In this particular case, any puffer created by using two switch engines is called an **ark**, and dozens can be found just by

[16] These 12-cell predecessors can be turned into 11-cell predecessors by simply removing a single cell from the block.

Free PDF. See conwaylife.com/book for pattern files and print version.

trying different positions and phases of nearby switch engines. For example, if we place two switch engines next to each other as in Figure 1.27, the result is an ark that leaves behind a multitude of different objects, but does so in a periodic way: the same collection of still lifes, oscillators, and gliders is created every 576 generations, offset by 48 cells diagonally.

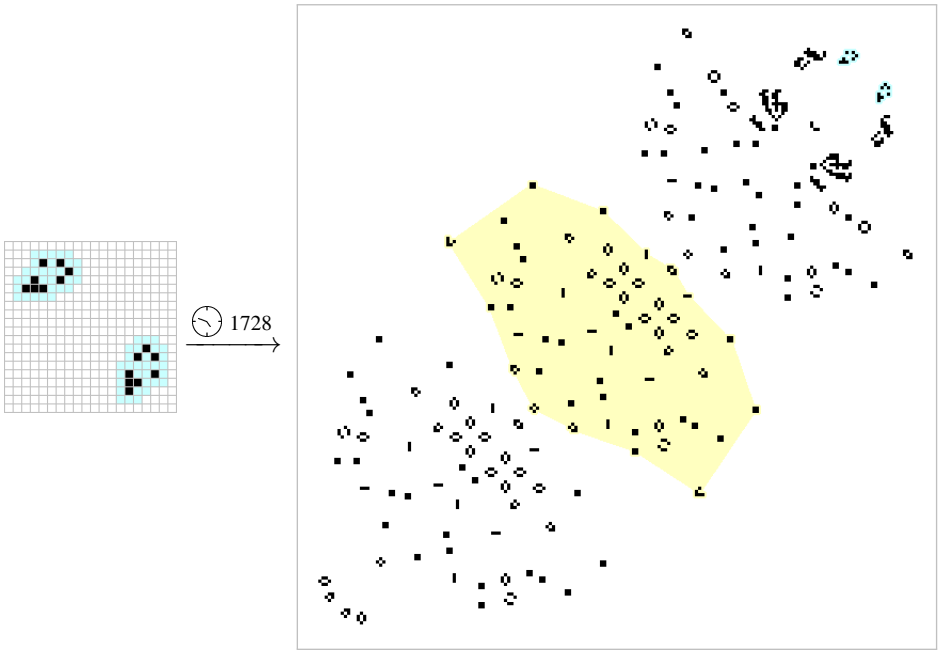


Figure 1.27: Two switch engines (outlined in aqua) can be used to stabilize each other and create an **ark**. In this particular ark, the switch engines leave behind two gliders, two toads, eight blinkers, and 42 still lifes (outlined in yellow) every 576 generations.

1.6 Methuselahs and Stability

Some of the patterns that we have investigated are extremely chaotic and take a long time to stabilize,[17] such as the B-heptomino and the random configuration depicted in Figure 1.5, which take 148 and 2901 generations to stabilize, respectively. More extreme than either of these examples is the 8-cell switch engine, which takes 3911 generations to stabilize—considerably longer than most other patterns with 8 or fewer cells. It seems somewhat natural to ask how far these examples can be pushed—that is, how long can a pattern of a given size take to stabilize? A pattern that takes exceptionally long to stabilize, relative to other similarly sized patterns, is called a **methuselah**.[18]

Before looking at more specific examples of methuselahs, we make it very clear up front that this definition is very imprecise: there is no completely objective way to say that some patterns are methuselahs while other patterns aren't. Instead, the classification usually takes place simply by comparing the lifespan of the pattern with the longest known lifespan of similarly sized patterns. Here, "size" may refer to either the number of live cells in the pattern, or to the area of its **bounding box**, which is the smallest rectangle that contains it.

No good search methods are known for finding methuselahs, so all of them have been found essentially by random guessing or exhaustive search. As an example, consider the **R-pentomino**,

[17] We will see shortly that properly defining what it means for a pattern to "stabilize" is very troublesome, but for now it just means that the pattern has broken down into non-interacting still lifes, oscillators, and spaceships.

[18] Named after the Biblical man Methuselah who reportedly lived the longest of any human—969 years.

Free PDF. See conwaylife.com/book for pattern files and print version.

which is depicted in Figure 1.28. This pattern is interesting for the fact that it takes considerably longer to stabilize than any other pattern that fits with in a 3×3 bounding box, and also much longer to stabilize than any other polyomino with 5 or fewer live cells (see Exercise 1.3).[19] The evolutionary sequence that evolves from it is also extremely important—it produces a B-heptomino (well, actually a **B-heptaplet** that evolves in the same way) in generation 28, an important object called a **Herschel** in generation 48, and the very first glider that was ever observed in Life[20] in generation 69.[21]

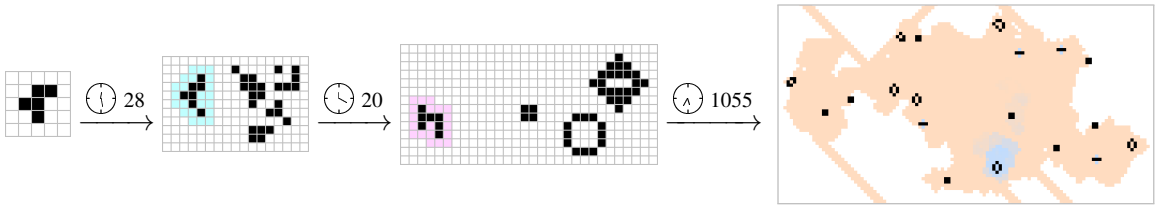


Figure 1.28: An R-pentomino takes 1 103 generations to stabilize—much longer than any other polyomino with 5 or fewer cells. Along the way, it produces numerous other important patterns such as the **B-heptaplet** highlighted in aqua and the **Herschel** highlighted in magenta.

We could similarly ask what the longest-lived pattern with n cells for $n = 6, 7, 8, \ldots$ are. The longest-lived such patterns that are known are summarized in Table 1.1, as long as we restrict our attention only to patterns with reasonably small bounding boxes (which of course is subjective). However, these patterns are only known to be optimal for $n \leq 9$; the longest-lasting known 10-cell methuselah in a small bounding box was not found until 2019, so above $n = 9$ there may still be new discoveries to be made.[22]

To highlight the difficulty of defining methuselahs in a rigorous way, note that there are 8- and 9-cell patterns that can be made to have lifespans as long as we like, since we can just aim a glider at a far away blinker or a block. However, these somewhat trivial examples can be avoided by requiring that a methuselah have a suitably small bounding box (which also rules out 8-cell methuselahs like the one displayed in Exercise 1.13). If we go this route and focus on the bounding box of a pattern, rather than its number of cells, then we find patterns like the one displayed in Figure 1.29,[23] which fits within a 16×16 bounding box and has a lifespan of 52 514 generations, which is longer than any other known pattern of this size.

Another potential problem that arises when discussing methuselahs and stability comes from the fact that we already saw patterns with just 12 cells that grow indefinitely: the block-laying and glider-producing switch engines[24] of Section 1.5. This is not too difficult to take care of: even though these patterns grow forever, they do so in a regular and predictable way. For example, to know what a block-laying switch engine looks like in generation 5 000 000, we don't actually need to evolve it 5 000 000 times: we could simply move the switch engine the correct number of spaces to the northeast and paste a trail of blocks behind it. We could thus say, for example, that the block-laying switch engine stabilizes once it enters the periodic portion of its evolution where it repeatedly lays 8 blocks every 288 generations.

These problems might seem simple enough to overcome by tweaking our definitions carefully, but they are just the tip of the iceberg when it comes to what can go wrong when trying to rigorously

[19] However, there are 5-cell patterns that are *not* polyominoes that last slightly longer, as demonstrated in Table 1.1.

[20] First noticed by Richard K. Guy in 1970.

[21] In fact, all of the unstable objects from Section 1.2 appear during the R-pentomino's evolution—see Exercise 1.5.

[22] Exhaustive computer searches for methuselahs with $n \leq 9$ cells were carried out by Paul Callahan in 1997, Tomas Rokicki in 2005, and Nick Gotts in 2019.

[23] Found by Daniel Mouscher in October 2021, based on a 52 513-generation methuselah found by Dylan Chen.

[24] This can easily be reduced to 11 cells by removing one of the cells in the block in either pattern.

Free PDF. See conwaylife.com/book for pattern files and print version.

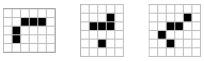
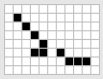
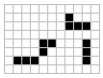
Cells	Methuselah	Name	Lifespan
5		R-pentomino grandparents	1 105
6		R-pentomino great-great-great grandparents	1 108
7		acorn	5 206
8		–	7 468
9		bunnies 9	17 410
10		bunnies 10b	17 431
11		–	23 334
12		–	23 801
13		Lidka	29 126

Table 1.1: The longest-lived known methuselahs with 5–13 cells. The methuselahs with 7 or fewer cells were all known by no later than 1971: the R-pentomino relatives were fairly well-known, so their exact discoverer is difficult to pin down, but the acorn was found by Charles Corderman. The methuselah with 8 cells was found by Tomas Rokicki in 2005, bunnies 9 was found by Paul Callahan in 1997, bunnies 10b was found by Nick Gotts in November 2019, and the 11-, 12-, and 13-cell methuselahs were found by Simon Ekström in May 2016, February 2017, and March 2017, respectively.

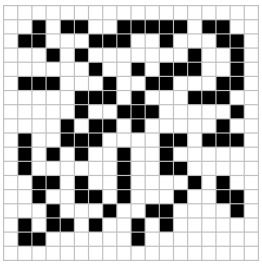


Figure 1.29: A methuselah that fits within a 16×16 bounding box and takes 52 514 generations to stabilize.

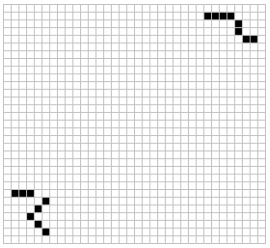


Figure 1.30: A 16-cell pattern that takes a whopping 736 692 generations to stabilize. The 8-cell objects at the top-right and bottom-left individually are just predecessors of the switch engine, so this pattern is an ark.

define methuselahs and what it means for a pattern to stabilize. As yet another example, consider the remarkable 16-cell pattern[25] depicted in Figure 1.30, which takes 736 692 generations to stabilize.

The "problem" with this pattern is that its non-stabilization is rather artificial; it evolves into adjacent block-laying and glider-producing switch engines, which interact in such a way as to fire a

[25] Found by Nick Gotts in 2005.

Free PDF. See conwaylife.com/book for pattern files and print version.

glider backward toward some debris every 2 304 generations. Every time this backward glider hits the debris, it transforms it into something slightly different and unpredictable. This process repeats more than 250 times before it breaks down and a glider destroys the debris enough that future gliders pass right through without touching it at all. However, by this point the pattern has been growing for over 700 000 generations into the huge mess shown in Figure 1.31.

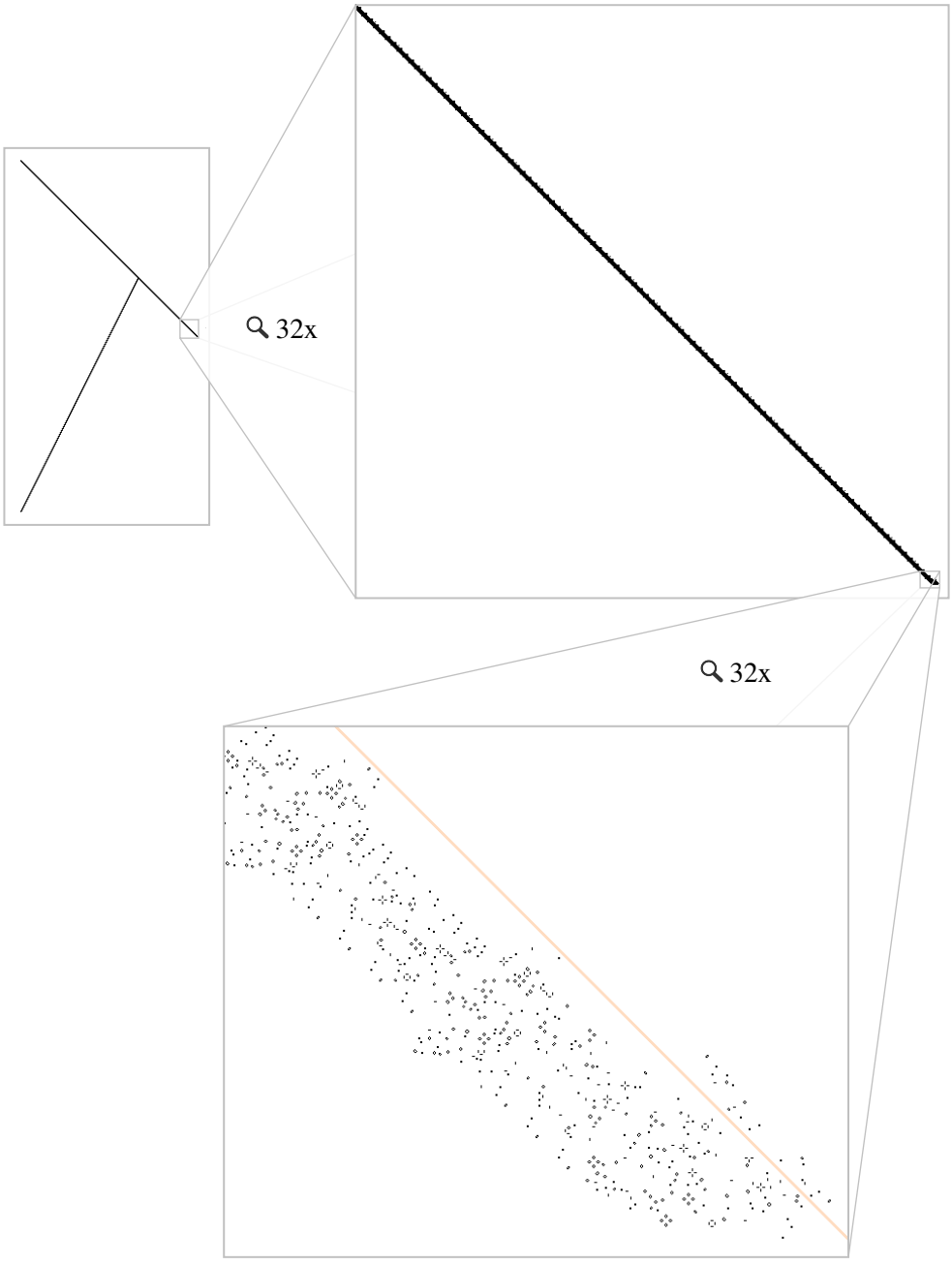


Figure 1.31: The pattern from Figure 1.30 leaves behind this absolutely massive ash after 736 692 generations. We have zoomed in by a factor of $32 \times 32 = 1\,024$ on the corner of the ash that the pattern started growing from, which is also the corner that contains the debris that gliders are fired at throughout its extensive lifespan. The orange line on the bottom-right depicts the southeastward path taken by gliders that now (just barely) miss hitting the debris.

Free PDF. See conwaylife.com/book for pattern files and print version.

While patterns like this one are certainly interesting, they are not quite in the spirit of what we want from methuselahs—even though the previous pattern did not completely stabilize for hundreds of thousands of generations, it was "mostly" stable for the vast majority of that time. Rather, we would like methuselahs to take a long time to stabilize based primarily on chaos that we cannot completely break down into simple reactions like gliders repeatedly hitting some debris. We will not try to make this precise, but rather just leave it as a phenomenon that we know when we see.

To briefly suggest some even more extreme examples of the weirdness that happens with stability, note that we will learn in Chapter 6 how to construct a pattern that computes the prime numbers. Should such a pattern be considered stable? Perhaps even more striking than this, in Section 8.8.4 we will then create a pattern whose long-term behavior we do not understand: it might continue to grow in a reasonably regular fashion indefinitely, or it might stabilize into a finite pattern. What we do know is that it continues to grow for at least its first $10^{2^{500\,000\,000}}$ generations, so if it does eventually stabilize and remain finite, it must do so after that point (and we have no hope of actually simulating the pattern long enough to find out the answer).

Even worse than this, in Chapter 9 we will demonstrate how to construct patterns that compute anything that can be computed by a computer. We can thus encode a huge variety of unsolved mathematical and computational problems into Life patterns, many of which have long-term behavior that we have absolutely no hope of determining.

1.7 Gardens of Eden

Oftentimes, a lot can be learned by considering Life in reverse: instead of investigating what a pattern evolves into, we investigate what evolves into it. That is, we look for parents (and grandparents, and great-grandparents, ...) of the pattern that we are interested in.[26] Some patterns, such as the block, have numerous parents (see Figure 1.32), which is part of the reason why they appear so frequently in the ash of random soups.

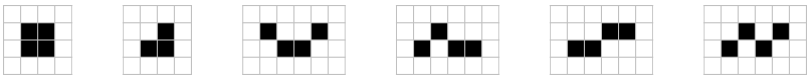


Figure 1.32: Six small parents of the block. From left-to-right, the first three of these objects are the block, **pre-block**, and **grin**, respectively, while the other three do not have names.

Other patterns, such as the clock, have relatively few parents and thus appear less frequently in random soups. Recall that the clock has only six cells and is period 2. Surprisingly, it appears in random soups much less frequently than some much larger objects like the period 3 pulsar or the period 15 pentadecathlon. It seems natural to ask whether or not there exists a pattern that does not have even a single parent. That is, does there exist a pattern that cannot ever appear in the evolution of any other pattern, but rather can only ever appear in generation 0?[27] A pattern with this property is called a **Garden of Eden**,[28] and we begin by showing that they do indeed exist.

The rough idea of why Gardens of Eden must exist is that some patterns (such as blocks) have lots of parents, so there are not enough parents left over for all other patterns. To prove this explicitly, we

[26] Strictly speaking, if a pattern has one parent then it must have infinitely many, since we could just add any pattern that dies in one generation far away. When discussing how many parents a pattern has, we will always ignore parents that have dying ash like this that has no effect on their evolution.

[27] Generation 0 refers to the starting configuration before applying the Life rules to it. Generation 1 is the pattern that is obtained by applying the Life rules once, and generation n is the pattern obtained by applying the Life rules n times.

[28] This term was coined in the context of cellular automata by John W. Tukey in the 1950s, long before Conway's Game of Life was introduced.

Free PDF. See conwaylife.com/book for pattern files and print version.

will barely need to use anything more than the fact that blocks and pre-blocks both evolve into the same thing.[29]

Theorem 1.1 — Existence of Gardens of Eden
There exist Gardens of Eden in Conway's Game of Life.

Proof. Let $n \geq 1$ be an integer and consider all patterns that fit within a $6n \times 6n$ square on the Life grid. The contents of the central $(6n-2) \times (6n-2)$ square in generation 1 only depend on the contents of the original $6n \times 6n$ square in generation 0. This central $(6n-2) \times (6n-2)$ square contains $(6n-2)^2$ cells, each of which can be alive or dead, so there are $2^{(6n-2)^2}$ distinct patterns that could fill this central square. We will now show that, if n is large enough, some of these patterns must have no parent.

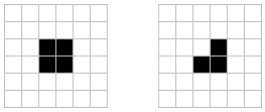


Figure 1.33: Two 6×6 tiles that evolve the same way no matter what is placed outside of the tiles.

To this end, partition the $6n \times 6n$ square into n^2 tiles, each of size 6×6. Each tile contains 36 cells, each of which can be alive or dead, so there are 2^{36} different tiles. However, the two tiles displayed in Figure 1.33 evolve in the same way regardless of what other tiles are placed around them. We thus conclude that in any pattern that uses the first tile, we could replace it by the second tile without changing the evolution of the overall pattern (see Figure 1.34). It follows that if we want to catalog all possible $(6n-2) \times (6n-2)$ patterns that can be present in generation 1, we only need to consider the children of patterns made from a set of $2^{36}-1$ (not 2^{36}) different tiles in generation 0.

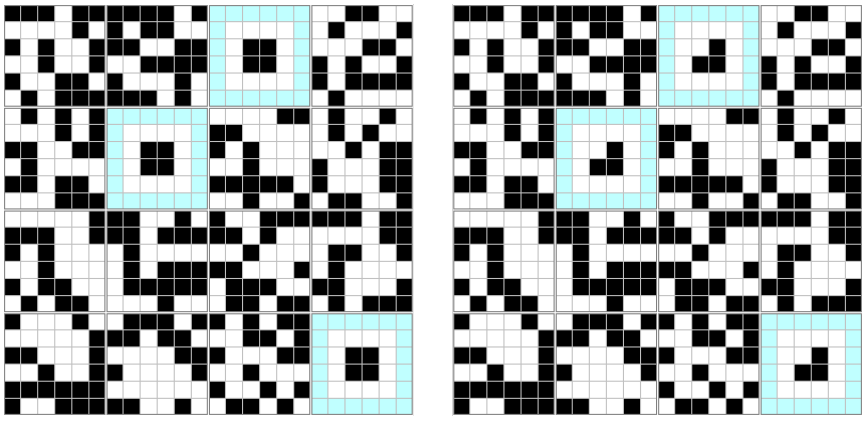


Figure 1.34: A $6n \times 6n$ pattern in the $n = 4$ case, broken down into 6×6 tiles. Each of the tiles containing only a block in the center (outlined in aqua) can be replaced by a tile with a pre-block in the center without affecting the evolution of the overall 24×24 pattern.

Since the $6n \times 6n$ square has n^2 of these tiles, there are at most $(2^{36}-1)^{n^2}$ different possible children (i.e., patterns present in generation 1). Since there are $2^{(6n-2)^2}$ distinct patterns that could fill the central $(6n-2) \times (6n-2)$ square, but at most $(2^{36}-1)^{n^2}$ of them appear in children of patterns in the $6n \times 6n$ square, all that remains is to show that $(2^{36}-1)^{n^2} < 2^{(6n-2)^2}$ when n is sufficiently

[29] We could instead use two other objects that evolve into the same thing in the proof: we just use blocks and pre-blocks because they are so simple.

Free PDF. See conwaylife.com/book for pattern files and print version.

large. Taking the n^2-th root of both sides of the inequality reduces it to $2^{36}-1 < 2^{(6-2/n)^2}$, which is indeed true when n is sufficiently large, since we can make $2/n$ arbitrarily close to 0 and hence $2^{(6-2/n)^2}$ arbitrarily close to $2^{6^2} = 2^{36}$ (in particular, we can make it larger than $2^{36}-1$).[30] ■

It is worth noting that the method used in the proof of Theorem 1.1 is very general and applies to any cellular automaton for which two distinct (finite) patterns evolve into the same pattern.[31] Also, even though it is non-constructive, it can be used to find (extremely loose) bounds on how large Gardens of Eden must be—see Exercise 1.16.

Now that we know that Gardens of Eden exist, our next goal is to actually find an explicit example of one.[32] Unfortunately, our method of proof of Theorem 1.1 is not of much help here, since it only shows that Gardens of Eden exist in extremely large regions (the smallest n for which the inequality $(2^{36}-1)^{n^2} < 2^{(6n-2)^2}$ holds is somewhere around $n \approx 10^{12}$), and it does not actually tell us how to find one in such a region. The pattern obtained by concatenating together all 2^{n^2} different $n \times n$ patterns into a rectangle of size $\left(n2^{n^2}\right) \times n$ is guaranteed to be a Garden of Eden, but it is exponentially larger than the upper bound provided by the theorem.

Our method of construction is to build a Garden of Eden one cell at a time, repeatedly choosing the new cell that we add to the pattern so as to result in it having as few parents as possible. More explicitly, we build this Garden of Eden as follows:[33]

- We consider the two possible states of a single cell. Out of the $2^9 = 512$ possible configurations of the 3×3 square centered at this cell, 372 lead to the central cell being dead and 140 lead to it being alive. Thus we make that central cell alive, so as to have fewer parents.
- We then consider the two possible states of a cell that is adjacent to the starting (alive) cell. Out of the $140 \times 2^3 = 1\,120$ possible configurations of the 4×3 rectangle centered at those two cells that lead to the first cell being alive, 703 lead to the second cell being dead and 417 lead to it being alive. We thus make the second cell alive, so as to have fewer parents.
- We continue in this way in a clockwise spiral around the initial cell, deciding whether each new cell will be alive or dead based on which of the two possibilities results in a pattern with fewer parents.

While this method might not seem to yield anything useful at first—after 40 cells, each of them has been chosen to be alive, and the number of parents has exploded to 4 624 592—the number of parents does eventually start to dwindle, and after 266 cells the pattern has no parents at all (and is thus a Garden of Eden).[34] This Garden of Eden is displayed in Figure 1.35.

The pattern that we have constructed is interesting for the fact that it is not only a Garden of Eden, but it remains a Garden of Eden regardless of which live and dead cells we place outside of the central 266-cell spiral. Patterns like this one that cannot be *any* part of the evolution of another pattern (equivalently, they are still a Garden of Eden no matter what other cells are placed around them) are called **orphans**.[35] Every pattern containing an orphan is (by definition) a Garden of Eden,

[30] In fact, not only is it the case that $(2^{36}-1)^{n^2} < 2^{(6n-2)^2}$ when n is large, but the ratio $(2^{36}-1)^{n^2}/2^{(6n-2)^2}$ can be made as small as we like by making n sufficiently large (see Exercise 1.18). In other words, not only do Gardens of Eden *exist*, but in fact almost all large patterns are Gardens of Eden.

[31] This more general theorem was proved by Edward F. Moore and John Myhill in the early 1960s [Moo62, Myh63], so Gardens of Eden were known to exist in Conway's Game of Life right from the moment it was introduced.

[32] The first explicit Garden of Eden in the Game of Life was constructed by Roger Banks in 1971.

[33] This method is due to Nicolay Beluchenko [Slo11].

[34] In step 262 there is a tie between the number of parents if the cell is chosen to be alive or dead. We (arbitrarily) chose the cell to be alive.

[35] This term was coined by Conway. Also, the proof of Theorem 1.1 actually demonstrates the existence of orphans, not just Gardens of Eden.

Free PDF. See conwaylife.com/book for pattern files and print version.

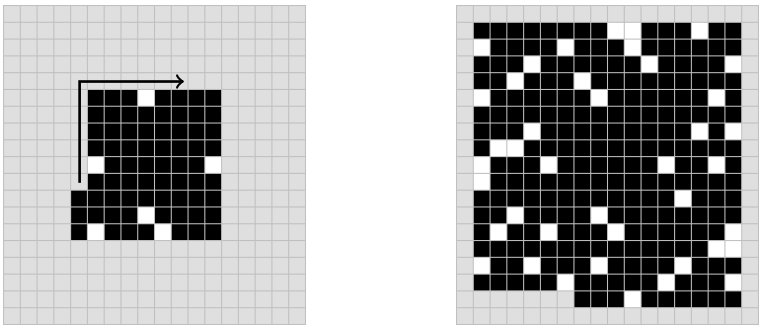
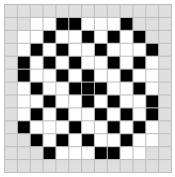


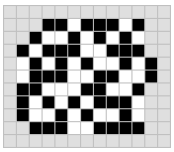
Figure 1.35: A partial Garden of Eden (left) that is being constructed one cell at a time by placing cells in a clockwise spiral, choosing whether they should be alive or dead based on which state results in fewer parents. The completed Garden of Eden (right) has 41 dead cells and 225 alive cells (the light gray cells are unspecified—the pattern remains a Garden of Eden regardless of whether they are alive or dead).

and remarkably the converse is also true—every Garden of Eden contains an orphan.[36]

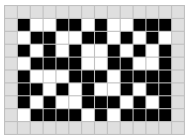
Now that we know that small Gardens of Eden (and orphans) exist, we would like to know how small and simple they can be. A complete answer to this problem is still open (and perhaps is currently one of the most actively researched problems in Life), but many partial results are known. The smallest known orphans and Gardens of Eden (in terms of number of cells and bounding box size) are presented in Figure 1.36.[37]



(a) An orphan with 45 live cells, found by Nicolay Beluchenko in 2009.



(b) An orphan with 88 specified cells, found by Steven Eker in 2017.



(c) An orphan in a bounding box with area $96 = 8 \times 12$, found by Steven Eker in 2016.

Figure 1.36: The current smallest-known orphans by (a) number of live cells, (b) total number of live and dead cells specified, and (c) bounding box area.

On the other hand, there are also results that show that orphans cannot be "too" small. For example, exhaustive computer searches have shown that there does not exist an orphan that fits within a 6×7 bounding box.[38] In a similar vein, we now show that there does not exist a Garden of Eden (and thus there does not exist an orphan) that has a bounding box with height one.[39]

Theorem 1.2 — No Thin Gardens of Eden
In Conway's Game of Life, every pattern that fits within a bounding box of height 1 has a parent. In other words, there do not exist Gardens of Eden with height 1.

Proof. We prove the theorem by explicit construction: we show how to find a parent of any 1-cell-thick pattern. The key step in our construction is to build a border that dies off completely in one

[36]This equivalence follows immediately from standard results in dynamical systems (see [Hed69], for example). It was made explicit by Jarkko Kari no later than 2010—see conwaylife.com/forums/viewtopic.php?p=51782#p51782.

[37]These patterns can be verified to be orphans via a computer program called *JavaLifeSearch*—see conwaylife.com/wiki/JavaLifeSearch and Exercise 1.19.

[38]This computation was carried out independently by Steven Eker and Marijn Heule in 2016.

[39]This theorem was originally proved by Jean Hardouin–Duparc in 1974 [HD74].

Free PDF. See conwaylife.com/book for pattern files and print version.

generation and guarantees that any junk placed inside of it does not expand outward in that generation. Figure 1.37 shows one such border: no matter what we place inside the $3 \times n$ box in the middle of the border, no cell outside of that $3 \times n$ box will be alive in the next generation.

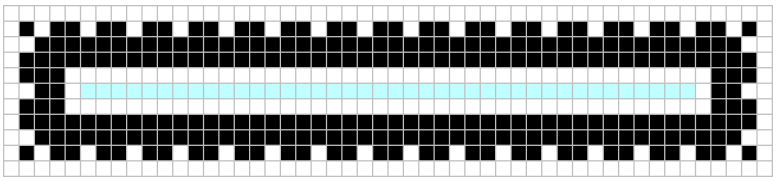


Figure 1.37: A border that can be used to help construct parents of 1-cell-thick patterns. No matter what we place in the central $3 \times n$ region, no cell outside of the $3 \times n$ region will be alive in the next generation. We will fill it in such a way so that the desired $1 \times (n-2)$ pattern evolves in the center row, highlighted in aqua).

To complete the proof, we just need to find a way to fill that $3 \times n$ box in such a way that it evolves into any $1 \times (n-2)$ pattern that we desire. One simple method that *almost* works is to put the desired pattern itself in the central row of the $3 \times n$ box, and its negation (i.e., flip all alive cells to dead and vice-versa) in the top and bottom rows of the $3 \times n$ box. It is straightforward to check that the cells in the top and bottom rows of the $3 \times n$ box will always be overpopulated, and thus those two rows will always be dead in the next generation. Furthermore, the central row evolves in the correct way in 7 out of every 8 possible configurations of three adjacent alive/dead cells (see Figure 1.38).

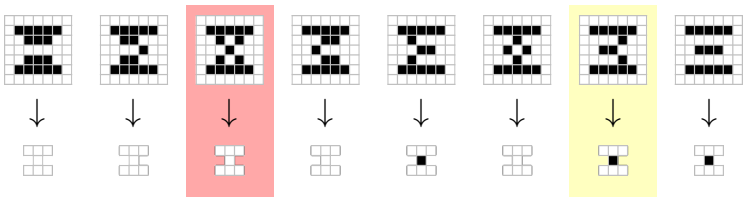


Figure 1.38: All except for one of the 8 different 3×3 squares (the one highlighted in red) in which the top and bottom rows are the negation of the middle row evolve so that the top and bottom rows die and the middle cell stays the same.

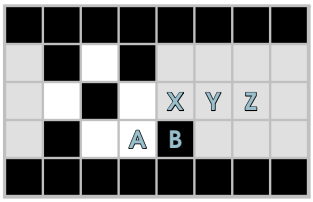


Figure 1.39: In order to correct the problematic 3×3 block from Figure 1.38, we set cell A to be dead and cell B to be alive, regardless of the state of cell X.

As a first attempt at fixing the problem that occurs when a single alive cell is between two dead cells in the middle row, simply set the bottom-right cell (i.e., cell A in Figure 1.39) of that 3×3 block as dead, rather than alive. To compensate for this cell being dead, we also set the cell to its immediate right (i.e., cell B in that figure) to be alive, regardless of whether cell X is alive or dead. This corrects the evolution of the problematic 3×3 arrangement highlighted in red in Figure 1.38: the central cell in this 3×3 block now stays alive (since it now has exactly 3 live neighbors) and the cell to its right stays dead (since it still has at least 4 live neighbors).

However, there is still a problem: if the 3×3 configuration to the immediate right (i.e., the one containing cells X, Y, and Z in Figure 1.39) is the one highlighted in yellow in Figure 1.38, then we also have to move the live cell at *its* bottom-right one cell to the right. If the *next* 3×3 configuration is also the one highlighted in yellow, we again move its bottom-right cell to the right, and we continue in this way until we finally encounter any other 3×3 configuration (or the end of the pattern). Straightforward case analysis then shows that the resulting pattern will evolve into the $1 \times (n-2)$ pattern in the central row. ■

Free PDF. See conwaylife.com/book for pattern files and print version.

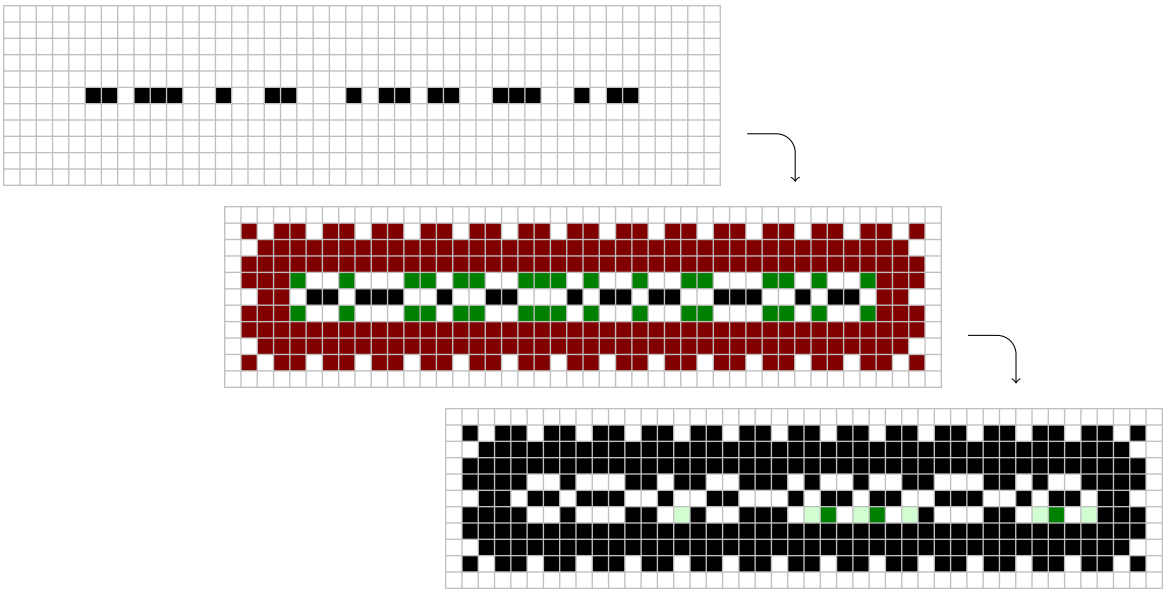


Figure 1.40: How to use the proof of Theorem 1.2 to construct a parent (at the bottom) of a pattern with height 1 (at the top). First, we place the border around the pattern (displayed in red in the middle image) and place the pattern's negation in the top and bottom rows of the box (displayed in green in the middle image). We then adjust the bottom row slightly (highlighted in green in the bottom image) whenever we see the troublesome configurations from Figure 1.39.

The method used in the proof of Theorem 1.2 is illustrated in Figure 1.40, where we start with a (randomly chosen) pattern with height 1 and explicitly construct a parent of it. More sophisticated techniques have been used to prove that there similarly do not exist orphans whose bounding boxes are 2 or 3 cells high.[40] On the other hand, there *does* exist an orphan (and thus a Garden of Eden) with height 5, as shown in Figure 1.41.[41] The question of whether or not there exists an orphan whose bounding box is 4 cells high remains open.

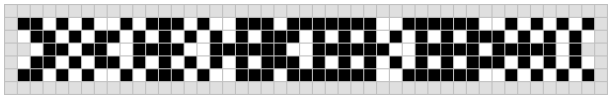


Figure 1.41: An orphan that fits within a 5×45 bounding box. No Garden of Eden or orphan with height less than 5 is currently known.

1.7.1 A Pattern with no Grandparent

There are many very difficult follow-up questions that can be asked about Gardens of Eden or patterns with properties related to those of Gardens of Eden. For example, does there exist a pattern that has a parent but no grandparent? In other words, does there exist a pattern with the property that all of its parents are Gardens of Eden? Although there is no known non-constructive argument like the one used in the proof of Theorem 1.1 that shows the existence of such patterns, it turns out that the answer to this question is "yes", and an example of such a pattern is presented in Figure 1.42. This pattern has the remarkable property that it has 17 920 distinct parents, every single one of which is a Garden

[40] This was proved by Steven Eker in 2016, using the same computer-assisted methods introduced by Jean Hardouin–Duparc.

[41] This orphan was also found by Steven Eker in 2016.

Free PDF. See conwaylife.com/book for pattern files and print version.

of Eden—one of these parents is displayed in Figure 1.43.[42]

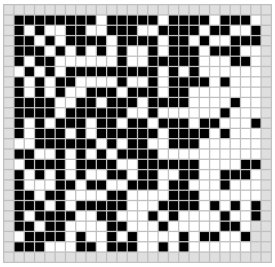


Figure 1.42: A pattern that has a parent but no grandparent.

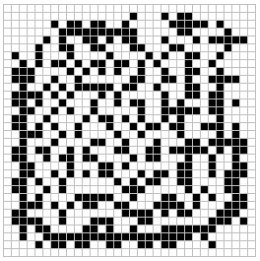


Figure 1.43: A parent of the grandparentless pattern from Figure 1.42.

The method used to construct this pattern is almost identical to the method we used to construct the Garden of Eden in Figure 1.35. That is, it was constructed one cell at a time, and each new cell that was added was chosen to be either alive or dead based on which of the two options resulted in the pattern having fewer grandparents.

1.8 Notes and Historical Remarks

Conway's Game of Life was initially studied by John Conway and some of his collaborators at Cambridge University, as well as a research group led by Bill Gosper at MIT, in 1969 and 1970. It then received mainstream attention in 1970 due to an article that Martin Gardner wrote about it in the magazine *Scientific American* [Gar70] (see Figure 1.44). Early discoveries by enthusiasts were sent to Gardner, who then shared those discoveries with Conway and, due to the overwhelming response to his article, wrote a follow-up article in February 1971 [Gar71].

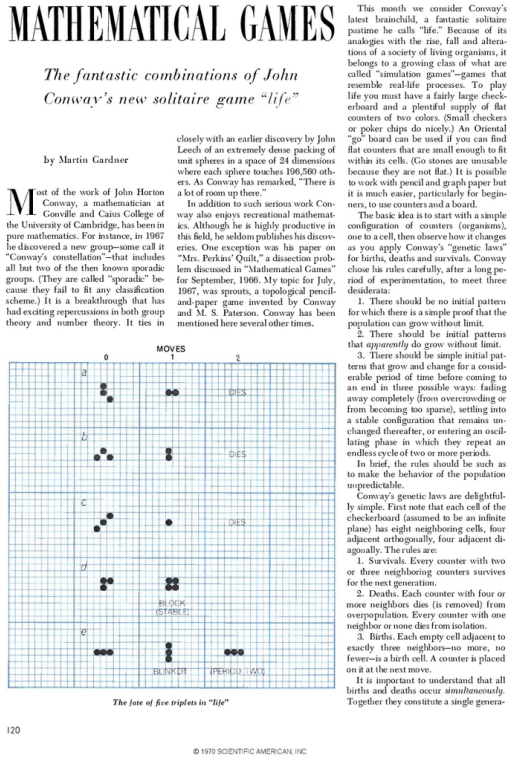
It was unsustainable for Martin Gardner to continue to serve as curator for all Life discoveries, and Scientific American would only agree to so many articles on the subject, so in March 1971 a quarterly newsletter called *Lifeline* was announced, which was edited and distributed by Robert Wainwright (see Figure 1.45). This newsletter had 11 editions and included the announcement of many of the objects that we saw in this chapter, including the twin bees gun, the switch engine (and its block-laying and glider-producing variants), and the acorn methuselah. Its final issue was dated September 1973.

From 1974 to 2009, Life enthusiasts mostly shared patterns via various private mailing lists, and collections of those patterns made their way to various personal web pages in the 1990s. In 2004, the LifeNews website was launched at `pentadecathlon.com`, which posted most interesting new Life discoveries as they were made. Since about 2009, most new discoveries in Life are now reported on the forums at `conwaylife.com`.

Although more advanced techniques are now known for constructing interesting Life objects (indeed, these techniques comprise the majority of the rest of this book), the idea of simply running randomly generated soups to completion to see what remains of them after they stabilize has been a consistent one. However, because computing power was at much more of a premium in 1970 than it is now, and there was no pre-made Life simulation software for early enthusiasts to use, these patterns

[42] Whether or not such a pattern exists was originally asked by Conway in October 1972. It was not resolved until May 2016, when the pattern displayed in Figure 1.42 was found by user "mtve" on the ConwayLife.com forums. Later that month, they even found a pattern that has a grandparent but no great-grandparent and a pattern with a great-grandparent but no great-great-grandparent. Finally, Ilkka Törmä and Ville Salo showed in January 2022 that, for every integer $n \geq 1$, there exists a pattern with a great^n-grandparent but no great^{n+1}-grandparent (see `conwaylife.com/forums/viewtopic.php?p=139854#p139854` for details).

Free PDF. See conwaylife.com/book for pattern files and print version.



MATHEMATICAL GAMES

The fantastic combinations of John Conway's new solitaire game "life"

by Martin Gardner

Most of the work of John Horton Conway, a mathematician at Gonville and Caius College of the University of Cambridge, has been in pure mathematics. For instance, in 1967 he discovered a new group—some call it "Conway's constellation"—that includes all but two of the then known sporadic groups. (They are called "sporadic" because they fail to fit any classification scheme.) It is a breakthrough that has had exciting repercussions in both group theory and number theory. It ties in closely with an earlier discovery by John Leech of an extremely dense packing of unit spheres in a space of 24 dimensions where each sphere touches 196,560 others. As Conway has remarked, "There is a lot of room up there."

In addition to such serious work Conway also enjoys recreational mathematics. Although he is highly productive in this field, he seldom publishes his discoveries. One exception was his paper on "Mrs. Perkins' Quilt," a dissection problem discussed in "Mathematical Games" for September, 1966. My topic for July, 1967, was sprouts, a topological pencil-and-paper game invented by Conway and M. S. Paterson. Conway has been mentioned here several other times.

This month we consider Conway's latest brainchild, a fantastic solitaire pastime he calls "life." Because of its analogies with the rise, fall and alterations of a society of living organisms, it belongs to a growing class of what are called "simulation games"—games that resemble real-life processes. To play life you must have a fairly large checkerboard and a plentiful supply of flat counters of two colors. (Small checkers or poker chips do nicely.) An Oriental "go" board can be used if you can find flat counters that are small enough to fit within its cells. (Go stones are unusable because they are not flat.) It is possible to work with pencil and graph paper but it is much easier, particularly for beginners, to use counters and a board.

The basic idea is to start with a simple configuration of counters (organisms), one to a cell, then observe how it changes as you apply Conway's "genetic laws" for births, deaths and survivals. Conway chose his rules carefully, after a long period of experimentation, to meet three desiderata:

1. There should be no initial pattern for which there is a simple proof that the population can grow without limit.
2. There should be initial patterns that *apparently* do grow without limit.
3. There should be simple initial patterns that grow and change for a considerable period of time before coming to an end in three possible ways: fading away completely (from overcrowding or from becoming too sparse), settling into a stable configuration that remains unchanged thereafter, or entering an oscillating phase in which they repeat an endless cycle of two or more periods.

In brief, the rules should be such as to make the behavior of the population unpredictable.

Conway's genetic laws are delightfully simple. First note that each cell of the checkerboard (assumed to be an infinite plane) has eight neighboring cells, four adjacent orthogonally, four adjacent diagonally. The rules are:

1. Survivals. Every counter with two or three neighboring counters survives for the next generation.
2. Deaths. Each counter with four or more neighbors dies (is removed) from overpopulation. Every counter with one neighbor or none dies from isolation.
3. Births. Each empty cell adjacent to exactly three neighbors—no more, no fewer—is a birth cell. A counter is placed on it at the next move.

It is important to understand that all births and deaths occur *simultaneously*. Together they constitute a single genera-

MOVES 0 1 2

a DIES

b DIES

c DIES

d BLOCK (STABLE)

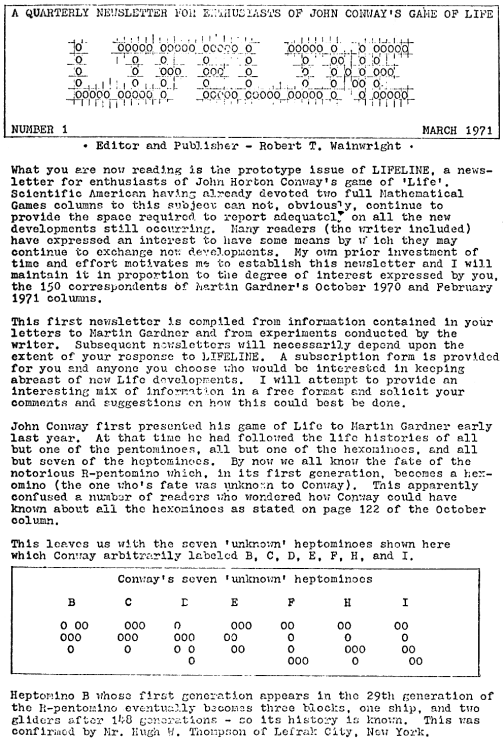
e BUNKER (PERIOD TWO)

The fate of five triplets in "life"

120

© 1970 SCIENTIFIC AMERICAN, INC

Figure 1.44: Conway's Game of Life was popularized by the October 1970 issue of *Scientific American*.



A QUARTERLY NEWSLETTER FOR ENTHUSIASTS OF JOHN CONWAY'S GAME OF LIFE

NUMBER 1 MARCH 1971

• Editor and Publisher - Robert T. Wainwright •

What you are now reading is the prototype issue of LIFELINE, a newsletter for enthusiasts of John Horton Conway's game of 'Life'. Scientific American having already devoted two full Mathematical Games columns to this subject can not, obviously, continue to provide the space required to report adequately on all the new developments still occurring. Many readers (the writer included) have expressed an interest to have some means by w ich they may continue to exchange new developments. My own prior investment of time and effort motivates me to establish this newsletter and I will maintain it in proportion to the degree of interest expressed by you, the 150 correspondents of Martin Gardner's October 1970 and February 1971 columns.

This first newslettor is compiled from information contained in your letters to Martin Gardner and from experiments conducted by the writer. Subsequent newsletters will necessarily depend upon the extent of your response to LIFELINE. A subscription form is provided for you and anyone you choose who would be interested in keeping abreast of new Life developments. I will attempt to provide an interesting mix of information in a free format and solicit your comments and suggestions on how this could best be done.

John Conway first presented his game of Life to Martin Gardner early last year. At that time he had followed the life histories of all but one of the pentominoes, all but one of the hexominoes, and all but seven of the heptominoes. By now we all know the fate of the notorious R-pentomino which, in its first generation, becomes a hexomino (the one who's fate was unknown to Conway). This apparently confused a number of readers who wondered how Conway could have known about all the hexominoes as stated on page 122 of the October column.

This leaves us with the seven 'unknown' heptominoes shown here which Conway arbitrarily labeled B, C, D, E, F, H, and I.

Conway's seven 'unknown' heptominoes

B C D E F H I

Heptomino B whose first generation appears in the 29th generation of the R-pentomino eventually becomes three blocks, one ship, and two gliders after 148 generations - so its history is known. This was confirmed by Mr. Hugh W. Thompson of Lefrak City, New York.

Figure 1.45: The front page of the first issue of *Lifeline,* a newsletter for Life enthusiasts that ran from March 1971 to September 1973.

were often evolved by hand using graph paper, checkers, or the board and stones from the game Go, to represent the Life grid and its live cells.

Furthermore, the methodology that was used back then was somewhat more systematic—instead of generating large random starting configurations, early Lifers investigated the evolution of all possible small starting configurations. These investigations were exactly where the original interest in the T-tetromino, R-pentomino, B-heptomino, stairstep hexomino, and other small polyominoes came from. In fact, the switch engine was originally found by Charles Corderman when he was investigating the evolution of nonominoes (i.e., polyominoes with 9 live cells), one of which evolves in the exact same way as the switch engine (see Figure 1.46).

Solomon W. Golomb invented the term "polyomino" in 1953, and Martin Gardner wrote a popular Mathematical Games article about them a decade before the first article on Conway's Game of Life. Perhaps partly because of this long-standing interest in polyominoes in the mathematical community, the fates of N-cell polyominoes were exhaustively researched very early on, giving rise to the common names of many of the active patterns discussed above. The Moore-neighborhood equivalent of polyominoes, the N-cell **polyplets**—cell groups that are kingwise-connected rather than rookwise-connected— are technically just as relevant as a potential source of novelty, but their evolutionary fates were much less thoroughly explored in the early years than the polyominoes.

Building upon this idea of finding objects by evolving random starting configurations, Achim Flammenkamp ran an automated computer search in 1994 that repeatedly generated random soups and evolved them, keeping a list of all objects that it found in the resulting ash. This search ran until it had accumulated 5×10^9 (non-distinct) ash objects, which contained a total of 48 distinct oscillators [Fla94]. He then performed this search again in 2004, this time accumulating 5×10^{10} ash objects,

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) A nonomino. **(b)** A switch engine.

Figure 1.46: The nonomino (a) evolves in the same way as the switch engine (b) after 2 generations.

and found over 3 500 distinct still lifes and over 80 different oscillators [Fla04].

Andrzej Okrasinski created a similar program that acted as a screensaver in November 2003. Over the subsequent 5 years, this screensaver cataloged over 4.7×10^{11} ash objects, including over 8 000 distinct still lifes and about 180 oscillators, but still the only spaceships that turned up were the four that we have already seen (the glider, LWSS, MWSS, and HWSS) [Okr03]. The screensaver also kept track of how long each starting configuration took to stabilize, and it found some new methuselahs in the process, including a 15-cell version of Lidka, which was presented in Table 1.1 (it was improved to a 13-cell version by David Bell and then to the more compact 13-cell version presented in that table by Simon Ekström).

In 2009, Nathaniel Johnston created a distributed online search to continue in this vein, called *The Online Life-Like CA Soup Search* (or *TOLLCASS* for short). This script had the advantage that multiple people could run it simultaneously on different computers, and their results were automatically uploaded to a central server that organized them. TOLLCASS also worked not just with Conway's Game of Life, but also with a handful of other Life-like cellular automata. The major downside of this script was that it was quite slow, and over the two years that it was active, it cataloged only about one third as many objects as Okrasinski's earlier search.

Finally, Adam P. Goucher launched another distributed online search in 2014 called *apgsearch*,[43] which is still active today[44] and is the current state-of-the-art when it comes to soup searching. Like TOLLCASS, it can be used with several different Life-like CA, but with the main advantage of being extremely fast. In the seven years since it began, it has cataloged about 10 000 times as many ash objects as all of the previous searches combined (and this number will likely become out of date very quickly). A summary of these various soup searches that have taken place over the years is provided by Table 1.2.

Search Name	Year(s)	Ash Objects Found	Notes
Flammenkamp	1994	5.0×10^9	first automated soup search
Flammenkamp	2004	5.0×10^{10}	–
Okrasinski	2003–2008	4.7×10^{11}	also found methuselahs
TOLLCASS	2009–2011	1.7×10^{11}	online search, multiple CA
apgsearch	2014–present	1.5×10^{15}	online search, multiple CA

Table 1.2: A summary of the different soup searches that have taken place over the years.

[43] Officially, the "apg" in apgsearch are not its author's initials, but rather stand for **a**sh **p**attern **g**enerator.

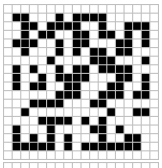
[44] It is available at `catagolue.hatsya.com/apgsearch`.

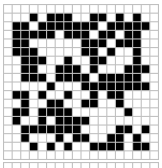
Free PDF. See conwaylife.com/book for pattern files and print version.

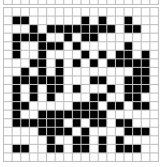
Exercises

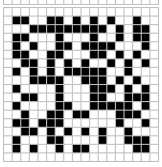
solutions to starred exercises on page 451

∗**1.1** [1/5] Evolve each of the following randomly generated 16×16 soups and describe the most unusual object that forms.

(a) 

(b) 

(c) 

(d) 

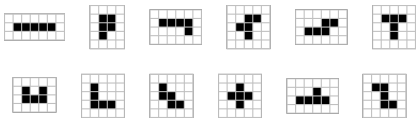
1.2 What is the longest lifespan of a pattern with:

(a) [1/5] 1 or 2 live cells?

(b) [2/5] 3 live cells?

(c) [3/5] 4 live cells? [Hint: You could try writing a computer program to help you.]

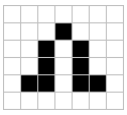
1.3 [1/5] All 12 different pentominoes are displayed below. What are their lifespans?



1.4 In this exercise, we will find some alternate stabilizations of the twin bees shuttle from Figure 1.22.

(a) [1/5] Try removing some of the blocks from the twin bees shuttle. Which combinations of blocks can you remove while preserving it as a period 46 oscillator?

(b) [2/5] The still life below is called a **hat**. Use a hat to stabilize one side of the twin bees shuttle.



∗**1.5** [2/5] Find a specific generation at which each of the following unstable objects appears in the evolution of the R-pentomino.

(a) A T-tetromino.

(b) A pre-honey farm.

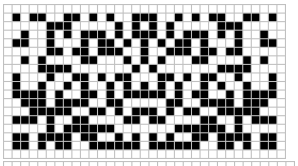
(c) A pi-heptomino.

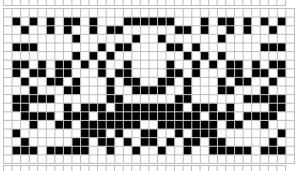
(d) A queen bee.

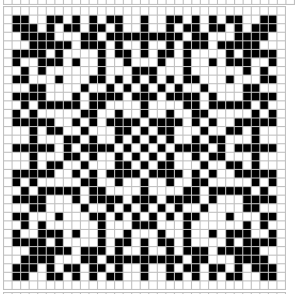
(e) Lumps of muck (in particular, generation 3 of the stairstep hexomino).

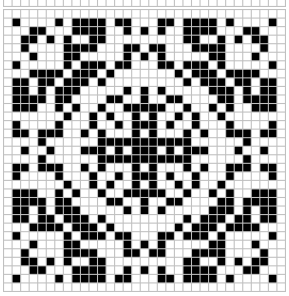
1.6 [3/5] Use two blocks and three queen bees to create a "double" Gosper glider gun: a gun that emits two streams of gliders.

∗**1.7** [1/5] Because so many interesting Life objects display some form of symmetry, it is often fruitful to investigate random starting configurations that are also symmetric. Evolve each of the following randomly generated symmetric soups and describe the most unusual object that forms.

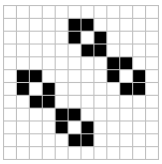
(a) 

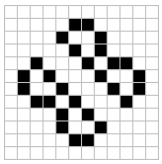
(b) 

(c) 

(d) 

∗**1.8** [3/5] There are 5 formations of 4 simple objects that commonly occur in ash: the traffic light, honey farm, and blockade that we saw in Section 1.2, and the **fleet** and **bakery** displayed below on the left and right, respectively. These arrangements are collectively called the **familiar fours**.



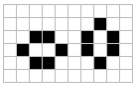


(a) Evolve Lidka until you see a fleet, and use this evolution to find a 7-cell predecessor of the fleet.

(b) There are at least two different 8-cell objects that evolve into a bakery. Find one of them. [Hint: Try evolving random soups until you find a bakery, or write a computer program that evolves many different 8-cell objects.]

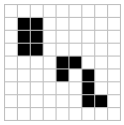
Free PDF. See conwaylife.com/book for pattern files and print version.

∗**1.9** [2/5] This problem concerns the configuration of two beehives displayed below.



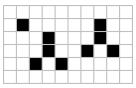
(a) Evolve the pattern until it stabilizes. What happens to the beehives?
(b) Create a period 30 oscillator that uses this reaction to stabilize two queen bees that are rotated 90 degrees from each other.

∗**1.10** [2/5] This problem concerns the pattern displayed below. The top-left object is a pre-beehive, while the bottom-right object is called **eater 1**.



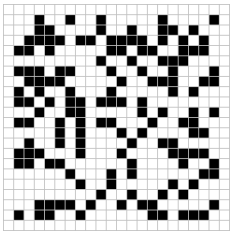
(a) Evolve the pattern until it stabilizes. What happens to the pre-beehive and eater 1?
(b) Evolve the pre-beehive and eater 1 individually (i.e., not next to each other). Describe what happens to them.
(c) Evolve a queen bee until you find a pre-beehive that it leaves behind.
(d) Use an eater 1 (instead of a block) to stabilize one side of the queen bee shuttle.

1.11 [1/5] The 9-cell methuselah displayed below is called **bunnies**.[45]



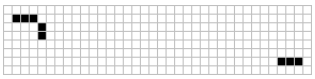
(a) What is its lifespan?
(b) It eventually evolves in the same way as each of bunnies 9 and bunnies 10b from Table 1.1. Find the first generation of each object's evolution where they match.

1.12 [1/5] The methuselah displayed below is named **Edna**.[46]

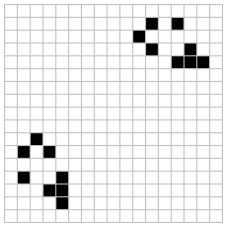


(a) What is its lifespan?
(b) What objects can be found in its ash after this pattern stabilizes? Are there any objects that we did not give names to in this chapter?

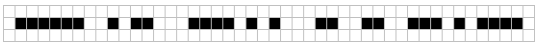
1.13 [1/5] An 8-cell methuselah with a longer lifespan than the 8-cell methuselah from Table 1.1 (but with a significantly larger bounding box) is displayed below.[47] What is its lifespan?



1.14 [2/5] An ark called **Noah's ark** is displayed below.[48] What is the period of this pattern, and what objects does it leave behind as it moves?



1.15 [2/5] Find a parent of the following 1-cell-thick pattern:



[45] Its 1-generation successor, which is called **rabbits**, was found by Andrew Trevorrow in 1986. This version was subsequently found independently by Robert Wainwright and Andrew Trevorrow.

[46] Found by Erik de Neve in January 2010.

[47] Found by Nick Gotts in 2019.

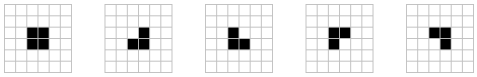
[48] This ark was found in 1971 by Charles Corderman, and was the first-discovered ark. Its name refers to the fact that the debris it leaves behind contains pairs of many different objects. The general term "ark" is derived from the name of this pattern.

Free PDF. See conwaylife.com/book for pattern files and print version.

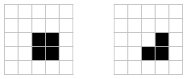
∗**1.16** [4/5] In the proof of Theorem 1.1, we claimed that the inequality $(2^{36}-1)^{n^2} < 2^{(6n-2)^2}$ holds whenever n is large enough.

(a) Find a formula for the smallest value of n for which this inequality holds, and then use computer software to compute its exact value.

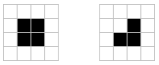
(b) We can get slightly smaller values of n that lead to Gardens of Eden by considering more tiles that evolve in the same way. Instead of just using the two 6×6 tiles that we presented in the proof of Theorem 1.1, we could have used the set of *five* tiles displayed below without changing the method of proof significantly. What is the relevant inequality that we need to check? What is the smallest value of n for which this new inequality holds?



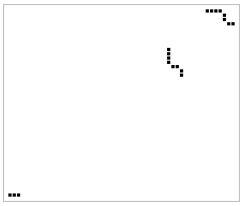
(c) Alternatively, we can get smaller values of n leading to Gardens of Eden by using 5×5 tiles. If we re-do the proof of Theorem 1.1 using the 5×5 tiles displayed below, what is the relevant inequality that we need to check? What is the smallest value of n for which this new inequality holds?



(d) Why can't we replace the 6×6 or 5×5 tiles with the 4×4 tiles displayed below? What part of the proof breaks down?



∗**1.17** [2/5] The 19-cell pattern displayed below consists of two switch engine predecessors and a blinker,[49] and takes a staggering 6526574 generations to stabilize.



(a) Evolve the pattern to see how it behaves. Why does it live so much longer than the ark in Figure 1.30?

(b) What causes this ark to stabilize?
[Hint: Compare the evolution of this ark with the evolution presented in Figure 1.31, which shows the stabilization happening as a result of a path being cleared for the backward-flying gliders.]

1.18 [3/5] In the proof of Theorem 1.1, we claimed that the inequality $(2^{36}-1)^{n^2} < 2^{(6n-2)^2}$ holds whenever n is large enough. Prove the stronger statement that

$$\lim_{n\to\infty} \frac{(2^{36}-1)^{n^2}}{2^{(6n-2)^2}} = 0,$$

and hence the vast majority of large patterns are Gardens of Eden. [Hint: Use the fact that $a^b = 2^{b\log_2(a)}$.]

1.19 A computer program called *JavaLifeSearch*[50] can be used to find predecessors of Life patterns (or show that none exist). Usage instructions and download links can be found at `conwaylife.com/wiki/JavaLifeSearch`.

(a) [3/5] Use JavaLifeSearch to find a predecessor of the following pattern:



(b) [4/5] Use JavaLifeSearch to verify that the patterns from Figure 1.36 really are orphans.

(c) [5/5] Use JavaLifeSearch to verify that the pattern from Figure 1.42 really does not have any grandparents.

[49] This pattern was found by Nick Gotts in February 2005.

[50] Created by Karel Suhajda, based on earlier programs by David Bell, Dean Hickerson, and Jason Summers.

Free PDF. See conwaylife.com/book for pattern files and print version.

Free PDF. See conwaylife.com/book for pattern files and print version.



2. Still Lifes

Stand still. The trees ahead and the bushes beside you are not lost.

David R. Wagoner

In this chapter, we investigate the simplest possible objects in Life: those which do not move at all, but rather stay exactly the same from one generation to the next. As we have already learned, such objects are called **still lifes**, and we have seen several examples of them, including the block, tub, boat, ship, beehive, and loaf. Some additional small still lifes are the **aircraft carrier**, **barge**, **snake**, **long boat**, **long snake**, and **eater 1** (see Figure 2.1). In fact, we have just listed every single still life with 7 or fewer live cells.

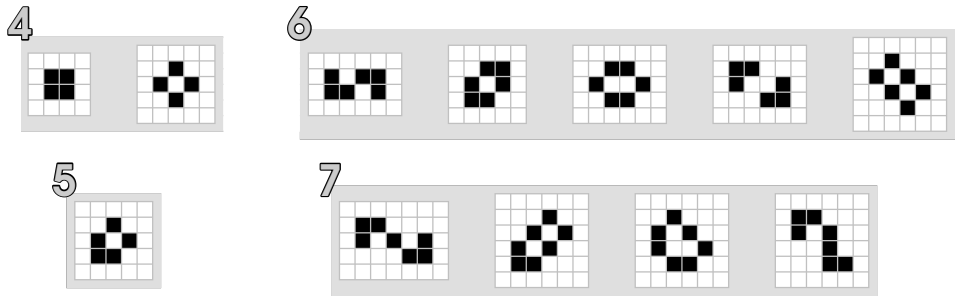


Figure 2.1: All still lifes with 7 or fewer live cells, arranged by their cell count. From left to right: (4 cells) block, tub, (5 cells) boat, (6 cells), snake, ship, beehive, aircraft carrier, barge, (7 cells) long snake, long boat, loaf, and eater 1 (which is sometimes called **fishhook**).

While it might seem like there is not much to say about such simple objects, they actually have a somewhat complicated structure and some interesting connections to other areas of mathematics. Furthermore, familiarity with still lifes, especially the small common ones, will be extremely important when constructing larger and more complicated patterns—we already saw this in the previous chapter,

Free PDF. See conwaylife.com/book for pattern files and print versio

where we strategically used blocks to turn queen bees and twin bees into oscillators and glider guns.

2.1 Strict and Pseudo Still Lifes

We saw in Figure 2.1 a complete list of all still lifes with 7 or fewer cells. We would like to continue in this way and create exhaustive lists of all still lifes with $8, 9, 10, \ldots$ live cells as well, but something strange happens in these cases—we have the freedom to place multiple disjoint copies of still lifes at various places in the plane. For example, we could place two blocks far away from each other at various locations, thus creating an infinite collection of (trivial) 8-cell still lifes.

Still lifes constructed in this way are in a sense not really any different from their component still lifes, so we would like to ignore them when creating exhaustive lists like this. With this in mind, we define a **strict still life** to be a still life that is either connected, or is disconnected but its different connected components cannot be partitioned into two or more sets that are still lifes by themselves (see Figure 2.2). The reason for not solely restricting our attention to connected still lifes is that some still lifes, like the aircraft carrier, are disconnected yet really are non-trivial—if we separate their two pieces, they are no longer stable.

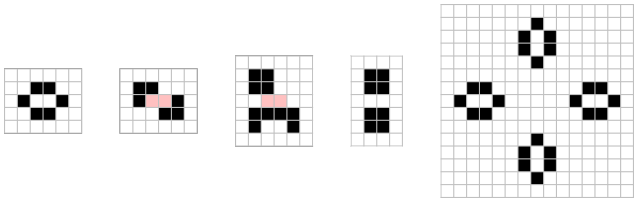


Figure 2.2: From left to right, these still lifes are the beehive, aircraft carrier, **block on table**, **bi-block**, and honey farm. The beehive is a strict still life because it is connected, the aircraft carrier is a strict still life because its two halves are not still lifes by themselves but together they overpopulate the cells highlighted in red, and the block on table is a strict still life for the same reason. However, the bi-block and honey farm are *not* strict still lifes, since they are made up of non-interacting simpler still lifes.

However, there is a somewhat weaker notion of whether or not a still life is "trivial", and it is highlighted in the difference between the bi-block and the honey farm from Figure 2.2. In the bi-block, the two blocks are close enough to each other that they are "almost touching" and their Moore neighborhoods overlap, whereas the beehives in the honey farm are far enough away from each other that they simply do not interact with each other at all. To capture this difference, we define a **pseudo still life** to be a disconnected still life with the property that its connected components can be partitioned into two or more sets that are individually still lifes,[1] and furthermore at least one dead cell has more than 3 live neighbors in the overall pattern but has fewer than 3 live neighbors in the subpatterns[2] (see Figure 2.3 for some examples).

With these definitions cleared up, we are now able to list the still lifes with small numbers of live cells. Since the number of still lifes with a given number of live cells grows so quickly, we do not explicitly list all of them,[3] and instead we just say how many there are of each size and give a few examples in Table 2.1. For space reasons, we only list the number of still lifes up to 23 cells in size, but we note that they have been computed up to 34 cells [Slo96, Slo00].

[1] In the early days of Life, it was not uncommon to define pseudo still lifes as requiring that they can be partitioned into *exactly* two individual still lifes. Matthew Cook demonstrated that the difference between these two definitions is actually quite important, since determining whether or not a pattern can be split into exactly two individual still lifes is relatively easy, whereas determining whether or not it can be split into two *or more* still lifes is quite hard [Coo03].

[2] This final restriction is made to capture the notion of the individual connected components "almost touching".

[3] For a complete list of all still lifes with 18 or fewer live cells (and massive lists of larger still lifes as well), see Mark Niemiec's Life pages at conwaylife.com/ref/mniemiec/lifepage.htm.

Free PDF. See conwaylife.com/book for pattern files and print version.

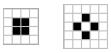
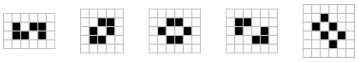
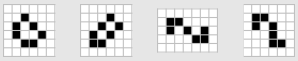
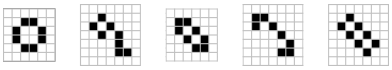
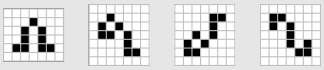
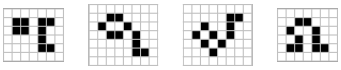
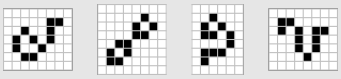
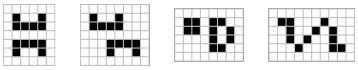
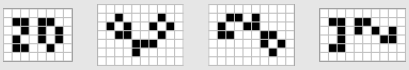
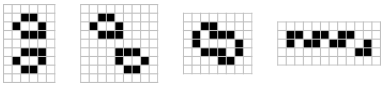
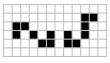
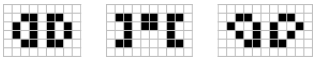
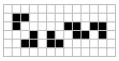
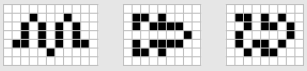
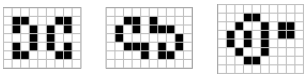
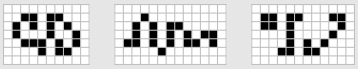
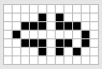
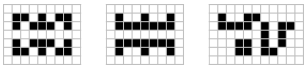
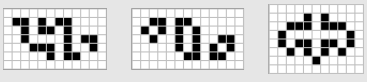
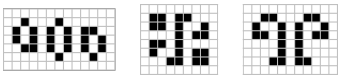
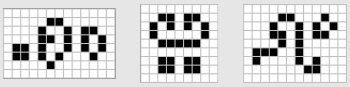
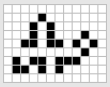
Cells	# of Strict	Examples	# of Pseudo	Example
4	2		0	–
5	1		0	–
6	5		0	–
7	4		0	–
8	9		1	
9	10		1	
10	25		7	
11	46		16	
12	121		55	
13	240		110	
14	619		279	
15	1353		620	
16	3286		1645	
17	7773		4067	
18	19044		10843	
19	45759		27250	
20	112243		70637	
21	273188		179011	
22	672172		462086	
23	1646147		1184882	

Table 2.1: A summary of the still lifes with 23 or fewer live cells.

Free PDF. See conwaylife.com/book for pattern files and print version.

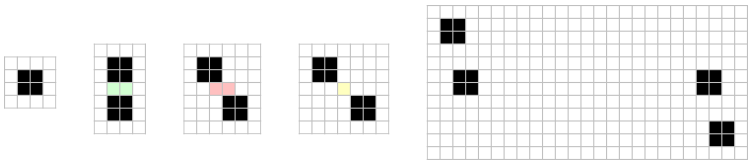
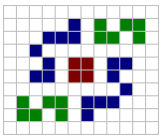
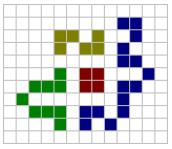


Figure 2.3: From left to right, these are the block, bi-block, two nameless configurations of two blocks, and the blockade. The bi-block is a pseudo still life because of the cells highlighted in green between the blocks that have more than 3 live neighbors. The middle configuration is not a still life at all since the two cells highlighted in red will come to life in the next generation. The fourth and fifth configurations are neither strict nor pseudo still lifes, since there is no dead cell that has more than 3 live neighbors between any of the pairs of blocks—the cell highlighted in yellow is in the neighborhood of both blocks but is still underpopulated (see Exercise 2.2).

Given that the definition of a pseudo still life involves being able to partition the still life into 2 *or more* component still lifes, it seems natural to ask whether or not there really are cases that can be decomposed into (for example) 3 still lifes but not 2. Still lifes with this property are indeed known, and the smallest such example is displayed in Figure 2.4(a). Similarly, there are pseudo still lifes that can be decomposed into 4 still lifes, but not 2 or 3, such as the one shown in Figure 2.4(b).[4] Somewhat surprisingly, this pattern does not continue: there does *not* exist a pseudo still life that can only be partitioned into 5 or more still lifes.



(a) A pseudo still life that can be partitioned into 3 still lifes, but not 2.



(b) A pseudo still life that can be partitioned into 4 still lifes, but not 2 or 3.

Figure 2.4: Pseudo still lifes that can be partitioned into (a) 3 still lifes and (b) 4 still lifes, but not fewer. The way of partitioning the still lifes is indicated by the different colors.

Theorem 2.1 — Four-Partitions of Pseudo Still Lifes
Every pseudo still life can be partitioned into 4 or fewer still lifes.

Proof. First, partition the pseudo still life into its (perhaps much more than 4) strict still life components. Around each of those strict still lifes, outline the region that extends by 1/2 cell in all directions, as demonstrated in Figure 2.5(b). The resulting regions do not overlap except for perhaps sharing borders with each other, so the four color theorem[5] says that we can use four colors to color these regions in such a way that no two regions with a common border have the same color.[6]

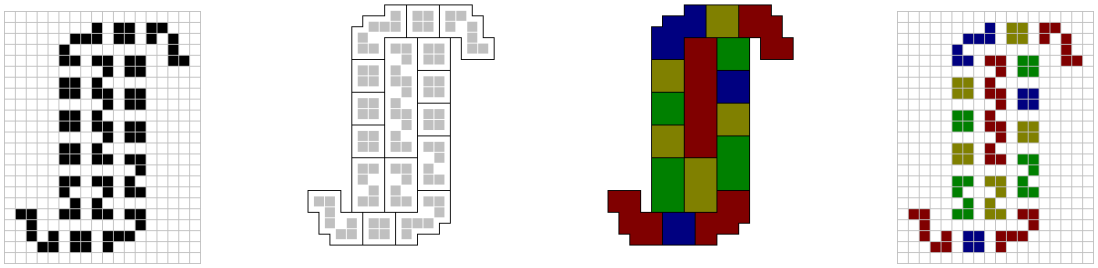
We claim that if we group the strict still lifes according to the color of the region that they are contained in then each of the (no more than 4) resulting patterns is a still life. To see this, we note that no bordering regions have the same color, so the only way that some of the strict still lifes that are grouped together can have any cells in common in their Moore neighborhoods is if those neighborhoods overlap at a corner (such as the yellow regions and yellow still lifes near the bottom of

[4]The first pseudo still lifes with these properties were found by Matthew Cook around 1998. These slightly smaller ones were found by Gabriel Nivasch in 2001, and they were proved to be minimal via computer search in early 2020.

[5]The four-color theorem is a famous (and notoriously difficult to prove) mathematical result that says that if we separate a 2D plane into regions, we can always use four or fewer colors to color the regions in such a way that no two adjacent regions have the same color.

[6]Regions that only touch at a corner are allowed to have the same color.

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) A pseudo still life that we want to partition into four stable sub-patterns.

(b) The regions that extend $1/2$ of a square out from each component strict still life.

(c) We can color these regions using four or fewer colors, by the four color theorem.

(d) We then partition the pseudo still life according to the coloring.

Figure 2.5: A demonstration of the proof that every pseudo still life can be partitioned into four (or fewer) sets that are individually stable.

Figures 2.5(c) and 2.5(d), respectively). However, in this case the in-between dead cell only has two live neighbors and thus the (no more than 4) configurations of still lifes are all stable. ■

2.2 Still Life Grammar

We saw in Table 2.1 that computers have been used to find all still lifes with small numbers of live cells. However, if we want to find a still life with a certain specific shape or combination of properties, there are typically much easier ways to find one than by using an exhaustive computer search. In this section, we will discuss some of the standard methods for constructing, combining, and extending still lifes.

As a simple first example, we note that many still lifes that we have seen come as parts of large families of still lifes that naturally build upon each other. For example, the long boat is obtained from the boat simply by adding an extra two diagonal cells, and the long ship and barge are similarly obtained from the ship and the tub, respectively. We can repeatedly lengthen any of these still lifes in the same way, adding two cells at a time, to construct (rather trivial) still lifes with any number of cells as in Figure 2.6.

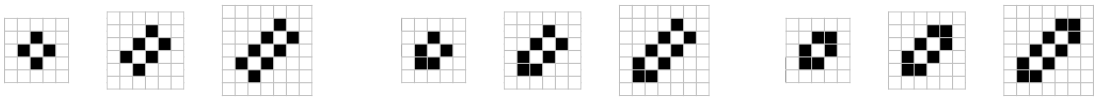


Figure 2.6: From left to right, these still lifes are the tub, barge, **long barge**, boat, long boat, **long long boat**, ship, **long ship**, and **long long ship**. Each of these still lifes can be made as long as we like by continuing the patterns in the obvious way.

Some similar families of still lifes come from the snake and long snake, where now we only have to add a single extra diagonal cell every time that we want to lengthen it, as in Figure 2.7.[7]

In all of these examples, the still lifes are made up of two very different components: the central portion that can be repeated indefinitely, and the stabilizing end pieces. The 3-cell pre-block on the ends of each side of the long snakes and long canoes is one commonly used end piece, and another one is the 4-cell **tail** depicted in Figure 2.8. When naming still lifes that use this component, we

[7]These elongated versions of still lifes are denoted by using the "long" prefix the number of times that it has been lengthened (e.g., a "long long snake" is a snake elongated by 2 cells). Alternatively, "very" and "extra" imply 1 and 2 extra levels of longness: $(n+1)$-th smallest $=$ long^n = very^{n-1} long = extra^{n-2} long. For example, a long^3 snake is a long long long snake is a very very long snake is an extra long snake.

Free PDF. See conwaylife.com/book for pattern files and print version.

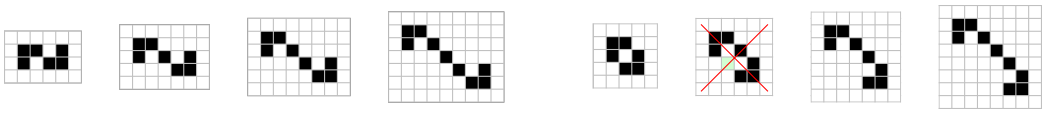


Figure 2.7: From left to right, these objects are the snake, long snake, **long long snake**, **long^3 snake**, ship, an object that is not a still life due to the green cell coming to life, **canoe**, and **long canoe**. Again, each of these still lifes can be made as long as we like by continuing the patterns.

typically use the "with tail" suffix so that the 4th still life in Figure 2.9, for example, is called the **tub with tail**.[8]

In order to construct larger still lifes, we use the fact that if every cell in a 2×2 square is alive, then every cell that is an immediate neighbor of that 2×2 square must be dead (i.e., the 2×2 square must in fact be a block with a border of dead cells around it) in order for the resulting pattern to be a still life. The reason for this is that each cell in the 2×2 square already has 3 live neighbors in that 2×2 square, so adding another one will overcrowd it, causing it to not be stable.[9]



Figure 2.8: A pre-block (left) and a **tail** (right) are two common ways of stabilizing objects (typically when the cells highlighted in aqua are alive).

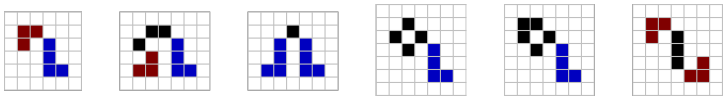


Figure 2.9: Several examples of still lifes that use pre-blocks (highlighted in red) and tails (highlighted in blue) to stabilize their ends. Note that the tails in the 4th and 5th still lifes are not actually required for stability, but just serve to make the still lifes larger.

This tells us that still lifes never have "thick" sections—they are made up of single-cell-thick "paths" of live cells that potentially branch, curve, and loop around on themselves, possibly plus some isolated blocks. Furthermore, it is often[10] possible to take a single-cell-thick path (which does not have overpopulated cells like in Figure 2.10) and add extra junk around it to suppress the birth of nearby dead cells, thus resulting in a still life (see Figure 2.11 for an example).

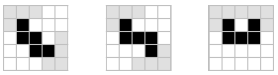


Figure 2.10: Three path segments that cause overcrowding of the central cell and thus must be avoided in still lifes.

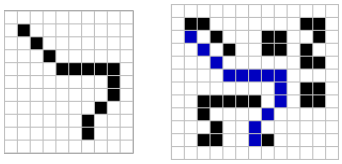


Figure 2.11: A single-cell-thick path of live cells (left) can typically be stabilized by adding other nearby objects and branching paths (right).

While there is no completely foolproof way of carrying out this procedure (some trial-and-error is typically used), some general rules of thumb include using pre-blocks or tails on the ends of the path, doubling the thickness of diagonal sections of the path, and stabilizing orthogonal sections of the path with either orthogonal sections of a new path or with other still lifes like blocks or snakes. When

[8] Some objects, like the boat, can have a tail added to them in two different ways, so names like **boat with tail** become insufficient for distinguishing between them. We thus use the **cis** and **trans** prefixes (borrowed from organic chemistry) to distinguish cases like this where they were two possible tail orientations. For example, the two boats with tails displayed in the 9-cell strict still life row of Table 2.1 are called the **cis-boat with tail** and the **trans-boat with tail**.

[9] The term **stable** is sometimes used to refer to still lifes, and it is sometimes used to refer to both still lifes and oscillators. Which of these two cases is meant is usually clear from context or irrelevant.

[10] But not always—see Exercise 2.12.

Free PDF. See conwaylife.com/book for pattern files and print version.

objects are used to stabilize other (non-touching) nearby objects, like how we used the blocks and the snake in Figure 2.11 to stabilize the top-right section of our original path, they are called **induction coils**.

Induction coils are most frequently used along long sections of orthogonally connected live cells. For example, a section of 5 orthogonally connected live cells can be stabilized by any still life that has a single cell farther in one direction than any of its other cells, such as a tub or an eater 1 (or almost any other still life with a tail), since this single cell overpopulates all 3 dead cells that would otherwise be born—see Figure 2.12. Blocks and snakes are useful because they can be placed next to each other with gaps of 1 or 2 dead cells between them, to stabilize rows of connected cells of any length, as in Figure 2.13 (which also illustrates several other common induction coils).

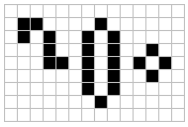


Figure 2.12: A tub, a boat, a loaf, a still life with a tail, or any other "pointy" still life can be used as an induction coil to prevent the birth of 3 orthogonally connected cells.

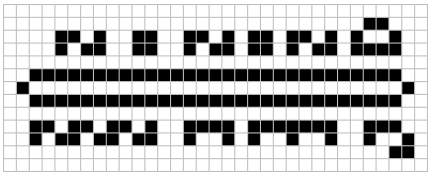


Figure 2.13: A demonstration of several induction coils being used to stabilize an object. The induction coil at the top-right corner is the **cap**, in the bottom-middle are a **table** and a way of extending it, and at the bottom-right is a **bookend**.

2.3 Eaters

Many of the complex patterns that we construct in later sections of this book will be based on sending gliders from one place to another, so we will need simple ways of creating, moving, and deleting gliders. Deleting gliders is the simplest of these tasks, and it can be done with objects called **eaters**: still lifes[11] with the property that if a glider (or another object) collides with them in the right away, the glider is deleted and the eater suffers no permanent damage.

The smallest, first discovered,[12] and most widely used glider eater is the 7-cell still life called eater 1, whose glider-eating reaction is displayed in Figure 2.14. The reason for eater 1's widespread use is not only due to its small size, but also because it returns to its original state only 4 generations after being hit by the glider. In other words, it has a **recovery time** of 4 generations, which is the fastest possible for any glider eater.[13]

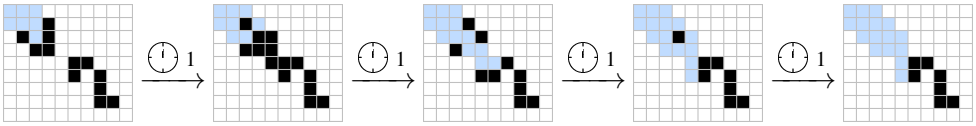


Figure 2.14: Eater 1 is a seven-cell still life that takes 4 generations to eat a glider.

In addition to eating gliders, eater 1 can also be used to eat lightweight spaceships, middleweight spaceships, blinkers, and numerous other objects (see Figure 2.15). The fact that eater 1 remains

[11] Strictly speaking, an eater does not need to be a still life, but still life eaters are used so much more frequently than other types of eaters that it is often assumed.

[12] Eater 1 itself was almost immediately discovered independently by several Life enthusiasts as the smallest asymmetric still life, but its eating properties were discovered by Bill Gosper's group at MIT in 1971.

[13] There are also other glider eaters that tie its recovery time of 4 generations—see Exercise 2.13. Computer searches have demonstrated that no still life can completely eat a glider and recover in only 3 generations.

Free PDF. See conwaylife.com/book for pattern files and print version.

unaffected by so many different types of debris hitting its top-left corner makes it extremely useful not just as an eater, but also as a stabilizer in numerous more complicated patterns. We already demonstrated this phenomenon in Exercise 1.10, where we used eater 1 to stabilize a queen bee, and we will see how eater 1 can be used to construct various other oscillators in Section 3.2. Furthermore, eater 1 will appear repeatedly when we construct glider loops and Herschel tracks in Sections 3.5 and 3.6, respectively.

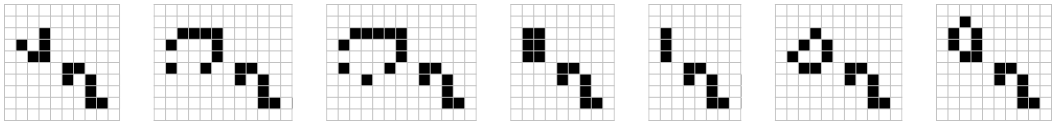


Figure 2.15: Eater 1 can be used to eat a glider, a lightweight spaceship, a middleweight spaceship, a pre-beehive, and many other objects.

In spite of the versatility of eater 1, there are other eaters that are sometimes more useful in certain situations. One example is **eater 2** (see Figure 2.16(a)): although it is quite a bit larger than eater 1 and has a recovery time of 5 generations (instead of 4), it has the advantages of being symmetric and being able to eat gliders traveling along 4 different parallel paths.

Similarly, **eater 5** (sometimes called the **tub with tail eater**, or **TWIT** for short) is a small eater[14] with a recovery time of 6 generations that is made up of two still lifes and is capable of eating gliders traveling along 2 different perpendicular paths. Eater 5 is especially useful for the fact that it can eat gliders traveling so close to its edge (in particular, the glider coming from the top-right corner in Figure 2.16(b)), so it can often be used to eat gliders in tight places where other eaters will not fit.



(a) Eater 2. **(b)** Eater 5.

Figure 2.16: Eater 2 (left) is capable of eating gliders traveling along 4 different parallel paths, while eater 5 (right) is capable of eating gliders traveling along 2 different perpendicular paths.

2.3.1 Rocks and "Almost" Eaters

All of the eaters that we have seen so far are temporarily disturbed when the glider hits them, and it takes a few generations for them to return to their initial state. However, there is no fundamental reason that an eater has to be disturbed by the object that it eats at all—an eater is called a **rock** if it does not even suffer temporary damage during the eating process. While there are no known rocks that eat gliders, there are rocks that eat other objects, and there are objects that can act as rocks for *multiple* gliders.

Here we present an example of the latter kind—an object that can destroy two gliders, suffer no damage in the process, yet cannot completely destroy just a single glider. In fact, this object is simply the snake, which is capable of turning a single glider into a boat, which then destroys (and is destroyed

[14] There are indeed eaters called "eater 3" and "eater 4" (as well as dozens of other known eaters), but they are somewhat less useful than eaters 1, 2, and 5, so we do not introduce them here.

Free PDF. See conwaylife.com/book for pattern files and print version.

by) a second glider coming in from the same direction as the first. This reaction is called the **boat bit**,[15] and is displayed in Figure 2.17.

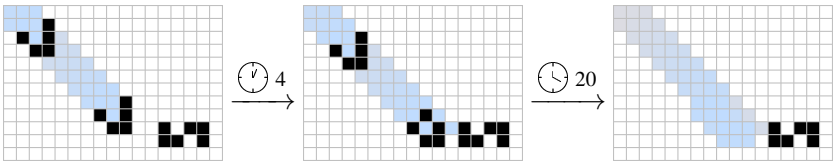


Figure 2.17: A **boat-bit** is a reaction in which a snake (or any other still life containing a pre-block) is used to turn a glider into a boat which then destroys a second glider coming in from the same position. Furthermore, the snake is not even temporarily disturbed at any point throughout this reaction.

While the snake is not technically an eater since it does not destroy each glider that hits it, but rather only destroys pairs of gliders, this is a technicality that is often unimportant. Typically when eating gliders, an entire stream of gliders (perhaps from a glider gun) are fired from the same position, and this boat-bit reaction gives the smallest known way of erasing such a glider stream (in particular, it contains only 6 cells and is slightly more compact than the 7-cell eater 1).

There are also many other reactions that use gliders to toggle a Life object between two or more different states, and they can almost all be used to eat multiple glider streams (though they are often useful for much more than this). For example, a single glider can be used to flip the orientation of a loaf, so a loaf can use this reaction twice to eat two gliders and end up back where it started, as in Figure 2.18. Again, a loaf by itself is not technically an eater since it cannot reconstruct itself after destroying a *single* glider, but we can turn it into an eater by placing a stable object next to it that will flip it back over after a glider hits it. One way of doing this results in the eater called **eater 3**, which is displayed in Figure 2.19.

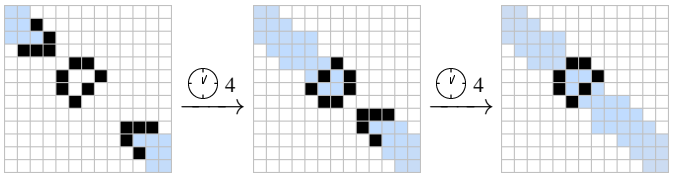


Figure 2.18: A glider can be used to flip a loaf. A loaf can thus be used to eat two gliders coming from opposite directions.

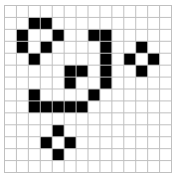


Figure 2.19: Eater 3 is based on the loaf-flipping reaction.

Finally, recall from Section 1.3 that a block is the smallest eater of all, as it can be used to eat a beehive (and for that matter, it can also eat a loaf). It can't be used to eat a single glider or even a single glider stream, but it can be used (like a loaf) to eat two gliders coming from opposite directions. This is possible because of the **(2,1) block pull** reaction displayed in Figure 2.20, in which a glider is destroyed while moving a block by 2 cells horizontally and 1 cell vertically. A second glider coming from the opposite direction then moves the block back to where it started.

While this reaction is perhaps of limited use when it comes to eating gliders (since the opposing gliders have to be positioned exactly right in order to return the block to its starting point), the general idea of using gliders to move blocks around the Life plane is a very useful one that we will explore in depth in Sections 5.7 and 8.6.[16]

[15] Its name comes from the fact that this reaction can be used to store a single bit of memory. We will explore methods like this one for simulating computation in Chapter 9.

[16] We will also see another use of colliding gliders with a block in Exercise 4.11.

Free PDF. See conwaylife.com/book for pattern files and print version.

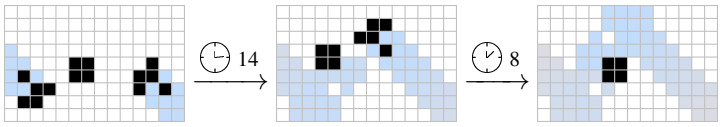
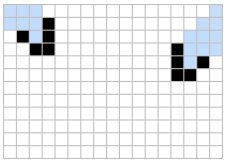


Figure 2.20: A **(2,1) block pull** is a reaction in which a glider pulls a block by 2 cells in one direction and 1 cell in the other. A block can thus be used to eat two gliders coming from opposite directions.

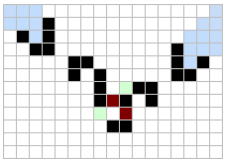
2.4 Welded and Constrained Still Lifes

We saw in Section 2.2 that there are many general methods for constructing a wide variety of still lifes of almost any size. We now focus on using these methods to combine multiple still lifes into a single still life that retains the properties of each of its components (such as the ability to eat gliders that are on specific paths)—a process that is called **welding**.

To give an example of why we might want to weld two still lifes, suppose that we want to erase two gliders that are in the positions shown in Figure 2.21(a). Since those gliders are so close to each other, there is no way to eat each of them with individual eaters such as eater 1—they will interfere with each other and no longer be stable, as shown in Figure 2.21(b).



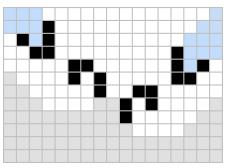
(a) Two gliders that we would like to eat.



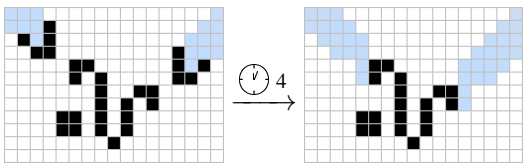
(b) Placing two individual eaters does not work.

Figure 2.21: The two gliders in (a) cannot be eaten by individual eater 1s, since they are too close together and the eater 1s will no longer be stable, as in (b). In particular, the dead cells highlighted in green now have 3 live neighbors and will be born, while the live cells highlighted in red now have 4 live neighbors and will die. Similar problems occur if we try to use other eaters like eater 2 or eater 5 as well.

To get around this problem, we combine two eater 1s into a single, larger eater. The key idea is that the only part of eater 1 that is actually involved in the glider eating reaction from Figure 2.14 is its pre-block (i.e., the 3-cell corner at its top-left)—its tail is just there to stabilize the pre-block. So to weld two eater 1s together, we place their pre-blocks in the appropriate spots and then replace their tails by a single connecting object in such a way that they are *both* stable, as in Figure 2.22. This stabilizing piece is typically constructed using a combination of the grammar that we introduced in Section 2.2, trial-and-error, and computer software.



(a) A placement of pre-blocks (and a bit of the tails) that eats the two gliders.



(b) A way of welding two eater 1s together to eat both of the gliders.

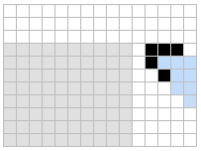
Figure 2.22: In order to weld two eater 1s together to make a single still life capable of eating both gliders, we keep both of their pre-blocks but delete their tails, as in (a). We are then free to choose the light gray cells to be alive or dead, and we want to do so in a way that makes the resulting object a still life. One possibility is shown in (b).

Welding these two eater 1s together might seem somewhat silly in isolation—we could just place two eater 1s so as to eat the gliders before they get so close together in the first place—but there are

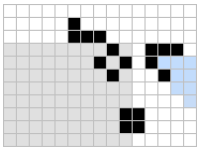
Free PDF. See conwaylife.com/book for pattern files and print version.

two main reasons why welding can be preferable to just placing individual still lifes far apart:

1) Later on in this book, we will be constructing large patterns that are made up of many other smaller patterns. Some of these sub-patterns might get in our way and restrict the amount of space we have to place the individual still lifes.
2) Similarly, because these large patterns that we will construct are already very large, it is often desirable to reduce their size as much as possible, by packing their components as close together as we can without them colliding. Welding still lifes is one of the main techniques for achieving tight packings of components.



(a) We would like to eat this glider, but our eater must be contained in the light gray region.

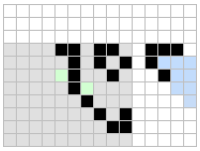


(b) An eater 5 almost works, but sticks just a bit outside of the light gray region.

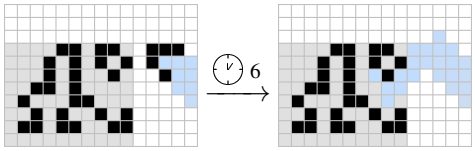
Figure 2.23: We are presented with the problem of eating the glider displayed in (a), but under the restriction that the eater we use must live entirely within the light gray region. The only eater that we have seen so far that even comes close to satisfying this restriction is eater 5, so we use that as our starting point in (b).

To give an example that highlights point (1) above, consider the problem of eating a glider that is positioned as in Figure 2.23(a), but under the restriction that the eater must be contained within the indicated region of the Life plane. Neither eater 1 nor eater 2 come even close to working—if they are positioned in such a way as to eat the glider, they both extend several rows outside of the specified region. However, eater 5 *almost* works—it extends 2 cells too far in one direction and just 1 cell too far in the other (see Figure 2.23(b)).

To make this eater fit within the specified region, we keep all of the live cells within the region, discard those outside of the region, and then try to add more live cells within the region in order to restore stability and its eating ability. We quickly find that we need to add some live cells to the left and to the bottom-left of the tub in order for the eating reaction to still work.



(a) An eater that almost works, but is not stable due to the two dead green cells that come to life in the next generation.



(b) We can add additional live cells to the west and south in order to overpopulate the two problematic cells, resulting in an eater that works.

Figure 2.24: An illustration of one way of transforming eater 5 into an eater that fits within the light gray region. The pattern (a) is not quite stable, since two nearby dead cells come to life in the next generation. The eater depicted in (b) is one solution to this problem, and it has a recovery time of 6 generations (just like eater 5 itself).

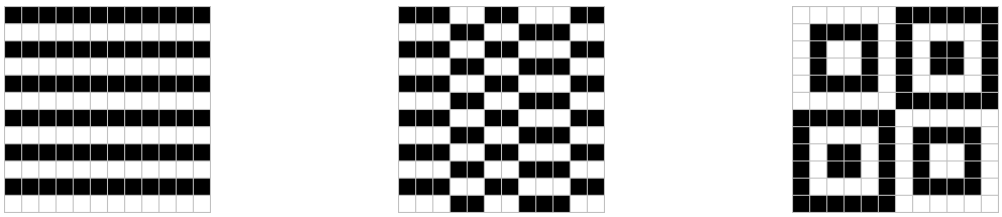
However, placing live cells near a tub is difficult to do while preserving stability, so we change the tub to a boat and arrive at the pattern in Figure 2.24(a). This pattern would work as an eater if it were stable, but unfortunately two nearby dead cells come to life in the next generation. To fix this problem, we just extend the pattern to the west and south so as to overpopulate those two cells, until we eventually arrive at a pattern that is completely stable, as in Figure 2.24(b). This eater is extremely

Free PDF. See conwaylife.com/book for pattern files and print version.

useful precisely because it eats a glider so close to its corner, and we will make heavy use of it in Section 3.6.

2.5 Still Life Density

From the very early days of Life, many examples of infinite still lifes were known with density $1/2$. That is, there are many ways of filling the plane in a stable way such that half of the cells are alive and half of them are dead—some examples are presented in Figure 2.25.



(a) Zebra stripes. (b) Chicken wire. (c) Onion rings.

Figure 2.25: When the plane is tiled with these patterns, they create infinite still lifes with density $1/2$.

It was a long-standing question whether or not an asymptotic density of greater than $1/2$ is attainable, or if $1/2$ really is the upper limit.[17] The following theorem shows that the latter is the case: there are no stable configurations that are more dense than those presented in Figure 2.25 (though there are others that also attain density $1/2$).[18]

Theorem 2.2 — Still Life Density (version 1)
A still life contained in an $n \times n$ bounding box has no more than $\lfloor n^2/2 \rfloor + 2n$ live cells. In particular, the asymptotic density of still lifes as $n \to \infty$ is no greater than $1/2$.

Proof. We prove the theorem by supposing that each cell (either alive or dead) in the Life plane has 2 tokens, and we will present a procedure for redistributing those tokens among each cell's orthogonal neighbors (i.e., each token stays within its original von Neumann neighborhood) in such a way that every live cell ends up having at least 4 tokens. If we can develop such a procedure, we know that in any $n \times n$ square there will be at most

$$\underbrace{2n^2}_{\substack{\text{original tokens in} \\ n \times n \text{ square}}} + \underbrace{8n}_{\substack{\text{tokens from } 4n \\ \text{neighbors of square}}} = 2(n^2 + 4n) \text{ tokens.}$$

On the other hand, if there are L live cells in an $n \times n$ square, then since every live cell has at least 4 tokens we know that

$$\begin{aligned} 4L &\leq \text{number of tokens on live cells in } n \times n \text{ square} \\ &\leq \text{number of tokens on } \textit{all} \text{ cells in } n \times n \text{ square} \\ &\leq 2(n^2 + 4n). \end{aligned}$$

[17] The conjecture that the asymptotic density of still lifes does not exceed $1/2$ was called the **still life conjecture**, which was first considered in *Lifeline* vol. 3 in September 1971. In 1992, an upper bound of $6/11$ on the density of an infinite still life was proved by Dean Hickerson using the method outlined in Exercise 2.19. This bound was then improved to $15/28$ by Hartmut Holzwart, and finally to $1/2$, hence proving the conjecture, by Noam Elkies in 1998 [Elk98].

[18] The original proof presented in [Elk98] is somewhat complicated and requires a decent amount of casework, but also finds the maximum asymptotic density of still lifes in many other Life-like cellular automata. The simpler proof provided here, which is specific to the Game of Life, was first presented in [CSdlB09].

Free PDF. See conwaylife.com/book for pattern files and print version.

Dividing this inequality by 4 gives $L \leq n^2/2 + 2n$, which shows that the asymptotic density of a still life cannot exceed $1/2$.

We thus now turn our attention to developing the token redistribution procedure that results in every live square having at least 4 tokens (thus proving the theorem). The main idea is to have dead cells give away their tokens to neighboring (in the von Neumann neighborhood sense) live cells. The explicit procedure that we use is described in Figure 2.26.



Figure 2.26: All 6 possible orientations of orthogonal neighbors around a dead cell, up to rotation and reflection—the states of the corner cells in light gray are irrelevant. The dark gray arrows indicate where the central dead cell gives its tokens: (a) if it has 1 or 2 live neighbors then it gives one token to each of them, (b) if it has 3 live neighbors then it gives one token to each of the neighbors that are opposite each other, and (c) if it has 0 or 4 live neighbors then it does not give any tokens away.

We now illustrate how many tokens each live cell in any still life ends up with. To start, we note that every live cell in a still life must have exactly 2 or 3 live neighbors, and there are 16 different configurations of 2 or 3 live neighbors around a single live cell, up to rotation and reflection. These 16 configurations are displayed in Figure 2.27. It suffices to observe how many tokens the central live cell ends up with in each of these 16 cases.

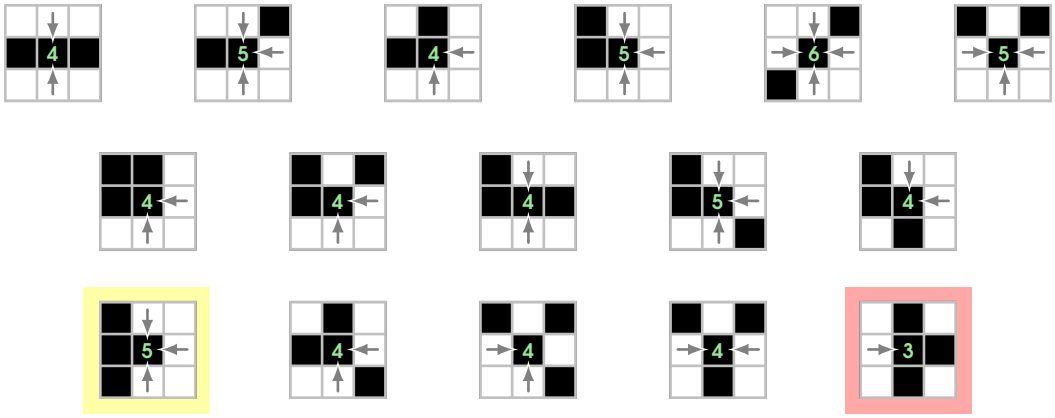


Figure 2.27: All 16 possible orientations of 2 or 3 live neighbors around a live cell, up to rotation and reflection. The dark gray arrows indicate which of its neighboring dead cells give tokens to it. The number of tokens that end up on the central live cell is indicated in green, and is always 2 more than the number of arrows pointing to that cell. The one problematic configuration that results in the central live cell having fewer than 4 tokens is the one at the bottom-right, outlined in red, but it can be fixed with the configuration at the bottom-left, outlined in yellow.

We see that in 15 of the 16 cases, the live cell ends up having at least 4 tokens, as desired. However, the configuration at the bottom-right of Figure 2.27 (outlined in red) results in the live cell only having 3 tokens. To fix this problematic configuration, we claim that it must always be directly to the left of the configuration at the bottom-left of Figure 2.27 (outlined in yellow). To see why this claim is true, notice that the middle-right live cell in the red configuration already has 3 live neighbors, so the 3 cells to its immediate right must be dead so as to avoid killing it by overpopulation—in other words, this cell is the central cell in the yellow configuration. In the other direction, notice that the middle-left live cell in the yellow configuration already has 3 live neighbors, so again the 3 cells to its immediate left must be dead so as to avoid it dying by overpopulation—in other words, this cell is the

Free PDF. See conwaylife.com/book for pattern files and print version.

central cell in the red configuration.

We have thus shown that the red and yellow configurations always occur together. We can thus simply transfer one of the tokens from the cell with 5 tokens to the cell with 3 tokens, resulting in every live cell having at least 4 tokens (see Figure 2.28), as desired. Since we have redistributed the tokens in such a way that every live cell now has at least 4 tokens, and no token has traveled outside of its original von Neumann neighborhood, we are done. ■

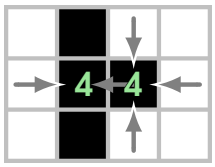


Figure 2.28: To fix the configuration from Figure 2.27 that results in a live cell only having 3 tokens, we transfer an extra token from a bordering live cell that has 5.

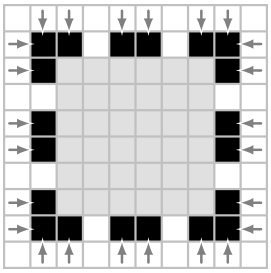


Figure 2.29: This border configuration is the one that causes the largest number of tokens to enter the $n \times n$ square: $4\lceil 2n/3 \rceil$ (the $n = 8$ case is illustrated here, so $4\lceil 16/3 \rceil = 24$ tokens enter the square region).

One interesting feature of the proof of Theorem 2.2 is that it tells us which 3×3 subpatterns of still lifes are the best at being packed densely: the patterns that result in the central cell having exactly 4 tokens. For example, every single live cell in the infinite still lifes in Figure 2.25 ends up with exactly 4 tokens. Similarly, it is straightforward to check that the densest still life in a 3×3 square is the ship (see Table 2.2), and every live cell in the ship also ends up with exactly 4 tokens. By contrast, each live cell in the tub (a less dense 3×3 still life) ends up with 5 tokens.

While it is nice to have an answer to the asymptotic version of the still life density problem, Theorem 2.2 does quite poorly at bounding the maximum number of cells in a given (finite) $n \times n$ square. For example, in the $n = 3$ case, that theorem says that a still life cannot have more than 10 live cells, which is trivially true since a 3×3 square only has 9 squares anyway. To improve the bound provided by the theorem, we note that our upper bound on the number of tokens that end up in the $n \times n$ square ($2n^2 + 8n$) can be improved without too much difficulty. In particular, far fewer than $8n$ tokens can actually enter the square from the $4n$ dead cells that neighbor it—this is because our token redistribution procedure never sends both of the tokens from one dead cell in the same direction, so in fact at most $4n$ tokens (1 token from each neighboring cell) can enter the square.

In fact, even this bound can be improved, because these $4n$ neighboring dead cells only send a token into the square if their neighbor inside the square is alive. However, only 2 out of every 3 cells on the outer edge of the square can be alive, or else they would cause a birth outside of the square and hence not be a still life. It follows that at most $4\lceil 2n/3 \rceil$ tokens can enter the square (see Figure 2.29). If we repeat the calculation that was done at the start of the proof of Theorem 2.2, we immediately arrive at the following result, which is possibly the best upper bound on the density of a finite still life that can be "easily" derived:

Theorem 2.3 — Still Life Density (version 2)
A still life contained in an $n \times n$ bounding box has no more than $\lfloor n^2/2 \rfloor + \lceil 2n/3 \rceil$ live cells.

For large values of n, this new bound is not too much better than the bound provided by Theorem 2.2, since the $n^2/2$ term is much larger than the $2n$ term that we improved anyway. However, for small values of n this bound is now good enough that it is sometimes exactly correct. In particular,

Free PDF. See conwaylife.com/book for pattern files and print version.

when $n = 2, 3$, or 5, this bound equals 4, 6, and 16, respectively, and it is straightforward to construct still lifes that attain these bounds—a block, a ship, and a 2×2 arrangement of 4 blocks. For other small squares, we can find the densest still lifes simply by brute-force search: Table 2.2 gives a summary of the densest patterns in $n \times n$ bounding boxes for $n = 2, 3, 4, \ldots, 10$.

n	Densest Still Life	Thm. 2.3 Bound	Maximal Live Cells	Density
2	 block	4	4	$2/2 = 1.000$
3	 ship	6	6	$6/9 \approx 0.6667$
4	 pond	11	8	$8/16 = 0.5000$
5	 four blocks	16	16	$16/25 = 0.6400$
6	 blocks and ship	22	18	$18/36 = 0.5000$
7	 –	29	28	$28/49 \approx 0.5714$
8	 nine blocks	38	36	$36/64 = 0.5625$
9	 –	46	43	$43/81 \approx 0.5309$
10	 –	57	54	$54/100 = 0.5400$

Table 2.2: The densest still lifes that fit within an $n \times n$ bounding box for $2 \leq n \leq 10$, as well as the upper bound on the population of such a still life guaranteed by Theorem 2.3. The examples displayed here are only unique when $n = 2, 3, 5, 7$, or 8.

Remarkably, we actually know a complete answer to the question of how many live cells a still life in an $n \times n$ bounding box can have. Various clever computer searches were used[19] to compute the answer when $n \leq 60$, the results of which are summarized in Table 2.3. For the $n \geq 61$ cases, the problem stabilizes quite a bit, and there is an explicit formula that is summarized by the following theorem. Proving this theorem is beyond the scope of this book, so the interested reader is directed to [CS12] for details of how it was derived. It is worth noting that the bound we proved in Theorem 2.3 is not too far from optimal: the $n^2/2$ term is right, and the linear term in our bound is $2n/3 \approx 0.6667n$, versus the following exact result which has a linear term of $17n/27 \approx 0.6296n$.

[19] These values for $n \leq 10$ were computed by Robert Bosch in 1999 [Bos99]. This was extended to $n \leq 15$ by Bosch and Michael Trick in 2004 [BT04], and to $n \leq 20$ by Javier Larrosa, Enric Morancho, and David Niso in 2005 [LMN05]. The remaining values were computed by Geoffrey Chu et. al., with the $n \leq 27$ values being computed in 2009 [CSdlB09], and a complete solution for all n being presented in 2012 [CS12].

Free PDF. See conwaylife.com/book for pattern files and print version.

n	$M(n)$	$M(n+10)$	$M(n+20)$	$M(n+30)$	$M(n+40)$	$M(n+50)$
1	0	64	232	497	864	1 331
2	4	76	253	531	907	1 382
3	6	90	276	563	949	1 436
4	8	104	302	598	993	1 490
5	16	119	326	633	1 039	1 545
6	18	136	353	668	1 085	1 602
7	28	152	379	706	1 132	1 658
8	36	171	407	744	1 181	1 717
9	43	190	437	782	1 229	1 776
10	54	210	467	824	1 280	1 835

Table 2.3: A summary of the maximum number of live cells $M(n)$ in a still life with an $n \times n$ bounding box for $1 \leq n \leq 60$.

Theorem 2.4 — Still Life Density (version 3)
For all $n \geq 61$, the maximum number of live cells $M(n)$ in a still life with an $n \times n$ bounding box is given by the formula

$$M(n) = \begin{cases} \lfloor n^2/2 + 17n/27 - 2 \rfloor, & \text{if } n \equiv 0,1,3,8,9,11,16,17,19,25,27, \\ & \quad 31,33,39,41,47, \text{or } 49 \pmod{54}, \text{ and} \\ \lfloor n^2/2 + 17n/27 - 1 \rfloor, & \text{otherwise.} \end{cases}$$

The related problem of finding the maximum density of an oscillator remains open, even in the (presumably) simpler case of infinite oscillators. Although it is possible for oscillators to have individual phases with density higher than $1/2$ (see Figure 2.30 for an example), it seems that their average density over all of their phases is never greater than $1/2$, just like still lifes.

Unfortunately, none of the three known proof techniques for the still life case (i.e., the method introduced by Elkies in [Elk98], the method we used to prove Theorem 2.2, and the computer-assisted method that was used to prove Theorem 2.4 in [CS12]) seem to be strong enough to prove this upper bound of $1/2$ on average oscillator density. However, similar techniques have been used to show that an infinite pattern cannot have average density (over any number of generations) that exceeds $8/13$.[20]

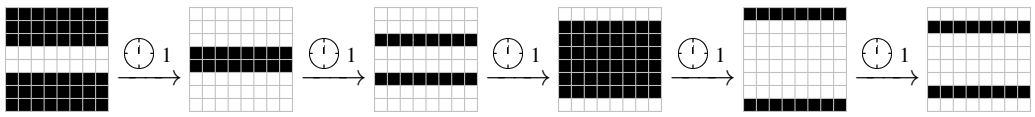


Figure 2.30: A period 6 infinite oscillator that has density $3/4$ in two of its phases. However, its average density over all of its phases is $(3/4+1/4+1/4+3/4+1/4+1/4)/6 = 5/12 \leq 1/2$.

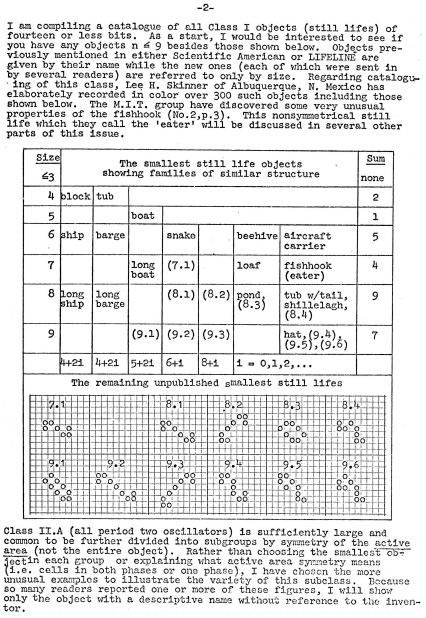
We could also ask for the densest possible individual phase of an oscillator. It is suspected that the highest possible density is $3/4$, which is attained by some of the phases of the oscillator in Figure 2.30, but no proof of this conjecture has been found either.

[20] This was proved by Hartmut Holzwart and Dean Hickerson in September 1992.

Free PDF. See conwaylife.com/book for pattern files and print version.

2.6 Notes and Historical Remarks

Right from the early days of Life, there was considerable interest in cataloging all small still lifes. This process was initiated by John Conway himself, who enumerated the still lifes with 7 or fewer live cells. Robert Wainwright, with the help of the Life community, then constructed all of them with 10 or fewer live cells by hand (see Figure 2.31). This effort was soon extended to 12 cells by Douglas Petrie and Everett Boyer (see Figure 2.32). David Buckingham independently went as high as 13 cells, which we recall there are 240 of, so cataloging them all by hand (and being sure that none were missed!) was no small feat. Peter Raynham then wrote a search program in the mid-1970s that verified the 13-cell still life counts and was also used by Buckingham to find all 14-cell strict still lifes.



-2-

I am compiling a catalogue of all Class I objects (still lifes) of fourteen or less bits. As a start, I would be interested to see if you have any objects $n \neq 9$ besides those shown below. Objects previously mentioned in either Scientific American or LIFELINE are given by their name while the new ones (each of which were sent in by several readers) are referred to only by size. Regarding cataloguing of this class, Lee H. Skinner of Albuquerque, N. Mexico has elaborately recorded in color over 300 such objects including those shown below. The M.I.T. group have discovered some very unusual properties of the fishhook (No.2,p.3). This nonsymmetrical still life which they call the 'eater' will be discussed in several other parts of this issue.

Size / #3	The smallest still life objects showing families of similar structure							Sum / none
4	block	tub						2
5			boat					1
6	ship	barge		snake		beehive	aircraft carrier	5
7			long boat	(7.1)		loaf	fishhook (eater)	4
8	long ship	long barge		(8.1)	(8.2)	pond, (8.3)	tub w/tail, shillelagh, (8.4)	9
9			(9.1)	(9.2)	(9.3)		hat, (9.4), (9.5), (9.6)	7
	4+2i	4+2i	5+2i	6+i	8+i	i = 0,1,2,...		

The remaining unpublished smallest still lifes

Class II.A (all period two oscillators) is sufficiently large and common to be further divided into subgroups by symmetry of the active area (not the entire object). Rather than choosing the smallest object in each group or explaining what active area symmetry means (i.e. cells in both phases or one phase), I have chosen the more unusual examples to illustrate the variety of this subclass. Because so many readers reported one or more of these figures, I will show only the object with a descriptive name without reference to the inventor.

Figure 2.31: A summary of all strict still lifes with 8 or fewer cells, and an incomplete summary of just 6 (out of 10) of the 9-cell still lifes. Originally published in *Lifeline* vol. 3 in September 1971.

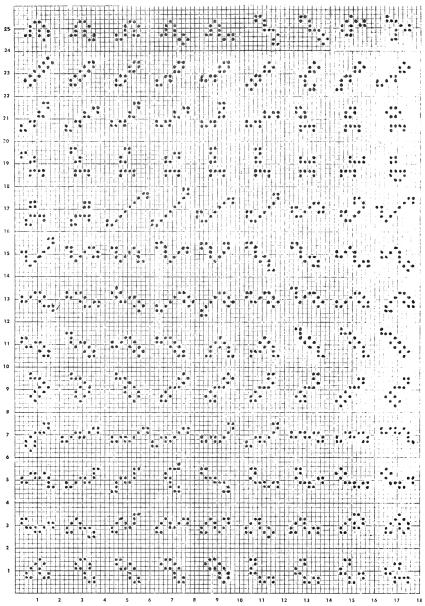


Figure 2.32: A summary of all 121 distinct 12-cell strict still lifes, compiled by hand by Douglas Petrie and Everett Boyer in 1973. Originally published in *Lifeline* vol. 10 in June 1973.

Much of the early difficulty with counting still lifes with more than 14 cells came not just from the fact that the search space was large, but also from the fact that even knowing *how* to search for larger still lifes becomes increasingly complicated. To give an idea of why this is the case, suppose we tried to construct strict still lifes by starting at their top-left corner, placing live cells one at a time, checking whether the resulting pattern is stable after each new cell is added. It might seem reasonable to guess that there is no reason to add additional nearby objects once we find a strict still life, since the resulting pattern would then be a pseudo still life. However, this is not actually the case: there are strict still lifes that will never be found if we stop the search when we first find stability, the smallest of which has 16 cells and is displayed in Figure 2.33.

This problem is not *too* difficult to get around, as there are still only a few different ways that connected components can be near each other, and they can each be coded into the search algorithm individually. For example, objects like the one in Figure 2.33 can be found by allowing the search to add a new domino near an already-stable domino part of the still life. However, each of these tweaks to the algorithm makes it more complicated and thus increases its running time.

Using these ideas, Mark Niemiec conducted a very successful still life search, cataloging all of them with 24 or fewer live cells by 1999. However, even his searches were not perfect; they missed

Free PDF. See conwaylife.com/book for pattern files and print version.

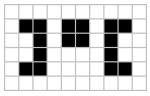


Figure 2.33: A strict still life that would never be found via a greedy still life search, since a block on table would be found before adding the second table.

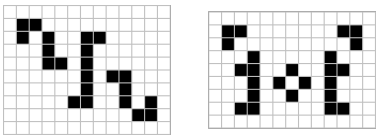


Figure 2.34: Two more strict still lifes that won't be found by a greedy still life search, and had to be added by hand when cataloging all still lifes with 22–24 cells.

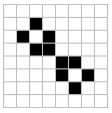
some still lifes that use a single cell to stabilize a long line of orthogonally connected cells, like those in Figure 2.34. These still lifes had to be added back into the count by hand, and they were the main reason why he did not extend his search to 25-cell still lifes—the number of exceptional still lifes that would need to be added back in by hand is too large, and could potentially lead to errors.[21]

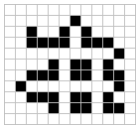
Finally, Simon Ekström wrote a program to search for still lifes in January 2017 that led to the most successful still life search to date.[22] This program has now been used to catalog all still lifes (both strict and pseudo) with 30 or fewer cells, count all still lifes with 34 or fewer cells, and also show that the pseudo still lifes from Figure 2.4 that can be partitioned into 3 or 4 still lifes, but not 2, are the smallest ones possible.

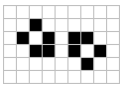
Exercises

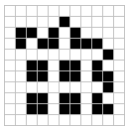
solutions to starred exercises on page 452

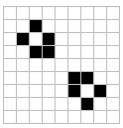
∗**2.1** [1/5] Classify each of the following still lifes as either a strict still life, a pseudo still life, or neither.

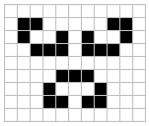
(a) 

(b) 

(c) 

(d) 

(e) 

(f) 

2.2 [2/5] A **quasi still life** is a still life that can be partitioned into two or more disjoint still lifes with overlapping Moore neighborhoods (just like pseudo still lifes), but with all cells that stay dead from underpopulation in the constituent still lifes remaining underpopulated in the overall pattern. For example, the cell highlighted in yellow in Figure 2.3 makes that configuration of two blocks a quasi still life.

(a) Which of the still lifes from Exercise 2.1 is a quasi still life?

(b) Show how to partition the following configuration of 4 blocks into...



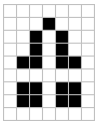
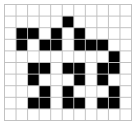
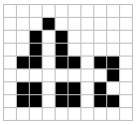
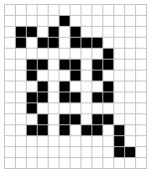
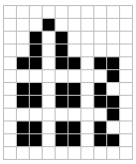
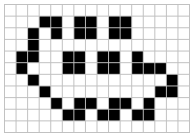
(i) two pseudo still lifes,
(ii) two quasi still lifes, and
(iii) a quasi still life and two strict still lifes.

[21] In fact, even after the manual corrections to his 24-cell still life counts, it was discovered in 2017 that six 24-cell strict still lifes and one 24-cell pseudo still life had been missed.

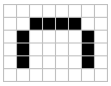
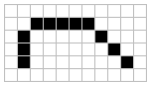
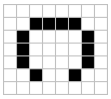
[22] His program is available online at `github.com/simeksgol/GoL_still_life_searcher`

Free PDF. See conwaylife.com/book for pattern files and print version.

∗**2.3** [2/5] Partition each of the following pseudo still lifes into the smallest number (either 2, 3, or 4) of still lifes possible.

(a) 
(b) 
(c) 
(d) 
(e) 
(f) 

∗**2.4** [2/5] For each of the following patterns, find a way of changing some nearby dead cells into alive cells so that the resulting pattern is a still life (similarly to how we turned the path in Figure 2.11 into a still life).

(a) 
(b) 
(c) 

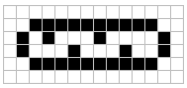
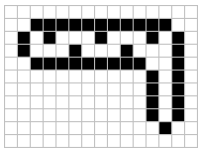
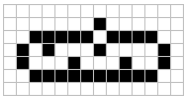
2.5 [3/5] Find all strict still lifes, distinct up to rotation and reflection, with:

- **(a)** 8 live cells, and
- **(b)** 9 live cells.

2.6 [3/5] Find all pseudo still lifes, distinct up to rotation and reflection, with:

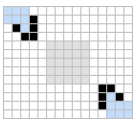
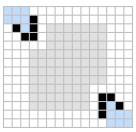
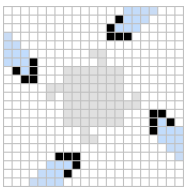
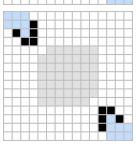
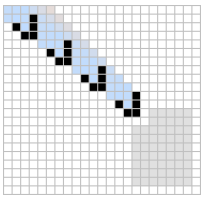
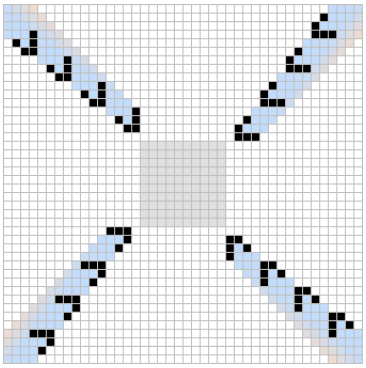
- **(a)** 10 live cells, and
- **(b)** 11 live cells.

∗**2.7** [2/5] For each of the following patterns, find a way of adding induction coils so as to create a still life.

(a) 
(b) 
(c) 

2.8 [1/5] Use the Gosper glider gun and an eater of your choice to create a period 30 oscillator.

∗**2.9** [2/5] For each of the following configurations, weld or modify eater 1s and/or eater 2s so as to create a single eater that can destroy all of the displayed gliders, yet lives entirely within the region specified by the light gray cells.

(a) 
(b) 
(c) 
(d) 
(e) 
(f) 

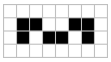
2.10 [2/5] Show how a single eater 2 can be used to eat...

- **(a)** a lightweight spaceship, and
- **(b)** a middleweight spaceship.

2.11 Recall eater 3 from Figure 2.19.

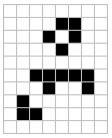
- **(a)** [1/5] Demonstrate how eater 3 can eat a glider.
- **(b)** [2/5] Find at least two still lifes that eater 3 can also eat when they are placed near its loaf.
- **(c)** [2/5] Use two copies of the loaf-flipping reaction in eater 3 to create a period 8 oscillator.

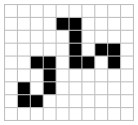
2.12 [4/5] Prove that it is not possible to stabilize the path of live cells displayed below into a still life, no matter what dead cells you change to alive.



Free PDF. See conwaylife.com/book for pattern files and print version.

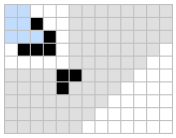
∗**2.13** [2/5] Show how each of the following still lifes can be used to eat a glider in only 4 generations, tying them with eater 1 as the fastest-known glider eaters.

(a) 

(b) 

2.14 [1/5] Show how a single loaf can eat both of the gliders displayed in Exercise 2.9(a).

∗**2.15** [3/5] Complete the incomplete glider eater displayed below in such a way that it lives entirely within the region specified by the light gray cells.



∗**2.16** [3/5] Prove that in a still life, a dead cell can have no more than six live neighbors. Provide an example of a still life that attains this bound.

2.17 [3/5] Find maximum-density still lifes in $n \times n$ bounding boxes for $n = 4, 6$, and 9 that are different from those displayed in Table 2.2.

2.18 [2/5] In the proof of Theorem 2.2, we described a procedure for distributing tokens on the Life grid in such a way that every live cell in a still life ends up with at least 4 tokens. Use this procedure to determine how many tokens end up on each live cell of the following still lifes:

(a) block,
(b) beehive,
(c) pond, and
(d) eater 1.

∗**2.19** [3/5] In the proof of Theorem 2.2, we described a procedure for distributing tokens on the Life grid that allowed us to prove that the asymptotic density of still lifes is no greater than $1/2$. If we instead use the much simpler token distribution scheme of "every dead cell gives each live neighbor (in the Moore neighborhood sense) one token", then we get a weaker upper bound, which you will now derive.

(a) Using this new token distribution scheme, how many tokens should each cell start with to ensure that no cell ends up with a negative number of tokens? [Hint: Use Exercise 2.16.]
(b) Find a lower bound on the number of tokens that each live cell receives.
(c) Mimic the calculation at the start of the proof of Theorem 2.2 to conclude that the asymptotic density of a still life is no greater than $6/11$.

2.20 [4/5] In this question, we consider the problem of finding the maximum density still life in a rectangular $m \times n$ bounding box.

(a) Prove that a still life with an $m \times n$ bounding box cannot have more than $\lfloor mn/2 \rfloor + m + n$ live cells. [Hint: Use the token distribution scheme from the proof of Theorem 2.2.]
(b) Prove that a still life with an $m \times n$ bounding box cannot have more than

$$\left\lfloor mn/2 + \frac{1}{2}\lceil 2m/3 \rceil + \frac{1}{2}\lceil 2n/3 \rceil \right\rfloor$$

live cells. [Hint: Use the argument that was used to prove Theorem 2.3.]
(c) Use the formula from part (b) to show that a still life with a $2 \times n$ bounding box cannot have more than

$$n + \lceil (n+2)/3 \rceil$$

live cells. Show that this bound is tight whenever $n \geq 2$ and $n \not\equiv 0 \pmod 3$. What do you expect the maximum number of live cells is when $n \equiv 0 \pmod 3$? Prove it.
(d) Write a computer program that calculates the maximum number of live cells in a still life with a $3 \times n$ bounding box for $n = 2, 3, \ldots, 10$. Compare your results with the bound from part (b).

2.21 [3/5] Construct an (infinitely large) oscillator of period at least 2 with average density over all of its phases equal to exactly $1/2$.

Free PDF. See conwaylife.com/book for pattern files and print version.

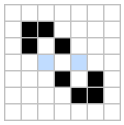


3. Oscillators

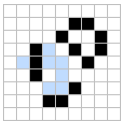
That it will never come again is what makes life so sweet.

Emily Dickinson

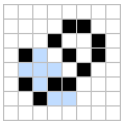
Recall that an oscillator is a pattern that returns to its initial phase after 2 or more generations, and the smallest number of generations required is its **period**.[1] Some oscillators that we have already seen occurred naturally, such as the blinker, pulsar, and pentadecathlon, and we also constructed some rather unnatural oscillators as well, such as the queen bee shuttle and twin bees shuttle. We have thus seen examples of oscillators with quite a few different periods: $2, 3, 15, 30$, and 46, and it seems natural to ask whether or not we can "fill in" these gaps and find oscillators with any period of our choosing. This is the main goal of this chapter: develop techniques for constructing as wide a variety of oscillators as possible, in the hope of finding one of every single period.



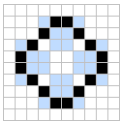
(a) bipole (p2).



(b) jam (p3).



(c) mold (p4).



(d) octagon 2 (p5).

Figure 3.1: Some more small naturally occurring (but rare) oscillators that can be found via computer-assisted soup searches. These oscillators were found by (a) early Lifenthusiasts at M.I.T. in 1970, (b,c) Achim Flammenkamp in 1988,[2] and (d) Sol Goodman and Arthur Taber in 1971.

While most of this chapter will be devoted to techniques for constructing oscillators, to start we note that there are some more oscillators that can be found simply by evolving random soups, as we

[1] Still lifes could be thought of as oscillators with period 1, but in practice (and in this book) the term "oscillator" refers to a pattern with period 2 or greater.

[2] Jam and mold got their names from the fact that they both consist of something on top of a loaf. To remember which is which, notice that the number of letters in their name matches their period.

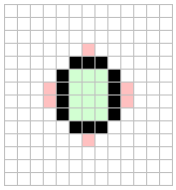
Free PDF. See conwaylife.com/book for pattern files and print versio

did in Section 1.1. However, these oscillators are a fair bit less common than those that we saw earlier, and thus are typically only seen in computer-assisted random searches, rather than ones done by hand. A selection of these naturally occurring (but rare) oscillators is presented in Figure 3.1, and we note that we use the shorthand notation "pn" to stand for "period n". For example, we often abbreviate expressions like "period 4 oscillator" as "p4 oscillator".

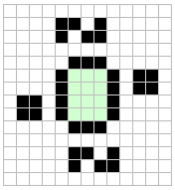
3.1 Billiard Tables

One of the oldest and simplest methods for creating oscillators by hand is to construct a **billiard table**: an oscillator in which all of the oscillating cells are enclosed entirely within some stable pattern.[3] To get an idea for why billiard tables might be a bit easier to create than general oscillators, let's consider the problem of constructing a billiard table where all of the oscillating cells are placed inside a 4×3 box (see Figure 3.2(a)). To make this oscillator work, we have to do two things:

1) Stabilize the outside of the pattern so that it does not self-destruct outwardly.
2) Place some debris inside the 4×3 box in such a way that it bounces around inside the box but never destroys it.



(a) A 4×3 box that we would like to stabilize. The first task is to place stable objects around the outside so as to overpopulate the red cells.



(b) One way of overpopulating the red cells with stable objects. All that remains is to find oscillating debris to place in the inner green section.

Figure 3.2: A 4×3 box that we would like to turn into an oscillator. The red cells in (a) have 3 live neighbors and thus will be alive in the next generation, unless we crowd them with induction coils. One way of crowding them is shown in (b).

Since there are only $2^{4\times3} = 4096$ different configurations of alive and dead cells that can be placed in a 4×3 box, if we can carry out both of the tasks outlined above then we will necessarily have an oscillator with period no larger than 4096 (this period is of course much higher than we expect to actually attain, since it would be a great shock if the inner box actually looped through all possible configurations).

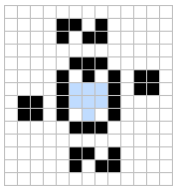
Task (1) above is typically straightforward to take care of: we can just place induction coils around the edges of the pattern to crowd the dead cells that currently have 3 live neighbors (cells in red in Figure 3.2(a)), just like we did to create still lifes in Section 2.2. There are typically numerous ways to do this, and they are usually not difficult to find, even by hand. An example is given in Figure 3.2(b), where the outside of the 4×3 box is stabilized by two blocks and two snakes.

Task (2) above is a bit more difficult to take care of, but since there are only 4096 possible configurations, many of which we clearly do not need to consider (such as the configuration with all cells alive), and the 4×3 box has four-fold symmetry, it actually does not take long to find oscillators by hand (and a computer program could easily be written to check all possibilities). For example, Figure 3.3 gives two oscillators that can be found in this way.

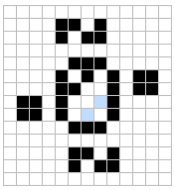
However, not all boxes are as fruitful as the 4×3 one was. For example, if we repeat the above procedure with a 5×3 box, it is still not difficult to stabilize the outside of the box, but there is no

[3] In an oscillator, the cells that oscillate are called its **rotor** and the cells that stay alive for all generations are called its **stator**. A billiard table is thus an oscillator that has its rotor enclosed within its stator.

Free PDF. See conwaylife.com/book for pattern files and print version.



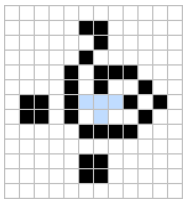
(a) Hertz oscillator (p8)



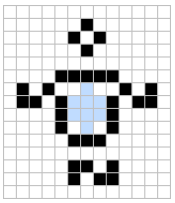
(b) Negentropy (p2)

Figure 3.3: Two billiard table oscillators based on a 4×3 box. Both of these oscillators were found by John Conway's research group by no later than 1971.

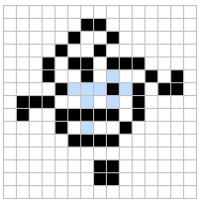
way to fill in its center to turn it into an oscillator (see Exercise 3.2). For this reason, somewhat more exotic regions than rectangular boxes are often used when constructing billiard tables, some examples of which are given in Figure 3.4. Note that these oscillators all use the same stabilization techniques that we learned in Section 2.2: blocks, tubs, and snakes are used as induction coils, and pre-blocks and tails are used to stabilize "corners" of objects (as at the top of the burloaferimeter in Figure 3.4(a), on the left and right of the cauldron in Figure 3.4(b), and on the left and right of the unnamed p10 in Figure 3.4(c)).



(a) Burloaferimeter (p7)



(b) Cauldron (p8)



(c) unnamed p10

Figure 3.4: Some simple billiard table oscillators that were known right from the early days of Life. They were found by (a) David Buckingham in 1972, (b) Don Woods and Robert Wainwright in 1971, and (c) David Buckingham no later than 1976.

3.2 Stabilizing Corners

Another method that is frequently used to construct oscillators by hand is to simply place different combinations of known objects and reactions together, such as how we combined blocks with queen bees to create the queen bee shuttle in Section 1.3. As another example, recall from Section 2.3 that eater 1 is extremely robust—it can withstand many different types of debris hitting its corner. One possibility for creating an oscillator then would be to use several eater 1s to "box in" an area containing some debris, and then that debris would just bounce around in the middle, leaving the eater 1s unharmed. The general idea here is the exact same as it was for billiard tables: create a stable outer area that can contain some debris that is resistant to changes caused by a chaotic central area.

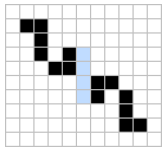


Figure 3.5: Two eaters is a period 3 oscillator found by Bill Gosper in 1971, consisting of two eater 1s eating and rebuilding their corners.

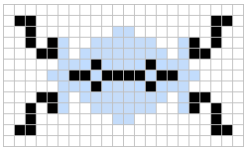


Figure 3.6: Snacker is a period 9 oscillator constructed by Mark Niemiec in 1972 by placing a pentadecathlon in the middle of four eater 1s.

Free PDF. See conwaylife.com/book for pattern files and print version.

The simplest oscillator that uses this technique is **two eaters**: a period 3 oscillator made up of two eater 1s that eat each other's corners and then rebuild themselves (see Figure 3.5). In order to construct less trivial oscillators, we could move the eater 1s away from each other and place some other object between them. In fact, we already used this technique in Exercise 1.10 to stabilize a queen bee with an eater 1. Another important oscillator that can be constructed in this way is the **snacker**: a period 9 oscillator that is constructed by placing a pentadecathlon in the middle of four eater 1s (see Figure 3.6).

Of course, this technique does not work for *all* types of debris and *all* arrangements of eater 1s, but it does not take long to find combinations that do work, even by hand. A wide variety of oscillators whose corners are stabilized by eater 1 are provided in Figure 3.7.

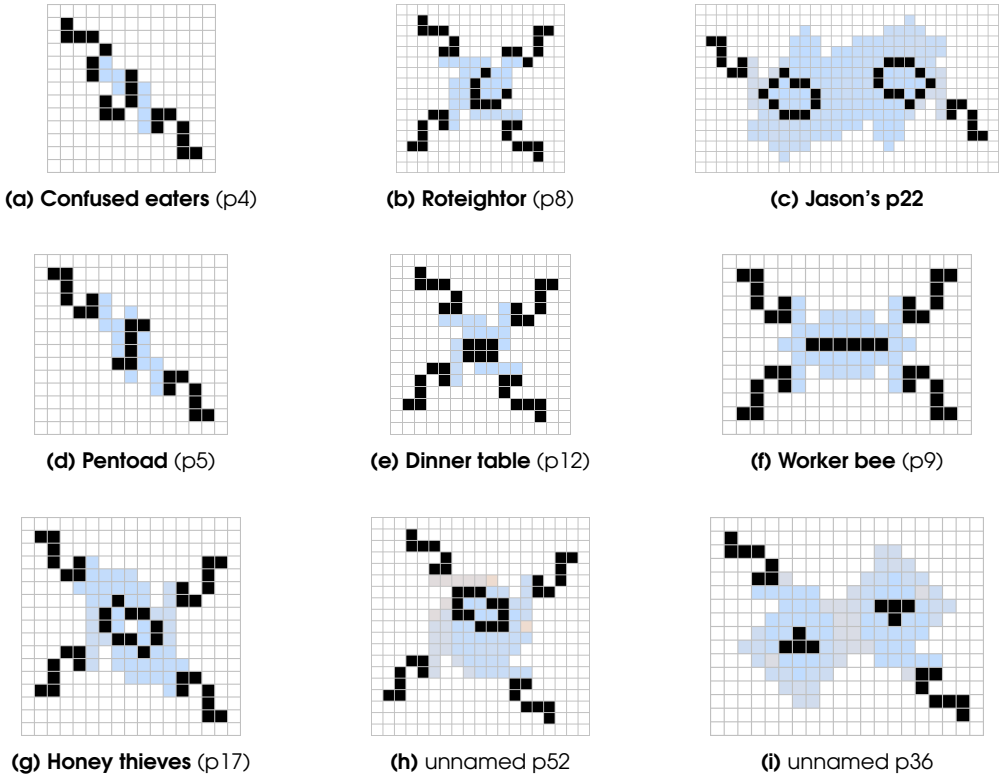
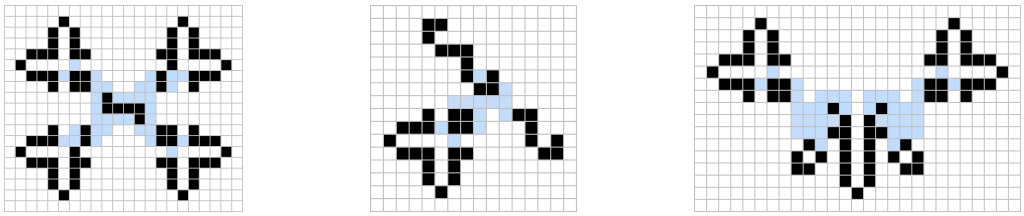


Figure 3.7: Eater 1s can be used to stabilize the corners of several oscillators with a wide variety of periods. These oscillators were found by (a,f) David Buckingham no later than 1972, (b,e) Robert Wainwright in 1972, (c) Jason Summers in 2000, (d) Bill Gosper in 1977, (g) Matthias Merzenich in 2014, (h) David Buckingham in 1977, and (i) Noam Elkies in 1995. The oscillators (e), (f), and (h) were the first known oscillators of their respective periods.

It is also possible to use patterns other than eater 1 to stabilize the corners of oscillators. For example, eater 2 is capable of eating most types of debris that eater 1 can, so we could replace each eater 1 by an eater 2 in many of the oscillators from Figure 3.7 without affecting them in a significant way (see Exercise 3.3). Some examples of oscillators that really *require* an eater 2, since an eater 1 does not suffice, are presented in Figure 3.8.

Furthermore, there is not necessarily any need to use glider eaters to stabilize the corners (for example, we used blocks in the queen bee shuttle in Section 1.3), nor do we have to use the same pattern in each of the corners. We explore these possibilities more in Section 3.4.

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) unnamed p6 **(b)** unnamed p12 **(c)** Buckingham's p13

Figure 3.8: Eater 2s can also be used to stabilize the corners of some oscillators. These oscillators were found by David Buckingham in (a) 1977 and (c) 1976, and by (b) Matthias Merzenich in 2015.

3.3 Composite Periods and Sparks

One very simple way to create oscillators with new periods is to place oscillators with smaller periods next to each other in a non-interacting way. For example, if we place a blinker on the same Life plane as a pulsar, the entire configuration does not return to its initial phase until generation $\text{lcm}(2,3) = 6$, even though the individual components only oscillate at periods 2 and 3. In order to avoid considering "uninteresting" combinations of oscillators like this one, it is typically required that an oscillator must have at least one cell that oscillates at its full period in order to be considered non-trivial.[4]

Somewhat surprisingly, it is in fact sometimes possible to create non-trivial oscillators with composite periods by combining lower-period oscillators. The key to this technique is to combine oscillators that emit **sparks**: configurations of cells that die when left alone.[5] As an example, consider the pentadecathlon, which gives off two sparks in one of its phases (see Figure 3.9). These sparks can be erased or manipulated without affecting the subsequent evolution of the pentadecathlon, which makes them very useful to us. For example, if we were able to find another oscillator (with a different period) that also emits a spark, we could place the sparks next to each other in such a way as to make them interact briefly and then die, resulting in a non-trivial oscillator whose period is the least common multiple of the component oscillators.

As an explicit example of how we can construct a non-trivial oscillator via this technique, we can place a (period 15) pentadecathlon next to a (period 9) snacker. This oscillator trivially has period $\text{lcm}(15,9) = 45$, but if we are careful about how we place its components, we can cause a small reaction to occur near their sparks just once every 45 generations, thus making the oscillator non-trivial (see Figure 3.10).

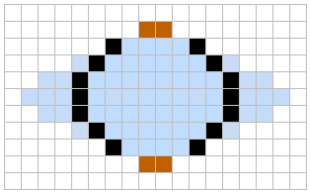


Figure 3.9: One phase of the pentadecathlon has two domino sparks (shown in orange) that immediately die off.

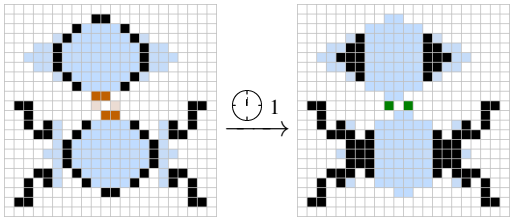


Figure 3.10: A pentadecathlon strategically placed next to a snacker makes a non-trivial period 45 oscillator, since the two cells shown in green on the right are only alive 1 generation out of every 45.

[4] Every oscillator we have seen so far is non-trivial.

[5] While the term "spark" technically refers to any pattern that dies, it is most commonly used to refer to a piece of an oscillator or spaceship that dies and is in a location that is unoccupied during its other phases.

Free PDF. See conwaylife.com/book for pattern files and print version.

Since so many small oscillators have sparks,[6] this is a fairly straightforward and flexible method for constructing oscillators with small composite periods. However, we will see in later chapters that sparks are also useful for much more, so it will be handy for us to look at a few different types of them in a bit more depth. For this reason, we now start introducing a large assortment of oscillators that provide sparks (called **sparkers**).

3.3.1 Types of Sparkers

The spark that the pentadecathlon emits, which consists of two cells orthogonally connected to each other, is called a **domino spark**. While we already know of two oscillators that give off domino sparks (the other one being the snacker), it will be useful to have a selection of them of various periods. We thus present some domino sparkers of periods 3–8 in Figure 3.11. Sparks that are far away from the "body" of an oscillator are typically much easier to work with, so the oscillators 3.11(d) and 3.11(f) are often very useful, despite being much larger than the oscillators 3.11(b) and 3.11(c) with the same periods.

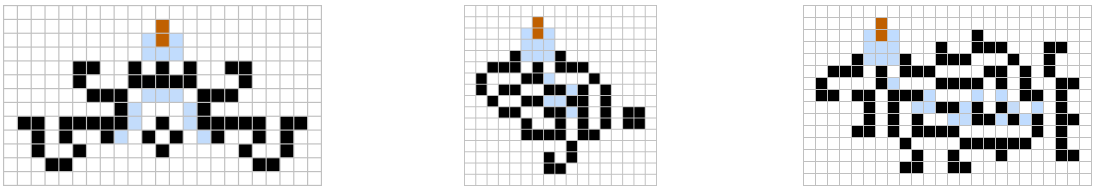


Figure 3.11: A collection of oscillators that emit a domino spark (highlighted in orange). These oscillators were found by (a,e,g) Noam Elkies in 1997, (b) Robert Wainwright in 1980, (c) Dean Hickerson in 1989, (d) Scot Ellison in 2010, (f) Dean Hickerson in 1995 (with improvements to make it smaller by Scot Ellison in 2007), and (h) Simon Norton in 1970. The p9 snacker and p15 pentadecathlon also emit domino sparks.

Notice that in each of the pentadecathlon, the snacker, and all of the domino sparkers provided in Figure 3.11, the domino spark is parallel to the closest edge of the oscillator that produces it. It is more difficult to construct oscillators that produce a domino spark that points perpendicular to the oscillator's nearest edge, but they do exist, and they are called **pipsquirters**. We collect some small pipsquirters in Figure 3.12. Note that the figure eight from Figure 3.11(h) can typically be thought of as both a regular domino sparker and as a pipsquirter, thanks to the multiple orientations of the sparks that it emits.

[6] Billiard tables are a notable exception, which makes them somewhat less useful than other oscillators.

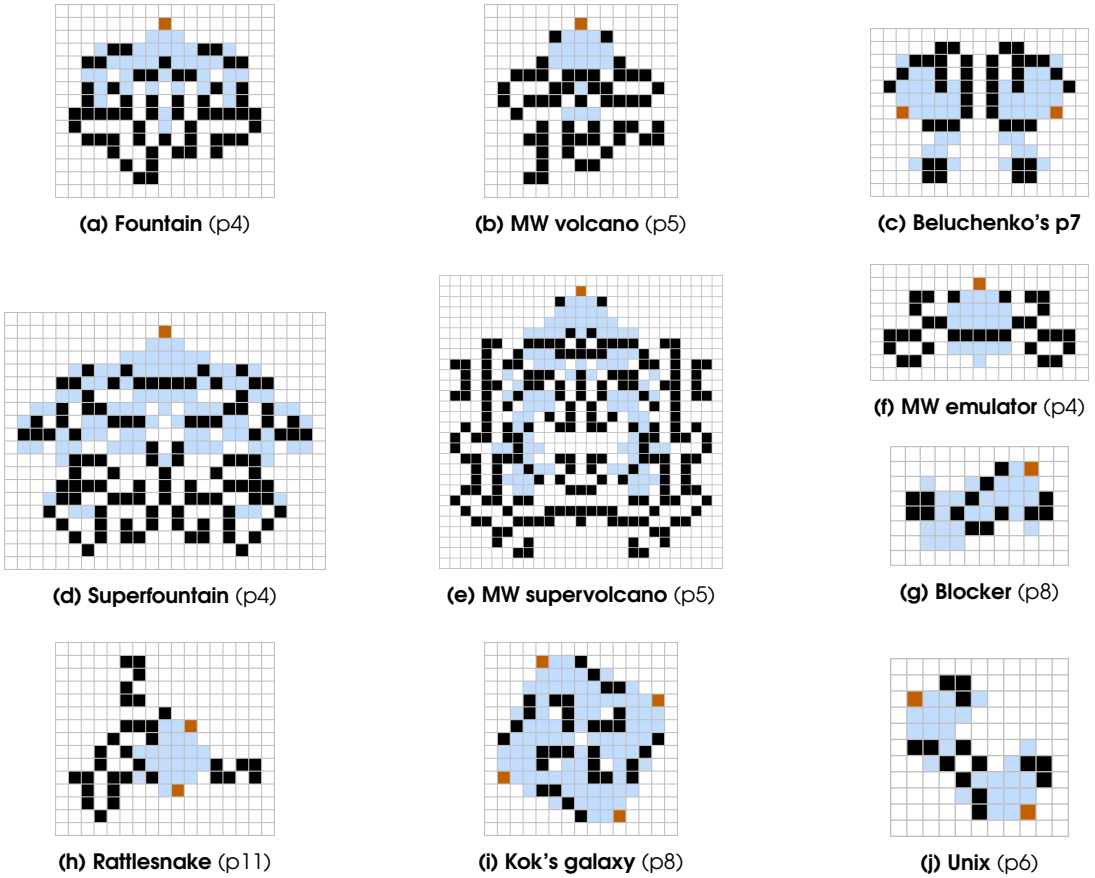
Free PDF. See conwaylife.com/book for pattern files and print version.



(a) p5 pipsquirter **(b)** p6 pipsquirter **(c)** p7 pipsquirter

Figure 3.12: Some pipsquirters: oscillators that produce a domino spark (highlighted in orange) perpendicular to their closest edge. The p5 pipsquirter (a) was found by David Eppstein in April 2003, but is somewhat less useful than the other pipsquirters since its spark is closer to the rest of the oscillator than in the others. The other two were found by Noam Elkies (b) in 1997, and (c) in 1999. The p8 figure eight from Figure 3.11(h) can often be used as a pipsquirter.

The other most commonly occurring spark is the one that consists of just a single isolated cell, called a **dot spark**. The period 4 mold from Figure 3.1(c) is one example of an oscillator that gives off a dot spark, and the middleweight spaceship from Figure 1.10 is an example of such a spaceship. Some other examples of oscillators that emit this spark are provided in Figure 3.13.



(a) Fountain (p4) **(b) MW volcano** (p5) **(c) Beluchenko's p7**

(d) Superfountain (p4) **(e) MW supervolcano** (p5) **(f) MW emulator** (p4)

(g) Blocker (p8)

(h) Rattlesnake (p11) **(i) Kok's galaxy** (p8) **(j) Unix** (p6)

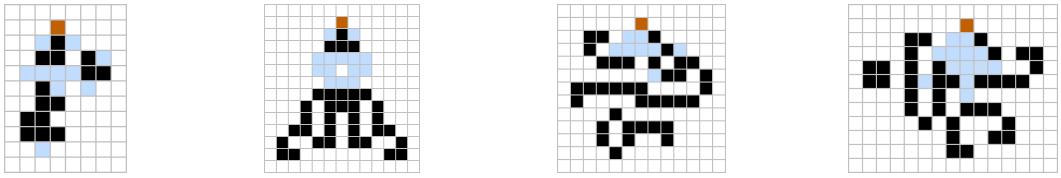
Figure 3.13: A collection of oscillators that emit a dot spark (highlighted in orange). These oscillators were found by Dean Hickerson in (a) 1994, (b) 1992, and (h) 2016, Nicolay Beluchenko in (c) 2009 and (d) 2006, (e) Dongook Lee in 2019, Robert Wainwright in (f) 1980 and (g) no later than 1983 (likely in the early 1970s), (i) Jan Kok in 1971, and (j) David Buckingham in 1976.

Names like "middleweight emulator" and "heavyweight volcano" that are used for some of these oscillators refer to the fact that they give off an arrangement of sparks very similar to that of the

Free PDF. See conwaylife.com/book for pattern files and print version.

middleweight and heavyweight spaceships from Figure 1.10. "Emulators" give their sparks off in the row adjacent to the body of the oscillator itself, while "volcanoes" emit them with a single row of space in between, and "supervolcanoes" emit them with two rows in between. Some supervolcanoes beyond the one from Figure 3.13(e) are presented in Exercise 3.8.

While two dot sparkers cannot be combined with each other to create a non-trivial oscillator (since two dot sparks cannot give the three live neighbors required to give birth to a new cell), they can instead be combined with other types of sparkers. Just like with domino sparkers, we typically prefer oscillators that emit their sparks far away from the rest of the oscillator, so supervolcanoes and the superfountain from Figure 3.13(d) are rather remarkable, despite their large size.

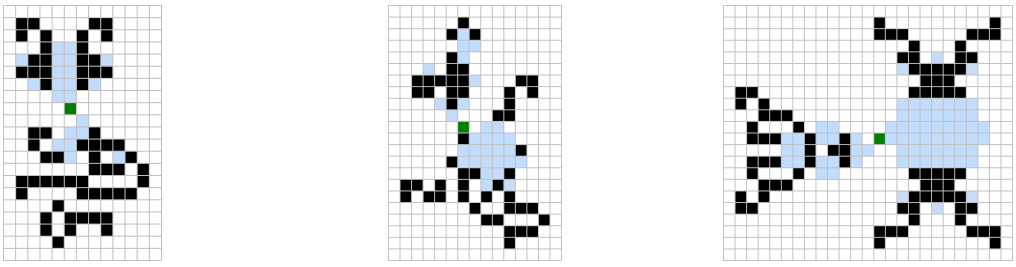


(a) Caterer (p3) **(b) T-nosed p4** **(c)** p4 thumb sparker **(d)** p9 thumb sparker

Figure 3.14: Some (a,b) finger sparkers and (c,d) thumb sparkers, with their sparks highlighted in orange. These sparkers were found by Dean Hickerson in (a) 1989 and (d) 1998, (b) Robert Wainwright in 1989, and (c) David Eppstein in 2000. Also, octagon 2 from Figure 3.1(d) is a period 5 finger sparker.

In the sparkers that we have seen so far, the spark was always separated from the rest of the oscillator by at least one dead cell. Perhaps surprisingly, sparks can also sometimes be useful even when they are directly connected to the rest of the oscillator, as long as the spark itself is in a location that is unoccupied during the oscillator's other phases. If the spark is connected orthogonally to another live cell then it is called a **finger spark**, and if it is connected diagonally to another live cell then it is called a **thumb spark**.[7] Some examples of finger and thumb sparkers are presented in Figure 3.14.

It is perhaps less obvious that these finger and thumb sparks can be used to create composite period oscillators, so we present some explicit examples in Figure 3.15. Since finger and thumb sparks are less accessible than the other sparks that we have seen (i.e., they are closer to the "body" of the oscillator), it is often difficult to combine them with each other, so we instead typically combine them with dot or domino sparks.



(a) p4 thumb on fumarole (p20) **(b)** Caterer on rattlesnake (p33) **(c)** T-nosed p4 on snacker (p36)

Figure 3.15: Some oscillators that work by combining finger and thumb sparks with other sparks. Finger sparks can be combined with either dot spark predecessors (as in (b)) or standard domino sparks (as in (c)), but thumb sparks are best paired with domino sparks (as in (a)). In all cases, the cell that oscillates at the full period (thus making the oscillator non-trivial) is highlighted in green.

[7] Some sources instead define a finger to be a connected spark that dies after 2 generations (instead of just 1). Neither the caterer nor the T-nosed p4 produce finger sparks under this alternate definition.

Free PDF. See conwaylife.com/book for pattern files and print version.

It is also worth pointing out that the caterer is particularly useful as a result of having such a small size and period. Not only is it the smallest known period 3 oscillator, but its spark is used in the construction of what are currently the smallest known oscillators of period 21, 24, 33 (Figure 3.15(b)), 39, 66, and 93.

Finally, there are two more types of sparkers that are somewhat less common, but possibly even more useful than the other sparkers that we have seen so far. Some sparkers create a **duoplet** spark (i.e., a spark consisting of two diagonally connected alive cells[8]), such as the twin bees shuttle that we constructed back in Section 1.4. Another type of spark, which it seems appropriate to call a **banana spark** based on its shape, is emitted by the variant of the queen bee shuttle (called a **buckaroo**) displayed in Figure 3.16(b). Recall that we demonstrated how to construct this oscillator in Exercise 1.10.



(a) Twin bees shuttle making some duoplet sparks **(b)** Buckaroo making a banana spark

Figure 3.16: Some oscillators that emit slightly less common sparks (highlighted in orange).

While these sparkers can be used to create higher-period oscillators just like the other sparkers, they are actually particularly useful for another reason: they can reflect a glider, changing its direction by 90 degrees, as illustrated in Figure 3.17. We will make extensive use of glider reflectors later in this chapter when we discuss glider loops, and also throughout most of the rest of this book.



Figure 3.17: Duoplet and banana sparks can be used to reflect gliders by 90 degrees.

3.4 Hasslers and Shuttles

In many of the oscillators that we saw in the previous section, the debris that was bounced around between the eaters was recognizable. For example, the dinner table in Figure 3.7(e) bounces around a pre-beehive, the honey thieves in Figure 3.7(g) bounce around a pre-honey farm (hence its name), the oscillator in Figure 3.7(i) stabilizes two T-tetrominoes, and the eaters in Figure 3.7(h) stabilize lumps of muck.

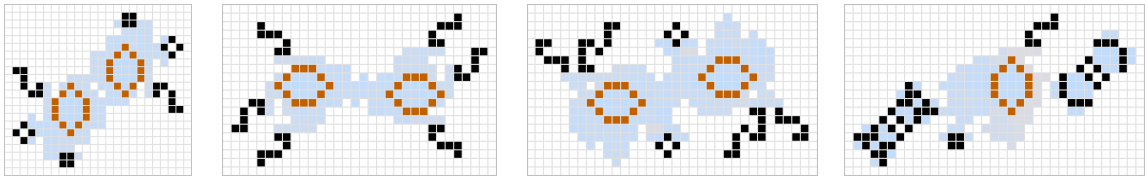
This observation suggests that it might be fruitful to look for oscillators in which the debris in the middle is one of the "standard" unstable evolutionary sequences. With this in mind, we say that one pattern **hassles** another one if it repeatedly moves or changes it, typically in a periodic way so as to create an oscillator or gun. Oscillators created in this way are called **hasslers**. In the special case when a hassler moves an object back and forth between two positions (such as the queen bee shuttle from Section 1.3), it is called a **shuttle**.

[8]The term "duoplet" technically refers to *any* connected two-cell object, but since the only other connected two-cell object is the domino spark, there is not much chance for ambiguity in using the term to refer to two diagonally connected cells.

Free PDF. See conwaylife.com/book for pattern files and print version.

3.4.1 Types of Hasslers

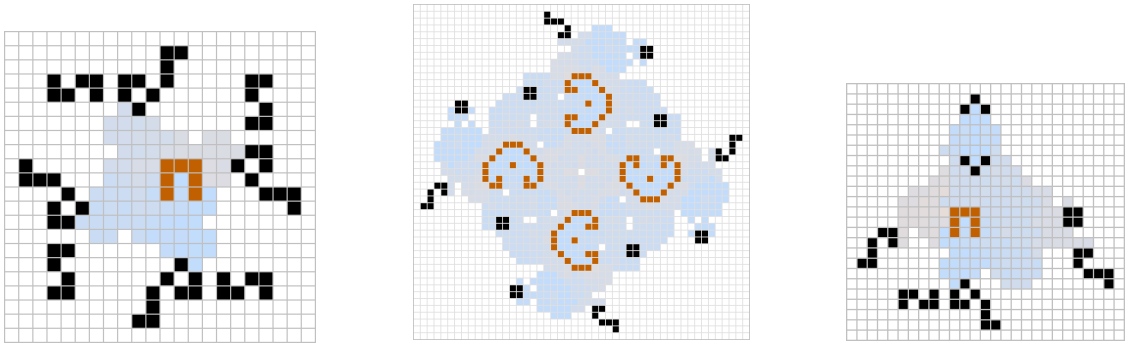
One of the objects that has proved most effective at being hassled in order to create oscillators is the pre-honey farm (which we first investigated back in Section 1.2). The honey thieves oscillator in Figure 3.7(g) shows how it can be hassled to create a period 17 oscillator, and several more honey farm hasslers are presented in Figure 3.18. While some of these oscillators were found by hand, such as the one in Figure 3.18(b), they are more commonly found via computer searches that try numerous placements of honey farms between various arrangements of small still lifes and sparkers.



(a) period 16 **(b)** period 18 **(c)** period 21 **(d)** period 32

Figure 3.18: Several oscillators that work by hassling pre-honey farms (highlighted in orange). These oscillators were found by (a,c) Dongook Lee in 2016, (b) Nico Brown in January 2015, and (d) David Raucci and "praosylen" in September 2021.

Another common type of hassler is based on the pi-heptomino that we introduced in Section 1.2. There are some known formations of still lifes and oscillators that can rotate pi-heptominoes by 90 degrees when placed nearby, so using multiple copies of these reactions results in an oscillator. There are also some configurations of stable patterns that can be used to tame the debris produced by a pi-heptomino, causing it to re-form in another location. Some examples of oscillators that work via these mechanisms are presented in Figure 3.19. Note that the one displayed in Figure 3.19(c) is particularly useful due to how sparky it is and the fact that it has the same period (46) as the twin bees shuttle. We devote the entirety of Section 6.3.2 to investigating things that can be done with it.



(a) gourmet (p32) **(b)** p37 pi-heptomino hassler **(c) Tanner's p46**

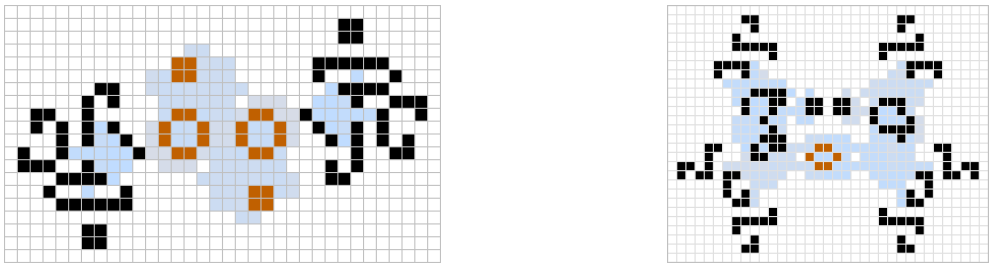
Figure 3.19: Some oscillators that work by hassling pi-heptominoes (highlighted in orange). These oscillators were found by (a) David Buckingham in 1978, (b) Nicolay Beluchenko in 2010, and (c) Tanner Jacobi in 2017. While the oscillator in (b) might not look like it is hassling pi-heptominoes, the chaotic objects the middle are in fact each the 9th generation of the pi-heptomino.

In the examples that we have seen so far, the object being hassled (a pre-honey farm or a pi-heptomino) was unstable, but it does not need to be—it is entirely possible to construct new oscillators by hassling still lifes. In particular, sparks from sparkers can sometimes be used to cause a collection of still lifes to temporarily explode, but then settle back into their original formation. One example of such a hassler is given in Figure 3.20(a), where a dot spark is used to hassle an arrangement of two ponds and two blocks, letting us create a fairly wide variety of oscillator periods. In particular, it takes

Free PDF. See conwaylife.com/book for pattern files and print version.

20 generations for these blocks and ponds to return to their original formations, so this reaction can be used to double the period of a sparker with period 10–19 or triple the period of a sparker with period 7–9.

Another example of a still life hassler, called **David Hilbert**,[9] is presented in Figure 3.20(b) (though it is perhaps more of a shuttle than just a hassler). In this oscillator, two B-heptominoes are hassled so that they then shuttle a beehive back and forth between two positions.



(a) p27 still life hassler **(b)** David Hilbert (p23)

Figure 3.20: Some oscillators that work by hassling still lifes (highlighted in orange). These oscillators were found by (a) Jason Summers in 2005 and (b) Luka Okanishi in 2019.

3.4.2 Types of Shuttles

We have already seen that queen bees and twin bees can be used to construct period 30 and period 46 shuttles, but these objects do not really highlight the variety of shuttle oscillators that exist. To give a slightly more exotic example, recall the T-tetromino that we saw back in Figure 1.11. Unlike the pre-honey farms and pi-heptominoes that we just learned can be hassled by still lifes, T-tetrominoes are typically hassled by oscillators (sparkers in particular).[10] This is because many common sparks can be used to reposition a T-tetromino, such as the one demonstrated in Figure 3.21.

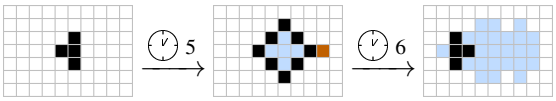


Figure 3.21: A dot spark (highlighted in orange) can be used to hassle a T-tetromino, moving it to the left by 2 cells and mirroring it over the course of 11 generations—resulting in the eventual pre-traffic-light descendant being pushed 3 cells to the left.

It would be theoretically possible to use this reaction together with two period 11 dot sparkers to create a period 22 T-tetromino shuttle oscillator, but unfortunately there is no known period 11 dot sparker whose spark is emitted far enough away from the oscillator to work.[11] However, one nice feature of shuttles is that we can mix-and-match different shuttling reactions, as long as the distances that they offset the central object (the T-tetromino in this case) are the same. For example, we can couple the previous hassling reaction with the one displayed in Figure 3.22, which moves a T-tetromino by the same amount, but only takes 9 generations (rather than 11) to do it.

By combining these two reactions to move a T-tetromino back and forth, it now takes a total of $11+9=20$ generations to return to its original position. Fortunately, we have seen some p4 and p5

[9] This was the first p23 oscillator found. Its name is a reference to the famous mathematician of the same name who published a list of 23 important unsolved (at that time) mathematical problems in 1900.

[10] The most well-known exception being the oscillator in Figure 3.7(i), which uses two eater 1s to hassle two T-tetrominoes.

[11] The p11 rattlesnake gives a dot spark, but there is no way to place the dot spark in such a way that the T-tetromino avoids colliding with the rattlesnake.

Free PDF. See conwaylife.com/book for pattern files and print version.

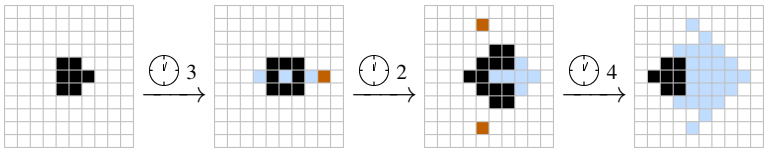
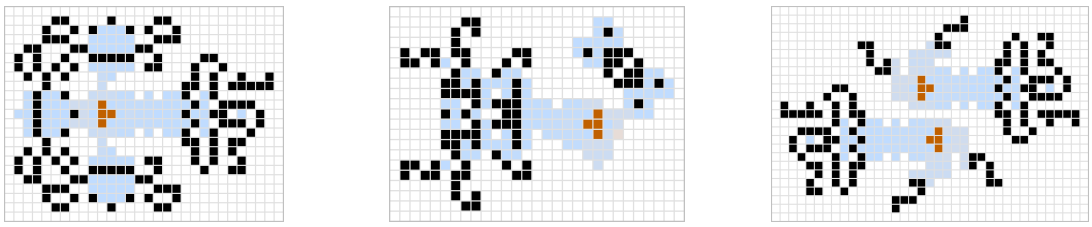


Figure 3.22: Three dot sparks (highlighted in orange) can be used to hassle generation 1 of a T-tetromino (i.e., the 7-cell object on the far left), moving it to the left by 2 cells and mirroring it over the course of 9 generations.

sparkers that can create the dot sparks needed to perform the hassling. A period 20 T-tetromino shuttle that uses these two shuttle reactions (assisted by a period 5 middleweight volcano and three period 4 middleweight emulators that have been welded together) is displayed in Figure 3.23(a).



(a) p20 T-tetromino shuttle **(b)** p24 T-tetromino hassler **(c)** p25 T-tetromino shuttle

Figure 3.23: Some oscillators that work by hassling and shuttling T-tetrominoes (highlighted in orange). These oscillators were found by (a) Noam Elkies in 1995 and (c) in 1994, and by (b) Bill Gosper in 1994.

There are also many other T-tetromino shuttling reactions that can be used to construct oscillators with periods that we have not yet seen, as in Figure 3.23. Notice that even though these oscillators themselves are new to us, the components used to construct them are not—they are all still lifes and low-period oscillators that we saw earlier. In particular, the period 24 oscillator in Figure 3.23(b) uses a figure eight and a fountain[12] to hassle a T-tetromino, and the period 25 oscillator in Figure 3.23(c) uses two middleweight volcanoes and four copies of eater 1 to hassle two T-tetrominoes.

One way to help us create even more shuttles is to place two T-tetrominoes next to each other,[13] creating a pattern called a **pre-pulsar**. Not surprisingly, this object is named in this way because, if left alone, it evolves into a pulsar, as in Figure 3.24 (in fact, this small arrangement of two T-tetrominoes is exactly why pulsars occur so frequently in random soups).

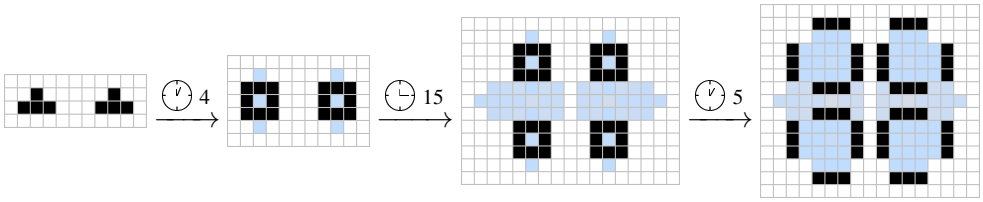


Figure 3.24: A **pre-pulsar** is an arrangement of two T-tetrominoes that duplicates itself and then evolves into a pulsar.

Our primary interest in the pre-pulsar comes from the fact that it is good at creating copies of itself, as illustrated by the fact that it duplicates itself over the course of 15 generations. This makes it a prime candidate for shuttling around to different locations, and indeed pre-pulsars can be hassled in

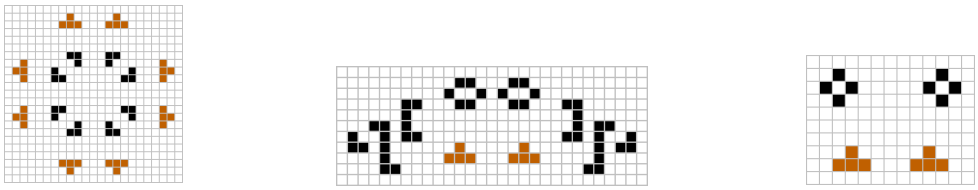
[12] The original larger form of the fountain from Figure 3.13 is used here, since the smaller form interferes with the reaction.

[13] Much like how placing a B-heptomino next to itself helped us create the twin bees shuttle in Section 1.4.

Free PDF. See conwaylife.com/book for pattern files and print version.

a remarkable number of different ways and with a variety of different timings—see Figure 3.25 for some examples.

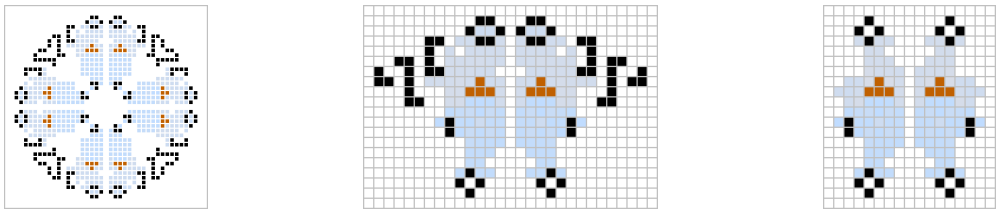
By combining these different shuttling reactions, we can create pre-pulsar shuttles with a wide variety of different periods. For example, the reactions in Figure 3.25 can be used to create oscillators with periods $12+14=26$, $14+15=29$, and $15+15=30$, which are demonstrated in Figure 3.26 (though the period 26 shuttle requires us to do some slight welding in order to pack the four 14-generation pre-pulsar pushers together tightly enough).



(a) 12 generations **(b)** 14 generations **(c)** 15 generations

Figure 3.25: Some pre-pulsar pushers that move a pre-pulsar by 3 cells in either (a) 12, (b) 14, or (c) 15 generations. In all cases, the pre-pulsar is highlighted in orange and moves away from the object that does the hassling.

Note that some combinations of the pre-pulsar pushers cannot work together to create shuttles. For example, we cannot use multiple copies of the 12-generation reaction in Figure 3.25(a) to create a shuttle with period $12+12=24$, since there is no known way of tying off the ends of such a shuttle—it would have to be infinitely large. Similarly, we cannot use two copies of the 14-generation reaction in Figure 3.25(b) to create a shuttle with period $14+14=28$ (though other combinations of hasslers can reach this period) since the two hasslers end up being too close and interfere with each other. Finally, we cannot use the 12-generation and 15-generation pre-pulsar pushers together to create a shuttle with period $12+15=27$, since the 12-generation reaction depends on the beacon, which has period 2 (which does not divide 27, so it becomes out of sync with the pre-pulsars).



(a) p26 pre-pulsar shuttle **(b)** p29 pre-pulsar shuttle **(c) eureka** (p30)

Figure 3.26: Some pre-pulsar shuttle oscillators constructed by pasting together the reactions from Figure 3.25. The period 30 shuttle (c) was found by David Buckingham in 1980.

3.5 Glider Loops and Reflectors

In the previous sections, we looked at methods of constructing oscillators that were largely ad hoc: they worked sometimes, but it was almost impossible to actually predict when they would work or what period the resulting oscillators would have. We now start looking at methods of constructing oscillators that are a bit more precise and calculated.

To begin, we simply notice that if we were to allow for oscillators that are infinitely large, we would easily be able to construct oscillators with any period of our choosing: we could just create an infinitely long line of gliders, and the period of the oscillator would be determined by the spacing

Free PDF. See conwaylife.com/book for pattern files and print version.

between the gliders. The closest together that we can place gliders without them crashing into each other is 14 generations apart, so this immediately would give oscillators of every period 14 or larger (and we have already seen examples of oscillators with every smaller period).[14]

If we make the usual restriction that the patterns we consider must have finite size, then this idea does not quite work, but the spirit of it does: what if we could make an oscillator by moving gliders between different positions so as to replace each other? To facilitate this, we would need to somehow manipulate the gliders so that they do not always travel in the same direction. In particular, we would like to find a stationary pattern (i.e., a still life or an oscillator) with the property that if a glider hits it then the stationary pattern is unaffected and another glider is output in a different direction. Such a pattern is called a **reflector**.

We already saw how to use duoplet and banana sparks to reflect gliders like this in Figure 3.17, so we can reflect a glider stream using any oscillator that emits one of these sparks with enough clearance, as long as the period of the glider stream is a multiple of the period of the oscillator. By using four carefully positioned reflectors, we can then create a track for a glider to follow, as we illustrate with buckaroos in Figure 3.27. That particular track takes a single glider 180 generations to traverse, and has two gliders on the track, resulting in an oscillator with period 90. By changing the number of gliders in the loop and moving the buckaroos farther away from each other, we can create an oscillator with any period that is a multiple of 30.

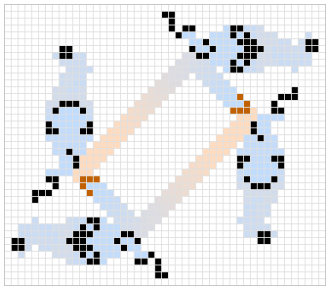


Figure 3.27: A period 90 oscillator consisting of two gliders bouncing around a track made up of four buckaroos.

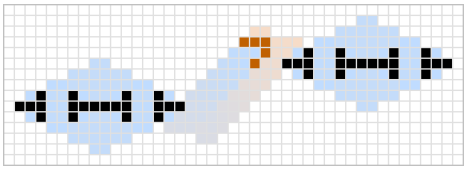


Figure 3.28: A period 60 oscillator consisting of a glider bouncing back and forth between two pentadecathlons.

There are also patterns that directly reflect a glider by 180 degrees, rather than 90 degrees. Possibly the simplest such pattern is the pentadecathlon, as demonstrated in Figure 3.28. Because the glider has period 4 and the pentadecathlon has period 15, and this reflection only happens if the phases of both objects match up just right, the smallest-period oscillator that uses a pentadecathlon to reflect a glider in this way has period 60. By spacing the pentadecathlons farther apart, we can similarly add 60 generations of travel to the glider in each direction, resulting in oscillators with period $60 + 120n$ for any integer $n \geq 0$.

Although 180-degree reflectors like this one result in simpler glider tracks (since you only need 2 reflectors instead of 4 to create a complete track), they are typically less useful than their 90-degree counterparts, since we can use two 90-degree reflectors to create a single 180-degree reflector, but we cannot use 180-degree reflectors to make a 90-degree reflector. Furthermore, we can only place a single glider on the track in Figure 3.28, regardless of how far apart we space the pentadecathlons, since the input and output paths of the reflection overlap each other.

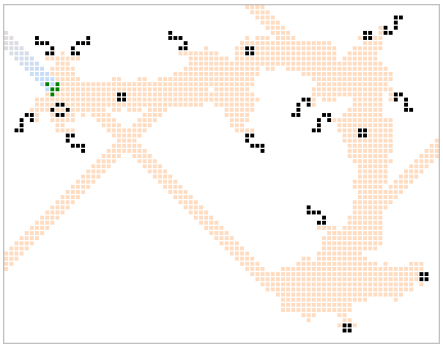
At this point, we have shown that there are oscillators with arbitrarily large periods, but we are still missing a lot of them—we only know how to construct an oscillator with every 30th period. The most obvious way to improve this result would be to find a glider reflector that is a still life rather than

[14] A detailed analysis of how close gliders can be without crashing into each other is presented in Appendix B.1.1.

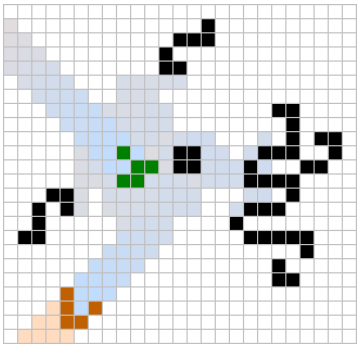
Free PDF. See conwaylife.com/book for pattern files and print version.

an oscillator, since then we would not need to worry about the oscillators being in the correct phase at the start of the collision. Glider reflectors consisting entirely of still lifes are called **stable reflectors**, and they would immediately allow us to construct an oscillator with every 8th (sufficiently large) period, with a single glider bouncing around a closed loop. Furthermore, if the input and output paths of the glider do not interfere with each other, we could construct oscillators with *every* sufficiently large period, simply by placing multiple gliders in the loop at whatever spacing we like.

The usefulness of a stable reflector for creating oscillators (and for other tasks that we will discuss later) depends heavily on how many generations are needed between subsequent gliders colliding with the reflector, which is called its **repeat time**. Certainly every reflector has a repeat time of at least 14 generations—recall that gliders that are fewer than 14 generations apart will collide with each other—but typically much more time than that is needed, since the glider hitting the reflector causes some chaos to happen in and around the reflector before the new glider is sent out in another direction. For example, one of the earliest known stable reflectors, called the **Silver reflector**,[15] worked by colliding a glider with a beehive and then funneling the resulting debris through a track of 17 other cleverly placed still lifes. It has a repeat time of 497 generations and is displayed in Figure 3.29(a).



(a) The **Silver reflector** with repeat time 497.



(b) The **Snark** with repeat time 43.

Figure 3.29: Some stable reflectors that reflect an input glider (highlighted in green) to one or more output positions (highlighted in orange). The (a) Silver reflector actually creates 3 output gliders (one to each of the northwest, northeast, and southwest), but they are far away and not displayed here.

The techniques used to construct the Silver reflector will be investigated in Section 3.6 (and then in much more depth in Chapter 7), but for now we just note that it is perhaps a bit slow for our purposes. We want to create oscillators for as many periods as possible, but this reflector only helps with oscillators of period 497 or greater. Fortunately, faster stable reflectors are now known, with the smallest and fastest one being the **Snark**,[16,17] with a repeat time of 43 generations (see Figure 3.29(b)).

We will see numerous uses for the Snark (and even a few uses for the Silver reflector) throughout the later chapters of this book, but for now we simply note that four Snarks can be used to construct

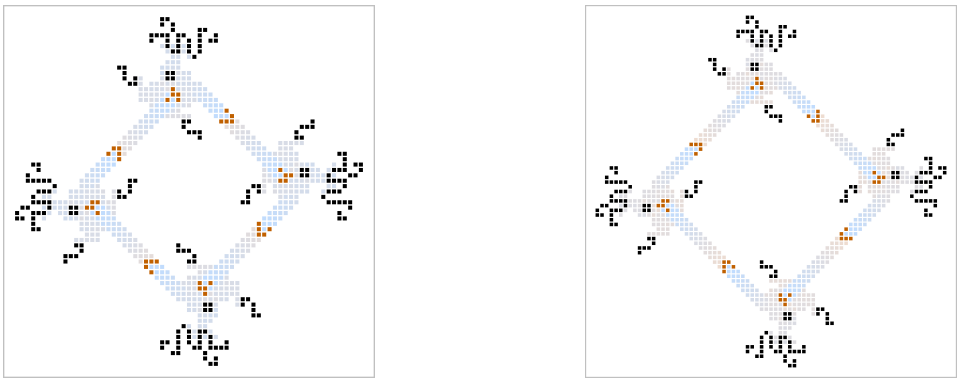
[15] Named after its discoverer, Stephen Silver, who constructed it in November 1998. We will discuss it in more depth in Section 7.9.

[16] The Snark was "almost" found by Dietrich Leithner sometime around 1998, but with the large unnamed still life at the right replaced by another block. In Leithner's original pattern, the glider was still reflected 90 degrees, but the second block was deleted in the process. It wasn't until about 15 years later, in April 2013, that the reflector was completed by Mike Playle, who wrote a custom search program to find a stable pattern that could replace the block and be unaffected by the reflection.

[17] The name "Snark" comes from Lewis Carroll's poem *The Hunting of the Snark*, in which ten characters try to hunt an imaginary animal bearing that name. Playle's search program that found the Snark was similarly called *Bellman*, after one of the main characters in the poem.

Free PDF. See conwaylife.com/book for pattern files and print version.

oscillators of any period of 43 or larger by reflecting gliders around in a loop. In fact, this method of construction gives the only known oscillators with period 43 or 53 (see Figure 3.30).



(a) A period 43 glider loop.

(b) A period 53 glider loop.

Figure 3.30: The Snark can be used to create oscillators of any period 43 or larger by reflecting multiple gliders (highlighted in orange) around in a loop. This method is demonstrated here with (a) a period 43 oscillator and (b) a period 53 oscillator.

3.6 Herschel Tracks

The idea behind glider loops is a straightforward one: by moving an object around a track, we can construct oscillators with large period. There isn't any fundamental reason why the object that we move around has to be a glider though—it could be a lightweight spaceship (assuming we could find a suitable reflector for it), for example, or even a small chaotic pattern whose debris we could control. It seems desirable to use a pattern that frequently occurs naturally, since then there would potentially be many different ways of reconstructing that pattern as we move it around the track.

The **Herschel** displayed in Figure 3.31 is one such pattern: it consists of only 7 cells, it explodes chaotically and takes 128 generations to stabilize, and it occurs frequently in the evolution of other patterns. For example, a Herschel is produced in generation 20 of the evolution of the B-heptomino (see Figure 3.32). Furthermore, a glider escapes from its debris at generation 21 of its evolution,[18] which means that it can be used in conjunction with glider loops or to create glider guns. Indeed, one of the advantages of creating oscillators via Herschels (as we will now do) instead of via glider loops is that they can be straightforwardly tweaked to create glider guns of the same periods.[19]

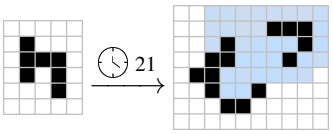


Figure 3.31: A **Herschel** is a chaotic pattern that emits a glider after 21 generations.

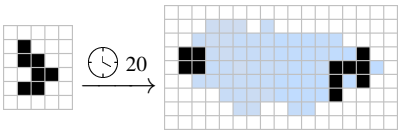


Figure 3.32: The B-heptomino takes 20 generations to evolve into a block and a Herschel.

In order to actually be able to make use of a Herschel, we will need to find a pattern that it can interact with (called a **conduit**) so as to move it from one place to another, since left alone a Herschel

[18] The glider that the Herschel emits was the first glider ever observed in the Game of Life. Richard K. Guy was checking the evolution of the R-pentomino (by hand on a Go board), which produced a B-heptomino after 28 generations, which then made a Herschel 20 generations later, which finally made a glider 21 generations after that.

[19] It is also worth keeping in mind that Herschel tracks were discovered 17 years earlier than the Snark, and that the techniques of this subsection are useful for much more than just creating oscillators and guns, as we will see in Chapter 7.

Free PDF. See conwaylife.com/book for pattern files and print version.

will just explode. Possibly the simplest Herschel conduit is the configuration of four blocks shown in Figure 3.33, which moves a Herschel and rotates it by 90 degrees over the course of 64 generations.[20] This conduit is called **R64**, with the "R" referring to the fact that the Herschel turns to the **r**ight and the "64" being the number of generations needed to move the Herschel (we will discuss conduit naming in more detail in Chapter 7 when we look at more general conduits).

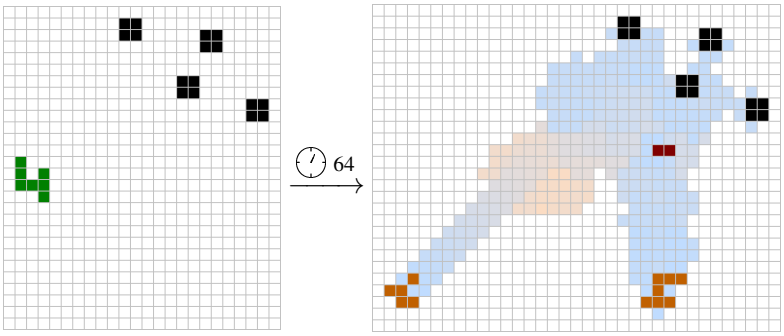


Figure 3.33: R64: A conduit consisting of four blocks that displaces and rotates a Herschel by 90 degrees in 64 generations (the domino spark on the right dies after one more generation and thus has no effect).

With this conduit, we can start playing a similar game to the one we played with glider reflectors: we can place multiple Herschel-turning conduits near each other in order to create a track that moves one or more Herschels from place to place, creating a wide variety of oscillators. For example, four of the 64-generation right-turn conduits can be used to create a period 256 track for a Herschel, resulting in glider guns and oscillators of period 256 (see Exercise 3.24). However, there are a couple of ways in which Herschel tracks built out of R64 conduits are more restrictive than glider loops built out of Snarks:

1) The output Herschel on the right in Figure 3.33 shoots a glider to the top-left, interfering with additional Herschels that enter the conduit next (unless we space consecutive Herschels very far apart on the track, which we sometimes don't want to do).
2) We cannot space out R64 conduits arbitrarily like we could with glider reflectors: the input and output locations of the Herschels for each conduit have to be lined up exactly in order for the track to work.

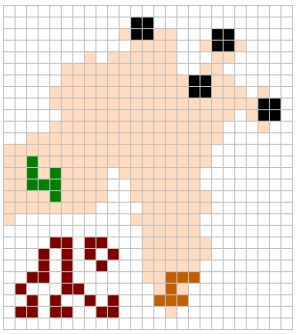
In order to take care of problem (1), we would like to use an eater to destroy the interfering glider from the output Herschel. However, this is actually rather tricky to do, as there is not enough room to place eater 1, eater 2, or eater 5 out of the way of the original Herschel while also destroying the glider before it crosses the path of a following Herschel. Instead, we can use the eater that we constructed back in Figure 2.24(b),[21] which lets this conduit accept Herschels that are spaced 61 or more generations apart (so its repeat time is 61 generations).

To help us make Herschel tracks of different sizes, and thus solve problem (2) above, we now introduce a second Herschel conduit. This conduit, which is called **Fx77**, moves a Herschel forward and flips it instead of rotating it (see Figure 3.34(b)). This displacement of the Herschel takes 77

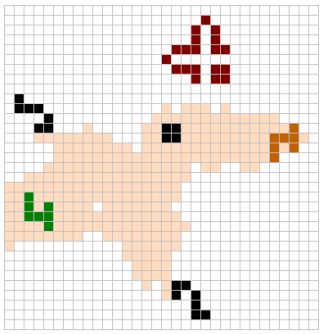
[20] This conduit was found by David Buckingham in September 1995.

[21] In fact, this is exactly *why* we constructed that eater: the constraints that we placed on the size of the eater in Figure 2.24(b) were designed exactly to make sure that it did not interfere with the input or output Herschel from this conduit (compare with Figure 3.34(a)).

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) A way of placing an eater near the R64 conduit so as to make its repeat time 61 generations.



(b) Fx77: A conduit that flips a Herschel in 77 generations and has a repeat time of 61 generations.

Figure 3.34: Two Herschel conduits that can be used to create oscillators of any period 61 or greater. Input Herschels are green, while output Herschels are orange. Eaters that are not required for the conduit to function, but are useful for erasing interfering gliders, are red.

generations to complete, and the conduit's repeat time is again 61 generations.[22] We note that the eater 2 located at the top of the conduit is not actually necessary for the conduit to function, but is just used to destroy the interfering glider from the output Herschel, so that we will be able to space Herschels together a bit more tightly on the track (just like the custom eater that we placed next to the R64 conduit in Figure 3.34(a)).[23]

It turns out that we can use these two Herschel conduits to create oscillators of any period 61 or greater. To give an idea of how we will construct these oscillators, let's first construct a simple loop that uses four copies of R64 and two copies of Fx77 on each side (see Figure 3.35(a)). A Herschel moves around this track in

$$\underbrace{4 \times 64}_{\text{R64 time}} + \underbrace{8 \times 77}_{\text{Fx77 time}} = 872 \text{ generations},$$

so we can place a single Herschel on this track in order to create a period 872 oscillator. However, since $872 = 2 \times 436$, we can also use this track to construct a period 436 oscillator by placing 2 Herschels at opposite ends of the track. Similarly, we can construct oscillators with period $872/4 = 218$ and $872/8 = 109$ by placing 4 and 8 equally spaced Herschels along the track, as demonstrated in Figure 3.35(b).

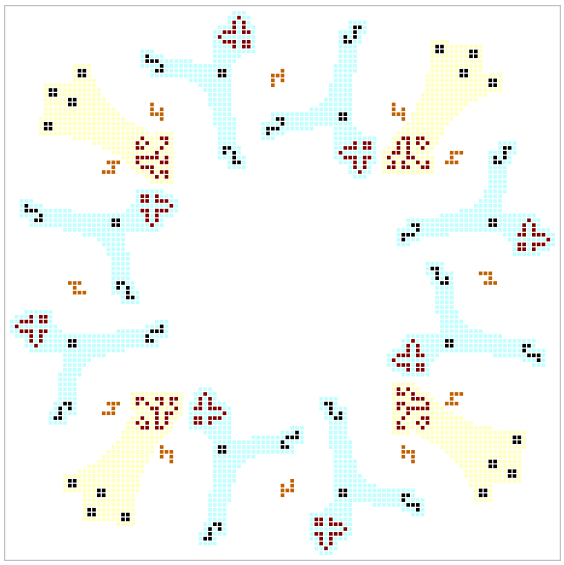
For this particular track, these are the only possible periods that we can construct, since the period must evenly divide 872 and must be at least 61 (the repeat time of the R64 conduit). However, we could make a larger Herschel track simply by using *four* copies of Fx77 on each side of the track instead of two, resulting in a track that takes $(4 \times 64) + (16 \times 77) = 1488$ generations for a Herschel to traverse. Since 1488 has prime decomposition $1488 = 2^4 \times 3 \times 31$, we could place 2, 3, 4, 6, 8, 12, 16, or 24 Herschels along the track to create oscillators with periods 744, 496, 372, 248, 186, 124, 93, or 62, respectively (see Exercise 3.27).

By arranging the R64 and Fx77 Herschel conduits in larger and larger squares like this, we can create oscillators with a wide variety of periods. To pin down exactly which periods are possible, we note that if we place $2k$ copies of Fx77 on each side of the track, then a Herschel requires

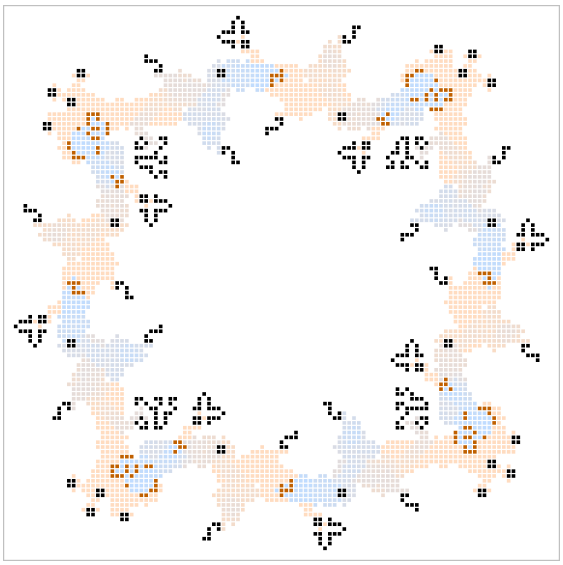
[22] The repeat time of the Fx77 conduit can be reduced to 57 ticks by suppressing its output glider with the same specialized eater used with the R64. However, that version of the Fx77 conduit does not allow room for an R64 conduit to follow it, and the R64 conduit can't be made to recover in 57 ticks in any case. 61 ticks is therefore the minimum period for combinations of these two conduits.

[23] We use eater 2 because we can place it slightly closer to the conduit without disrupting it than we can place eater 1.

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) By lining up conduits so that their input and output Herschel locations (highlighted in orange) overlap, we get a track that Herschels can move along (clockwise, in this case).



(b) Placing 8 Herschels (highlighted in orange, and not all in their usual phase) on the track (a) results in a period 109 oscillator, since the track has a length of $8 \times 109 = 872$ generations.

Figure 3.35: The Herschel track (a) consists of 4 copies of the R64 conduit (highlighted in yellow) and 8 copies of the Fx77 conduit (highlighted in aqua). Oscillators can be made by equally spacing a number of Herschels on the track that evenly divides the amount of time it takes for a Herschel to go around the track (872 generations in this case). An explicit oscillator is illustrated in (b) with 8 Herschels and period $872/8 = 109$.

$(4 \times 64) + (8k \times 77) = 256 + 616k$ generations to traverse the track. To create an oscillator with period p, we require $256 + 616k$ to be an integer multiple of p. At this point, a problem emerges: $256 + 616k \equiv 4 \pmod 7$ and $256 + 616k \equiv 3 \pmod{11}$ for all k, so these tracks will never give an oscillator whose period is a multiple of 7 or 11.[24] However, they can be used to construct oscillators with all other periods (no smaller than 61):

Theorem 3.1 — Herschel Track Oscillators of Most Periods 61 or Greater
Let $p \geq 1$ be a positive integer. There exists an integer $k \geq 1$ such that $256 + 616k$ is a multiple of p if and only if p is not a multiple of 7 or 11. Square Herschel tracks consisting of 4 R64 conduits and $2k$ Fx77 conduits per side can thus be used to create oscillators of any period 61 or larger that is not a multiple of 7 or 11.

Proof. Note that $256 + 616k$ is a multiple of p if and only if there exists an integer $a \geq 1$ such that $pa = 256 + 616k$. The problem then becomes to determine for which values of p there exist integers a and k such that $pa - 616k = 256$. This problem is solved exactly by Bézout's identity (Theorem A.1), which tells us that a and k exist if and only if 256 is an integer multiple of $\gcd(p, 616)$. Since the only prime factors of 616 that are not prime factors of 256 are 7 and 11, it follows that a and k exist if and only if p is not a multiple of 7 or 11. ■

Fortunately, we can construct oscillators to fill in the missing periods (i.e., the multiples of 7 and the multiples of 11) by just stitching together R64 and Fx77 conduits in different ways (i.e., by creating tracks that aren't squares). In particular, if we place an R64 conduit immediately after just

[24] If you are not comfortable with modular congruence, now would be a good time to read Appendices A.1 and A.2.

Free PDF. See conwaylife.com/book for pattern files and print version.

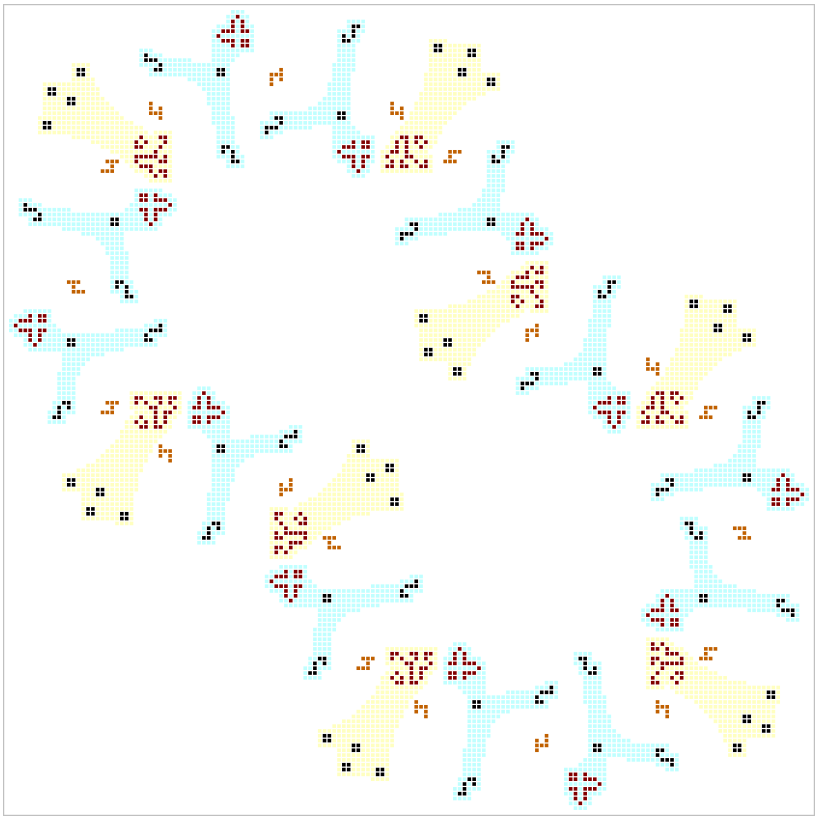


Figure 3.36: A Herschel track that just uses a single Fx77 conduit on some sides so that the next R64 turns the track outward instead of inward. Using this technique, we can construct a wide variety of Herschel tracks of almost any shape.

one Fx77 conduit instead of two, we can turn the track in the opposite direction. For example, the Herschel track in Figure 3.36 is slightly more exotic than the square tracks that we constructed earlier, and consists of 8 copies of R64 and 12 copies of Fx77.

We can think of the track in Figure 3.36 as coming from "folding in" the top-right and bottom-left corners of the square track that has four copies of Fx77 on each side. When we fold in the corners of the track like this, we replace two corner R64 conduits and eight of the Fx77 conduits (two on each side adjacent to each R64) with six R64 and four Fx77 conduits, which has the effect of decreasing the time that it takes the Herschel to traverse the track by

$$(2\times 64+8\times 77)-(6\times 64+4\times 77)=52 \text{ generations.}$$

We could similarly fold in the corners of larger square Herschel tracks, potentially multiple times, reducing the length of the track by 52 generations each time.

The point of doing this is that if we fold in the corners of a large square enough times, we end up with a track whose length is a multiple of 77 and thus can be used to construct oscillators whose period is a multiple of 7 and/or 11. In particular, if we take a square Herschel track with 26 or more Fx77 conduits on each side (i.e., $k \geq 13$) and we fold in the corners 76 times,[25] then it will take a Herschel $(256+616k)-(76\times 52)=77(8k-48)$ generations to traverse the track. This new family of tracks gives us exactly what we want:

[25] In general, if we have $2k$ Fx77 conduits on each side of the square track, we can fold in the corners up to $k(k-1)/2$ times. To be able to fold the corners in 76 times, we need $k(k-1)/2 \geq 76$, so $k \geq 13$.

Free PDF. See conwaylife.com/book for pattern files and print version.

Theorem 3.2 — Herschel Track Oscillators of All Periods 61 or Greater
Let $p \geq 1$ be a positive integer. Then there exists an integer $k \geq 13$ such that $77(8k-48)$ is a multiple of p. The Herschel conduits R64 and Fx77 can thus be used to create oscillators of any period 61 or larger.

Proof. The proof of this fact is very similar to that of Theorem 3.1: we note that $77(8k-48)$ is a multiple of p if and only if we can find positive integers a and k such that $pa - 77 \times 8k = -(77 \times 48)$. Bézout's identity (Theorem A.1) tells us that a and k exist if and only if 77×48 is a multiple of $\gcd(p, 77 \times 8)$, which is always the case since 48 is a multiple of 8. ■

It is perhaps worth noting that these Herschel tracks with $k \geq 13$ are extremely large, and for many periods it is possible to find much smaller Herschel tracks that work; we only chose this particular family of Herschel tracks because it is relatively straightforward to analyze, not because it is efficient. Furthermore, even though we have only discussed Herschels in the context of creating oscillators so far, they are useful for much more. For example, if we look back at the Silver reflector in Figure 3.29(a), we see the Fx77 conduit near its top-center. Indeed, that reflector works by converting the input glider into a Herschel, which is then funneled along a Herschel track without suppressing the gliders that it creates (and making use of some other conduits that we have not yet seen). We explore conduits and tracks like these ones in much more depth in Chapter 7.

3.7 Omniperiodicity

We have now seen numerous techniques for constructing oscillators of different periods, so we finally return to the question that we have been dancing around for the entirety of this chapter: do there exist oscillators with all periods in Conway's Game of Life? If so, then Life is said to be **omniperiodic**, and this omniperiodicity problem is one of its oldest and most well-studied questions.

While remarkable progress on this problem has been made over the years, and in fact we have already seen how to construct oscillators with *almost* any period, omniperiodicity in general remains unsolved, as there are still gaps that we do not know how to fill in. In particular, we do not know how to construct oscillators with period 19, 34, or 41.[26] A complete summary of the status of the omniperiodicity problem is given in Table 3.1, but we note that this table will likely become out of date reasonably quickly, as oscillators of new periods have been found somewhat regularly in recent years. In particular, new oscillator periods have been constructed in each of the years 2002, 2009, 2010, 2013, 2019, and 2022.[27]

Note that all of the periods for which we do not know the answer are "intermediate" in size. Small periods (say with period 8 or less) can often be found via clever computer searches, but higher periods are much more difficult to find in this way due to the size of the search space expanding so quickly. On the other hand, oscillators with high periods (say with period 30 or more) can often be found by combining various hassling and shuttling reactions in different ways, and of course very high period oscillators can be constructed via glider loops and Herschel tracks. Similarly, composite periods can often by constructed simply by placing sparks of lower-period oscillators next to each other. The result of combining these various techniques together is that there is somewhat of a rift, where we

[26] Even though we could just place a period 2 blinker next to a period 17 honey thieves to create a period 34 oscillator, we recall that this type of construction is considered "trivial" and is thus ignored because no cell oscillates at the full period.

[27] Noam Elkies found the first period 27 oscillator in 2002, Nicolay Beluchenko found the first known period 37 and 51 oscillators in 2009, Matthias Merzenich found the first known period 31 oscillator in 2010, the Snark-based oscillators of periods 43 and 53 were found in 2013, Luka Okanishi completed the first period 23 oscillator in 2019, and David Raucci discovered the first period 38 oscillator in 2022.

Free PDF. See conwaylife.com/book for pattern files and print version.

Period	Year of First Discovery	Examples
2	1970	bipole, blinker, clock, negentropy, toad
3	1970	caterer, jam, pulsar, two eaters
4	1970	confused eaters, mold, p4 thumb sparker, T-nosed p4
5	1971	octagon 2, pentoad
6	1972	unix, Figure 3.8(a)
7	1972	burloaferimeter, hebdarole
8	1970	blocker, cauldron, figure eight, Hertz oscillator, roteightor
9	1972	p9 thumb sparker, snacker, worker bee
10	$\leq$1976	Figure 3.4(c)
11	1977	rattlesnake
12	1972	dinner table, Figure 3.8(b)
13	1976	Figure 3.8(c)
14	1970	tumbler (see Exercise 1.1(b))
15	1970	pentadecathlon
16	1983	Figure 3.18(a)
17	1997	honey thieves
18	$\leq$1991	Figure 3.18(b)
19		none known
20	1995	p4 thumb sparker on fumarole, Figure 3.23(a)
21	1995	Figure 3.18(c)
22	1997	Figure 3.7(c)
23	2019	Figure 3.20(b)
24	1994	unix on figure eight, Figure 3.23(b)
25	1994	Figure 3.23(c)
26	1983	Figure 3.26(a)
27	2002	Figure 3.19(b)
28	1973	fountain on hebdarole
29	1980	Figure 3.26(b)
30	1970	eureka, queen bee shuttle, heavyweight volcano on unix
31	2010	p31 domino sparker (see Exercise 3.12)
32	1978	gourmet
33	1997	caterer on rattlesnake
34		no non-trivial examples known
35	1995	p5 pipsquirter on hebdarole
36	1984	T-nosed p4 on snacker, Figure 3.7(i)
37	2009	Figure 3.19(b)
38	2022	Exercise 3.14
39	2000	caterer on p13 sparker (see Exercise 3.12(a))
40	$\leq$1991	fumarole on blocker
41		none known
42	1994	unix on hebdarole
43+	2013 (some earlier)	Snark-based glider loops
61+	1996 (some earlier)	Herschel tracks

Table 3.1: A summary of the status of the omniperiodicity problem in Conway's Game of Life. The only three periods for which it is unknown how to construct an oscillator are 19, 34, and 41. Note that some large-period oscillators were found much earlier than 1996 or 2013, and these are not cataloged in this table. For example, the twin bees shuttle has period 46 and was found in 1971.

Free PDF. See conwaylife.com/book for pattern files and print version.

know lots of oscillators with small periods and lots of oscillators with large periods, but relatively few oscillators with prime periods in between those two extremes.

3.8 Phoenices

Before closing this chapter, we make a brief detour and consider the period 2 oscillator shown in Figure 3.37, which has the interesting property that all of its live cells die every generation, yet the pattern as a whole lives. A pattern with this property is called a **phoenix**, after the mythological bird that is cyclically reborn after dying. It seems natural to ask whether or not there exist more exotic patterns that are phoenices, such as spaceships or puffers. We begin by proving that the answer to this question is "no": every phoenix eventually evolves into an oscillator.[28] There's no such thing in Life as a phoenix spaceship.

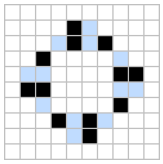


Figure 3.37: A period 2 phoenix oscillator.

Theorem 3.3 — Phoenices are Oscillators
In Conway's Game of Life, no phoenix can ever extend more than one cell outside of its original bounding box. In particular, every phoenix evolves into an oscillator.

Proof. To prove the result, we look at the rightmost edge of the phoenix's bounding box in generation 0, as depicted by the dark gray cells in Figure 3.38.

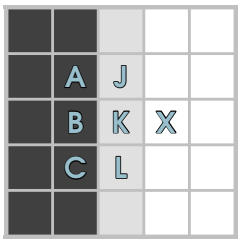


Figure 3.38: A diagram illustrating the fact that no phoenix can extend outside of its original bounding box (depicted by the dark gray cells) by more than one cell (depicted by the light gray cells). In particular, if the phoenix is initially confined to the dark gray cells, then cell X can never be born.

Suppose for a contradiction that some cell at a distance of 2 from the original bounding box is eventually born, and let X be the first such cell, which is born for the first time in generation n. Then cells J, K, and L in Figure 3.38 must each have been alive in generation $n-1$, and since the pattern is a phoenix, they must each have been dead in generation $n-2$. However, for cell K to be born in generation $n-1$, each of cells A, B, and C must have been alive in generation $n-2$, and thus must be dead in generation $n-1$. We have thus shown that in generation $n-1$, cell K is alive and has exactly two live neighbors (J and L) and thus lives on to generation n, so the pattern is not actually a phoenix.

The fact that every phoenix eventually evolves into an oscillator follows immediately: there are 2^{wh} different patterns that fit inside a $w \times h$ box, so any pattern that stays confined to such a box forever must return to a previous phase (and thus oscillate) after no more than 2^{wh} generations. ■

[28] This theorem was originally proved by Stephen Silver in January 2000.

Free PDF. See conwaylife.com/book for pattern files and print version.

Before proceeding, we note that the technique used in the proof of Theorem 3.3 is actually a very common one: to prove that a pattern with a certain property does not exist, consider what happens to the pattern at one of its far edges. We will use this same general method to prove another theorem about phoenices momentarily, and we will use it again in Section 4.5 to find bounds on how fast spaceships can travel.

Now that we know that all phoenices evolve into oscillators, we are left with the question of what periods they can have. We already saw an example of a period 2 phoenix, but to date no one has found any phoenices with period 3 or higher. The following theorem[29] shows that no phoenix of period 3 exists.

Theorem 3.4 — No Phoenices with Period 3
In Conway's Game of Life, there does not exist a phoenix oscillator with period 3.

Proof. Suppose there were a phoenix with period 3. Consider the uppermost row in the rotor of the phoenix, and let A be the leftmost cell in that row that is in the rotor, as in Figure 3.39. Without loss of generality, suppose that cell A is alive in generation 3.



Figure 3.39: A diagram depicting the top-left corner of a hypothetical period 3 phoenix oscillator. Cell A is the leftmost cell in the uppermost row of the oscillator, and exactly three of J, K, L, or M must be alive in the generation before A is alive.

We know that cell A must have three neighbors that are alive in generation 2, and they must be three of the cells J, K, L, or M in Figure 3.39. Suppose (for a contradiction) that cell K is alive in generation 2. Since K has at least two live neighbors in generation 2 (at least two of J, L, or M), it must in fact have at least four live neighbors (in order for it to die in generation 3). It follows that we can list K's (alive and dead) neighbors as follows:

- one cell (to its top-left) that is never alive,
- cell A, which is alive in generation 3,
- at least four neighbors that are alive in generation 2, and
- exactly three neighbors (its parents) that are alive in generation 1.

Since none of the neighbors that are alive in generations 1, 2, or 3 can be the same (since it is a period 3 phoenix), we have shown that K has at least nine neighbors, which is a contradiction that shows that K must in fact be dead in generation 2. It follows that each of J, L, and M are alive in generation 2.

Since L must have three parents that are alive in generation 1, they must be cells K, X, and Y in Figure 3.40. However, there is then only room for cell X to have at most two parents that are alive in generation 0 (in particular, the cell to its right and the cell to its lower-right), so it can't be born in generation 1, which is the contradiction that shows that this period 3 phoenix does not actually exist. ■

A computer-assisted proof has also been used to show that no phoenices of period 5 exist,[30] but essentially nothing else is known about the (non-)existence of phoenices with period 4 or greater. On

[29] This theorem was also originally proved by Stephen Silver in January 2000.
[30] This proof was completed by Alex Greason in September 2019.

Free PDF. See conwaylife.com/book for pattern files and print version.



Figure 3.40: A diagram that shows that no period 3 phoenix oscillator exists. Cell A is alive in generation 3, cells J, L, and M are alive in generation 2, and cells K, X, and Y are alive in generation 1. Cell X has only two neighbors (to its right and lower-right) that could potentially be alive in generation 0, so there was in fact no way for it to be born.

the other hand, if we relax the restriction on phoenices a bit and instead ask whether or not there exist higher-period oscillators with the property that every cell in the oscillator actually oscillates (i.e., the oscillator has no stator), it turns out that the answer is yes. For example, every cell in the period 3 oscillator displayed in Figure 3.41 oscillates, even though it is not a phoenix.

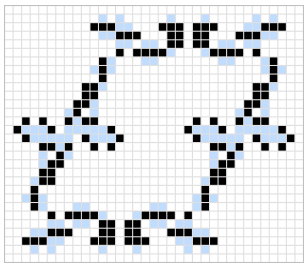


Figure 3.41: A period 3 oscillator with the property that every cell oscillates. Found by Jason Summers in August 2012.

3.9 Notes and Historical Remarks

The search for new oscillators has been one of the most consistently active research areas throughout the history of Life. Much of its motivation comes from the omniperiodicity problem—it's remarkable that such a seemingly simple problem still remains unsolved after all these years, despite the various techniques that have been developed for constructing oscillators. To illustrate just how incomplete our methods of constructing and searching for oscillators are, consider the small period 16 oscillators displayed in Figures 3.42 and 3.43. Despite their simplicity and small size, they were not discovered until July 2016 and February 2020, respectively, and they were found in possibly the least enlightening way possible—as a result of evolving computer-generated random soups (by apgsearch).

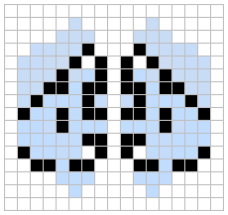


Figure 3.42: Rich's p16 is a period 16 oscillator that was found in a random soup by Rich Holmes in July 2016.

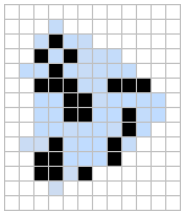


Figure 3.43: Rob's p16 is a period 16 oscillator that was found in a random soup by Rob Liston in February 2020.

No matter how they were discovered, oscillators can be highly useful in pattern engineering, since

Free PDF. See conwaylife.com/book for pattern files and print version.

sparkers in particular can be used for a great many purposes. We already saw that they can be used to create higher-period oscillators with composite periods, but they also form the basic building blocks of many of the large patterns that we will construct in later chapters. For example, the entirety of Chapter 6 is devoted to using oscillators to emulate logic circuits that can perform computations.

Many of the small billiard table oscillators that are known were found by David Buckingham, who discovered dozens of them throughout the 1970s and 1980s. Herschel tracks were also primarily his invention—he spent several years cataloging patterns that moved B-heptominoes and Herschels from one place to another, and by October 1996 he had a complete toolkit capable of constructing oscillators and gliders guns with any period at least 61.

Almost immediately, Paul Callahan used Herschel tracks to construct the first explicit stable glider reflector [BC98],[31] which had a repeat time of 4 840 generations. It didn't take long for him to make some optimizations, resulting in a stable reflector with repeat time of 894 generations. Various optimizations were then made with the help of Dean Hickerson, getting the repeat time down to 747 in November 1996, followed by David Buckingham reducing the repeat time to 672 in May 1997, Stephen Silver reducing it to 623 in October 1997, Paul Callahan reducing it to 575 in November 1998, and finally Stephen Silver reducing it to 497 (as in the Silver reflector of Figure 3.29(a)) a few days later.

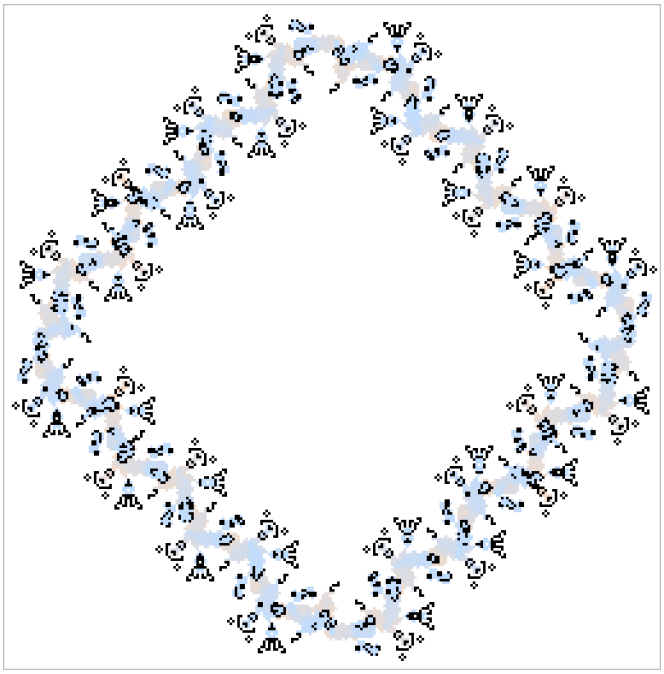


Figure 3.44: A period 56 oscillator that was constructed using periodic Herschel conduits. It was built by Dietrich Leithner in December 1997.

A 180-degree stable reflector with repeat time of 202 generations, called the **boojum reflector**,[32] was found by Dave Greene in April 2001, and a 180-degree stable reflector with repeat time of 106 generations, called the **rectifier**, was found by Adam P. Goucher in March 2009 (see Exercise 3.23). Finally, the Snark that we saw in Figure 3.29(b), with a repeat time of just 43 generations, was found by Mike Playle in April 2013.

[31] Although it had been known since the early 1970s that stable reflectors must exist in Life [Wai74, BCG82], no specific examples had been found prior to Callahan's construction.

[32] In Lewis Carroll's epic poem *The Hunting of the Snark*, a "Boojum" is a particularly dangerous type of Snark.

Free PDF. See conwaylife.com/book for pattern files and print version.

Although Buckingham's original Herschel conduits were used to construct oscillators with period 61 or higher, faster-recovering conduits can be used to create oscillators with period 58, 59, and 60 as well (see Exercise 3.29). In December 1997, Dietrich Leithner found fast oscillating Herschel conduits that even allow for the construction of oscillators with period 56 and 57 (see Figure 3.44 and Exercises 3.31 and 3.32). In fact, these oscillating conduits were used to construct the first known period 57 oscillator.

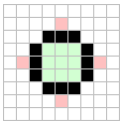
Herschel tracks remain one of the pinnacles of Life technology to this day, although they have evolved somewhat—many objects other than Herschels and B-heptominoes can be moved around tracks and converted into each other. We will return to this topic and investigate it more thoroughly in Chapter 7.

Exercises

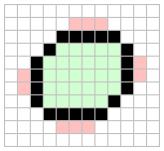
solutions to starred exercises on page 453

∗**3.1** [2/5] For each of the following boxes, use induction coils to crowd the red cells and then find something that you can put in the green cells so as to make a billiard table oscillator.

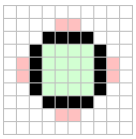
(a)



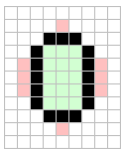
(b)



(c)



3.2 [3/5] Consider the 5×3 box displayed below as a potential starting point for constructing a billiard table oscillator.



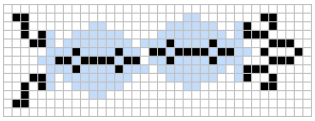
(a) Find a stable way to crowd the red cells, just like we used two blocks and two snakes to crowd the outside of a 4×3 box in Section 3.1.

(b) Show that there is no way to fill in the green cells so as to make this pattern an oscillator (try writing a computer program to help you).

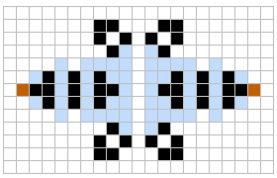
3.3 [2/5] Replace the eater 1s with eater 2s in the oscillators from the following figures, without destroying the oscillator or altering its period:

(a) Figure 3.7(b),
(b) Figure 3.7(c),
(c) Figure 3.7(f),
(d) Figure 3.7(h), and
(e) Figure 3.7(g) (you can only replace two of the eater 1s).

3.4 [1/5] The period 9 snacker oscillator from Figure 3.6 can be stabilized on one side in ways other than using two eater 1s. Use the method shown below to extend this oscillator to one that makes use of at least 5 interacting pentadecathlons.

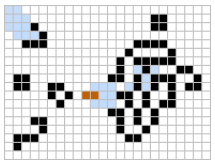


∗**3.5** [2/5] Use the sparks from either the T-nosed p4 or the T-nosed p6 (displayed below) and another oscillator of your choice to create non-trivial oscillators with...



(a) period 12,
(b) period 20, and
(c) period 24.

3.6 One of the primary uses of pipsquirters is their ability to reflect a glider when a boat, block, and eater 1 are placed nearby as shown below.



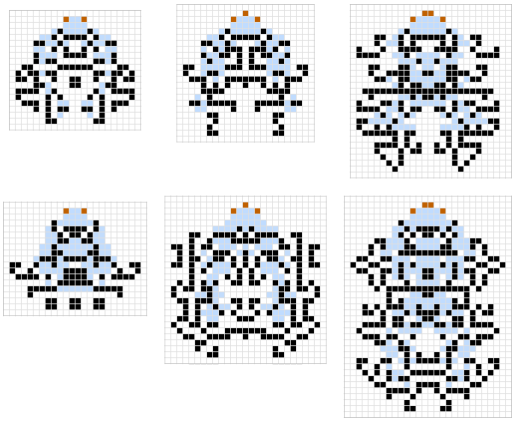
The reflector above has period 6. Use this same reaction to create a glider reflector with...

(a) [2/5] period 7, and
(b) [3/5] period 8.

∗**3.7** [1/5] Show how the snakes can be replaced by eater 1s in the period 32 "gourmet" oscillator from Figure 3.19(a).

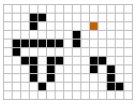
Free PDF. See conwaylife.com/book for pattern files and print version.

3.8 Several period 4 and 5 lightweight, middleweight, and heavyweight supervolcanoes are displayed below,[33] and they are particularly useful due to how far they push away their sparks.



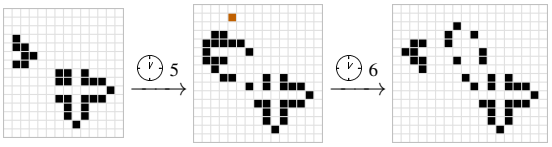
(a) [2/5] Use the sparks from two supervolcanoes to construct a period 20 oscillator.
(b) [2/5] Use a supervolcano and a thumb sparker to create a non-trivial period 20 oscillator.
(c) [3/5] Use a supervolcano to help you construct a T-tetromino shuttle.

∗**3.9** [2/5] Consider the oscillator and spark (in orange) shown below:

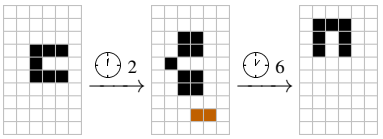


(a) What is the period of the oscillator when the spark is not present?
(b) Construct a new oscillator by using a period 4 sparker to provide the orange spark. What is the period of this oscillator?
(c) The oscillator constructed in part (b) is called a **phase shift oscillator**. Give a simple formula for the possible periods of this family of phase shift oscillators.

3.10 [2/5] A dot spark and an eater 2 can be used to reflect a B-heptomino in 11 generations, as shown below. Use this reaction to create a period 22 oscillator.

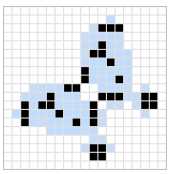


∗**3.11** [4/5] A domino spark can be used to rotate a pi-heptomino by 90 degrees in 8 generations, as shown below. Use this reaction to create a period 32 oscillator. [Hint: The spark from a figure eight does not quite work. Try some other domino sparkers—you may need to weld them together in order to make them fit together closely enough.]

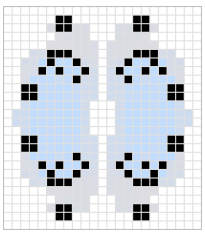


∗**3.12** [2/5] This exercise introduces some sparkers with higher period than we saw in Section 3.3.1. For each of them, determine their period, find an accessible spark that they emit (the spark might only be present in one of its phases not displayed below), and use that spark to construct a non-trivial oscillator with higher period.

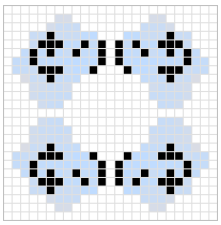
(a)



(b)

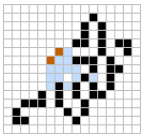


(c)

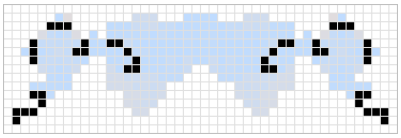


3.13 [3/5] Use the hassling reaction from Figure 3.20(a) to create an oscillator with...

(a) period 24, and
(b) period 21, using the following p7 duoplet sparker:



3.14 [2/5] Show how the following period 38 oscillator[34] can be used to reflect a glider.

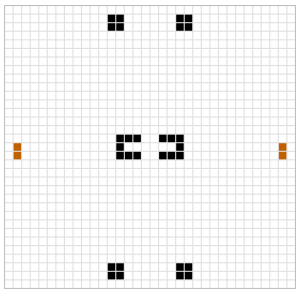


[33] The p4 lightweight, p5 middleweight, and p5 heavyweight supervolcanoes were found by Noam Elkies in 2010, Dongook Lee in 2019, and Karel Suhajda in 2004, respectively. The rest were found by Matthias Merzenich in 2010.

[34] Found by David Raucci in January 2022, based on an earlier infinite p38 pattern that Jason Summers found in August 2000.

Free PDF. See conwaylife.com/book for pattern files and print version.

∗**3.15** [3/5] A partial pi-heptomino hassler is displayed below. Complete it by placing sparkers so that domino sparks are present at the indicated locations 33 generations after the phase shown here.



[Hint: What will the period of this oscillator be? Make sure that the hassler you pick has a compatible period.]

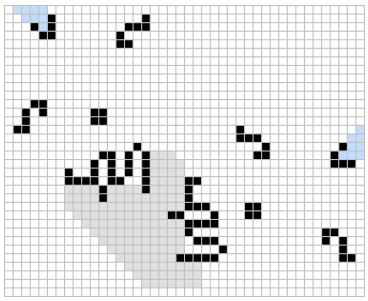
3.16 [2/5] Use two pentadecathlons to construct a glider loop oscillator with period 180.

3.17 [2/5] Use four Snarks to construct an oscillator with period 180.

∗**3.18** [2/5] Create a glider loop (of any period) that makes use of exactly six Snarks.

∗**3.19** [3/5] Sometimes it is useful to weld together reflectors, just like we welded together eaters in the previous chapter.

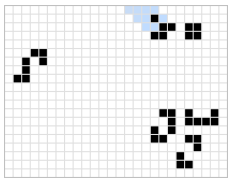
(a) Weld together the following two partial Snarks (modify only the light gray cells), thus creating a single still life that can reflect gliders from two different sides.



(b) Modify the reflector from part (a) to create a single reflector that can reflect gliders coming from four different sides.

∗**3.20** [2/5] The Snark reflector works by converting the input glider into another object that we have seen before, and then converting that object into the output glider. What is the name of the intermediate object?

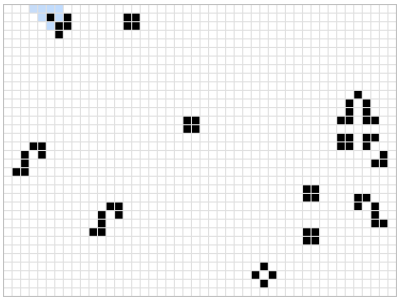
∗**3.21** [2/5] The pattern displayed below[35] might be called an "almost Snark", since it can *almost* be used as a 90-degree glider reflector. Describe what goes wrong and why it cannot reflect more than one glider.



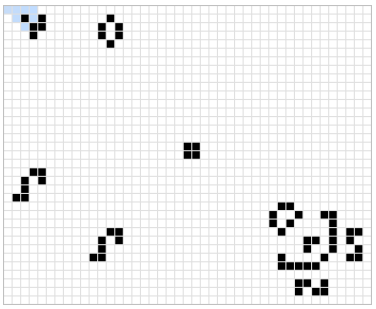
3.22 [3/5] Create a glider loop that uses at least one buckaroo and at least one twin bees shuttle. What is the smallest possible period of such an oscillator?

3.23 [3/5] For each of the following 180-degree stable reflectors, construct an oscillator that works by using two copies of the reflector to bounce a glider back and forth, and determine what the possible periods of such oscillators are.

(a) **Boojum reflector**:



(b) **Rectifier**:



3.24 [3/5] Recall the 64-generation 90-degree Herschel conduit in Figure 3.33.

(a) Use four copies of the conduit to create a gun that shoots 4 gliders every 256 generations.

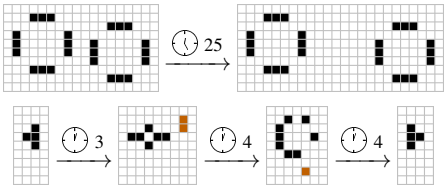
(b) Modify the pattern from part (a) to create a period 256 oscillator.

(c) Can you add additional Herschels to this track to make an oscillator with period 128 or 64? Either explicitly construct an example or explain the problem that arises.

[35] Found by Tanner Jacobi in 2015.

Free PDF. See conwaylife.com/book for pattern files and print version.

∗**3.25** [4/5] Two reactions that move traffic lights and T-tetrominoes are displayed below (the top one is called a **traffic jam**):



Use these reactions (and optionally any of the other T-tetromino hassling reactions we have seen) to create an oscillator.

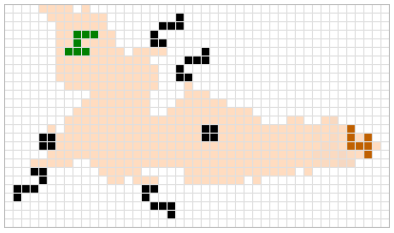
∗**3.26** [3/5] What type of sparks does the traffic jam reaction from the top of Exercise 3.25 produce? Use this spark and the oscillator constructed in Exercise 3.25 to create a periodic glider reflector.

3.27 [3/5] Use the R64 and Fx77 conduits to construct Herschel track oscillators or guns with the following periods. [Hint: Mimic our construction in Figure 3.35.]

- **(a)** Period 218 oscillator.
- **(b)** Period 62 oscillator.
- **(c)** Period 69 glider gun.
- **(d)** Period 61 oscillator.
- **(e)** Period 70 glider gun.

3.28 [2/5] By placing different numbers of Herschels on the track from Figure 3.36, oscillators of which periods can be created?

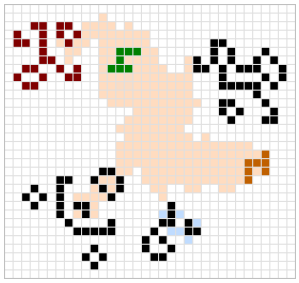
3.29 Consider the Herschel conduit displayed below, which rotates a Herschel counter-clockwise in 112 generations and has a repeat time of 58 generations. This conduit is called **L112**.



- **(a)** [3/5] Construct a period 109 oscillator by using only the L112 and Fx77 conduits (and possibly some eaters to destroy stray gliders).
- **(b)** [4/5] The L112 conduit has a slightly faster repeat time than the R64 conduit (58 generations instead of 61). Use the L112 and Fx77 conduits to create an oscillator with period 58, 59, or 60.
  [Hint: Refer back to the footnote in Section 3.6 about the Fx77 conduit's repeat time.]

∗**3.30** [3/5] Based on the phoenix oscillator shown in Figure 3.37, we might be tempted to guess that Theorem 3.3 can be improved to show that no phoenix can ever leave its original bounding box at all. Show that this conjecture is false. That is, find a phoenix that does in fact leave its original bounding box.

3.31 [4/5] Consider the (oscillating!) Herschel conduit displayed below, which rotates a Herschel counter-clockwise and flips it in 65 generations, and has a repeat time of 57 generations. Construct a period 57 oscillator by using this new conduit and the Fx77 conduit (and possibly some eaters to destroy stray gliders).

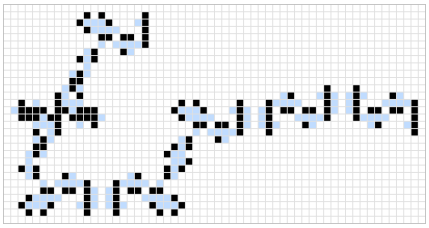


3.32 [3/5] Consider the period 56 Herschel track oscillator depicted in Figure 3.44. Break this oscillator down into its individual conduits, and for each conduit state (a) how many generations it takes a Herschel to traverse the conduit, and (b) what the conduit's repeat time is. Also break down each conduit into still lifes, oscillators, and eaters that we have seen earlier.

∗**3.33** [4/5] Prove that there is no period 4 oscillator with exactly one cell that oscillates at the full period.

3.34 [4/5] Prove that there is no period 4 oscillator whose rotor consists of exactly 4 cells, which take turns being alive for one generation out of each four.

∗**3.35** [2/5] The period 3 oscillator presented in Figure 3.41 was constructed by putting together the various period 3 "pieces" displayed below. Use these same pieces to construct a larger period 3 oscillator with at least 200 live cells in one of its phases.



Free PDF. See conwaylife.com/book for pattern files and print version.

4. Spaceships and Moving Objects

> Life is like riding a bicycle; to keep your balance, you must keep moving.
>
> Albert Einstein

We now shift our focus from stationary objects, like still lifes and oscillators, to moving objects, like spaceships and puffers. Recall that a spaceship is a pattern that returns to its initial phase after some number of generations, but at a different location from where it started. Just as was the case for oscillators, the **period** of a spaceship is the smallest number of generations needed for it to return to its initial phase.

We can also talk about the **speed** of moving objects: the number of cells that they move on average per generation. Since no object in the Life plane could possibly move at a speed of greater than 1 cell (in the Moore neighborhood sense) per generation, this speed is typically referred to as the **speed of light** and is denoted by c, while other speeds are represented as fractions of c. For example, since the glider moves diagonally by 1 cell every 4 generations, on average it moves $1/4$ cell per generation, so it has a speed of $c/4$. Similarly, the light/middle/heavyweight spaceships move 2 cells every 4 generations, and thus have a speed of $2c/4 = c/2$ (see Figure 4.1).

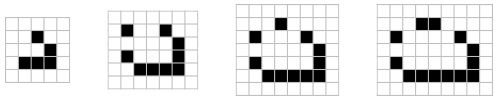


Figure 4.1: The four basic spaceships in Conway's Game of Life. From left to right, these are the glider (which moves diagonally at a speed of $c/4$) and the light/middle/heavyweight spaceships (which each move orthogonally[1] at a speed of $c/2$).

Although some of the first objects that we ever encountered in Life were spaceships, constructing new ones is actually quite a difficult problem, and a far smaller variety of spaceships is known than

[1]The term **orthogonal** means straight left–right or up–down, as opposed to diagonal.

Free PDF. See conwaylife.com/book for pattern files and print versio

of still lifes or oscillators. In fact, finding new spaceships is so difficult that the four "basic" types of spaceships, together with some of their tagalongs (which we will introduce shortly), were the only known spaceships for the first 19 years of Life's history.[2] Due to the difficulty of constructing new spaceships, instead of focusing on methods of construction, much of this chapter will focus on investigating what we can *do* with the spaceships that we already have.

4.1 The Glider

The glider is the single most useful spaceship that exists,[3] since its small size and frequent appearance from random soups make it easy to generate and manipulate. We have already seen numerous methods of generating gliders: the Gosper glider gun of Section 1.3, the twin bees gun of Section 1.4, and the glider-producing switch engine of Section 1.5. We will not present any additional methods of constructing gliders here, but we will see some as we progress through the book, including some moving sources of gliders in Section 4.4, and glider guns of any period of our choosing in Chapter 8.

We have also seen that we can delete gliders (Section 2.3) and that we can reflect gliders (Section 3.5). When we start manipulating gliders via these types of patterns, it will be important for us to be able to talk about the relative timings and positions of different gliders, so we now introduce some terminology that lets us do so.

4.1.1 Color of a Glider

When reflecting a glider, it is important to be aware of the glider's **color**,[4] which is a property of a glider that stays constant as it moves, but can change when it hits a reflector. Specifically, imagine that the Life grid is colored with two colors like a checkerboard, with adjacent cells (in the von Neumann neighborhood sense) always having different colors. The color of a glider is the color of its leading cell when it is in the phase that can be rotated to look like the glider in Figure 4.2.

It is worth emphasizing two potential points of confusion regarding glider color:

- The color of a glider's leading cell in its phases *other* than the one from Figure 4.2 is irrelevant. For example, all of the gliders in Figure 4.3 have the same color as each other, since after evolving them into the correct phase, their leading cell is on a white cell of the checkerboard pattern.
- We typically consider the color of a glider as a relative property, not an absolute one. That is, we talk about two gliders having the same or different color, but it is not often useful to talk about a single glider having a certain color.



Figure 4.2: This glider's color is white since its leading cell (circled in red) is located at one of the white cells on the checkerboard pattern.

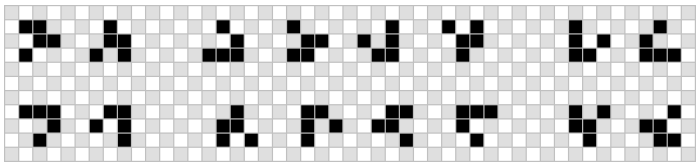


Figure 4.3: A collection of 16 gliders that all have the same color as each other. If any of these gliders were moved to the left, right, up, or down by 1 cell then their color would change.

[2] Martin Gardner wrote in a 1985 follow-up to his two *Scientific American* articles that "As for spaceships... no new ones have been discovered other than those already known to Conway in 1970."[Gar83] Dean Hickerson found the first truly new spaceships via computer search in 1989.

[3] And arguably the single most useful *pattern* that exists.

[4] The term **parity** is sometimes used instead of color.

Free PDF. See conwaylife.com/book for pattern files and print version.

To illustrate why a glider's color is important, consider the task of reflecting a glider as in Figure 4.4. We might first try to use the Snark to perform the reflection, but we quickly find that no matter how we place the Snark in the path of the input glider, the output glider never quite ends up where we want it. The reason for this is that the desired input and output gliders have opposite colors, but the Snark always produces an output glider that has the same color as its input. For this reason, the Snark is called a **color-preserving** reflector, and we instead need a **color-changing** reflector to get the output glider to travel through the desired location. One color-changing glider reflector that we have already seen is the twin bees shuttle (recall from Figure 3.16(a) how it can be used as a reflector), which can reflect the glider in the desired manner as in Figure 4.5.

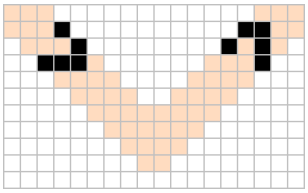


Figure 4.4: A path along which we would like to reflect a glider. Notice that the reflected glider has the opposite color of the original glider.

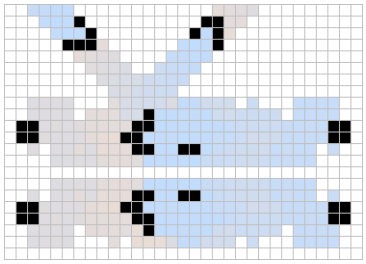


Figure 4.5: Twin bees shuttle is a color-changing reflector that can be used to reflect the glider in the desired way.

Keeping track of a glider's color also helps us keep track of which types of glider loops are and are not possible. For example, it is not possible to construct a glider loop that uses 3 Snarks and a single twin bees shuttle, since the twin bees shuttle will change the glider's color, but it won't be changed back before hitting the twin bees shuttle again, and thus can't possibly return back to where it started. In general, every glider loop must make use of an even number of color-changing reflectors.

4.1.2 Glider Lanes and Timing

Sometimes it is useful to compare not just the color of two gliders, but also the exact amount by which their positions differ. More specifically, we would like to be able to discuss how far gliders are in front of other gliders (even if they are somewhat offset to the side of each other), and we would also like a way of discussing how far to the side of each other they are. We can only really make these comparisons if the gliders are traveling in the same direction, so all gliders that we consider throughout this section are (arbitrarily) chosen to travel from the top-left to the bottom-right.

First, we partition the Life plane into diagonal lines of cells with slope -1, which we call **lanes**. We choose one of these lanes to be "lane 0" and then number the lanes so that they increase from left to right. Then we say that the lane of a glider is the lane occupied by the glider's leading cell when it is in the phase displayed in Figure 4.6, much like we defined the color of a glider in terms of the location of this phase's leading cell.[5] For brevity, we sometimes refer to a single lane separation as a **half diagonal** (or **hd** for short) and a two-lane separation as a **full diagonal** (or **fd** for short). For example, if two gliders are separated by 4 lanes then we might say that their spacing is "4hd" or "2fd".

Comparing the **timing** of gliders is straightforward if they are in the same lane: we choose some glider in that lane to have timing 0 when it is in the phase depicted in Figure 4.6, and we say that the timing of any other glider in that lane is the number of generations needed to move the glider with timing 0 to its position. To extend this definition so that we can compare the timing of gliders in different lanes, we then say that gliders of the same color have the same timing as each other if

[5] In fact, a glider's color can just be thought of as its lane modulo 2.

Free PDF. See conwaylife.com/book for pattern files and print version.

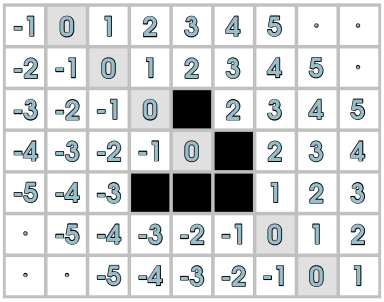


Figure 4.6: A glider that is in lane 0, which is highlighted in light gray. The numbers on the grid indicate the lane number of each cell in the plane, and the lane of every glider in the plane is determined by the location of its leading cell when it is in the phase displayed here.

they are on the same diagonal line of slope 1,[6] and we say that a glider of the opposite color has timing 2 generations higher than a glider that is one cell to its left. These timing rules are illustrated in Figure 4.7.

Just as with glider color, we are typically not interested in the absolute lane number or absolute timing of a single glider, but rather we talk about the number of lanes between two gliders and the number of generations by which their timing differs.

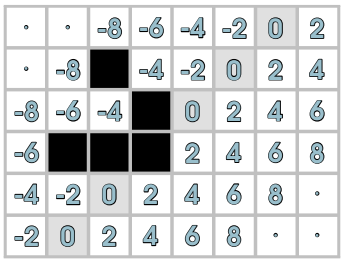


Figure 4.7: A glider that has timing 0, with the diagonal line of light gray cells indicating the other locations of leading cells that are considered to have timing 0. The numbers on the grid indicate the timing of a glider when its leading cell is in that location.

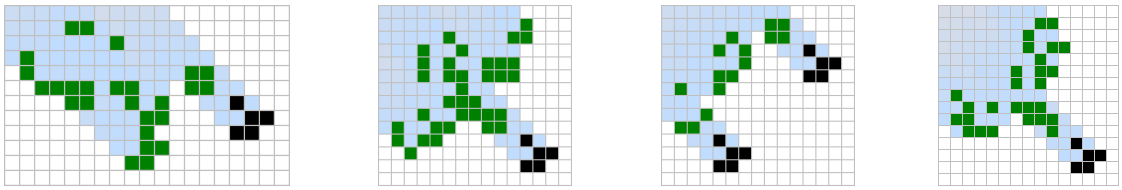
4.1.3 Tagalongs

Some spaceships, especially ones that emit accessible sparks, are capable of carrying other objects along with them as they move. Such objects are called **tagalongs**, and we now look at some examples of objects that can tag along with the glider. Although the glider does not really have any sparks, its rearmost cell is nonetheless far enough away from the body of the glider that it can carry some tagalongs with it, as demonstrated in Figure 4.8. Notice that these tagalongs do not actually touch the glider in any of these examples. Rather, a dying cell at the back end of the glider adds a third neighbor to a dead cell at the front of the tagalong. By the time a new cell is born at that location, the glider has moved away, so every phase of the glider always has a ring of dead cells around it—and yet, if the glider is not present, the tagalong will collapse.

One useful feature of tagalongs is that they often emit sparks as well, so we can sometimes chain them together or use them to produce reactions that are impossible with the original spaceship. For example, the B29 tagalong emits both a dot spark and a domino spark (displayed at the top center in Figure 4.8(a)), which other tagalongs can latch onto, as in Figure 4.9. Furthermore, *this* new tagalong also emits a dot spark, which allows it to pull additional tagalongs, and in this way we develop a sort

[6] That is, the same diagonal that goes from bottom-left to top-right.

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) B29 (b) Canada goose (c) crab (d) Orion 2

Figure 4.8: Some tagalongs (highlighted in green) that can trail behind a glider. They were found by (a) Hartmut Holzwart in 2004, and Jason Summers in (c) 2000, and (b,d) 1999.[7]

of grammar for $c/4$ diagonal spaceships: there are dozens of different $c/4$ diagonal tagalongs known, and we can construct a wide variety of $c/4$ diagonal spaceships by attaching these tagalongs to each other in different ways.

Similarly, the crab emits a dot spark (seen at its back left in Figure 4.8(c)) that can be used to turn a tub into a barge, and then a long barge, and then a long long barge, and so on, as illustrated in Figure 4.10. A pattern with this property is called a **tubstretcher**, or more generally a **wickstretcher** if we do not care about which particular object is stretched.

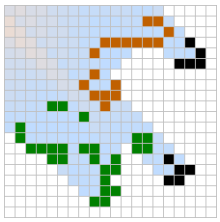


Figure 4.9: The B29 can pull a tagalong (highlighted in orange) that was found by Nicolay Beluchenko in 2005.

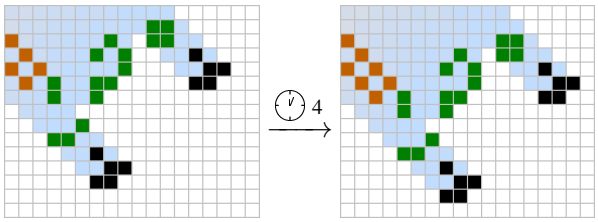


Figure 4.10: A pattern based on the crab called a **tubstretcher** that lengthens a tub (highlighted in orange) by 2 cells every 4 generations.

While we have seen infinitely growing patterns before (such as the Gosper glider gun in Section 1.3 and some switch engine puffers in Section 1.5), it is worth observing that all infinitely growing patterns that we saw previously worked by creating many disconnected small objects, whereas this one is quite different in that it creates a single arbitrarily large object.

4.2 The Light, Middle, and Heavyweight Spaceships

Although the glider is the easiest spaceship to create and manipulate, the light, middle, and heavyweight spaceships are often better at manipulating *other* objects. The reason that they can interact with so many other patterns in such a wide variety of ways is that they frequently emit accessible sparks. Every second generation, the LWSS emits a dot spark and a thumb spark, the MWSS emits two dot sparks and a thumb spark, and the HWSS emits a dot spark, a domino spark, and a thumb spark (see Figure 4.11).

One useful feature of these sparks is their ability to destroy other nearby objects, as demonstrated in Figure 4.12. The most common use of these reactions is to adjust the debris left behind by puffers. For example, suppose that we had a $c/2$ orthogonal puffer that left behind a combination of blocks, blinkers, and gliders as it moved. By carefully positioning some middleweight spaceships behind this

[7] The name "Orion 2" is a reference to a similar, but larger, spaceship called **Orion** that was found by Hartmut Holzwart in 1993.

Free PDF. See conwaylife.com/book for pattern files and print version.

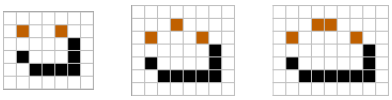


Figure 4.11: The light, middle, and heavyweight spaceship each give off several sparks (depicted in orange) that are extremely useful.

puffer, we could eliminate the unwanted debris (typically the blocks and blinkers), leaving only the desired output (typically the gliders).

On the other hand, it is also often useful to have other objects destroy these spaceships. We already saw how an eater 1 can destroy an LWSS or MWSS in Figure 2.15, and an eater 2 can destroy an LWSS or MWSS as in Exercise 2.10. Destroying an HWSS is slightly trickier, but three eaters that work are presented in Figure 4.13. The first of these eaters has the downside of being rather large, whereas the second eater has a very high recovery time; the pond and block are completely destroyed by the HWSS, leaving behind a beehive and a glider which then collide, miraculously reconstructing the block and pond exactly in their original positions.[8] The third eater is small and has a fast recovery time, but at the expense of having period 2 and thus only being able to destroy streams of heavyweight spaceships with even period.[9]

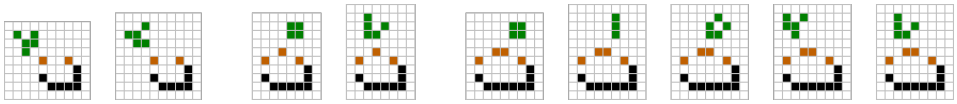


Figure 4.12: Light, middle, and heavyweight spaceships about to destroy some common small objects.

4.2.1 Flotillae and Tagalongs

Just like we could attach various tagalongs to the back of a glider, we can also use tagalongs to extend the light, middle, and heavyweight spaceships. Some of these tagalongs are presented in Figure 4.14. The **hivenudger** of Figure 4.14(d) (whose name comes from the fact that it pushes a pre-beehive) is somewhat versatile in that any of the lightweight spaceships at its corners can be replaced by a middleweight or heavyweight spaceship (see Exercise 4.12).

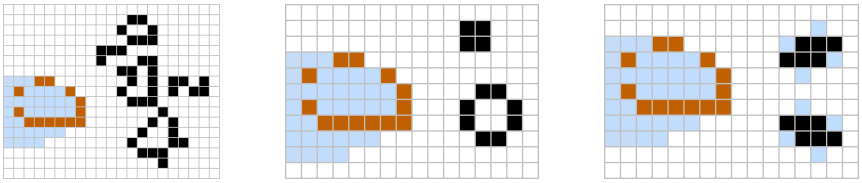


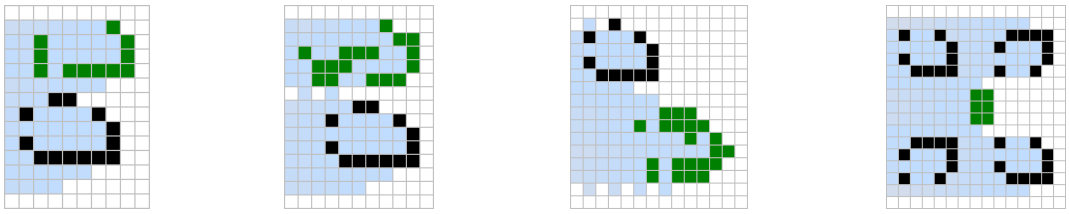
Figure 4.13: Three methods of eating a heavyweight spaceship. The arrangement of two toads on the right is called **killer toads**, and it can also eat many other objects, such as an MWSS in the same position.

The MWSS tagalong in Figure 4.14(c) is somewhat special for the fact that it attaches to the front end of a spaceship, rather than its side or rear. Tagalongs with this property are sometimes called **pushalongs**, and they are quite a bit rarer than other types of tagalongs, since it is not common for spaceships to have accessible sparks near their front that can support another object. There are

[8] This reaction in which a glider turns a beehive into a pond and block is called a **honey bit**.

[9] The large stable HWSS eater was found by Dean Hickerson in 1999 (with a slight size improvement by Karel Suhajda in 2003), and the eater comprised of a pond and a block was found by Brice Due in 2007. The killer toads have been known since the very early days of Life.

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) Sidecar **(b)** HWSS tagalong **(c)** MWSS tagalong **(d) Hivenudger**

Figure 4.14: Some small tagalongs (highlighted in green) for light, middle, and heavyweight spaceships. These tagalongs were found in 1992 by (a,b,d) Hartmut Holzwart and (c) David Bell.

two other particularly useful tagalongs for the light, middle, and heavyweight spaceships, called the **Schick engine** and **Coe ship**. We will introduce and thoroughly investigate these tagalongs in Section 4.4.

Just like we distinguished between strict still lifes and pseudo still lifes in Section 2.1, we similarly distinguish between spaceships[10] and **pseudo spaceships**, which are flotillae in which none of the component spaceships actually change their evolution at all, but at least one dead cell has more than 3 live neighbors in the flotilla but has fewer than 3 live neighbors when only one of the component spaceships is present. Some pseudo spaceships involving lightweight and heavyweight spaceships are displayed in Figure 4.15.

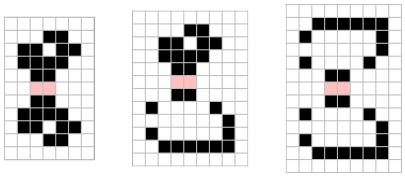


Figure 4.15: There are three ways of placing an LWSS, MWSS, and/or HWSS next to each other so as to create a pseudo spaceship. In all three of these cases, the component spaceships evolve in the exact same way as they would individually, yet in some of their phases the flotilla overpopulates some cells (highlighted in red) that the spaceships individually do not.

It is also possible for the sparks of two light, middle, and heavyweight spaceships to interact with each other as they move, similar to how we used the sparks of two oscillators to interact with each other in Section 3.3. Objects created from multiple smaller interacting spaceships like this are called **flotillae**, and some examples involving a lightweight spaceship and a middleweight spaceship are presented in Figure 4.16. Since flotillae can be made up of multiple copies of any of these three standard $c/2$ orthogonal spaceships–LWSS, MWSS, or HWSS–it is often convenient to refer to any of these components as an **xWSS**. This term comes from the idea of "x" being an unknown, which can be replaced by one of the other letters, "L", "M", or "H".

Although there are numerous[11] ways of creating flotillae, they are a bit less satisfying than what we got when we combined oscillator sparks in Section 3.3. The main reason these flotillae do not really get us too much that is genuinely "new" is that each of the light, middle, and heavyweight spaceships have the same period and speed, so every flotilla constructed from them will also have the same period and speed.

One way to construct flotillae that are a bit less trivial is to consider what might happen if we

[10]The term "strict spaceship" is not used in practice, and if the term "spaceship" is used unqualified then it is typically assumed that it cannot be broken down into smaller spaceships.

[11]To be explicit, the number of flotillae involving two xWSSes are: 1 LWSS on LWSS (pseudo), 3 LWSS on MWSS, 8 LWSS on HWSS (1 is pseudo), 5 MWSS on MWSS, 15 MWSS on HWSS, and 10 HWSS on HWSS (1 is pseudo).

Free PDF. See conwaylife.com/book for pattern files and print version.

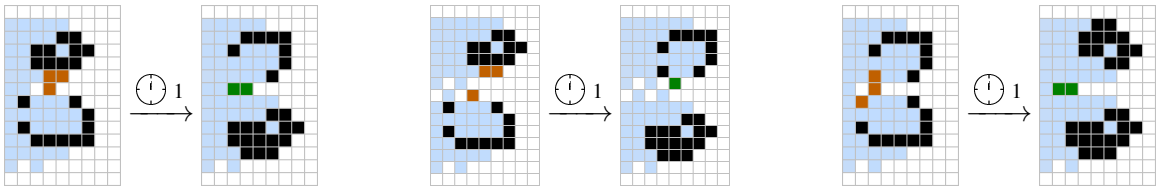


Figure 4.16: There are three ways of placing an LWSS and an MWSS next to each other in order to create a flotilla. The orange sparks interact in such a way as to give birth to the green cells that would not be born in either ship individually.

were to construct a spaceship that followed the same pattern as the light, middle, and heavyweight spaceships, but is even longer than the heavyweight spaceship, such as the one displayed in Figure 4.17. This object is called an **overweight spaceship** (or **OWSS** for short), but its name is deceiving—it is not actually a spaceship at all. The reason for this is that the 3-cell "spark" that it emits is not actually a spark, as it survives to subsequent generations and leads to the OWSS's destruction.

However, we can use sparks from light, middle, and heavyweight spaceships to prevent those three cells from being born in the first place, thus creating flotillae involving overweight spaceships, as in Figure 4.18. That is, we can use two light, middle, and/or heavyweight spaceships to turn an overweight spaceship into an object that is *actually* a spaceship. An overweight spaceship can thus be thought of as a tagalong for the light, middle, and heavyweight spaceships, and in fact is one of the most versatile of known tagalongs. Overweight spaceships of any length can be stabilized by using an appropriate arrangement of smaller spaceships along their sides. It is even possible to stabilize a large overweight spaceship by a smaller overweight spaceship, which is then stabilized by a true spaceship, as long as the outermost layer of this flotilla consists of true light, middle, and/or heavyweight spaceships (see Exercises 4.9 and 4.10).

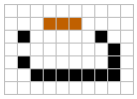


Figure 4.17: An **overweight spaceship**, which is not actually a spaceship since the three orange cells do not form a spark (i.e., they do not die) and they thus interfere with its evolution.

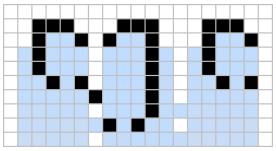


Figure 4.18: A flotilla that uses two lightweight spaceships to suppress the 3-cell "spark" of the OWSS, thus creating a new spaceship.

4.3 Corderships

Up to this point, we have not seen any spaceships that travel at a speed other than that of the "standard" spaceships—$c/4$ diagonally (e.g., the glider, crab, or orion 2) or $c/2$ orthogonally (e.g., xWSSes, the sidecar, or hivenudger). Our first foray into the realm of other speeds is via the switch engine, which is the chaotic object that we introduced in Section 1.5 that travels at a speed of $c/12$ diagonally.

We already saw several methods for using switch engines to stabilize each other so as to create puffers that left behind predictable debris (recall that a puffer created in this way was called an ark). However, using switch engines to stabilize each other and erase their debris entirely (thus creating a spaceship) is much more difficult.[12] Spaceships constructed in this way are called **Corderships**,[13]

[12]The first ark was found in 1971, whereas the first spaceship based on switch engines was not found until 1991.

[13]Named after Charles Corderman, who discovered the switch engine and most of the simple puffers based on it in 1971.

Free PDF. See conwaylife.com/book for pattern files and print version.

and the main ingredient in the construction of most of them is the reaction between two switch engines displayed in Figure 4.19.

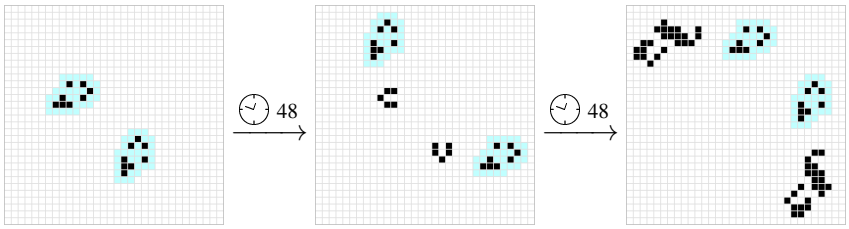


Figure 4.19: Two switch engines (highlighted in aqua) that *almost* stabilize each other. After 48 generations, they reappear but farther apart and with a couple of inconsequential 5-cell sparks. After the next 48 generations they return to their original relative positions, but along with some troublesome debris that leads to their destruction.

In this reaction, the debris from two switch engines causes overcrowding that destroys both sets of debris, while leaving both switch engines intact. However, this only works for the first 48 generations of the switch engines' evolution (where the switch engines start close to each other and move farther apart). For the next 48 generations (where the switch engines start far apart and move closer together), the pieces of debris from the switch engines are too far apart from each other to interact, and thus survive to cause problems later on.

One potential way of fixing this problem (i.e., suppressing the debris that forms between generations 48 and 96) would be to place even more switch engines next to each other, so that each switch engine alternates between which of its neighbors it uses to suppress its debris every 48 generations, as in Figure 4.20.

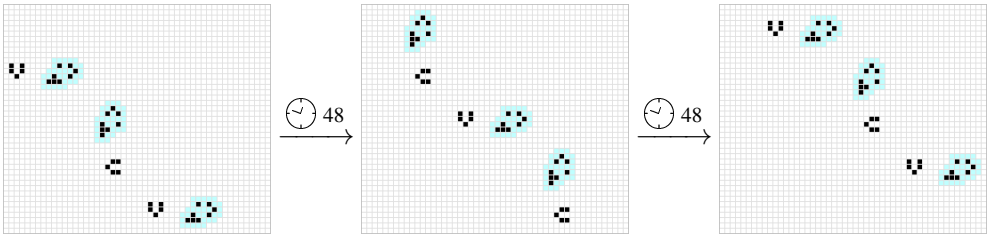


Figure 4.20: An infinitely long wave of switch engines (highlighted in aqua) forever bounce off of each other and suppress each other's debris, creating an object that moves at a speed of $c/12$ diagonally.

The problem with this technique is that the object it creates must be infinitely long to actually be stable, which we don't want—we are only interested in spaceships of finite size. One method of stabilizing the edges of this pattern is to observe that the debris created by the switch engines on the far edges of this arrangement (i.e., the debris that causes problems and eventually leads to the destruction of the switch engines) temporarily creates a block, as shown in Figure 4.21. If we could destroy the remaining debris sometime between when the block is created and when the debris destroys the pattern, we would then have a very orderly puffer that creates a single-file trail of blocks on both of its ends.

While a clean puffer like this isn't what we were originally looking for, it would get us almost all the way to a spaceship, since it turns out that this exact same spacing of blocks can be used to destroy the debris left behind by these edge switch engines (see Figure 4.22). By putting these two facts together, we now have a scheme for how we could construct a Cordership:

1) The front of the Cordership will be made up of a row of switch engines leaving behind two trails of blocks,

Free PDF. See conwaylife.com/book for pattern files and print version.

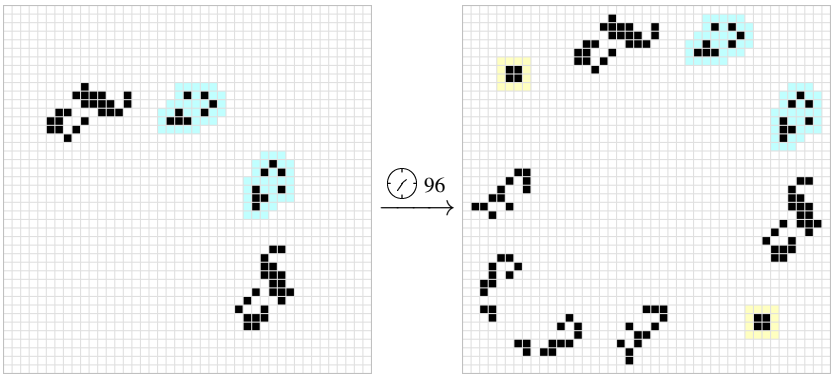


Figure 4.21: The debris left behind by each outermost switch engine temporarily creates a block (highlighted in yellow). This block is subsequently destroyed by the debris at the rear.

2) In the middle of the Cordership will be some switch engines destroying the left over debris of the front switch engines, and
3) At the back of the Cordership will be another row of switch engines, which destroys and is stabilized by the two trails of blocks.

There are many different ways to put these steps together, with perhaps the simplest being the completed Cordership displayed in Figure 4.23. This ship uses 4 switch engines in the front row to create the trails of blocks, 2 switch engines in the middle to clean up some debris, and 4 switch engines in the back to follow and destroy the trails of blocks.[14] This ship is called the **10-engine Cordership**, based on the fact that it uses 10 switch engines.[15]

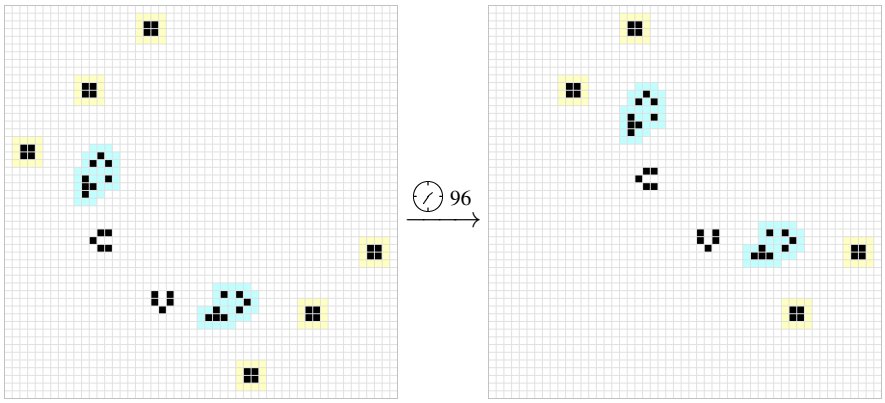


Figure 4.22: A trail of blocks (highlighted in yellow) can destroy and be destroyed by the debris left behind by a switch engine.

There are numerous different ways to put together Corderships, but most of the large (and somewhat out of date) Corderships have the same basic structure: some switch engines at the front leave behind some debris that is cleaned up by, and stabilizes, some switch engines at the back. Another reaction that can be used at the front of a Cordership is investigated in Exercise 4.15, and another reaction that can be used at its rear is presented in Exercise 4.16. While the reactions that we

[14] It might seem desirable to just use 2 switch engines at the front and back, as in Figures 4.21 and 4.22, but then there would not be enough room to place a switch engine in the middle to tame the debris of the front switch engines. However, it is possible to use just 3 switch engines in the front and back (see Exercise 4.15).

[15] The first ever Cordership, which used 13 switch engines, was constructed by Dean Hickerson in April 1991. He also built the smaller 10-engine Cordership seen here by no later than April 1992.

Free PDF. See conwaylife.com/book for pattern files and print version.

have seen all lead to somewhat large Corderships, some particularly clever Corderships are known that use as few as 2 switch engines (see Exercises 4.18 and 4.20).[16]

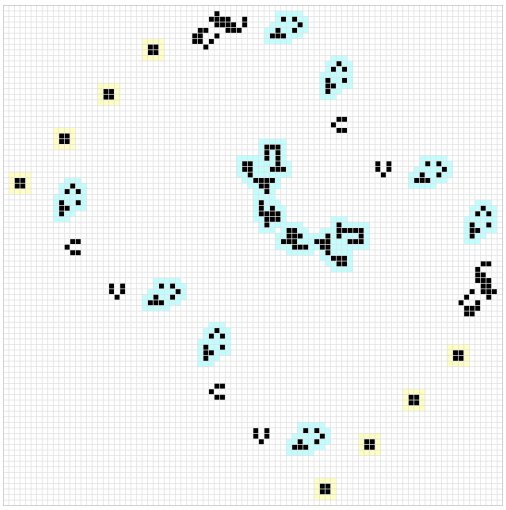


Figure 4.23: A **10-engine Cordership**, which is a $c/12$ diagonal spaceship with period 96. In the orientation depicted here, it travels to the top-right, with the 4 switch engines at the front (highlighted in aqua) laying tracks of blocks (highlighted in yellow). The 2 central switch engines (which are out of phase from the other switch engines and thus look unusual) destroy the leftover debris from the front switch engines, and the 4 rear switch engines destroy and are stabilized by the trail of blocks.

One of the most useful features of Corderships is the collection of pulsating sparks that are produced by the rear row of switch engines, which can interact with other objects as the ship moves. For example, Figure 4.24 presents 2 ways in which the 10-engine Cordership can reflect a glider 90 degrees, a way of using it to reflect a glider 180 degrees, and a method of turning a glider into an LWSS. Furthermore, by just changing the back end of the Cordership slightly, we can get a completely new set of sparks to work with, which allow for an even wider set of reactions (see Exercise 4.16).

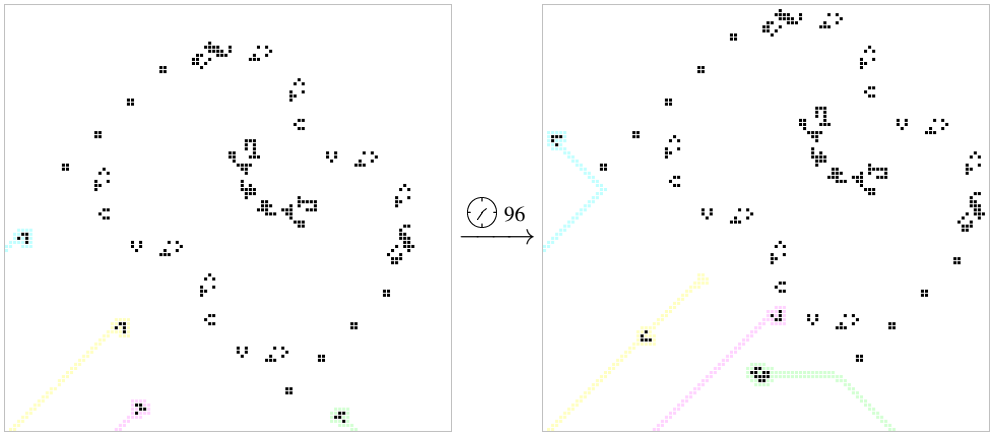


Figure 4.24: The pulsating sparks at the back of the 10-engine Cordership from Figure 4.23 can be used to reflect a glider by 90 degrees (highlighted in aqua and magenta), reflect a glider by 180 degrees (highlighted in yellow), and turn a glider into an LWSS (highlighted in green).

[16] Paul Tooke ran computer searches in 2004 that tested hundreds of thousands of ways of colliding 2 switch engines, and none were found that produce a spaceship. It wasn't until December 2017 that user "praosylen" on the ConwayLife.com forums found a working 2-engine Cordership.

Free PDF. See conwaylife.com/book for pattern files and print version.

4.4 Puffers and Rakes

Recall from Section 1.3 that we can use the Gosper glider gun to create an endless stream of gliders starting from a fixed location. While this is certainly a useful feature, there are times when we want something that acts like a gun (i.e., something that creates an endless stream of gliders) but is itself moving as well. In other words, we want to construct a **rake**—a spaceship that creates additional spaceships as it travels. We break down the creation of such an object into two steps:

1) First, we construct a spaceship that leaves debris behind itself as it moves—we recall from Section 1.5 that objects with this property are called **puffers**. The only puffers that we have seen so far are based on the switch engine, but it should seem believable that $c/2$ orthogonal puffers exist too, since the light, middle, and (especially) heavyweight spaceships have such strong sparks that they should be able to interact in such a way as to leave debris behind that does not destroy the spaceships themselves.
2) Second, we use additional light, middle, or heavyweight spaceships to transform the debris from step (1) into a glider. Again, the reason we expect this to work is that we have a lot of freedom with how we can make one of these spaceships interact with other objects, due to the variety of sparks that they emit.

4.4.1 The Space Rake

To make step (1) above explicit, we take inspiration from how we constructed switch engine-based puffers in Section 1.5: we place additional objects near some chaotic object that is *almost* stable so as to tame its debris enough that it doesn't self-destruct. This time, we use a B-heptomino instead of a switch engine, since we recall from Figure 1.19 that it creates some debris and moves forward by 5 cells in 10 generations.[17] Since the B-heptomino moves orthogonally at a speed of $c/2$, it seems reasonable to try to stabilize it by light/middle/heavyweight spaceships.

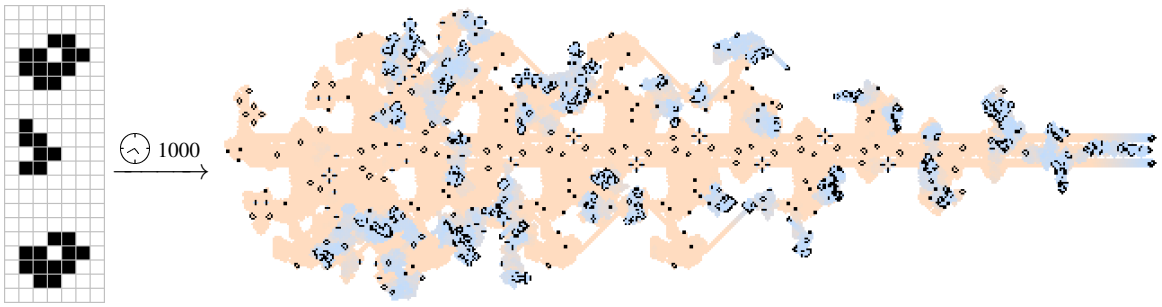


Figure 4.25: A puffer composed of a B-heptomino that has been stabilized by two lightweight spaceships. The debris in the image on the right is extremely chaotic, taking more than 5000 generations to settle down, but never interferes with the B-heptomino or the lightweight spaceships.

One way of taming the B-heptomino's debris is to use a lightweight spaceship on either side of it—the single-cell spark on its back end is just strong enough to overpopulate the interfering portion of the debris behind the B-heptomino, thus stabilizing it as in Figure 4.25.[18] The debris left behind this puffer is extremely chaotic, taking a whopping 5532 generations to stabilize. However, after that

[17] It is not too surprising that the B-heptomino can be made to move at a speed of $c/2$ orthogonally when suitably stabilized—after all, in 2 out of their 4 phases, the front 3 columns in the light/middle/heavyweight spaceships themselves are exactly a B-heptomino.

[18] This puffer was found by Bill Gosper sometime in the early 1970's. It does not have a standard name, but is sometimes referred to simply as "puffer 2", since it was the second puffer to be found.

Free PDF. See conwaylife.com/book for pattern files and print version.

point it becomes periodic with period 140, and we can indeed see that it never interferes with the puffer itself.

Now that we have a puffer to work with, we turn to task (2) outlined earlier: we use the sparks from light, middle, and heavyweight spaceships to tame the puffer debris and turn it into something useful like a glider. Even just by placing these spaceships near the debris by hand in a few different locations and phases, it does not take long to find interesting combinations. For example, if we place an extra lightweight spaceship as in Figure 4.26, the debris hits its spark in such a way as to die off completely, thus resulting in a period 20 spaceship called the **ecologist**.

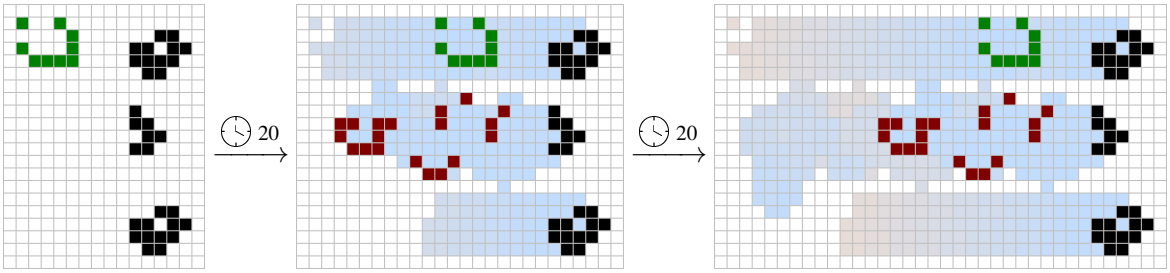


Figure 4.26: If we add an extra lightweight spaceship (displayed in green) to the puffer, its debris is suppressed, resulting in a spaceship called the **ecologist**. Even though the debris (displayed in red) dies off completely, it becomes somewhat large before doing so.

The dying debris that trails behind the ecologist actually becomes somewhat large, and it moves off to the side of the ecologist opposite the lightweight spaceship. In other words, the ecologist has an extremely large and accessible spark that trails behind it, and we can hit this spark with even more spaceships in order to change it into something more useful. This time, we finally hit the jackpot: if we hit the debris with a lightweight spaceship in just the right spot, it is transformed into a glider that travels toward the northeast. Furthermore, if we move that lightweight spaceship slightly, the debris is instead transformed into a glider that travels toward the southwest. We have thus succeeded in creating *two* rakes: one in which the gliders travel forward along with the rake itself (see Figure 4.27), and one in which the gliders travel backward away from the rake (see Figure 4.28). These are called the **forward** and **backward space rake**, respectively.

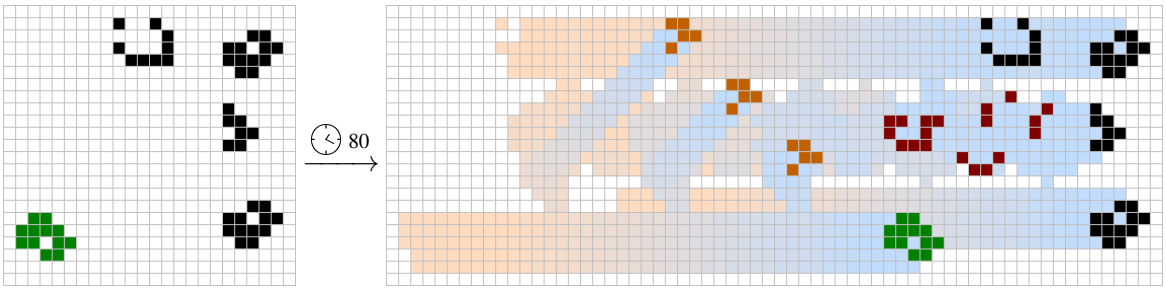


Figure 4.27: The (forward) **space rake** creates a forward-moving glider every 20 generations. It is constructed by adding yet another lightweight spaceship (displayed in green) to the ecologist in such a way as to transform its large spark into a glider.

4.4.2 The Schick Engine

While space rakes are extremely useful due to the fact that we can use them to fire gliders in any direction that we like as they travel, one of their drawbacks is that they actually fire *too many* gliders to be useful in some circumstances. Specifically, they fire one glider every 20 generations, so they

Free PDF. See conwaylife.com/book for pattern files and print version.

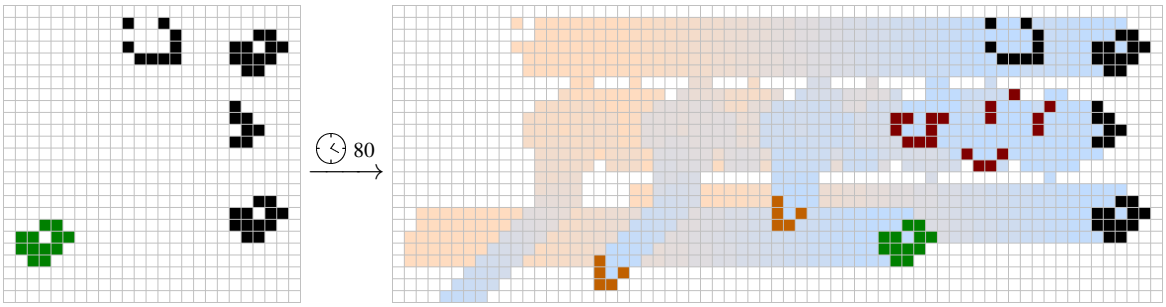


Figure 4.28: The **backward space rake** creates a backward-moving glider every 20 generations, using a slightly differently positioned lightweight spaceship (displayed in green) than the forward space rake.

have a horizontal distance of 10 cells between them. However, many of the objects that we will want to construct with rakes are more than 10 cells wide, so it will be useful for us to have a way of thinning out these gliders. Our method for doing this is the **Schick engine**: another spaceship that, just like the ecologist, consists of some dying junk trailing behind some lightweight spaceships (see Figure 4.29).[19]

There are two key features that make the Schick engine useful for us:

1) It has period 12 instead of period 20, so its debris could potentially be used to interact with only *some* of the gliders emitted by the space rake rather than all of them.
2) Its trailing spark "pulsates"—it sticks out quite far in some generations, but then retracts back during other generations. It thus seems believable that we could line things up so that some gliders coming from the space rake hit the Schick engine's spark, while others pass by it completely unharmed.

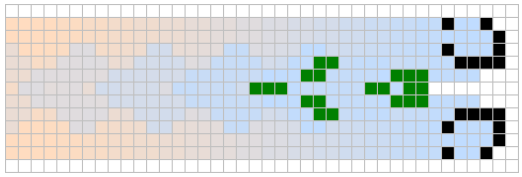


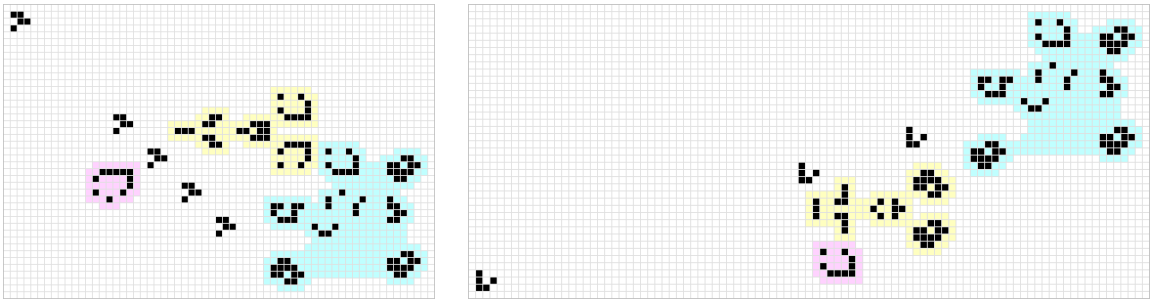
Figure 4.29: The **Schick engine** is a period 12 spaceship that consists of a pulsating tagalong (displayed in green) trailing behind two lightweight spaceships.

Indeed, it only takes a little bit of experimentation to find almost exactly what we want: if we line the forward space rake and the Schick engine up as in Figure 4.30(a), one third of the gliders are cleanly destroyed, one third of the gliders are left untouched, and one third of the gliders are turned into blocks. In order to destroy those blocks (thus completely eliminating two out of every three gliders from the space rake), we can simply use a middleweight spaceship, as in the block-destroying reaction from Figure 4.12. Putting this all together gives us the forward rake in Figure 4.30(a) that emits one glider every 60 generations (and since its speed is $c/2$, the horizontal distance between gliders is 30 cells).

A very similar game can be played with the backward space rake: if we place a Schick engine as in Figure 4.30(b), then one third of the gliders from the backward space rake are destroyed, one third are left untouched, and one third explode into a chaotic mess. If we add an additional lightweight spaceship, that chaotic mess can also be destroyed, resulting in a backward rake that emits one glider every 60 generations (and hence 30 horizontal cells).

[19]The Schick engine was found by Paul Schick in 1972, and it can be re-discovered just by experimenting with placing different small objects behind two lightweight spaceships (see Exercise 4.22).

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) A forward p60 rake. **(b)** A backward p60 rake.

Figure 4.30: Rakes that emit one glider every 60 generations, based on the (a) forward and (b) backward space rakes (outlined in aqua) that emit one glider every 20 generations. A Schick engine (outlined in yellow) is positioned so that it destroys 1/3 of the gliders that pass by, leaves 1/3 of the gliders untouched, and turns the remaining 1/3 of the gliders into other objects (either (a) a block or (b) a chaotic mess) that is cleaned up by the spaceship outlined in magenta.

4.4.3 The Coe Ship

We can build another family of $c/2$ puffers and rakes by using an object called the **Coe ship**,[20] which is the $c/2$ spaceship displayed in Figure 4.31. Just like the Schick engine, it has a large trailing spark that pulsates throughout its period, making it very useful for interacting with other moving objects. The advantage of having this additional spaceship at our disposal is that it has period 16 (versus the space rake's period of 20 and the Schick engine's period of 12), and can thus interact with the rakes we have already created in non-trivial ways.

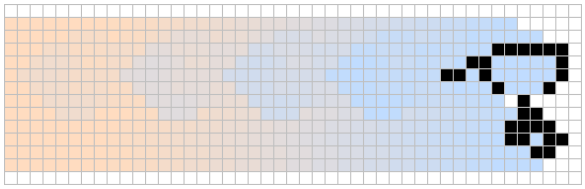


Figure 4.31: The **Coe ship** is a $c/2$ orthogonal period 16 spaceship that consists of some pulsating debris trailing behind a lightweight spaceship and a deformed heavyweight spaceship.

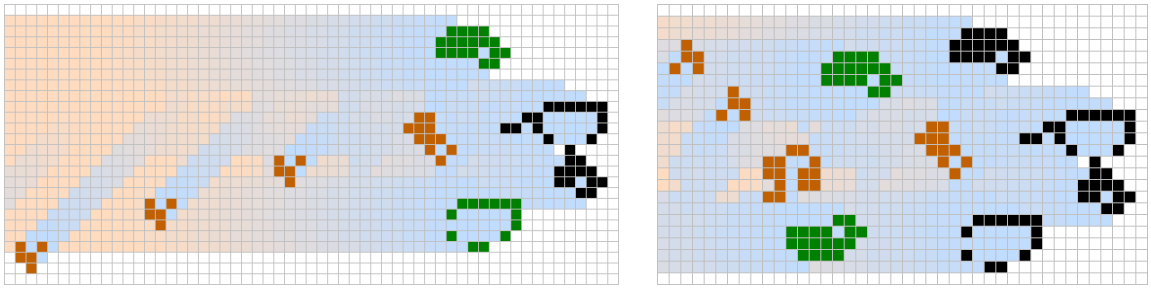
Most notably, we can add some xWSSes behind the Coe ship in order to cause its spark to spawn an endless wave of gliders, just like we did when we created the space rake from the ecologist. In particular, by placing two heavyweight spaceships as in Figure 4.32(a), we can create a period 16 backward rake. To turn this backward rake into a forward rake, we can place two additional heavyweight spaceships in such a way that they reflect the backward-moving glider so that it becomes a forward-moving glider, as in Figure 4.32(b).[21]

Now that we have rakes of multiple different periods (16, 20, and 60), we can strategically combine their glider waves in order to create rakes of even more periods. For example, if we place a backward space rake next to a backward Coe rake so that their glider streams cross each other as in Figure 4.33(a), then every $\text{lcm}(16,20) = 80$ generations 9 gliders are produced (4 from the space rake and 5 from the Coe rake). Of these 9 gliders, 4 collide with each other and die completely, 2 collide and create a single forward-moving glider, and 3 simply avoid all of the other gliders and thus

[20] Named after Tim Coe, who found it in 1995.

[21] This configuration of two heavyweight spaceships works to turn any $c/2$ backward rake with period a multiple of 4 and at least 16 into a forward rake. For example, this gives us another way to turn the period 20 and period 60 backward space rakes into forward space rakes (see Exercise 4.23).

Free PDF. See conwaylife.com/book for pattern files and print version.



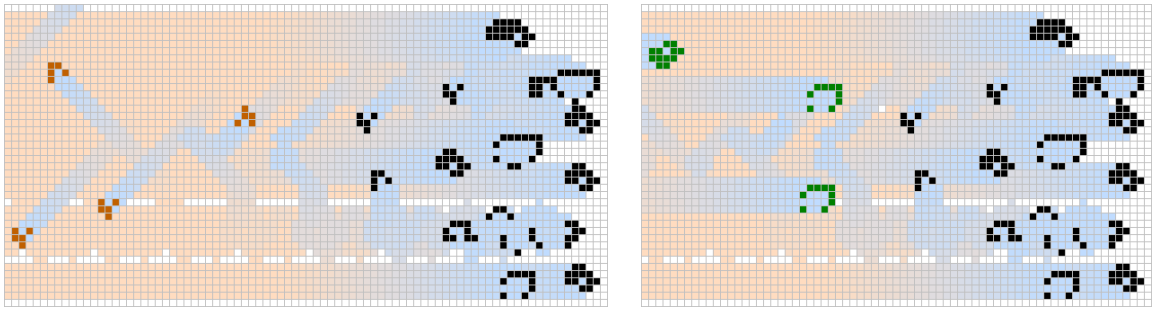
(a) A backward p16 rake. **(b)** A forward p16 rake.

Figure 4.32: Period 16 (a) backward and (b) forward rakes constructed by using heavyweight spaceships (displayed in green) to interact with the spark behind a Coe ship.

continue travelling backward (for a total of 4 surviving gliders produced every 80 generations).

We can then erase some (or all) of these glider waves by using a lightweight spaceship as in Figure 4.12, thus creating forward or backward rakes of period 80. One particular placement of three lightweight spaceships that erase all of the output gliders is shown in Figure 4.33(b)—a period 80 forward rake can be created by removing the top-right of the green lightweight spaceships, and a period 80 backward rake can be created by removing the top-left of the green lightweight spaceships.

We can also repeat this exact same procedure with a period 60 space rake rather than the period 20 version, and thus create forward and backward rakes with period $\text{lcm}(16, 60) = 240$ (see Exercise 4.24), but this is the highest period rake that can be constructed using these techniques. A method for creating even higher-period rakes (and in fact rakes with arbitrarily high period) is presented in Section 4.6.2.



(a) A forward and backward p80 rake. **(b)** A stabilization of the rake.

Figure 4.33: When a backward space rake and a backward Coe rake are carefully placed next to each other, their streams cross in such a way that they produce (a) 3 backward gliders and 1 forward glider every 80 generations. The (b) lightweight spaceships displayed in green destroy those 4 output gliders. Removing the top-right green LWSS results in a period 80 forward rake, whereas removing the top-left green LWSS results in a period 80 backward rake.

4.5 Speed Limits

We now consider the problem of determining which speeds spaceships can attain. We have seen numerous examples of diagonal spaceships that move at $c/4$ (with the glider being the prototypical example) and also several orthogonal spaceship that move at $c/2$ (such as xWSSes). We have also seen a few slower spaceships, such as the diagonal $c/12$ Corderships. However, we have not seen any spaceships that are faster than the "basic" spaceships that we are already familiar with. The following

Free PDF. See conwaylife.com/book for pattern files and print version.

theorem shows that there is a reason for this: no faster spaceships exist.[22]

Theorem 4.1 — Spaceship Speed Limits
The maximum diagonal and orthogonal speeds that a finite object (e.g., a spaceship, puffer, or rake) can travel through empty space are $c/4$ and $c/2$, respectively.

Proof. We begin by proving the $c/4$ speed limit for diagonal spaceships. Consider the grid of cells given in Figure 4.34. If the spaceship is contained within the region of light gray cells in generation 0, then suppose (in order to establish a contradiction) that cell X is alive in generation 2.

If cell X is alive in generation 2, then cells A, B, and C must be alive in generation 1. It follows that cells A and C must have 3 live neighbors in generation 0, so each of K, L, M, N, and B must be alive in generation 0. However, this implies that cell B must have at least four live neighbors in generation 0, so there is no way for it to survive to generation 1, which gives the desired contradiction.



Figure 4.34: A diagram that illustrates Life's diagonal $c/4$ speed limit. If a pattern is contained within the light gray cells in generation 0, it must be on and below the diagonal line of dark gray cells in generation 2.

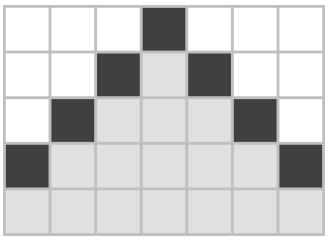


Figure 4.35: A diagram that illustrates Life's orthogonal $c/2$ speed limit. If a pattern is contained within the light gray cells in generation 0, it must be on and below the diagonal lines of dark gray cells in generation 2.

We have shown that cell X can not be alive in generation 2. In other words, if the spaceship is contained within the region of light gray cells in generation 0, then it will be on and below the diagonal line of dark gray cells in generation 2. By using this argument again, we see that the spaceship must be on and below the diagonal line containing cell X in generation 4. It follows that no spaceship (or puffer, or rake...) can travel faster than $c/4$ diagonally.

To see that the $c/2$ speed limit holds for orthogonal ships, just use two diagonal lines as in Figure 4.35. If a spaceship is contained within the region of light gray cells in generation 0, then we already showed that it must be on and below the diagonal lines defined by the dark gray cells in generation 2. It follows that it can not travel faster than $c/2$ orthogonally. ■

Since there is an upper bound on how fast spaceships can travel, it perhaps seems natural to ask whether or not there is also a lower bound—a slowest speed at which spaceships can travel. This question is a bit beyond us at this point, but in Chapter 11 we will see that such a lower bound does not exist. That is, we can construct spaceships that move as slowly as we like.

4.5.1 Wires and Signals

It is important to note that Theorem 4.1 only applies to objects travelling through a **vacuum**—a sea of dead cells. If a portion of the Life plane is filled with a repeating non-empty pattern, such as the zebra stripes from Figure 2.25(a), then an object may be able to travel through it at up to lightspeed (i.e., a

[22] This theorem was originally proved by Conway himself, very shortly after introducing the Game of Life.

Free PDF. See conwaylife.com/book for pattern files and print version.

speed of c).[23] An object that moves through a non-empty pattern like this is called a **signal**,[24] and the pattern that it is able to move through is called a **wire**. Some simple examples of lightspeed signals that can travel through a zebra stripes wire are presented in Figure 4.36.

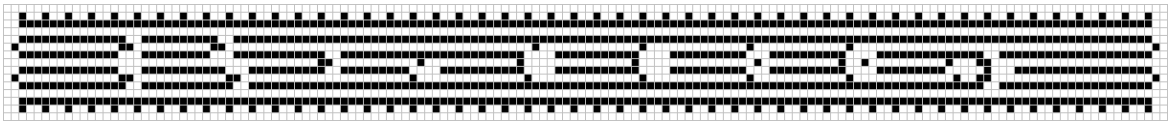


Figure 4.36: A collection of lightspeed signals that travel through zebra stripes. Each of the signals displayed here (i.e., the deformations in the middle of the stripes) travel to the right by 1 cell per generation. These signals were found (in no particular order) by Alan Hensel in 1995, Noam Elkies in 1997, Gabriel Nivasch in 1999, and in some cases, unknown Lifers in the early 1970's.

With these signals in mind, all of which travel at the speed of light, it seems natural to wonder what other signal speeds are possible. Our first result of this section shows that we will have to branch out somewhat to find slower signals, since every signal that travels through zebra stripes parallel to the stripes must do so at lightspeed.[25]

Theorem 4.2 — Zebra Stripes Parallel Speed Limit
Every finite signal that moves parallel through a zebra stripes wire travels at a speed of c.

Proof. Since the signal is finite, it has some leading edge (which we assume is moving to the right). To the right of this leading edge the stripes are regular and unbroken, but to the left of it there is at least one irregularity. Now suppose for a contradiction that there exists a signal that travels through the stripes at a speed slower than c. Then there must exist some generation (which we will call generation 0) with the property that the leading edge in generation 1 is one cell farther to the right than in generation 0, but then does not change in generation 2, as illustrated in Figure 4.37.

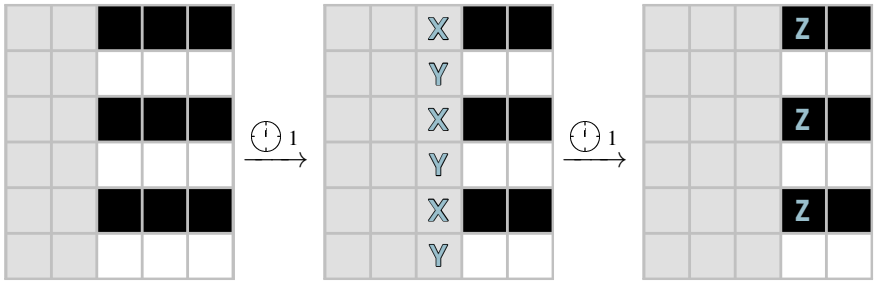


Figure 4.37: A diagram illustrating the fact that every signal that moves parallel through zebra stripes travels at a speed of c. If the signal is contained within the region of light gray cells and its leading edge ever moves to the right (denoted by the Xs and Ys in the middle generation), then it must continue moving by 1 cell every generation.

First note that all of the cells marked Y must be dead in generation 1, since they have at least 4 live neighbors in generation 0. But then each of the cells marked X must be alive in generation 1 or else the cells marked Z would be dead in generation 2 due to only having one live neighbor. We have thus shown that the leading edge of the signal in fact did not advance to the right at all between generations 0 and 1, which is the desired contradiction that completes the proof. ■

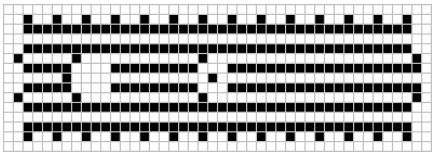
[23] No object, whether in a vacuum or not, can possibly have a speed greater than c, since in one generation it can only affect its 8 neighbors.

[24] Objects that move through a vacuum (such as gliders) are also sometimes called signals, since they can be thought of as carrying information between two locations.

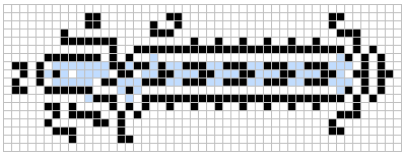
[25] This theorem was originally proved by Dean Hickerson in 1993.

Free PDF. See conwaylife.com/book for pattern files and print version.

In order to make use of signals and turn them into useful composite patterns, we need an object that can create the signal (called a **source**) and an object that can destroy the signal (called a **sink**). It will also be useful to have an object that can reflect the signal around a track (called a **signal elbow**), so that we have some flexibility in positioning it where we want it.[26] Sources and sinks for various signals have been known for quite some time, and some examples for lightspeed signals are presented in Figure 4.38. When sources and sinks are combined, like in Figure 4.38(b), the resulting pattern is a billiard table oscillator with period equal to that of the signal source.



(a) A sink (at the far right end of the zebra stripes) that cleanly destroys two different lightspeed signals.



(b) A period 5 source and sink for a lightspeed signal (which moves to the right) combine to form a period 5 oscillator. Found by Dean Hickerson in 1995.

Figure 4.38: Some sources and sinks for lightspeed signals travelling through a zebra stripes wire.

On the other hand, no reasonably small signal elbows (for any signal) are known to date, despite a considerable amount of effort on the part of Life enthusiasts.[27] The discovery of a quickly recovering elbow would be significant, since we could use it to move a signal around in a loop—much like we used reflectors to move a glider around a loop in Section 3.5—creating oscillators with any sufficiently large period. Furthermore, since signals can move so much faster than gliders, it might be possible to construct signal loops of period 19, thus putting the omniperiodicity problem of Section 3.7 to rest.

Although signals that travel parallel to the stripes in zebra stripes are the most common type, there are also signals that travel in the perpendicular direction, such as those displayed in Figure 4.39. However, these signals are somewhat less useful than their parallel counterparts, since (a) all known perpendicular signals are rather large compared to the known parallel signals, and (b) they cannot travel at lightspeed, as shown by the following theorem.[28]

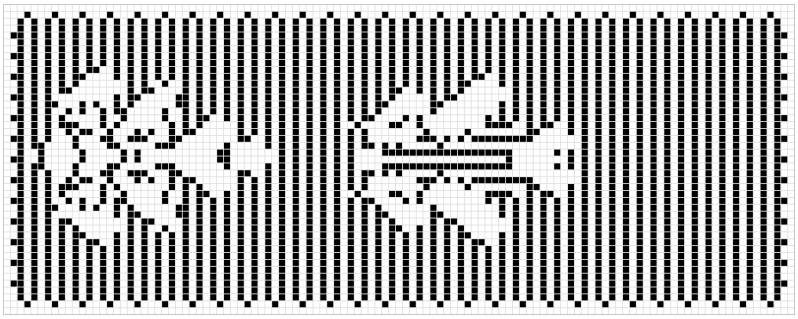


Figure 4.39: Some signals that travel perpendicularly through zebra stripes at a speed of $2c/3$ (to the right). Both of these signals were found by Hartmut Holzwart in 2006.

[26] Just like signals are analogous to spaceships, there is also an analogy between sources and glider guns, sinks and eaters, and signal elbows and reflectors.

[27] Very large signal elbows are known that work by converting signals into things like Herschels, which are moved around tracks and then re-converted into signals, but they are all very slow and have a very large repeat time.

[28] This theorem was originally proved by Hartmut Holzwart in 2006.

Free PDF. See conwaylife.com/book for pattern files and print version.

Theorem 4.3 — Zebra Stripes Perpendicular Speed Limit
The maximum speed at which a finite signal can travel perpendicularly through zebra stripes is $2c/3$.

Proof. The proof of this theorem is similar in style to those of Theorems 4.1 and 4.2. Consider the grid of cells given in Figure 4.40. If the signal is contained within the region of light gray cells in generation 0, then we first show that cells marked with an X will still be alive and cells marked with a Y will still be dead in generation 1.

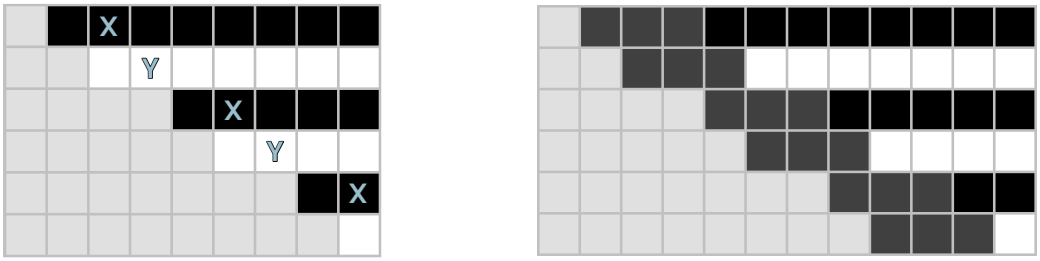


Figure 4.40: A diagram illustrating the $2c/3$ speed limit for a finite signal travelling perpendicularly through zebra stripes. If a signal is contained within the light gray cells in generation 0, then cells X and Y (left) cannot change state in generation 1, so it must be on and to the left of the dark gray cells (right) in generation 3.

To see that the cells marked with an X must still be alive generation 1, simply observe that they have either 2 or 3 live neighbors in generation 0: the cells to their immediate left and right, and possibly the light gray cell to their bottom-left. To see that the cells marked with a Y must still be dead in generation 1, we note that regardless of the state of the light gray cells, they have at least 4 live neighbors (the 3 cells above them and the cell to their bottom-right).

By using this fact 3 times, we see that any object that is contained within the light gray region in Figure 4.40 in generation 0 will be located on and to the left of the dark gray cells in generation 3. Since the region containing the dark gray cells is just the original light gray region shifted up by 2 cells, it follows that the signal cannot travel more than 2 cells perpendicular to the stripes every 3 generations. In other words, its speed is no greater than $2c/3$. ■

While we have an upper bound on the speed of signals travelling perpendicular to stripes in a zebra stripes wire, there is still no known lower bound on their speed. No such signal that travels at a speed slower than $2c/3$ is known, but no proof that signals must travel this fast is known either. In particular, whether or not there exist perpendicular $c/2$ signals has been an open question since 2006.[29]

All of the signals that we have seen so far travel orthogonally, but there are also diagonal signals that move through wires that are a bit more complicated than stripes. Some examples of diagonal signals travelling through diagonal wires at various speeds (specifically $2c/3$, $5c/9$, and $c/2$) are presented in Figure 4.41, along with sinks that absorb the $2c/3$ and $5c/9$ signals. The $2c/3$ diagonal signal is the fastest one known to date—there are currently no known diagonal signals that travel through a stable (p1) wire at the speed of light, without destroying the wire.

These diagonal signals are quite exciting for the fact that we "almost" know how to turn them around a corner (and recall that if we could do this, we could likely construct oscillators for all of the

[29]The existence of such a signal probably would not be particularly useful, as we usually prefer faster signals to slower ones, but it would nonetheless be nice to have an answer one way or the other, since we already know about lower speed limits in a vacuum and for signals travelling parallel through stripes.

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) A $2c/3$ signal. **(b)** A $5c/9$ signal. **(c)** A $c/2$ signal.

Figure 4.41: Some diagonal signals (travelling to the bottom-right) that were found by (a,b) Dean Hickerson in 1997 and (c) Hartmut Holzwart in 2003. The signals themselves are sometimes difficult to distinguish from the surrounding wire, so they are highlighted in orange. Note that both halves of the $c/2$ signal are indeed needed, since the wire is offset by 1 cell between the two half signals.

currently unknown periods). For example, Figure 4.42 demonstrates a corner that is able to reflect the $2c/3$ diagonal signal by 90 degrees, but has the unfortunate side-effect of duplicating the signal as it is reflected. Since we do not know how to reflect the duplicated signal, we cannot use this corner more than once and cannot create a closed loop with it.

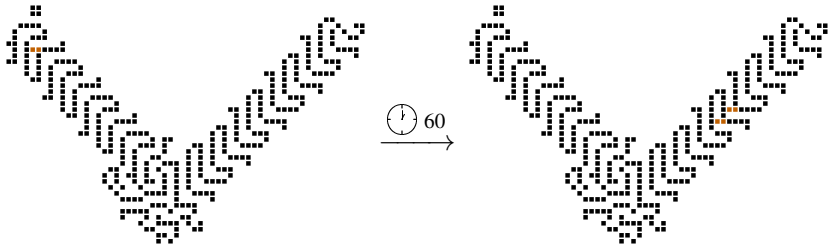


Figure 4.42: A corner that is able to reflect the $2c/3$ diagonal signal (highlighted in orange) by 90-degrees, but which also creates a second copy of that signal in the process and thus cannot be used twice.

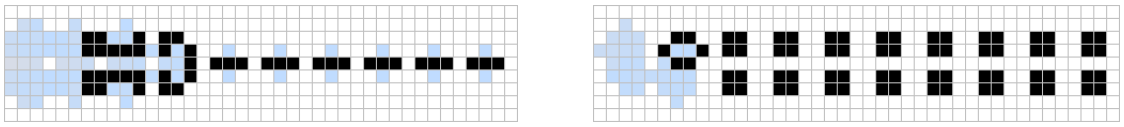
4.5.2 Fuses and Wicks

While signals pass through wires in such a way as to leave the wires undamaged, it is also possible for objects to pass through wires and destroy the wire in the process. When this happens, the wire is instead called a **wick**, and the objects that "burns" through the wick is called a **fuse**. Wicks and fuses are significantly easier to find than signals, since we can often just place random debris near a regular repeating pattern to make it burn. For example, the fuse displayed in Figure 4.43(a) can be rediscovered by hand in less than a minute just by placing random configurations of alive cells near the row of blinkers.

Fuses that leave nothing behind as they burn are said to burn *cleanly* and are typically much more useful than their dirty counterparts. Two particularly frequently used clean fuses, which both travel orthogonally at a speed of $2c/3$, are presented in Figure 4.43. Another one that travels slightly faster at $4c/5$ is presented in Exercise 4.28.

It is perhaps worth noting that the bi-block fuse from Figure 4.43(b) uses the same configuration of a block and beehive as in Figure 1.16, where the block destroys the beehive. However, the presence of the second block in the bi-block changes the reaction so that *both* the beehive and first block are destroyed, while the second block is transformed into another beehive, thus letting the process repeat.

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) A **blinker fuse**. **(b)** A **bi-block fuse**.

Figure 4.43: Two $2c/3$ orthogonal fuses with (a) period 18 and (b) period 12.

4.5.3 Teleportation

Much like we can use still lifes and oscillators in order to speed up information transmission beyond the spaceship speed limits of Theorem 4.1, we can also use chaotic reactions and spaceship collisions. For example, the collision of 3 gliders displayed in Figure 4.44 (called the **fast forward force field**[30]) has the remarkable property that if the lightweight spaceship is not present, the gliders simply destroy each other and leave nothing behind, but if the lightweight spaceship is present, then a copy of it reappears 6 generations later, 11 cells in front of its original position.

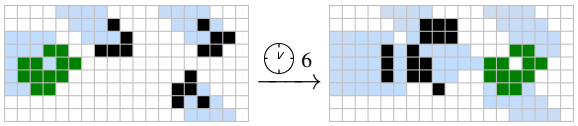


Figure 4.44: The **fast forward force field**: if the lightweight spaceship (displayed in green) is not present, the 3 gliders destroy each other and leave nothing behind, but in the configuration displayed here they "teleport" the LWSS to the right by 11 cells in just 6 generations.

Not only does this glider collision help move an LWSS at faster than the $c/2$ speed limit, but it seems to move it even faster than the speed of light! Since we know that no information can propagate through the Life plane at a speed exceeding c, it is worth investigating this reaction in a bit more detail. To this end, a generation-by-generation breakdown of how this reaction works when the LWSS is and is not present is provided in Figure 4.45.

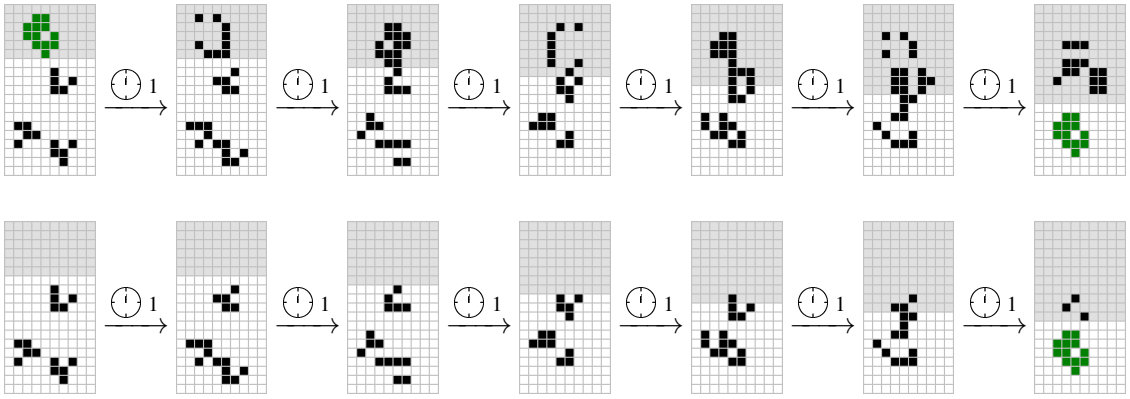


Figure 4.45: The fast forward force field with an incoming LWSS (top row) and without an incoming LWSS (bottom row). The lowest row of cells affected by the LWSS is displayed with a light gray background, and this leading row never progresses more than 1 cell per generation, showing that this reaction does not violate the lightspeed speed limit. The 3-cell spark at the bottom-right subsequently destroys the LWSS, whereas the larger spark at the top-right dies without touching the LWSS.

[30] Found by Dietrich Leithner in 1994. The "forward" in the name of the fast forward force field officially refers to the science fiction writer Robert L. Forward. However, "fast Forward force field" looks a bit too strange for the authors' tastes, so we opt not to capitalize it.

Free PDF. See conwaylife.com/book for pattern files and print version.

This breakdown reveals that the LWSS itself is not actually transmitted through the glider collision, but rather the glider collision produces an LWSS as its output regardless, and the presence of an input LWSS just determines whether or not a spark forms that subsequently destroys the output LWSS. We thus conclude that even though this reaction does speed up the LWSS past a speed of $c/2$, it does not actually speed it up past a speed of c. Indeed, the reaction is not actually done after 6 generations, since at that point we could still not use the output LWSS as a signal, since it is present at that generation regardless of whether or not the input LWSS was present. Instead, we would have to wait another 18 or so generations for the front of the LWSS in the bottom row of Figure 4.45 to be destroyed, by which time it would have travelled a total of 20 cells in 24 generations, for a total speed of $20c/24 = 5c/6$.

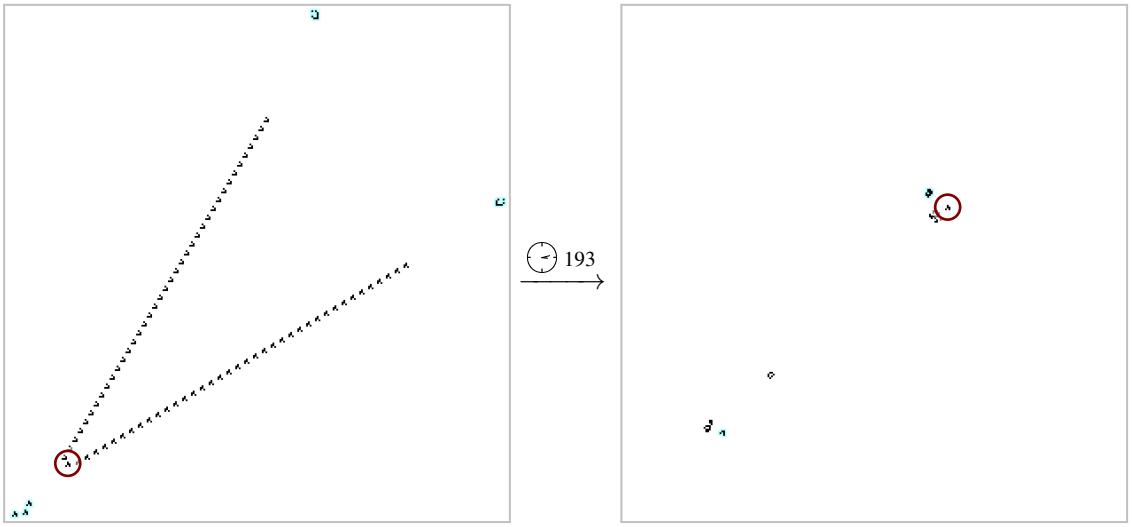


Figure 4.46: A diagonal collision of gliders, found by Jason Summers in 1999, that is able to teleport a glider from the bottom left (circled in red) to the top right at the speed of light. If the input glider is not present, the reaction just destroys itself, leaving nothing behind. The gliders and lightweight spaceships outlined in aqua are just there to clean up some leftover debris.

Another reaction that has a very similar flavor is the long glider collision displayed in Figure 4.46. This reaction teleports a single glider a distance of about 150 cells to the top right over the course of 193 generations—much faster than the usual $c/4$ diagonal spaceship speed limit. In fact, the input glider travels through the diagonal glider collision at a speed of exactly c, but it takes a few generations for the reaction to get going at the start and for it to calm down at the end.

Although this reaction fills in a big gap that has been missing from our collection of Life circuitry—recall that up until now we had no way of transmitting information diagonally at the speed of light—actually making use of it is somewhat tricky, since it is difficult to generate waves of gliders that are so closely spaced. We will discuss some methods for overcoming this obstacle in Chapters 6 and 7. However, even with the currently best-known methods, a pattern that is able to generate this configuration of gliders and thus make use of this diagonal lightspeed reaction would be extremely large.

Free PDF. See conwaylife.com/book for pattern files and print version.

4.6 Speed and Period Status

So far we have only seen spaceships with a very select few different speeds—specifically $c/2$ orthogonal, $c/4$ diagonal, and $c/12$ diagonal. Similarly, we have only seen a dozen or so different spaceship periods, all of which are multiples of 4. We now briefly catalog what spaceship speeds are known and how to construct spaceships with a wider variety of periods.

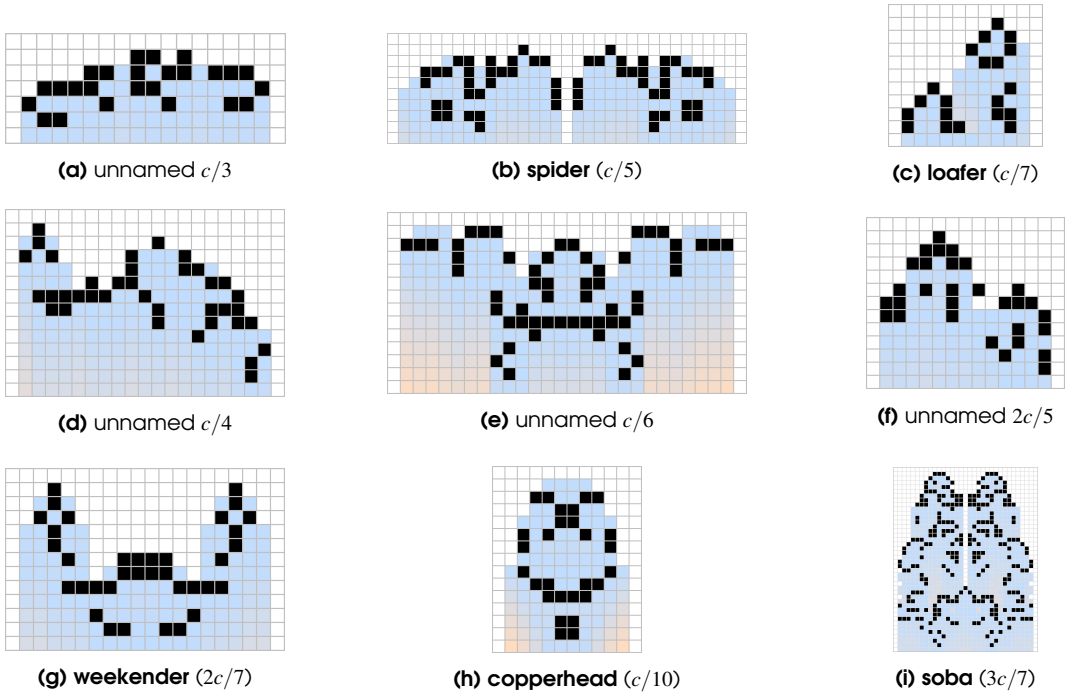


Figure 4.47: A collection of small orthogonal spaceships of various speeds, all oriented so that they travel upward. These spaceships were found by (a) Dean Hickerson in 1989, (b) David Bell in 1997, Josh Ball (c) in 2013 and (d) in 2012, (e) Hartmut Holzwart in 2009, (f) Paul Tooke in 2000, (g) David Eppstein in 2000, (h) ConwayLife.com forum user "zdr" in 2016, and (i) Dylan Chen in 2020.

4.6.1 Spaceship Speeds

Figure 4.47 provides a collection of the smallest known orthogonal spaceships (in terms of number of alive cells) of several different speeds that we have not yet seen.[31] In fact, these 9 spaceships (plus the $c/2$ spaceships that we are already familiar with) represent the only 10 speeds for which **elementary** orthogonal spaceships have been constructed. By an elementary spaceship, we simply mean one that acts "as a whole" rather than by piecing together many smaller reactions.[32] For example, Corderships are not elementary since they are constructed by making use of multiple reactions based on switch engines.

The Life community has had slightly less luck finding elementary diagonal spaceships of different speeds, primarily due to the diagonal speed limit being slower than the orthogonal speed limit. Indeed, slow spaceships typically need to have a higher period than fast spaceships (e.g., a spaceship with speed c/n must have period at least n), and the search space for high-period objects is much larger

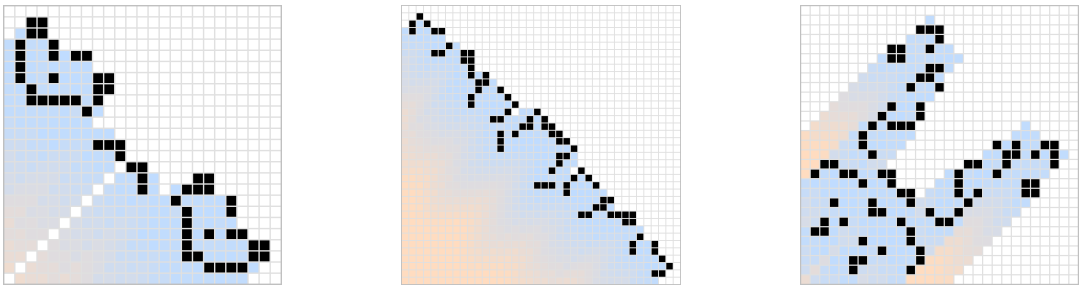
[31] We do not dwell on the exact methods used to find these spaceships, as they were all found via computer search rather than methods that can be mimicked by hand.

[32] This definition is admittedly vague, and essentially impossible to make precise.

Free PDF. See conwaylife.com/book for pattern files and print version.

than it is for low-period objects,[33] so the effectiveness of computer searches drops off quickly.

Nonetheless, three new diagonal spaceships, each travelling at a speed that we have not yet seen, are displayed in Figure 4.48. Together with the $c/4$ spaceships that we are already familiar with, these spaceships represent the only 4 diagonal speeds that have been attained by elementary spaceships.



(a) unnamed $c/5$ **(b)** unnamed $c/6$ **(c) lobster** $(c/7)$

Figure 4.48: The smallest known diagonal spaceships of some unusual speeds, all oriented so that they travel toward the top right. These spaceships were found by Matthias Merzenich in (a) 2010 and (c) 2011, and (b) by Josh Ball in 2011.

Even though all of the spaceships that we have seen so far travel either orthogonally or diagonally, other directions of travel (i.e., diagonally at slopes other than ± 1) are indeed possible. We call a spaceship that travels in one of these non-standard directions an **oblique spaceship**, and we say that its speed is $(x,y)c/n$ if it travels a distance of x cells horizontally and y cells vertically over the course of n generations. Since oblique spaceships all have speed no greater than $(2,1)c/6$ and period equal to at least 6 (see Exercise 4.26), they are quite difficult to find via computer search. However, some elementary oblique spaceships are indeed known, and the first one to be found is displayed in Figure 4.49.[34]

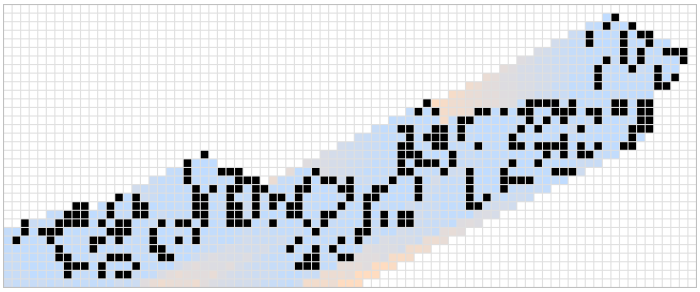


Figure 4.49: Sir Robin is an oblique spaceship with speed $(2,1)c/6$. It was found by Adam P. Goucher in March 2018, based on a partial spaceship that was found by Tomas Rokicki.

Although only a handful of speeds have been realized by elementary spaceships, there is a very large world of **engineered** spaceships that we have not yet looked at—spaceships (typically with thousands or millions of live cells) that work by using simple reactions over and over again in order to carefully move themselves forward. These spaceships are typically not constructed by hand, but rather with the help of custom-designed computer programs that place the individual component reactions together in such a way as to stabilize each other.

[33] This is the same reason that period-specific computer searches have been so effective at finding oscillators with periods below 8, but have had relatively little success with higher periods.

[34] Ships like this one, which travel 2 cells horizontally for every 1 cell that they travel vertically, are called **knightships**, in reference to the knight from chess that moves in the same way. This particular knightship is called **Sir Robin**, after a knight from Monty Python.

Free PDF. See conwaylife.com/book for pattern files and print version.

While we are not yet in a position to discuss the specifics of how these engineered spaceships are pieced together, we will return to this problem in Chapters 10 and 11. For now, we simply list in Table 4.1 a summary of what spaceship speeds are attainable by which methods.

Speed	Direction	Examples
$c/2$	orthogonal	LWSS, MWSS, HWSS
$c/3$	orthogonal	Figure 4.47(a)
$c/4$	orthogonal	Figure 4.47(d)
$c/5$	orthogonal	spider
$2c/5$	orthogonal	Figure 4.47(f)
$c/6$	orthogonal	Figure 4.47(e)
$c/7$	orthogonal	loafer
$2c/7$	orthogonal	weekender
$3c/7$	orthogonal	soba
$c/10$	orthogonal	copperhead
$17c/45$	orthogonal	"caterpillar" engineered spaceship (Section 10.2)
$31c/240$	orthogonal	"silverfish" engineered spaceship (Section 10.1)
all speeds $< c/4$	orthogonal	"caterloopillar" engineered spaceships (Section 10.4)
$c/4$	diagonal	glider
$c/5$	diagonal	Figure 4.48(a)
$c/6$	diagonal	Figure 4.48(b)
$c/7$	diagonal	lobster
$c/12$	diagonal	Corderships
all speeds $< c/4$	diagonal	"Demonoid" engineered spaceships (Section 11.7)
$(2,1)c/6$	slope 2	Sir Robin
$(23,5)c/79$	slope $23/5$	"waterbear" engineered spaceship (Section 10.3)
all speeds $< (1,1)c/579$	all slopes $\neq 1$	"Geminoid" engineered spaceships (Section 11.1.3)

Table 4.1: A summary of the different spaceship speeds that are known to be attainable. Many of these speeds are only attained by engineered spaceships, which we will not discuss in detail until Chapters 10 and 11. When this table says things like "all speeds" or "all slopes", it should be understood that it means all *rational* speeds or slopes, as it is not possible for a spaceship to have irrational speed or slope.

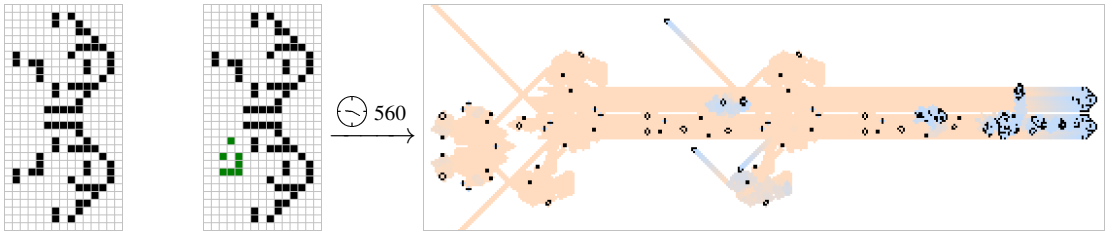
It is worth noting that methods for constructing engineered spaceships demonstrate the existence of spaceships that travel arbitrarily slowly, so there cannot possibly be a variant of Theorem 4.1 that provides a lower bound on the speed of spaceships. In fact, at least in the orthogonal direction we know how to construct about half of all possible speeds, since the caterloopillar construction of Section 10.4 can produce spaceships of any rational speed below $c/4$, leaving only the interval of speeds between $c/4$ and $c/2$ unsolved. Since we already know explicit examples of $c/3$, $2c/5$, $2c/7$, and $3c/7$ orthogonal spaceships, we are left with $3c/8$ as the simplest (i.e., smallest potential period) orthogonal speed for which no spaceship is known. Similarly, the Demonoid construction of Section 11.7 can produce diagonal spaceships of any rational speed below $c/4$, but they all have extremely large periods.

Free PDF. See conwaylife.com/book for pattern files and print version.

4.6.2 Spaceship Periods

Although the primary goal when constructing spaceships is to develop new speeds or directions that have not been realized before, we could also ask how to construct spaceships with a wide variety of periods. When the period of a spaceship is important to us, we are careful not to reduce the fraction that represents its speed. For example, a spaceship with period 14 could travel at $c/2$ with a period 14 spark, and if we were intent on specifying its period, we might say that it travels at $7c/14$. Alternatively, there could be a $c/7$ spaceship with a p14 spark—we do not know of one, but we would call it a $2c/14$ spaceship. If we wish to emphasize that the light, middle, and heavyweight spaceships have period 4 then we would say that they travel at $2c/4$ instead of at $c/2$.

Since a spaceship's period is so closely tied to its speed, and we do not yet know how to construct spaceships of all rational speeds below $c/2$, it should not be surprising that we also do not yet know how to construct spaceships of all periods. However, there is still a lot that we can say about spaceship periods. For example, we have already seen spaceships with period 4 (e.g., the four basic spaceships) and with period 3 (e.g., the one in Figure 4.47(a)). There are also spaceships with period 2, such as the one displayed in Figure 4.50(a), and this period is minimal (if a spaceship had period 1 then it would have to move at lightspeed, which we know is impossible by Theorem 4.1).



(a) p2 spaceship **(b)** p8 puffer

Figure 4.50: (a) A small period 2 spaceship that was found by Dean Hickerson in 1989, and (b) a puffer that can be obtained from this spaceship by changing the shape of one of its rear sparks (displayed in green).

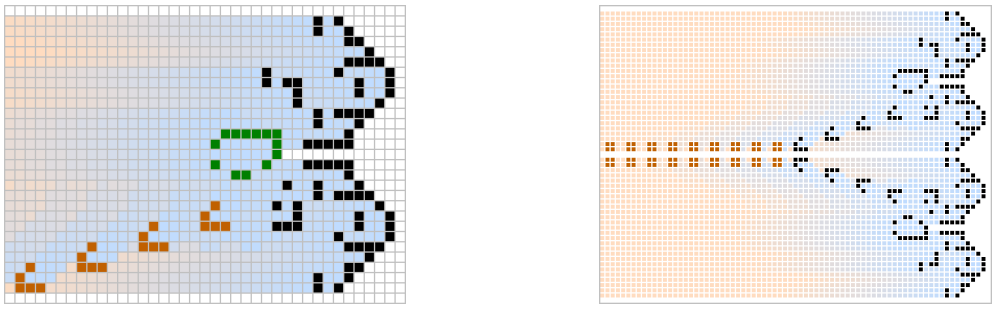
The exciting thing about this period 2 spaceship is that we can actually use it to construct spaceships with arbitrarily large periods. To see how this works, first notice that we can change one of its rear sparks so as to produce a puffer, as in Figure 4.50(b). With this puffer in hand, we play a similar game to the one that we played in Section 4.4.1—we add a nearby heavyweight spaceship so as to transform this puffer's debris into a glider, thus creating the period 8 rake depicted in Figure 4.51(a). This rake has a much lower period than any of the other rakes we have seen so far, and its usefulness lies in the fact that two of them can be combined in such a way that their glider streams collide, creating exactly the bi-block wick that we saw in Figure 4.43(b).[35]

If we were able to start the beehive fuse that burns through this wick, eventually the fuse would catch up with the glider collision (since it burns at $2c/3$, which is faster than the rake's speed of $c/2$), stop burning, and the wick would start being reconstructed again. We would thus have a spaceship that gradually swaps back and forth between being quite small (when its wick is entirely burned up) to being quite large (just before its wick starts burning again), and we could make its period as large as we like just by increasing the distance separating the bi-block puffer from the fuse-igniting reaction.

The pattern displayed in Figure 4.52 does even better; it not only repeatedly re-ignites the bi-block fuse, but it releases a single glider every time as well, so it not only lets us create spaceships with arbitrarily large periods, but even rakes with arbitrarily large periods. Specifically, every additional

[35] We will discuss which glider collisions produce which objects thoroughly in Chapter 5.

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) p8 rake **(b)** p8 bi-block puffer

Figure 4.51: (a) A small period 8 rake constructed by using a heavyweight spaceship (displayed in green) to tame the debris left behind by the puffer from Figure 4.50(b), and (b) a way of arranging two of these rakes so as to leave a trail of bi-blocks behind them.

bi-block between the front and back halves of the rake increases its period by 32.[36] To turn this rake into a spaceship, an additional LWSS can be added to delete the rake's output glider via the reaction that we saw back in Figure 4.12.[37]

Another method of constructing adjustable-period spaceships,[38] based on the blinker fuse of Figure 4.43(a), is presented in Exercise 4.36. This technique has the advantage that it can be used to construct spaceships of all sufficiently large periods that are multiples of 4 (instead of just every 32nd period), but the disadvantage that adjusting its period is slightly more complicated.

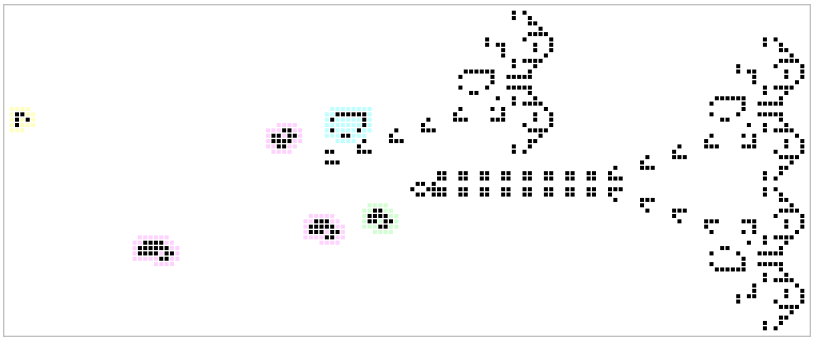


Figure 4.52: An **adjustable rake**, which can be made to have any period of the form $264+32n$, where $n \geq 0$ is an integer, by increasing the length of the bi-block wick in the middle. In the form displayed here, it has period 392. The HWSS highlighted in aqua destroys the gliders in the stream directly below it, creating a far-away banana spark in the process. This spark, together with the spark from the LWSS highlighted in green ignites the bi-block fuse, but also leaves behind some debris. The three ships outlined in magenta transform that debris into the output glider outlined in yellow.

4.7 Notes and Historical Remarks

While the glider, lightweight spaceship, middleweight spaceship, and heavyweight spaceship were all found by hand in 1970, very little was known about spaceships for the first two decades of Life. All early spaceship discoveries were simple modifications of those 4 standard spaceships, such as flotillae

[36] Each extra bi-block adds a horizontal width of 4 cells to the wick, so the $c/2$ front end takes an extra 8 generations to lay it down. Since the difference in speed between the front and back ends of this rake is $2c/3 - c/2 = c/6$, the fuse then burns for an extra $4 \times 6 = 24$ generations, for a total of $8 + 24 = 32$ generations added to the rake's period.

[37] A slightly smaller method of turning this rake into a spaceship is described in Exercise 4.34.

[38] Developed by David Bell in 1992, with help from Dean Hickerson.

Free PDF. See conwaylife.com/book for pattern files and print version.

and tagalongs like the Schick engine. The first *truly* new spaceships to be found were several period 2 spaceships by Dean Hickerson in July 1989, including the one that we saw in Figure 4.50(a).[39]

Hickerson's search program continued to find new types of spaceships throughout the year, including the first $c/3$ spaceship in August, the first $c/4$ orthogonal spaceship in December, and the first diagonal spaceship other than the glider in December (see Figure 4.53). It also found the first $2c/5$ spaceship in 1991, and his same algorithm continues to be used to this day to find new spaceships and oscillators.[40]

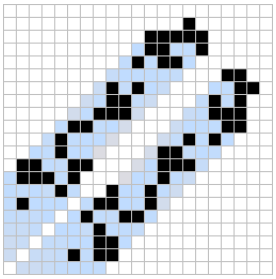


Figure 4.53: The **big glider**: a $c/4$ period 4 diagonal spaceship, and the first diagonal spaceship other than the glider to be discovered. Found by Dean Hickerson in December 1989.

Some other Life enthusiasts continued using search programs to find new orthogonal spaceships throughout and after the 1990s, with some of the most notable discoveries being:

- the first $c/5$ orthogonal spaceship, found by Tim Coe in 1996;
- the first $2c/7$ orthogonal spaceship, the weekender, found by David Eppstein in 2000 [Epp02];
- the first $c/6$ orthogonal spaceship, found by Paul Tooke in 2000;
- the first $c/7$ orthogonal spaceship, the loafer, found by Josh Ball in 2013;
- the first $3c/7$ orthogonal spaceship, the **spaghetti monster**, found by Tim Coe in 2016; and
- the first $c/10$ orthogonal spaceship, the copperhead, found by ConwayLife.com forum user "zdr" in 2016.

The small size of the loafer and copperhead demonstrate just how limited our search techniques for spaceships really are. The copperhead especially was a shocking discovery, as it had gone unnoticed for over 45 years of Life, but it could have been found in one hour using the publicly available search program "gfind" that David Eppstein wrote to find the weekender. Even more shockingly, it was then found in random ash generated by apgsearch less than a month after its initial discovery (see Exercise 1.7(b)).[41]

Diagonal spaceships are somewhat more difficult to search for than orthogonal spaceships due to their lower speed limit and thus higher periods (see Exercise 4.26), with search techniques not producing the first new diagonal speeds of $c/5$, $c/6$, and $c/7$ until 2000 (by Jason Summers), 2005 (by Nicolay Beluchenko), and 2011 (by Matthias Merzenich), respectively.

Oblique spaceships are even more difficult to find, with the first elementary one (Sir Robin) not being found until 2018 (by Adam P. Goucher and Tomas Rokicki), despite considerable effort having

[39] It is not known exactly which period 2 spaceship was found first—they were all found very close together by the same computer searches.

[40] His algorithm is now implemented in programs called lifesrc and JavaLifeSearch. See `conwaylife.com/wiki/Lifesrc` for documentation and download locations.

[41] However, this is perhaps slightly misleading. The copperhead was found from evolving a random symmetric soup, and it's likely that there would not have been as much of a push to search symmetric soups if the copperhead had not come along in the first place.

Free PDF. See conwaylife.com/book for pattern files and print version.

been put into finding one over the preceding 20 years. There was a remarkably close call in March 2004, when Eugene Langvagen found the small pattern displayed in Figure 4.54. This pattern is roughly $1/4$ of the size of Sir Robin and is *almost* an elementary knightship—after 6 generations it has moved by 2 cells horizontally and 1 cell vertically, except with the state of just 2 of its cells incorrect.

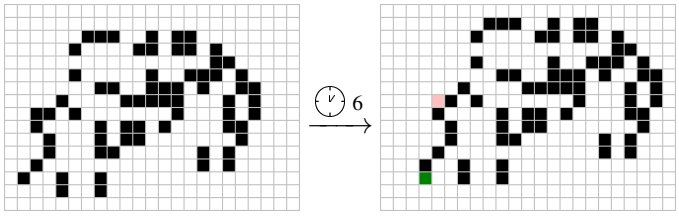


Figure 4.54: An "almost knightship" that travels to the right by 2 cells and up by 1 cell over the course of 6 generations, except with one extra cell born (shown in green) and one extra cell dead (shown in red).

No other spaceship speeds or directions have been found via computer search, but a lot of success has been had by stitching together multiple copies of simple reactions in clever ways. The first spaceship that was found in this way was the 13-engine Cordership, which was found in 1991 by Dean Hickerson, who also found most of the other early Corderships. The next spaceship to be found via construction was the caterpillar, which is a $17c/45$ orthogonal spaceship based on the reaction displayed in Figure 4.55.

Even though this reaction by itself is unstable, it can be chained together with itself and other reactions to create a true spaceship (see Section10.2). However, the details of how these reactions fit together are considerably more complicated than they were for Corderships, and had to be carried out by a computer program written by Gabriel Nivasch (with help by David Bell and Jason Summers) in 2004. The completed caterpillar spaceship has over 11.8 million live cells, and was the largest interesting Life pattern by live cell count until being surpassed in 2018 by the 0E0P metacell with 18.6 million live cells (see Chapter 12).

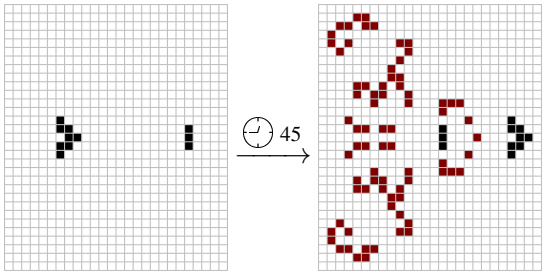


Figure 4.55: A reaction in which a pi-heptomino collides with a blinker in such a way as to move forward by 17 cells in 45 generations while moving the blinker backward by 6 cells. The cells displayed in red on the right die off completely in another 24 generations.

The next engineered spaceship to be constructed was **Gemini**, which was created by Andrew J. Wade in 2010. The original form of this spaceship had speed $(1024, 5120)c/33699586$ and was the first known oblique spaceship. Furthermore, its construction can be altered in a rather systematic way to create spaceships of any direction and arbitrarily slow speeds.[42] Several other types of massive

[42]The existence of oblique spaceships and arbitrarily slow spaceships was already known in the early 1970s [Wai74, BCG82], but Gemini was the first explicit construction. Despite having over 800 000 live cells, it was orders of magnitude smaller and faster than such a spaceship was expected to be.

Free PDF. See conwaylife.com/book for pattern files and print version.

engineered spaceships have been constructed since the Gemini, and we will investigate them in depth in Chapters 10 and 11.

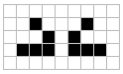
Interest in wires has dwindled in recent years, simply because signals on wires are more difficult to manipulate than signals (gliders in particular) in a vacuum. If we want to send a signal from one location in the Life plane to another, it is typically simpler to just point a glider in the right direction rather than requiring that a particular wire stretches all the way between those two locations. Furthermore, we have a lot of machinery for repositioning and re-timing gliders, but hardly any such machinery for signals (we do not know of a single "efficient" signal elbow, for example). Dean Hickerson wrote a computer program to search for such a signal elbow, with the intention of then creating a fast signal loop that solved the omniperiodicity problem. While that computer search was unsuccessful, it did lead to many of the known billiard table oscillators.

Exercises

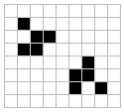
solutions to starred exercises on page 455

∗**4.1** [1/5] Determine whether the given pair of gliders have the same or the opposite color as each other.

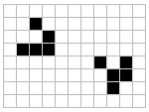
(a)



(b)



(c)



∗**4.2** [1/5] Determine whether the specified glider reflector is color-preserving or color-changing.

(a) Buckaroo (see Figure 3.16(b)).
(b) Relay (see Figure 3.28).
(c) Boojum reflector (see Exercise 3.23(a)).
(d) Rectifier (see Exercise 3.23(b)).
(e) Bouncer reflectors (see Figure 6.36).

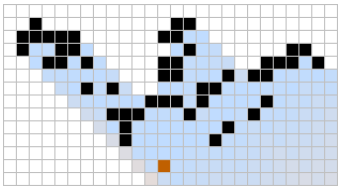
4.3 Recall the tubstretcher that we introduced in Figure 4.10.

(a) [1/5] Modify the tubstretcher so that it stretches a boat instead of a tub (it would now be called a **boat-stretcher**).
(b) [2/5] Modify the tubstretcher so that it stretches two tubs instead of just one.
(c) [1/5] The pattern that you constructed in part (b) grows by 4 cells every 4 generations. Place another period 4 object (either a spaceship or an oscillator) on the Life plane so that the total number of live cells on the plane grows by exactly 1 every generation.

4.4 [2/5] Show how two copies of the tagalong from Figure 4.14(b) can stabilize each other, resulting in a $c/2$ orthogonal spaceship that does not contain an xWSS.

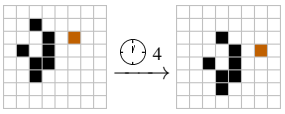
4.5 [3/5] There are 5 different flotillae that consist of exactly two middleweight spaceships. Find them all.

∗**4.6** [2/5] The following $c/4$ diagonal spaceship is called a **swan**,[43] and it emits a very accessible spark.

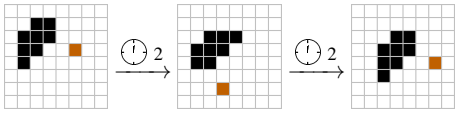


(a) Use this spark to create a tubstretcher.
(b) Find a way of colliding a glider with this spark so that two gliders are produced—one traveling southeast and one traveling southwest.

4.7 [2/5] The $c/4$ diagonal tagalong displayed below can be attached to many $c/4$ diagonal spaceships that emit a spark. Attach it to three different spaceships.



4.8 [3/5] The $c/4$ diagonal tagalong displayed below is rather difficult to use since most spaceships that emit sparks in the desired positions collide with each other.



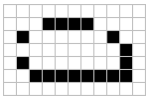
(a) Find a way to place this tagalong between two spaceships.
(b) This tagalong is called the **glider emulator**, since it leaves behind a spark that behaves very similarly to the trailing cell of a glider. Use this spark to attach a copy of the Canada goose tagalong to the spaceship that you constructed in part (a).

[43] Found by Tim Coe in 1996.

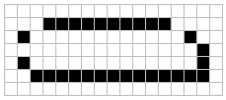
Free PDF. See conwaylife.com/book for pattern files and print version.

∗**4.9** [2/5] Use exactly two heavyweight spaceships to stabilize each of the following overweight spaceships, turning them into flotillae.

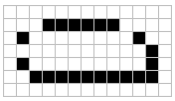
(a)



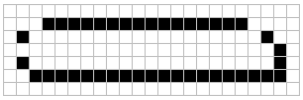
(b)



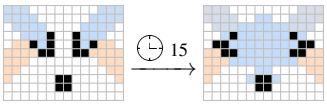
(c)



∗**4.10** [3/5] Use (potentially many) light, middle, heavy, and/or overweight spaceships to create a flotilla that includes the following overweight spaceship:



4.11 [4/5] The following collision of two gliders with a block is called a **rephaser**, since it alters the phase and path of the gliders. It is useful since it pushes each glider over by 3 lanes, so it can be used to separate closely spaced glider streams that Snarks are not small enough to separate.



(a) If two glider streams are travelling in the same direction but on different lanes, how many lanes must they be offset from one another by in order for us to be able to use a Snark to separate them (i.e., reflect one of the gliders without interfering with the other)?

(b) Use the rephaser, together with some Snarks, to separate two glider streams that are just 16 lanes apart (which should be less than your answer to part (a)).

(c) Use *two* rephasers, together with some Snarks, to separate two glider streams that are just 10 lanes apart.

4.12 Recall the hivenudger that was introduced in Figure 4.14(d).

(a) [2/5] Create a hivenudger that uses a lightweight spaceship, a middleweight spaceship, and 2 heavyweight spaceships.

(b) [2/5] Create a hivenudger that uses a Coe ship as one of its rear corners.

(c) [4/5] How many different hivenudgers can be constructed by using xWSSes at its four corners?
[Hint: The answer is not $3^4 = 81$. Why not?]

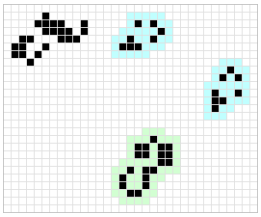
4.13 [3/5] Similar to how we introduced the color of a glider, we could talk about the color of any diagonal spaceship, and it would be important to be familiar with that spaceship's color if we were to reflect it around a track. However, color is only *sometimes* a useful property when reflecting orthogonal spaceships.

(a) Explain why it is *not* necessary to consider the color of a loafer that we reflect around a track.

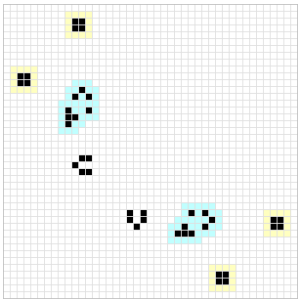
(b) Explain why it *is* useful to consider the color of an xWSS that we reflect around a track. Why are we more restricted when reflecting these spaceships than when reflecting loafers?

4.14 [2/5] Create a new puffer by replacing one of the lightweight spaceships in Figure 4.25 with a middleweight spaceship.

∗**4.15** [4/5] A reaction is displayed below in which two switch engines (highlighted in aqua) bounce off of each other and a third switch engine (highlighted in green) cleans up some debris. Use this reaction to create a Cordership.
[Hint: A 7-engine Cordership is possible.]



∗**4.16** [4/5] A reaction is displayed below in which two switch engines (highlighted in aqua) bounce off of each other and destroy some trails of blocks (highlighted in yellow).



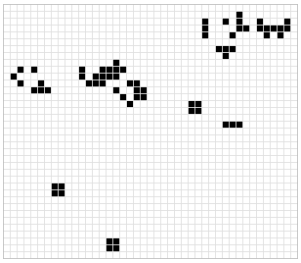
(a) How does this reaction differ from the reaction displayed in Figure 4.22?

(b) Show how this reaction can be used to reflect a glider by 90 degrees.
[Hint: What sparks does this reaction emit?]

(c) Use this reaction to create a Cordership. Use the reaction from part (b) to show how this Cordership can reflect a glider.

Free PDF. See conwaylife.com/book for pattern files and print version.

4.17 [3/5] Here is an arrangement of two switch engines that evolves into a puffer for a single diagonal row of blocks:



(a) Find a position for a second copy of this puffer diagonally behind the first one, such that the blocks from the first puffer suppress the creation of blocks in the second puffer, resulting in an adjustable-length 4-engine Cordership.[44]

(b) Replace the second puffer with a diagonal mirror-image of itself, and also delay it by some number of ticks so that the two halves of the Cordership are no longer in phase with each other. Make sure that the resulting pattern is still a working 4-engine Cordership.

∗**4.18** [3/5] A Cordership that makes use of just 3 switch engines is displayed below.



(a) What happens to this spaceship if you remove the central switch engine?

(b) Find a way of using the sparks at the back end of this Cordership to reflect a glider by 90 degrees.

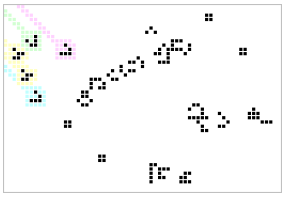
4.19 [2/5] Find versions of the Schick engine that, instead of using two lightweight spaceships, use...

(a) two middleweight spaceships,

(b) two heavyweight spaceships, and

(c) one lightweight spaceship and one heavyweight spaceship.

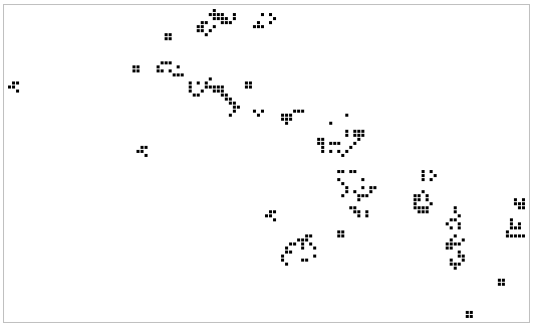
∗**4.20** [3/5] A Cordership that makes use of just 2 switch engines[45] is displayed below, along with several incoming gliders that it can reflect in different ways.



(a) Use the 180-degree reflection highlighted in aqua to bounce a glider back and forth between two receding Corderships.

(b) Explain why the other 180-degree reflections (highlighted in green and magenta) cannot be used to make similar glider-bouncing shuttles.

∗**4.21** [3/5] A **Corderrake** is displayed below, which is a $c/12$ rake based on switch engines that shoots gliders sideways as it moves.[46]



(a) How many switch engines does this Corderrake use?

(b) Use this rake, together with other Corderships that we have seen in this chapter, to construct a diagonal rake that shoots gliders behind it (rather than to its side).[47]

(c) Use this rake, together with other Corderships that we have seen in this chapter, to construct a diagonal rake that shoots lightweight spaceships.

[44] First constructed by Michael Simkin in November 2014.

[45] Found by user "praosylen" on the ConwayLife.com forums in December 2017.

[46] Both the Corderrake and the 3-engine Cordership were found by Paul Tooke in 2004.

[47] Objects like this that shoot gliders parallel to their direction of motion are sometimes not considered rakes, since the gliders end up travelling single-file, rather than "raking out" a portion of the Life plane.

Free PDF. See conwaylife.com/book for pattern files and print version.

∗**4.22** [3/5] Recall the Schick engine that was introduced in Section 4.4.2.

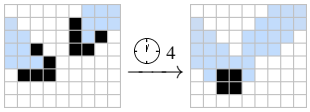
(a) Find a six-cell object that, when placed behind two lightweight spaceships, evolves into a Schick engine. [Hint: Just truncate the spark in one of the Schick engine's phases.]

(b) The six-cell object from part (a) follows the same evolutionary sequence as a commonly occurring unstable object that we have already seen. What object is it?

∗**4.23** [2/5] Use the configuration of two heavyweight spaceships displayed in green in Figure 4.32(b) to reflect the gliders from a backward space rake (either the period 20 or period 60 version), creating a forward rake.

4.24 [2/5] Create a period 240 forward rake and a period 240 backward rake by replacing the period 20 space rake with the period 60 variant in Figures 4.33(a) and 4.33(b).

4.25 [3/5] It is possible to collide two gliders in such a way as to create a block, as displayed below.



(a) Use this glider collision and two forward space rakes to create a pattern that leaves a trail of blocks behind it, spaced 10 cells apart from each other.

(b) Use two copies of the backward rake from Figure 4.30(b) to create a pattern that leaves a trail of blocks behind it, spaced 30 cells apart from each other.

(c) Try altering the two-glider collision slightly to see what other types of objects you can make with two gliders. Use two rakes to create a pattern that leaves behind a trail of some object other than a block, such as a blinker or a pond.

4.26 [4/5] Suppose that a spaceship travels x cells horizontally and y cells vertically throughout its period.

(a) Show that its speed cannot exceed $(x,y)c/(2x+2y)$.

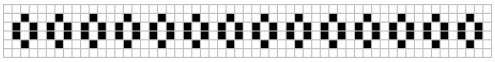
(b) Show that its period must be at least $2x+2y$.

(c) Show that if a spaceship has period 3 then it must be orthogonal and have speed $c/3$.

(d) What are the possible directions and speeds of period 4 spaceships? Give examples to show that all of the possibilities you list are attainable.

4.27 [3/5] Give an example of an infinitely large pattern that moves through the (otherwise empty) Life plane at a speed of c. Why does this pattern's existence not contradict Theorem 4.1?

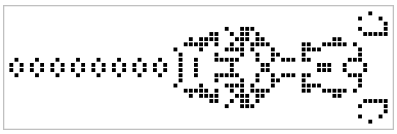
4.28 Consider the beehive wick displayed below.



(a) [2/5] Place a live cell near one end of this beehive wick so as to make a $4c/5$ fuse.

(b) [3/5] Find a $4c/5$ fuse that *cleanly* burns through this wick.[48] [Hint: Try placing some *symmetric* debris near the end of the wick.]

4.29 [4/5] Use the beehive puffer below and the $4c/5$ fuse from Exercise 4.28(b) to create an adjustable-period spaceship.



[Hint: Use the same general method that we used to construct the adjustable-period rake in Figure 4.52: use gliders and/or sparks to start the fuse burning and then use standard spaceships to clean up debris.]

∗**4.30** [2/5] Consider the three signals displayed in Figure 4.41.

(a) Find the period of each of these signals.

(b) Find the minimum number of generations that must separate two copies of these signals on the same wire.

4.31 [2/5] Modify the wick in Figure 4.43(b) so that it has slope $1/2$ and is burned through cleanly by the same beehive reaction.

∗**4.32** [3/5] Show how the $c/5$ diagonal spaceship in Figure 4.48(a) can be used to reflect a glider by 90 degrees.

4.33 [3/5] In Figure 4.51(a) we showed a period 8 rake. What is the smallest period that a rake could conceivably have (i.e., without gliders colliding)?

4.34 [3/5] Recall the adjustable rake displayed in Figure 4.52.

(a) Turn this rake into a spaceship by adding a single lightweight spaceship.

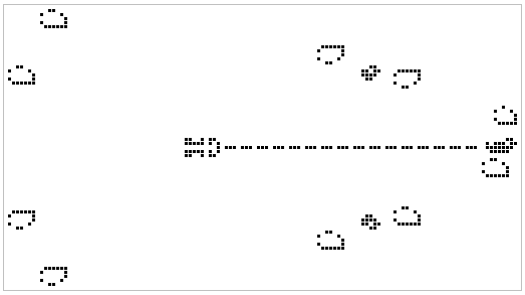
(b) Turn this rake into a spaceship by removing the three spaceships highlighted in magenta and adding two middleweight spaceships.
[Hint: What debris is left over if you remove the three spaceships highlighted in magenta?]

4.35 [2/5] Create a rake with period equal to exactly 1000.

[48] This fuse was originally found by Dean Hickerson in June 1993.

Free PDF. See conwaylife.com/book for pattern files and print version.

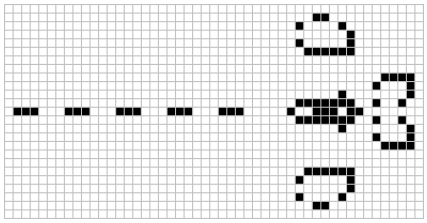
4.36 The spaceship displayed below has period 1 136.[49]



(a) [2/5] Watch this spaceship evolve and describe how it works. For example, which part of the pattern lays the blinker wick? What effect do the heavyweight spaceships at the back have?

(b) [3/5] Show how the rear heavyweight spaceships can be moved so as to create a puffer with larger period. Use this method to explicitly construct a puffer with period greater than 1500.

4.37 [3/5] The blinker puffer below works by using two heavyweight spaceships to hassle the spark behind a Schick engine.



(a) Place a heavyweight spaceship near the blinker fuse so as to delete it via one of the reactions from Figure 4.12.

(b) Aim a period 60 space rake at the blinker fuse so as to create puffers for various objects. You should be able to create puffers that make a single (i) ship, (ii) loaf, or (iii) blinker every 60 generations.

[49] It was constructed independently by David Bell and Dean Hickerson in 1992.

Free PDF. See conwaylife.com/book for pattern files and print version.

Free PDF. See conwaylife.com/book for pattern files and print version.

II Circuitry and Logic

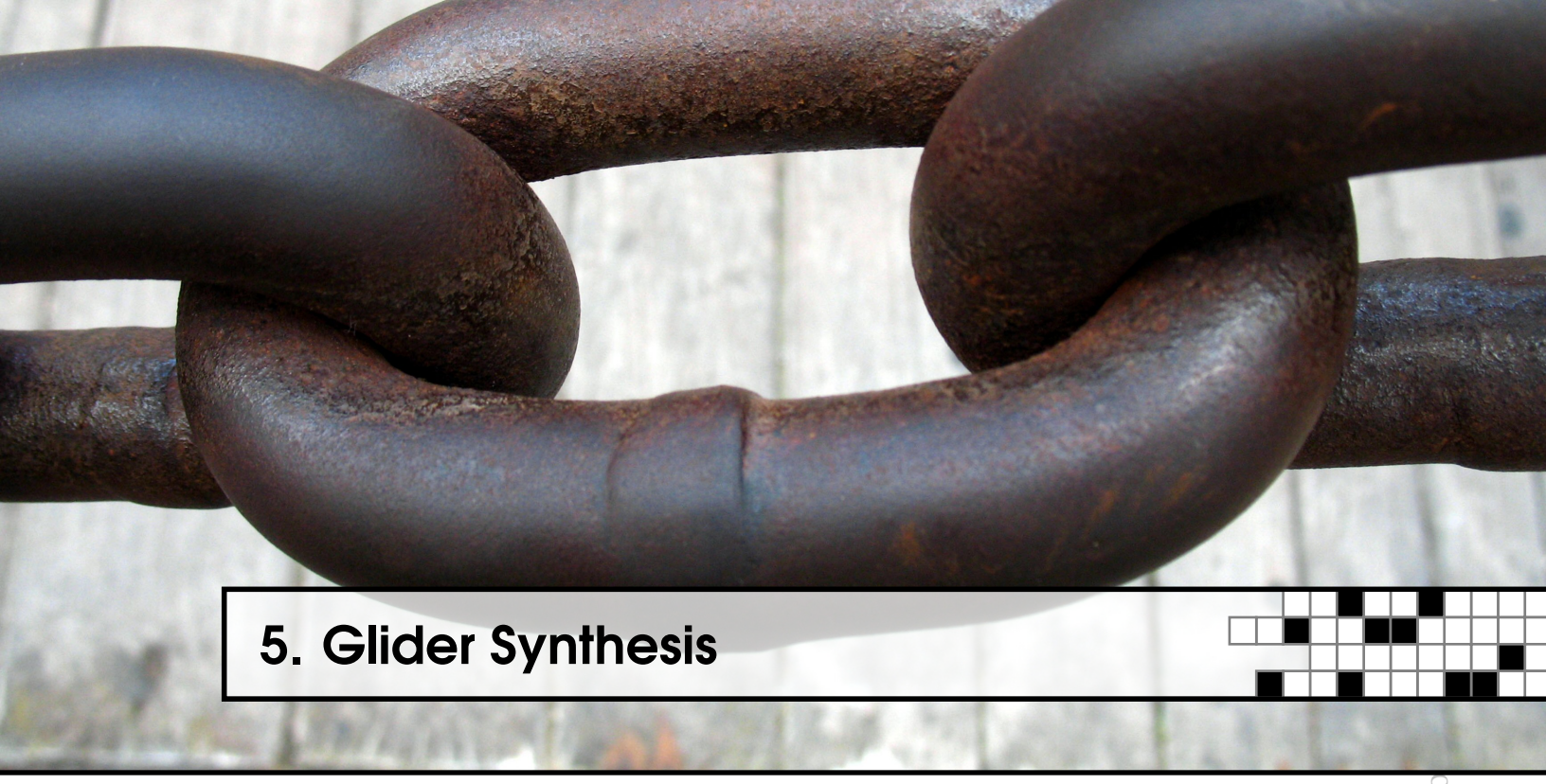
5 Glider Synthesis 121
5.1 Two-Glider Syntheses
5.2 Syntheses Involving Three or More Gliders
5.3 Incremental Syntheses
5.4 Synthesis of Moving Objects
5.5 Developing New Syntheses
5.6 A Gosper Glider Gun Breeder
5.7 Slow Salvo Synthesis
5.8 Notes and Historical Remarks
Exercises

6 Periodic Circuitry 153
6.1 Period 30 Circuitry
6.2 Primer
6.3 Period 46 Circuitry
6.4 Bumpers and Bouncers
6.5 Glider Timing and Regulators
6.6 Notes and Historical Remarks
Exercises

7 Stable Circuitry 183
7.1 Herschel Conduits
7.2 From Herschels to Gliders
7.3 From Gliders to Herschels
7.4 From Gliders to Gliders
7.5 Synthesizing Objects via Conduits
7.6 Period Multipliers and Small High-Period Guns
7.7 Converters for Other Objects
7.8 Factories
7.9 Notes and Historical Remarks
Exercises

8 Guns and Glider Streams 221
8.1 Glider Deletion
8.2 Glider Insertion
8.3 Streams of Other Spaceships
8.4 Glider Guns of Any Period
8.5 True-Period Guns
8.6 Slide Guns
8.7 Armless Guns
8.8 Slow and Irregular Guns
8.9 Notes and Historical Remarks
Exercises

Free PDF. See conwaylife.com/book for pattern files and print version.



5. Glider Synthesis

Life isn't about finding yourself. Life is about creating yourself.

George Bernard Shaw

In previous chapters, we saw several different patterns that were capable of generating an endless supply of gliders: glider guns like the Gosper glider gun create streams of gliders coming from a fixed location, and rakes like the space rake create waves of gliders that come from a moving source. We have also seen a few patterns (mostly based on the switch engine) capable of creating other objects like blocks or other small still lifes.

In this chapter, we develop a systematic method of constructing patterns that turn these simple objects like gliders and blocks into more complicated objects. In particular, we look at how to collide gliders with each other and with other objects so as to create (or "synthesize") new ones. For example, Figure 5.1 illustrates a method of colliding three gliders so as to produce a lightweight spaceship.

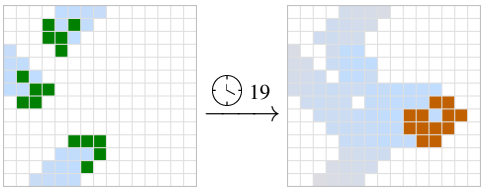


Figure 5.1: Three gliders colliding in such a way as to create a lightweight spaceship.

We call this a **3-glider synthesis** of the lightweight spaceship, and it is useful because we already know of some patterns (i.e., glider guns) that generate gliders, so we can now create patterns that generate lightweight spaceships: we can just aim the output of three glider guns so as to collide with each other in the right way. For example, Figure 5.2 uses three Gosper glider guns to create a period 30 lightweight spaceship gun.

There is nothing particularly special about the lightweight spaceship in this example—we will

Free PDF. See conwaylife.com/book for pattern files and print versio

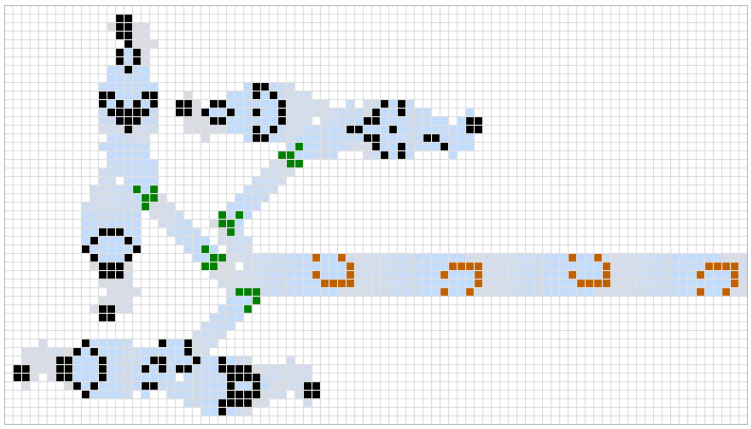


Figure 5.2: A lightweight spaceship gun constructed by using three Gosper glider guns (top-right, top-left, and bottom-left) to shoot three gliders at each other, which collide in the orientation depicted in Figure 5.1 in order to create a lightweight spaceship.

see throughout this chapter that we can collide gliders from glider guns to create a wide variety of moving objects. We can also use gliders to synthesize stationary objects, but for this the glider source has to be moving in order to prevent subsequent gliders from colliding with the synthesized object. Indeed, once we have a large catalog of glider syntheses, the world of Life will open up considerably for us—we will be able to create patterns that construct almost any object in almost any location on the Life plane that we like.

As one somewhat technical note before we proceed, we require that the gliders in a synthesis could arrive at their positions from arbitrarily far away, since our ultimate goal is to make use of these syntheses via guns and rakes. An example of a glider collision that is *not* considered a true glider synthesis, since there is no way to get the gliders in the indicated positions, is provided in Figure 5.3.

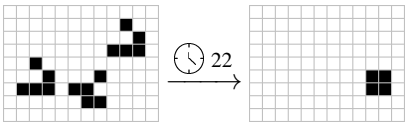


Figure 5.3: A collision of three gliders that produces a block. However, if the two rightmost gliders came from farther away, they would collide with each other before reaching the displayed configuration, so this is not a valid 3-glider synthesis.

5.1 Two-Glider Syntheses

Once we have a glider synthesis in hand, it is typically straightforward to make use of it, since glider streams can typically be created just by lining up glider guns or rakes in the right spots. But how can we come up with glider syntheses in the first place? For example, how was the three-glider synthesis of a lightweight spaceship in Figure 5.1 found? The simplest answer, as unsatisfying as it might seem, is to just collide a bunch of gliders together in different ways and catalog which collisions lead to which objects. Once we have a good selection of "simple" objects that we know how to synthesize, we can then try colliding gliders with those objects to create more complicated patterns, and so on.

To begin down this road, we look at the simplest type of glider synthesis we can perform: synthesis using a collision of only two gliders. We already saw a few two-glider collisions in Section 4.4.3 when we aimed two glider waves at each other so as to destroy some gliders and reflect others, and again in Section 4.6.2 when we aimed two glider waves at each other to create a bi-block wick. Fortunately,

Free PDF. See conwaylife.com/book for pattern files and print version.

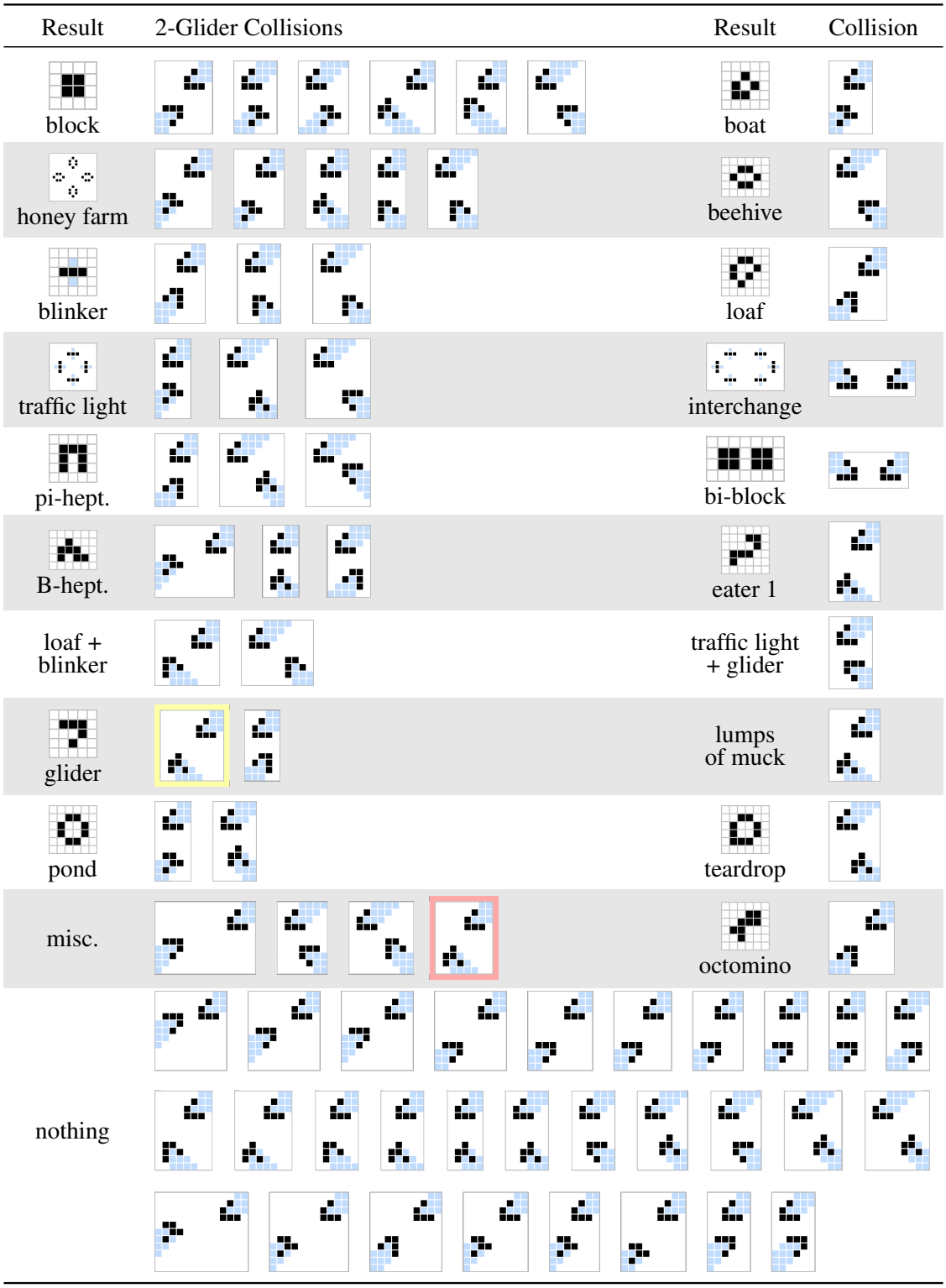


Table 5.1: A summary of the results of all 71 possible 2-glider collisions. The four "misc" collisions yield somewhat messy combinations of common objects like blocks and blinkers. The rightmost of the "misc" collisions (highlighted in red) is sometimes called the **two-glider mess**, as it takes 530 generations to stabilize—more than any of the other collisions. The left glider-producing collision (highlighted in yellow) is called the **kickback reaction**, since it produces an output glider traveling in a different direction than either of the input gliders.

Free PDF. See conwaylife.com/book for pattern files and print version.

cataloging all two-glider syntheses is not too difficult, since there are only 71 different ways that two gliders can collide.[1] A full summary of exactly which two-glider collisions lead to which objects being created is provided by Table 5.1.

While the objects that we can create with just two gliders are not particularly exciting, many of them, such as eater 1 and the B-heptomino (which we recall evolves into a Herschel), are essential building blocks of complex patterns. It is also worth pointing out that the two collisions that result in a single glider can in fact sometimes be useful. In particular, the one of these collisions that is on the left in Table 5.1 is called the **kickback reaction**, and it is useful for the fact that the output glider travels in a different direction than either of the two input gliders. This feature can help us navigate gliders around tight spots and simplify many complicated glider syntheses (see Exercise 5.13, for example).

5.2 Syntheses Involving Three or More Gliders

When moving from two-glider syntheses to three-glider syntheses, things become much more complicated—it is no longer possible to list all collisions and catalog their output, since there are far too many possibilities. To see why this is, recall that the two-glider mess takes 530 generations to stabilize, and we could fire a third glider from dozens of different positions and hundreds of different timings to collide with that chaotic mess. Even worse, that two-glider mess produces multiple gliders, which we could hit with the third glider after an arbitrarily long period of time so as to synthesize a new object as far away as we like.

On the other hand, being unable to catalog all of these three-glider collisions is perhaps not too much of a loss, since we expect that the majority of them would lead to uninteresting combinations of blocks, blinkers, and other common objects that we already know how to synthesize with just two gliders. We thus only present, in Table 5.2, some of the particularly useful three-glider syntheses of more exotic objects. However, we stress that this table is not complete: there are known three-glider syntheses not presented in this table, and it is entirely possible that there are simple objects with three-glider syntheses that have not yet been found.[2]

Although the 3-glider syntheses of a single glider from Table 5.2 (which are called **tees**) might seem somewhat silly at first glance, they have the useful property that the synthesized glider travels perpendicular to each of the input gliders[3]—a property that is not shared by any of its 2-glider syntheses. For this reason, these syntheses are actually quite useful when trying to manipulate glider positions or reduce the number of directions used in glider syntheses of more complicated objects (see Exercise 5.15, for example).

The three-glider collision that creates a glider-producing switch engine is quite exciting, as it is our first example of a synthesis of an infinitely growing pattern. Being able to synthesize queen bees is similarly exciting, since a Gosper glider gun is made up of nothing more than two blocks and two queen bees, all of which we now know how to synthesize. By simply synthesizing all of these objects in the correct positions and phases, we are now able to use gliders to synthesize a pattern that creates additional gliders (see Figure 5.4),[4] and we could conceivably use rakes to create Gosper glider guns

[1] There does not seem to be a nice "mathematical" way to arrive at the number 71 here: it's just a matter of arranging gliders in all possible different orientations and phases and seeing which ones result in collisions.

[2] For example, the pentadecathlon and switch engine were not known to be synthesizable via only three gliders until the syntheses given in Table 5.2 were found by Heinrich Koenig and Luka Okanishi in April 1997 and March 2017, respectively. Similarly, it was not known how to create *any* infinitely growing object using just 3 gliders until Michael Simkin found a 3-glider collision that produces a glider-producing switch engine in October 2014.

[3] The first two of these syntheses work by colliding two of the gliders so as to create a banana spark, which then reflects the third glider as in Figure 3.17.

[4] Synthesizing the Gosper glider gun "piece-by-piece" like this is then quite straightforward, and requires 10 gliders. However, slightly smaller syntheses of the Gosper glider gun are known, requiring as few as 8 gliders.

Free PDF. See conwaylife.com/book for pattern files and print version.

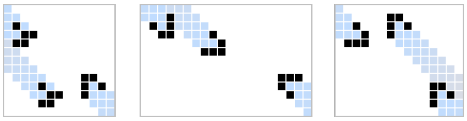
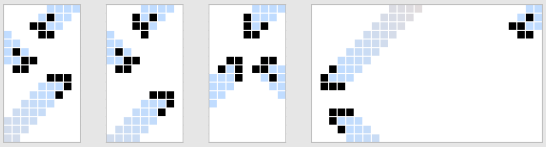
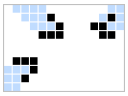
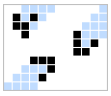
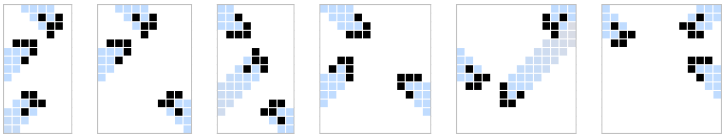
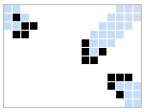
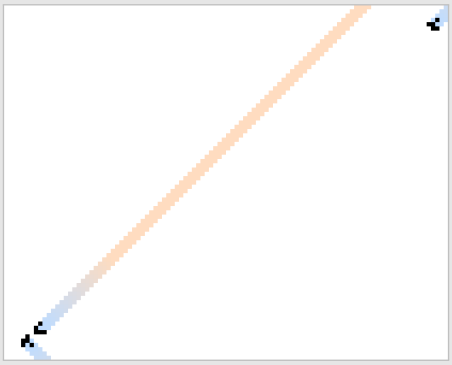
Result	3-Glider Collisions
glider	
lightweight spaceship	
middleweight spaceship	
heavyweight spaceship	
pentadecathlon	
pulsar	
queen bee	
R-pentomino	
switch engine	
glider-producing switch engine (plus junk)	

Table 5.2: A selection of useful 3-glider syntheses. The 3-glider collision that creates a glider-producing switch engine is not a true glider synthesis due to the fact that it also creates a wide assortment of other debris, but it is nonetheless noteworthy for being the only known way of generating infinite growth with just 3 gliders.

Free PDF. See conwaylife.com/book for pattern files and print version.

that in turn create other patterns.

We have a fairly wide variety of objects that we can now synthesize with gliders, but there are still some rather fundamental objects that are missing from our lists of 2- and 3-glider syntheses. For example, if we wanted to synthesize a twin bees gun, we would first have to know how to synthesize twin bees, which (as far as we know) requires at least 4 gliders. It is also sometimes beneficial to be aware of glider syntheses that make use of more gliders, even when syntheses involving fewer gliders exist. For example, the 3-glider synthesis of the heavyweight spaceship given in Table 5.2 is actually quite difficult to make use of, since the two gliders coming in from the top-right are so close together that they cannot be generated by many of the glider sources that we have seen (like adjacent Gosper glider guns).

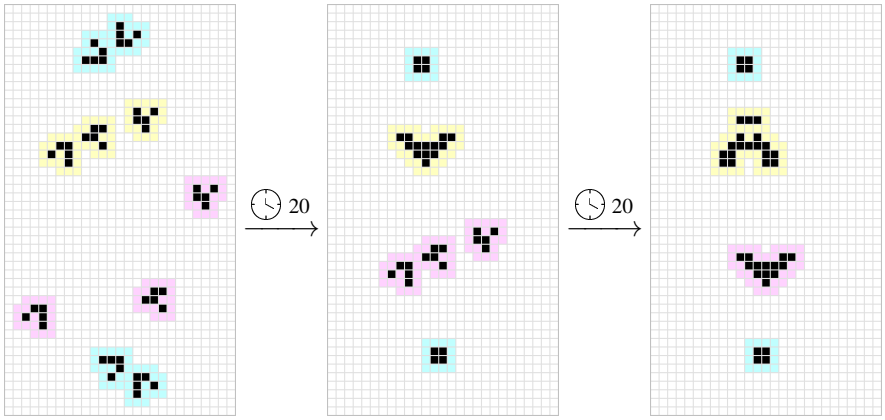


Figure 5.4: A ten-glider synthesis of a Gosper glider gun, which is composed of two syntheses of blocks (highlighted in aqua) and two syntheses of queen bees. The queen bee synthesis highlighted in magenta is the same as the one highlighted in yellow, but delayed by 20 generations in order to make the queen bees collide in their proper phases.

For this reason, we often prefer glider syntheses that use widely spaced gliders, even if that comes at the expense of a larger number of gliders. In particular, there are 4-glider syntheses of a heavyweight spaceship that consist of one glider coming from each direction, and thus are often much easier to make use of. We present a summary of useful syntheses that involve 4 or more gliders in Table 5.3.

5.3 Incremental Syntheses

Glider syntheses can quickly become cumbersome to use as the number of gliders involved increases. While the guns and rakes that we have already developed let us fire two streams of gliders that collide in any of the 71 possible ways illustrated in Table 5.1, as soon as three or more gliders are involved, it can be quite difficult to coordinate them so that they all arrive at the desired time. For example, it would be challenging to implement the ten-glider synthesis of a Gosper glider gun from Figure 5.4 "directly", as it would require us to coordinate the precise positioning and timing of all ten gliders.

To get around this problem, we make use of **incremental synthesis**: we build up complicated patterns by first synthesizing a small stable pattern (like a block or a pond), and then collide only a few gliders with that stable pattern to create a slightly more complicated stable pattern, and so on until we arrive at the pattern we actually want. The main benefit of this type of synthesis is that we do not need to be particularly precise with the timing of the gliders—the different stages of synthesis can take place as many generations apart from each other as we like, so we only need to coordinate a few gliders at a time.

Free PDF. See conwaylife.com/book for pattern files and print version.

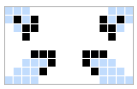
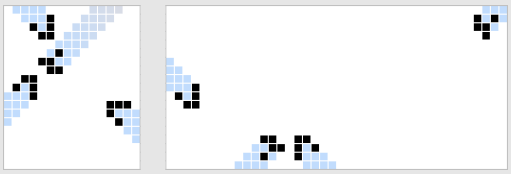
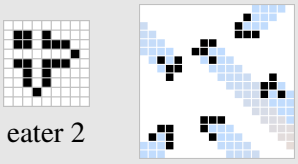
Result	Glider Syntheses	Result	Synthesis
clock		twin bees	
HWSS		eater 2	

Table 5.3: Some useful glider syntheses involving 4 or more gliders. Most of these syntheses have been known for a long time, but the eater 2 synthesis was found by Tanner Jacobi in November 2014 (previously, no synthesis with fewer than 17 gliders was known, though there were syntheses of slight variants of eater 2 with as few as 8 gliders).

As an example, consider the incremental synthesis of the fumarole that is presented in Figure 5.5. While there is a glider synthesis for the fumarole that involves only 7 gliders (see Exercise 5.1), this 12-glider synthesis is somewhat easier to make use of, since each stage of its synthesis requires no more than 3 synchronized gliders.

Incremental syntheses are also advantageous for the fact that we do not have to find ways of synthesizing complicated patterns from scratch, but rather we can keep a catalog of syntheses of various objects, and then to synthesize a new object we can use the most similar still life that we already know how to synthesize as our starting point. For example, the incremental synthesis of the fumarole in Figure 5.5 works by synthesizing a still life that looks similar to the stator of the fumarole (i.e., its stable bottom half), and then the last step of the synthesis creates the fumarole's rotor (i.e., its oscillating top half). This is the most commonly used technique for synthesizing oscillators, and generally billiard table oscillators are much more difficult to synthesize than other oscillators precisely because their rotors are contained within their stators and thus this synthesis technique does not work for them.

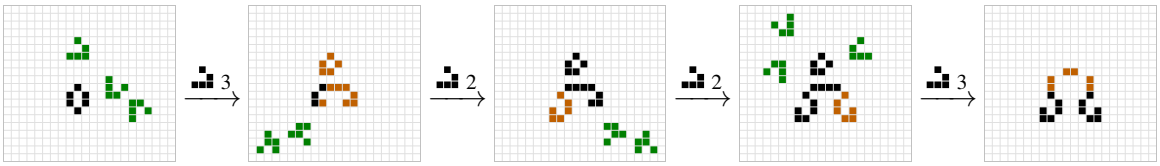


Figure 5.5: A 12-glider incremental synthesis of a fumarole (2 gliders that are not shown are required to synthesize the initial beehive). The cells that were created or modified in the previous step of the synthesis are shown in orange, and the gliders that will be involved in the next collision are shown in green.

By using the reactions listed in Table 5.4, it is straightforward to come up with an incremental synthesis of more complicated objects as well. For example, we can synthesize a Gosper glider gun in a way that never requires more than two synchronized gliders at a time by using incremental syntheses to turn a pond into a ship into a queen bee, as in Figure 5.6. We note that this synthesis uses a total of 12 gliders instead of the 10 gliders used in Figure 5.4, but we consider this a small price to pay for the benefit of not having to synchronize as many gliders in any of the steps of synthesis.

Because incremental synthesis typically works by first constructing a still life or a collection of

Free PDF. See conwaylife.com/book for pattern files and print version.

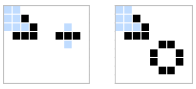
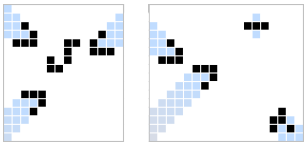
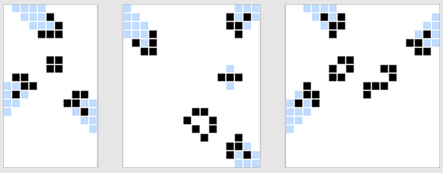
Result	Last Step of Incremental Synthesis
tub	
honey farm	
queen bee	
hat	
ship	
tub with tail	
snake	
T-tetromino / traffic light	
pentadecathlon	
LWSS	
MWSS	
HWSS	

Table 5.4: A selection of useful incremental syntheses that can be used to construct many of the simple Life objects that we have seen. Note that the arrangement of a loaf and a blinker in the HWSS incremental synthesis is the exact arrangement produced by the 2-glider syntheses of a loaf and blinker in Table 5.1.

Free PDF. See conwaylife.com/book for pattern files and print version.

still lifes, and then transforming those still lifes into the desired object, the Life community has gone to great lengths to synthesize still lifes. In fact, syntheses are known for all 190 540 strict still lifes with 20 or fewer live cells.

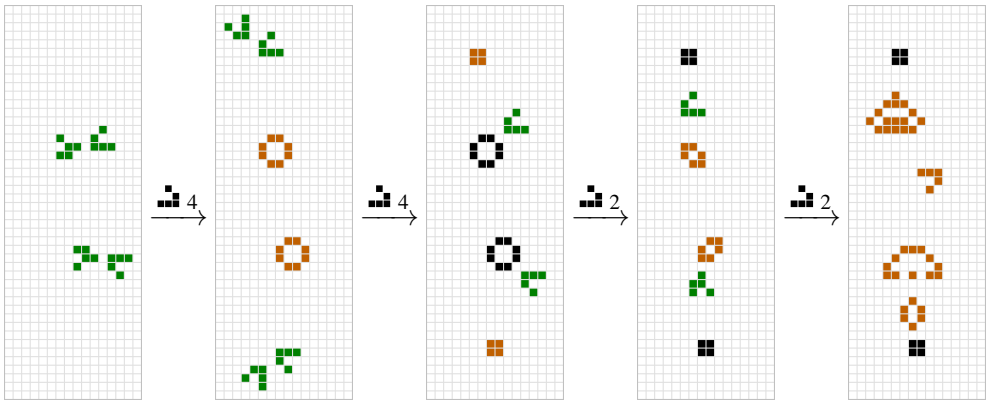


Figure 5.6: An incremental synthesis of the Gosper glider gun. Each stage in the synthesis works by either using two gliders to construct a simple still life or using just one glider to transform one object into another one. The objects that were created or modified in the previous step of the synthesis are shown in orange, and the gliders that will be involved in the next collision are shown in green.

5.4 Synthesis of Moving Objects

Incremental synthesis is most useful when trying to construct objects that are made up of many simpler objects. We already demonstrated this fact when constructing glider syntheses of the Gosper glider gun, and some other perfect examples include many of the spaceships, puffers, and rakes that we introduced in Chapter 4. These objects are mostly composed of many copies of small objects like lightweight spaceships, B-heptominoes, and switch engines that we already know how to synthesize, so we can just synthesize each component individually. However, it can sometimes be tricky to construct the various bits and pieces of these spaceships and puffers with timing that works, so we make these syntheses explicit.

5.4.1 The Ecologist and Space Rake

Recall that the ecologist of Figure 4.26 is made up of three lightweight spaceships and a B-heptomino, so we can synthesize it just by synthesizing those four individual pieces. The only part of the synthesis that is somewhat complicated is the construction of the B-heptomino, since it is not stable without a lightweight spaceship on both sides of it (so we cannot place it first), but there is not enough room for gliders to synthesize it between already-placed lightweight spaceships (so we cannot place it last). One solution is to synthesize the B-heptomino at the same time as the final lightweight spaceship, as in Figure 5.7.

There is another way of synthesizing the ecologist that has the advantages of never using more than two synchronized gliders at a time, and only ever using gliders that come from two of the four possible directions. The way that this synthesis works is to first create four lightweight spaceships (three in the positions that we want and one where the B-heptomino should be), each using a two-glider synthesis of a block followed by the two-glider block-to-LWSS synthesis from Table 5.4. Then we can fire one additional glider at the central lightweight spaceship to transform it into the desired B-heptomino (to actually get this final glider in the proper orientation though, we make use of the 2-glider kickback reaction from Table 5.1). The tricky final stage of this synthesis is made explicit in Figure 5.8.

Free PDF. See conwaylife.com/book for pattern files and print version.



Figure 5.7: An incremental synthesis of the ecologist. The first two steps simply use one of the three-glider syntheses of lightweight spaceships, and the third step uses a two-glider synthesis of a block. The fourth step is slightly more complicated, since we have to construct the B-heptomino at the same time that we construct the final lightweight spaceship below it. The top two gliders form the B-heptomino, while the bottom two gliders and the block construct a lightweight spaceship via one of the incremental syntheses that we saw in Table 5.4.

Once we have constructed the ecologist, it is straightforward to turn it into the space rake or its backward variant just by synthesizing the extra lightweight spaceship in the correct position, as in Figure 5.9 (also see Exercise 5.19).

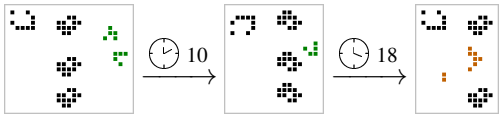


Figure 5.8: Another incremental synthesis of the ecologist. This synthesis has the advantage of never using more than two gliders at a time, and they can all arrive from the lower-left and lower-right (see Exercise 5.18).

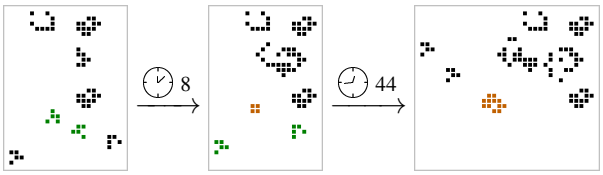


Figure 5.9: An incremental glider synthesis of a (forward) space rake. The same technique can be used to synthesize the backward space rake (see Exercise 5.19).

5.4.2 The Schick Engine and Coe Ship

To create a glider synthesis of the Schick engine from Section 4.4.2, we recall that it is made up of two lightweight spaceships, followed by a spark that is essentially just a T-tetromino (see Exercise 4.22). The various syntheses of lightweight spaceships and T-tetrominoes that we have seen can thus be used together in a straightforward manner to synthesize the Schick engine, as in Figure 5.10. The only difficulty comes from having to carefully choose ways of synthesizing the individual components so that they do not interfere with each other.

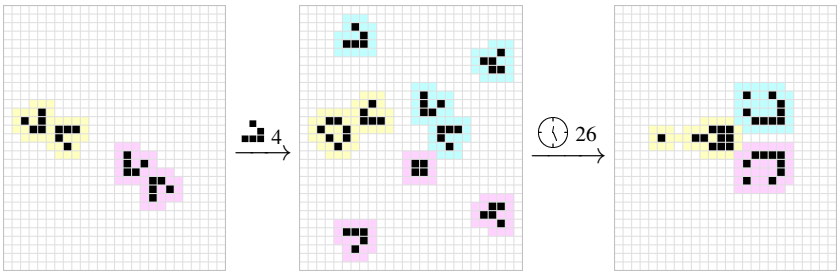


Figure 5.10: A glider synthesis of a Schick engine. Lightweight spaceships are created by the magenta and aqua glider collisions, and a T-tetromino (which becomes the Schick engine's trailing spark) is created by the gliders outlined in yellow.

With this synthesis of the Schick engine in hand, we are also now able to synthesize the period 60 variants of the space rake that we saw in Section 4.4.2—we just synthesize the space rake and then

Free PDF. See conwaylife.com/book for pattern files and print version.

synthesize a Schick engine next to it in the appropriate position, or vice-versa (see Exercise 5.17(b)).

Synthesizing the Coe ship is slightly less straightforward, since we have not yet seen how to synthesize the deformed heavyweight spaceship that makes up half of it. Perhaps the simplest way to synthesize this object is to first synthesize a lightweight and middleweight spaceship next to each other, and then use two additional gliders to transform the MWSS into the deformed HWSS, as illustrated in Figure 5.11.

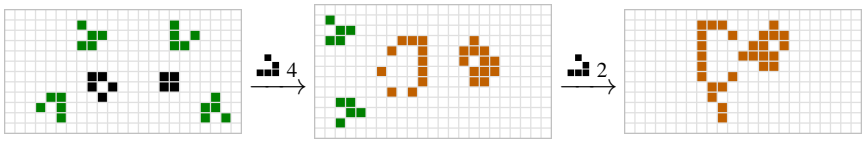


Figure 5.11: An incremental glider synthesis of a Coe ship that works by synthesizing a lightweight spaceship next to a middleweight spaceship, and then transforming that middleweight spaceship into the necessary deformed heavyweight spaceship.

5.4.3 Corderships

Corderships make up another family of objects that are prime candidates for glider synthesis, since they are also made up of several easily synthesized components—switch engines. The difficult part of synthesizing a Cordership is that the switch engines have to be synchronized with each other, making an incremental synthesis difficult. Since each switch engine requires at least three gliders to synthesize, we expect the 10-engine Cordership from Figure 4.23 to require at least 30 gliders to synthesize. Furthermore, the 6 front and middle switch engines will have to be constructed at almost the exact same time as each other.[5]

With these difficulties in mind, it seems desirable to try synthesizing a Cordership that uses as few switch engines as possible—the 2-engine Cordership from Exercise 4.20 or the 3-engine Cordership from Exercise 4.18. Here we focus on the 3-engine Cordership, and we leave the task of synthesizing its 2-engine counterpart to Exercise 5.14.

As expected, it is not too difficult to use $3 \times 3 = 9$ gliders to synthesize the three switch engines that make up this Cordership. However, one small problem arises after we use these 9 gliders: while the Cordership is indeed synthesized correctly, the switch engines take a few generations to sync up with each other, so some unwanted debris is left behind. To prevent this debris from forming, we use 2 extra gliders—one for each of the outermost switch engines—to suppress some chaotic reactions that are normally suppressed by the central switch engine. Altogether, this results in the 11-glider synthesis of the 3-engine Cordership displayed in Figure 5.12.

5.5 Developing New Syntheses

So far the glider syntheses that we have seen have been restricted to two basic types: (a) syntheses that were found just by crashing gliders together and cataloging what objects were created, and (b) syntheses that can be made simply by piecing together the syntheses of type (a). We now branch out and demonstrate how to construct glider syntheses of more exotic objects that do not decompose in any natural way into simpler objects that we already know how to synthesize.

Perhaps the most common way to develop new glider syntheses is to find a soup that leaves behind the pattern of interest in its ash, and try to reverse-engineer the part of the soup's evolution that led to

[5]Despite these obstacles, Stephen Silver created a 60-glider synthesis of the 10-engine Cordership in 1998 (using a 4-glider synthesis of a switch engine, since the 3-glider synthesis was not yet known).

Free PDF. See conwaylife.com/book for pattern files and print version.

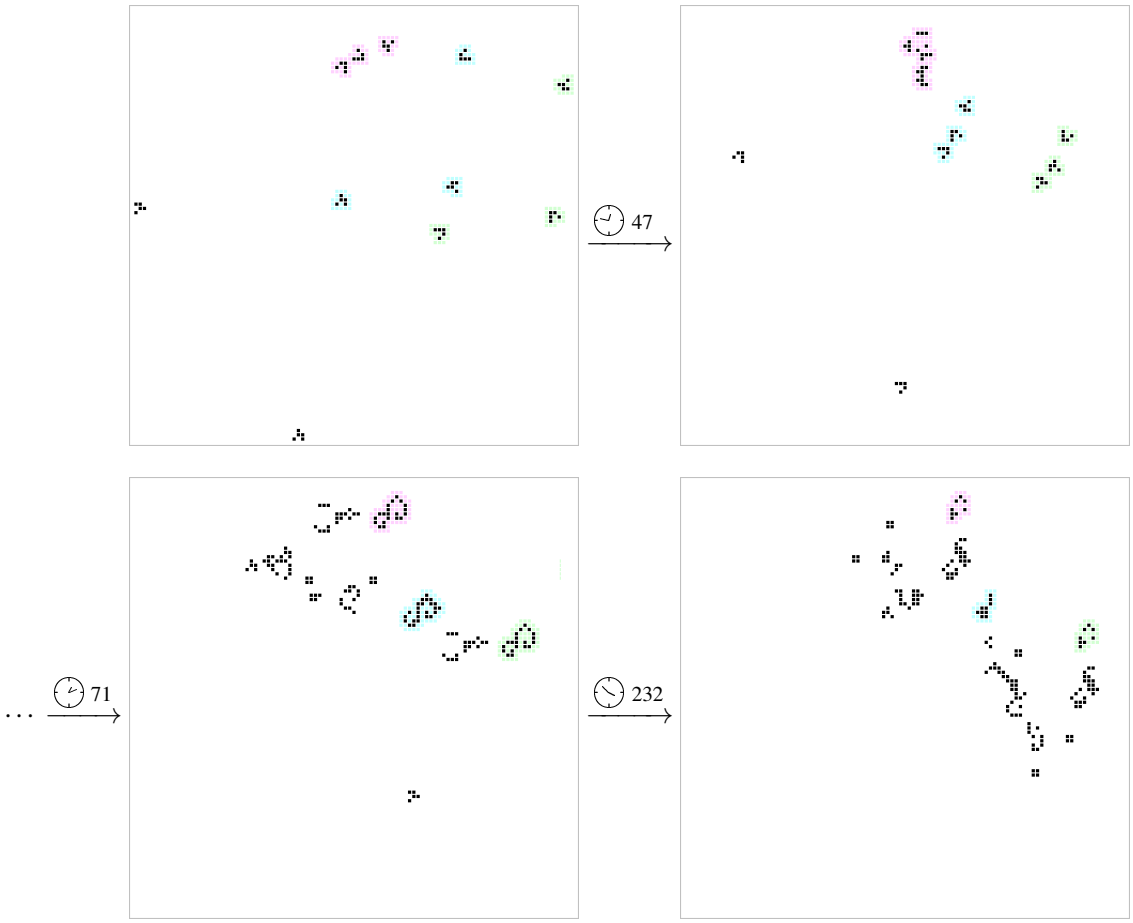


Figure 5.12: An 11-glider synthesis of the 3-engine Cordership. The 3 switch engines are synthesized by the 3-glider syntheses highlighted in aqua, magenta, and green. The remaining 2 gliders clean up some extra junk that would otherwise be left behind by the outermost switch engines.

the object's formation. To illustrate this method, consider Rich's p16 (the period 16 oscillator that we introduced in Figure 3.42), which was found by evolving the soup displayed in Figure 5.13.

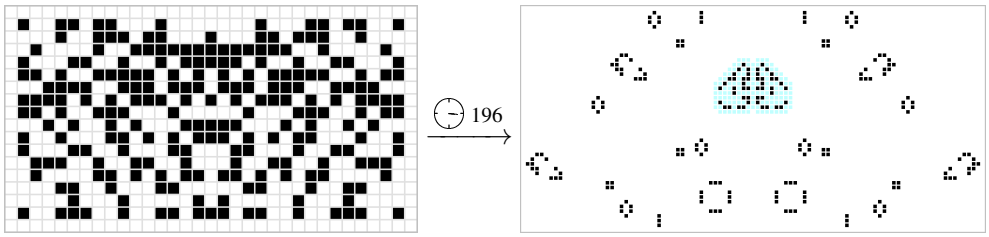
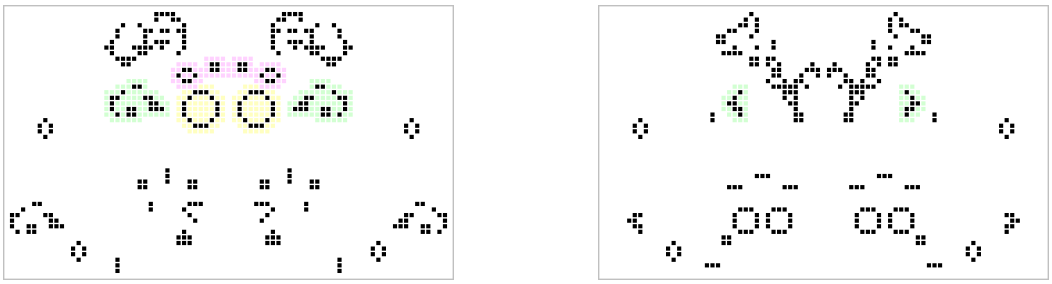


Figure 5.13: The soup that led to the discovery of Rich's p16 (highlighted on the right in aqua).

To develop a glider synthesis for this oscillator, we evolve the soup to just before its formation (in this case, slightly earlier than generation 196) and try to isolate the reaction that creates the oscillator. In this step, it is important to keep in mind that the goal is to go back far enough that all (or at least most) of the reactions are ones that we are familiar with, such as a T-tetromino or a queen bee colliding with some still lifes. In this particular case, if we go to generation 164 of the soup's evolution then the reaction has decomposed enough that we can identify all of the important pieces of the oscillator's

Free PDF. See conwaylife.com/book for pattern files and print version.

creation: Rich's p16 (plus a lot of extra debris) forms when two honey farm predecessors and two B-heptominoes collide in a specific way with two blocks and two beehives (see Figure 5.14).



(a) Generation 164. **(b)** Generation 155.

Figure 5.14: Rich's p16 is created (a) when two honey farm predecessors (highlighted in yellow) and two B-heptominoes (highlighted in green) collide with two blocks and two beehives (highlighted in magenta). The honey farm predecessors can be identified by comparing them with Figure 1.12, and the B-heptomino can be identified by (b) going backward to generation 155 of the soup.

We now have enough tools at our disposal to synthesize this oscillator, since we know how to synthesize blocks, beehives, honey farms, and B-heptominoes. However, the B-heptominoes in this reaction are quite messy, and create a lot of leftover debris that we would then have to clean up with several additional gliders (see Exercise 5.2). To reduce the number of required gliders somewhat, we notice that only three cells on one side of the B-heptominoes interact with the rest of the reaction, and only for two generations. Another easy-to-synthesize object with the same configuration of three cells is the eater 1, which can thus replace the B-heptominoes as in Figure 5.15. Fortunately, using eater 1s in this way does result in a much smaller amount of debris being created around Rich's p16.

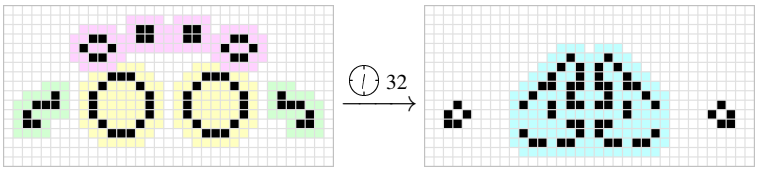


Figure 5.15: A reaction of common objects that results in Rich's p16 (and two unwanted boats). This is the same reaction as in Figure 5.14(a), but with the B-heptominoes replaced by eater 1s (highlighted in green).

The only somewhat tricky part of synthesizing this collection of objects is creating the two honey farm predecessors, since they are unstable and we thus have to create them after creating the still lifes. For this reason, we opt to use the incremental honey farm synthesis from Table 5.4, so that instead of having to use 4 coordinated gliders in the final stage of synthesis (2 for each honey farm), we only need to use 2 coordinated gliders (plus 2 blocks that we can place much earlier). A complete incremental synthesis of this oscillator is presented in Figure 5.16.[6] We stress that there is nothing particularly clever about this synthesis; it just makes use of the 2-glider syntheses from Table 5.1, the incremental honey farm synthesis we already mentioned, and one final glider collision (which is easily found by hand) to destroy the left-over boats.

[6]This synthesis was originally found by Charlie Neder, using the method we outlined in this section, and posted to the ConwayLife.com forums less than a day after the oscillator's discovery.

Free PDF. See conwaylife.com/book for pattern files and print version.

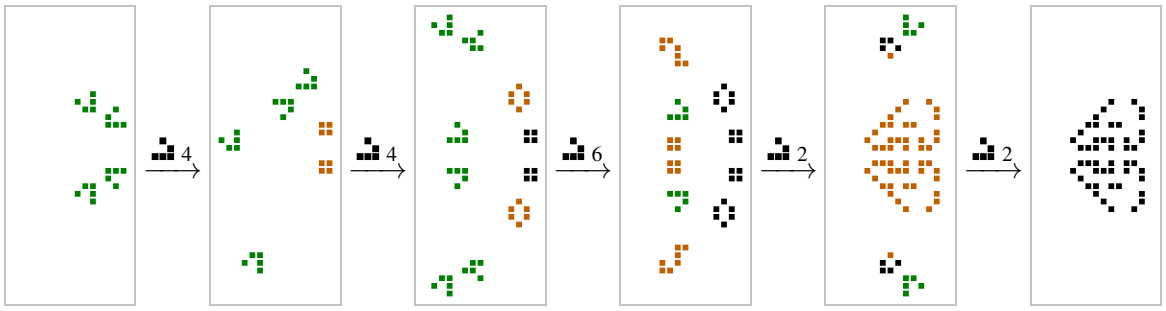


Figure 5.16: An 18-glider incremental synthesis of Rich's p16. The first three stages of synthesis just involve placing still lifes that can each be synthesized by 2 gliders. The next stage synthesizes the honey farm predecessors, and the final stage uses 2 additional gliders to destroy the 2 leftover boats. Objects that were created or modified in the previous step of the synthesis are shown in orange, and the gliders that will be involved in the next collision are shown in green.

5.6 A Gosper Glider Gun Breeder

We have now learned quite a few recipes for synthesizing different objects with gliders. To demonstrate how useful these glider syntheses can be, we now switch focus a bit and start *using* these glider syntheses to create new types of patterns. In particular, we now construct our first **breeder**: an object that grows quadratically (as opposed to objects like guns or rakes, which grow linearly). In other words, we will construct an object that does not just fill up some *strip* of the Life plane, but instead fills an ever-expanding *region* of the Life plane.

The idea behind this breeder is simple enough: we use the rakes that we introduced in Section 4.4 to fire gliders in such a way that those gliders synthesize Gosper glider guns (using the incremental synthesis from Figure 5.6). Since the rakes will create Gosper glider guns at a linear rate, and the Gosper glider guns will then each create gliders at a linear rate, the overall growth of our pattern will be quadratic.

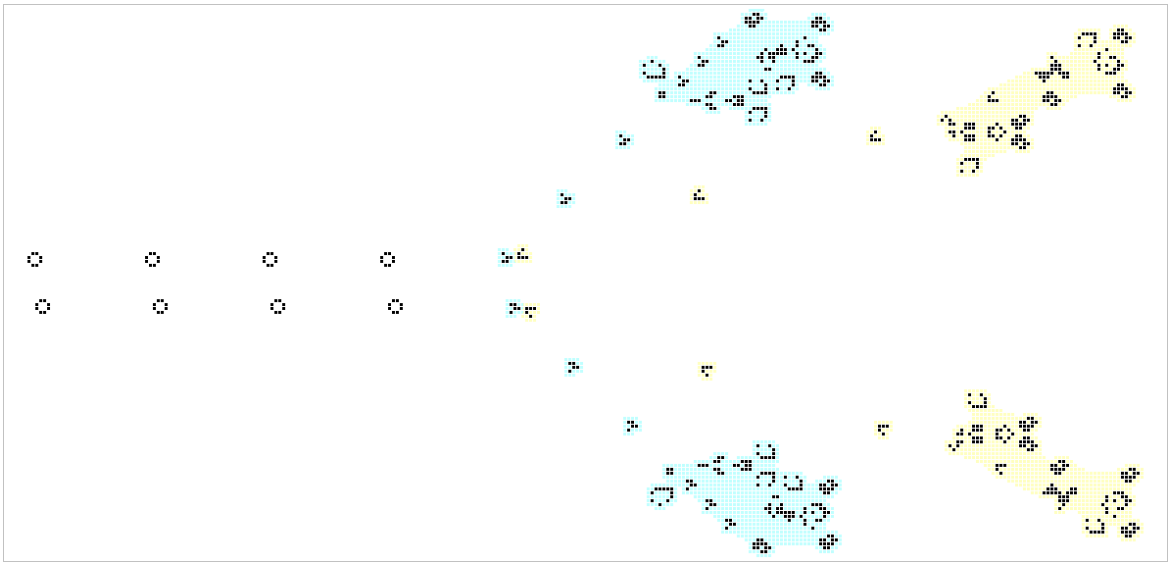


Figure 5.17: The front end of our breeder features two copies of the period 60 backward space rake (outlined in yellow), followed by two copies of the period 60 forward space rake (outlined in aqua), all traveling to the right. The gliders from these rakes collide in such a way as to create ponds, which is the first step of the incremental synthesis of a Gosper glider gun presented in Figure 5.6.

Free PDF. See conwaylife.com/book for pattern files and print version.

To make this construction explicit, we first note that we cannot possibly use the period 20 space rakes directly, since they can only be used to construct objects that are 10 cells apart. Since Gosper glider guns are wider than that, they would collide and destroy each other. We thus make use of the slightly more complicated period 60 variants of the space rake that we introduced in Section 4.4.2, which will produce glider guns that are 30 cells apart from each other.

Now we just go through the Gosper glider gun's incremental synthesis step by step, lining up rakes so that the gliders they produce collide in the desired way. For example, for the first step of the synthesis (i.e., creating the two initial ponds), we need four rakes, which we can position as in Figure 5.17.

Once we have positioned those four rakes, they create two endless lines of ponds with a distance of 30 cells between each consecutive pair. At this point, we simply move on to the next portion of the glider synthesis: we construct two blocks near each pair of ponds, which we can do with four more rakes. We then need to hit the ponds with a single glider each in order to turn them into ships, which we do with two more rakes. Lastly, we use two final rakes to fire gliders at the pair of ships, turning each of them into queen bees and thus completing the Gosper glider gun synthesis. The resulting breeder is displayed in Figure 5.18.[7]

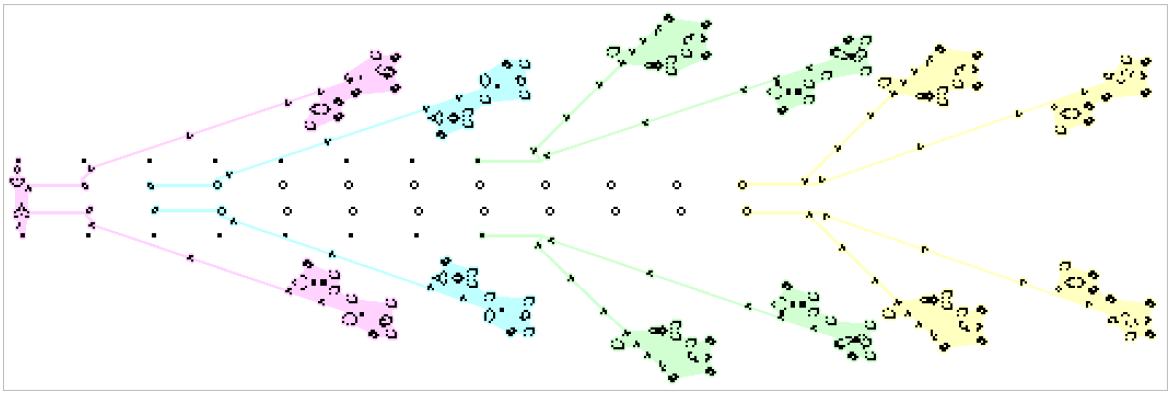


Figure 5.18: A breeder constructed using an incremental glider synthesis of a Gosper glider gun. First, on each side, a forward and a backward p60 space rake collaborate to create a pond as in Figure 5.17 (outlined on the right in yellow), followed by another paired forward and backward space rake creating a block (outlined in green). Next, a backward space rake converts the pond into a ship (outlined in aqua), and finally another backward space rake converts the ship into a queen bee (outlined in magenta), thus completing the synthesis of the Gosper glider gun.

In generation 0 (i.e., its starting phase), this breeder has a population of 2038 cells. The quadratic growth of this breeder can be quantified by noting that every 60 generations, an additional 44-cell Gosper glider gun is created, and each of those guns creates two additional 5-cell gliders, for a total population in generation $60n$ of $5n^2 + 44n + 2038$. This quadratic growth is illustrated in Figure 5.19, which shows the expanding triangular region of gliders behind generation 1200 of the breeder.

While this breeder is quite large, we have purposely not optimized it, in order to make the mechanisms that make it work more apparent. The space rakes can be moved much closer to each other without compromising its functionality (see Exercise 5.23), and by cleverly manipulating the space rake debris it is actually possible to place more than one of the still lifes at the same time, thus halving the total number of space rakes from 12 to 6. We present and make use of a much more compact Gosper glider gun breeder that comes from these ideas a bit later, in Section 6.2.2.

[7] The first breeder was found by Bill Gosper in the early 1970s. It used the exact same ideas that we used in the construction of our breeder, but used a different puffer in place of the space rakes that we used.

Free PDF. See conwaylife.com/book for pattern files and print version.

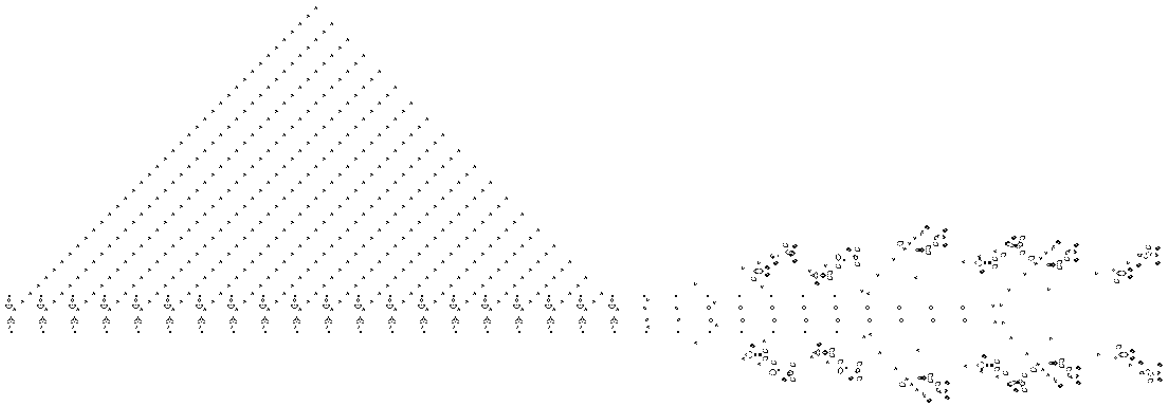


Figure 5.19: After running the breeder for 1200 generations, we see a triangle of gliders form at the west, above the line of Gosper glider guns that has been synthesized. As the breeder moves farther to the east, the triangle continues to expand to the northeast.

5.7 Slow Salvo Synthesis

In Section 5.3, we saw that we can often break glider syntheses down into bits and pieces that are individually easy to understand and make use of. In this section, we take this idea to its most extreme by considering **slow salvos**, which are collections of gliders[8] that (a) are **slow**: only one glider interacts in the synthesis at a time, and (b) form a **salvo**: all of the gliders come from the same direction.

In order for the gliders in a slow salvo to be able to actually synthesize anything, we need another object (called a **seed**) for it to crash into. While we could in principle use any object, it is typical to use a block as a starting point, since it is the simplest and most symmetric stationary object. Furthermore, it is typical to only consider either **p1 slow salvos** or **p2 slow salvos**, which are slow salvos in which the intermediate objects that are created after every glider collision are still lifes or some combination of still lifes and period 2 oscillators, respectively.[9] This perhaps seems like quite an extensive list of restrictions, but we have in fact already seen a few objects that can be created via p1 slow salvos. For example, we saw in Table 5.4 that we could use a pond as a seed in a p1 slow salvo to produce a ship, and then a queen bee.

Despite how restrictive slow salvos seem at first, it is a remarkable fact that they are exactly as general as standard glider syntheses. That is, if we can synthesize an object with gliders at all then we can synthesize it with a p2 slow salvo.[10] In order to pin down exactly why this is the case, we now introduce several new reactions that allow us to systematically create slow salvo syntheses from standard glider syntheses.

5.7.1 Creating and Moving Blocks

Our first step toward showing that slow salvos can construct anything that regular glider syntheses can is to demonstrate how we can create almost any arrangement of any number of blocks in the Life plane that we like. The first reaction that we will need is the one displayed in Figure 5.20, which uses a slow salvo of 2 gliders to turn a single block into two blocks.

While the two resulting blocks are in very different positions than the initial block, it turns out

[8] Technically, a salvo can be made up of any spaceships, but the term almost always refers to gliders.

[9] P2 salvos are advantageous because blinkers are so easy to synthesize, and we will see shortly that the clock is also a useful tool when working with slow salvos. We could analogously consider pn slow salvos for some $n \geq 3$, but in practice not much is gained by doing so, since objects with period 3 or higher typically aren't any easier to create in intermediate stages of synthesis than objects of period 1 or period 2.

[10] However, the slow salvo might contain considerably more gliders than other glider syntheses.

Free PDF. See conwaylife.com/book for pattern files and print version.

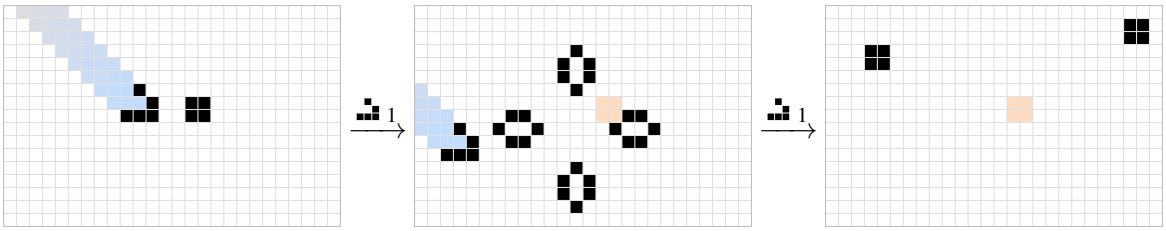
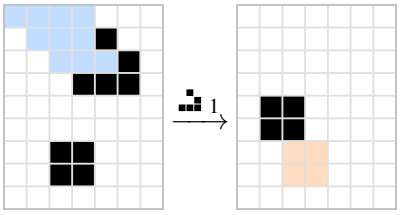


Figure 5.20: A 2-glider slow salvo can turn one block into two blocks.

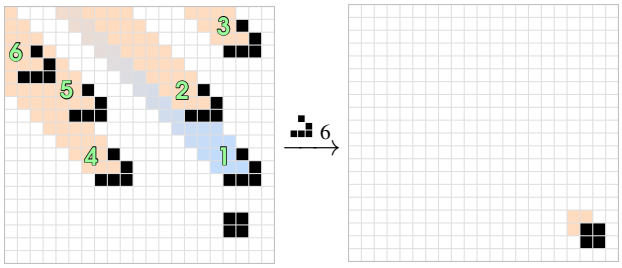
that this does not particularly matter, as we can also use p1 slow salvos to move a block from one position on the Life plane to any other. To show that such a movement is possible, we use the two block-moving reactions displayed in Figure 5.21. The first of these reactions uses a single glider to move a block up 2 cells and left 1 cell (in fact, this is exactly the (2,1) block pull that we saw way back in Figure 2.20), and the second reaction uses six gliders (in a p1 slow salvo) to move a block down and to the right 1 cell.

If we repeat these two reactions, then we can move a block a single cell up, down, left, or right. For example, to move a block up a single cell, we could perform the movement in Figure 5.21(a), followed by the movement in Figure 5.21(b) (for a total cost of 7 gliders). Similarly:

- To move a block right by a single cell, we could perform the movement in Figure 5.21(a), followed by the movement in Figure 5.21(b) twice.[11]
- To move a block left by a single cell, we could perform the movement in Figure 5.21(a), but reflected along the diagonal from the top-left to the bottom right. Then perform the movement in Figure 5.21(b).
- To move a block down by a single cell, we could first move it left by a single cell and then perform the movement in Figure 5.21(b).



(a) The (2,1) block pull is a reaction in which a single glider pulls a block by 2 cells in one direction and 1 cell in the other.



(b) A 6-glider slow salvo moving a block down and right by 1 cell.

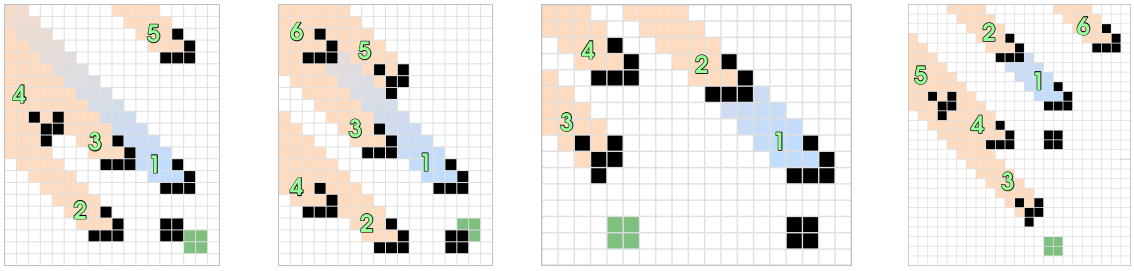
Figure 5.21: Two methods of moving a block around with p1 slow salvos. The order in which the gliders should come in is indicated by the green numbers—their exact timing does not matter, as long as they are far enough apart that each collision settles down before the next glider arrives. Repeating these slow salvos allows us to move a block to any position on the Life plane.

Now that we know how to use p1 slow salvos to move a block by a single cell in any direction, we can easily repeat these reactions over and over until the block is moved to whatever position we

[11] This requires a total of 13 gliders, which is far from optimal—another p1 slow salvo is known that accomplishes the same movement via only 7 total gliders. However, we are just interested in presenting the conceptually simplest method of moving blocks, not necessarily the most efficient one. Paul Chapman and Dave Greene put together an extensive table of the cheapest known block-moving slow salvos, which is available at b3s23life.blogspot.com/2004/09/glues-slow-salvo-block-move-table.html (note that the values in their table assume that the gliders come from the bottom-right, rather than the top-left).

Free PDF. See conwaylife.com/book for pattern files and print version.

desire. However, there are a few other block-moving slow salvos that are worth making note of, since they help us move blocks around the plane a bit more quickly. In particular, if we are willing to make use of p2 salvos instead of just p1 ones, then we gain access to the salvos displayed in Figure 5.22. There are hundreds of salvos of this type known.[12] It is worth noting that the **(2,1) block push** of Figure 5.22(a) *pushes* the block by the same amount that the $(2,1)$ block pull of Figure 5.21(a) pulls it.



(a) $(2,1)$ block push. **(b)** $(1,-1)$ block move. **(c)** $(11,0)$ block pull. **(d)** $(11,0)$ block push.

Figure 5.22: Some p2 slow salvos that move a block by various amounts. The output location of the block is highlighted in green.

5.7.2 One-Time Turners

Just like we can use gliders to create and move blocks around the Life plane, we can also use blocks to move gliders around the Life plane. We can thus use a p1 slow salvo to create arrangements of blocks that then change the direction of other gliders in the salvo, thus creating salvos that come from multiple directions.

To be a bit more explicit, consider the arrangement of blocks presented in Figure 5.23, which can be used to rotate a glider by 90 degrees, but is destroyed in the process (contrast this with reflectors like the Snark, which rotate gliders but remain unchanged). Reflectors like this are called **one-time turners**, and they are one of the key ingredients that let us emulate arbitrary glider syntheses via slow salvos. There are also numerous other 90-degree one-time turners involving various numbers of blocks (and sometimes other small still lifes—see Exercise 5.31), but this turner involving just two blocks is perhaps the simplest, and for now it is enough for our purposes.

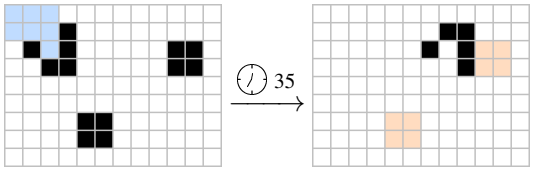


Figure 5.23: A **one-time turner** composed of two blocks that rotates a glider by 90 degrees and destroys the blocks at the same time.

Importantly, because this one-time turner is made up of nothing other than blocks, we can construct it using a p1 slow salvo via the techniques that we developed in the previous section. An explicit 12-glider p1 slow salvo that does the job (and also leaves us with an extra block that we can then use to construct additional turners) is displayed in Figure 5.24. This salvo works by using 2 gliders to duplicate the initial block (as in Figure 5.20), 8 gliders to move one of these blocks to a new position, and then 2 more gliders to duplicate this block again.

[12] A collection of them is available at conwaylife.com/forums/viewtopic.php?p=8182#p8182.

Free PDF. See conwaylife.com/book for pattern files and print version.

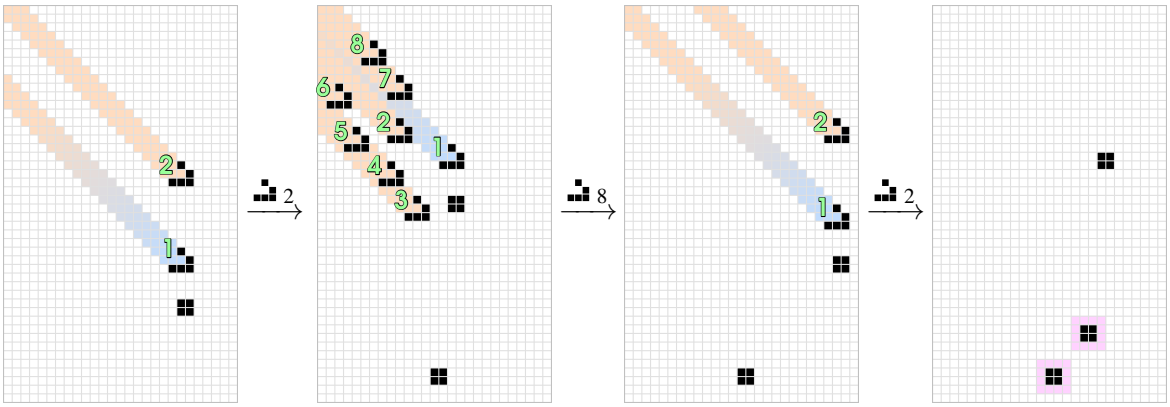


Figure 5.24: A 12-glider p1 slow salvo creating the one-time turner from Figure 5.23 (highlighted in magenta) and leaving behind another block to create additional turners or other objects with. The first 2 gliders duplicate the initial block via the reaction of Figure 5.20, the next 8 gliders move one of the blocks down and to the right, and then the final 2 gliders duplicate that moved block.

We can thus use a total of 13 gliders in a p1 slow salvo (12 to create the one-time turner and 1 to use and destroy it) to create 1 glider in a perpendicular direction, while still having a block left over to perform additional constructions with. Using this technique, we are now able to emulate multi-directional glider syntheses via unidirectional syntheses.

For example, consider the four-glider syntheses of the clock that we saw in Table 5.3, which each use one glider coming from all four possible directions. A unidirectional synthesis of the clock can be constructed by using a p1 slow salvo to construct four one-time turners—one of the gliders does not need to be turned at all, two gliders need to be turned once each, and one glider needs to be turned twice—and then firing four gliders at the one-time turners with the proper timing. An explicit configuration of one-time turners that works is displayed in Figure 5.25.[13]

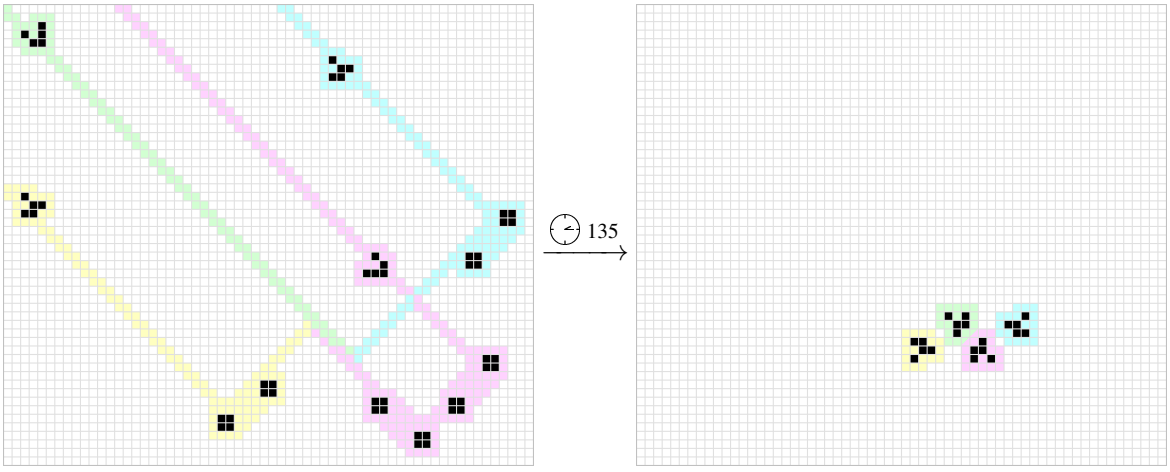


Figure 5.25: One-time turners (which can be synthesized with a p1 slow salvo) can be used to allow unidirectional glider waves (like the one on the left) to emulate multi-directional glider syntheses (like the one on the right). The gliders on the right are in exactly the position of the clock synthesis that we saw in Table 5.3.

[13] It is possible to create a p1 slow salvo that builds this arrangement of blocks by hand, but there are also computer scripts that automate this process—see Exercise 5.38.

Free PDF. See conwaylife.com/book for pattern files and print version.

Unfortunately, this construction does not give us a true p1 slow salvo for constructing a clock, since the final four gliders in the synthesis (i.e., the ones that hit the one-time turners) must be synchronized with each other—we are not free to space them arbitrarily far from one another. In order to fix this problem, we need to introduce some methods for using blocks to duplicate gliders and adjust their timing.

5.7.3 Splitters and Timing

The first ingredient that we need in order avoid having to send synchronized gliders at our one-time turners is a method of turning one glider into many gliders. Just like our one-time turner, we would like this pattern to be composed entirely of blocks, and we would like it to be cleanly destroyed during the glider-duplicating reaction. Patterns with these properties are called **blockic splitters**,[14] and one example is provided in Figure 5.26.

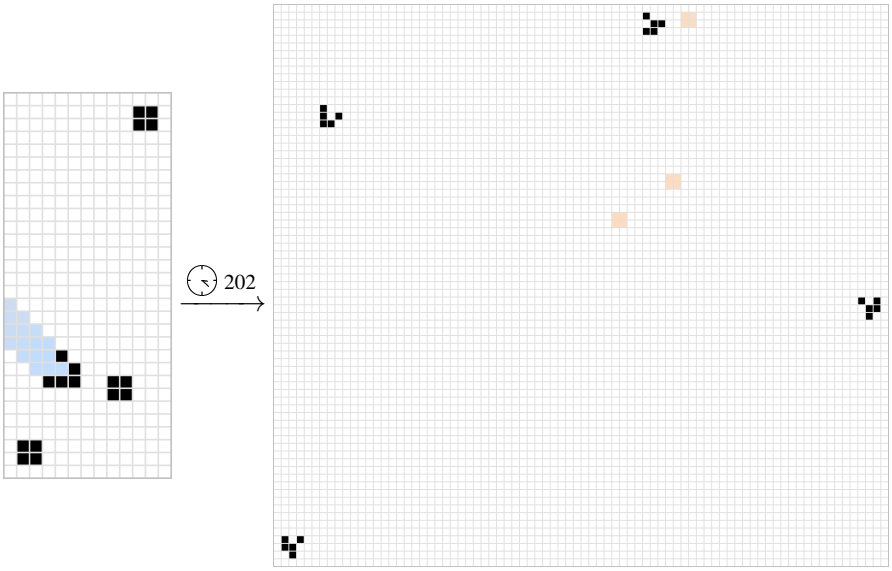


Figure 5.26: A **blockic splitter**, which uses three blocks to turn one glider into four gliders (while destroying the blocks in the process).

This splitter *almost* gets us what we need—it lets us turn one glider into four, and we could chain multiple splitters together to turn one glider into as many as we desire. We can use one-time turners to move these gliders around the Life plane, with the intention of emulating multi-direction glider syntheses, but unfortunately we are faced with two brand new problems:

- The one-time turner that we introduced in Figure 5.23 preserves the glider's color, so we have no way of obtaining glider color combinations other than those provided to us by the splitter in Figure 5.26 (in which the bottom-left glider has the same color as the input glider, and the other 3 gliders have the opposite color). Since the 4-glider synthesis of a clock uses 2 gliders of each color, in order to make a slow salvo synthesis of the clock we would have to use this splitter multiple times and then use blocks to delete the excess gliders, which is quite wasteful.
- The one-time turner from Figure 5.23 gives us no control over timing, and we need some way of making sure that all of the desired gliders in a given synthesis not only arrive at the right place, but also at the right time.

[14] The term **blockic** refers to the fact that it is composed entirely of blocks. There are also splitters (and one-time turners) made up of other still lifes (or even blinkers), but we do not consider them here since blocks are enough for our purposes.

Free PDF. See conwaylife.com/book for pattern files and print version.

Fortunately, there is a common solution to both of the above problems, and it is simply to introduce several new one-time turners. In particular, the numerous 180-degree one-time turners presented in Table 5.5 are capable of either preserving or changing the color of a glider, and also offsetting its timing by any amount that we desire.

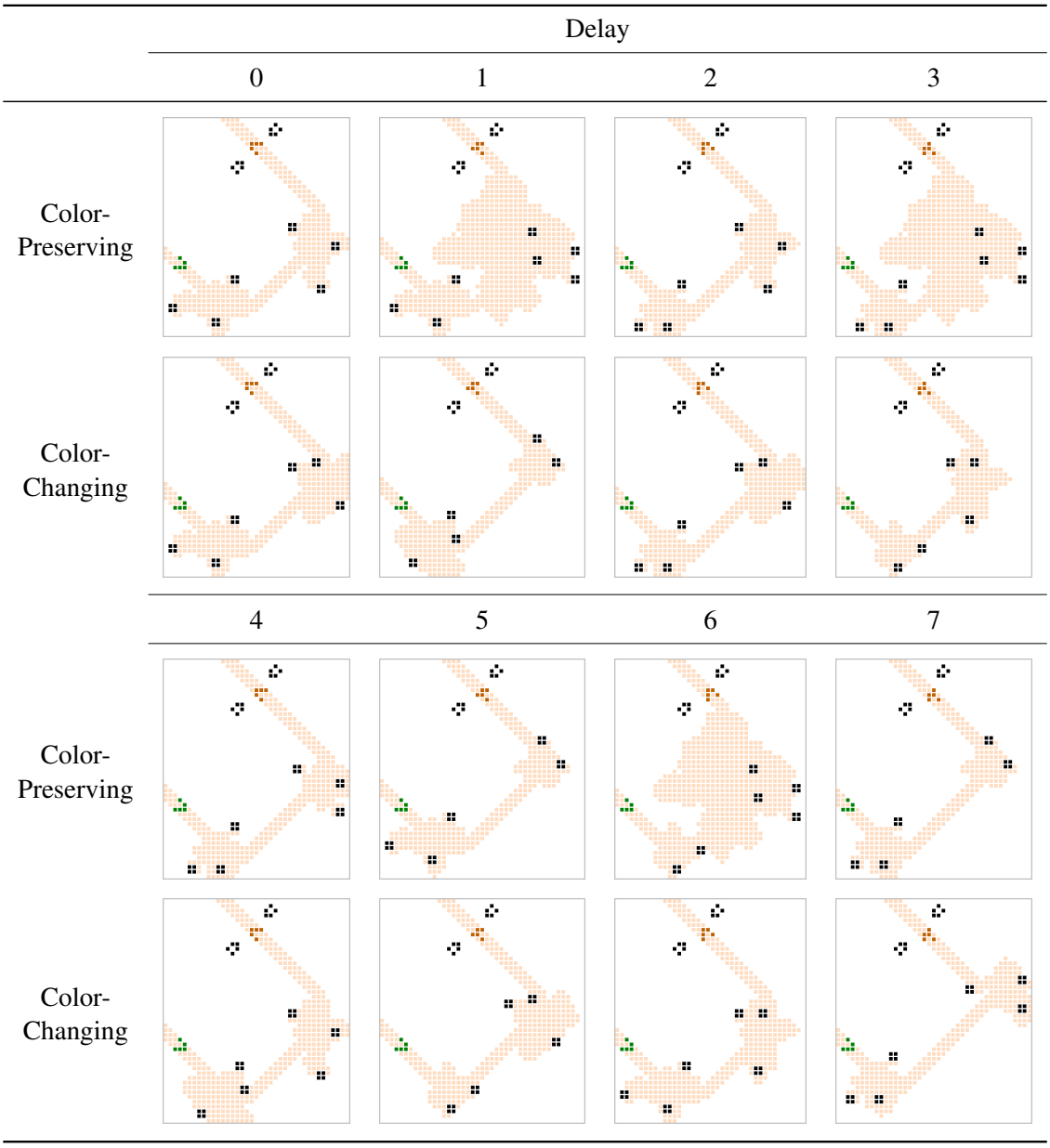


Table 5.5: By using different 90-degree one-time turners together, we can create this collection of 180-degree one-time turners that lets us set a glider to any timing and color. In all cases, the input glider is highlighted in green and comes in from the top-left, while the location of the output glider exactly 200 generations later is highlighted in orange. The boats are not required for the turners to function, but just serve to make it easier to line up multiple different turners.

Free PDF. See conwaylife.com/book for pattern files and print version.

In order to make use of these 180-degree one-time turners to put gliders in (almost) any *position* that we want, we first simply use copies of the 90-degree one-time turner from Figure 5.23 and the 180-degree one-time turners with offset 0 from Table 5.5. Importantly, we always place at least one 180-degree turner in the path of each glider, even if it is not required to get its positioning right (we will need that 180-degree turner to get the timing right momentarily).

Even though the positioning of the gliders is now correct, their timing will likely be horribly wrong. To fix this problem, we now focus on the glider that gets to its destination last—we will synchronize all of the other gliders with this one. To slow down those other gliders, we note that moving a 180-degree turner 1 cell farther away causes the glider to reach its destination 8 generations later (it now has to travel 1 cell, and thus 4 generations, farther in both directions). To delay a glider by n generations, we can thus move its 180-degree turner farther away by $\lfloor n/8 \rfloor$ generations and then replace it by the turner from Table 5.5 that has the same effect on its color but has offset n mod 8.

We now have a complete set of glider-preserving or glider-changing turners capable of delaying a glider by any of $0, 1, 2, \ldots, 7$ generations. So we have all the tools we need to position *and* time gliders as we see fit. An example of how we can use the 4 gliders produced by the splitter in Figure 5.26 to synthesize a clock is presented in Figure 5.27. Note that we placed a 180-degree one-time turner in the path of each of the four gliders so that we could use the technique for correcting their timing described in the previous paragraph, and since each of those 180-degree turners is simply made up of two 90-degree one-time turners, we are able to separate their two halves in order to more easily control the glider's output position.

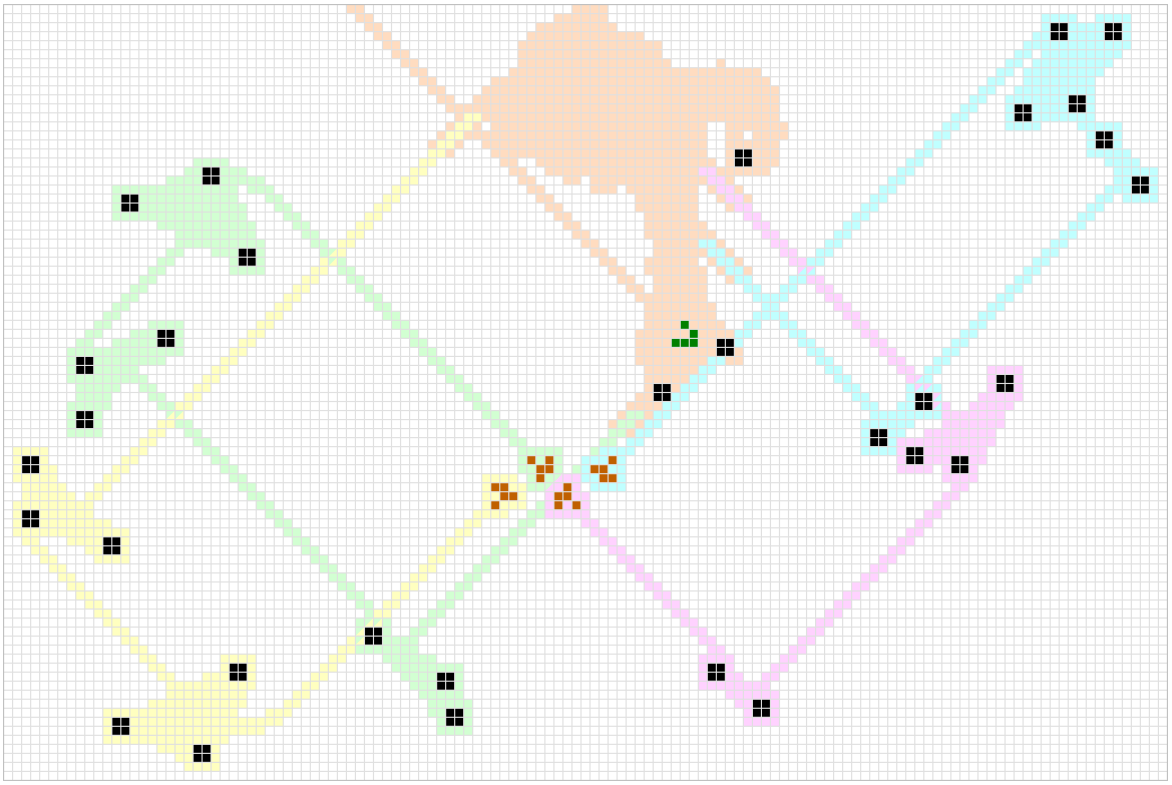


Figure 5.27: A **blockic seed** for a clock: The blockic splitter at the top-center (highlighted in orange) turns the single incoming glider into four gliders whose paths are outlined in yellow, green, aqua, and magenta. Those gliders are repeatedly reflected via one-time turners in such a way that they end up in the indicated positions required to synthesize a clock after 537 generations. We could probably get away with using fewer one-time turners, but by placing a 180-degree turner in the path of every glider it is much easier to get their timing right.

Free PDF. See conwaylife.com/book for pattern files and print version.

A configuration of simple still lifes, like the one in Figure 5.27, that synthesizes a particular object when hit by a glider (or gliders) is called a **seed** (and a seed made up entirely of blocks is called a **blockic seed**). Seeds are particularly useful since they immediately give us a p1 slow salvo for constructing an object—a clock in this case—since we "just" have to use a p1 slow salvo to construct the still lifes in the seed and then fire one additional glider to trigger the synthesis.

However, actually building a slow salvo synthesis for a seed is a tedious process. The 31-block seed for a clock from Figure 5.27 requires hundreds of slow gliders to construct (see Exercise 5.39)—recall that we already needed 12 gliders just to create a single 2-block one-time turner in Figure 5.24. For this reason, slow salvo syntheses of seeds are typically built by computer scripts, with *slsparse* (see `conwaylife.com/wiki/Slsparse` for tutorials and a download link) being the most widely used.

5.7.4 Tight Packings of Gliders

There is still one final problem that might arise when trying to emulate an arbitrary glider synthesis via a slow salvo synthesis, and that is the fact that some syntheses involve packings of gliders coming from the same direction that are too close together for us to place with our one-time turners. This was not a problem for the clock synthesis in Figure 5.27 since each of the gliders in the synthesis comes from a different direction. However, if we were to try to use a p1 slow salvo to emulate the 3-glider synthesis of an HWSS displayed in Table 5.2, it is not obvious that we can position the two gliders that are heading southwest close enough to each other (since one of the gliders might interfere with any one-time turner that we try to use to position the other glider). To fix this glider-packing problem, we introduce one final reaction: the **clock inserter**,[15] which uses two opposing gliders to transform a clock into a perpendicular glider, as in Figure 5.28.

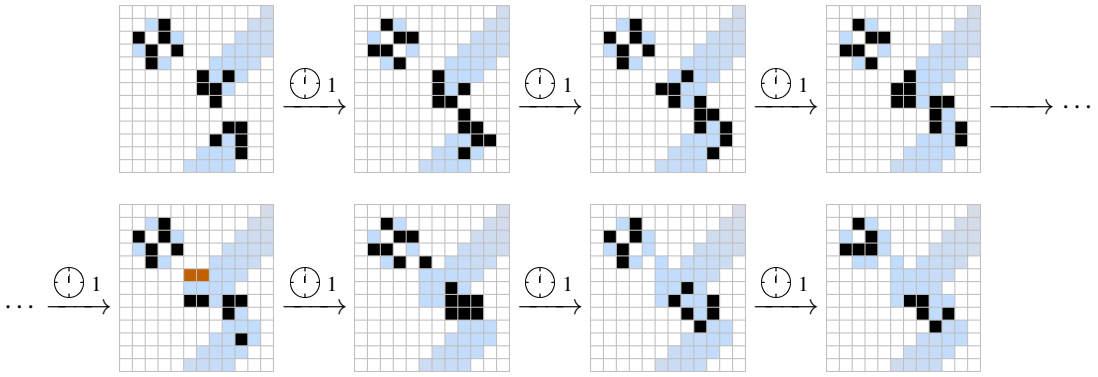


Figure 5.28: The **clock inserter** is a reaction that very cleanly collides two gliders in such a way as to transform a clock into a perpendicular glider. The two-glider collision produces a domino spark in generation 5 (shown in orange) and then dies off, while the domino spark initiates the clock-to-glider transformation. The debris at the bottom-right in generation 8 dies off in 5 more generations.

The key facts that make this reaction so useful for us are (1) that the output glider appears at a small offset and moves through the space just vacated by the colliding input gliders, and (2) the reaction is almost effortless—it does not disrupt any other cells around the clock and output glider, and thus does not disrupt other nearby gliders either. This reaction can thus be used to place a glider very close in front of, or to the side of, another already-placed glider (see Figure 5.29 for an illustration of how closely it can place a new glider to other gliders). Because of this, we can use this reaction to build any arrangement of gliders, no matter how tight, as long as we start by placing the gliders in the

[15] Found by Martin Grant in December 2014.

Free PDF. See conwaylife.com/book for pattern files and print version.

back first. Actually *proving* that the clock inserter reaction can be used to build any arrangement of gliders is somewhat technical and messy,[16] since there are many possible ways for gliders to be near each other, so we defer the proof to Appendix B.1.

With this reaction in hand, we can now use a p2 slow salvo to construct any arrangement of gliders in the plane that we desire. First, we "rewind" the desired glider synthesis back as far as we like, so that it consists of four salvos of gliders—one coming from each direction.[17] We then construct the four salvos one glider at a time. If the gliders in a salvo are spaced sufficiently far apart, we can insert gliders into it via one-time turners. If the gliders are spaced close together, we instead first synthesize a clock (via p1 slow salvo synthesis, such as in Figure 5.27) and then use the clock inserter to insert the close gliders one at a time. We have thus finally proved the following theorem:

Theorem 5.1 — Universality of p2 Slow Salvo Synthesis
Every pattern that can be constructed via glider synthesis can be constructed by a p2 slow salvo glider synthesis.

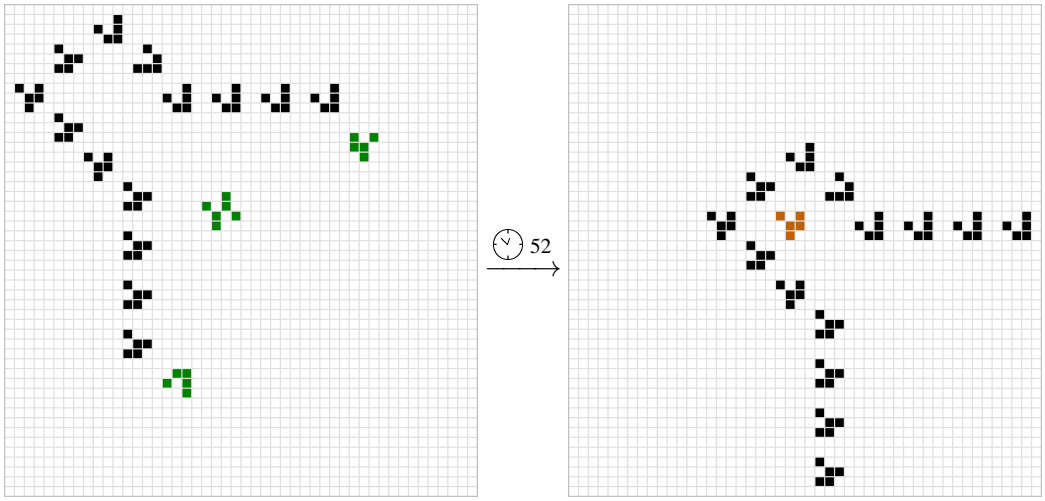


Figure 5.29: A demonstration of how the clock inserter (shown in green on the left) can be used to place a glider (shown in orange on the right) closely beside and in front of other already-placed gliders.

The only p2 object that we needed to prove Theorem 5.1 was the clock, which we used to place clusters of tightly packed gliders. We can get around having to use clocks (and thus avoid p2 objects altogether) by instead using the blockic seed of a clock from Figure 5.27. More specifically, if we wish to use a clock insertion reaction, we first split a single glider into three via blockic splitters—one of those gliders then hits the seed, creating a clock, and the other two are reflected so as to trigger the clock insertion reaction as in Figure 5.28. Since this entire process is carried out by a single glider in the slow salvo synthesis, the result is a p1 slow salvo synthesis. We thus have the following strengthening of Theorem 5.1:[18]

[16] Chris Cain wrote a script that automatically uses clock inserters to build glider arrangements in 2014, and then he and Dave Greene used the script to build such a wide array of tight glider arrangements that this could probably be considered the first proof that p2 slow salvos can build any configuration of gliders. Cain's script can be found at `conwaylife.com/forums/viewtopic.php?p=15133#p15133`.

[17] Recall that the gliders must be able to reach their positions from arbitrarily far away in order to be considered a valid glider synthesis.

[18] There are some mild technicalities that we have glossed over here—see Exercise 5.41.

Free PDF. See conwaylife.com/book for pattern files and print version.

Theorem 5.2 — Universality of p1 Slow Salvo Synthesis
Every pattern that can be constructed via glider synthesis can be constructed by a p1 slow salvo glider synthesis.

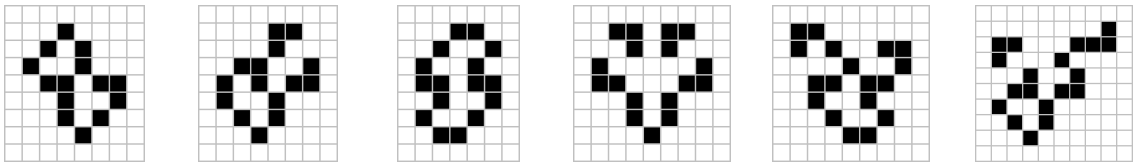
5.8 Notes and Historical Remarks

The importance of glider synthesis was known essentially as soon as the glider itself was found in 1970, with common folklore being that we could send gliders as signals throughout the Life plane and collide those gliders in different ways to simulate arbitrary computations. This basic idea has been refined and made more precise repeatedly over the past 50 years, to the point that there are now explicit patterns that do exactly this—they collide gliders so as to perform arbitrary computations and build almost any pattern of our choosing (we will delve deeply into the specifics of how these patterns work in Chapters 9 and 11).

The first specific and explicit uses of glider syntheses were demonstrated in 1971, when Bill Gosper constructed the first breeder (essentially the breeder that we built in Section 5.6) as well as the first lightweight and middleweight spaceship guns. These patterns demonstrated the kind of leap in complexity that is possible when taking advantage of glider synthesis, and it led to a surge in interest in the topic over the following years.

By 1973, the majority of "basic" Life objects were synthesizable, including lightweight, middleweight, and heavyweight spaceships, switch engines (and their block-laying and glider-producing counterparts), commonly occurring oscillators like the pentadecathlon and pulsar, and all still lifes and oscillators with 7 or fewer cells other than the clock and the long snake. Syntheses of larger composite patterns were even being discovered by this point, with Douglas Petrie constructing the 11-glider synthesis of the Schick engine displayed in Figure 5.10 in 1973. The majority of these early syntheses were developed by David Buckingham, Mark Niemiec, and Douglas Petrie.

Over the following decades, David Buckingham continued to develop syntheses for still lifes and oscillators, and he completed syntheses of all of them with 14 or fewer cells by no later than 1992. His technique was to make heavy use of incremental synthesis, building the objects from the inside out, only using a couple of gliders at a time to tweak the outermost portion of the object that he was synthesizing. This method works very well when synthesizing objects that have "end pieces" like tails that can be removed or altered without affecting the pattern's stability. However, compact still lifes like the one in Figure 5.30(a) are much more difficult to construct, and this particular still life (which was the last 14-cell still life to be synthesized) was initially synthesized using a massive incremental synthesis involving more than 30 gliders.



(a) 14 cells. **(b)** 15 cells. **(c)** 16 cells. **(d)** 17 cells. **(e)** 18 cells. **(f)** 19 cells.

Figure 5.30: The final still lifes with 14–19 cells to be synthesized by gliders.

Mark Niemiec continued on with this work, creating a large database that he used to automatically generate syntheses of thousands of still lifes by piecing together known reactions [Nie03, Nie10]. This greatly reduced the number of still life syntheses that needed to be found by hand, and in 2013 he, with help from Martin Grant, completed syntheses of all 15-cell still lifes (as well as syntheses for all except for a few hundred 16-, 17-, and 18-cell still lifes).

Free PDF. See conwaylife.com/book for pattern files and print version.

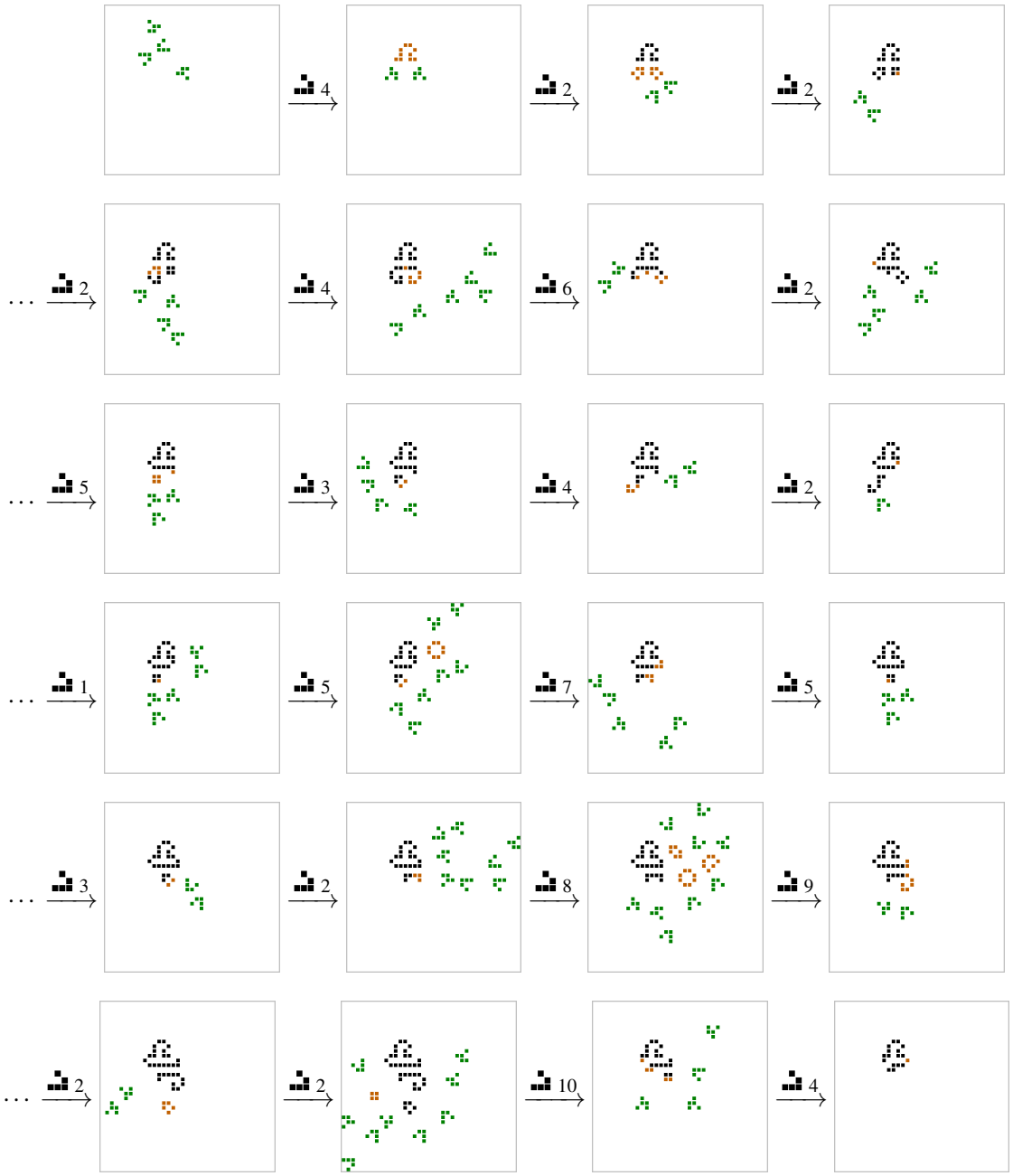


Figure 5.31: A summary of a massive 94-glider incremental synthesis of a hard-to-construct 17-cell still life (shown at the bottom right). Each stage in the synthesis works by using a glider collision to transform one still life into another. The cells that were created or modified in the previous step of the synthesis are shown in orange, and the gliders that will be involved in the next collision are shown in green.

Glider syntheses for the remaining 16-cell still lifes were then found via a collaborative effort on the ConwayLife.com forums, mostly led by Martin Grant, Matthias Merzenich, and Mark Niemiec, with the final synthesis (see Figure 5.30(c)) being completed in January 2014. Another five-month collaborative effort, led by the same group of people, completed syntheses of the 17-cell still lifes in May 2014. A comparatively quick two-month effort finished the remaining syntheses of 18-cell still lifes in November of that year, and then another four months of work resulted in the synthesis of the last 930 or so 19-cell still lifes in February 2020. Finally, the community worked for about nine

Free PDF. See conwaylife.com/book for pattern files and print version.

months to synthesize the remaining 1 000 or so 20-cell still lifes by March 2021.

To give an idea of the size of this achievement, recall from Table 2.1 that there are 190 540 different strict still lifes with 20 or fewer cells. Not only is this a huge number of distinct objects to synthesize, but the syntheses themselves are often monstrously large. For example, Figure 5.31 shows how one of the "problematic" 17-cell still lifes was constructed by using a whopping total of 94 gliders and over 20 stages of incremental synthesis.

Along with pushing these techniques to synthesize all still lifes with 21 live cells, a parallel project involves trying to reduce the number of gliders required to synthesize these objects. It has been known for decades how to synthesize all still lifes with 8 or fewer cells via 4 or fewer gliders. Similarly, glider syntheses are known for constructing every still life with $9, 10, 11, 12$, and 13 live cells via $5, 5, 7, 7$, and 8 or fewer gliders, respectively.[19] Recent efforts have attempted to continue this pattern and synthesize all small still lifes in fewer than 1 glider per live cell. This project was completed for 14-, 15-, 16-, and 17-cell still lifes in October 2016, November 2016, May 2017, and September 2019, respectively. For example, we now know how to construct the 17-cell still life from Figure 5.31 (which was first synthesized by 94 gliders) via just 10 gliders—see Figure 5.32.

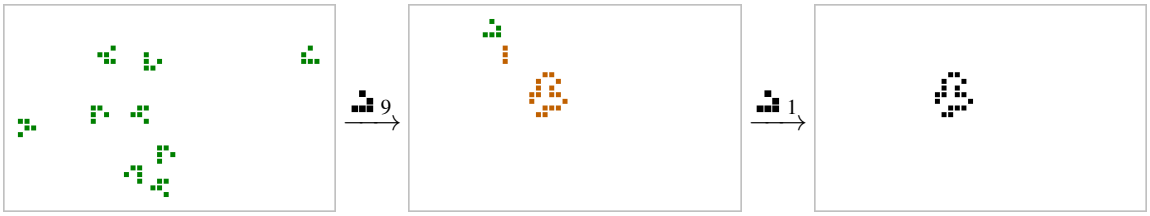


Figure 5.32: A 10-glider synthesis of the 17-cell still life from Figure 5.31 that was constructed with the help of a soup that was found by apgsearch.

Remarkably, it was even shown in September 2020 that every glider-synthesizable pattern (regardless of its size) can be synthesized via 17 or fewer gliders.[20] However, it turns out that there are patterns that are not glider-synthesizable in the first place. In January 2022, Ilkka Törmä and Ville Salo found a configuration of live and dead cells with the property that, if it occurs at any generation, then it must occur at the same location in all previous generations. There is a 306-cell still life that contains this configuration of cells, and thus cannot be constructed by glider synthesis or any other method.[21]

The largest database of glider syntheses that is currently maintained, *Shinjuku*,[22] contains hundreds of thousands of the cheapest-known ways of using gliders to convert one object into another, and thus the cheapest-known ways of synthesizing them via incremental synthesis.

[19] It was not known how to synthesize the 9-cell long^3 snake with fewer than 6 gliders until February 2015, when Matthias Merzenich derived a 5-glider synthesis from an apgsearch soup.

[20] This can only be done via an extremely convoluted process known as the **reverse caber tosser** method, or **RCT** for short, which involves putting some of the gliders unimaginably far away, in a way that encodes an arbitrarily long slow-salvo construction recipe. See `conwaylife.com/wiki/RCT`.

[21] See `conwaylife.com/forums/viewtopic.php?p=140258#p140258` for details. In October 1972, Conway offered a $50 prize for an answer to the question of whether or not a non-synthesizable stable pattern like this exists. Unfortunately, he passed away shortly before its solution was found almost 50 years later.

[22] Created by Jeremy Tan in April 2019, and named for Shinjuku Station (the busiest railway station in the world). See `gitlab.com/parclytaxel/Shinjuku` for technical details and code, and `catagolue.hatsya.com/syntheses` for the syntheses themselves.

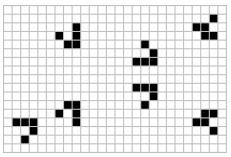
Free PDF. See conwaylife.com/book for pattern files and print version.

Exercises

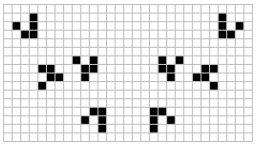
solutions to starred exercises on page 457

5.1 [1/5] Many glider syntheses work by using a small number of gliders to create the desired object plus some debris, and then additional gliders to clean up the debris. In each of the following syntheses, identify which gliders are used to clean up debris.

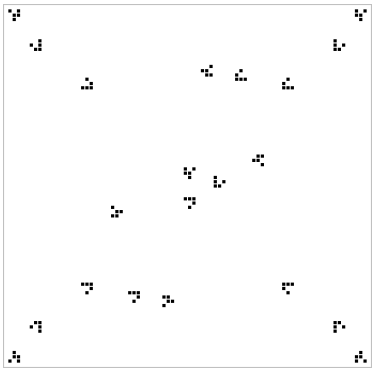
(a)



(b)



(c)



***5.2** [1/5] Gliders can be used to destroy essentially any unwanted debris that is left over after a synthesis. Use a single glider to destroy each of the following objects (and also destroy the glider in the process):

(a) A block.
(b) A beehive.
(c) A blinker.
(d) A ship.
(e) An LWSS.

5.3 [2/5] Use multiple gliders to completely destroy each of the following objects.
[Hint: Use one or two gliders to break the object down into simple ash objects like those from Exercise 5.2 and then use additional gliders to clean up those simpler objects.]

(a) A queen bee shuttle.
(b) A copperhead.
(c) Rich's p16.
(d) A Snark.

***5.4** [1/5] There are exactly 6 distinct ways for a glider to collide with a block. List them all and describe the result of each collision.

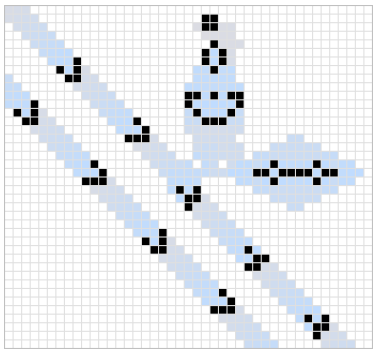
***5.5** [2/5] Use the syntheses from this chapter to create a 7-glider synthesis of eater 5.

5.6 [3/5] Use a lightweight spaceship synthesis from this chapter to construct a backrake that creates a period 120 stream of lightweight spaceships.

5.7 [2/5] Use three Gosper glider guns to create a middleweight spaceship gun.

5.8 [3/5] Consider the heavyweight spaceship synthesis in Table 5.2.

(a) Why can't you use three Gosper glider guns and this synthesis to create a heavyweight spaceship gun?
(b) One way to overcome this problem is to use **glider pushers**[23] (displayed below) to repeatedly push one glider stream closer to another. How many glider pushers would you need to use to fix the problem from part (a)?



5.9 [2/5] Use a two-glider synthesis of a block and a three-glider synthesis of a queen bee to create a 7-glider synthesis of a queen bee shuttle.

5.10 Consider the four-glider synthesis of the twin bees presented in Figure 5.3.

(a) [2/5] Use this synthesis to synthesize a twin bees shuttle.
(b) [3/5] Use this synthesis to synthesize the twin bees gun from Figure 1.23.

5.11 [3/5] Create a glider synthesis of the glider pusher from Exercise 5.8.

5.12 [3/5] Construct a glider synthesis of the period 36 oscillator from Figure 3.7(i).

[23] This glider pusher was found by Dietrich Leithner in December 1993.

Free PDF. See conwaylife.com/book for pattern files and print version.

∗**5.13** By using the 2-glider **kickback reaction** from Table 5.1 that changes the direction of a glider, we can decrease the number of directions used in many glider syntheses at the expense of increasing the number of gliders required. Use this technique to create glider syntheses for each of the following patterns with the property that all gliders come from just two different directions.

- **(a)** [2/5] A switch engine.
- **(b)** [2/5] A clock.
- **(c)** [3/5] A 3-engine Cordership.

∗**5.14** [4/5] Recall the 2-engine Cordership from Exercise 4.20.

- **(a)** Construct a glider synthesis of this Cordership.
- **(b)** Construct a glider synthesis of this Cordership that uses gliders coming only from two different directions.
  [Hint: Refer back to Exercise 5.13.]

∗**5.15** [2/5] By using one of the 3-glider **tee** collisions from Table 5.2, which produce a glider perpendicular to each of the input gliders, we can modify many glider syntheses so that all input gliders come from two antiparallel directions. Use this technique to create glider syntheses of each of the following patterns, with the property that all gliders come from two opposing directions.

- **(a)** A switch engine.
- **(b)** A clock.
- **(c)** Twin bees.

5.16 [3/5] Construct a glider synthesis for a Gosper glider gun that is stabilized on one end by an eater 1 instead of a block (as in the buckaroo of Figure 3.16(b)).

∗**5.17** [3/5] In Figure 5.9, we showed how to use gliders to turn an ecologist into a forward space rake.

- **(a)** Complete this incremental glider synthesis of the space rake (i.e., construct it in its entirety from an arrangement of gliders).
- **(b)** Show how to synthesize a period 60 space rake. [Hint: Recall the synthesis of the Schick engine from Figure 5.10.]

5.18 [4/5] For incremental syntheses of moving objects like the ecologist from Figure 5.8, there tend to be many choices that can be made, since individual components may be constructed earlier or later in time, and the location where they must be constructed will change accordingly.

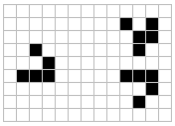
- **(a)** Make a two-direction incremental glider synthesis of the ecologist in which every glider comes from the southwest or southeast.
  [Hint: You can use two gliders to create a block and then two more gliders to turn it into an LWSS.]
- **(b)** Find the four gliders that synthesize the first LWSS in this ecologist. To construct that LWSS four generations sooner without disrupting the final recipe, how far and in what direction should each of these gliders be moved?
  [Hint: Divide the gliders into two pairs based on their direction of travel. Each pair will move a different distance in a different direction.]

∗**5.19** [3/5] In Figure 5.9 we showed how to use gliders to turn an ecologist into a forward space rake. Use similar techniques to turn an ecologist into a backward space rake.

5.20 [3/5] Create an arrangement of guns that synthesizes the fast forward force field from Figure 4.44. Use another gun to fire lightweight spaceships that are teleported through this fast forward force field.

5.21 [3/5] Create a breeder that uses three rakes to synthesize glider-producing switch engines via the 3-glider collision displayed in Table 5.2.
[Hint: You will need to use rakes with very high period.]

5.22 [2/5] Use the following 3-glider collision to reduce the 18-glider synthesis of Rich's p16 in Figure 5.16 down to 16 gliders:[24]



5.23 [3/5] The breeder from Figure 5.18 can be made quite a bit smaller by moving its space rakes closer together (and rephasing them as necessary). Use this technique to reduce the breeder's width by at least 50 cells.

5.24 [3/5] Modify the breeder from Figure 5.18 so that the space rakes still move to the east, but the Gosper glider guns shoot gliders to the northwest instead of the northeast.

5.25 [2/5] Construct a breeder that creates Gosper glider guns that are stabilized on at least one side by eater 1s instead of blocks. [Hint: Use the glider synthesis from Exercise 5.16.]

[24] This 16-glider synthesis was found by Martin Grant.

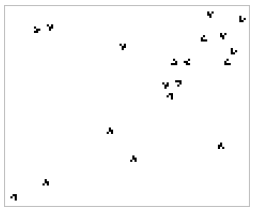
Free PDF. See conwaylife.com/book for pattern files and print version.

5.26 [3/5] Create a breeder that uses several rakes to synthesize block-laying switch engines as they move.
[Hint: Use a 3-glider synthesis of a switch engine together with the reaction from Figure 1.25.]

5.27 [3/5] Create a breeder that uses several rakes to synthesize twin bees guns as they move, using the glider synthesis that you constructed in Exercise 5.10.

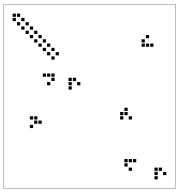
5.28 [4/5] Use a p1 slow salvo to create the arrangement of 8 blocks in Figure 5.25.

∗**5.29** A 19-glider synthesis of a boatstretcher (refer back to Figure 4.10 and Exercise 4.3) is displayed below.



(a) [2/5] Remove some gliders so as to produce a glider synthesis of the crab.
(b) [3/5] Add some extra gliders to destroy the boatstretcher (but not the boat itself) after it has been synthesized. Use this method to show that a fixed number of gliders (say 30 or so) can be used to synthesize arbitrarily large strict still lifes.

5.30 [2/5] The 7-glider reaction below attaches a period 2 oscillator called a **barberpole** to the end of a stretched tub or boat. Show how this reaction can be combined with the method of Exercise 5.29 to synthesize arbitrarily large oscillators with a fixed number of gliders (say 40 or so).



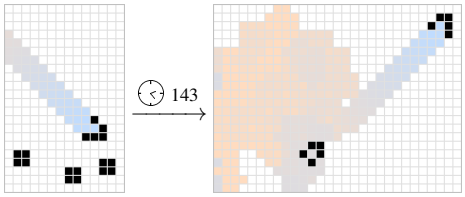
∗**5.31** [2/5] In this exercise, you will construct some limited-use turners that are different from the blockic one-time turners that we saw in this chapter.

(a) Show that a single boat can be used as a 90-degree one-time turner.
(b) Show that a single eater 1 can be used as a 90-degree one-time turner.
(c) Show that a single long boat can be used as either a 90-degree or 180-degree one-time turner.
(d) Show that the following arrangement of two blinkers can be used as a 90-degree one-time turner.



∗**5.32** In this exercise, we practice moving gliders around one-time tracks.

(a) [2/5] Use four boats to move a glider around a square track once, and then leave the track in the same direction that it started in (destroying the track in the process).
(b) [3/5] The following pattern might be called a "two-time turner", since it can be used to turn two gliders (by firing the second glider at the boat that the first glider produces). Explain why it is *not* possible to use four copies of this pattern to move a single glider around a square track twice (destroying the track in the process).



(c) [3/5] Use three copies of the two-time turner, together with some additional one-time-turners, to move a glider around a square track twice, and then leave the track in the same direction that it started in (destroying the track in the process).

∗**5.33** [3/5] One-time turners can be used as tracks for gliders, allowing us to create moving objects that travel at a different speed at the front than at the back (such patterns are called **growing spaceships**).

(a) Use two copies of the blinker puffer from Exercise 4.36 to lay two blinker fuses that a glider bounces back and forth between, using the reaction from Exercise 5.31(d).
(b) Use rakes of your choosing to synthesize a track of boats that a glider uses as one-time turners and destroys (as in Exercise 5.31(a)).

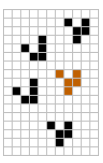
∗**5.34** [3/5] Find a way of placing a block near a clock so that a single glider (rather than a pair of gliders, as in Figure 5.28) can trigger the clock inserter reaction.

∗**5.35** [2/5] Use a clock inserter to insert the orange glider into each of these glider salvos, similar to how we inserted the orange glider into the salvo in Figure 5.29. In all cases, the input gliders to the clock inserter must come only from the southwest and northeast.

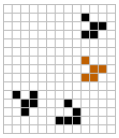
(a)



(b)



(c)



Free PDF. See conwaylife.com/book for pattern files and print version.

∗**5.36** [2/5] Many reactions can be used as inserters other than the clock inserter from Figure 5.28 (however, none are quite as good as the clock inserter itself).

(a) Use one of the "tee" 3-glider collisions from Table 5.2 to insert a glider 15 generations in front of another glider.

(b) Use an eater 1 to insert a glider 15 generations in front of another glider.
[Hint: Refer back to Exercise 5.31(b).]

(c) Use the clock inserter to insert a glider 14 generations in front of another glider.

5.37 [4/5] Create blockic seeds for each of the following objects.
[Hint: These can all be created using the same techniques that we used to make the blockic seed in Figure 5.27.]

(a) A lightweight spaceship.

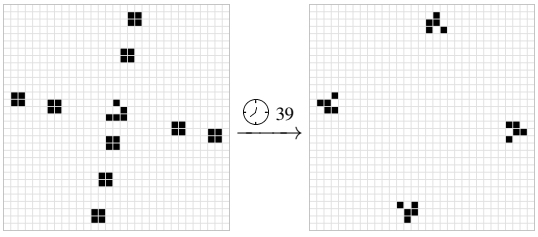
(b) A pulsar.

(c) A switch engine.

5.38 [5/5] Create a slow salvo that turns a single block into the configuration of 8 blocks displayed in Figure 5.25.
[Hint: You can do this by hand via 60 or so gliders by repeating the method of Figure 5.24. However, there is a search program called *slsparse* that builds slow salvos like this automatically. Tutorials and a download link can be found at `conwaylife.com/wiki/Slsparse`.]

5.39 [5/5] Use slsparse (`conwaylife.com/wiki/Slsparse`) to create a slow salvo that constructs the blockic seed for a clock from Figure 5.27, and thus a slow salvo for the clock itself.

5.40 [4/5] A commonly used blockic splitter that turns one glider into four gliders is displayed below. This splitter has the advantage of being much faster and cleaner than the one in Figure 5.26, but the disadvantage of requiring 9 blocks instead of just 3. Use this splitter as part of a blockic seed for a clock.



∗**5.41** [3/5] The argument that we used to improve Theorem 5.1 to Theorem 5.2 relied on us being able to use blockic splitters to create three gliders: one to trigger a clock seed and two to trigger the clock insertion reaction. How can we ensure that the clock is in the correct phase when the latter two gliders arrive to trigger the clock insertion?

5.42 [2/5] By chaining together multiple blockic splitters, we can turn one glider into any number of gliders. Create a blockic seed that turns one glider into exactly...

(a) 7 gliders.

(b) 10 gliders.

(c) 9 gliders.

∗**5.43** [5/5] In the proof of universality of the clock inserter in Appendix B.1, we defined a glider's "rank" as its lane number plus its timing. Let w be a real number and suppose that we instead defined a glider's rank as its lane number plus w times its timing.

(a) What parts of this proof change if we use $w = 2$?

(b) What is the slope of the lines of constant rank in the Life plane when $w = 2$ (recall that the lines of constant rank have slope 3 when $w = 1$)?

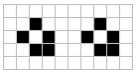
(c) The largest value of w for which the proof still works is $w = 7/3$. However, there is one extra technicality in this case—what is it, and how can it be overcome?

(d) What is the smallest value of w for which the proof still works?
[Hint: There will be a technicality similar to the one from part (c) that has to be overcome if w is minimal.]

(e) What are the possible slopes of lines of constant rank in the Life plane as w ranges from its minimal to maximal values? These are the slopes that we can use to define the "front" of a glider salvo for clock-insertion purposes.

∗**5.44** In this exercise, we demonstrate how to construct seeds for Corderships.

(a) [2/5] Show how a single glider can be fired at the following pair of boats so as to synthesize (generation 2 of) a switch engine.



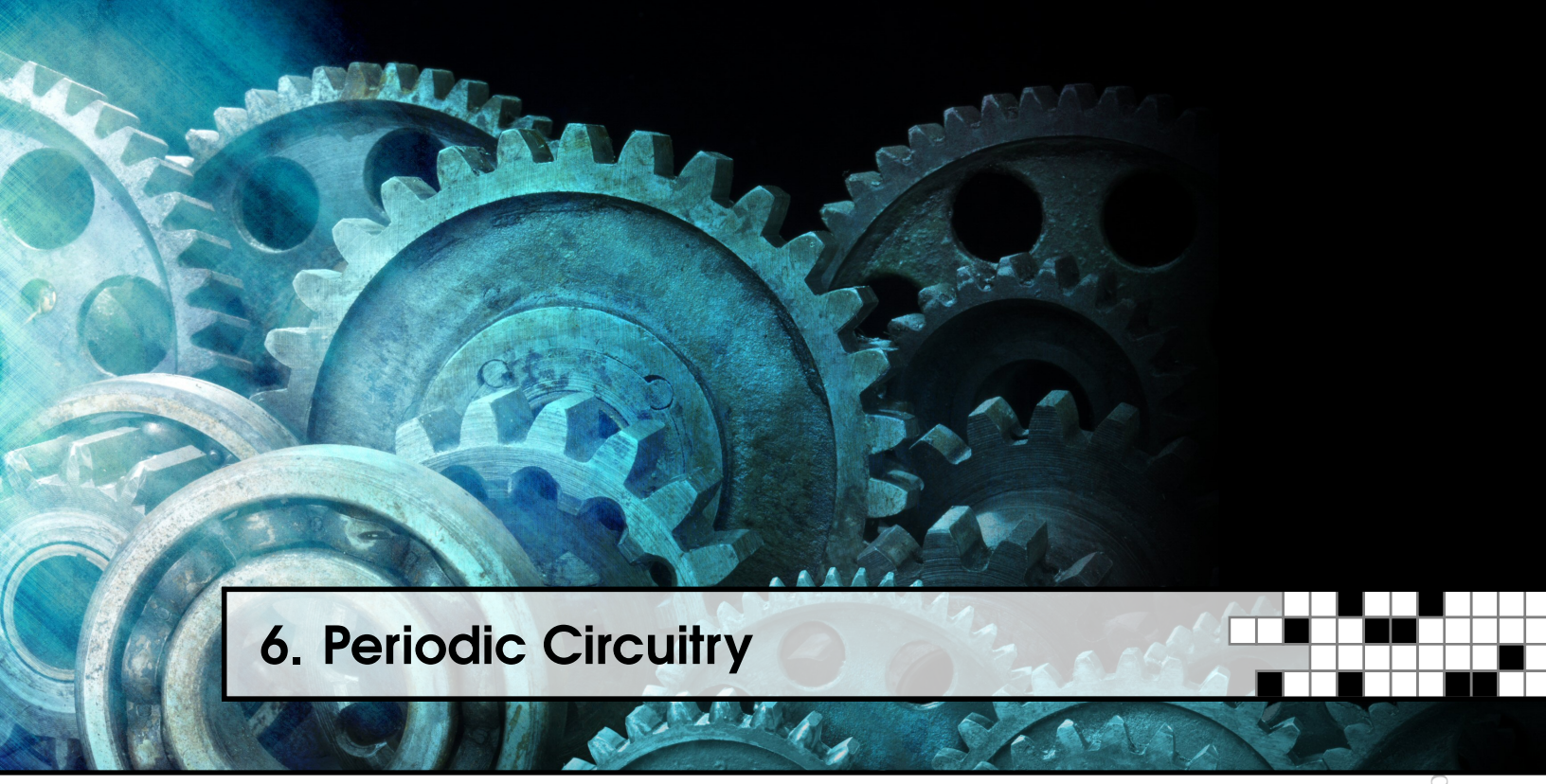
(b) [2/5] Create a glider synthesis that first creates the pair of boats displayed in part (a) and then creates a switch engine from them.

(c) [5/5] Use your solution to part (a) to construct a seed that, when hit by 2 gliders, synthesizes a 2-engine Cordership (refer back to Exercise 4.20).
[Hint: The seed will need to include two copies of this boat configuration, some blocks on the side, and also some blocks in the middle to stabilize the first period of each switch engine. It might be helpful to write a computer program to find a configuration of central blocks that works.]

(d) [3/5] Create a glider synthesis that first creates your seed from part (c) and then creates a 2-engine Cordership from it.

Free PDF. See conwaylife.com/book for pattern files and print version.

Free PDF. See conwaylife.com/book for pattern files and print version.



6. Periodic Circuitry

Sometimes you're a glider, sometimes a spaceship, and sometimes just a hole.

David Goodenough

The breeder that we constructed in Section 5.6 demonstrated a very important fact about how we will proceed with Life from this point on—we typically construct large patterns that do unusual things via a two-step process:

1) First, we design a schematic that illustrates the rough shape of the object and where the gliders (our standard building blocks) will travel. For example, for the breeder we wanted to construct a pattern that synthesizes an endless row of Gosper glider guns. We thus need the source of gliders that create those guns to move (i.e., we need them to come from rakes), so we reasonably quickly arrive at the schematic shown in Figure 6.1.

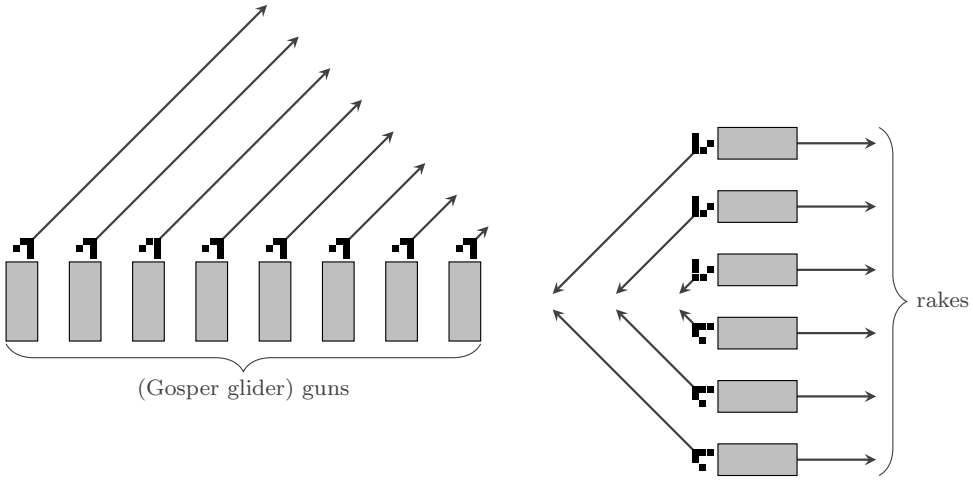


Figure 6.1: A schematic that gives a rough picture of how the breeder that we constructed in Section 5.6 works.

Free PDF. See conwaylife.com/book for pattern files and print versio

2) Next, we fill in the details—we place guns, rakes, reflectors, and other components that actually make the gliders (and potentially other spaceships) follow the tracks indicated and perform the required syntheses. For example, when constructing the breeder in Section 5.6, we arranged a total of 12 period 60 space rakes at the right-hand-side of the pattern to carry out the indicated Gosper glider gun synthesis.

Depending on the complexity of the schematic, implementing the second step above might be rather tricky, as manipulating glider paths and timing can be a somewhat fiddly and time-consuming affair. In this chapter, our goal is to develop circuitry that can help us do exactly this. That is, we start introducing circuitry that makes it easier to move gliders (and other spaceships) around tracks in such a way that we can make them appear wherever we want, whenever we want.

For now, we focus on periodic circuitry based on oscillators. The advantage of this type of circuitry is that it is typically smaller and simpler than stationary circuitry (which we explore in the next chapter, and is instead based on still lifes). However, it has the disadvantage that it can be difficult or impossible to make circuitry based on different periods work together. For example, if a period 5 oscillator is used to reflect a glider in some part of a mechanism, then the glider stream we are working with must have a period that is a multiple of 5, and all other connected circuitry must also work at the same period.

6.1 Period 30 Circuitry

The simplest set of circuitry that exists is based off of the queen bee shuttle and thus has period 30. For example, we already saw that we can use the queen bee shuttle to construct the Gosper glider gun (Figure 6.2) and buckaroo (Figure 6.3), which create and reflect gliders, respectively. We now present some other oscillators and circuits with compatible periods that can be used in conjunction with period 30 glider streams, and we also investigate how we can use these objects to manipulate glider streams in even more exotic ways.

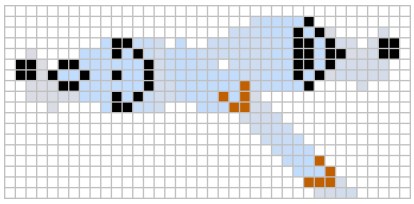


Figure 6.2: A Gosper glider gun producing gliders at a spacing of 30 generations.

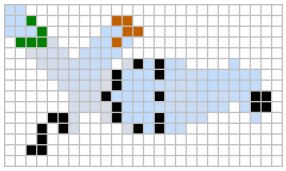


Figure 6.3: A buckaroo can reflect a glider by 90 degrees, from the position marked in green to the one in orange 30 generations later.

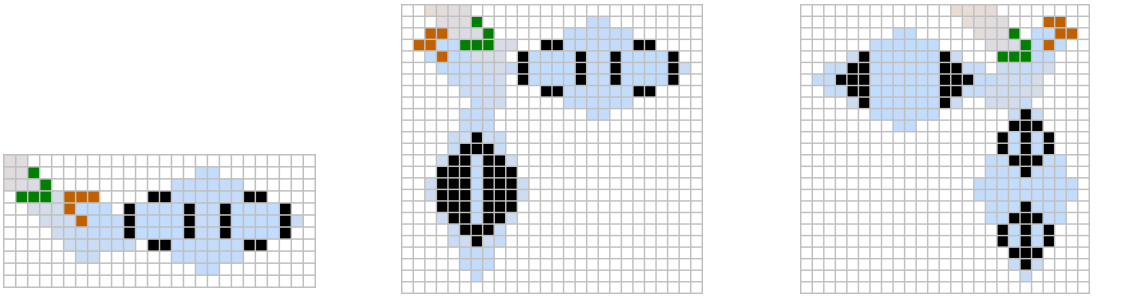
6.1.1 Reflectors

Because the pentadecathlon has period 15, which is a divisor of 30, it works very well with other period 30 circuitry. In particular, it can be used to reflect gliders in numerous different ways that are shown in Figure 6.4. The reflection shown in Figure 6.4(a) is the same one that we saw back in the period 60 oscillator of Figure 3.28, but the other two are new.

In particular, Figure 6.4(b) shows how two pentadecathlons can be used as a 180-degree reflector with slightly different timing and positioning than the one that we already knew about (i.e., the one in Figure 6.4(a)), and Figure 6.4(c) shows that two pentadecathlons can be used to create a 90-degree color-preserving reflector. There are also a few other period 30-friendly ways to reflect gliders aside from using pentadecathlons and buckaroos. We will see some color-changing 90-degree methods in

Free PDF. See conwaylife.com/book for pattern files and print version.

Figures 6.36(e) and 6.36(f), and two more color-preserving 90-degree options in Exercises 6.24(c) and 6.24(d).[1]



(a) A pentadecathlon reflecting a glider by 180 degrees.

(b) Two pentadecathlons reflecting a glider by 180 degrees.

(c) Two pentadecathlons reflecting a glider by 90 degrees.

Figure 6.4: Some period 15 pentadecathlon-based reflectors that work well with glider streams whose period is a multiple of 15 (the reflector (c) requires period at least 45). Input gliders are green, while the location where they will be 30 generations later is displayed in orange.

6.1.2 Inverters

If we have an *irregular* glider stream—one in which gliders are separated by a fixed period, except some gliders are missing—it is often useful to invert the stream. That is, we would like to have a mechanism for replacing the gliders in the stream by empty gaps, and replacing the empty gaps by gliders.

One simple method of implementing this inversion (which works at any period) is to use a glider gun to fire a regular stream at the irregular stream so that gliders collide so as to annihilate each other. Then any gliders that are present in the irregular stream will be destroyed by the gun, while any gliders that are missing in the irregular stream will be generated by the gun, as illustrated in Figure 6.5.

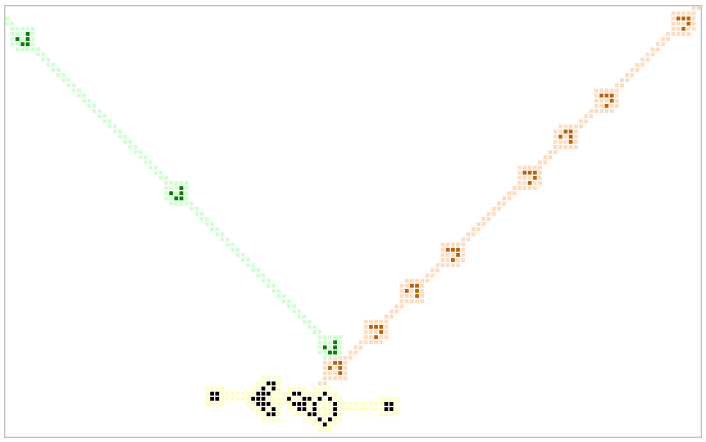


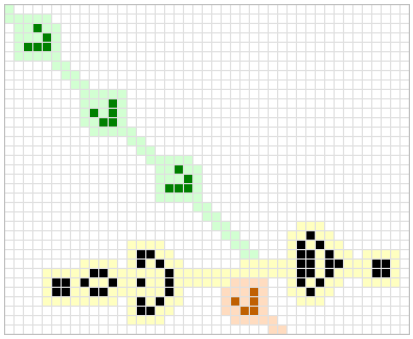
Figure 6.5: A glider stream (highlighted in green) can be inverted by a glider gun (highlighted in yellow). In this case, a p120 input stream is converted into a p30 output stream (highlighted in orange) that is missing one out of every four gliders.

At period 30, there is a modification of this reaction that has the advantage that the irregular input stream and its inversion travel in the same direction. Recall that the Gosper glider gun works by

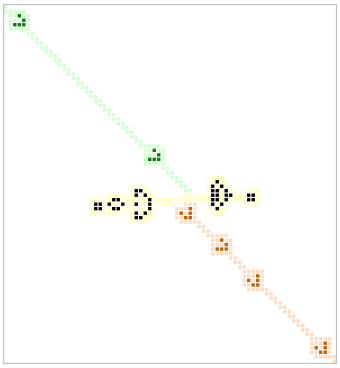
[1] It is also trivially the case that stable reflectors work with glider streams whose period is a large enough multiple of 30 (at least 60 in the case of the Snark).

Free PDF. See conwaylife.com/book for pattern files and print version.

colliding two queen bees with each other so that they create a small explosion that then generates a glider. If we carefully fire a glider at that small explosion, we can suppress its creation of a glider, resulting in a reaction called an **inline inverter**.[2] The output stream travels in the same direction as the input stream, but is slightly offset (see Figure 6.6).



(a) If an input glider is *not* present then an output glider will appear at the location marked in orange 30 generations after the phase shown here (i.e., the output stream is 5 cells further south than the input stream).



(b) An inline inverter converting a thin glider stream into a thick one. A period 120 glider stream is converted into a period 30 stream that is missing one out of every four gliders, just like in Figure 6.5.

Figure 6.6: An **inline inverter** is a reaction involving the Gosper glider gun in which the gun fails to produce a glider if it receives a glider as input at the same time.

These inverters can be used to create many new types of patterns that we have not yet seen, such as glider guns with extremely large period. For example, we can use one Gosper glider gun to feed gliders into an inline inverter, creating a period 30 oscillator made up of a glider stream of any finite length of our choosing. We can then trigger this oscillator to release a glider by destroying one of the gliders in the central stream. For example, we can use one of the two-glider collisions from Table 5.1 to have a glider bounce off of the glider stream, resulting in a single other glider escaping from the stream, as in Figure 6.7.

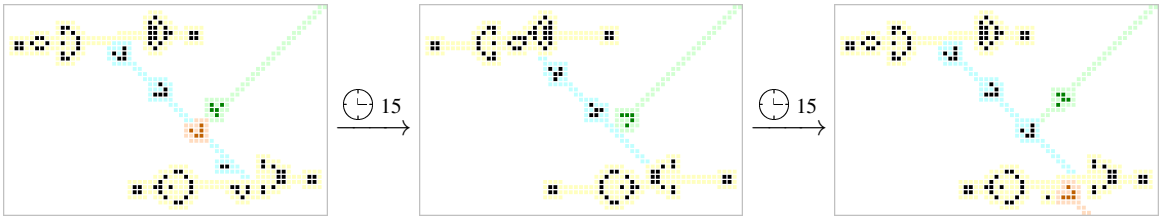


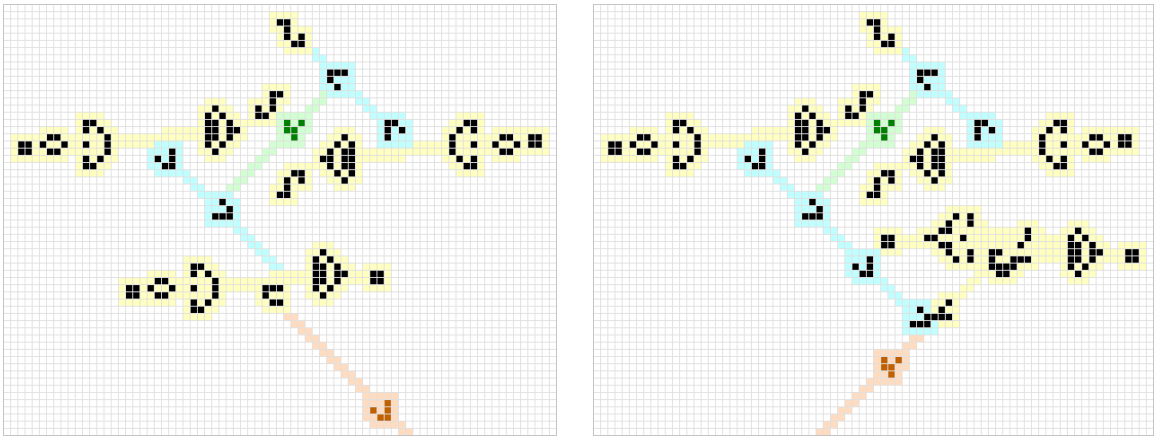
Figure 6.7: Two Gosper glider guns can be placed near each other so as to create a finite stream of gliders (highlighted here in aqua) between them. Here, we bounce a single glider (in green) off of one of those gliders (in orange) so that it is destroyed and thus released by the inline inverter to the southeast.

By bouncing a glider back and forth between two of these finite-length glider streams, we can create glider guns with arbitrarily large periods. In particular, by shifting these two mechanisms away from each other by multiples of 15 cells, we can create glider guns with period equal to $120n$ for any integer $n \geq 1$. Figure 6.8 illustrates some guns of this type with period 120.

Another particularly useful inversion reaction is provided in Figure 6.9, in which a glider stream of any period 20 or larger collides with an *irregular* stream of the same period (i.e., a stream with some

[2]Discovered by David Bell. There are also inline inverters based on some other glider guns—see the upcoming Figure 6.29(b).

Free PDF. See conwaylife.com/book for pattern files and print version.

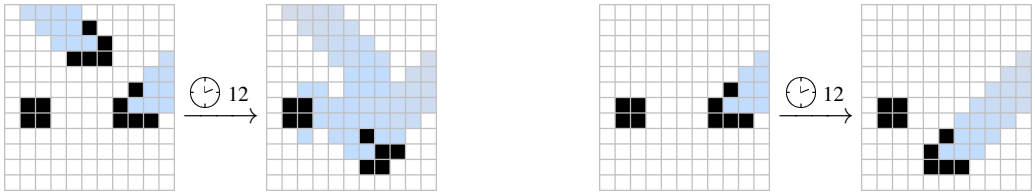


(a) p120 gun from inline inverter

(b) p120 gun from collision-based non-inline inverter

Figure 6.8: Three Gosper glider guns (highlighted in yellow, the bottom of which acts as an inverter) can be used to create a period 120 glider gun. The glider displayed in green bounces back and forth between the two aqua glider streams, destroying one glider in each stream every time it is reflected. The glider that is destroyed in the left stream is then created (in orange) by the inverter. In (a) the bottom glider gun is an inline inverter, whereas in (b) it is a non-inline inverter.

gliders potentially missing). If a glider is missing in the irregular glider stream, then the corresponding glider in the regular stream passes by unharmed. On the other hand, if a glider is present in the irregular glider stream, then a collision occurs that generates a glider in a different direction. There are thus two output glider streams: one that contains exactly the same gliders as the irregular input stream, and one that is the inverse of that stream (i.e., it contains a glider if and only if the irregular input stream did not).



(a) This collision causes the top-right glider to be erased and the top-left glider to be slightly offset.

(b) If the top-left glider is omitted, the top-right glider passes by unharmed.

Figure 6.9: A stream inverter that splits the glider stream coming in from the northeast based on which gliders are present in the stream coming in from the northwest. This reaction can be used with any glider stream of period 20 or higher. The output stream going to the southeast is a duplicate of the input stream coming from the northwest, whereas the output stream going to the southwest is its inverse.

This reaction can thus be used as a *non-destructive* inverter: it produces an inverted glider stream, but does not use up the original irregular stream in the process. By applying another inverter to the inverted stream that this reaction produces, we can now duplicate arbitrary glider streams (even irregular ones), as in Figure 6.10. Patterns like this one are called **glider duplicators**, and they are extremely important because they let us use a single input signal (i.e., a glider) to trigger multiple different reactions.

6.1.3 Miscellaneous Period 30 Circuits

There are also a few other useful tricks that can be done with period 30 glider streams in order to make them easier to work with. For example, one object that can be used to help push glider streams

Free PDF. See conwaylife.com/book for pattern files and print version.

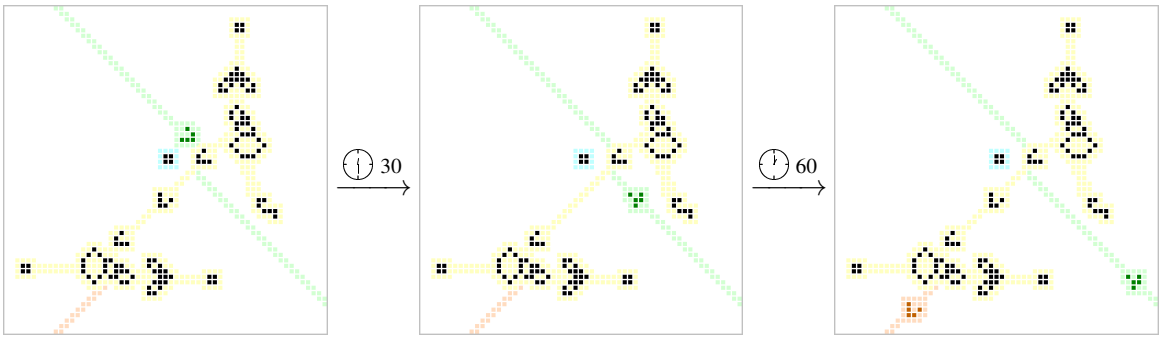


Figure 6.10: A glider duplicator that makes use of the inline inverter and the reaction from Figure 6.9. The input glider from the northwest (highlighted in green) is copied to the southeast and destroys the corresponding glider going southwest. The southern inline inverter then creates a glider going southwest (highlighted in orange), thus duplicating the original glider.

closer together (which is extremely useful when trying to implement some tight glider syntheses) is a **glider pusher**,[3] which is a combination of a (half-stabilized) queen bee and a pentadecathlon that push a glider over by a single lane (see Figure 6.11).[4]

Two other simple reactions that make use of the debris and sparks left behind by queen bees and pentadecathlons are shown in Figures 6.12 and 6.13. These reactions let us convert gliders directly into lightweight spaceships (rather than having to synthesize them via 3 gliders) and then convert them back into gliders if desired. Note, however, that the former reaction can only be used with streams of period at least 60, since otherwise the output LWSS collides with the next input glider.

It is worth comparing the LWSS-to-glider converter of Figure 6.13 to the pentadecathlon-based reflectors of Figures 6.4(b) and 6.4(c). In all three cases, the reaction is made up of two pentadecathlons, the first of which converts the input object (either an LWSS or a glider) into a block and the second of which converts that block into the output glider.

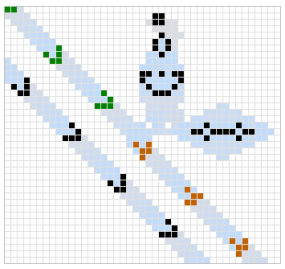


Figure 6.11: A **glider pusher** pushes a glider away by one lane.

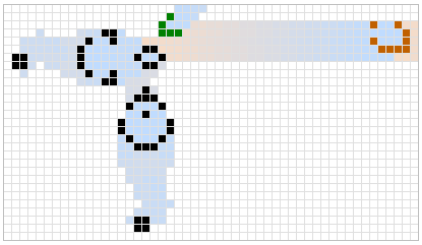


Figure 6.12: Two queen bees can convert a glider into an LWSS.

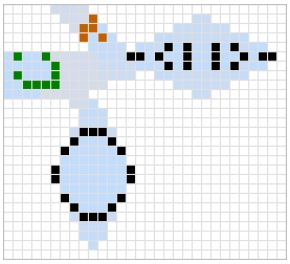


Figure 6.13: Pentadecathlons can convert an LWSS into a glider.

One final piece of machinery that is a bit more heavy-duty than the other period 30 reactions that we have illustrated is the **toggle**,[5] which is a glider gun that can be switched on and off by firing a single glider into it (contrast this with the inline inverter, which is a glider gun that can be switched off by firing a *constant stream* of gliders into it). The toggle works by exploiting the fact that a glider can strike a particular phase of an LWSS in a way that destroys the LWSS and creates a perpendicular output glider—but if the incoming glider is delayed by 30 generations then both are simply destroyed.

[3] Found by Dietrich Leithner in December 1993.
[4] We first saw this glider pusher in Exercise 5.8(b).
[5] Constructed by Dean Hickerson in April 1996.

Free PDF. See conwaylife.com/book for pattern files and print version.

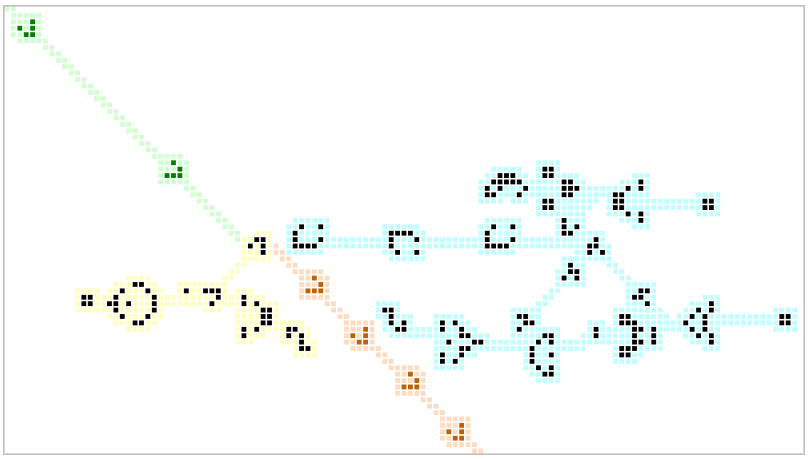


Figure 6.14: A **toggle** that bounces gliders from a Gosper glider gun (highlighted in yellow) off of a stream of lightweight spaceships (highlighted in aqua). The first incoming glider on the green stream from the northwest turns off the output stream of gliders (highlighted in orange) until another glider comes in behind it to turn the stream back on.

6.2 Primer

We now have enough circuitry and tools at our disposal that we can start to construct some really unusual and interesting patterns. To illustrate how to put these pieces together in non-trivial ways, we now construct a pattern that computes prime numbers. More specifically, we build a pattern that emits a stream of lightweight spaceships with the property that the n-th spaceship in the stream is present if and only if n is prime. That is, we construct a gun that emits a stream of lightweight spaceships with spacing as in Figure 6.15. We call guns of this type **primers**.[6]



Figure 6.15: A stream of lightweight spaceships in which only the prime-indexed spaceships are present.

6.2.1 An LWSS Gun with an Irregular Stream

Before building the desired primer itself, let's first construct a (much simpler) pattern that emits a stream of lightweight spaceships with the property that the n-th spaceship in the stream is present if and only if n is not a multiple of 2 or 3. Since we do not have any methods for creating streams of spaceships with irregular spacing directly, we will instead create a regular stream of lightweight spaceships, and then strategically destroy the unwanted spaceships in it (i.e., we destroy all lightweight spaceships corresponding to multiples of 2 or multiples of 3).

Figure 6.16 shows how we can fire a glider at a lightweight spaceship so as to destroy them both.[7] Thanks to this reaction, we can create the desired stream of lightweight spaceships simply by aiming two glider guns at a stream of lightweight spaceships: one with double the period of the LWSS stream (to eliminate the multiples of 2) and one with triple its period (to eliminate the multiples of 3). A schematic for such a pattern is presented in Figure 6.17.

[6] The first primer was constructed by Dean Hickerson in November 1991. It used the same techniques that we will demonstrate here, but some of the component reactions were slightly different (e.g., instead of the inline inverter, it used the stream inversion method of Figure 6.5).

[7] Reactions like this one, where two small objects are used to destroy each other, can easily be found by hand. Just try a few collisions with slightly different positioning and timing, and one of them will likely work.

Free PDF. See conwaylife.com/book for pattern files and print version.

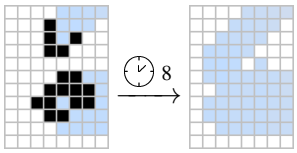


Figure 6.16: A glider and a lightweight spaceship destroying each other.

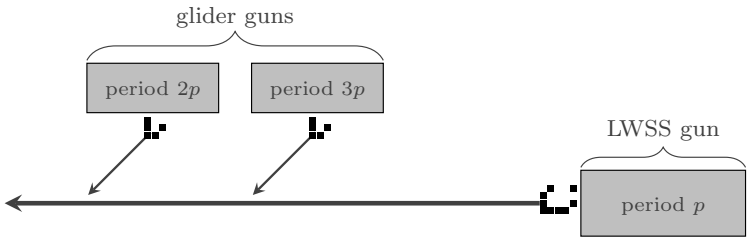


Figure 6.17: A schematic for an LWSS gun that fires an irregular stream of spaceships with the property that the n-th spaceship in the stream is present if and only if n is not a multiple of 2 or 3. Gliders are used to delete every second and every third spaceship in the LWSS stream.

To actually construct such a pattern, we make use of the period 30 circuitry that we have seen so far. If the period of the LWSS gun that we use is p, then we need glider guns with periods $2p$ and $3p$ as well. Since we do not yet know how to construct guns with periods 60 or 90 (we will learn how to do so in Section 8.1), it seems natural to use the period $120n$ guns based on the inline inverter that we introduced in Figure 6.8(a). In particular, we arrange three period 120 guns so as to synthesize a lightweight spaceship, and then aim period 240 and 360 guns at the resulting LWSS stream in order to thin it out. The final pattern is displayed in Figure 6.18.

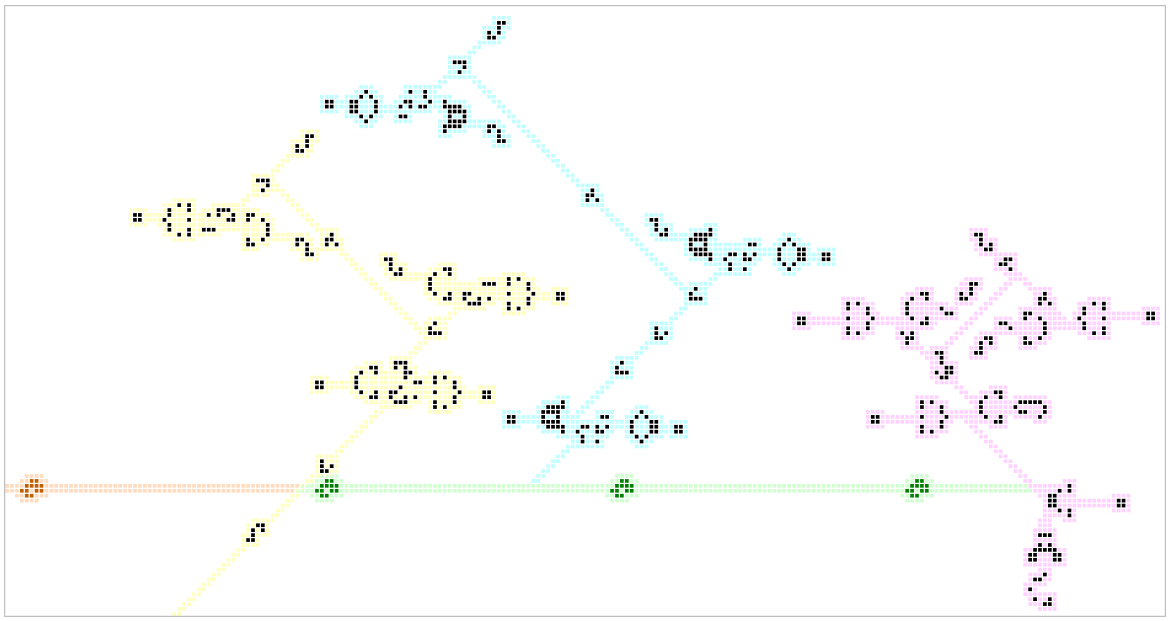


Figure 6.18: A gun that shoots an irregularly spaced stream of lightweight spaceships. In particular, the period 120 inline inverter gun makes use of the glider-to-LWSS reaction from Figure 6.12 (highlighted in magenta) to create a period 120 stream of lightweight spaceships (highlighted in green). Period 240 and period 360 inline inverter guns (highlighted in yellow and aqua, respectively) then destroy every LWSS with a position in the stream that is a multiple of 2 or 3, respectively.

6.2.2 The Prime-Generating Gun Itself

The gun that we just constructed contains the key idea used in our prime-generating gun—we create a period $p = 120$ stream of lightweight spaceships and then aim glider guns with periods $2p$, $3p$, $4p$, $5p$, and so on at the gun to delete any lightweight spaceships whose position in the stream is a multiple of

Free PDF. See conwaylife.com/book for pattern files and print version.

2, 3, 4, 5, . . ., respectively.

The tricky part is that we now have to find a way to construct infinitely many glider guns (like the breeder) with ever-increasing periods (unlike the breeder). Fortunately, the period $120n$ guns based on the inline inverter are perfect candidates for this task, since their periods are determined solely by how far apart the two Gosper glider guns at their ends are. Thus one way of constructing glider guns with ever-increasing periods is to send one set of rakes north to synthesize Gosper glider guns and another set of rakes east to do the same. The diagonal distance between these endpoint Gosper glider guns, along which a single glider will travel, increases without bound, giving us glider guns of larger and larger periods. We thus arrive at the schematic presented in Figure 6.19 for synthesizing these high-period glider guns and thus our primer.

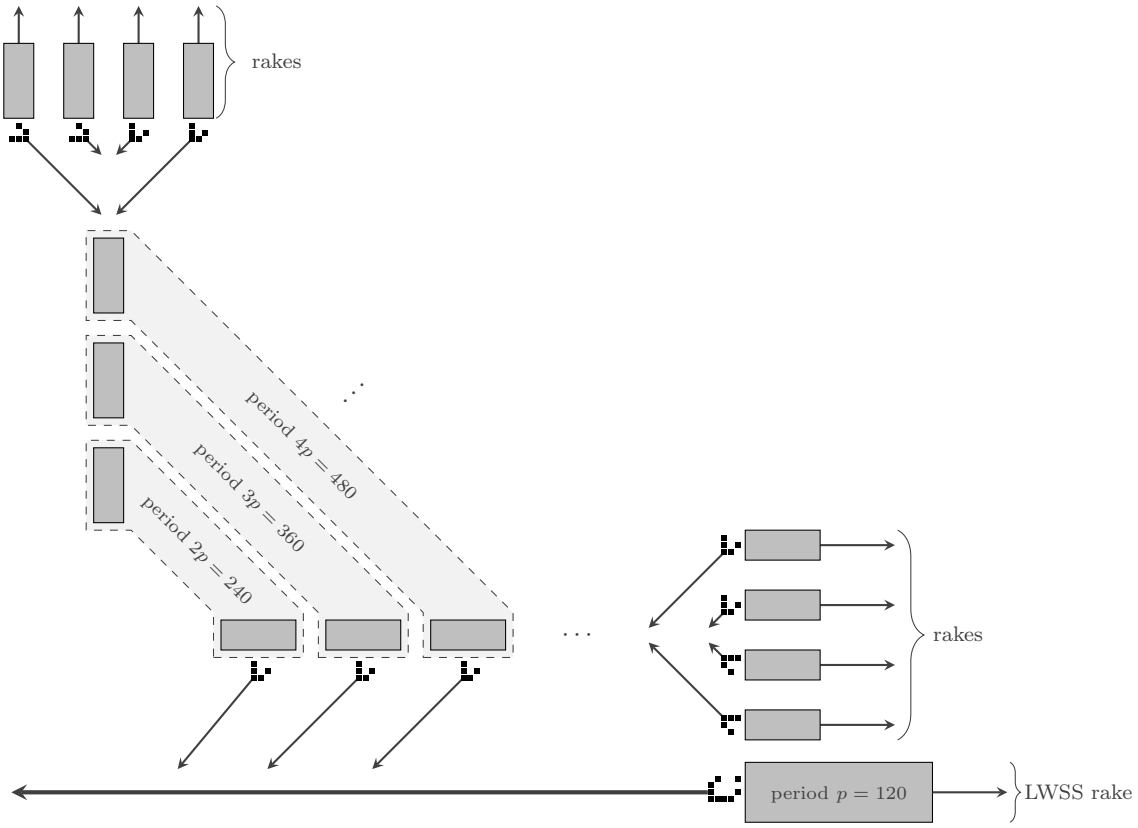


Figure 6.19: A schematic for a primer. For each integer $n \geq 2$, eastward and northward sets of rakes leave behind halves of the period $120n$ glider gun based on the inline inverter, which is used to destroy every LWSS in the bottom period 120 stream whose position is a multiple of n. The only spaceships that escape to the far left are the ones whose position is not evenly divisible by any smaller integer $n \geq 2$ (i.e., the primes).

This is by far our biggest construction to date—the breeder from Section 5.6 was already quite large, and it just laid one row of Gosper glider guns, while this construction requires us to lay *three* rows of Gosper glider guns (two going eastward and one going northward). It is thus worthwhile to present a more compact Gosper glider gun breeder that we can use as a component in our primer—see Figure 6.20. Although this breeder looks quite different from our original one of the surface, it functions in more or less the same way by making use of several period 60 space rakes to gradually build the Gosper glider guns piece-by-piece.

Unfortunately, there are actually quite a few subtle problems with the previous schematic that prevent us from building it as-is:

Free PDF. See conwaylife.com/book for pattern files and print version.

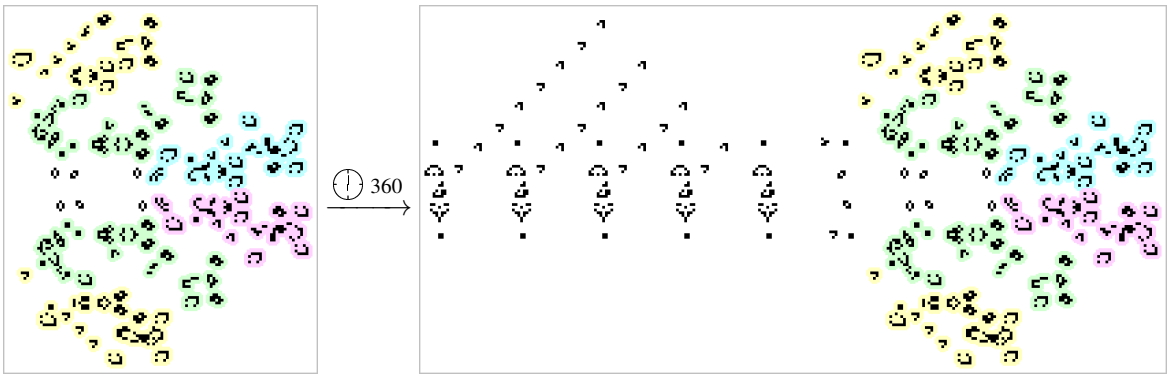


Figure 6.20: A compact breeder that uses 6 period 60 space rakes to carry out an incremental synthesis of a Gosper glider gun. It gets away with fewer space rakes than our original breeder from Figure 5.18 (which used 12) since it cleverly manipulates the debris behind the two frontmost space rakes to create a ship, which is the most difficult part of the Gosper glider gun's incremental synthesis to create.

1) The only Gosper glider gun breeders that we know how to build so far have period 60, which results in Gosper glider guns that are separated by 30 cells, and thus inline inverter guns with periods that increase by 240 (not 120). In order to solve this problem, our first instinct might be to create a period 240 stream of lightweight spaceships instead of a period $p = 120$ stream. However, we do not yet know how to do this.
   As an alternate solution, we use the breeders to create glider guns with periods $3p$, $5p$, $7p$, $9p$, ... (instead of $2p$, $3p$, $4p$, $5p$, ...) to delete any lightweight spaceships whose position in the stream is a multiple of 3, 5, 7, 9, ..., respectively. We then manually place a single glider gun of period $2p$ to delete the leftover lightweight spaceships whose positions in the stream are a multiple of 2.
2) With the Gosper glider guns just 30 cells apart, we cannot fire gliders back and forth between them diagonally—the bouncing gliders would collide with the blocks that stabilize the glider guns.
   In order to solve this problem, we do the same thing that we did in Figure 6.8(a) to squeeze Gosper glider guns closer together—we replace one of the stabilizing blocks by an eater 1 (as in the buckaroo of Figure 6.3). To create a row of these modified Gosper glider guns, we can use the breeder displayed in Figure 6.21, which is just a slight modification of the breeders that we have already seen.[8]
3) When the eastward breeders construct their two rows of Gosper glider guns, if their construction is not synchronized perfectly then the bottom row will release some gliders in the time between when it is constructed and when gliders from the top row are fed into it. This problem can be solved by using a fixed-length row of middleweight (or heavyweight) spaceships below those guns to destroy the excess gliders as in Figure 4.12.
4) A period $3p$ gun (for example) will erase all lightweight spaceships whose position in the stream is a multiple of 3, *including* 3 *itself*, while we only want it to erase the multiples of 3 greater than 3. To solve this final problem, we can place a single block below each inline inverter gun in such a way that it destroys (and is destroyed by) the first glider released from the gun, thus preventing that first LWSS from being destroyed.

[8] We already constructed a (much larger) breeder for this modification of the Gosper glider gun back in Exercise 5.25.

Free PDF. See conwaylife.com/book for pattern files and print version.

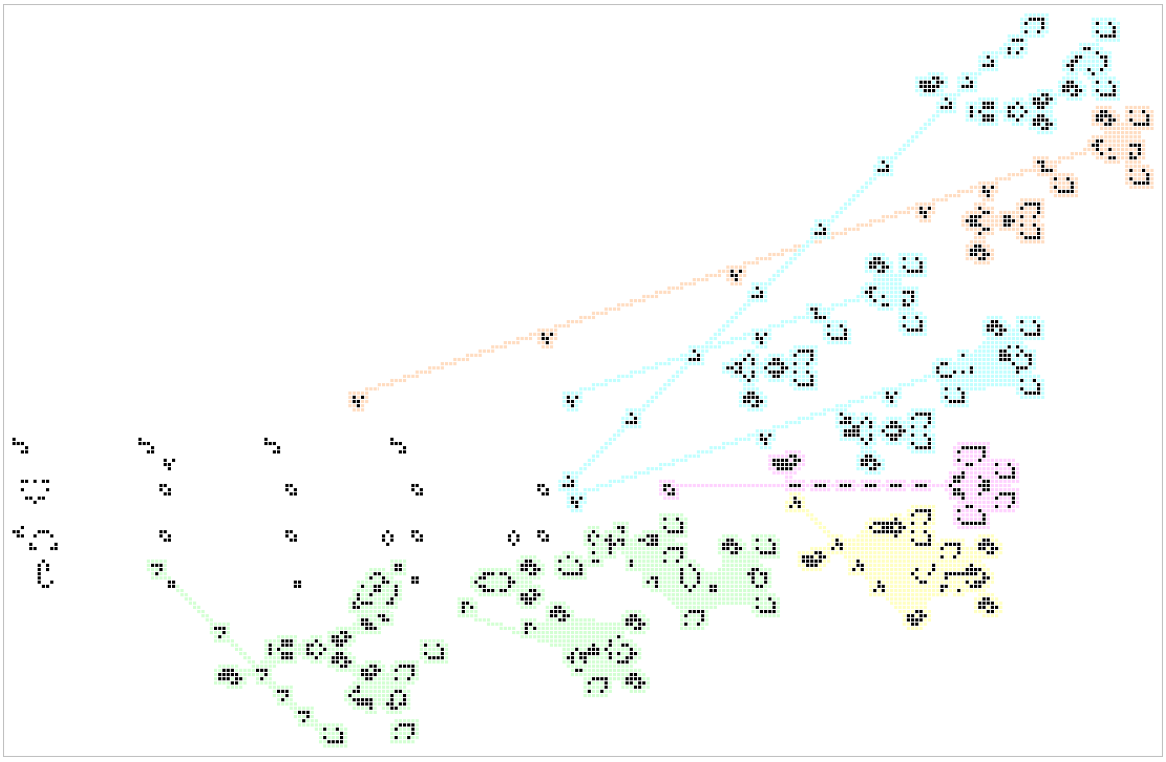


Figure 6.21: A Gosper glider gun breeder that is stabilized on one side by an eater 1 instead of a block. The bottom half (highlighted in green) is the same as the bottom half of the compact breeder from Figure 6.20. The bottom-right space rake (highlighted in yellow) turns one of the blinkers from the blinker puffer (highlighted in magenta and introduced in Exercise 4.37) into a ship via the incremental synthesis that we saw in Table 5.4. The rightmost space rake (highlighted in orange) synthesizes the top ship into a queen bee, while the remaining three space rakes (highlighted in aqua) synthesize the eater 1.

After putting all of these pieces together, we arrive at our completed primer in Figure 6.22. We emphasize that static pictures really cannot do this pattern justice, so the reader is strongly encouraged to interact with and explore this pattern in Life-viewing software.[9]

6.3 Period 46 Circuitry

Just like we can use the queen bee shuttle and Gosper glider gun as the basis of a set of period 30 circuitry, we can also use the twin bees shuttle (Figure 1.22) and the related twin bees gun (Figure 1.23) to build up a set of period 46 circuitry. For example, the twin bees shuttle can be used to reflect gliders and lightweight spaceships, and even convert gliders and lightweight spaceships *into* each other, as displayed in Figure 6.23.[10]

We saw one of these glider reflections way back in Section 3.3.1—the twin bees shuttle creates a duoplet spark that does the reflection.[11] The other reflections and the glider-to-LWSS-to-glider conversions all make use of the large spark that is produced when one side of the twin bees shuttle is stabilized by just a single block.

[9] Like Golly: `golly.sourceforge.net`.

[10] The 180 degree glider reflection takes 92 generations to complete, since the twin bees shuttle first converts the incoming glider into a block and then converts that block into the output glider. It was found by Dean Hickerson.

[11] This reflection is the one highlighted in green in Figure 6.23(a).

Free PDF. See conwaylife.com/book for pattern files and print version.

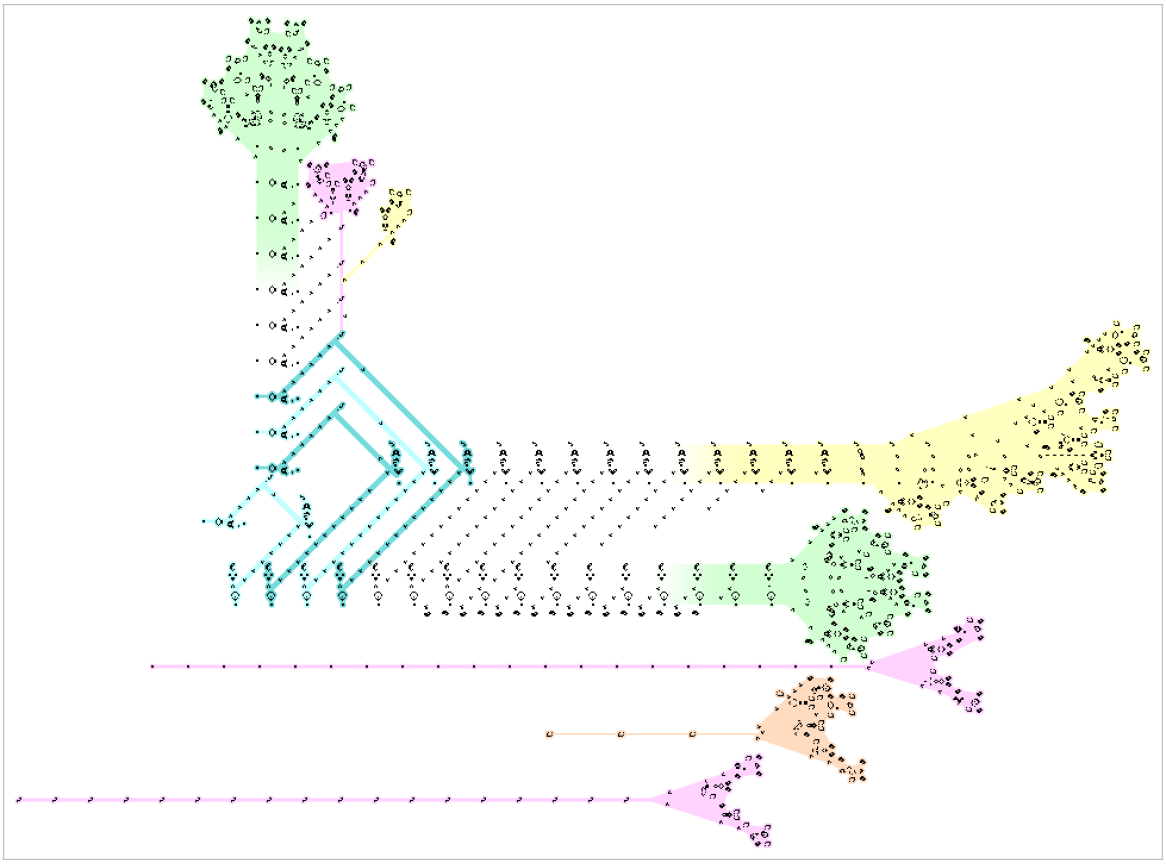
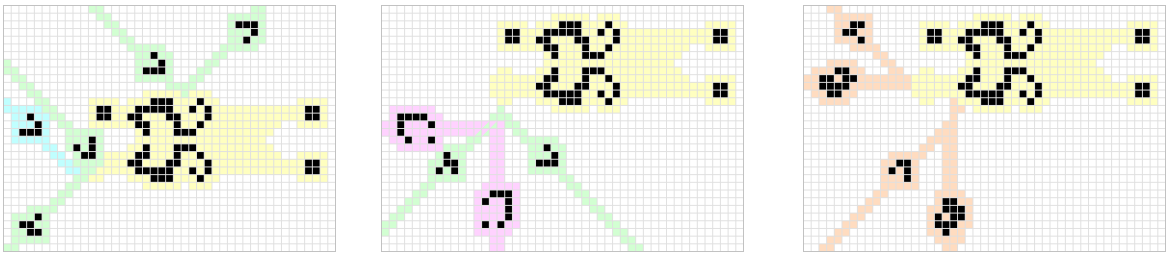


Figure 6.22: The completed primer. The period 120 stream of lightweight spaceships (highlighted in orange and synthesized by 3 space rakes as in Exercise 5.6) moves to the left and is thinned out by the inline inverter guns (highlighted in aqua). The leftmost inline inverter gun has period 240 and has a different shape from the others (which have periods 360, 600, 840, and in general $360+240n$ for $n \geq 0$). The rows of Gosper glider guns are laid by two copies of the compact breeder from Figure 6.20 (highlighted in green) and the row of Gosper glider guns that are stabilized by eater 1s instead of blocks are laid by the breeder that we presented in Figure 6.21 (highlighted in yellow). A single space rake going north (also highlighted in yellow) produces the gliders that bounce along the long diagonal of the inline inverter guns, and the remaining pairs of space rakes (highlighted in magenta) simply synthesize rows of block and eater 1s.

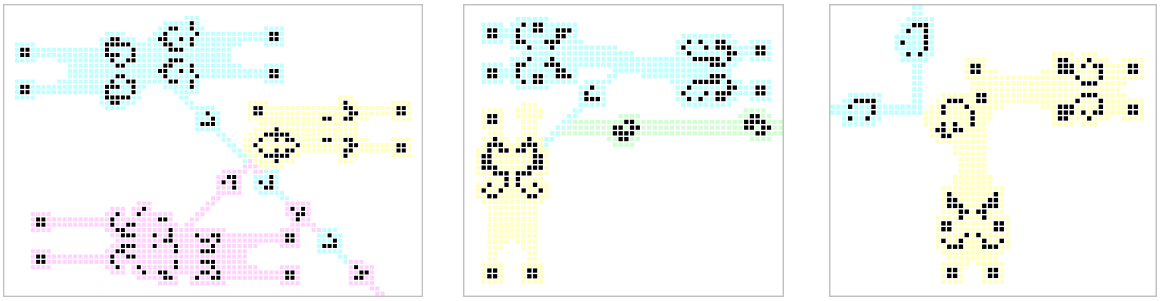


(a) three glider reflections **(b)** two more reflections **(c)** glider → LWSS and back

Figure 6.23: The twin bees shuttle (highlighted in yellow) can be used to reflect a glider by 90 degrees in three different ways (highlighted in green), reflect a glider by 180 degrees (highlighted in aqua), reflect a lightweight spaceship by 90 degrees (highlighted in magenta), and turn a glider into a lightweight spaceship and back (highlighted in orange).

Free PDF. See conwaylife.com/book for pattern files and print version.

One of the advantages of these new reflections over the old reflection based on the duoplet spark is that they happen near the corner of the shuttle, so they can be used to (for example) merge two period 46 glider streams into a single period 23 stream, as in the period 23 glider gun displayed in Figure 6.24(a). Similarly, by making use of the glider-to-LWSS converter we can easily create period 46 LWSS guns without needing to synthesize them via three glider guns (see Figure 6.24(b)). We can even use a careful arrangement of two twin bees shuttles to convert a lightweight spaceship into a middleweight one, and thus create small middleweight spaceship guns (see Figure 6.24(c) and Exercise 6.12). By replacing the input LWSS by an MWSS, this final LWSS-to-MWSS converter can also be used to simply reflect an MWSS by 90 degrees.



(a) a period 23 glider gun **(b)** a period 46 LWSS gun **(c)** LWSS-to-MWSS converter

Figure 6.24: Various reflection and conversion reactions involving twin bees that can be used to create some useful guns. (a) is a period 23 glider gun constructed by reflecting one period 46 stream into the gaps in another period 46 stream, (b) is a lightweight spaceship gun that uses the glider-to-LWSS conversion from Figure 6.23(c), and (c) is an LWSS-to-MWSS conversion that (for example) allows for the construction of a small MWSS gun.

There are also a few other useful reactions involving the twin bees that can do things like emit two side-by-side streams of gliders (guns like this are called **double-barrelled guns**). We explore some of these reactions in Exercise 6.19.

6.3.1 Ticker Tapes and Memory Cells

One of the most useful reactions involving a twin bees shuttle that we have not yet seen is the one displayed in Figure 6.25 in which the shuttle reflects a glider by 90 degrees while simultaneously duplicating it (similar to the p30 glider duplicator that we saw in Figure 6.10).

One interesting thing that we can do with a glider duplicator is attach it to a glider loop so as to produce any (finite) irregular sequence of gliders of our choosing, simply by having that sequence of gliders in the loop itself. For example, the pattern displayed in Figure 6.26 makes use of a loop in which one glider is present, then the next two are missing, then the next three are present, then four are missing, five are present, and six are missing, so as to create a gun that produces lightweight spaceships with that same spacing.[12] We can thus encode any message in binary and create a gun that emits that message if we interpret the presence of a spaceship as a "1" and the absence of a spaceship as a "0".

By placing several of these guns near each other (with different sequences of gliders in their glider loops), we can even create **ticker tape guns**, which emit any *visual* message of our choosing, rather than just messages encoded in binary. To do this, we think of lightweight spaceships as pixels, with the presence of a ship being black and the absence of a ship being white. Each of the loop-based guns emits the pixels (ships) in a single row of the image being produced, so we use n of the guns with m

[12] This gun produces lightweight spaceships (via the glider-to-LWSS reaction of Figure 6.23(c)) instead of gliders just to make the output easier to visualize.

Free PDF. See conwaylife.com/book for pattern files and print version.

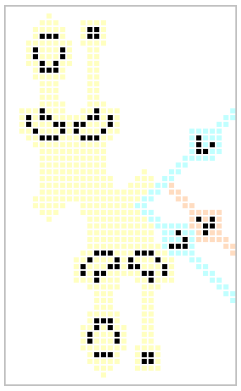


Figure 6.25: A twin bees shuttle reflecting and duplicating a glider.

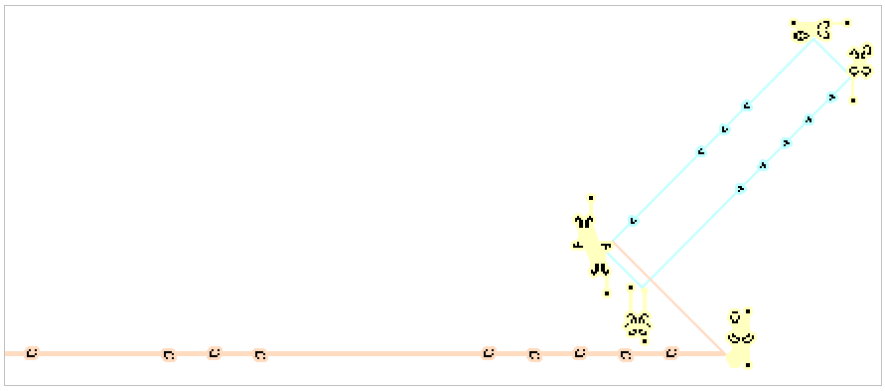


Figure 6.26: A gun that produces an irregular lightweight spaceship stream corresponding to the bitstring 100111000011111000000 (i.e., one LWSS present, then two missing, then three present, and so on). This stream (highlighted in orange) is produced by duplicating the gliders in the loop (highlighted in aqua) that have this same spacing and then performing a few reflections and a glider-to-LWSS conversion via twin bees shuttles (highlighted in yellow).

gliders each in the loops to create an $n \times m$ image. For example, Figure 6.27 illustrates a ticker tape gun that repeatedly produces an 8×21 array of lightweight spaceships that spells out "HI!".[13]

Once we start talking about building things like computers in the Game of Life (which we will do in Chapter 9), loops like this will be quite useful since they can serve as pieces of memory for the computer to make use of (for example, the gun in Figure 6.26 stores the bitstring "100111000011111000000", which we can read from memory by detecting the spaceships that are constantly emitted from the gun). To make loops like this truly useful for storing pieces of information though, we need to be able to easily alter the state of the memory (i.e., delete gliders from the loop and also insert new gliders into it).

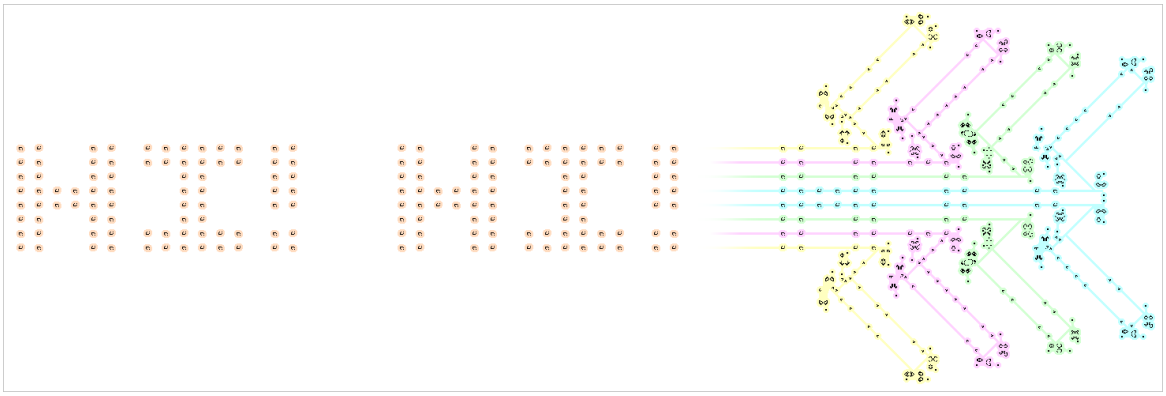


Figure 6.27: An arrangement of 8 loop-based guns that produce an array of lightweight spaceships that spell out "HI!" (highlighted in orange). Each gun produces one row of the output image (highlighted in yellow, magenta, green, and aqua), and the arrangement of gliders in each loop determines which lightweight spaceships are produced in their row.

Deleting a glider from the loop is simple, since we can just fire another glider from elsewhere in the Life plane so as to collide with it, thus deleting them both. To turn this deletion operation into an

[13] Alan Hensel put together the first ticker tape gun in June 1994, which printed out the digits "0123456789". It was adapted by Brice Due and Dave Greene in 2005 and 2006 to print out the Golly logo, and has been used as the header of the Golly homepage (golly.sourceforge.net/) ever since.

Free PDF. See conwaylife.com/book for pattern files and print version.

insertion operation, we can simply place two stream inverters along the loop and apply the deletion operation somewhere along its inverted portion. A pattern that makes use of these ideas is called a **memory cell**, and one is illustrated in Figure 6.28 along with the mechanisms for altering the contents of its memory.

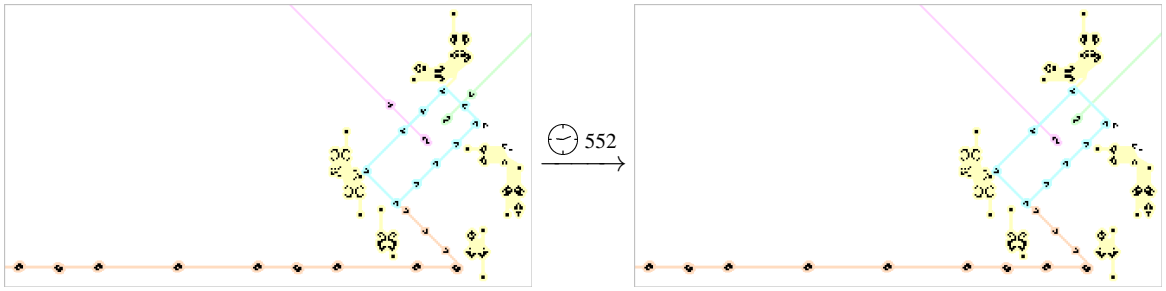
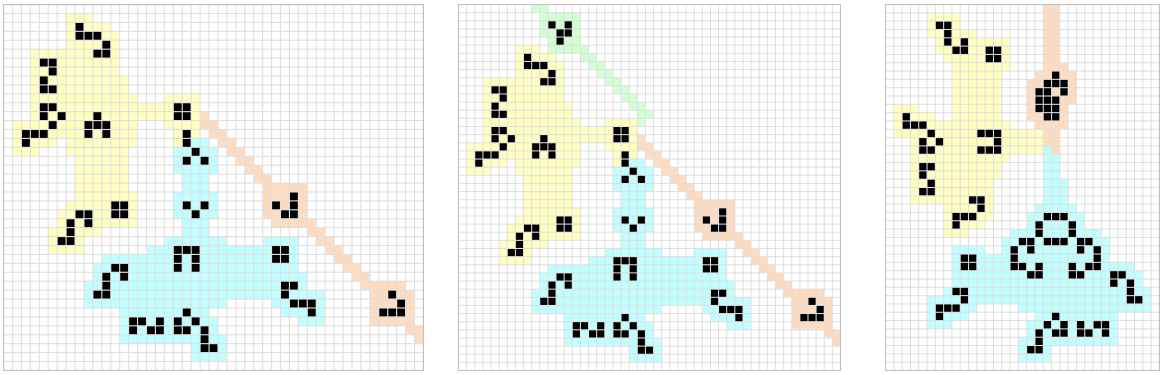


Figure 6.28: A period $12 \times 46 = 552$ memory cell for which the 12-bit string "111010111011" is encoded along its glider loop (highlighted in aqua) and is thus repeatedly fired out as a stream of lightweight spaceships (highlighted in orange). Bits can be switched from 1 to 0 (i.e., gliders can be deleted) by sending in a glider from the northwest (highlighted in magenta) to collide with the loop. Since the northeast side of the glider loop is inverted, bits can be switched from 0 to 1 (i.e., gliders can be inserted) by similarly sending in a glider from the northeast (highlighted in green) to collide with that side of the loop. As shown here, the 8th and 10th bits are flipped, thus changing the bitstring in memory to "111010**1**0**1**111".

6.3.2 Tanner's p46

One of the most useful tools for extending our period 46 toolset beyond just things that can be done with twin bees is the extremely sparky period 46 oscillator called **Tanner's p46** that we saw back in Figure 3.19(c). As is typical of extremely sparky oscillators, Tanner's p46 can be combined with other sparkers, or additional copies of itself, to create numerous useful objects. For example, it can be combined with a twin bees shuttle to create a new p46 glider gun (see Exercise 6.17), or it can be used to reflect a glider by 180 degrees (see Exercise 6.18), but neither of these facts are too exciting since we already know how to construct p46 glider guns and reflectors of roughly the same size via twin bees.



(a) an edge-shooting glider gun **(b)** an inline inverter **(c)** a small MWSS gun

Figure 6.29: Two guns formed by placing two copies of Tanner's p46 next to each other so that their sparks interact. The glider gun from (a) can be used as an inline inverter, as in (b), where the incoming glider (highlighted in green) prevents a glider from being fired from the gun at that point in the stream. The gun in (c) is the smallest known MWSS gun.

Free PDF. See conwaylife.com/book for pattern files and print version.

What is slightly more interesting is that we can carefully place two copies of Tanner's p46 next to each other so as to construct an **edge-shooting glider gun**,[14] as illustrated in Figure 6.29(a). Edge shooters like this one are particularly useful when trying to line up streams of gliders in tight positions, since they can be positioned as close as we like to existing glider streams on one side (compare this with guns like the Gosper glider gun and twin bees guns, which produce the stream of gliders near their center and thus require some clearance on either side of the other glider stream).

Furthermore, this glider gun can act as a p46 inline inverter (see Figure 6.29(b)) just like the Gosper glider gun can act as a p30 inline inverter.[15] Even more remarkably, two of these oscillators can be placed next to each other so as to create the p46 middleweight spaceship gun displayed in Figure 6.29(c),[16] which is currently the smallest known MWSS gun of any period, and also the smallest known gliderless gun of any period.

As an example of what can be done with the edge-shooting gun of Figure 6.29(a), we note that it can be used to make a heavyweight spaceship gun via the 3-glider HWSS synthesis that we saw back in Table 5.2, even though this synthesis contains closely spaced gliders that cannot be placed by the "usual" guns that we have seen so far. An example of such a gun is presented in Figure 6.30.

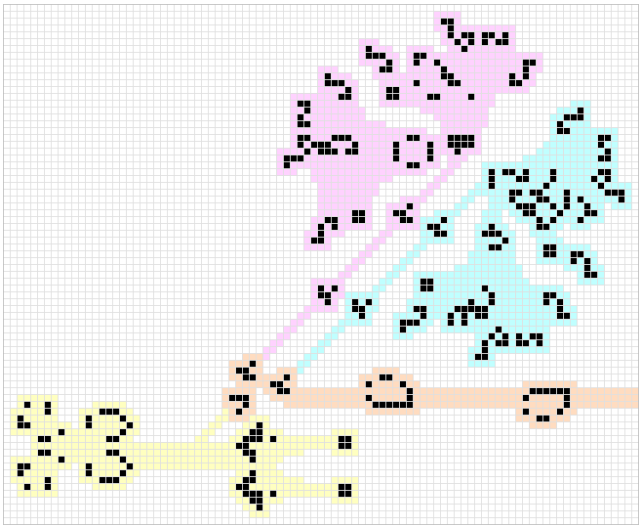


Figure 6.30: A period 46 heavyweight spaceship gun that uses two copies of the edge shooter from Figure 6.29(a) (highlighted in aqua and magenta) to place the required gliders in the HWSS synthesis close together.

6.3.3 Heisenburps

We saw back in Figure 6.10 that it is possible to construct periodic circuits that duplicate gliders, though the timing and positioning of the output gliders typically do not match those of the input glider.[17] We could of course use various other circuits to correct the positioning and timing of one of the output gliders, but remarkably there exist duplicators for which this is not necessary, as the input glider is not actually affected at all (even temporarily) by the duplication process.

[14] Found by David Bell just three days after Tanner's p46 itself was found.

[15] This inline inverter, just like the original p30 one, was found by David Bell.

[16] Found by Matthias Merzenich just one day after Tanner's p46 was found.

[17] In fact, we saw that there are *stationary* circuits that duplicate gliders way back in Figure 3.29(a), but these circuits will remain a bit more mysterious until the next chapter.

Free PDF. See conwaylife.com/book for pattern files and print version.

Duplicators of this type are called **Heisenburps**[18], and by far the simplest one known is displayed in Figure 6.31.[19] In this configuration, the spark from two perpendicular twin bees is suppressed slightly by a passing glider so that instead of dying off, the spark turns into another glider. Importantly, the passing glider itself is not affected at all in the process, but rather just serves to overpopulate other nearby cells (much like how we used induction coils to overpopulate and stabilize objects in Section 2.2).

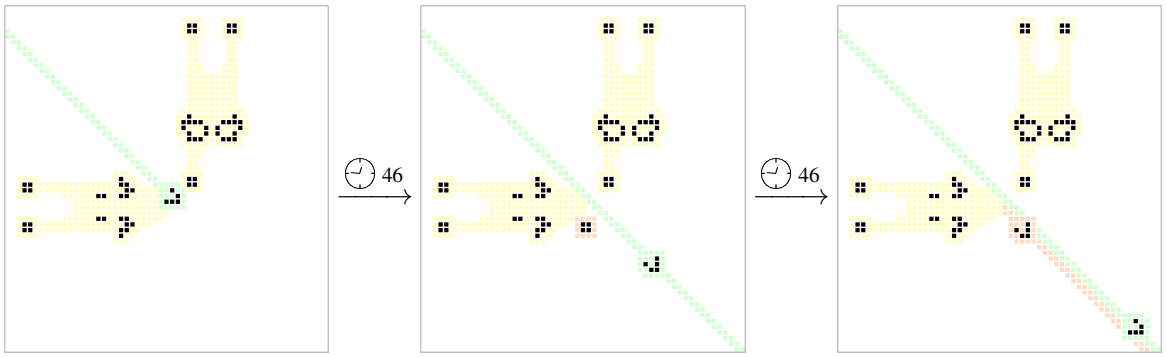


Figure 6.31: A small period 92 Heisenburp that uses two twin bees shuttles (highlighted in yellow) to copy a glider (highlighted in green) without affecting it. The first 46 generations are used to create a block (highlighted in orange) from the glider, and the next 46 generations are used to turn the block into another glider.

There are also Heisenburps that can copy other spaceships, or even use one type of passing spaceship to create a completely different kind of object—all that is needed is a spark whose evolution is changed sufficiently by the passing spaceship that it turns into something that moves out of the way. For example, the configuration of two twin bees shuttles in Figure 6.32, called the **MWSS out of the blue**,[20] detects a passing lightweight spaceship and releases a middleweight spaceship travelling in the opposite direction in response.

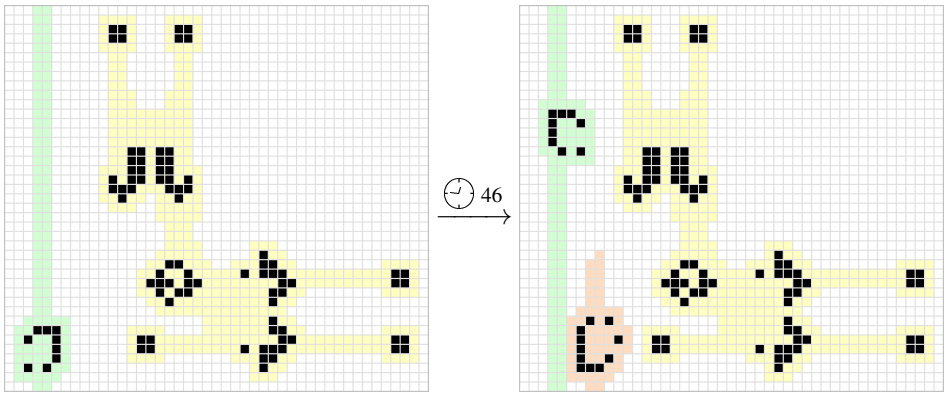


Figure 6.32: The **MWSS out of the blue** is a reaction in which two twin bees shuttles are triggered to create a middleweight spaceship when a lightweight spaceship passes by. The LWSS is not affected at all by the reaction.

There are also some reactions that can be used to construct Heisenburps via other periodic circuitry

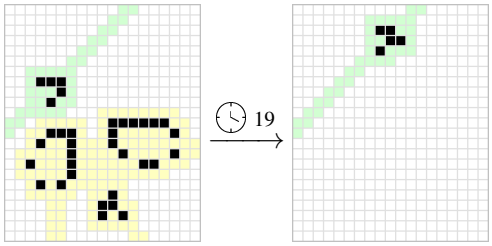
[18]Named for Heisenberg's uncertainty principle, which says that it is impossible to detect a particle without affecting it in some way. Heisenburps show that this principle does *not* hold in Conway's Game of Life, since we can detect and duplicate a particle (glider) without affecting it at all.

[19]Found by Brice Due in January 2005.

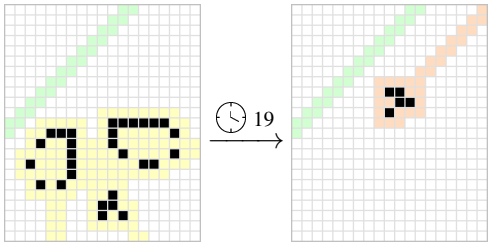
[20]Found by Peter Rott in November 1997.

Free PDF. See conwaylife.com/book for pattern files and print version.

that we have already seen. One of the simplest such reactions is the one displayed in Figure 6.33,[21] in which a middleweight and heavyweight spaceship collide with a glider in such a way that if another passing glider is present, then the three colliding spaceships cleanly destroy each other, but if that passing glider is absent then they produce a single glider. This reaction has a repeat time of 35 generations and thus can be used to construct Heisenburps of any period 35 or greater.



(a) Mutual annihilation if a nearby glider is present.



(b) Production of a glider if no nearby glider is present.

Figure 6.33: A collision of a glider, lightweight spaceship, and heavyweight spaceship (highlighted in yellow) that results in (a) nothing if there is a nearby passing glider (highlighted in green), and (b) a single glider (highlighted in orange) if there is not a nearby passing glider. Since the passing glider is unaffected by the collision, this reaction can be used as a basis for a Heisenburp.

To actually construct a Heisenburp from this reaction, we just combine reactions that we saw earlier—we know how to create streams of gliders, middleweight spaceships, and heavyweight spaceships, so we have no problem creating the three objects needed to fuel the reaction, and then we can just use the stream inverter of our choice to "fix" the duplicated output stream (since the reaction creates a glider if and only if an input glider is *not* present, which is the opposite of what we want). Figure 6.34 demonstrates one way to put these pieces together using period 46 circuitry, but any set of sufficiently high period circuits work.

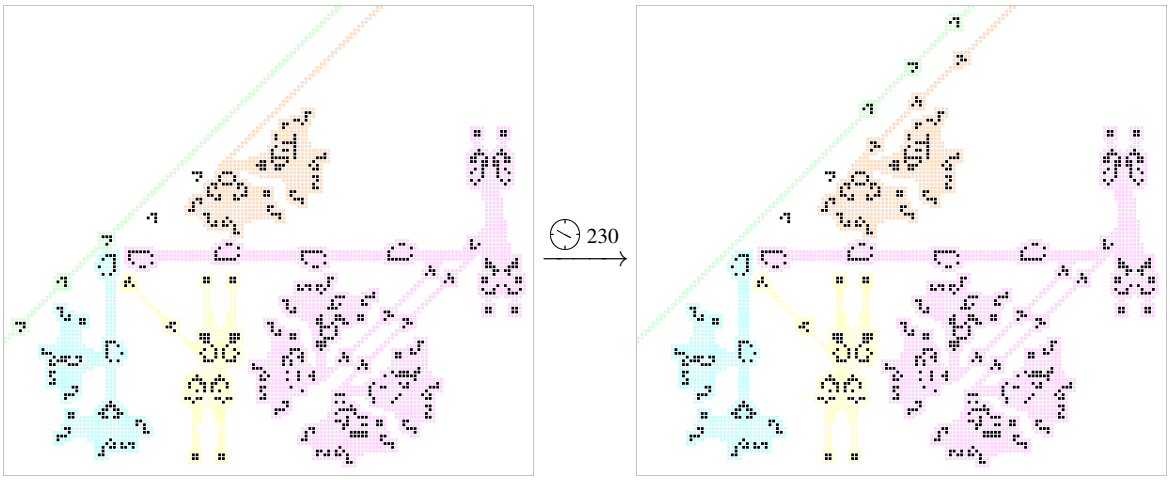


Figure 6.34: A period 46 Heisenburp that uses the glider + MWSS + HWSS reaction from Figure 6.33. These spaceships are all provided by guns that we have already seen—the MWSS gun (highlighted in aqua) and HWSS gun (highlighted in magenta) were introduced in Figures 6.29(c) and 6.30, respectively. The central three-spaceship collision produces a (non-highlighted) glider traveling northeast, which is destroyed by an inline inverter (highlighted in orange and originally introduced in Figure 6.29(b)). However, if an input glider (highlighted in green) is present then no glider is fed into the inline inverter, leading to the input glider's duplication.

[21] Found by Jason Summers in June 1999, as was the first-known period 46 Heisenburp based on it.

Free PDF. See conwaylife.com/book for pattern files and print version.

In fact, it is even possible to construct *stable* Heisenburps (i.e., Heisenburps made up entirely of still lifes, rather than oscillators and guns). However, they work a bit differently than their periodic counterparts, instead using a spark from the to-be-detected spaceship to cause a still life to explode into a usable signal. We describe in more detail how to construct stable xWSS Heisenburps via this technique in Section 7.8.2. However, a glider Heisenburp must have an oscillating component, as gliders have no accessible sparks that could be detected by still lifes.

6.4 Bumpers and Bouncers

Some periodic circuitry works based only on the existence of certain sparks, and thus can be made to work at any period for which we know of oscillators that emit that spark. For example, the glider-reflecting reactions of Figure 3.17 fall into this category, since any oscillator that produces the correct banana spark (such as a buckaroo) or duoplet spark (such as the twin bees shuttle) can be used to implement the reflection. However, most of the oscillators that emit these sparks have rather large period, so we now introduce two other spark-based glider reflectors that can make use of low-period oscillators.

The first of these reactions is called the **bumper** (displayed in Figure 6.35),[22] which consists of an eater 1, a loaf, and a spark of almost any type. This reflector is versatile enough that it can be used with a wide variety of sparkers, including some with period as low as 3 (see Figure 6.35 for numerous examples). Since this reflector is color-preserving, just like the Snark, it perhaps does not seem too exciting at first glance (after all, stable reflectors work at *any* period). However, there are at least three reasons why bumpers are still useful:

1) It is smaller (and, for many periods, easier to synthesize) than the Snark.
2) The Snark has a repeat time of 43 generations, whereas the bumper has a repeat time of just 34 generations. However, the input glider stream must be compatible with the period of the oscillator, so for example the p3, p4, and p6 bumpers actually have repeat time 36, and the p5 and p7 bumpers actually have repeat time 35.
3) Its reflected gliders have different timing than those reflected by the Snark. Specifically, a bumper produces a glider that has timing exactly 5 generations delayed from that of a glider reflected by a Snark. This is a useful feature when fine-tuning the positioning of gliders along glider tracks, as we can replace Snarks by bumpers to rephase the gliders (similar to how we used a wide variety of different one-time reflectors in Table 5.5 to get precise glider timings).

There is another closely related reaction called the **bouncer** (displayed in Figure 6.36),[23] which consists of an eater 1, a boat, a block, and a domino spark that is typically provided by a pipsquirter. The bouncer is somewhat less versatile than the bumper since the domino spark is tricky to position correctly, but it also has the advantage of having a repeat time of just 22 generations (versus the bumper's 34 and the Snark's 43). Furthermore, these reflectors are color-changing,[24] which provides us with some extra flexibility that we did not yet have (recall that both the Snark and the bumper are color-preserving).[25]

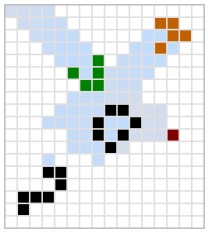
[22] The p3 bumper was found in April 2018 by Arie Paap, but the reaction itself and most of the other bumpers were found in April 2016 by Tanner Jacobi.

[23] The bouncer, as well as most of the low-period pipsquirters that it can use, were found by Noam Elkies in September 1998.

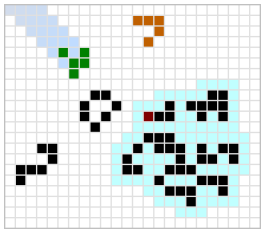
[24] Their names are designed to help us remember which reaction is color-preserving and which one is color-changing—the bum**p**er is color-**p**reserving while the boun**c**er is color-**c**hanging.

[25] For a more complete and up-to-date collection of bumpers and bouncers, see conwaylife.com/wiki/Bumper_and_bouncer_gallery.

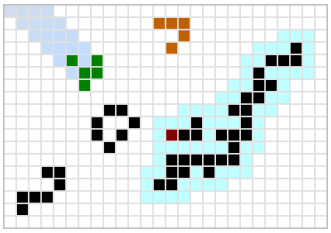
Free PDF. See conwaylife.com/book for pattern files and print version.



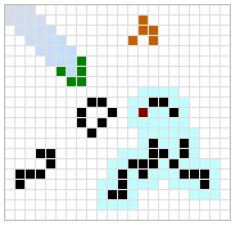
(a) The bumper itself. A spark must be present at the location marked in red 11 generations after the input glider's phase shown here.



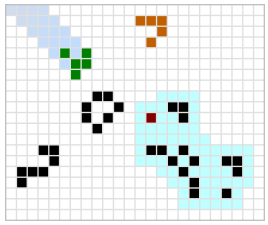
(b) A p3 bumper that uses a custom oscillator.



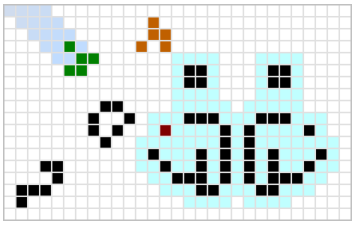
(c) Another p3 bumper that uses a custom oscillator.



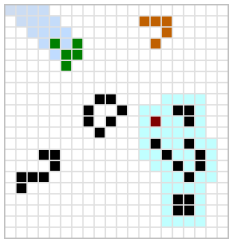
(d) A p4 bumper that uses a custom oscillator.



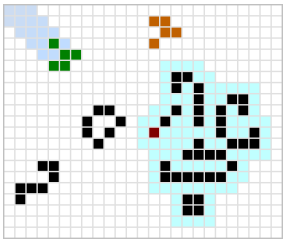
(e) A p6 bumper reflector that uses the unix sparker.



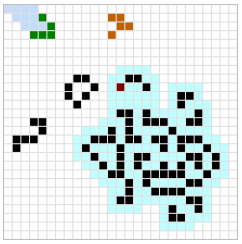
(f) A p7 bumper that uses Beluchenko's p7 from Figure 3.13(c).



(g) A p8 bumper reflector that uses the blocker.



(h) A p9 bumper reflector that uses the p9 thumb sparker.



(i) A p11 bumper reflector that uses a custom oscillator.

Figure 6.35: The bumper is a glider reflector that relies on a single spark. Because the spark is somewhat distant from the reflection reaction itself, it can be provided by a wide variety of different oscillators (outlined in aqua) and can thus work with many different periods. Note that the period 4 and 11 bumpers use a slightly different spark based on hassling a block. Some bumpers of other periods (including a very useful p5 bumper) are explored in Exercise 6.24.

6.5 Glider Timing and Regulators

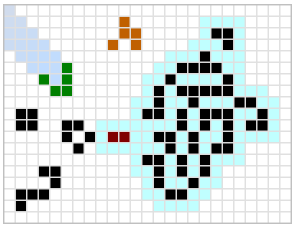
Oftentimes when trying to construct a large pattern, we know there will be a signal coming in from somewhere else in a mechanism—usually a glider arriving on some particular path. Once this input comes in, we want to perform a specific action, but it should only occur on a specific schedule, so as to be compatible with some set of circuitry. For example, we might want to send out a glider to be reflected back by a receding spaceship that has a known speed and position, but if the timing isn't exactly right then the reflection reaction won't work.

To illustrate how we can overcome this problem, suppose for now that we have a glider at some specific location, and we need to relocate it so that it appears at another pre-specified location with a certain timing. If the input glider is on some kind of schedule—say, it only has a possibility of showing up once every 30 ticks, or more generally once every n ticks for any reasonably high value of n—then this job is easy. We just line up a period n gun with an inverter of the same period, and then find a way for the arriving glider to cleanly delete a glider from the stream that hits the inverter.

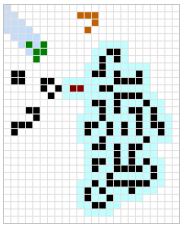
Free PDF. See conwaylife.com/book for pattern files and print version.



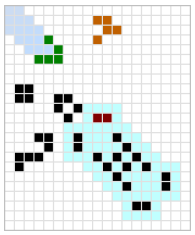
(a) The bouncer itself. A spark must be present at the red location, or one cell farther south, 14 generations after the input glider's phase shown here.



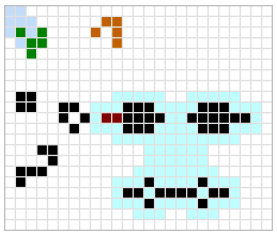
(b) A p6 bouncer.



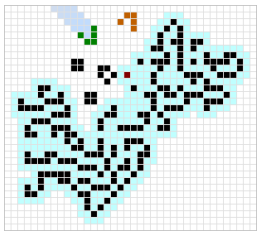
(c) A p7 bouncer.



(d) A p8 bouncer that uses a figure eight.



(e) A p15 bouncer based on an oscillator called **Karel's p15**.



(f) A closely related p5 reflector that uses a dot sparker.

Figure 6.36: Pipsquirters (as well as a handful of other domino sparkers) can be used to with the bouncer to reflect gliders. This allows for the creation of several new reflectors with many periods.

The inverter then sends out a glider in response to the resulting hole in the stream, thus producing an output glider at the desired location and timing.

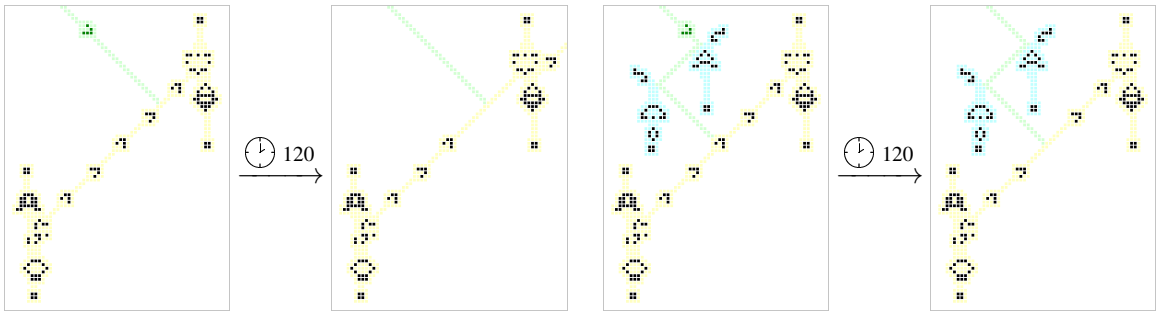
The only somewhat tricky part here is getting the input glider to cleanly destroy a single glider in the stream that will be inverted, since we do not have control over the inverter's position or timing (they are determined by where and when we want the output glider to appear). However, we can change the positioning and timing of the input glider in a variety of ways, the simplest of which is to add some additional reflectors to the input glider's path (see Figure 6.37). Some trial-and-error is required here to produce a configuration of reflectors that results in the desired clean glider annihilation, but sooner rather than later, along the same lines as Exercises 5.1–5.4, a workable two-glider collision will be found. As shown in Table 5.1, there are a lot of options that result in clean mutual annihilation—as long as we can make adjustments to change the collision timing, the odds are heavily in our favor that we will find one.

What if n (the period to which the input and output gliders are aligned) is low enough that we can't build a period n gun? For example, suppose that all we know about the input stream of gliders is that they are separated by generation gaps that are multiples of 8. We cannot build a period 8 gun to aim at an inline inverter (recall that the lowest-period glider stream possible is period 14), but we could instead reflect the glider with Snarks and period 8 bouncers and bumpers, which can be used to produce any possible output lane and sufficiently slow timing.

What if we go one step further and suppose that we do not know *anything at all* about the input glider's timing, but we still want to be able to insert it into periodic circuitry (i.e., we want to align that glider to some specific period)? Unfortunately, neither of the previously mentioned techniques work in this scenario:

- We cannot simply have the input glider interrupt a stream from a glider gun as in Figure 6.37. That method will work for some timings, but other timings will result in multiple gliders being

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) A glider (unfortunately) destroying 3 gliders. **(b)** Once re-timed, just one glider is destroyed.

Figure 6.37: An input glider (highlighted in green) being re-timed to fit with existing p30 circuitry by a Gosper glider gun firing into an inline inverter (highlighted in yellow). As positioned in (a), the input glider destroys three of the gliders in the stream, so three gliders are released to the northeast. We can correct this problem by using two buckaroos (highlighted in aqua in (b)) to reflect the glider twice and thus change the timing with which it hits the glider stream between the two Gosper glider guns. After some trial-and-error, we find the configuration in (b) where just a single glider is destroyed.

deleted or even chaotic explosions.

- We cannot simply reflect the input gliders to get whatever timing we want, as we can only arrange reflectors so as to get the first glider on the input stream to have the correct timing—not subsequent gliders (e.g., if two gliders come in along the input stream with a separation of 58 gliders, there is no way to use just Snarks so as to widen that gap to 60 generations).

What we need instead is a reaction that always works the same way, producing an output glider aligned to a particular period, no matter what the precise timing of the input glider is (as long as the separation between consecutive input gliders is no smaller than the output period). A mechanism that accomplishes this for a particular output period is called a **regulator**, and a mechanism that can be adjusted to work at any period is known as a **universal regulator**.

6.5.1 A Universal Regulator

We now describe how the first universal regulator was built.[26] The heart of this regulator is actually the boat bit from way back in Section 2.3.1, which is displayed again in Figure 6.38. In particular, if we use an eater 1 (instead of a snake as in Figure 2.17), then a duoplet spark can be used to test whether or not the boat bit is present (i.e., whether or not the input glider has arrived yet). In particular, the duoplet spark does not come close enough to the eater 1 to affect it, but if the boat bit is present then the boat and eater are both destroyed, producing an output glider as in Figure 6.39. A beehive is also created, but this can easily be destroyed later.

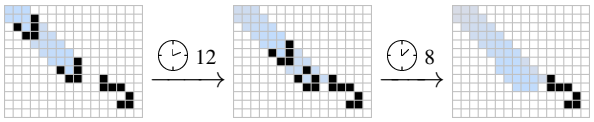


Figure 6.38: A boat-bit using an eater 1. The first input glider creates a boat, and the second destroys it.

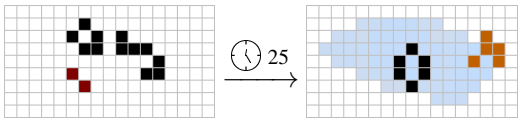


Figure 6.39: A duoplet spark turning a boat bit into a glider and beehive.

The reason that this reaction is so useful is that the output glider always has a precise known timing relative to the creation of the duoplet spark, rather than relative to the timing of the input glider. Thus if we create the duoplet spark on some periodic schedule then the output glider will

[26] By Paul Chapman in March 2003.

Free PDF. See conwaylife.com/book for pattern files and print version.

necessarily follow the same schedule. We could use a known oscillator (like the twin bees shuttle, as in Figure 3.16(a)) to create this spark, but the downside of doing so is that the regulator will then be tied to the period of that oscillator. Instead, we use a spaceship collision to create it, as we are then only restricted to periods for which we know how to create glider guns (which, as we will see in Chapter 8, is not actually a restriction at all). One method of colliding a glider and an LWSS to create this spark is shown in Figure 6.40, as is the entire glider → boat bit → synchronized glider conversion process. This glider-and-LWSS collision has the additional advantage of destroying the beehive that was left behind by the duoplet spark in Figure 6.39, so streams of these two spaceships clean up the mess from their own reaction.

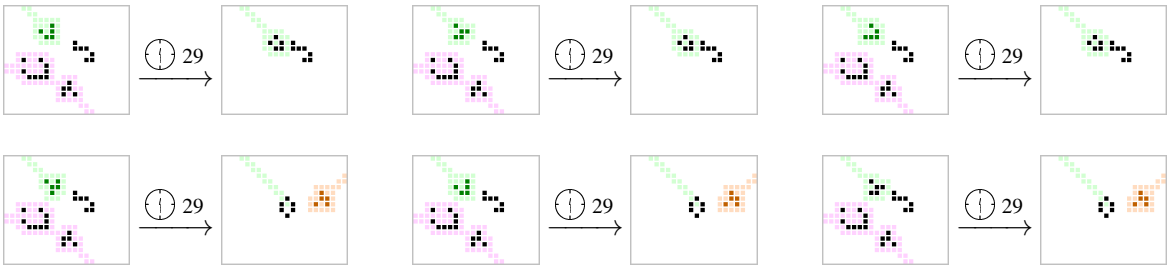


Figure 6.40: An input glider (highlighted in green) with six different timings hitting an eater 1 so as to become a boat bit. The glider-and-LWSS collision (highlighted in magenta) creates a duoplet spark that turns that boat bit into an output glider (as in Figure 6.39). If the timing is such that the boat bit was already created before the collision occurs, as in the bottom three input timings, an output glider is created (highlighted in orange), and its timing does not depend on that of the input glider. If the boat bit has not yet been formed, as in the top three input timings, nothing happens and the output glider will not be created until we repeat the glider-and-LWSS collision.

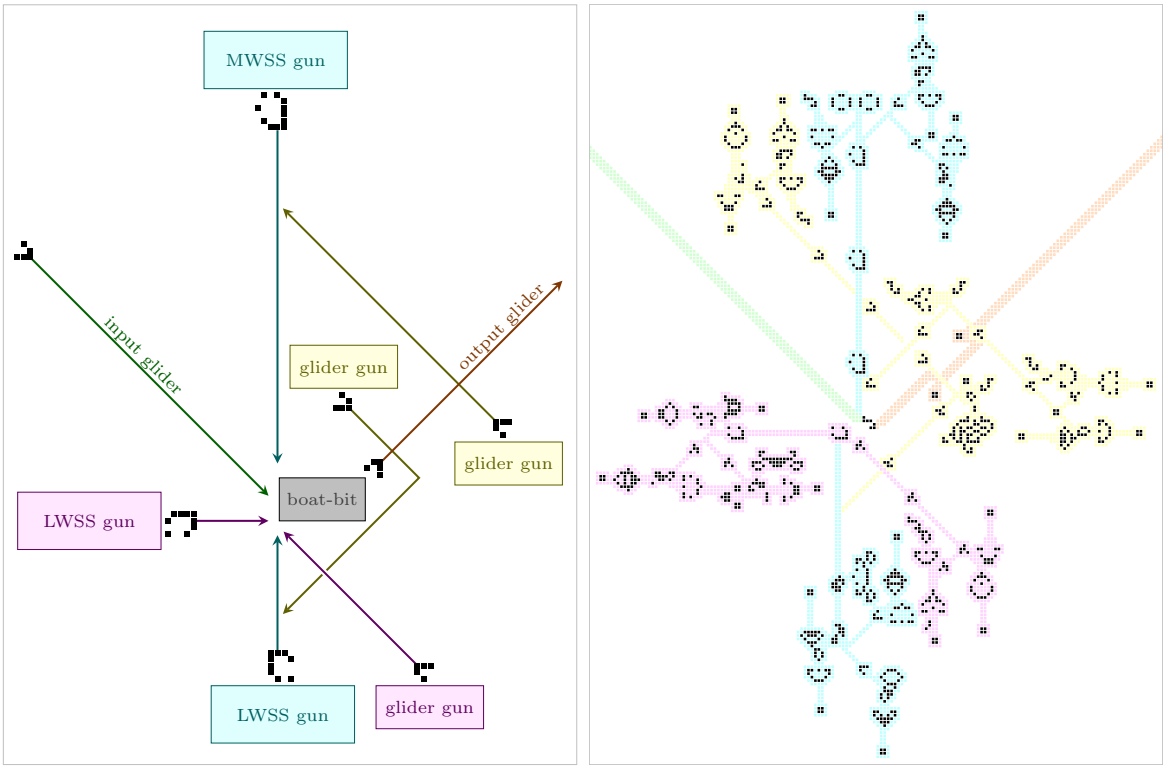
We now have almost everything that we need to construct a universal regulator. The only trick is that we need a way to reconstruct the eater 1 after it is destroyed by the reaction in Figure 6.39. We have already seen that two gliders can be used to synthesize an eater 1, but for spacing reasons we will instead use a head-on LWSS-and-MWSS collision. However, if we just aim LWSS and MWSS guns at each other so as to implement this synthesis, we arrive at a problem: if no input glider is detected during a period, then the LWSS/MWSS pair will collide with an already-existing eater 1 instead of synthesizing it. This problem can be resolved by using streams of gliders to suppress the lightweight and middleweight spaceships, and then using the output glider to suppress one of those suppressing gliders (thus letting a single LWSS and MWSS through, so the eater 1 is rebuilt).

A schematic for, and period 60 implementation of, this universal regulator are displayed in Figure 6.41, but we emphasize that a universal regulator with any (sufficiently large) period can be constructed using these same reactions by swapping out all of the period 60 machinery for reactions and guns that work with other periods. It is also worth noting that we have not yet explicitly seen some of the mechanisms that we use here to turn period 30 guns into period 60 guns. Techniques like these will be covered in depth in Chapter 8 (see Exercise 6.31 as well).

6.6 Notes and Historical Remarks

Because of their simplicity, many of the period 30 and period 46 mechanisms that we investigated in this chapter were discovered as early as the 1970s and '80s, so most of the earliest "interesting" patterns that were constructed in Conway's Game of Life make use of these components. Dean Hickerson built many of the most well-known of these constructions throughout the 1990s, including the first primer (which made use of essentially the same techniques, but slightly different components,

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) A schematic for a universal regulator.

(b) A period 60 implementation of this regulator.

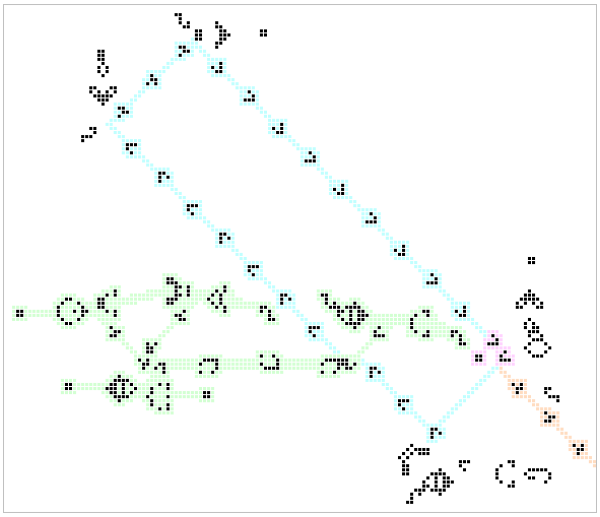
Figure 6.41: A universal regulator's (a) schematic and (b) period 60 implementation. This device takes in a glider with *any* timing from the northwest input lane (highlighted in green) and produces an output glider along the northeast lane with timing that is synchronized to some period (highlighted in orange). The left LWSS gun and bottom-right glider gun (highlighted in magenta) are used to create the duoplet spark reaction from Figure 6.40 that is at the heart of the regulator. The top MWSS gun and bottom LWSS gun (highlighted in aqua) then rebuild the central eater 1 (thus allowing more input gliders), but only if the output glider has destroyed a glider from each of the suppressing glider streams (highlighted in yellow). These suppressing glider paths in the regulator (b) were adjusted for spacing reasons.

as the one we constructed in Section 6.2). Some other interesting constructions that he made mostly from these components include:

- A **twin primer**—a gun that emits a stream of lightweight spaceships with the property that the n-th spaceship in the stream is present if and only if both $n-2$ and n are prime (see Exercise 6.9). This pattern is particularly interesting since it is currently unknown whether or not there are infinitely many twin primes (in fact, this is one of the biggest open problems in all of mathematics), so it is unknown whether or not it emits infinitely many lightweight spaceships.
- A pseudo-random glider generator based on using the toggle from Figure 6.14 within a glider loop. If the output from a toggle is looped around and fed back into itself, the resulting stream of gliders ends up looking almost completely random as it turns itself off and on in increasingly erratic intervals. If we then place a glider duplicator somewhere on the loop, this random-looking stream of gliders can be emitted as an extremely high-period gun.
  Specifically, if there is space for n gliders in the loop and we think of the k-th emitted glider as a bit $b(k)$ (where the k-th glider's presence means $b(k)=1$ and its absence means $b(k)=0$),

Free PDF. See conwaylife.com/book for pattern files and print version.

then the effect of the toggle is that $b(k) = b(k-1) \oplus b(k-n)$ for all $k > n$.[27] This sequence of bits is extremely erratic and can have very large period even when n is relatively small [Slo99]. For example, the $n = 25$ case is illustrated in Figure 6.42, where the period of the sequence of bits is 10961685 and thus the period of the gun itself is $30 \times 10961685 = 328850550$.



(a) the pseudo-random gun

```
1111111111111111111111111
0101010101010101010101010
0110011001100110011001100
0100010001000100010001000
0111100001111000011110000
0101000001010000010100000
0110000001100000011000000
0100000001000000010000000
0111111110000000011111111
1010101011111111101010101
0011001101010101001100110
0010001001100110001000100
0011110001000100001111000
0010100001111000001010000 ···
```

(b) the first 400 bits of its output

Figure 6.42: A period $30 \times 10961685 = 328850550$ pseudo-random glider gun. It sends gliders around a $25 \times 30 = 750$-generation track (highlighted in aqua) that contains a toggle (highlighted in green) and the glider duplication/inversion reaction introduced in Figure 6.9 (highlighted on the right in magenta). By feeding the output of the toggle back into itself, the arrangement of 25 gliders along the track becomes quite unpredictable and random-looking, and thus so does the output glider stream (highlighted in orange) that follows the 10961685-bit string whose first 400 bits are displayed in (b).

- A pattern that uses four breeders that aim Gosper glider guns so as to repeatedly invert each other and have asymptotic growth related to $\pi \approx 3.14159$. This pattern fills the plane with triangular regions that alternate back and forth between being filled and being empty.
  Specifically, if we use $T(n,t)$ to denote the triangle in the northeast quadrant with vertices at $(0,0)$ (i.e., the center of the pattern), $(0,t/(2n))$, and $(t/(2n+4),0)$, then in generation t the triangle $T(1,t)$ is filled with gliders except that the smaller triangle $T(3,t)$ within it is empty, except that the smaller triangle $T(5,t)$ within it is filled with gliders, except that $T(7,t)$ is empty, and so on (see Figure 6.43).

  Since the triangle $T(n,t)$ has area

$$A(n,t) = \frac{1}{2} \times \text{base} \times \text{height} = \frac{1}{2} \times \frac{t}{2n+4} \times \frac{t}{2n} = \frac{t^2}{16}\left(\frac{1}{n} - \frac{1}{n+2}\right),$$

  the number of cells in the glider-filled portion of the Life plane in the northeast quadrant in generation t (when t is large) is approximately

$$\begin{aligned} A(1,t) - A(3,t) + A(5,t) - \cdots &= \frac{t^2}{16}\left(\left(1 - \frac{1}{3}\right) - \left(\frac{1}{3} - \frac{1}{5}\right) + \left(\frac{1}{5} - \frac{1}{7}\right) - \cdots\right) \\ &= \frac{t^2}{16}\left(1 - \frac{2}{3} + \frac{2}{5} - \frac{2}{7} + \cdots\right). \end{aligned} \tag{6.1}$$

[27] Here, $\oplus$ refers to the XOR, or mod 2 addition, of two bits. That is, $0 \oplus 0 = 1 \oplus 1 = 0$ and $0 \oplus 1 = 1 \oplus 0 = 1$.

Free PDF. See conwaylife.com/book for pattern files and print version.

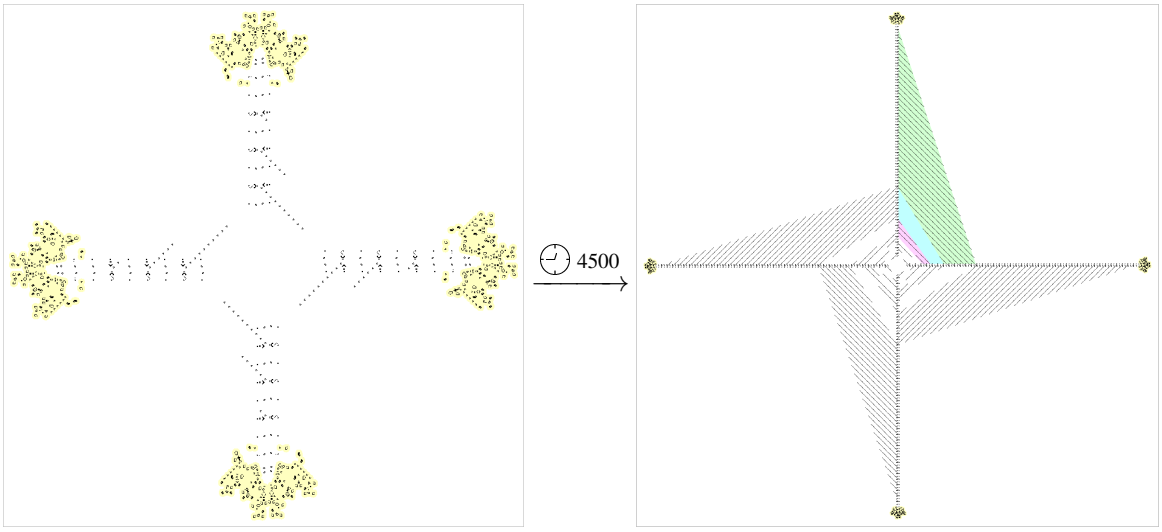


Figure 6.43: An arrangement of four breeders (highlighted in yellow) that produce Gosper glider guns that fire at each other so as to invert each others' streams (shown on the left after the breeders have been moving away from each other for 500 generations). In generation t (shown here on the right with $t = 5000$), the green triangular region $T(1,t)$ is filled with gliders, except the aqua triangular region $T(3,t)$ within it is empty, except the magenta triangular region $T(5,t)$ within it is filled with gliders, and so on.

If we use the fact that $\arctan(x)$ has Taylor series $\arctan(x) = x - x^3/3 + x^5/5 - \cdots$ for $-1 \le x \le 1$, we learn that $\arctan(1) = 1 - 1/3 + 1/5 - \cdots$. If we also use the fact that $\arctan(1) = \pi/4$, and combine these equalities with the sum (6.1), we learn that the occupied area in the northeast quadrant contains roughly

$$\frac{t^2}{16}\left(1 - \frac{2}{3} + \frac{2}{5} - \frac{2}{7} + \cdots\right) = \frac{t^2}{16}\big(2\arctan(1) - 1\big) = \frac{(\pi-2)t^2}{32}$$

cells (when t is large). Since the gliders fill this occupied area with density $1/90$, and this same calculation applies to all four quadrants of the plane, we see that this pattern's population in generation t (again, when t is large) is approximately

$$\frac{4}{90} \times \frac{(\pi-2)t^2}{32} = \frac{(\pi-2)t^2}{720}.$$

In fact, John Conway and Bill Gosper used little more than some basic period 30 circuitry way back in the 1970s and 1980s [BCG82, Chapter 25] to demonstrate that the Game of Life is **universal**—anything that can be computed (by a regular computer, for example), can be computed in the Game of Life by cleverly arranging some period 30 circuitry and manipulating gliders. Thus, for example, we knew that prime numbers could be computed in the Game of Life for several years before the first primer was explicitly constructed.

The advantage of the new circuitry and constructions that we have available to us now is that they are significantly smaller than the ones that would result from implementing the construction outlined by Conway and Gosper's universality proofs (which require sending gliders across absolutely vast distances for even the simplest of computations). There is also something quite appealing about having *explicit* constructions of these patterns. Actually arranging components as required by Conway and Gosper's proofs would typically be infeasible in practice, and the resulting patterns would not be terribly interesting to look at. We will return to the topic of computational universality in Chapter 9.

Free PDF. See conwaylife.com/book for pattern files and print version.

Exercises

solutions to starred exercises on page 460

*6.1 [1/5] In the glider gun in Figure 6.8(a), we used eater 1s instead of blocks to stabilize some of the Gosper glider guns (as in the buckaroo of Figure 3.16(b)). Explain why.

6.2 [1/5] Construct a glider gun based on the inline inverter with...

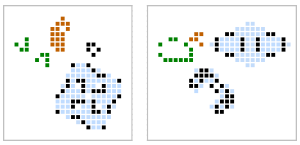
(a) period 240.
(b) period 360.

6.3 [2/5] Use two inline inverters to create a pattern that advances a stream of gliders by 20 generations, similar to how the fast forward force field of Figure 4.44 advances a stream of lightweight spaceships.

6.4 [2/5] Construct a period 240 lightweight spaceship gun...

(a) using three inline inverter guns.
(b) using one inline inverter gun and the reaction from Figure 6.12.

6.5 The periodic circuits displayed below can be used to convert two gliders into an HWSS and an HWSS back into a glider.[28]



(a) [2/5] Use the left reaction to create a period 120 HWSS gun.
(b) [3/5] Create a glider loop that uses both of these reactions to convert a glider into an HWSS and back.

6.6 The arrangement of queen bees below can be used to convert any xWSS into a glider.

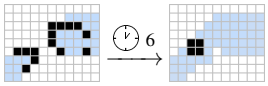


(a) [3/5] Use this reaction as well as one of the reactions from Exercise 6.5 to create a glider loop that converts a glider into an HWSS and back.
(b) [1/5] Use this reaction to create a gun that emits a glider stream (rather than an LWSS stream as in Figure 6.22) that is spaced according to the prime numbers.

6.7 [2/5] Construct an oscillator that makes use of the reactions in each of Figures 6.12 and 6.13 to send a glider around a track in such a way that it is converted into an LWSS and back during each loop.

6.8 [3/5] Construct a pattern that emits a stream of lightweight spaceships with the property that the n-th spaceship in the stream is present if and only if n is not a multiple of 2, 3, or 5.

6.9 [3/5] The collision of a glider with an LWSS shown below is called a **filter**. It results in a single block that destroys the next incoming glider, letting the next incoming LWSS pass by unharmed, thus doubling the period of an LWSS stream.



Add this reaction (and potentially some guns and eaters, as necessary) to the primer to create a **twin primer**—a pattern that creates a stream of lightweight spaceships for which the n-th LWSS in the stream is present if and only if $n-2$ and n are both prime.

6.10 [4/5] One method of making a pattern smaller is to change it into the form that it would have looked several generations earlier (this is called **rewinding** the pattern). Use this technique, along with moving its various components closer together, to reduce the area of the bounding box of the primer from Figure 6.22 from $951 \times 696 = 661\,896$ to 500 000 or less.

6.11 [2/5] Use the patterns from Figure 6.23 to create a loop oscillator in which a glider is converted into a lightweight spaceship and back at some point in the loop.

6.12 [2/5] Use the patterns from Figure 6.24 to construct a period 46 middleweight spaceship gun.

6.13 [1/5] Modify the sequence of gliders in the loop of the gun from Figure 6.26 so that it produces lightweight spaceships corresponding to the bitstring "11011101001101110100 1".

6.14 [3/5] Rebuild the ticker tape gun from Figure 6.26 so that it uses p30 circuitry instead of p46. Relevant mechanisms can be found in Figures 6.10 and 6.12.

6.15 [3/5] Using the techniques used in Figure 6.27, create a ticker tape gun that produces smiley faces.

[28] These reactions were found and simplified by David Buckingham, Bill Gosper, Dieter Leithner, and Peter Rott.

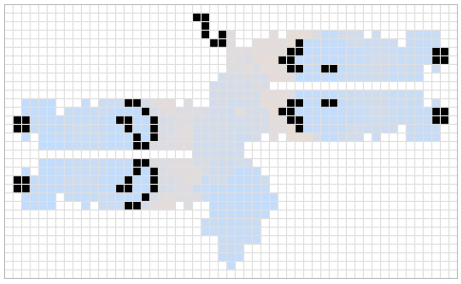
Free PDF. See conwaylife.com/book for pattern files and print version.

6.16 [2/5] Fire some gliders at the memory cell from Figure 6.28 so as to change the bitstring that it stores to "101010101010".

6.17 [3/5] Create a glider gun that works by colliding the spark from Tanner's p46 with a twin bees shuttle that is stabilized only on one side.
[Hint: This gun can be found by hand via trial and error, or you could write a computer program to try lots of positionings.]

6.18 [3/5] Show how Tanner's p46 can be used to reflect a glider by 180 degrees.
[Hint: The phi spark emitted by Tanner's p46 is quite similar to some of the sparks emitted by the pentadecathlon. Mimic the reflection from Figure 6.4(a).]

6.19 An arrangement of two twin bees shuttles that is particularly useful due to the extremely large spark that it produces is displayed below.



- **(a)** [3/5] Collide the spark from this object with that of Tanner's p46 to create a glider gun.
  [Hint: Writing a computer program might be helpful here.]
- **(b)** [4/5] Collide the spark from this object with the large-spark version of the twin bees shuttle from Figure 6.23 so as to create a double-barrelled glider gun (i.e., a gun that emits two side-by-side gliders per period).[29]

6.20 [3/5] Use the 9-glider synthesis from Exercise 5.14 to construct a gun that fires 2-engine Corderships.
[Hint: Start off by choosing an appropriately large period. Corderships cannot follow behind each other at period 30 or 46, for example, so you need to use larger-period guns.]

6.21 [3/5] Construct a p46 LWSS gun, and then apply an MWSS out of the blue to its output to turn it into a p92 LWSS gun.
[Hint: The MWSS that the MWSS out of the blue reaction produces can collide with an LWSS so that they cleanly destroy each other.]

6.22 [4/5] Construct a version of the Heisenburp from Figure 6.34 that uses period 30 circuitry instead of period 46.

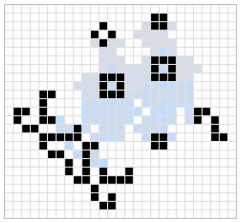
∗**6.23** [1/5] Explain why it is not possible to construct glider loop oscillators consisting only of the following sets of reflectors, and experiment with them to convince yourself that they really are impossible.

- **(a)** Three bumpers and one bouncer.
- **(b)** Three bouncers and one Snark.

∗**6.24** Construct a bumper reflector with...

- **(a)** [1/5] period 16.
- **(b)** [1/5] period 22.
- **(c)** [1/5] period 4, making use of the fountain sparker.
- **(d)** [2/5] period 15.
- **(e)** [3/5] period 5.
  [Hint: Modify the sparker from Figure 6.36(f).]

∗**6.25** [2/5] Use the **skewed p29 pre-pulsar shuttle** displayed below to create a p29 bouncer reflector.[30]



6.26 [3/5] Use the reaction from Figure 6.9, together with reflectors and eaters as necessary, to create a device that duplicates and inverts a glider stream with...

- **(a)** period 43.
- **(b)** period 36.
- **(c)** period 25.

∗**6.27** [2/5] Explain why the period 11 bumper from Figure 6.35(i) is not particularly useful for glider streams of period below 121.
[Hint: There are smaller reflectors that can be used in its place.]

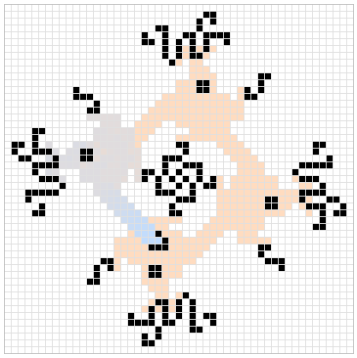
6.28 [2/5] There are two reflectors in Figure 6.41(b) that reflect yellow glider streams. What are the names of these reflectors (we have discussed them each earlier)?

6.29 [2/5] What is the repeat time of the universal regulator from Figure 6.41(b)? Explain why it is not 60.

[29] This double-barrelled gun was originally found by Dieter Leithner.
[30] This reflector was originally found by Matthias Merzenich in August 2013.

Free PDF. See conwaylife.com/book for pattern files and print version.

∗**6.30** The smallest-period single-glider Snark loop has period 216 and is displayed below. Note that the still life in the center of this loop is the weld of four eater 1s from Exercise 2.9(c).



(a) [1/5] Move each of the Snarks outward by 1 cell (e.g., move the northern Snark north by 1 cell) and un-weld the central eater 1s. What is the period of the resulting glider loop? Explain how you could have determined this period without explicitly constructing this modified glider loop.

(b) [2/5] Replace the Snarks in the glider loop from part (a) by p4 bumpers in a way that keeps the glider on the same lanes. What is the period of the resulting glider loop? Explain how you could have determined this period without explicitly constructing this modified glider loop.

(c) [1/5] Replace one of the p4 bumpers in the glider loop from part (b) with a p6 bumper.

(d) [1/5] If you replace one of the p4 bumpers in the glider loop from part (b) with a p5 bumper, the loop stops working. Why?

(e) [2/5] Move some of the bumpers in the glider loop from part (b) closer together so as to alter the period of the loop. Once you have made a loop with a suitable period, replace one of the p4 bumpers by a p5 bumper.

∗ **6.31** [2/5] The lightweight spaceship guns in Figure 6.41(b) each contain a middleweight emulator. Describe what role this oscillator plays in the gun.

∗**6.32** Changing the number of gliders in the loop portion of the pseudo-random glider-generating gun from Figure 6.42 can drastically change its period.

(a) [1/5] If you move the northwest portion of the loop northwest by an additional 15 cells (and rephase reflectors as necessary), how many additional gliders fit in the loop?

(b) [3/5] Determine the period of the gun constructed in part (a). [Hint: The period is massive. Use computer software to help you and/or compute the period from the recurrence relation given in the text.]

(c) [2/5] Move the northwest portion of the loop northwest by an *additional* 15 cells (again, rephase reflectors as necessary). Determine the period of this gun. [Hint: Its period is now small enough that you *may* be able to compute it by hand if you are clever.]

Free PDF. See conwaylife.com/book for pattern files and print version.

Free PDF. See conwaylife.com/book for pattern files and print version.

7. Stable Circuitry

> The greatest use of a life is to spend it on something that will outlast it.
>
> William James

In the previous chapter, we illustrated how we could manipulate gliders (and sometimes other moving objects like lightweight spaceships) via periodic components like oscillators. We now investigate how to similarly perform these tasks via stable objects (i.e., still lifes), which has the advantage of not restricting the period of the glider streams that can make use of the circuitry. We actually already know of some very important stable circuits, such as the Snark (a stable glider reflector) from Figure 3.29(b), but typically they are more difficult to construct due to the fact that stable objects can not provide any sparks for gliders to make use of.

For this reason, stable circuitry typically focuses not just on manipulating gliders, but also on manipulating other (easier to manipulate) objects like Herschels, just as we did in Section 3.6. We will even spend quite a bit of time converting moving objects of one type into another type (e.g., a glider into a Herschel or vice-versa). We start by giving Herschel tracks a more thorough treatment.

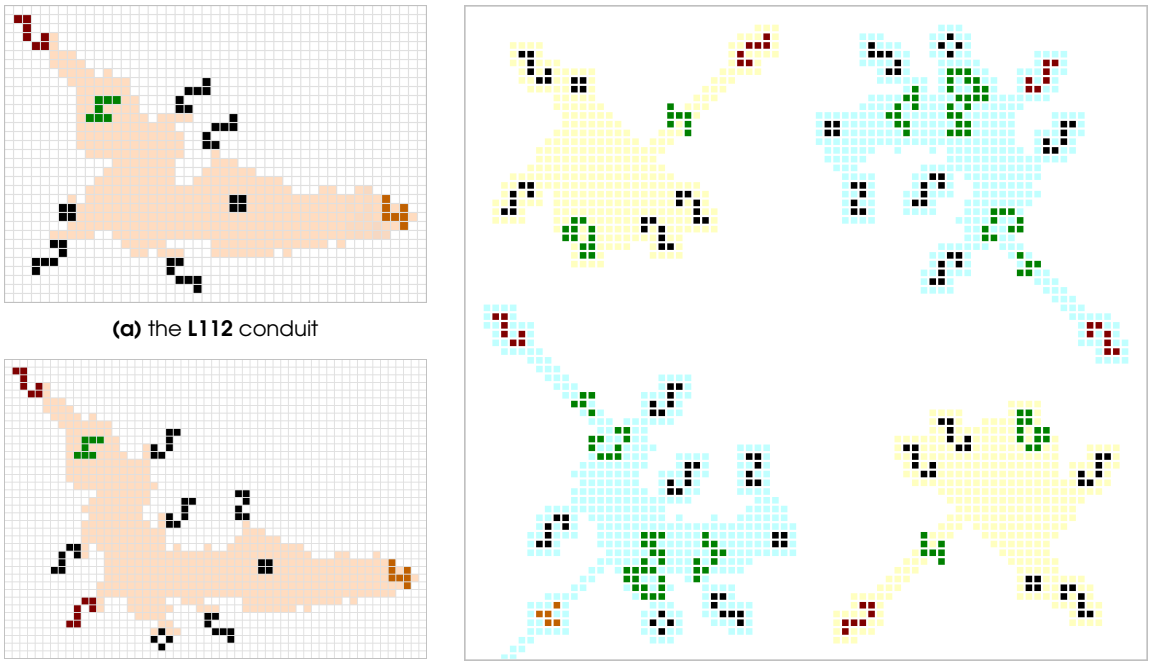
7.1 Herschel Conduits

There are numerous Herschel conduits other than the R64 and Fx77 conduits that we saw back in Figure 3.34. Although those two conduits suffice to create oscillators and guns of any sufficiently large period, certain periods are quite unwieldy. For example, the smallest period 67 oscillator or gun that can be created using just these two conduits requires 4 R64 conduits and 28 Fx77 conduits to make a track of length

$$\underbrace{4 \times 64}_{\text{R64 time}} + \underbrace{28 \times 77}_{\text{Fx77 time}} = 2412 = 36 \times 67 \text{ generations},$$

Free PDF. See conwaylife.com/book for pattern files and print versio

which we place 36 equally spaced Herschels on. However, if we extend our collection of conduits a little bit, then we can construct a Herschel-based period 67 gun that uses just 8 Herschels on an $8 \times 67 = 536$-generation track made up of 4 conduits, as in Figure 7.1.[1]



(a) the **L112** conduit

(b) the **L156** conduit

(c) a p67 glider gun

Figure 7.1: A period 67 glider gun can be constructed from just four Herschel conduits. The track (c) uses two copies of the L112 conduit from (a) (highlighted in yellow) and two copies of the L156 conduit from (b) (highlighted in aqua). Both of these conduits rotate an input Herschel (displayed in green) counter-clockwise by 90 degrees into the output position (displayed in orange), with L112 taking 112 generations to do so and L156 taking 156 generations. Eaters displayed in red are used to delete escaping gliders.

The added flexibility of these additional conduits not only lets us greatly reduce the size of Herschel tracks, but also makes it much easier to position them in certain situations (if we want a Herschel to output a glider at a specific location or with a certain timing, for example). For this reason, it is useful to have a large list of Herschel conduits available to choose from. For ease of reference, conduits are named according to the orientation of the output Herschel relative to that of the input Herschel,[2] together with the number of generations that it takes for the output Herschel to appear. For example, the R64 conduit is named that way because it turns the input Herschel to the **r**ight in **64** generations. More generally, the prefix used when naming conduits is one of:

R: right (clockwise) turn	**F:** forward (no turn)
L: left (counter-clockwise) turn	**B:** backward (180-degree turn)

For example, the L112 and L156 conduits are named for the fact that they rotate a Herschel left (i.e., counter-clockwise) over the course of 112 and 156 generations, respectively. We also insert an "x" between two parts of the conduit's name if it mirrors the Herschel (as in the Fx77 conduit from

[1] We saw one of these extra conduits, L112, way back in Exercise 3.29.

[2] Be somewhat careful here—the naming does not care about the *position* of the output Herschel relative to the input Herschel, only the change in its orientation.

Free PDF. See conwaylife.com/book for pattern files and print version.

Figure 3.34(b)).[3] A conduit's name can thus have one of 8 possible prefixes (i.e., R, Rx, L, Lx, F, Fx, B, or Bx) corresponding to the 8 possible orientations of the output Herschel that they produce, as indicated in Figure 7.2.

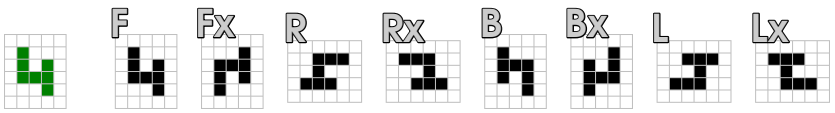


Figure 7.2: Herschels can be oriented in one of 8 ways, which we label here relative to the (arbitrarily chosen) canonical input orientation displayed on the left in green.

With this naming scheme out of the way, we now catalog some of the smallest, quickest, and most useful Herschel conduits that are known in Table 7.1. Keep in mind that this list is nowhere near complete—there are well over 100 known Herschel conduits made up of small still lifes.[4] All of these conduits release at least one glider and can thus be used to construct guns, but of particular note is L156, which releases a glider from its corner in a somewhat different orientation than most of the others (we made use of this glider in the period 67 gun from Figure 7.1). It is also worth finding the R64, Fx77, L112, and L156 conduits that we are already familiar with in this table, to see where they fit in with other Herschel conduits.

7.1.1 Conduit Terminology

Back in Section 2.3 we discussed eaters, such as eater 1, that can delete other objects without suffering any permanent damage. Eater 1s are very common components in Herschel conduits. They do very often act as eaters, simply deleting unwanted output gliders, or deleting blocks or blinkers or other objects generated as the active reaction passes through the conduit. However, their function is not always limited to eating.

A more general term for the still lifes that make up a conduit is **catalyst**. Like an eater, a catalyst is not permanently affected by a passing active reaction. It is temporarily altered (unless it is a rock as described in Section 2.3.1) but eventually recovers to its original form with no permanent damage.[5] That passing active reaction is not necessarily any well-defined object like a glider, block, beehive, or blinker. Very often it is simply some unnamed chaotic mess, and the catalyst just causes it to evolve differently from the way it would have without the catalyst's presence.

The majority of the conduits in Table 7.1 contain a special type of catalyst called a **transparent** catalyst. This is a small common object, usually a block or a beehive, that is temporarily destroyed completely by an active reaction, but reappears a few generations later in exactly the same location. The presence of the transparent catalyst modifies the active reaction in precisely the right way to produce a copy of itself.

Transparent catalysts are rare and difficult to find, but extremely useful when they do appear. They often make it possible for an active reaction to pass "through" their location and move into a new region that the reaction would not otherwise have reached.

[3] To make the "x means mirror" notation unambiguous, we need to specify the direction in which the mirroring is done, so we (arbitrarily) choose to mirror along the single side of the Herschel that is perpendicular to its other two sides (if you prefer, you can simply refer to Figure 7.2 to see how the mirroring is done).

[4] For a reasonably complete collection of known small Herschel conduits, see conwaylife.com/forums/viewtopic.php?f=2&t=2347

[5] The term "catalyst" is borrowed from chemistry, where catalysts affect an ongoing chemical reaction without themselves being used up.

Free PDF. See conwaylife.com/book for pattern files and print version.

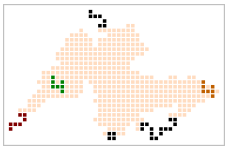
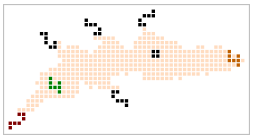
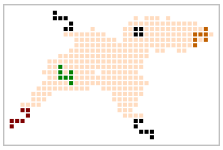
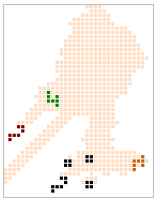
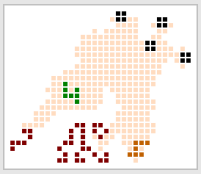
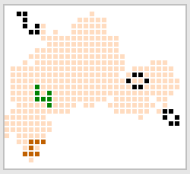
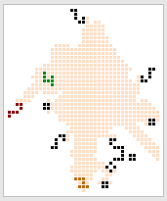
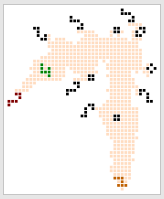
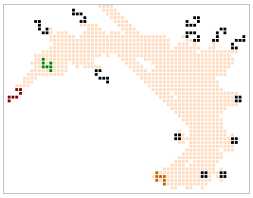
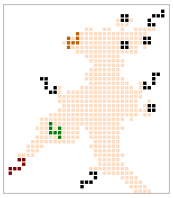
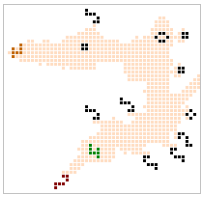
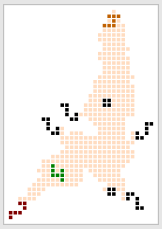
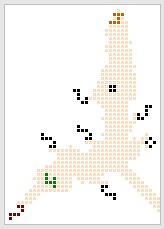
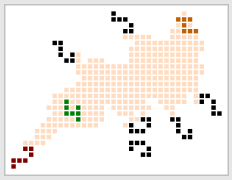
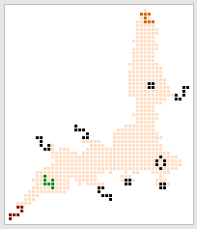
	Rotating conduits		Rotating and reflecting conduits	
F:	 F116 (138)	 F117 (63)	 Fx77 (57)	 Fx119 (231)
R:	 R64 (61)	 R126 (125)	 Rx140 (260)	 Rx164 (65)
B:	 B60 (43)	 B245 (278)	 Bx106 (134)	 Bx202 (65)
L:	 L112 (58)	 L156 (62)	 Lx86 (134)	 Lx163 (60)

Table 7.1: A collection of small and fast Herschel conduits that can produce a Herschel in any orientation. The number in parentheses beside each conduit's name is its repeat time. Input Herschels are displayed in green and output Herschels are displayed in orange. Eaters displayed in red just destroy stray gliders (potentially reducing the conduit's repeat time) but are not required for the conduit to work.

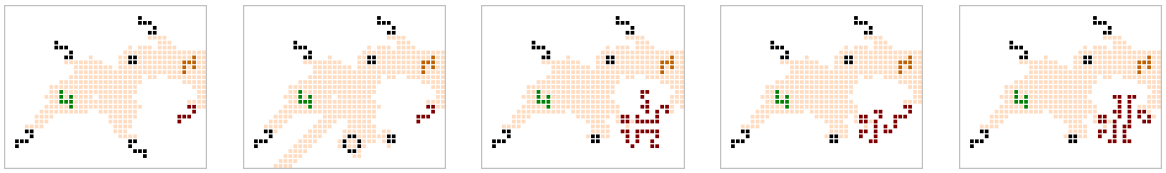
7.1.2 Conduit Variants and Tight Squeezes

Since the name of a conduit depends only on the orientation and timing of its output Herschel relative to the input one, there can be many different Herschel conduits with the same name. This feature is by design, as we typically do not care about what exactly happens in the intermediate steps of the conversion that a conduit implements. For this reason, we say that two conduits are **variants** of each other if they both take the same input and produce the same output in the same spacetime location—no matter what happens along the way.

Conduit variants are sometimes useful for getting around tight spacing issues. For example, when connecting conduits together, the standard version of those conduits often won't fit—catalysts in the two conduits overlap each other and can't be welded—but an alternate variant will fit. To illustrate

Free PDF. See conwaylife.com/book for pattern files and print version.

this phenomenon, consider the variants of the Fx77 conduit displayed in Figure 7.3.

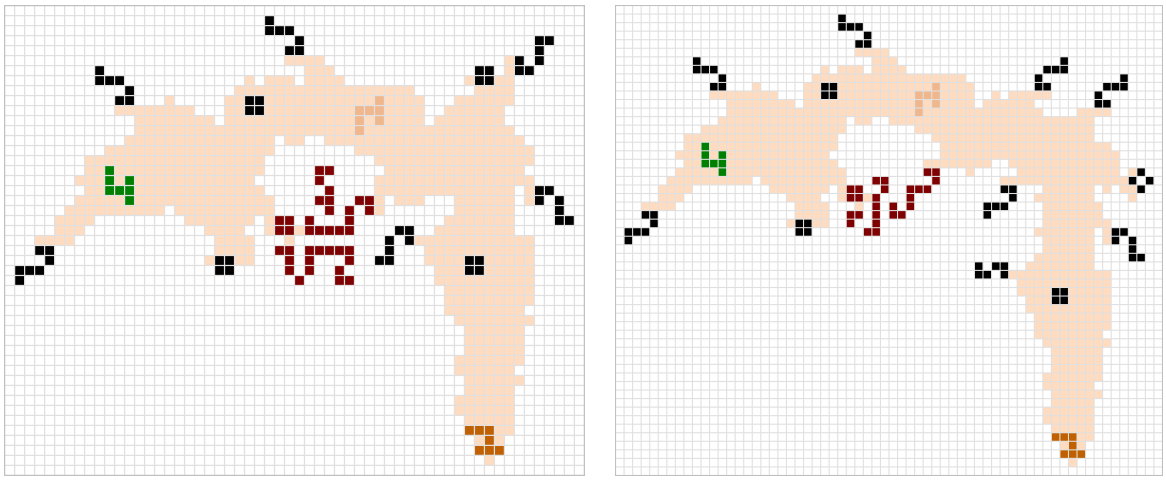


(a) Fx77a (b) Fx77b (c) Fx77c (d) Fx77d (e) Fx77e

Figure 7.3: Five variants of the Fx77 Herschel conduit, along with eaters or welds (displayed in red) that are capable of destroying the first block left behind by the output Herschel. The "a" variant is the one that we have used up until now (e.g., in Section 3.6 and Table 7.1).

To understand why it is useful to have so many variants of Fx77, notice that many Herschel conduits have an eater 1 in one of two standard positions directly above their input Herschel (if that input Herschel is in its canonical input orientation). For example, such an eater 1 makes numerous appearances in Table 7.1, and is present directly above the input Herschel in each of F117, Fx77, R126, Rx164, B245, Bx106, Bx202, L112, L156, Lx86, and Lx163. The purpose of this eater 1 is to erase a block that is left behind early in the Herschel's evolution. The complicated-looking welded eaters in the "c", "d", and "e" variants of Fx77 are just the best known ways to save a row or two while still allowing for that eater to appear in one of those two standard eater positions in the following Herschel conduit.

For example, to save some space at the bottom of the Fx77 conduit when it is followed by an L112 conduit, we could use the "c" variant of Fx77, since its welded eater has the same orientation as the block-destroying eater in L112. However, to save some space at the bottom of the Fx77 conduit when it is followed by an L156 conduit (whose block-destroying eater has the opposite orientation as L112), we would instead have to use the "d" variant of Fx77, since its welded eater also has that opposite orientation (see Figure 7.4). The "e" variant of Fx77 can also be used in the same situations as the "c" variant, but not the "d" variant.



(a) Fx77c (left) with L112 (right) attached to its output. **(b)** Fx77d (left) with L156 (right) attached to its output.

Figure 7.4: We can attach (a) L112 to the output of Fx77c and (b) L156 to the output for Fx77d, but not L112 to Fx77d or L156 to Fx77c.

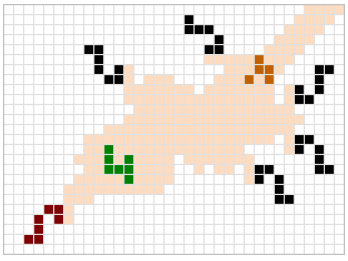
Free PDF. See conwaylife.com/book for pattern files and print version.

7.2 From Herschels to Gliders

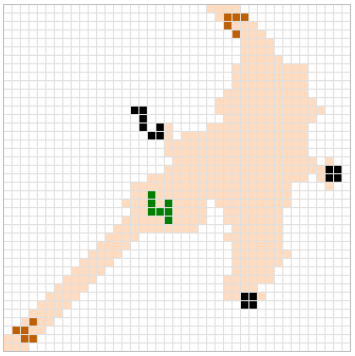
To be able to make better use of Herschel tracks and glider reflectors, it will be useful for us to be able to convert Herschels into gliders and vice-versa. Converting a Herschel into a glider is straightforward since it emits a natural glider after 21 generations anyway, so we just need to destroy the rest of the Herschel after that time. For this reason, dozens of Herschel-to-glider converters are known, though we are primarily interested in ones that produce gliders traveling in directions or on lanes *other* than that of the first natural glider. We of course could use Snarks to reflect the natural glider into any direction of our choosing, but this is somewhat large and cumbersome compared to the conduits that produce these other-directional gliders directly.

Of the numerous known conduits of this type, many even produce *multiple* gliders travelling in different directions, and thus can be used to duplicate a signal (once combined with a glider-to-Herschel conduit that we will see shortly). In particular, the collection of simple conduits displayed in Figure 7.5 can be used to convert a Herschel into one, two, or three gliders travelling in any of the four desired output directions. There are also some simple conduits that turn a Herschel into four gliders, all travelling in different directions—see Exercise 7.29.

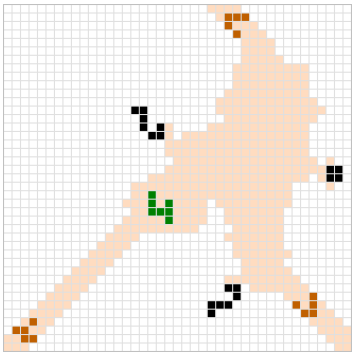
Just like glider guns become easier to use when the glider that they produce is near their edge (we encountered one of these **edge shooters** in Figure 6.29(a)), so too do Herschel-to-glider converters. This makes the conduits from Figures 7.5(b) and 7.5(c) especially useful—the glider that they release to the northwest occupies a diagonal lane close to its edge, so they can be used to produce tight glider spacings (assuming we are able to generate the input Herschels, which we will see how to do shortly).



(a) A conduit that converts a Herschel to a glider traveling antiparallel to the first natural glider.



(b) A conduit that converts a Herschel into two perpendicular gliders.



(c) A conduit that converts a Herschel into three gliders traveling in different directions.

Figure 7.5: A small collection of converters that transform a Herschel (displayed in green) into one or more gliders (displayed in orange). Any unwanted gliders from the last two converters can simply be destroyed by an eater (as we did in (a) via the eater displayed in red).

Despite not being an edge shooter, the conduit from Figure 7.5(a) is useful for a similar reason. In that conduit, the output glider lane and one other nearby lane are **transparent**—gliders on that lane can pass safely through without colliding with the conduit. This conduit can thus be used to insert gliders into a stream just like an edge shooter, as illustrated in Figure 7.6.[6]

Before we present any more Herschel-to-glider conduits, it will be useful for us to have a way to name them, just like we do with Herschel conduits. Just like the name of a Herschel conduit contains information about the orientation and timing of the output Herschel compared to the input Herschel, we want the name of a Herschel-to-glider conduit to tell us the direction, position, and timing of the output glider relative to the input Herschel.

[6] A slight variant of this Herschel-to-glider converter that is better in some situations is presented in Exercise 7.4.

Free PDF. See conwaylife.com/book for pattern files and print version.

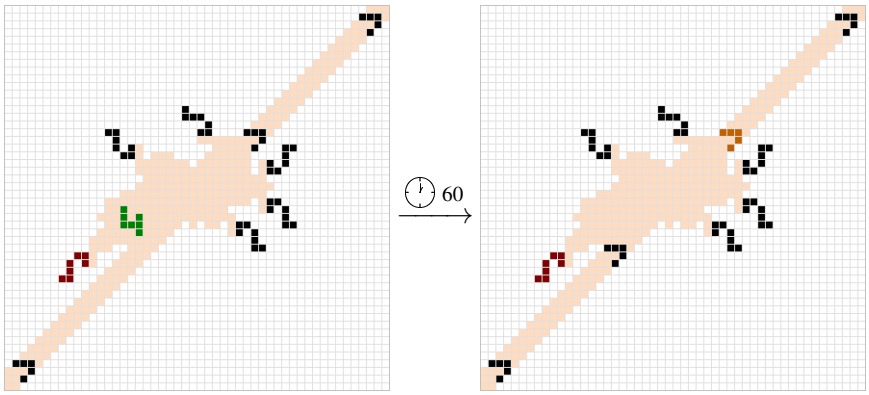
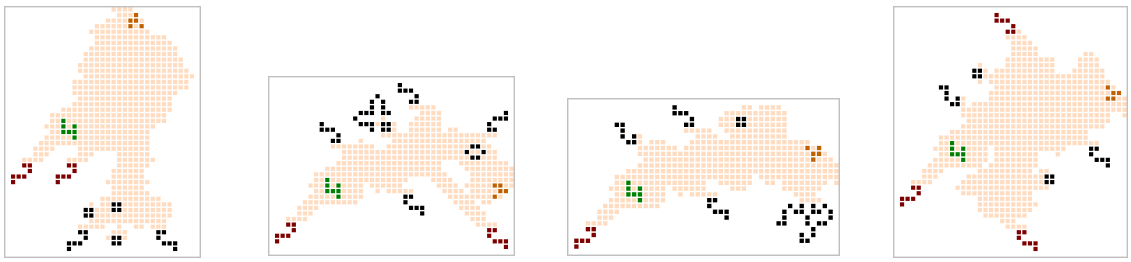


Figure 7.6: The conduit from Figure 7.5(a) has a transparent lane that can be used to insert a glider into a stream (which has period 60 here).

With this in mind, these conduits are given names of the form `<direction><lane>T<timing>`, where:

- `<direction>` is one of `NW`, `NE`, `SW`, or `SE`, indicating the direction (northwest, northeast, southwest, or southeast) of the output glider relative to the canonical input phase of the Herschel that we have been using since Figure 7.2, and
- `<lane>` is the lane number of the output glider, and `<timing>` is similarly its timing, also relative to the input Herschel.[7]

For example, the conduit in Figure 7.5(a) is called **NE5T-4** (to be clear, the dash in this name is a minus sign, not a separator—the timing of the output glider is -4). The conduits in Figures 7.5(b) and 7.5(c) are both called **NW31T120**, though the version that produces two gliders would need a longer name to identify both outputs: **NW31T120_SE7T14**.[8] The first and second natural gliders (i.e., the southwest and northwest gliders in Figure 7.5(c)) are often not included in the name of a Herschel-to-glider NW31T120_SE7T14 conduit unless they are its only output. Exactly how the timing and lane of a glider is computed relative to a Herschel is not particularly important for our purposes—it's just nice to know where these names come from.



(a) Fx119 inserter (156) **(b)** NE30T3 (67) **(c)** SE39T32 (126) **(d)** SE39T59 (155)

Figure 7.7: Some edge-shooting Herschel-to-glider converters. The number in parentheses is the repeat time of the conduit. Note that (a) is the Fx119 Herschel conduit from Table 7.1 with one more eater, and all four of these edge shooters can fire additional gliders by erasing the eaters in red. Also, (b) is not strictly speaking an edge shooter, but the only part of it southeast of the escaping glider is a single eater that can easily be placed as far out of the way as we like, so it can still be used to inject gliders close to other passing gliders.

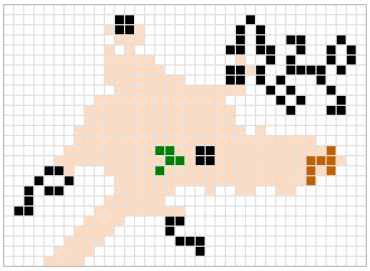
[7] Refer back to Section 4.1.2 if you need a refresher on glider lanes and timing.
[8] This conduit is common enough that its name is often abbreviated as **NW31**.

Free PDF. See conwaylife.com/book for pattern files and print version.

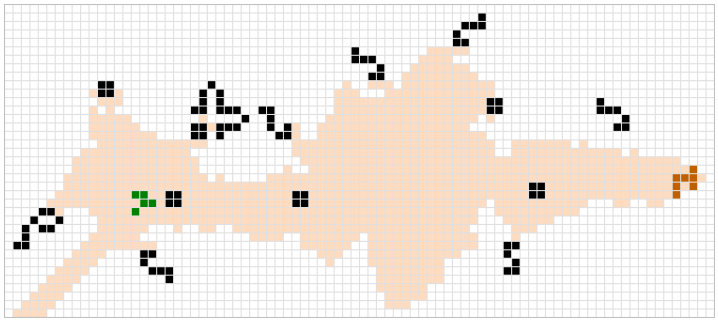
With all of this taken care of, some other Herschel-to-glider edge shooters are displayed in Figure 7.7.[9]

7.3 From Gliders to Herschels

Despite the simplicity of converting a Herschel into a glider, the reverse conversion of a glider into a Herschel is much trickier to implement. Despite considerable computer searches that have been carried out, only one small and fast stable converter of this type has ever been found. It is called the **syringe**,[10] and two slightly different versions of it are displayed in Figure 7.8. A third, slightly more compact, version of the syringe is presented in Exercise 7.8.



(a) The "standard" version of the syringe, which has repeat time 78 and takes 84 generations to convert a glider into a Herschel.



(b) A larger version of the syringe that has a repeat time of 115 generations and takes 250 generations to convert a glider into a Herschel.

Figure 7.8: The **syringe** is a conduit that converts a glider into a Herschel. The smaller version (a) is very fast, but makes use of an unusual large welded still life. The larger version (b) has the disadvantage of being slower and bulkier, but the advantage of only using "standard" components, which makes it easier to synthesize.

While the canonical version of the syringe that is displayed in Figure 7.8(a) is much more compact and quick, its variant from Figure 7.8(b) has the advantage of consisting entirely of easy-to-synthesize pieces like blocks and eater 1s. Its most complicated-to-synthesize component is the eater 2, which is much simpler to construct via advanced synthesis techniques like slow-salvo (Section 5.7) or single-channel (Section 11.2) synthesis than the complicated, unnamed still life from the canonical syringe. We call any conduit of this type (i.e., entirely made up of small, easy-to-synthesize still lifes and p2 oscillators) **Spartan**, and they will be the main type of conduit used throughout most of the mega-constructions that we will see in Chapters 9–12.[11]

7.4 From Gliders to Gliders

To give an example of just how useful the syringe is, we note that it can easily be combined with the Herschel-to-glider converter from Figure 7.5(a) to create the small and fast color-changing stable reflector displayed in Figure 7.9 (recall that the Snark is color-preserving). Explicitly, this reflector uses a syringe to convert the input glider into a Herschel, which is then converted back to a glider

[9] A more-or-less complete collection of Herschel-to-glider converters is available for download at `conwaylife.com/forums/viewtopic.php?f=2&t=1682`

[10] The syringe's name comes from the idea that it "injects" a glider into a Herschel track. It was found in March 2015 by Tanner Jacobi, using the same "Bellman" program that was used to find the Snark.

[11] The exact definition of "Spartan" changes from time to time, as glider synthesis technology improves and so it becomes "easy" to synthesize a wider class of objects. The original definition, from 2004, included only still lifes with 7 or fewer live cells. Nowadays, the definition typically includes any object whose slow salvo synthesis can be automatically compiled by the computer program *slsparse* (see `conwaylife.com/wiki/Slsparse`), which is a much wider class of objects.

Free PDF. See conwaylife.com/book for pattern files and print version.

(with opposite color and rotated 90 degrees from the input glider). However, since its repeat time is 78 generations,[12] the smaller and faster bouncer reflectors from Section 6.4 are still more useful in many situations.

A slightly smaller and faster stable color-changing reflector can be made by using a conduit called the **Bandersnatch**,[13] which changes the color of a glider, but does not change its direction. Conduits of this type are sometimes called **zero-degree reflectors**, and they can be turned into proper 90-degree reflectors simply by attaching a Snark to their input or output, as in Figure 7.10. This new color-changing reflector is not only a bit smaller than ones based on the syringe, but it also has a slightly lower repeat time of just 70 generations. Furthermore, it is a bit easier to construct via glider synthesis, since the Bandersnatch is Spartan.

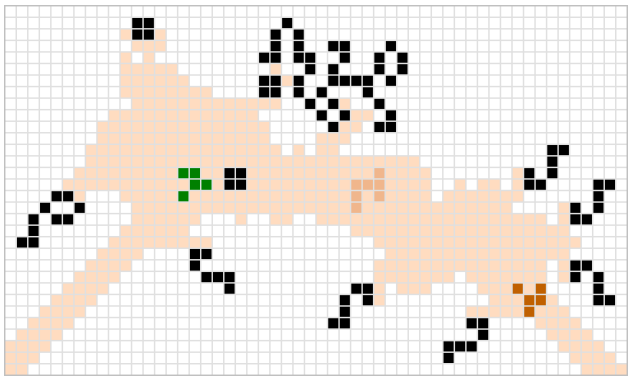


Figure 7.9: A stable color-changing reflector with a repeat time of 78 generations. A syringe converts a glider to a Herschel, which is then converted back into a glider by the conduit from Figure 7.5(a).

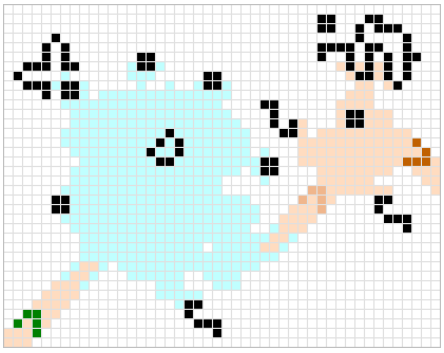


Figure 7.10: A stable color-changing reflector with a repeat time of 70 generations. A Bandersnatch (highlighted on the left in aqua) changes the color of a glider, which is then reflected by a Snark.

We now go one step farther and build conduits that don't just reflect a glider, but also duplicate it and produce *multiple* output gliders. It should not be surprising that such a conversion is possible now, as we can use the syringe to convert a glider into a Herschel, and a Herschel can be made to emit as many gliders as we like by pushing it along a track without destroying its first natural gliders. Perhaps the simplest way to make this construction explicit is to use a syringe followed by the Herschel-to-3 gliders converter from Figure 7.5(c). If we want to increase the number of output gliders, we can just insert some Herschel tracks between these two end pieces. For example, every copy of the Fx77 conduit that we insert increases the number of output gliders by 1 (see Figure 7.11(b)).

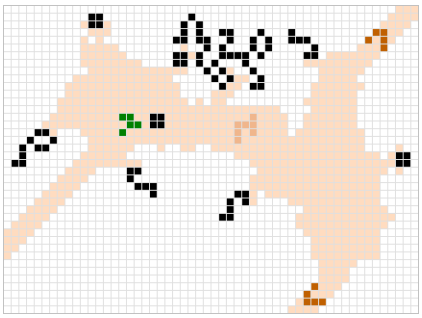
Now that we know how to construct conduits that convert one glider into any number of gliders, we can create conduits that take a single glider as input and then implement arbitrary glider syntheses. All we have to do is first convert the input glider into the number of gliders required for the synthesis, and then use Snarks (or other stable reflectors) to reposition and rephase those gliders so as to actually perform the synthesis.

However, we run into the exact same problem that we ran into when using one-time turners to perform slow salvo synthesis in Section 5.7: even though it is straightforward to reflect gliders into the correct *positions*, having them all show up at the right *time* is rather tricky. Fortunately, we can solve this problem in the same way that we did back then—we create a family of conduits that can either preserve or change the color of a glider, and can also give us any mod-8 timing of the output glider of

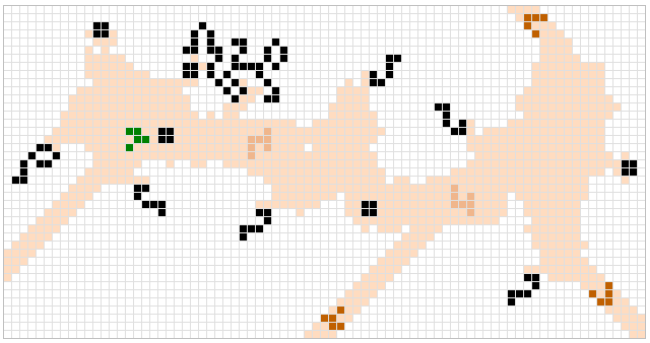
[12] The syringe still works when its input gliders have a gap of 74 or 75 generations, but *not* 76 or 77 generations. We call this phenomenon **overclocking**.

[13] Found by Martin Grant and ConwayLife.com forums user "Entity Valkyrie" in June 2020. Like the Snark, it was named after a fictional creature in Lewis Carroll's poem *The Hunting of the Snark*.

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) A glider duplicator that uses a syringe and the Herschel-to-3-gliders converter from Figure 7.5(c).



(b) A glider tripler that uses a syringe, the Fx77 conduit, and the Herschel-to-3-gliders converter.

Figure 7.11: Some conduits that use the syringe to duplicate a glider. By extending the length of the track by inserting more copies of the Fx77 conduit, we can increase the number of emitted gliders.

our choosing. One such collection of conduits can be constructed by using a syringe to turn the input glider into a Herschel, followed by a Herschel-to-glider conduit to transform it back, possibly with one or more Herschel conduits placed in between to change the relative timing of the input and output gliders—see Table 7.2.[14]

7.5 Synthesizing Objects via Conduits

Now that we can use stable conduits to reposition gliders however we like (via the Snark and the color-changing reflectors of Figure 7.9 and 7.10), *and* adjust the timing of gliders however we like (via the collection of rephasers in Table 7.2), we can construct stable circuits that take in a single input glider and generate any object that we know how to construct via glider synthesis. To illustrate this procedure, we now construct a stable circuit that transforms a glider into a lightweight spaceship via the first 3-glider synthesis displayed in Table 5.2:

1) First, it will be easier to adjust the timing of gliders later if they are all traveling in the same direction, so we add Snarks at what will become the end of the circuit to reflect the three incoming gliders into the positions required for synthesis. This arrangement is located at the southeast corner of Figure 7.12(a), with all three input gliders coming from the northwest.
2) Next, we start working on what will become the start of the circuit—we use the glider tripler from Figure 7.11(b) to turn the input glider into three gliders. This glider tripler is located at the north end of Figure 7.12(a).
3) We then add reflectors to the output of the glider tripler so as to put the three gliders into the correct lanes to meet up with the component that we constructed in step (1) above. Note that we use the color-changing reflector from Figure 7.9 for the northeast glider since its color after exiting the tripler does not match that of the lane that we need to put it on. At this point, we have constructed the pattern displayed in Figure 7.12(a)—we have put the three gliders into the correct positions, but we still need to fix their timing.
4) To fix the mod-8 timing of the gliders, we choose one of the three gliders to have the "right" timing, and we synchronize the other gliders with it. It typically works best (e.g., results in less "fiddling" and a smaller circuit) if we choose the glider that gets to its intended position *last* to be this reference glider, and in this case that is the southwestern glider. From left to right,

[14] This collection was compiled by Simon Ekström in August 2015. He also put together a few more—see conwaylife.com/forums/viewtopic.php?p=21708#p21708.

Free PDF. See conwaylife.com/book for pattern files and print version.

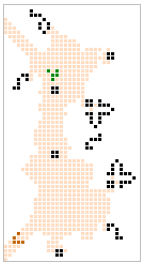
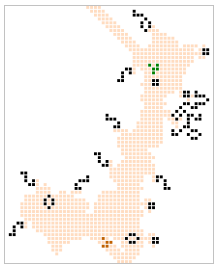
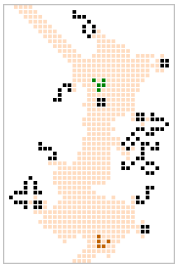
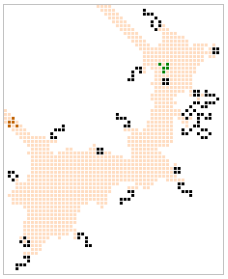
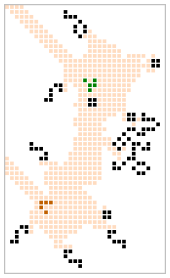
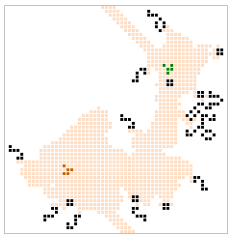
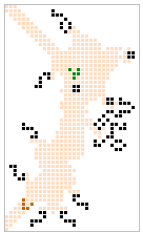
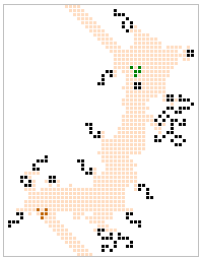
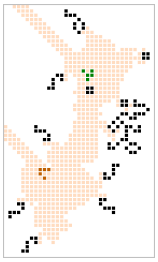
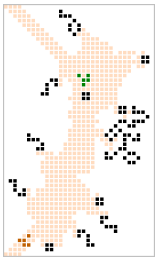
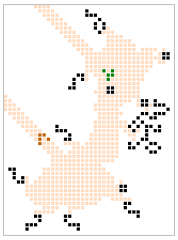
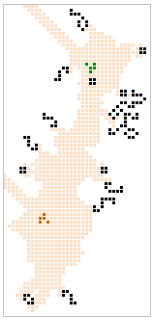
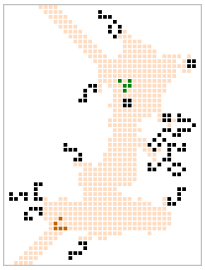
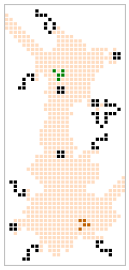
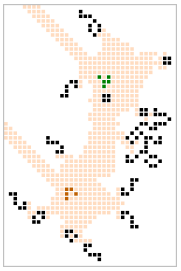
	Delay			
	0	1	2	3
Color-Preserving (CP)	Snark			
Color-Changing (CC)				
	4	5	6	7
Color-Preserving (CP)				
Color-Changing (CC)				

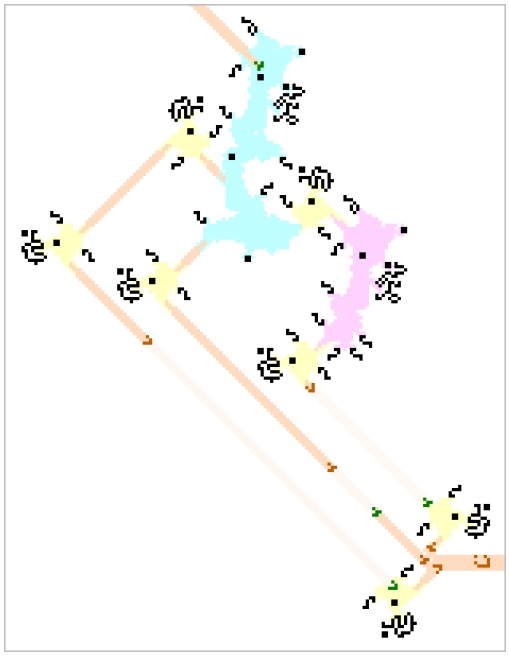
Table 7.2: A collection of stable glider rephasers that can be used to put gliders into any timing and color relative to each other that is desired (compare with Table 5.5, which did the same thing via one-time-turners instead of stable circuits). In all cases, the input glider is highlighted in green and comes in from the top-left, while the location of the output glider is highlighted in orange.

Figure 7.9:

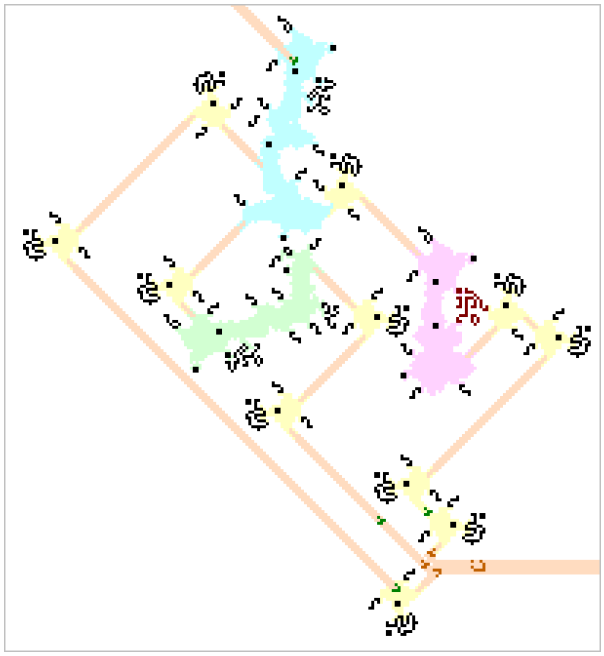
we then want to delay the gliders by 0, 252, and 163 generations, respectively, which (mod 8) equal 0, 4, and 3, respectively. We thus insert a delay-4 CP reflector (from Table 7.2) into the path of the middle glider, and we change the delay-3 CC reflector in the northeast path into the delay-6 CC reflector.

Free PDF. See conwaylife.com/book for pattern files and print version.

5) Now that all gliders have the correct mod-8 timing, we just need to adjust their timings by multiples of 8 generations. This can be done simply by moving any 180-degree reflection that a glider goes through closer or farther away, as in Figure 7.13 (if there is no 180-degree reflection in a glider's path, we can simply insert two of them). After making these final timing adjustments, we have the completed glider-to-LWSS circuit displayed in Figure 7.12(b).[15]



(a) An incomplete glider-to-LWSS circuit that puts three gliders into the correct positions to synthesize an LWSS, but with incorrect timings.



(b) A completed glider-to-LWSS circuit that uses rephasers from Table 7.2 and trombone slides to correct the glider timings.

Figure 7.12: A stable glider-to-LWSS conduit that works by using a glider tripler (highlighted in aqua) to turn one glider into three, which are then repositioned and rephased so as to synthesize a lightweight spaceship. Snarks (highlighted in yellow) are color-preserving and are treated as "free" reflectors that do not alter the mod-8 timing of gliders. One color-changing reflector (highlighted in magenta) is used on the northeast glider's path (the one shown in (b) delays the glider's mod-8 timing by 3 more generations than the one shown in (a)), and in (b) we use a reflector that delays the middle glider's mod-8 timing by 4 generations (highlighted in green).

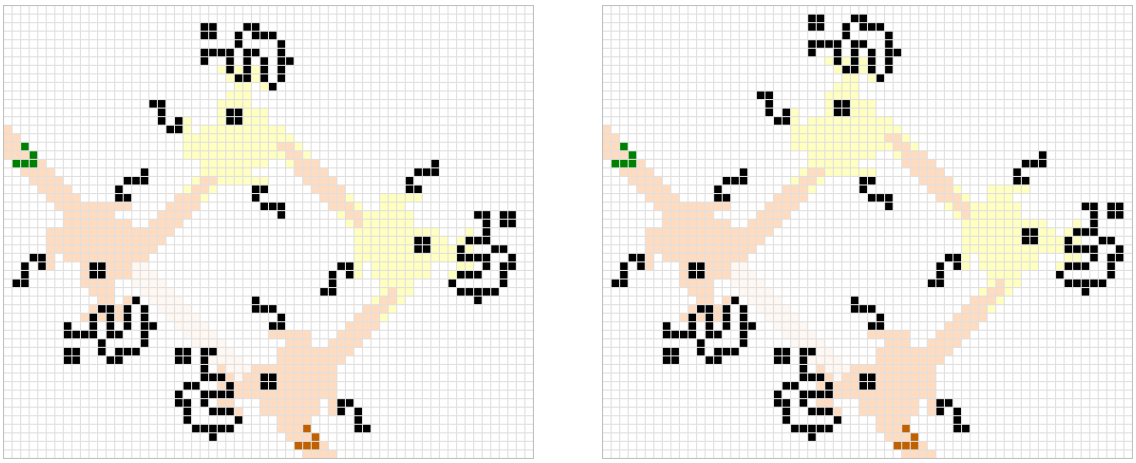
These techniques can be extended straightforwardly to let us construct circuits that convert a glider into any other object that we know how to synthesize, though the details become quite a bit more fiddly as the number of gliders used in the synthesis increases. To illustrate how far we can push these methods, we now construct a stable circuit that converts a glider into the 2-engine Cordership that was introduced in Exercise 4.20.

As our starting point, we need a glider synthesis of the 2-engine Cordership to make use of, and for simplicity we use the 2-direction synthesis from Exercise 5.14(b). To space these gliders out a bit more and make them even easier to synchronize, we use Herschel edge shooters to put the gliders in their correct places and reflect the input gliders so as to all come from the same direction (much like we did in the southeast corner of Figure 7.12(a) when constructing the glider-to-LWSS circuit). This bottom portion of our circuit is displayed in Figure 7.14.

Next, we need to turn one glider into 10 gliders and synchronize them with the positions indicated in Figure 7.14. While it is possible to do this "directly", it is easier to instead just synchronize just 2

[15] Somewhat smaller stable glider-to-LWSS converters are known—see Exercise 8.9.

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) A path that takes 276 generations.

(b) A path that takes 284 generations.

Figure 7.13: A **trombone slide** is a 180-degree reflector that can be freely moved closer or farther away along a glider's path, delaying the glider by 8 generations for each cell that it is moved away. The only difference between (a) and (b) is that we moved the north and east Snarks (highlighted in yellow) northeast by 1 cell in (b), thus delaying the glider by 8 generations. Note that placing the four Snarks along a glider's path as in (a) delays it by 144 generations—this delay can be made up for (if necessary) by similarly delaying all other gliders that we are working with.

or 3 gliders at a time. For example, if we build a mechanism that splits one glider into the 3 leftmost synchronized gliders, then we will just need to synchronize 8 gliders instead of 10 (the 7 leftover gliders that we have not yet dealt with plus the 1 extra glider needed to produce the 3 leftmost gliders). One such mechanism, as well as mechanisms that synchronize the next 3 gliders, the next 2 gliders, and the final 2 gliders, are presented in Figure 7.15. Once we place all 4 of these mechanisms, we just have to synchronize 4 gliders instead of 10—one for each of the mechanisms.

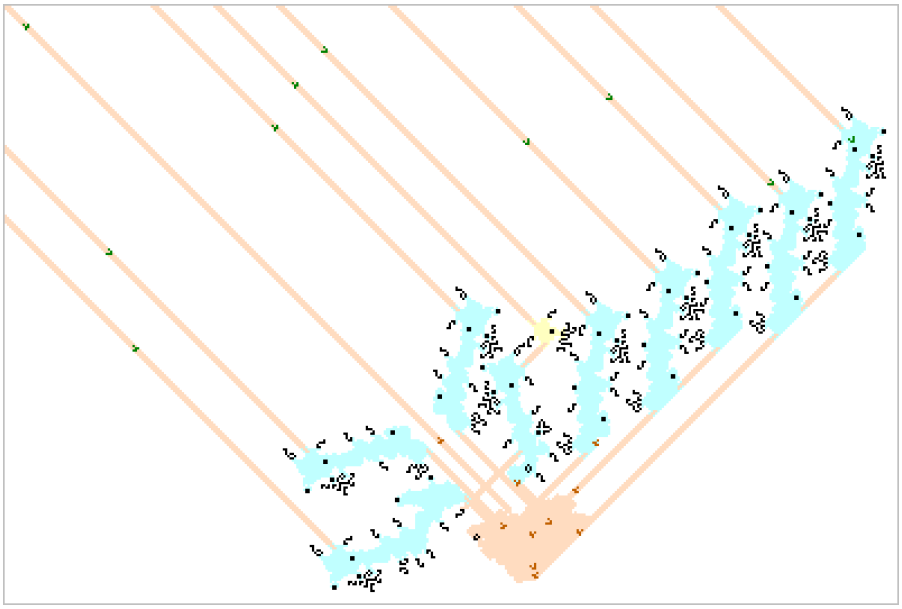


Figure 7.14: An arrangement of Herschel edge shooters (highlighted in aqua) that places 10 gliders coming from the northwest (displayed in green) into the correct positions (displayed in orange) 738 generations later to synthesize to synthesize a 2-engine Cordership.

Free PDF. See conwaylife.com/book for pattern files and print version.

We can then just iterate this procedure, building up the circuit in layers of gliders that synchronize 2 gliders at a time, thus halving the number of gliders that need to be synchronized at each layer. The first layer reduced the number of gliders that we need to synchronize from 10 to 4, the next layer reduces it from 4 to 2, and the final layer reduces it from 2 to 1 (and thus completes the circuit). A completed circuit that turns a glider into a 2-engine Cordership in this way is displayed in Figure 7.16.

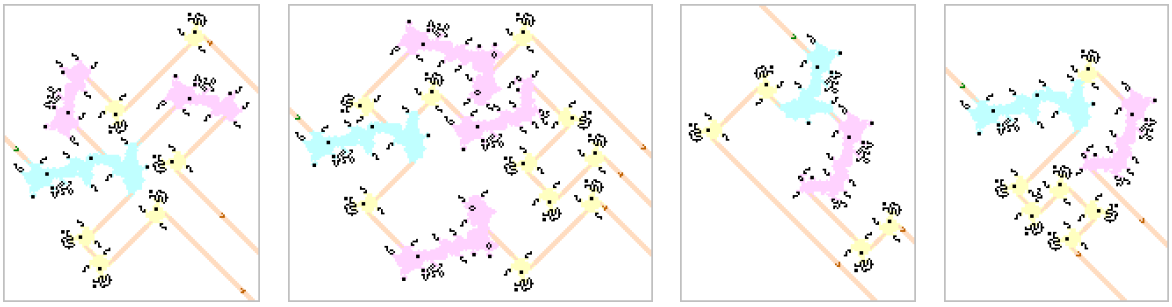


Figure 7.15: Four circuits that, from left to right, synchronize the 3 leftmost input gliders in Figure 7.14, the next three gliders, the next two gliders, and the final two gliders, respectively. All four of these circuits can be constructed using the same methods that we used to build the glider-to-LWSS circuit in Figure 7.12. Glider duplicators are highlighted in aqua, Snarks are highlighted in yellow, and reflectors that change a glider's color and/or mod-8 timing are highlighted in magenta.

It is worth pointing out that there are other stable circuitry toolkits besides the one from Table 7.2 for adjusting the relative timings and positions of gliders. For example, instead of always using the duplicators from Figure 7.11 and then adjusting the timing of gliders via numerous different reflectors (as we have done so far), we could instead make use of numerous different glider duplicators to *directly* put two gliders into any desired relative positioning (i.e., same or different color) and mod-8 timing of our choosing, and then only use Snarks for all reflections. However, introducing a second toolkit to do something that we already know how to do is perhaps overkill, so we do not present it here.[16]

7.6 Period Multipliers and Small High-Period Guns

Herschel tracks can be used to construct guns of any period at least 62, via the techniques introduced in Section 3.6 and revisited in Section 7.1. In fact, Herschel tracks give rise to guns somewhat more naturally than they give rise to oscillators—we just remove one or more of the eaters in a Herschel track oscillator to turn it into a gun.[17] However, now that we are familiar with the syringe, there is a much smaller and simpler way to create glider guns of any period at least 78.[18]

The pattern displayed in Figure 7.17 is made up of two identical halves that take in a glider, convert it to a Herschel (via a syringe) and then convert it back into another glider that is fed into the other half. Eight input gliders serve to start the gun, and as displayed it will have period 80. To increase the period of this gun by n, simply (a) move the top half of the gun (i.e., the top 33 rows of the pattern) northeast by n cells, and (b) adjust the 8 input gliders so as to each be separated by $80+n$ generations.

While these guns are reasonably simple and straightforward to construct, they suffer the drawback that they increase in size quite quickly with their period. In particular, a period p gun of this type

[16] The interested reader can find this toolkit at conwaylife.com/forums/viewtopic.php?f=2&t=2784.

[17] Even though we could construct Herschel track oscillators of period 61, a period 61 Herschel track *gun* is not so simple since the escaping gliders collide with subsequent Herschels if we remove an eater. Actually, this same problem occurs for periods 62–68, but can be overcome with the help of L156 (see Exercise 7.37).

[18] Thanks to overclocking, this same method works for periods 74 and 75 too—see Exercise 7.38.

Free PDF. See conwaylife.com/book for pattern files and print version.

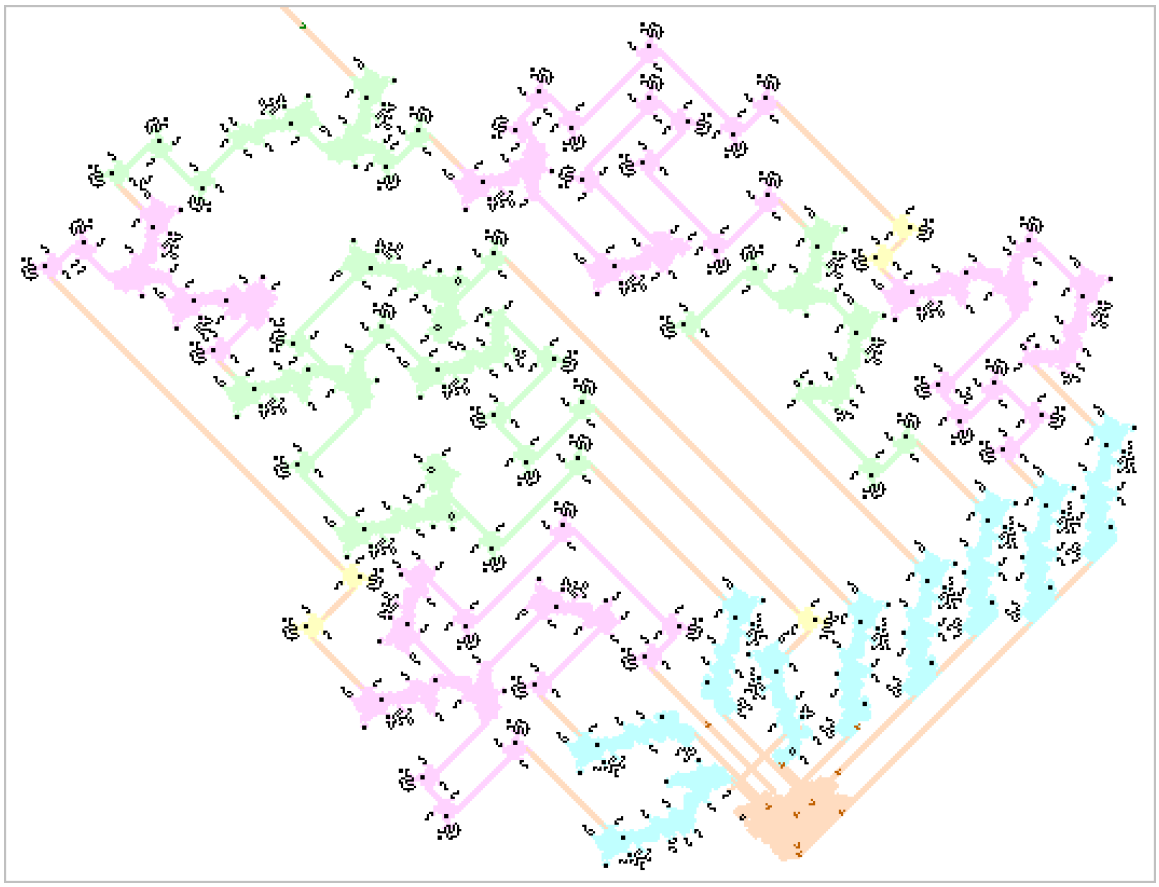


Figure 7.16: A stable circuit that converts a single glider (displayed in green at the northwest) into a 2-engine Cordership (which is synthesized by the 10 gliders displayed in orange at the southeast). Those 10 gliders are synchronized, 2 or 3 gliders at a time, by the various mechanisms highlighted in green and magenta (the four southernmost of which were displayed in Figure 7.15) and fed into the arrangement of Herschel edge shooters (highlighted in aqua) that we originally saw in Figure 7.14.

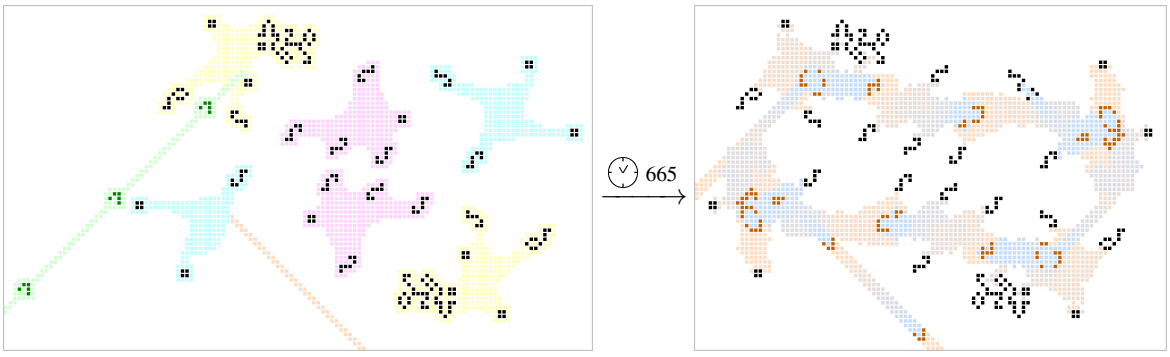
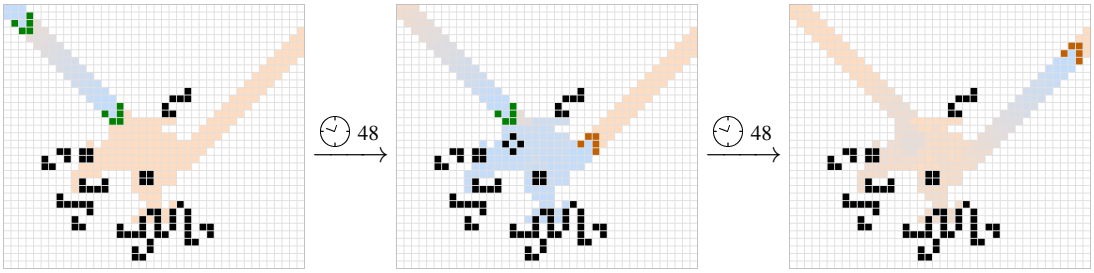


Figure 7.17: A period 80 adjustable glider gun. Eight gliders come in from the southwest (highlighted in green) to start the gun. Those gliders are fed into a syringe (highlighted in yellow) and then proceed clockwise through an F117 conduit (highlighted in magenta) and the herschel-to-glider converter from Figure 7.5(b) (highlighted in aqua). The glider then enters the bottom half of the gun, which is identical except that it is missing one eater and thus emits a stream of gliders to the southeast (highlighted in orange). Constructed by Matthias Merzenich in September 2015.

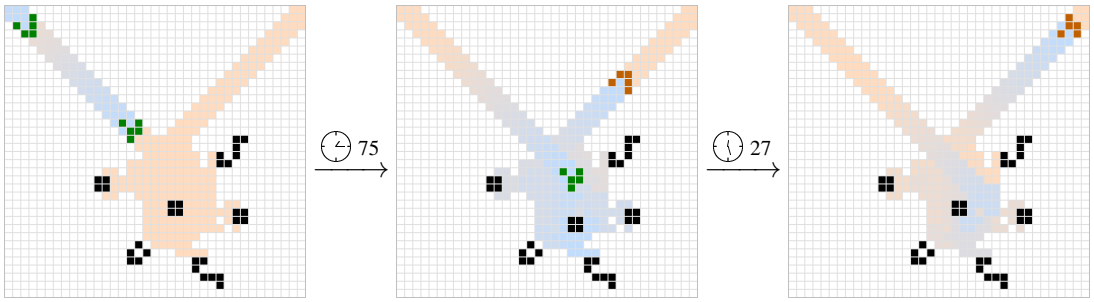
Free PDF. See conwaylife.com/book for pattern files and print version.

has bounding box of size $(p+18)\times(p-14)$, and it seems natural to ask how much smaller we can make them. We can obtain a rough lower bound on the size of a period p gun by noting that an $m\times m$ bounding box contains m^2 cells, each of which can be in one of two states, so there are 2^{m^2} distinct patterns that fit within such a box. It follows that no gun or oscillator in an $m\times m$ box can have period larger than $p=2^{m^2}$, since after that many generations, it must return to a phase that was already seen earlier. By flipping this around and solving for m, we see that there cannot exist a period p gun inside a box of size less than $\sqrt{\log_2(p)}\times\sqrt{\log_2(p)}$.

It turns out that this lower bound is "essentially" tight—there exists a constant C such that it is possible to construct period p guns inside a box of size $\left(C\sqrt{\log_2(p)}\right)\times\left(C\sqrt{\log_2(p)}\right)$ for all large p. In other words, we can construct period p glider guns with bounding box of length and width that are $\Theta\left(\sqrt{\log(p)}\right)$.[19] The key objects used in the construction of such guns are **period multipliers**—conduits that only produce an output signal for every two or more (identical) input signals that are received, and thus multiply the period of the input stream. The two simplest such conduits are the **semi-Snarks** displayed in Figure 7.18, which reflect every second input glider and thus can be used to double the period of a glider stream.[20]



(a) The **color-preserving (CP) semi-Snark**, which has a repeat time of 48 generations. It is the same as a Snark (compare it with Figure 3.29(b)), but with one of its eater 1s replaced by a custom catalyst that, when hit by a glider, creates a tub that destroys the next glider. Found by Tanner Jacobi in October 2017.



(b) The **color-changing (CC) semi-Snark**, which has a repeat time of 51 generations. The first glider is reflected and moves the central block, and the second glider simply moves the block back via the $(2,1)$-block pull from Figure 5.21(a). Found by Sergey Petrov in July 2013.

Figure 7.18: Semi-Snarks are stable conduits that produce one glider for every two input gliders.

For example, placing one of these semi-Snarks along the output lane of the 536-generation Herschel track from Figure 7.1(c) produces a period $2\times536=1\,072$ gun. Placing a second semi-Snark in the path of the output gliders then doubles its period again, resulting in the period $4\times536=2\,144$ gun in Figure 7.19(a). We can of course place many more semi-Snarks along the output path of the gun, resulting in very compact guns with exponentially large period. For example, Figure 7.19(b) shows a gun that consists of a p256 gun with 92 semi-Snarks arranged in a spiral pattern along its

[19] See Appendix A.3 if you need a refresher on big-Θ notation.

[20] Two other conduits that perform the same period-doubling task are presented in Exercises 7.20 and 7.21. They have a similar size and repeat time, but are sometimes useful for adjusting glider timing.

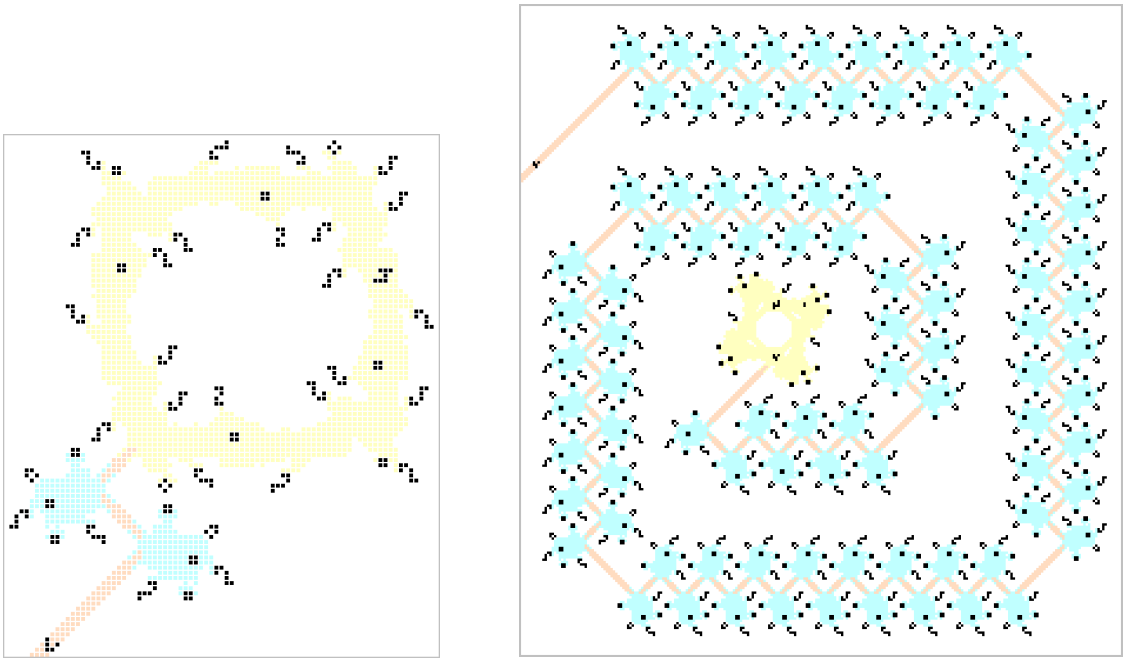
Free PDF. See conwaylife.com/book for pattern files and print version.

output path. Each semi-Snark doubles the gun's period, resulting in a ridiculously large overall period of

$$256 \times 2^{92} = 2^8 \times 2^{92} = 2^{100} = 1\,267\,650\,600\,228\,229\,401\,496\,703\,205\,376,$$

despite fitting inside a bounding box of size just 240×260.

By simply extending this spiral pattern even farther, we can construct guns of period $p = 2^n$ inside a bounding box of size roughly $\left(30\sqrt{\log_2(p)}\right) \times \left(30\sqrt{\log_2(p)}\right) = (30\sqrt{n}) \times (30\sqrt{n})$.[21] We still have some work to do though if we want to be able to construct compact high-period guns like these ones for *any* large period, rather than just periods that are divisible by a large power of 2. For example, this method cannot help us construct a small gun with period equal to a prime number like 3 413 277 319.



(a) a period $4 \times 536 = 2\,144$ gun **(b)** a period 2^{100} gun

Figure 7.19: Two guns that use semi-Snarks to increase their periods. The gun in (a) uses two semi-Snarks (highlighted in aqua) to multiply the period of a p536 gun (highlighted in yellow) by 4. The gun in (b) uses 92 semi-Snarks to multiply the period of the p256 central gun (which consists of four copies of the R64 conduit and is called the **machine gun**) by 2^{92}, for a total period of $256 \times 2^{92} = 2^8 \times 2^{92} = 2^{100}$.

To get one step closer to this goal, we now show how we can use semi-Snarks to multiply the period of a glider stream by *any* positive integer, not just powers of 2.[22] The key observation that lets us do this is that if we line up an arrangement of semi-Snarks, then firing a glider at those semi-Snarks counts down in binary (if we interpret a semi-Snark that blocks a glider as a "1" and a semi-Snark that lets a glider pass as a "0"). In particular, this means that if we place semi-Snarks in an on-off pattern corresponding to a number's binary representation, then the circuit encodes how many gliders are destroyed before the first glider is able to pass through it, as illustrated in Figure 7.20.

However, this method of blocking a specific number of gliders only works the first time, since the first glider that makes its way through the entire circuit resets all of the semi-Snarks to "1", rather than to their original bits. For example, after the 19th glider passes in Figure 7.20, the semi-Snarks will

[21] We do not particularly care about the factor of 30 here—constants like this do not matter much compared to the square root and logarithmic factors when n is large.

[22] There are also small **tremi-**, **quadri-**, and **quinti-Snarks** that multiply the period of a glider stream by 3, 4, and 5, respectively—see Exercises 7.22, 7.23, and 7.24.

Free PDF. See conwaylife.com/book for pattern files and print version.

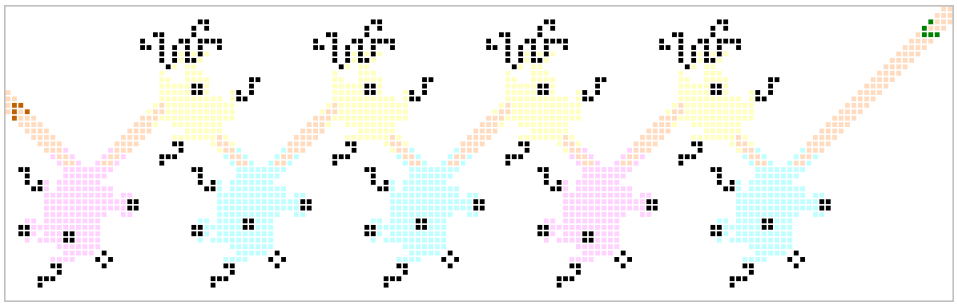


Figure 7.20: If we interpret a semi-Snark that blocks a glider (highlighted in magenta) as a "1" and a semi-Snark that lets a glider pass (highlighted in aqua) as a "0", then this arrangement of semi-Snarks corresponds to the binary representation 10010_2 of the number 18. The input glider toggles the least significant bit and causes the semi-Snarks to count down in binary, so this arrangement blocks the first 18 input gliders but lets the 19th glider pass.

encode the number $11111_2 = 31$ and will thus block the next 31 (not 18) gliders. To fix this problem, we can use the output glider to toggle whichever semi-Snarks we want to be set back to "0" before the next input glider hits the circuit. One method of implementing this toggle is presented in Figure 7.21.

We now know how to multiply the period of a glider gun by any amount of our choosing—just build a circuit like the one in Figure 7.21 and place it along the output path of the gun (just like we placed semi-Snarks along the output path of a gun to multiply its period by 2 in Figure 7.19). While this lets us create small guns of many new periods, it does not get us *all* periods (for example, no small prime-period gun can be created in this way).

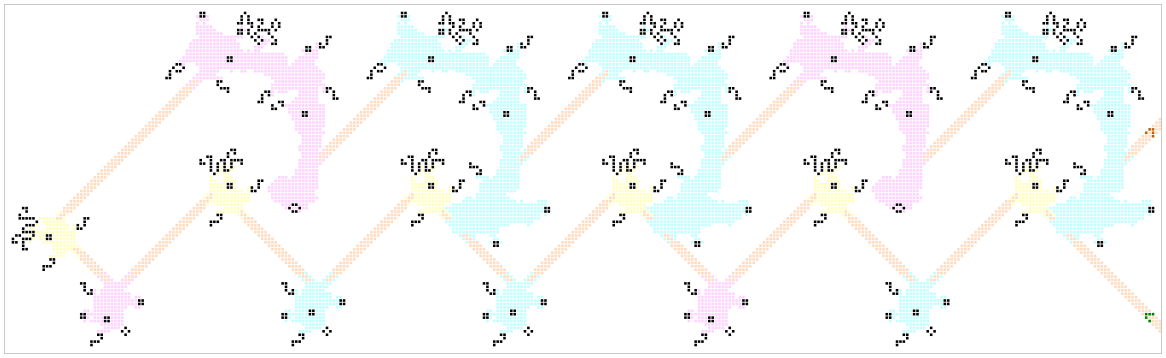


Figure 7.21: A stable circuit that blocks 18 out of every 19 input gliders (displayed in green at the southeast). As in Figure 7.20, the arrangement of semi-Snarks in the bottom row corresponds to the binary representation 10010_2 of the number 18, and the components in the top row of the circuit use the output glider (i.e., the 19th input glider) to reset the semi-Snarks to their original configuration. In particular, the aqua circuits at the top inject another glider into the bottom half of the circuit that toggles a semi-Snark back into the "0" position, whereas the magenta circuits at the top do not.

We can get around this problem by constructing a mechanism that *adds* a small number of generations to the period of a gun. One way to do this is to notice that the adjustable glider gun from Figure 7.17 can be turned on and off. That figure already demonstrates how to turn it on—just feed in one or more gliders from the southwest. To turn it off, we can fire a glider at either the east or west side so as to destroy the glider(s) that turned it on.

To then add a given number of generations to the period of this gun, we can use its output to turn the gun off for that many generations before turning it back on (with that precise timing of when the gun is turned back on being handled by trombone slides and the glider rephasers that we saw in Table 7.2). For example, if we start with a period 616 version of the adjustable gun from Figure 7.17

Free PDF. See conwaylife.com/book for pattern files and print version.

(really we are starting with the period 77 version of that gun, but we place only one glider along its track instead of 8, so that its period is $77 \times 8 = 616$), then we can add 1 087 generations to its period (for a total period of $616 + 1087 = 1\,703$) by attaching the mechanism displayed in Figure 7.22 to it. By adjusting the spacing and rephaser used in the trombone slide of this mechanism, any number of generations at least 1 087 can be added to the period of the gun.

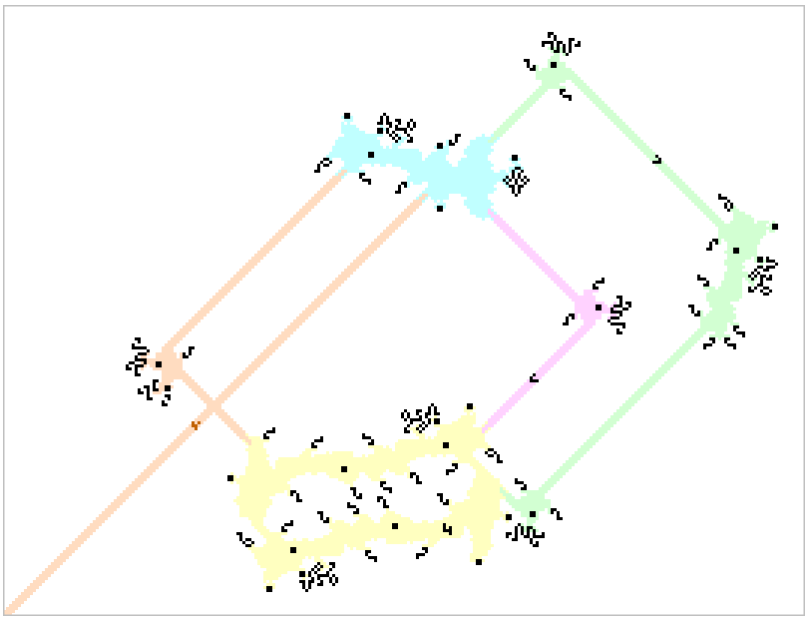


Figure 7.22: A period $616 + 1087 = 1703$ gun that works by feeding the output of a period 616 gun (highlighted in yellow) into a glider tripler (highlighted in aqua). One of the outputs of this tripler escapes to the southwest as the output of the gun, one goes southeast (highlighted in magenta) to destroy the Herschel/glider in the p616 gun, thus turning it off, and one goes northeast (highlighted in green) to turn it back on 471 generations later. By making the trombone slide at the northeast larger, and possibly changing the rephaser that it uses, we could also delay the glider that restarts the p616 gun by any number of generations larger than 471.

By combining all of these techniques, we can now construct small[23] guns of *any* (sufficiently large) period. For example, to construct a small gun with large prime period 3 413 277 319, we could start with a p616 glider gun. To figure out how to adjust it, we first compute

$$\lfloor (3\,413\,277\,319 - 1\,087)/616 \rfloor = 5\,541\,032,$$

so we want to multiply the p616 gun's period by this amount. Since 5 541 032 has binary representation $10101001000110010101000_2$, we arrange semi-Snarks (via the method of Figure 7.21) according to this bitstring, except we save some space by wrapping them in a spiral (like we did in Figure 7.19(b)) instead of placing them in a straight line. Finally, since

$$(3\,413\,277\,319 - 1\,087) \equiv 520 \pmod{616},$$

we use the mechanism from Figure 7.22 with an additional 520-generation delay to add $1\,087 + 520 = 1\,607$ generations to its period. The completed gun then has period

$$(616 \times 5\,541\,032) + 1\,607 = 3\,413\,277\,319,$$

[23] Just like earlier, by "small" we mean that if the gun has period p then the length and width of its bounding box are both $\Theta(\sqrt{\log(p)})$.

Free PDF. See conwaylife.com/book for pattern files and print version.

as desired, and is displayed in Figure 7.23.[24]

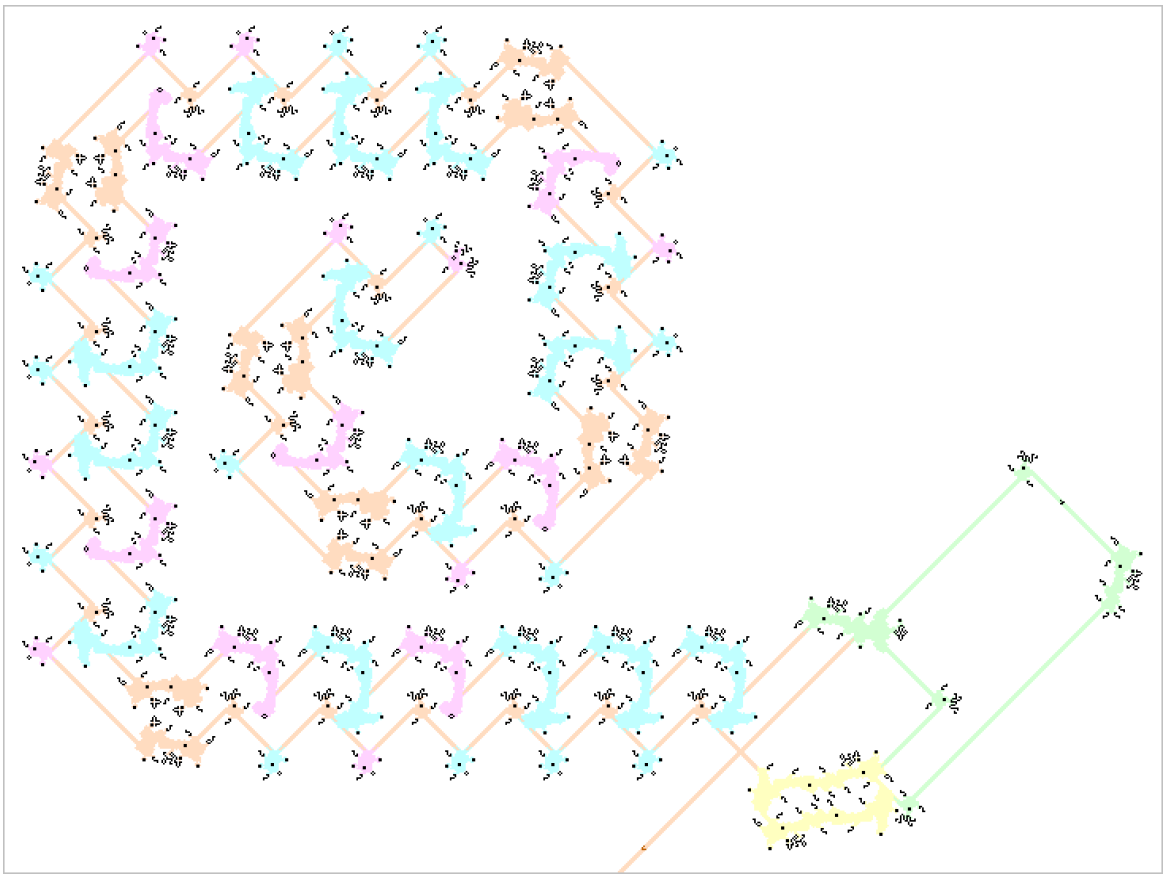


Figure 7.23: A period 3413277319 gun that uses two separate mechanisms to modify the period of a p616 gun (highlighted in yellow). First, its period is multiplied by $5541032 = 10101001000110010101000_2$ using the method of Figure 7.21—the binary representation is encoded by the semi-Snarks that are arranged in a spiral pattern, with the most significant 1 being the color-preserving semi-Snark highlighted in magenta at the center of the spiral. Next, its period is increased by 1607 via the mechanism of Figure 7.22 that toggles the p616 gun off for a brief period of time and then back on, resulting in a total period of $(616 \times 5541032) + 1607 = 3413277319$.

7.7 Converters for Other Objects

In addition to converting gliders to Herschels and back, it is sometimes also convenient to be able to convert between other commonly occurring moving objects. Herschel tracks are the most well-developed and well-used tracks, mainly because the Herschel is so much more mobile than other small chaotic objects, but there's no fundamental reason why we couldn't create a track that (for example) converts a glider into a Herschel, which is converted into an R-pentomino, followed by a pi-heptomino, and finally back into a glider. For ease of reference, we give these various frequently converted objects single-letter abbreviations as indicated in Figure 7.24.

Conduits that convert these objects into each other are typically referred to via abbreviations of the form "X-to-Y", where X and Y are the single-letter codes described by Figure 7.24. For example, a

[24] Most of the components that went into the construction of this gun were assembled by Chris Cain in December 2017, as was a script that automatically compiles a gun of this type for any period at least 1703. This script can be found at conwaylife.com/forums/viewtopic.php?p=54265#p54265.

Free PDF. See conwaylife.com/book for pattern files and print version.

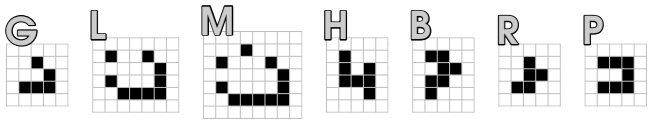


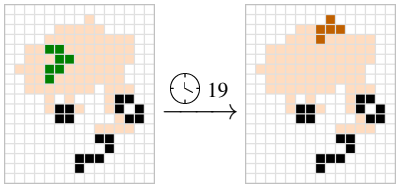
Figure 7.24: For brevity, when manipulating common small objects, we typically use the single-letter abbreviations G (glider), L (lightweight spaceship), M (middleweight spaceship), H (Herschel), B (B-heptomino), R (R-pentomino), and P (pi-heptomino). It is sometimes useful (for example, when naming conduits that convert between these types of objects) to be able to refer to a canonical phase and orientation of these objects, which are displayed here.

conduit that converts a Herschel into a pi-heptomino would be referred to as an H-to-P conduit. More specifically, we name converters in a manner similar to the naming scheme for Herschel conduits from Section 7.1, but with the single-letter abbreviations of the input and output objects pre- and post-pended. That is, we give these converters names of the form

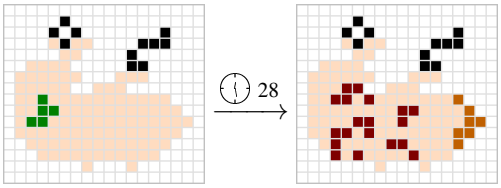
<input code><orientation><timing><output code>,

where <input code> and <output code> are the single-letter abbreviations of the input and output objects of the converter, <orientation> is the orientation prefix (i.e., R, Rx, L, Lx, F, Fx, B, or Bx) that describes how the output of the converter is rotated and/or reflected relative to the canonical phase displayed in Figure 7.24, and <timing> is the number of generations that it takes for the input object to be converted into the (canonical phase of the) output object.[25]

For example, the conduit displayed in Figure 7.25(a) takes 19 generations to convert a B-heptomino into an R-pentomino. Since the output R-pentomino is reflected top-to-bottom from its canonical phase shown in Figure 7.24 and then rotated to the left (i.e., counter-clockwise), it gets <orientation> prefix "Lx" (compare with Figure 7.2, where we introduced these <orientation> prefixes for Herschels). This conduit is thus named **BLx19R**. Similarly, the conduit displayed in Figure 7.25(b) takes 28 generations to convert an R-pentomino into a B-heptomino,[26] and the output heptomino is in its canonical orientation (i.e., it has <orientation> prefix "F") and is thus named **RF28B**.[27]



(a) BLx19R: A conduit that converts a B-heptomino to an R-pentomino in 19 generations.



(b) RF28B: A conduit that converts an R-pentomino to a B-heptomino in 28 generations.

Figure 7.25: Some conduits that convert B-heptominoes into R-pentominoes and back. The cells highlighted in red in (b) die off in another 6 generations.

When the converter in question just sends a Herschel to a Herschel, we typically omit the "H" <input code> and <output code> from its name, and thus recover the naming scheme that we developed for Herschel tracks back in Section 3.6. When the object that is being converted has

[25] An unfortunate collision in this naming convention is that "B" means both "180-degree rotation" and "B-heptomino" (and similar collisions happen with "L" and "R"). This is somewhat undesirable, but we can tell what the "B" stands for by its position in the name (e.g., it is a B-heptomino in BLx19R, but it is a 180-degree rotation in RB57P).

[26] Actually, this conduit does not produce the B-heptomino itself, but rather the **B-heptaplet** that evolves in the same way.

[27] The RF28B conduit appears in some of the Herschel conduits that we saw back in Table 7.1: Bx202, L156, and Rx164 (see Exercise 7.33).

Free PDF. See conwaylife.com/book for pattern files and print version.

additional symmetry, conduits that manipulate it can have multiple valid names. For example, the conduit displayed in Figure 7.26 takes 8 generations to turn a pi-heptomino either right (clockwise) or left (counter-clockwise), depending on its orientation. This conduit could thus be called either **PR8P** or **PL8P**, and we typically just pick the name that corresponds to the orientation of the conduit that we are using.



(a) PL8P: Turning a pi-heptomino left.

(b) PR8P: Turning a pi-heptomino right.

Figure 7.26: PL8P or **PR8P**: A conduit that takes 8 generations to turn a pi-heptomino by 90 degrees.

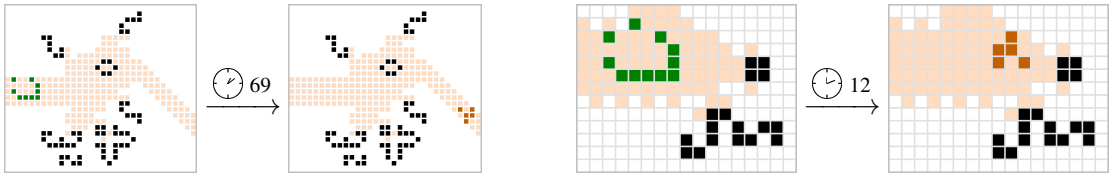
It is worth pointing out that we have seen this conduit before—it was used in Figure 3.19(a) to create the p32 "gourmet" oscillator, and also as part of Tanner's p46 in Figure 3.19(c). In a sense, those oscillators (and a few other hasslers that we have seen) function in the same way as Herschel track oscillators. We will see another example of how we can build hasslers out of these types of conduits in Exercise 7.28.

When the input or output of a conduit is a spaceship like a glider, the naming convention used is a bit more intricate and involves codes of the form

<input code><direction><lane>T<timing><output code>,

much like in Section 7.2 (and if either of the input or output object are a glider, we omit the corresponding <input code> or <output code>). Just as was the case back then, we do not dwell on the details of how the <lane> or <timing> values are computed, but instead we just note that the basic movement of the conduit can be gleaned by reading its name.

For example, a conduit named **LSE11T-8** transforms a lightweight spaceship into a glider traveling southeast in lane 11 (i.e., roughly 11 cells to the right of where the LWSS hits the conduit) and timing -8 (i.e., delayed by 8 generations from the destruction of the LWSS), whereas a conduit named **MSW-1T1** transforms a middleweight spaceship into a glider traveling southwest on lane -1 and timing 1 (i.e., the glider starts its journey almost exactly when and where the MWSS hits the conduit). These converters are displayed in Figure 7.27.



(a) LSE11T-8: Turns an LWSS into a glider. Found by Ivan Fomichev in October 2015.

(b) MSW-1T1: Turns an MWSS into a glider. Found by Matthias Merzenich in July 2013.

Figure 7.27: Some conduits that turn xWSSes into gliders.

A selection of conduits that convert between these basics types of objects is presented in Table 7.3. We emphasize that this collection of converters is not even close to complete, but rather just consists of some particularly small, fast, and/or important representative examples.[28] We also clarify that a

[28] A much larger and more complete catalog of stable converters is available online at conwaylife.com/forums/viewtopic.php?f=2&t=1849

Free PDF. See conwaylife.com/book for pattern files and print version.

cell being empty in Table 7.3 does not mean that we know of *no* converters of the indicated type, but rather that we do not know of any simple, fast, and small converter of that type. By combining the various small converters using the techniques we have already seen, we can convert essentially any type of object to any other type of object, as long as we are comfortable using large converters that make use of several intermediate conversions. For example, to convert a glider into a pi-heptomino, we could perform the following sequence of conversions (see Exercise 7.31):

$$\text{glider} \xrightarrow{\text{syringe}} \text{Herschel} \xrightarrow{\text{HF95P}} \text{pi-heptomino.}$$

from \ to	glider (G)	Herschel (H)	B-hept. (B)	R-pent. (R)	pi-hept. (P)
G	Snark, Figure 7.9	syringe	–	–	–
H	Figure 7.5	Section 7.1	HFx58B	HLx69R	HF95P
B	BSE22T31	BFx59H	BRx46B	BLx19R	BF22P
R	RNW3T46	RR56H	RF28B	RFx36R	RF29P
P	PNW6T138	PF81H	PR9B	PR127R	PL8P

Table 7.3: Some conduits that convert one type of object into another. Most of these conduits are quite old and well-known.

7.8 Factories

Somewhat surprisingly, it is not only useful to convert moving objects into other moving objects like Herschels or gliders, but also into stationary objects like still lifes and oscillators. A conduit that implements such a conversion is called a **factory**, as is any other pattern that repeatedly creates a still life or oscillator.

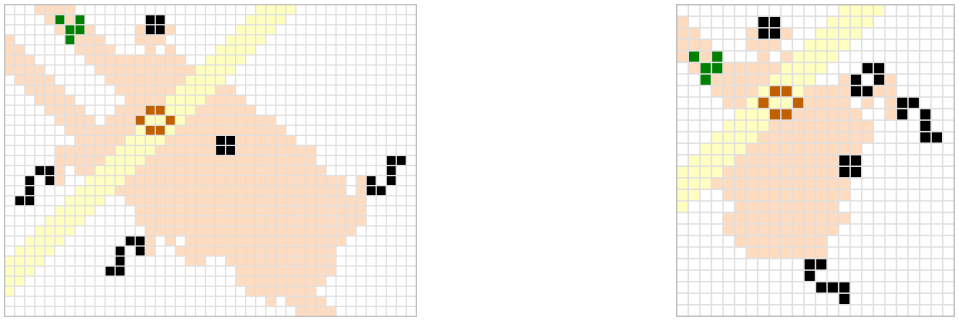
In most cases,[29] a factory will self-destruct if we do not make use of the still life or oscillator

[29] But not all cases—see Figure 7.31.

Free PDF. See conwaylife.com/book for pattern files and print version.

that it creates before it tries to make another one. For example, the queen bee can be thought of as a beehive factory (refer back to Figure 1.15), which self-destructs unless we regularly destroy the beehives that it leaves behind (thus creating a queen bee shuttle).[30] We can similarly construct factories for other small still lifes and oscillators by aiming two or more glider guns at each other so that the glider collisions synthesize the desired object. Once again though, we must regularly make use of and destroy that synthesized object, or else it will interfere with subsequent syntheses and cause the factory to explode.

As an example that is slightly more relevant to our current interests, consider the conduits displayed in Figure 7.28 that convert a glider into a beehive.[31] The reason that these conduits are useful is that they can act as logic circuits that test whether or not a signal (i.e., a glider) has come in from the northwest. Indeed, when such a signal comes in, a beehive is created, and we can test whether or not this has happened by sending in a glider from the north*east*. If a beehive has been created then it will block (and be destroyed by) this test glider, whereas if no beehive has been created then this test glider will pass through the conduit to the southwest unimpeded.



(a) Glider stopper (b) Beehive stopper

Figure 7.28: Some factories that use a glider (displayed in green) to create a beehive (displayed in orange). If the beehive is present, it destroys (and is destroyed by) a single glider coming from the northeast on the lane highlighted in yellow. Otherwise, that glider passes through the factory unharmed. Found by (a) Paul Callahan in 1996 and (b) Tanner Jacobi in 2015.

Since the still life or oscillator that is created by a factory does not move, it is especially important that it is created near the factory's edge, thus making it accessible to other nearby circuitry. For this reason, the edgy Herschel-to-beehive conduit displayed in Figure 7.29(a) is particularly useful,[32] as is the edgy Herschel-to-loaf factory of Figure 7.29(b).

Since a boat can be used as a one-time turner (see Exercise 5.31), factories that create boats can be used to reflect gliders by 90 degrees. For example, the Herschel-to-boat factories of Figure 7.30(a) can be used to reflect gliders heading northwest so that they instead go northeast. Another Herschel-to-boat factory, called the **demultiplexer**, is displayed in Figure 7.30(b).[33] While this conduit is significantly less edgy than those of Figure 7.30(a), it has the advantage that if the boat is *not* present then that glider simply passes through the factory unharmed. It can thus serve as a logic circuit in much the same way as the beehive-based factories from Figure 7.28—a glider coming in from the northeast can be used to test whether or not a Herschel has been fed into the conduit.

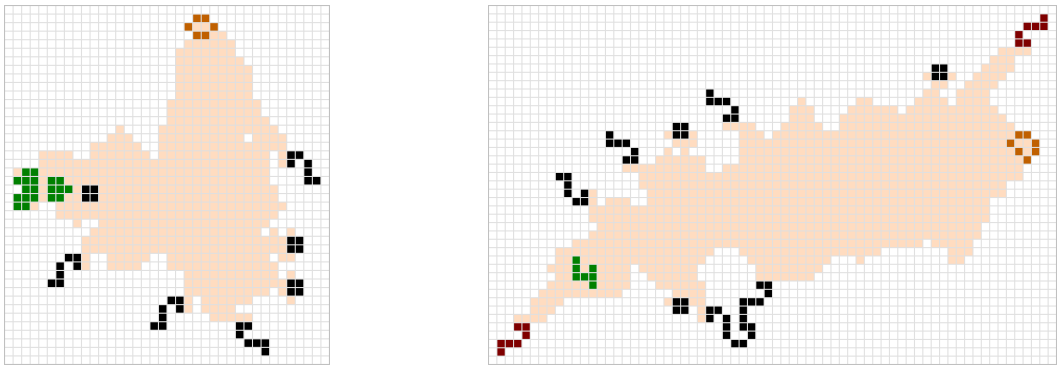
[30] For an almost-oscillator that similarly acts as a block factory, see Exercise 7.41.

[31] The **glider stopper** in Figure 7.28(a) is bigger than the **beehive stopper** in Figure 7.28(b), but has the advantage of also producing 2 extra output gliders while making its beehive.

[32] The input object in this conduit is actually a great-grandparent of a Herschel. The block interferes with it before it has a chance to fully evolve into a Herschel, but it can nonetheless be attached to other circuits that output Herschels without a problem.

[33] Found by Brice Due in August 2006.

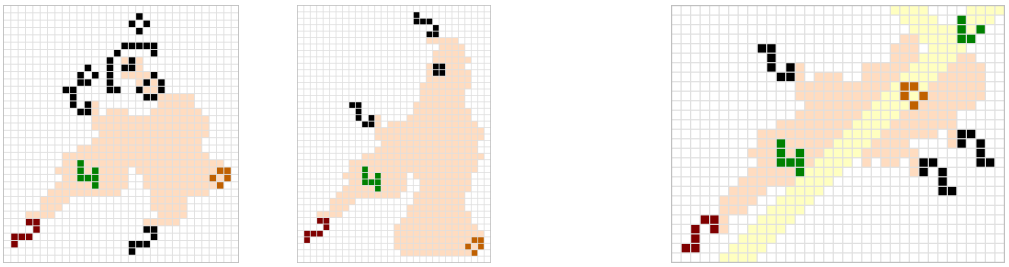
Free PDF. See conwaylife.com/book for pattern files and print version.



(a) Herschel-to-beehive **(b)** Herschel-to-loaf

Figure 7.29: Some edgy conduits that can convert a Herschel (displayed in green) into a small still life (displayed in orange). The (b) loaf factory was found by Adam P. Goucher in 2009.

The other most common type of still life factory is one that creates a block, and several of them are displayed in Figure 7.31. We note that Herschels create a block on their own (called the **first natural block**[34]) in generation 37. For this reason, some Herschel-to-block factories (like the two leftmost ones in Figure 7.31) are particularly simple and work just by cleaning up the input Herschel's debris after it makes that block. The two rightmost factories in Figure 7.31 are somewhat larger and more complicated than the other two, but are useful for the fact that they product blocks so far away from their circuitry.



(a) Herschel-to-boat **(b)** Demultiplexer

Figure 7.30: Some edgy Herschel-to-boat factories that can be used to reflect gliders. In (b), the input glider follows one of the two paths highlighted in yellow: it is reflected if the boat is present, and it passes straight through the factory otherwise.

Furthermore, these two more complicated block factories have the remarkable property that they do not self-destruct if an input Herschel is received while the output block is still present. In the second-from-the-right block factory, a second Herschel moves the block and produces an output glider, and a third Herschel simply destroys that block, returning it to its original state. This conduit can thus act as a period tripler for a glider or Herschel stream (see Exercise 7.40).

In the rightmost of these block factories, a second Herschel simply has no extra effect—if the output block is already present, it is temporarily destroyed but then rebuilt in the exact same spot. Factories with this remarkable property of being able to accept multiple input signals without affecting the output objects are called **keepers**.

[34] In analogy with the first natural glider that they make in generation 21.

Free PDF. See conwaylife.com/book for pattern files and print version.

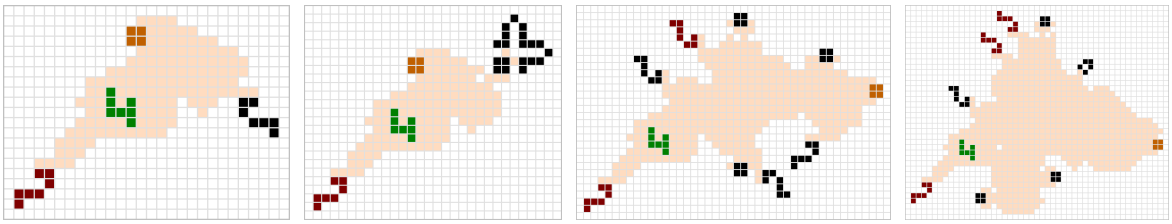


Figure 7.31: Some Herschel-to-block factories.

7.8.1 Highway Robbers

While we have seen numerous glider reflectors so far—the Snark, the Silver reflector, bumpers, bouncers, and Herschel-track-based reflectors—they all accept their input glider somewhat near their center. It is thus difficult to use these mechanisms to reflect a stream of gliders that is just a few lanes away from another stream. As an application of still life factories, we now construct a stable glider reflector that overcomes this problem and reflects gliders from a particular lane without being affected whatsoever by gliders on any further away lanes (even directly adjacent ones). A pattern with this property is called a **highway robber**.[35]

While there is no known "quick" or "direct" stable pattern that carries out this task (like the Snark or the syringe for their tasks), we have seen all of the tools needed to construct a somewhat large and slow one. The key observation that we need is that many still lifes, if they are hit at their edge by a passing glider, produce some chaotic debris or a perpendicular glider. In order to construct a highway robber, we will funnel that debris or glider through a conduit back into a factory that replaces the destroyed still life (and also sends off one or more signals along the way).

Of the still lifes that we know how to create via factories, the loaf is best-suited to this task since a glider hitting its edge almost immediately produces a perpendicular glider, along with some extra debris.[36] We can simply place a nearby eater 1 so as to clean up that debris, as displayed in Figure 7.32, so that the loaf extracts the glider from its lane.

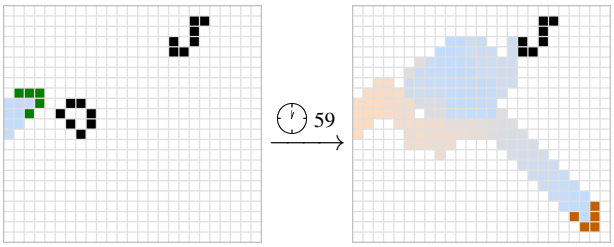


Figure 7.32: When a glider just barely hits a loaf, a perpendicular glider and some debris are created. An eater 1 can be used to absorb the extra debris.

All that we have to do to create a highway robber out of this reaction is use the newly created perpendicular glider to recreate the loaf. Doing so is just a matter of feeding the glider into a syringe to turn it into a Herschel, and then feeding that Herschel into the Herschel-to-loaf factory of Figure 7.29(b). The resulting highway robber is displayed in Figure 7.33. While it is somewhat large and slow, only mild improvements on this design are known (see Exercise 7.35).

[35] The name refers to the fact that highway robbers "steal" gliders from their lane.

[36] If hit on its edge by a glider, a (1) block quickly moves via the $(2,1)$ block pull of Figure 2.20, a (2) beehive (depending on its orientation) quickly dies or evolves into lumps of muck, and a (3) boat (depending on its orientation) quickly dies, produces a block, produces a honey farm, or makes debris that is difficult to clean up since it is on the opposite side of the glider.

Free PDF. See conwaylife.com/book for pattern files and print version.

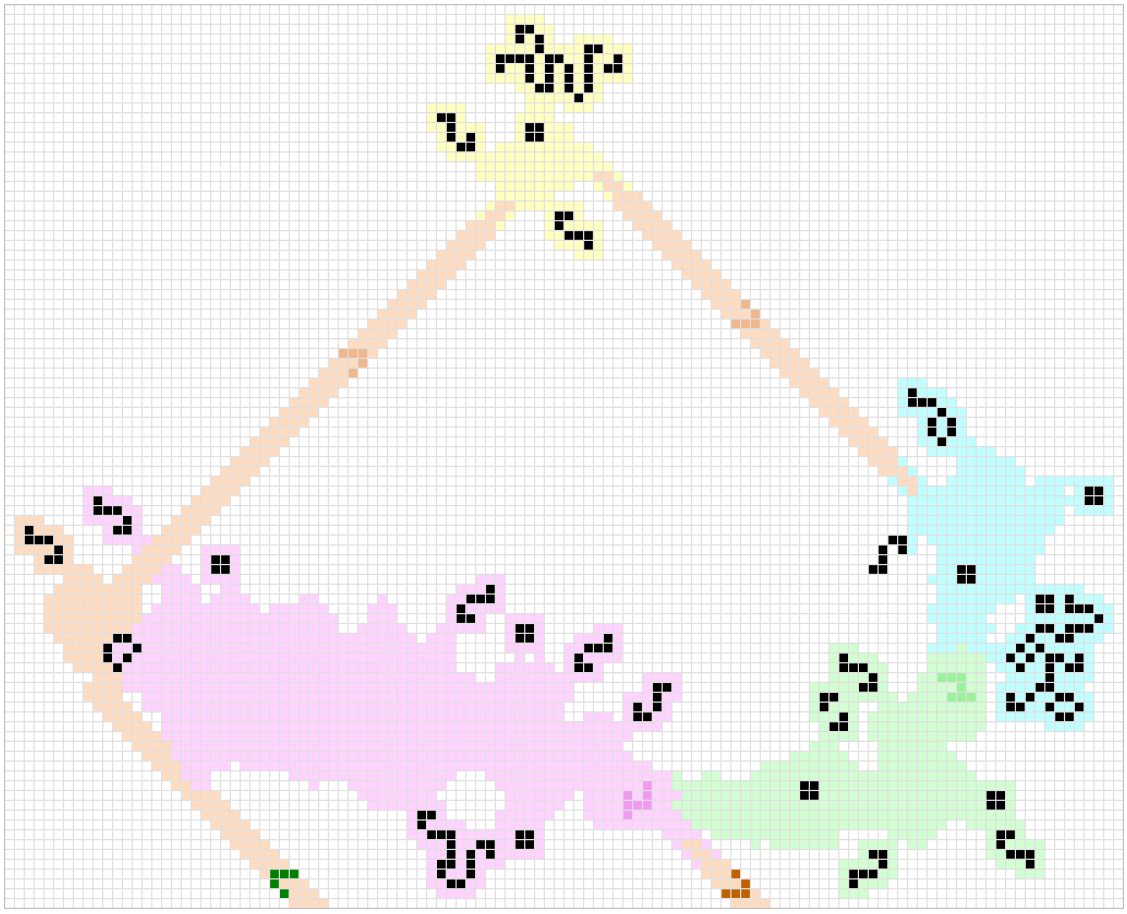


Figure 7.33: A highway robber that was constructed by Chris Cain in March 2015. It uses the glider-loaf collision of Figure 7.32 to create a perpendicular glider that is fed into a Snark (highlighted in yellow) and then a syringe (highlighted in aqua), an L112 conduit (highlighted in green), and finally a Herschel-to-loaf factory (highlighted in magenta) so as to recreate the loaf.

7.8.2 A Stable Heisenburp

Still life factories can also be used to construct stable Heisenburps—stable patterns that can detect the presence of a spaceship without affecting it at all, even temporarily (recall that we saw some *periodic* Heisenburps in Section 6.3.3). No such pattern can be constructed to detect the presence of a glider, since gliders do not emit any sparks. However, xWSSes are plenty sparky, and we now construct a stable Heisenburp for an MWSS.

Much like our highway robber, our starting point is a reaction in which an MWSS triggers a still life to explode into a chaotic mess and/or a glider. Unfortunately, none of the still lifes that we have seen factories for are particularly well-suited to this task,[37] so we instead use a ship as illustrated in Figure 7.34. This is slightly inconvenient, since we do not know of any clean and edgy ship factories. However, a Herschel on its own evolves into a configuration of two blocks, two gliders, and a rather edgy ship, as illustrated in Figure 7.35.[38]

Both of these reactions are a bit dirtier than we would like—the ship-plus-spark explosion

[37] If triggered by an MWSS's central spark, a (1) block either dies quickly or destroys the MWSS, a (2) beehive destroys the MWSS, a (3) boat destroys the MWSS, and a (4) loaf in one orientation produces some chaotic junk, but it is not known how to quickly or easily convert that junk into one of our standard signals like a Herschel or a glider.

[38] So, in a sense, the empty Life plane is a somewhat dirty Herschel-to-ship factory.

Free PDF. See conwaylife.com/book for pattern files and print version.

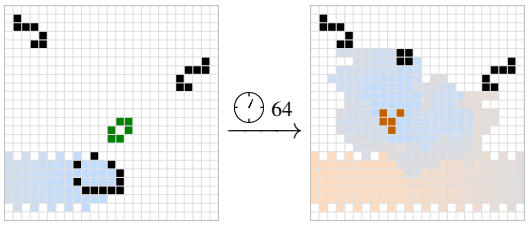


Figure 7.34: The spark from an MWSS can cause a ship to explode (without destroying the MWSS). Some eater 1s can be used to turn that explosion into a block and a glider.

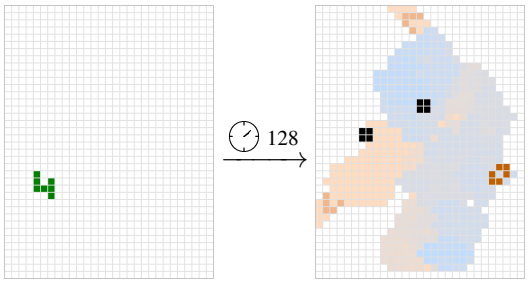


Figure 7.35: A Herschel on its own evolves into two blocks, two gliders, and a ship.

creates an extra block that we will have to clean up, and the Herschel-to-ship conversion creates two extra blocks and two extra gliders. However, all of these extra bits and pieces can be cleaned up straightforwardly by using the stable circuitry techniques that we learned in this chapter to redirect those extra gliders (and possibly create more of them) so that they collide with, and destroy, the extra blocks. A completed stable MWSS Heisenburp that uses these ideas is displayed in Figure 7.36.[39]

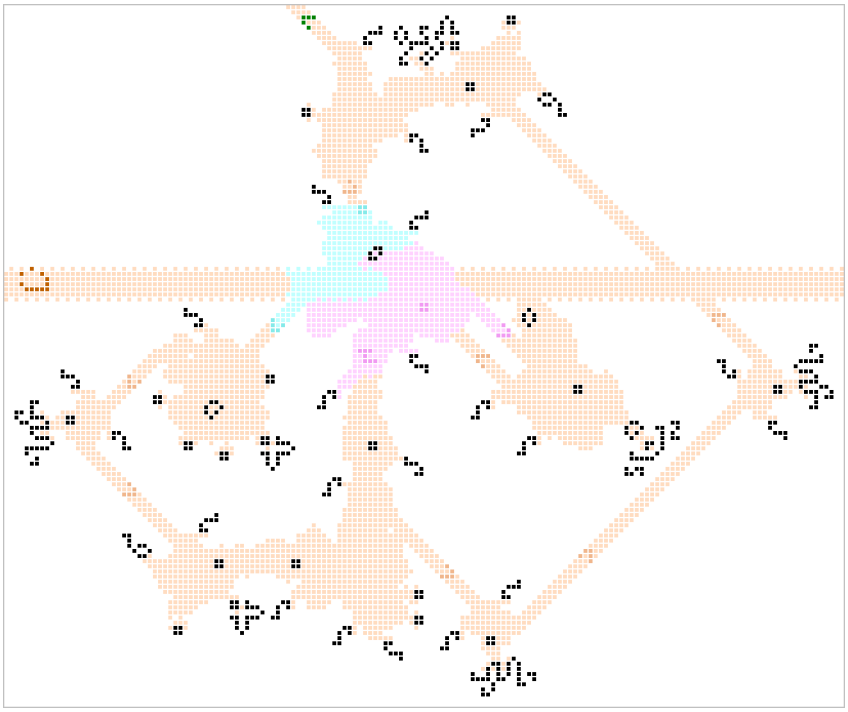


Figure 7.36: A stable MWSS-detecting Heisenburp that was constructed by ConwayLife.com forums member "Entity Valkyrie" in June 2020, largely based on earlier work by Martin Grant. The MWSS's spark triggers the central ship to create a glider as in Figure7.34 (highlighted in aqua), and stable circuitry is then used to manipulate that glider into rebuilding the ship as in Figure 7.35 (highlighted in magenta). Some glider duplicators and reflectors are used to destroy extra stray blocks.

[39] Similar ideas can be used to construct a stable HWSS Heisenburp—see Exercise 7.44.

Free PDF. See conwaylife.com/book for pattern files and print version.

7.9 Notes and Historical Remarks

While stable circuitry for carrying out the tasks described in this chapter has existed for a couple of decades at this point, it became significantly smaller and easier to work with in the mid-2010s. For example, the discovery of the Snark in 2013 allows us to easily reposition gliders without having to make use of large engineered reflectors like the Silver reflector from Figure 3.29(a).

The discovery of the syringe in 2015 similarly shrank most stable circuits considerably. It has always been easy to convert a Herschel into a glider, but the smallest and fastest way to accomplish the reverse task in previous years was to use the reaction from Figure 7.37 in which a glider is converted into a Herschel and an extra "junk" beehive.[40] That beehive must be destroyed before the conduit can be used again—a task that can be carried out by using stable conduits to redirect a Herschel's first natural glider back to the beehive. The simplest sequence of stable conduits that does the job is Fx77 $\rightarrow$ L112 $\rightarrow$ Fx77, and the resulting glider-to-Herschel conduit is called the **Callahan G-to-H** (see Figure 7.38). Because of the slow creation of its cleanup glider, it has a rather high repeat time of 575 generations.

Indeed, the Silver reflector that we saw way back in Figure 3.29(a) works in almost the exact same way—it uses the exact same initial component to turn the input glider into a Herschel and junk beehive, and even follows it up by Fx77 and L112 conduits. The only difference is that the final Fx77 conduit from the Callahan G-to-H is replaced by the NW31 conduit from Figure 7.5(b). Since NW31 puts an output glider on the same lane as Fx77, but much sooner, this gives Silver reflectors a slightly smaller repeat time of 497 generations.

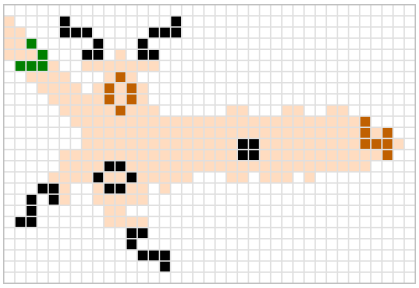


Figure 7.37: A stable conduit that converts a glider into a Herschel and a beehive. By using a second glider to clean up the beehive, this can be thought of as a 2G-to-H converter.

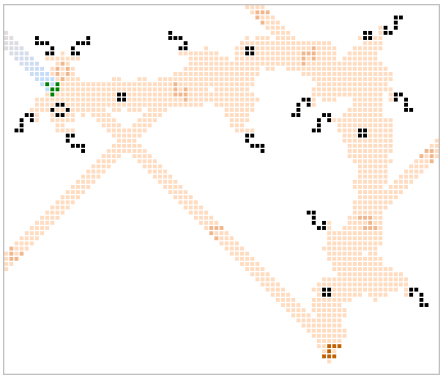


Figure 7.38: A **Callahan G-to-H** that converts a single glider into a Herschel via the reaction of Figure 7.37.

Silver reflectors are fairly large and slow by modern standards. Oddly enough, though, they're still top-of-the-line technology for certain applications:

- It costs just about the same number of gliders to synthesize them as any other known reflector.
- The construction recipe is very simple because Silver reflectors (and Callahan G-to-Hs) are Spartan.
- Silver reflectors have two separate transparent output lanes, so they can be used as merge circuits.
- Silver reflectors can be trivially adjusted to produce output gliders in any or all of the four diagonal directions (see Exercise 7.46), so they're also fairly efficient signal splitters.
- The various output gliders can be blocked or released to produce either color-changing or color-preserving 90-degree reflections.

Free PDF. See conwaylife.com/book for pattern files and print version.

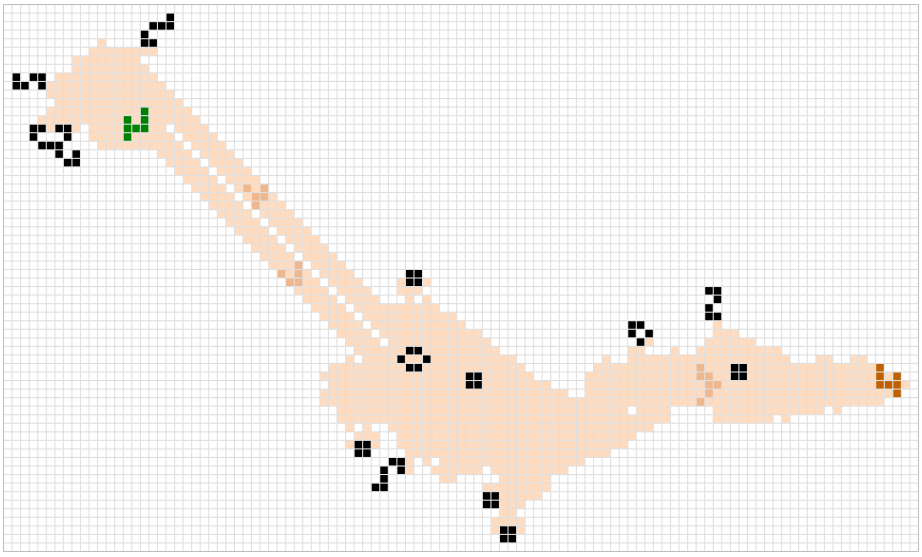


Figure 7.39: A Herschel transceiver that is made up of two components that can be separated by an arbitrary amount—a conduit that converts a Herschel into two gliders at the top-left (found in May 1997 by Paul Callahan), and a conduit that converts those two gliders back into a Herschel at the bottom-right (found in October 1996, also by Callahan). This device appeared in lots of pre-2015 circuitry, but was made almost completely obsolete by the discovery of the syringe.

In general, Herschel-to-glider converters and other conduits can be appended to a syringe to create compact circuits with each of the Silver reflector's good qualities (constructibility, merge capability, signal-splitting ability, or two colors of glider output), as well as many other possible logic-circuit functions. However, duplicating *all* of those qualities in a single converter requires adding so much circuitry that the result may actually be larger than a Silver reflector.

In order to transmit a Herschel from one place in the Life plane to another one that is far away, nowadays we can simply convert it into a glider, and then convert it back via a syringe. Callahan's G-to-H could be used for this same task as of 1998, but a much smaller and faster method of doing so was already known by May 1997. Indeed, a device called a **Herschel transceiver**, which is displayed in Figure 7.39, converts a Herschel into a *pair* of gliders travelling the same direction (but potentially on different lanes—such gliders are said to be in **tandem**) and then back into a Herschel.

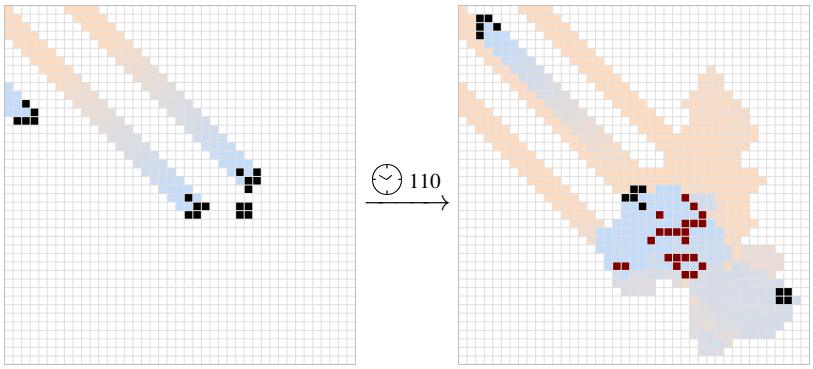


Figure 7.40: A reaction in which 3 gliders push a block southeast by 10 cells and send a pair of tandem gliders back, separated by just 2 lanes and thus suitable for input into the Herschel receiver from Figure 7.39. The debris highlighted in red dies off in 25 more generations.

[40]Found by Paul Callahan in November 1998.

Free PDF. See conwaylife.com/book for pattern files and print version.

The receiving half of this Herschel transceiver (which is called a **Herschel receiver**) just uses the first input glider to erase its beehive, so the relative timing of the input gliders does not really matter. Furthermore, there are several other ways that a glider can collide with a beehive so that they destroy each other, so the second glider can be any of 6, 5, 2, or -2 lanes to the right of the first glider without affecting the receiver's functionality. Several more transmitting halves (i.e., **Herschel transmitters**) for this receiver were found in the next year or two,[41] but they were relatively awkward, needing sparks from oscillators or other extra cleanup, so they were seldom or never used in larger patterns. One H-to-G6 conduit (i.e., Herschel transmitter that converts a Herschel into a pair of tandem gliders offset from each other by 6 lanes) that *is* still useful is presented in Exercise 7.34.

With the appearance of the syringe in 2015 (with a repeat time of 78 generations, instead of the Herschel transceiver's 117), it became cheaper and faster to send a single glider and then convert it back to a Herschel with a syringe whenever necessary. As a result, tandem gliders pretty much became obsolete, with a few odd exceptions. When we happen to have two gliders anyway, or when a pair of tandem gliders naturally produces some other reaction of interest, it is still sometimes easier to catch them with a Herschel receiver. One such reaction that we will make use of later is the one displayed in Figure 7.40 that uses 3 gliders to push a block forward while sending a pair of tandem gliders back.

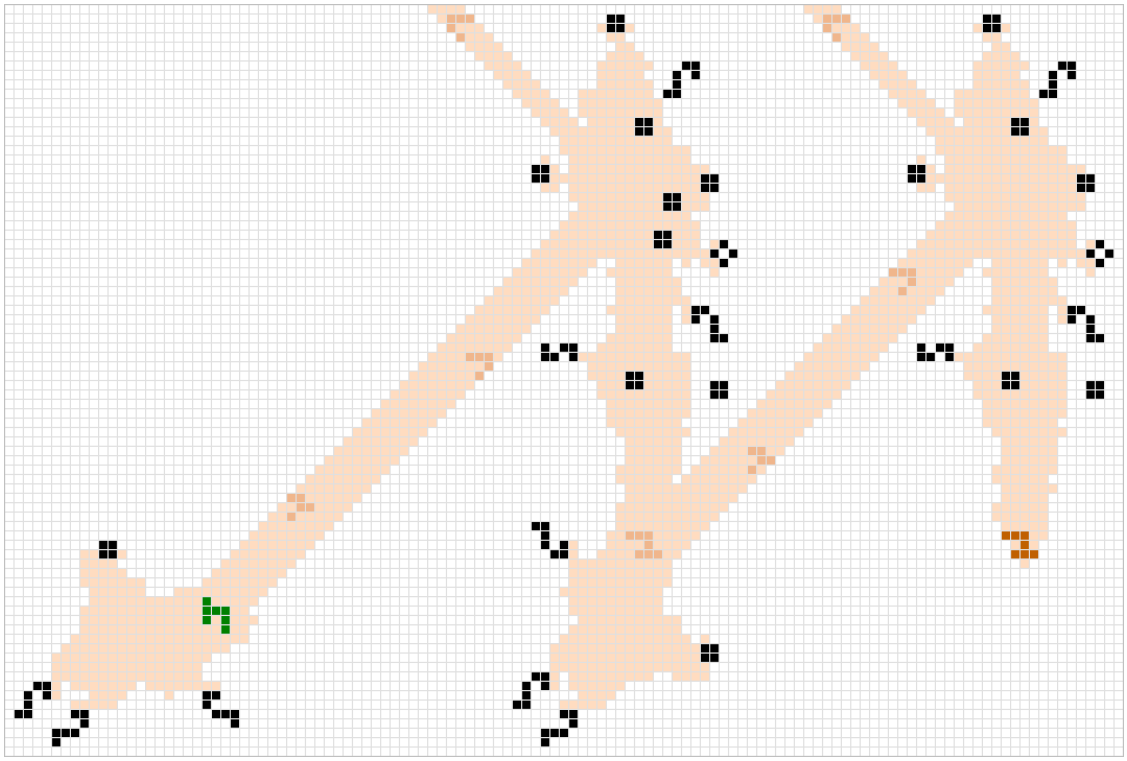


Figure 7.41: A Herschel transceiver that works via tandem glider pairs that are separated by 4 lanes. The receiver that it uses is ambidextrous—it can be configured to work regardless of which of the two input gliders arrives first, as demonstrated here by the middle and right receivers here taking different configurations of input gliders.

Another Herschel transceiver that is sometimes useful is the one displayed in Figure 7.41 that works via gliders that are separated by 4 lanes instead of 2, 5, or 6.[42] This transceiver has two advantages over the one that we saw earlier: it produces a quick 90-degree glider output in addition to

[41] By Paul Callahan and by Dieter Leithner.

[42] Found by Sergei Petrov in December 2011.

Free PDF. See conwaylife.com/book for pattern files and print version.

its Herschel output, and it is **ambidextrous** (i.e., it can be configured to accept the gliders from the transmitter in either order). For this reason, this transceiver is still a fairly efficient way to quickly split a single signal into multiple signals via Spartan components.

Alternatively, prior to the discovery of the syringe, there were stable conduits known for converting some spaceships *other* than gliders into Herschels. For example, the conduit displayed in Figure 7.42 converts a Coe ship into a Herschel. However, the reverse transformation of a Herschel back into the original spaceship was much less straightforward, and would be performed by converting the Herschel into multiple gliders that would then synthesize the original spaceship (a Coe ship in this case) via a glider synthesis like the one that we saw in Figure 5.11.

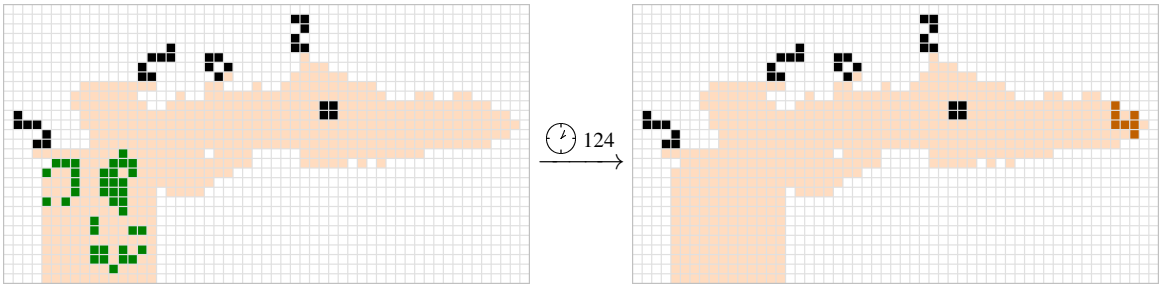
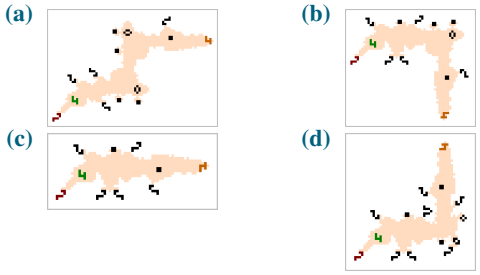


Figure 7.42: A stable conduit that converts a Coe ship into a Herschel. Found by Stephen Silver in September 1997.

Exercises

solutions to starred exercises on page 461

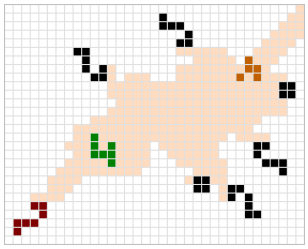
∗**7.1** [1/5] Give the name (using the naming scheme of Table 7.1) for each of the following Herschel conduits, as well as their repeat time.



7.2 [1/5] Construct a stable pattern that eats a Herschel. [Hint: If you cannot find a "direct" way to do this, convert a Herschel into something that you already know how to eat.]

∗**7.3** [2/5] Modify the conduit from Figure 7.6 so that it has two side-by-side transparent lanes instead of just one. [Hint: There are other ways to eat a glider.]

∗**7.4** [2/5] A conduit that is very similar to NE5T-4 (see Figure 7.6) is presented below. Explain why this conduit might be more useful than NE5T-4 in some situations.



7.5 [1/5] In the F117 conduit from Table 7.1, the six catalysts serve five different functions. Determine which of those still life(s) can be classified as...

- **(a)** a glider eater,
- **(b)** a block eater,
- **(c)** a transparent catalyst,
- **(d)** a catalyst that is *not* simply an eater, and
- **(e)** a rock.

∗**7.6** [1/5] The syringe works by converting the input glider into an intermediate object that we have seen before, and then converting that object into a B-heptomino, which creates a glider (which is eaten) and the output Herschel. What is the name of the intermediate object?

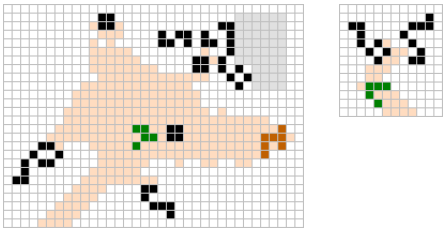
Free PDF. See conwaylife.com/book for pattern files and print version.

7.7 [2/5] In the Fx77 conduit from Table 7.1, the four catalysts serve four different functions. Determine which of those still life(s) can be classified as...

- **(a)** a glider eater,
- **(b)** a blinker eater,
- **(c)** a transparent catalyst, and
- **(d)** a catalyst that is *not* simply an eater.

Also, explain why the catalyst (d) is not simply a block eater, unlike the somewhat similarly placed eater 1 in the F117 conduit.

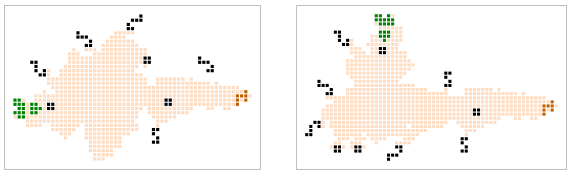
∗**7.8** [3/5] A slightly more compact version of the syringe can be constructed by modifying its large unnamed still life. Complete the partial syringe on the left below by filling in the light gray cells appropriately. Make sure that the resulting pattern can eat the glider that the output Herschel emits.



[Hint: Make use of the modification of eater 5 that is displayed on the right.]

7.9 [3/5] Construct a glider synthesis of the large version of the syringe displayed in Figure 7.8(b).

∗**7.10** [2/5] The right half of the large version of the syringe from Figure 7.8(b) is a conduit that is called F166 (displayed below on the left). Create another large version of the syringe by replacing F166 with Lx200 (displayed below on the right). What is the repeat time of this new version of the syringe?



[Side note: In both of these conduits, the input is a great-grandparent of a Herschel. The reason that these two conduits can be attached to the syringe is that the input Herschel's first natural glider is suppressed by the initial collision with the block.]
[Hint: You might have to fiddle with the orientation of some eaters so that Lx200 does not collide with the other half of the syringe.]

7.11 [1/5] Create a stable color-changing reflector that consists of a Snark followed by a Bandersnatch (whereas the reflector from Figure 7.10 has them in the opposite order).

7.12 [2/5] Bandersnatch-based glider reflectors can be used in place of some of the reflectors from Table 7.2.

- **(a)** Which reflector from that table can be replaced by a Bandersnatch together with a Snark (i.e., the reflector from Figure 7.10)?
- **(b)** Which reflector from that table can be replaced by a Bandersnatch together with *two* Snarks?

∗**7.13** [2/5] Make a stable conduit that converts one glider into exactly...

- **(a)** 4 gliders.
- **(b)** 5 gliders.
- **(c)** 10 gliders.

7.14 [1/5] One of the still lifes near the eastern edge of the glider-to-LWSS circuit in Figure 7.12(b) is displayed in red. Why is it singled out like this—what makes it different from the other still lifes in that circuit?

7.15 Use any glider syntheses of your choosing to construct a stable circuit that converts a single glider into the following objects.

- **(a)** [3/5] A middleweight spaceship.
- **(b)** [4/5] A heavyweight spaceship.
- **(c)** [5/5] A Schick engine.

7.16 Another toolkit for duplicating and synchronizing gliders in stable circuitry, which can be used instead of Table 7.2, is provided at `conwaylife.com/forums/viewtopic.php?p=41563#p41563`.

- **(a)** [4/5] Use this toolkit to build a stable glider-to-LWSS converter.
- **(b)** [5/5] Use this toolkit to build a stable glider-to-2-engine-Cordership converter.

∗**7.17** The trombone slide displayed in Figure 7.13(a) delays a glider by 144 generations.

- **(a)** [1/5] Bring the north and east Snarks as far southwest as possible without breaking the trombone slide. How many generations is the glider delayed by after making this change?
- **(b)** [1/5] Bring the south and east Snarks as far northwest as possible without breaking the trombone slide. Does this change affect how much the glider is delayed?
- **(c)** [3/5] By welding two Snarks together, bring the south and east Snarks northwest by 1 more cell than you did in part (b).

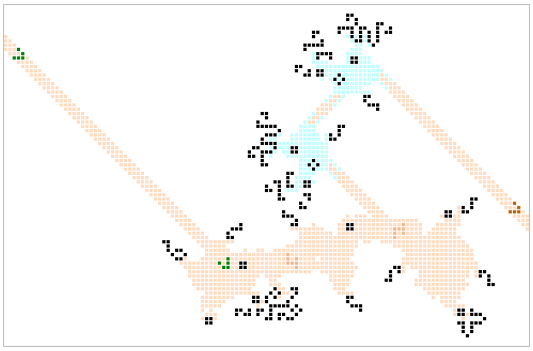
∗**7.18** [2/5] The 2-direction glider synthesis of the 2-engine Cordership from Exercise 5.14(b) consists of 11 gliders, but when we used it to start our construction of the glider-to-(2-engine Cordership) circuit in Figure 7.14, we only needed 10 input gliders. Explain this discrepancy—what happened to the 11th glider?

Free PDF. See conwaylife.com/book for pattern files and print version.

7.19 [2/5] In Figure 7.21, we saw a stable circuit that destroys all except for 1 out of every 19 input gliders. Construct a similar circuit that destroys all except for 1 out of every...

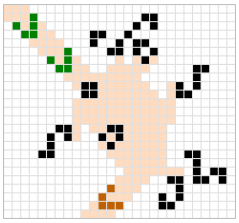
- **(a)** 32 gliders.
- **(b)** 6 gliders.
- **(c)** 47 gliders.

***7.20** [3/5] The conduit below uses two (color-preserving) semi-Snarks to transform two input gliders into a single output glider.



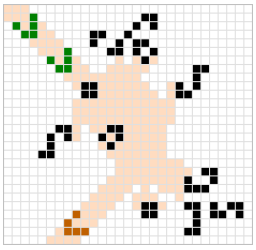
In this exercise, we use this device to develop an alternate set of stable color-preserving glider rephasers that can implement arbitrary glider delays, which can be used instead of the rephasers from Table 7.2.

- **(a)** Attach a syringe, the NW31 variant from Figure 7.5(c), and two welded Snarks to the input of this conduit so that its two input gliders are produced by just a single input glider to the syringe.
- **(b)** How can you adjust the semi-Snarks so that this circuit produces an output glider with a different mod-8 timing? Use this technique to build circuits that produce the output glider exactly 1, 6, or 7 generations slower than the output glider shown in part (a).
- **(c)** Create an additional four circuits that produce output gliders of the four remaining mod-8 timings by replacing one or both of the semi-Snarks in the circuits from part (b) with the following color-preserving **semi-cenark**:[43]



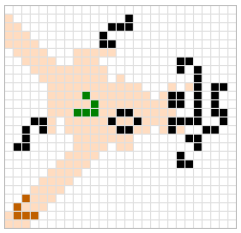
7.21 The color-preserving glider rephasers from Exercise 7.20 can be made color-changing in (at least) three different ways.

- **(a)** [2/5] Method 1: Replace the Fx77 conduit in one of those glider rephasers by the Herschel conduit from Exercise 7.1(c), thus creating a color-changing rephaser.
  [Side note: Doing this in all 8 CP rephasers would create a complete set of CC rephasers for all mod-8 timings.]
- **(b)** [3/5] Method 2: the CP semi-Snarks and/or semi-cenarks in these conduits can be replaced by a CC semi-Snark and/or the following CC semi-cenark:[44]



  Which mod-8 timings of color-changing rephasers can be reached in this way? Why are we more limited, and unable to reach all 8 possible timings?
- **(c)** [2/5] Method 3: Which component from this chapter could be added before or after one of the CP rephasers to turn it into a CC rephaser?

7.22 The following pattern is called a **tremi-Snark**,[45] since it only produces a single output glider for every three input gliders that it receives.



- **(a)** [1/5] What is its repeat time?
- **(b)** [1/5] Is it color-changing or color-preserving?
- **(c)** [3/5] Use tremi-Snarks to make a glider gun with period 3^{50}.

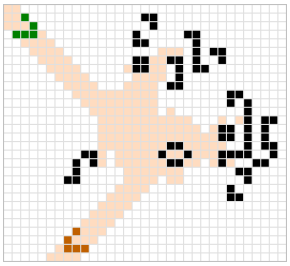
[43] Found by Tanner Jacobi in November 2017, and named for the fact that it converts each pair of incoming gliders into a six-cell pattern called a **century**, which is then converted into the output glider.

[44] Also found by Tanner Jacobi in November 2017.

[45] Found by Tanner Jacobi in September 2017.

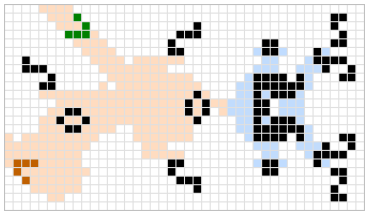
Free PDF. See conwaylife.com/book for pattern files and print version.

7.23 The following pattern is called a **quadri-Snark**,[46] since it only produces a single output glider for every four input gliders that it receives.



(a) [1/5] What is its repeat time?
(b) [2/5] The quadri-Snark is color-preserving. Use conduits that we have seen to construct a similar stable circuit for reflecting one out of every four input gliders, but which is color-changing.
(c) [3/5] Use quadri-Snarks to make a glider gun with period 2^{100} that fits in a smaller bounding box than the one from Figure 7.19(b).

7.24 The following pattern is called a **quinti-Snark**,[47] since it only produces a single output glider for every five input gliders that it receives.



(a) [1/5] Is it color-changing or color-preserving?
(b) [1/5] What is its repeat time?
(c) [2/5] The quinti-Snark is not stable—it contains a p5 heavyweight volcano as one of its pieces. Explain why, in spite of this, it can be used with any glider stream of period at least its repeat time (not just glider streams whose period is a multiple of 5).

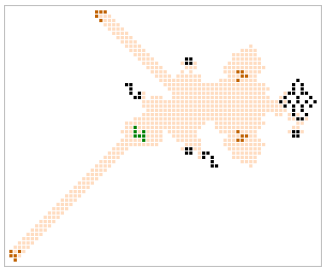
7.25 [2/5] The semi-Snark in the mechanism from Figure 7.22 serves to increase the period of that gun by 616 generations. Explain why it is there—why can we not remove it and rearrange the other components so as to create a gun with period $1\,703 - 616 = 1\,087$?

7.26 [1/5] Show how the LWSS-to-glider converter from Table 7.3 can also function as an MWSS-to-glider converter.

∗**7.27** [1/5] Use two copies of the B60 Herschel conduit to make a glider gun (the resulting gun is called the **Simkin glider gun**).[48] What is its period, and how could you determine that period based only on the name of this conduit?

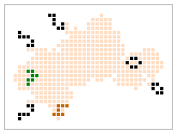
7.28 [1/5] Use two copies of the RFx36R conduit to make an oscillator. What is its period, and how could you determine that period based only on the name of this conduit?

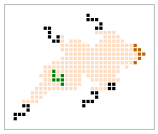
7.29 [2/5] The conduit displayed below converts a Herschel into 4 gliders, all traveling in different directions. Break this conduit down into elementary conduits (i.e., conduits that convert named objects into each other and cannot be broken down any further).



[Hint: We saw one of these conduits in Table 7.3—which one?]

∗**7.30** [2/5] The conduit below on the left is called BR146H and the one on the right is a variant of the HFx58B conduit from Table 7.3.





(a) Explain why the original HFx58B conduit cannot be used as an input to BR146H.
(b) Attach the variant of HFx58B to both the input and output ends of BR146H. What is the name of this composite conduit?

7.31 [2/5] String together conduits that we have seen so as to create a stable conduit that converts...

(a) a glider into a pi-heptomino,
(b) a glider into an R-pentomino,
(c) a lightweight spaceship into a Herschel, and
(d) a middleweight spaceship into a B-heptomino.

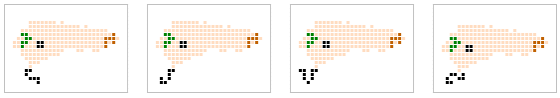
[46] Found by Tanner Jacobi in October 2017.

[47] Found by (you guessed it) Tanner Jacobi in October 2018. Some other periodic quinti-Snarks are also known, but this is the most useful one for the reason explained in part (c) of this exercise.

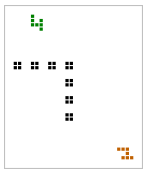
[48] It is named for discoverer, Michael Simkin, who found it in April 2015.

Free PDF. See conwaylife.com/book for pattern files and print version.

***7.32** [2/5] Here are some variants of the BFx59H conduit from Table 7.3:

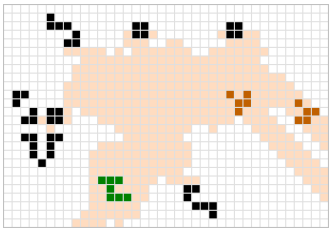


Use the conduits HFx58B, BLx19R, RF28B, and BFx59H (including their variants, if necessary—see Exercise 7.30 for a HFx58B variant) to move the Herschel highlighted below in green to the position marked in orange $58+19+28+59=164$ generations later, while avoiding the blocks.



***7.33** [2/5] The L156 conduit from Figure 7.1(b) is made up of 3 conduits that convert the input Herschel into other well-known named objects before converting it back into a Herschel. What are the intermediate objects that the Herschel is converted into, and what are the names of those 3 conduits?

7.34 [3/5] An H-to-G6 conduit that can be used in place of the H-to-G5 conduit at the top-left corner of Figure 7.39 is presented below.[49]

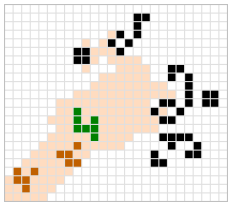


Use two copies of this conduit, together with two copies of the Herschel receiver from the bottom-right corner of Figure 7.39, to create a glider gun. Place 6 signals (i.e., Herschels and/or tandem glider pairs) on the track and separate the components so that the gun has period 127.

7.35 Recall the highway robber from Figure 7.33.

- **(a)** [1/5] Calculate its repeat time.
- **(b)** [4/5] With the help of a Bandersnatch, rebuild that highway robber so as to have a smaller repeat time.

7.36 An H-to-G3 conduit called **SW1T43** is displayed below.[50]



- **(a)** [2/5] The tandem glider pair that is produced by this conduit is exactly the same as is required in one of the "tee" syntheses from Table 5.2. Which one?
- **(b)** [3/5] Use this conduit to create a **double-barrelled gun** that repeatedly fires this tandem glider pair.
- **(c)** [2/5] Fire the output of another glider gun at the output of your gun from part (b) so that the gliders collide in the tee formation from part (a), thus producing a perpendicular glider.

7.37 Recall that Herschel tracks can be used to create glider guns with periods as low as 62.

- **(a)** [2/5] Create a square Herschel track oscillator that uses four R64 conduits (one at each of its corners) and 16 Fx77 conduits (four on each of its sides). If you place just a single Herschel on this track, it should have period 1488.
- **(b)** [3/5] Place 24 Herschels on this track, creating an oscillator with period $1488/24=62$.
  [Hint: You may have to fiddle with eaters a bit to be able to pack the Herschels together this closely.]
- **(c)** [1/5] What goes wrong if you try to turn this Herschel track into a period 62 glider gun?
- **(d)** [3/5] Create a period 62 glider gun by modifying this Herschel loop so that it uses L156 conduits for its corners instead of R64 conduits.

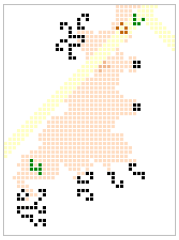
***7.38** [2/5] Recall the period 80 adjustable true period glider gun from Figure 7.17.

- **(a)** Adjust the gun to have period 97.
- **(b)** Adjust the gun to have period 78.
- **(c)** Try to adjust the gun to have period 76 or 77 and explain why it does not work.
- **(d)** Adjust the gun to have period 74 or 75.
- **(e)** Try to adjust the gun to have period 73. Explain two independent reasons why it does not work.

[49] This conduit was found by Matthias Merzenich in December 2011.
[50] This conduit was found by Simon Ekström in October 2015.

Free PDF. See conwaylife.com/book for pattern files and print version.

7.39 A boat factory that behaves much like the demultiplexer from Figure 7.30(b), in that a glider can detect the presence of its boat, is displayed below.

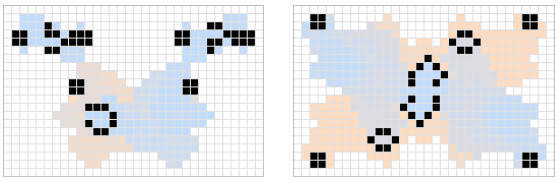


The main advantage of this factory over the demultiplexer is that it does not self-destruct if a second input Herschel is received before the boat's presence is tested. Instead, a second input Herschel simply toggles the boat back off.

- **(a)** [1/5] Find a Herschel conduit from Table 7.1 that can be prepended to this boat factory. Make sure that the test glider's lane coming from the northeast remains transparent (i.e., unobstructed).
  [Hint: One of the eaters in this boat factory is very recognizable.]
- **(b)** [2/5] Attach Snarks and a syringe to one of the glider output lanes from part (a) so as to create a composite semi-Snark. It will have a fairly slow recovery time, somewhere around 650 generations.
  [Hint: To make the syringe fit, you will have to move an eater out of the way.]
- **(c)** [3/5] Attach stable conduits to one of the glider output lanes from part (a) so that if the test glider detects and destroys the boat, it subsequently rebuilds it.
  [Side note: This gives a new type of logic circuit with the property that if two gliders follow each other closely then one glider passes through unharmed (since the other has not yet rebuilt the boat), but if the gliders are separated by a lot them they are both blocked.]
- **(d)** [3/5] Adjust the circuit from part (c) so that if a pair of input gliders has a separation of 400 generations then one of them passes through the circuit, but if they have a separation of 800 generations then they are both blocked.

7.40 [1/5] Create a period 90 glider gun by combining a Gosper glider gun, a syringe, and the second-from-the-right Herschel-to-block factory from Figure 7.31.

7.41 [3/5] The sparky oscillators below are called **Wainwright's p72** (or sometimes **two blockers hassling R-pentomino**) and **Achim's p144**.[51]



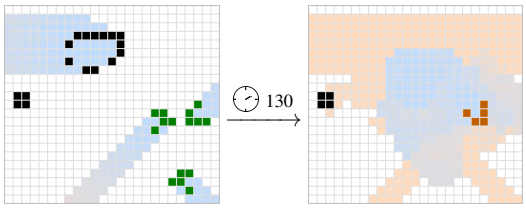
- **(a)** Removing the block from one of the corners of Achim's p144 turns it into a block factory. Place Wainwright's p72 near the block that the factory produces so that its sparks delete that block and create a glider (and thus a p144 glider gun).[52]
- **(b)** Construct a slightly smaller p144 glider gun by using Rich's p16 instead of two blockers hassling R-pentomino.[53]

7.42 [2/5] The block that is created by the center-left Herschel-to-block factory from Figure 7.31 can be used to cleanly destroy (and be destroyed by) a glider coming from the northwest or from the southeast.

- **(a)** How many adjacent lanes can a glider coming from the northwest be placed on so as to cleanly destroy the block?
- **(b)** How many adjacent lanes can a glider coming from the southeast be placed on so as to cleanly destroy the block? Why is this answer different from that of part (a)?

∗**7.43** [2/5] Identify each of the stable conduits used in the stable Heisenburp of Figure 7.36.
[Hint: We saw most of them in the main text of this chapter. However, one of them was introduced earlier in this chapter's exercises, and another one was introduced in Chapter 3's exercises.]

7.44 [4/5] In the reaction below, three gliders synthesize a block on a ship, and a passing HWSS then turns that pseudo still life into a glider.



Add stable circuitry that turns the output glider into the three (synchronized) input gliders, thus creating a stable HWSS Heisenburp.

[51] Found by Robert Wainwright in 1990 and Achim Flammenkamp in 1994, respectively.
[52] First constructed by Bill Gosper in July 1994.
[53] First constructed by Chris Cain in 2016.

Free PDF. See conwaylife.com/book for pattern files and print version.

7.45 [3/5] Find a way of appending just a single periodic reflector to the conduit from Figure 7.37 (instead of three stable conduits, as in the Callahan G-to-H) so that the output Herschel's first natural glider destroys the junk beehive. What is the repeat time of the resulting periodic G-to-H conduit?

∗**7.46** [2/5] Modify just a single conduit in the Silver reflector from Figure 3.29(a) so that it produces an output glider in all four directions.

Free PDF. See conwaylife.com/book for pattern files and print version.



8. Guns and Glider Streams

> Assembling large arrays of guns and puffers—other than getting the timing and positioning right—is about [as] difficult as assembling large objects out of Lego bricks.
>
> Mark D. Niemiec

We have seen that being able to efficiently create and manipulate streams of gliders is extremely important when constructing complex objects in Life since we can use collisions between those gliders to synthesize other objects. However, so far we have only seen how to create glider streams with a few different spacings: the period 30 stream produced by the Gosper glider gun (Section 1.3), the period 46 stream produced by the twin bees gun (Section 1.4), and the streams of period 62 and higher than we can construct via the Herschel tracks of Sections 3.6 and 7.1. In this chapter, we look at how to construct glider guns that create gliders of any spacing and almost any positioning that we like.

8.1 Glider Deletion

The first technique that we present for constructing guns of different periods is one that erases every second glider in a stream of period at least 29,[1] thus doubling its period (the semi-Snark of Section 7.6 can similarly delete every second glider in a glider stream, but it only works at periods 48 and greater). The way this technique works is to fire two streams of the same period at each other in such a way that they synthesize a block,[2] and that block then destroys the next glider in one of the streams. A particular orientation of glider streams that works is displayed in Figure 8.1—one of the glider streams is erased completely (half of its gliders are destroyed in the glider-to-block collision, and the other half are destroyed by the subsequent block), while the other glider stream has half of its gliders destroyed.

We can use this reaction to straightforwardly double the period of any glider gun with period 29 or higher, which we demonstrate in Figure 8.2 with the period 30 Gosper glider gun and the period 46

[1] There is a similar reaction that can be used to double glider streams of period at least 28—see Exercise 8.1.

[2] In particular, this glider collision is the fifth of the block-producing collisions listed in Table 5.1.

Free PDF. See conwaylife.com/book for pattern files and print versio

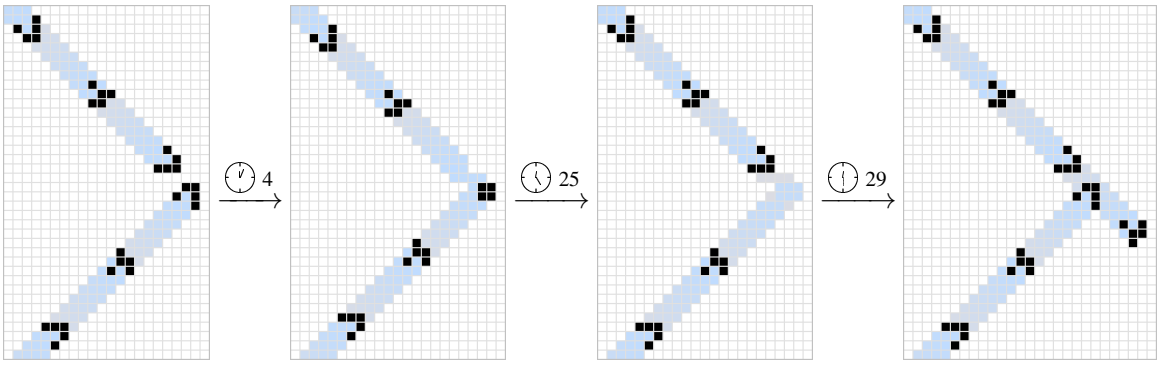
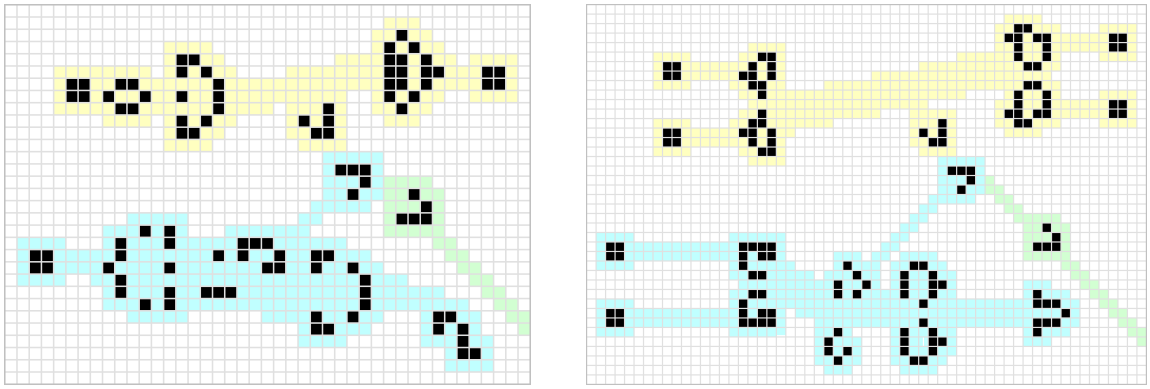


Figure 8.1: If we shoot two glider streams of period at least 29 at each other in the proper phase, one of the streams is destroyed completely while the other stream is thinned out by a factor of 2. This works because the gliders collide to form a block, and that block then destroys the next glider in the bottom stream, letting the next glider from the top stream pass by unharmed.

twin bees gun.[3]

By iterating this procedure, we can repeatedly double the period of a glider stream, thus multiplying the period of any glider stream (with period at least 29) by any power of two. However, what if we want to multiply the period by another number, such as 3 or 5? Well, we could simply try firing glider streams at each other in different ways and see if any of them do the trick—after all, we saw in Section 5.1 that there are only 71 ways for two glider streams to hit each other, so it won't take too long to check them for useful combinations. And indeed, it turns out that a period-tripling configuration exists for sufficiently slow even-period glider streams,[4] as demonstrated in Figure 8.3.



(a) A period 60 gun that uses two Gosper glider guns.

(b) A period 92 gun that uses two twin bees guns.

Figure 8.2: We can use two copies of any gun with period at least 29 to create a gun with double the period. This is demonstrated in (a) with two period 30 Gosper glider guns creating a period 60 gun, and (b) two period 46 twin bees guns creating a period 92 gun. In each case, the individual guns are outlined in yellow, gliders that collide to create a block are outlined in aqua, and gliders that survive to create the new, thinner glider stream travelling to the southeast are outlined in green.

The way that this period tripler works is largely the same as it was for the period-doubling collision from Figure 8.1, only slightly more complicated:

[3]The gun in Figure 8.2(a) is not quite the smallest known period 60 gun by bounding box area—see catagolue.hatsya.com/object/gun_60/b3s23. Similarly, a slightly smaller period 92 gun can be constructed by colliding a twin bees shuttle with Tanner's p46.

[4]The collision we describe here is the third one listed in the "misc" section of Table 5.1.

Free PDF. See conwaylife.com/book for pattern files and print version.

- When the first gliders in the streams collide, they leave behind a pair of blocks.
- When the next gliders in the streams hit the blocks, they are turned into a single blinker.
- That blinker destroys the third glider in one of the streams, and the third glider in the other stream passes by unharmed.

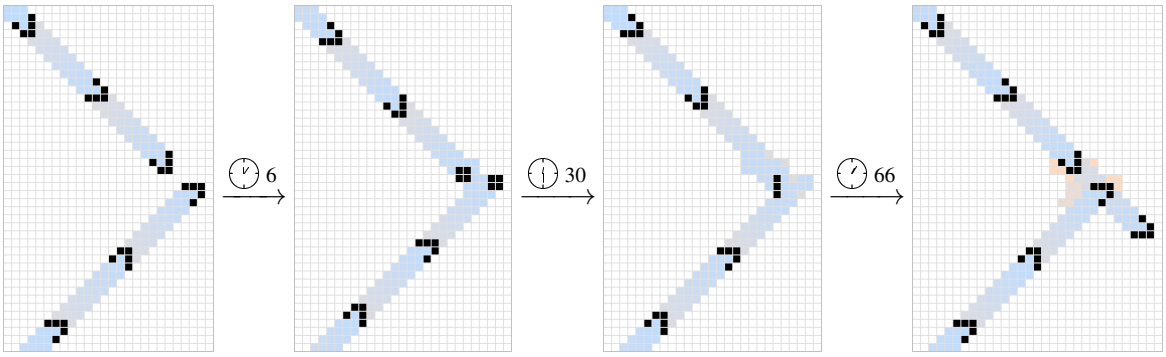


Figure 8.3: If we shoot two even-period glider streams of period at least 34 at each other in the proper phase, one of the streams is destroyed completely while the other stream is thinned out by a factor of 3.

Firing two glider streams at each other in this way has the effect of completely destroying one of the streams while destroying only two-thirds of the gliders in the other. This reaction is demonstrated in Figure 8.4, where we use two period 46 twin bees guns to create a period $3 \times 46 = 138$ gun (this reaction only works for glider streams of period 34 or higher, so we cannot apply it to the Gosper glider gun). Also, just like before, we can use this reaction repeatedly with more and more copies of the original gun, each time multiplying the gun's period by 3.

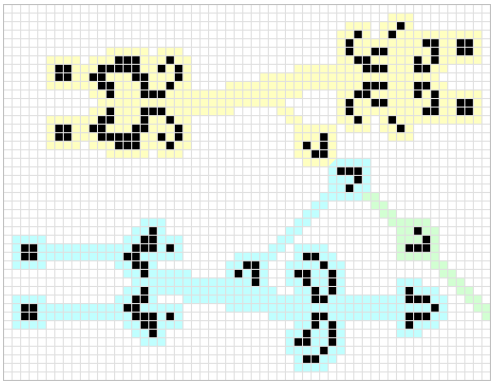


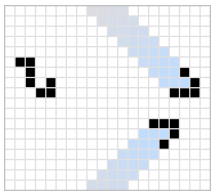
Figure 8.4: We can use two copies of any gun with period at least 34 to create a gun with triple the period. Here, two period 46 twin bees guns are aimed at each other so as to create a period 138 gun.

By combining these two period-multiplying reactions together, we can multiply a gun's period by any number whose prime factorization contains only 2 or 3, with at least one factor of 2. We can thus use this technique to thin out a glider stream by a factor of 4, 6, or 8, for example. We can multiply a gun's period by 9 only if the gun's period is even; otherwise a glider will strike the blinker that makes up one of the stages of the tripling reaction, at the wrong phase.[5]

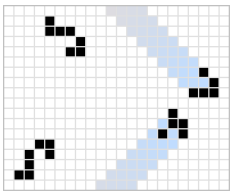
To fill in some of the gaps in the above list of possible multipliers, we present in Figure 8.5 some methods for thinning out a stream by a factor of 5 or 7.

[5] We can use the stable circuitry techniques from Section 7.6 to thin out a glider stream by any factor of our choosing, but these collision-based techniques are a bit smaller and easier to use, at least for small factors.

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) A collision that thins out even-period glider streams with period at least 108 by a factor of 5.

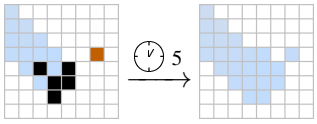


(b) A collision that thins out even-period glider streams with period at least 46 by a factor of 7.

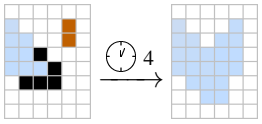
Figure 8.5: Some additional methods of multiplying the period of glider streams by small prime numbers.

8.1.1 Filters

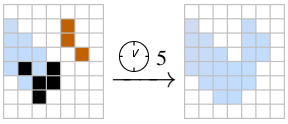
Another way of deleting some gliders within a stream is to use a **filter**, which is an oscillator that pulsates or gives off a spark in such a way as to destroy some of the gliders in the stream, but not all of them (similar to how we used a Schick engine to erase some of the gliders released by a space rake in Section 4.4.2). Gliders can be destroyed by essentially any type of spark (see Figure 8.6), so we can use many high-period sparkers to destroy gliders and thin out glider streams.



(a) A dot spark deleting a glider.



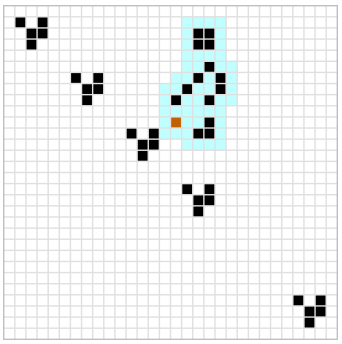
(b) A domino spark deleting a glider.



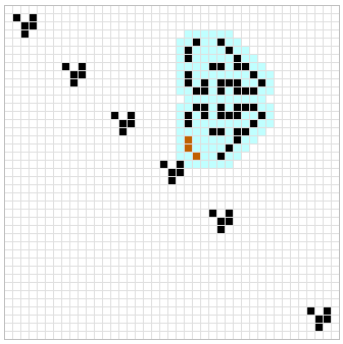
(c) A banana spark deleting a glider.

Figure 8.6: Many different sparks can be used to delete gliders.

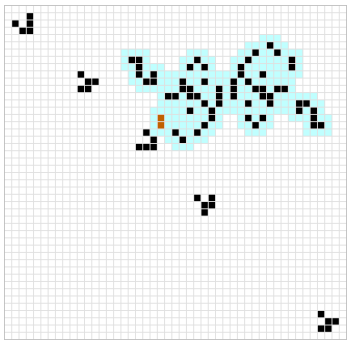
For example, if we place a period 8 blocker next to a glider stream of period $8n+4$ ($n \geq 2$), it destroys every second glider in the stream, thus doubling its period. Similarly, the period 16 oscillator "Rich's p16" from Figure 3.42 emits a banana spark that can be used to double the period of any glider stream with period $16n+8$ ($n \geq 1$),[6] and the period 22 oscillator from Figure 3.7(c) emits a domino spark that can be used to double the period of any glider stream with period $22n+11$ ($n \geq 1$). These period-doubling filters are illustrated for streams of periods 20, 24, and 33 in Figure 8.7.



(a) The dot spark from a blocker can be used to destroy every other glider in a stream of period 20 (or $8n+4$ for any $n \geq 2$).



(b) The banana spark from Rich's p16 can be used to destroy every other glider in a stream of period 24 (or $16n+8$ for any $n \geq 1$).



(c) A domino spark from Jason's p22 can be used to destroy every other glider in a stream of period 33 (or $22n+11$ for any $n \geq 1$).

Figure 8.7: Some filters (highlighted in aqua) that can be used to double the period of certain glider streams.

[6] Rob's p16 works just as well for this purpose—see Exercise 8.42.

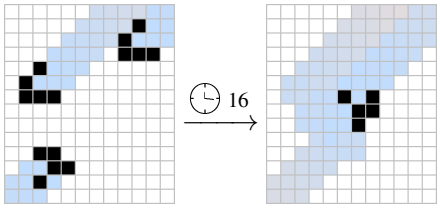
Free PDF. See conwaylife.com/book for pattern files and print version.

For a filter oscillator to work correctly, it's generally necessary that the period of the oscillator shares some factor (greater than one) with the period of the glider stream being thinned. Otherwise, all possible combinations of oscillator spark and glider position will eventually appear in the evolution of the pattern, and some of those combinations will usually cause a chaotic explosion instead of a clean glider deletion.

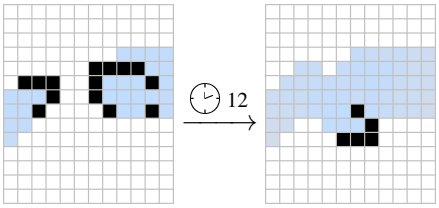
8.2 Glider Insertion

Our next technique is one that is complementary to that of the previous section: it lets us thicken glider streams (i.e., it lets us decrease their period) rather than thin them. The main idea behind this trick is to collide two or more spaceships in such a way as to produce a single glider that fits in a stream's gap. Unfortunately, this technique in its simplest form is very rarely useful, since there are only two 2-glider collisions that result in a single glider (refer back to Table 5.1), one of which results in a glider going in the *same* direction as one of the input gliders, and the other of which results in a glider going in the *opposite* direction of one of the input gliders. In particular, neither of these glider-producing 2-glider collisions results in a glider travelling *perpendicular* to both input gliders, which is what we really want.

Fortunately, there are a few other ways of colliding spaceships together to get around this problem. The first such method is to use one more glider: the 3-glider collisions called **tees** that we introduced back in Table 5.2 produces an output glider that is perpendicular to all 3 input gliders (see Figure 8.8(a)). Another solution to this problem is to instead collide a glider with a lightweight spaceship as in Figure 8.8(b), which has the advantage that an LWSS never travels in the same direction as a glider, so we typically have a wider selection of possible orientations to make the collision happen without interfering with other gliders in the stream.



(a) A **tee**: a collision of 3 gliders that results in a single glider perpendicular to the input gliders.



(b) A collision between a lightweight spaceship and a glider that rotates the glider by 90 degrees.

Figure 8.8: Two methods of changing the direction of a glider so that it can be inserted into another glider stream. These techniques help us create glider guns with a wide variety of periods, and are also extremely useful for creating guns that make use of syntheses with tightly spaced gliders.

These collisions can be used to insert additional gliders into an already-existing glider stream, thus decreasing its period. For example, the glider gun displayed in Figure 8.9 uses the LWSS–glider collision to insert additional gliders into a period 46 glider stream coming from a twin bees gun, thus converting it into a period 23 gun (compare with the period 23 gun that we saw back in Figure 6.24(a)).

Unfortunately, the tee from Figure 8.8(a) cannot be used to produce glider streams with period smaller than 25, and the LWSS–glider collision from Figure 8.8(b) is similarly limited to periods of at least 22; the temporary debris created by these collisions interferes with the rest of the glider stream at lower periods. So for example, these reactions cannot be used to transform multiple Gosper glider guns into a period 15 gun, despite the fact that period 15 glider streams are indeed possible.

One way of constructing guns for even tighter glider streams (all the way down to period 14, which is the smallest possible) is to instead use the reaction from Figure 8.10(a), in which two sparks transform a lightweight spaceship into a glider with essentially no debris occurring outside of that

Free PDF. See conwaylife.com/book for pattern files and print version.

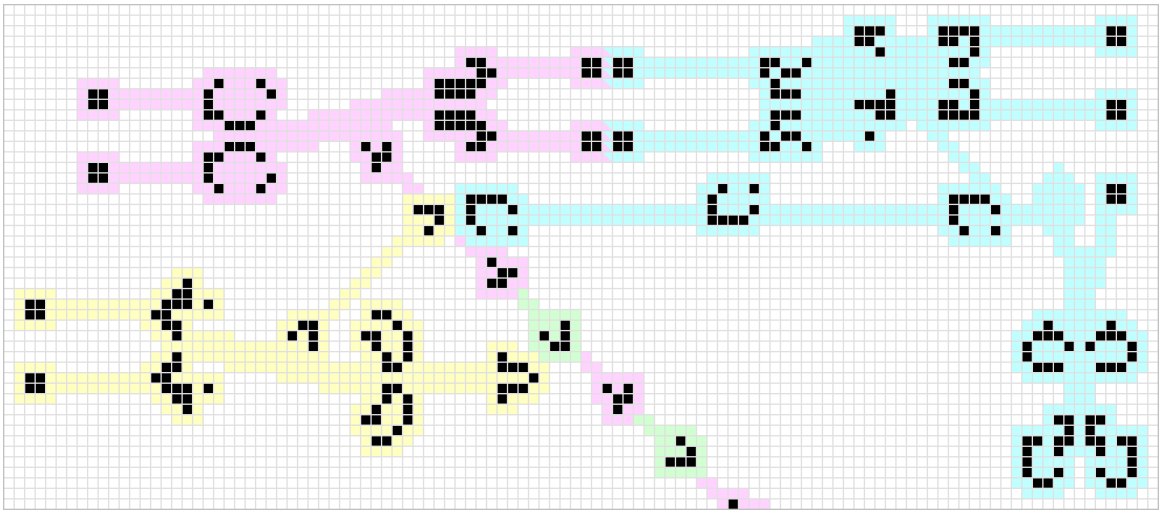


Figure 8.9: A period 23 glider gun constructed from several period 46 glider guns. The top-left glider gun in magenta fires a period 46 stream of gliders. To interlace this stream with another period 46 stream of gliders (outlined in green), we use the LWSS–glider collision of Figure 8.8(b): the glider in that collision is created by the glider gun outlined in yellow, while the LWSS is created by the LWSS gun (which we introduced in Figure 6.24(b)) outlined in aqua.

transformation.[7] The first of these sparks (which can be of essentially any type, but is typically a dot, a finger, or a domino coming from a pipsquirter) starts destroying the lightweight spaceship and can be provided by a wide variety of oscillators. The second spark (which must be a domino or duoplet) then transforms the dying LWSS into the desired glider. However, that second spark is a bit tricker to get into place, as it must be provided directly adjacent to the glider stream, so it is typically provided by another dying object like an LWSS or a Herschel. Some methods of providing these two sparks are illustrated in Figure 8.10(b–f).

To illustrate the utility of this reaction, we can now construct a period 15 glider gun by making use of several copies of the p30 Gosper glider gun, much like we constructed a period 23 glider gun out of many copies of the p46 twin bees gun in Figure 8.9. In particular, since the reaction of Figure 8.10(f) works so naturally at period 15, we make use of it via 7 Gosper glider guns: 1 to create the initial p30 glider stream, and 3 for each of the two lightweight spaceships that we smash together between a pentadecathlon and middleweight volcano. The completed p15 gun is displayed in Figure 8.11.

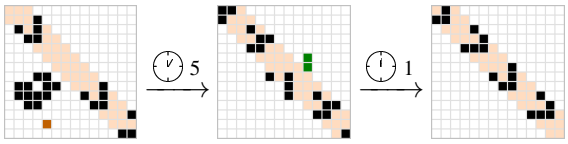
8.3 Streams of Other Spaceships

Since most ways of implementing the glider insertion reaction of Figure 8.10 rely so heavily on the use of lightweight spaceships, we now briefly discuss how to efficiently create and manipulate lightweight (as well as middleweight and heavyweight) spaceships. We of course can create streams of any of these xWSSes simply via the various 3-glider syntheses that we saw in Chapter 5, but we are going to focus on reactions that let us save space over this naïve method.

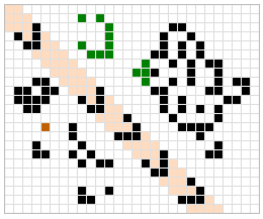
One particularly cheap way of creating lightweight and middleweight spaceships is the conduit displayed in Figure 8.12, which converts a glider and a Herschel into whichever of the other two spaceships is desired (the only difference in these two reactions is the relative timing between the input Herschel and glider). Since the Herschel that is used in that collision creates a glider anyway, and that glider can simply be reflected back so as to collide with the next Herschel, this conduit can

[7] This reaction was found by Dietrich Leithner no later than 1994.

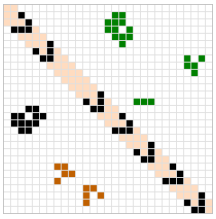
Free PDF. See conwaylife.com/book for pattern files and print version.



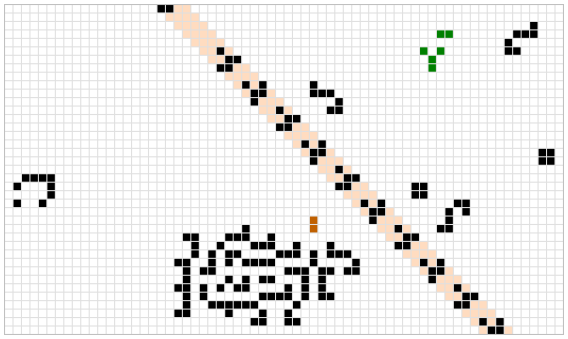
(a) A clean glider insertion using an LWSS and two sparks.



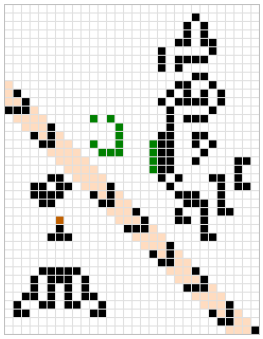
(b) p6 insertion via unix and pipsquirter.



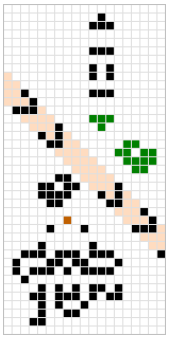
(c) Insertion that works at any period.



(d) Insertion via a pipquirter and a Herschel.



(e) p4 insertion.



(f) p15 insertion.

Figure 8.10: (a) A dot spark (in orange), followed by a domino spark 5 generations later (in green) can be used to transform a lightweight spaceship into a glider so cleanly that it can be used to fill in gaps in p14 glider streams—the tightest streams possible. The first spark can be provided by (b,d,e,f) a wide variety of different oscillators or (c) a two-glider collision. The second spark can be provided either via (b,e,f) an LWSS being destroyed by an oscillator, (c) a blinker + glider + LWSS collision, or (d) a Herschel conduit.

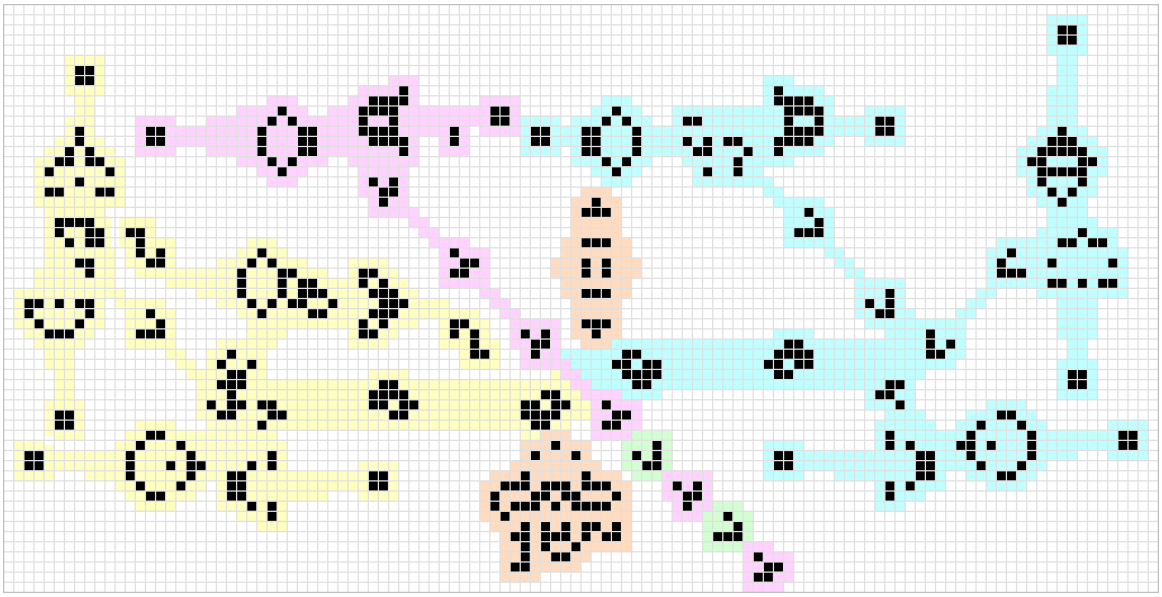
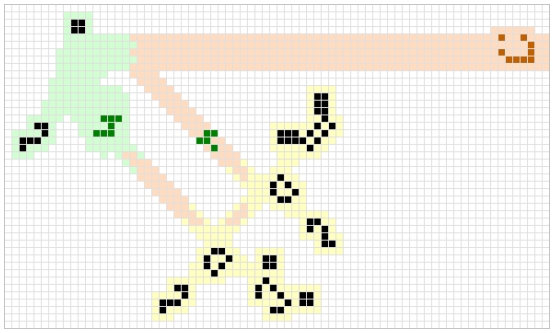


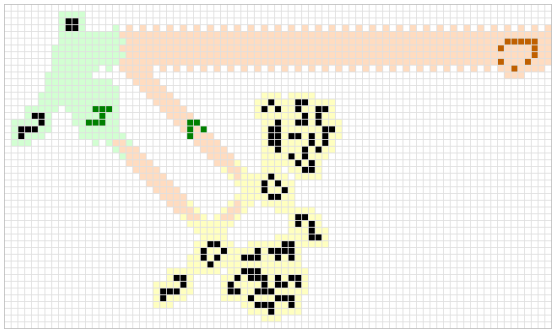
Figure 8.11: A period 15 glider gun constructed from several period 30 Gosper glider guns. The top-left glider gun in magenta fires a period 30 stream of gliders. To interlace this stream with another period 30 stream of gliders (outlined in green), we use the LWSS–LWSS–pentadecathlon–middleweight volcano collision of Figure 8.10(f).

Free PDF. See conwaylife.com/book for pattern files and print version.

turn a Herschel stream into an LWSS stream or an MWSS stream reasonably directly.[8] However, the tight spacing of the required reflectors is such that this technique works best for streams whose period is a multiple of 3 or 4, since p3, p4, and p6 bumpers are some of the only reflectors that fit so close together.



(a) A period 156 Herschel stream being converted into an LWSS stream.



(b) A period 171 Herschel stream being converted into an MWSS stream.

Figure 8.12: A conduit (highlighted in green) that can convert a Herschel and glider collision into either (a) an LWSS or (b) an MWSS. By reflecting the first natural glider of the input Herschel (via the bumpers highlighted in yellow), this conduit can convert a Herschel stream into either an LWSS stream or an MWSS stream.

To create a reasonably small lightweight or middleweight spaceship gun of a large period, we can now simply attach this mechanism to the output of any sufficiently high-period glider gun of our choosing, with a syringe in between (as well as other Herschel conduits, as needed, for spacing reasons). For example, the gun in Figure 8.13(a) uses this technique to fire a period 84 stream of lightweight spaceships.

However, since this mechanism uses a stream of Herschels as input, we can often construct even smaller LWSS guns by feeding Herschels into it directly from a Herschel track, rather than feeding them in via a glider gun and a syringe. A slightly smaller period 84 LWSS gun that uses this technique is displayed in Figure 8.13(b).

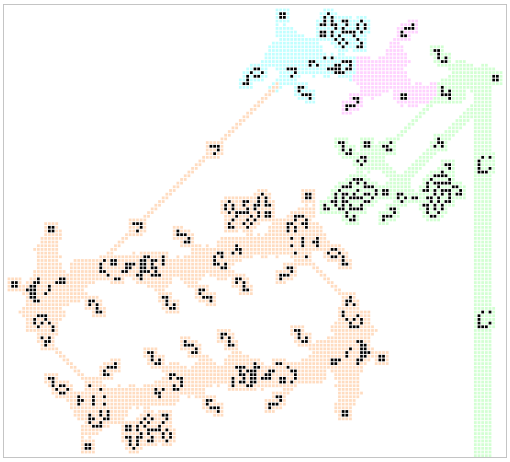
We can also delete lightweight spaceships from a stream in much the same ways that we do with gliders. For example, a spark of pretty much any type can destroy a lightweight spaceship, so we can use sparky oscillators to filter (i.e., thin out) LWSS streams. Figure 8.14 illustrates some examples of how this technique can be used to double, or even triple, streams of certain periods. Another method of doubling the period of an LWSS stream, which has the advantage of working at any sufficiently large period, was presented in Exercise 6.9.

Similarly, if we want to *decrease* the period of an xWSS stream, we can perform insertions just like we did with gliders. One possible method of inserting an LWSS into a stream is to make use of one of the three-glider syntheses from Table 5.2. However, even the fastest and cleanest of these syntheses can only decrease the period of an LWSS stream down to 23. Since the minimum possible periods of LWSS, MWSS, and HWSS streams are 14, 16, and 18, respectively, we would like a cleaner reaction that can produce these even tighter streams. One particularly tight reaction of this type is displayed in Figure 8.15. It uses a whopping 10 gliders to carefully place a lightweight spaceship between two others that are 36 generations apart, resulting in a p18 LWSS stream.

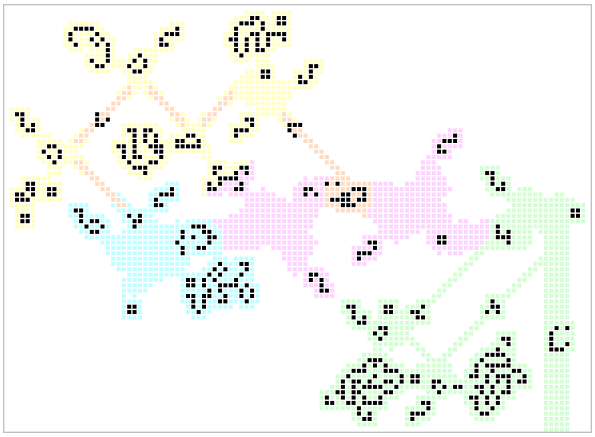
Fortunately, we do not actually need to come up with new insertion mechanisms for streams of

[8] This reaction only works with *streams* of Herschels, rather than individual Herschels, since each Herschel must collide with the first natural glider of a Herschel that came before it. The period of that stream can be modified straightforwardly by adjusting the reflectors—see Exercise 8.11.

Free PDF. See conwaylife.com/book for pattern files and print version.



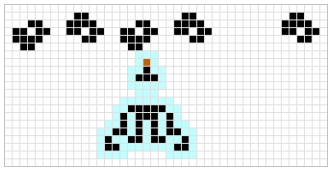
(a) A p84 version of the adjustable glider gun from Figure 7.17 (highlighted in orange) whose output is converted into lightweight spaceships.



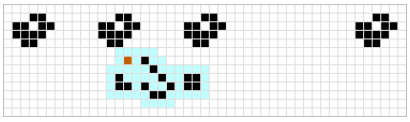
(b) A p84 Herschel loop whose output is converted into LWSSes. The loop is sustained by the first natural glider of a Herschel being reflected (by the reflectors highlighted in yellow) back into a syringe.

Figure 8.13: Two p84 lightweight spaceship guns that make use of the Herschel-to-LWSS mechanism of Figure 8.12(a) (highlighted in green). Syringes are highlighted in aqua and Fx77 Herschel conduits (highlighted in magenta) are used for spacing reasons.

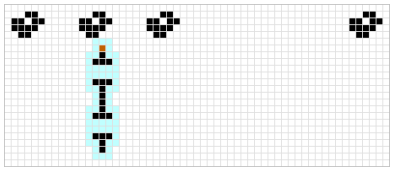
middleweight or heavyweight spaceships, since there are numerous ways of colliding spaceships with the back end of an LWSS so as to "upgrade" it to an MWSS, and there are similarly numerous ways of "upgrading" an MWSS to an HWSS. Some methods that work even with tightly packed period 18 xWSS streams are displayed in Figure 8.16.



(a) The finger spark from a T-nosed p4 can be used to destroy every other LWSS in a stream of period $4n+2$ for any $n \geq 3$.



(b) The dot spark from a blocker can be used to destroy every other LWSS in a stream of period $8n+4$ for any $n \geq 2$.



(c) The thumb and domino sparks from a pentadecathlon can be used to destroy two out of every three LWSSes in a stream of period $15n+5$ or $15n+10$ for any $n \geq 1$.

Figure 8.14: Some filters (highlighted in aqua) that can be used to double or triple the period of certain LWSS streams.

Just like glider insertion can be used to create glider guns with small periods, LWSS insertion can be used to create xWSS guns with small periods. We leave the explicit construction of such guns to Exercise 8.8 though, as we have carried out this type of task several times already at this point (and we will do it again for glider guns in the next section). Even though LWSS and MWSS streams can be packed to have periods as small as 14 and 16, respectively, the particular reactions that we introduced here only work down to period 18.[9]

[9] A tighter LWSS-insertion mechanism was found in January 2022 by Martin Grant, allowing for the construction of p16 LWSS and MWSS guns. No p14 or p15 LWSS guns are known yet.

Free PDF. See conwaylife.com/book for pattern files and print version.

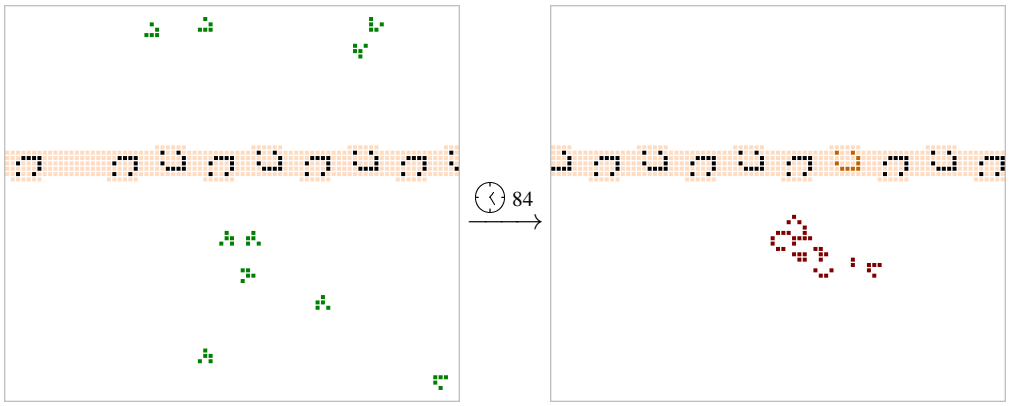
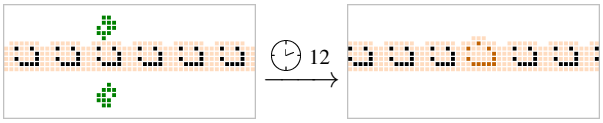
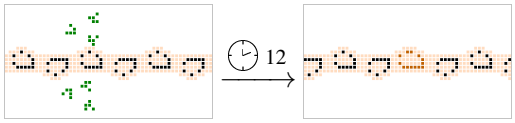


Figure 8.15: A 10-glider collision that can be used to insert an LWSS into streams with period as low as 36, resulting in LWSS streams with period as low as 18. The cells displayed in red die off in another 72 generations. Found by Chris Cain in October 2018.



(a) Two lightweight spaceships can be used to convert an LWSS into an MWSS.



(b) Six gliders can be used to convert an MWSS into an HWSS.

Figure 8.16: Some ways of firing lightweight spaceships or gliders at LWSS and MWSS streams so as to "upgrade" them to MWSS and HWSS streams. These reactions are clean enough to work at the minimum possible periods (16 for the LWSS → MWSS conversion and 18 for the MWSS → HWSS conversion).

8.4 Glider Guns of Any Period

We have already seen a few methods of constructing glider guns of any suitably large period of our choosing—we can use Herschel tracks as described in Sections 3.6 and 7.1 to create guns of any period 62 or greater, the adjustable-period glider gun of Figure 7.17 to create guns of any period 80 or greater, and the period multipliers of Section 7.6 to create guns with extraordinarily large periods. However, none of those methods can create a glider gun with very *small* period, like 14.

By making use of the glider insertion reaction of Figure 8.10, we can now fill in this gap in our knowledge and construct glider guns of all periods 14 and greater—we just create a gun whose period is a suitably large multiple of the period that we actually want, and then repeatedly use multiple copies of this gun and the glider insertion reaction to fill in the gaps in its glider stream. For example, to create a period 14 gun, we could use Herschel tracks to construct a gun of period $14 \times 5 = 70$ to create a p70 glider stream, and then use multiple copies of that gun to do glider insertion 4 times between each pair of gliders in that stream.[10] However, the resulting p14 gun would be quite large—recall that creating a p15 gun required *seven* p30 guns in Figure 8.11, and that was just to perform a single glider insertion.

In order to construct a somewhat smaller p14 glider gun using these same ideas, we instead use a period $14 \times 6 = 84$ stream as our starting point, so that we can make use of the syringe in our circuitry (recall that its repeat time is 78 generations). It is straightforward to construct a period 84 glider gun using the adjustable-period gun of Figure 7.17, for example,[11] but if we are slightly more clever, then we can actually build a gun that places *two* of the six gliders that we need to bring a p84 stream down

[10] The first period 14 glider gun was built by Dietrich Leithner in November 1994 using these same ideas (and the same glider insertion reaction).

[11] In fact, we already did this in Figure 8.13.

Free PDF. See conwaylife.com/book for pattern files and print version.

to p14. In particular, if we use the NE30T3 H-to-2G edge-shooting conduit from Figure 7.7(b), we can reflect one of its output gliders so as to be on the same lane as the other one. Figure 8.17 illustrates how to use this technique to construct a p84 gun that emits 2 out of every 3 gliders in a p28 stream (or equivalently, 2 out of every 6 gliders in a p14 stream).[12]

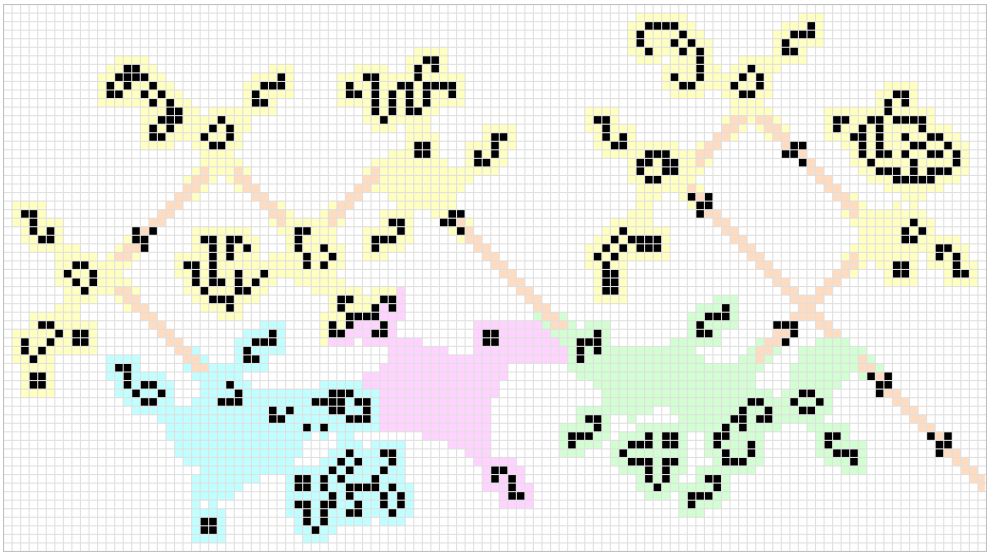


Figure 8.17: A p84 gun that emits 2 out of 3 gliders in a p28 stream. The central Herschel is fed through NE30T3 (highlighted in green) so as to create two output gliders, one of which is reflected (via standard reflectors like bumpers, bouncers, and Snarks, highlighted in yellow) into the same lane as the other one. The first natural glider of the central Herschel is similarly reflected and then injected into a syringe (highlighted in aqua) so as to recreate itself (just as in the gun from Figure 8.13(b)).

By making use of these reactions, we can create the period 14 glider gun displayed in Figure 8.18. This particular gun is not optimized at all—by squeezing its components together more carefully, it could be made roughly 50% smaller (when measuring size according to bounding box area). The smallest known p14 gun[13] is roughly half as tall, wide, and populous as ours, but uses basically the same mechanisms that we used here—they are just packed together somewhat more cleverly and rely on three period 42 guns instead of five period 84 guns.

8.5 True-Period Guns

While we now know how to construct glider guns of every period greater than or equal to 14, many of them are somewhat artificial—they oscillate at a higher period than (and necessarily a multiple of) the stream that they produce. For example, the gun of Figure 8.11 actually oscillates at period 30, despite producing a glider stream of period 15, and the gun of Figure 8.18 oscillates at period 84, despite producing a glider stream of period 14. Such guns are called **pseudo-period guns**, and the new low-period guns that we introduced in this chapter are all of this type.

There is nothing inherently wrong with pseudo-period guns, but it seems natural to ask whether or not we can construct guns of arbitrary periods that actually oscillate at the same period as their stream (i.e., guns that do not make use of techniques like glider insertion to decrease the period of the stream that they produce). We call such guns **true-period guns**, and examples include all of the guns that we saw prior to this chapter, such as the p30 Gosper glider gun and the p46 twin bees gun.

[12] By rearranging the reflectors in this gun slightly, its period can be adjusted relatively straightforwardly—see Exercise 8.10.

[13] See catagolue.hatsya.com/object/gun_14/b3s23.

Free PDF. See conwaylife.com/book for pattern files and print version.

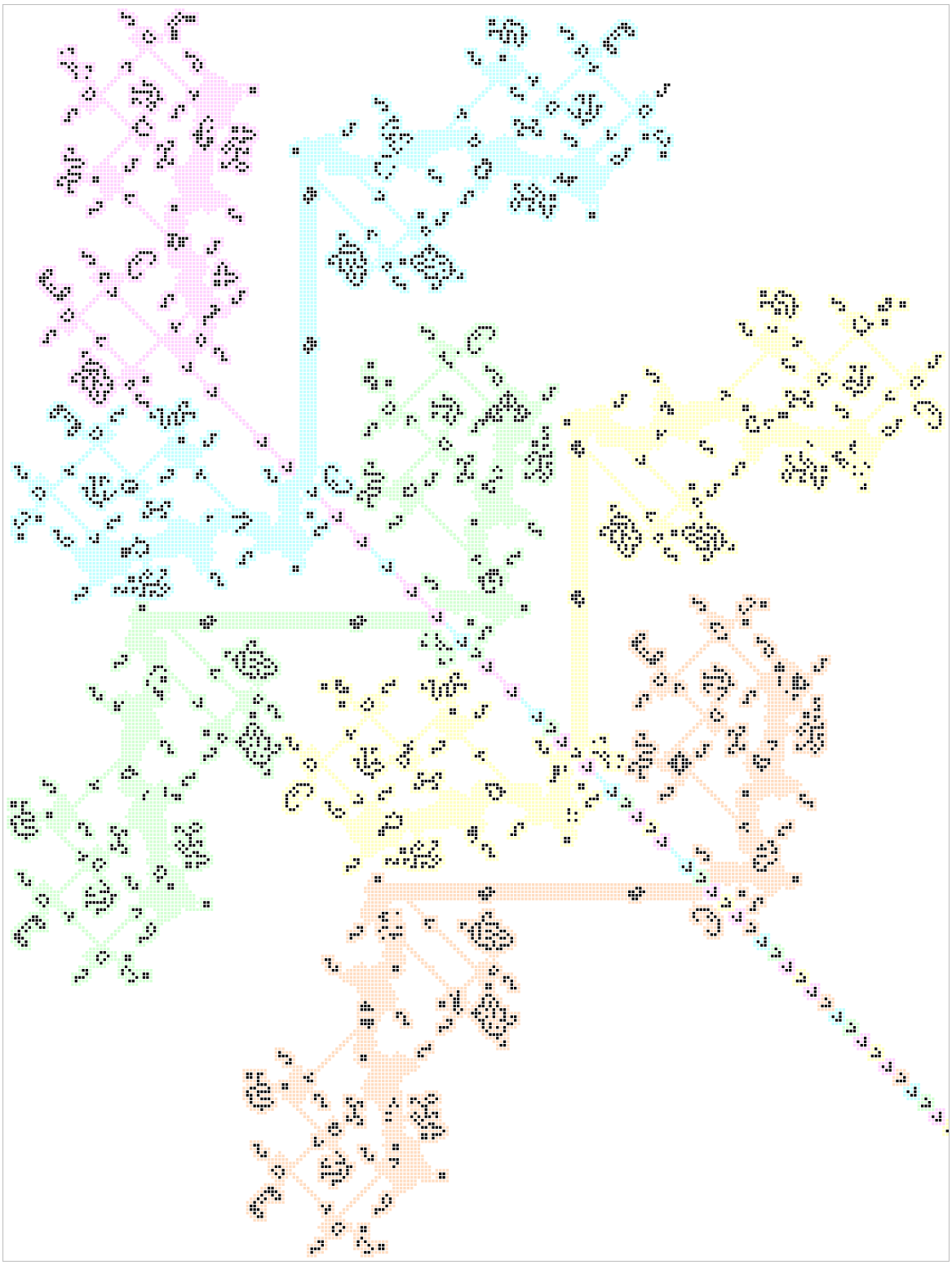


Figure 8.18: A period 14 glider gun that uses the p84 two-glider gun from Figure 8.17 (highlighted in magenta), as well as four copies of a p84 glider insertion mechanism (highlighted in aqua, green, yellow, and orange) to reduce the period of the stream down to $84/6 = 14$. The glider insertion mechanisms works via the reaction from Figure 8.10(d), which is fed by the p84 LWSS gun of Figure 8.13(b) and a Herschel that is generated by another copy of the same p84 Herschel loop.

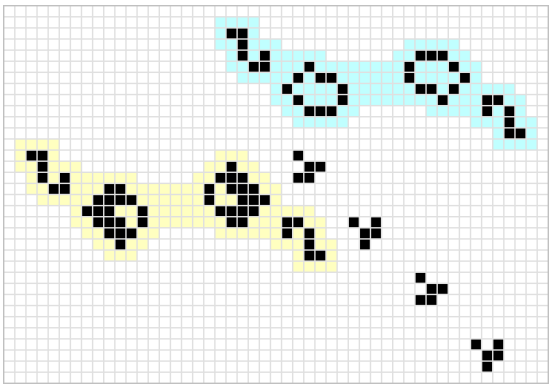
Free PDF. See conwaylife.com/book for pattern files and print version.

In general, using glider insertion to decrease the period of a gun's stream results in a pseudo-period gun, but using filters or the glider deletion methods of Section 8.1 to *increase* the period of its glider stream results in a true-period gun (e.g., the p60 gun from Figure 8.2(a) that uses two Gosper glider guns is a true-period gun). Constructing true-period guns with small period[14] is much more difficult than the same task for pseudo-period guns, since we cannot just manipulate gliders themselves to get the period we desire, but rather must actually develop new glider-generating mechanisms.

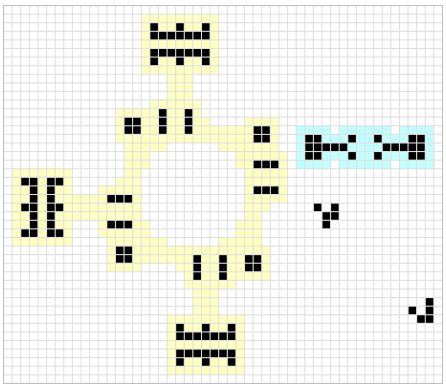
There are just a few methods that are known for creating true-period glider guns, and it is typically difficult to get them to work for small periods. A brief summary of the known true-period guns of these few small periods is presented in Table 8.1, and the next few subsections illustrate and describe their construction.[15]

8.5.1 Spark Collisions: True-Period 22, 30, 33, 36, 45, and 46 Guns

Most of the simplest true-period glider guns work simply by having sparks from different oscillators collide with each other in such a way as to create a glider. In fact, the very first guns to be discovered—the p30 Gosper glider gun[16] and the p46 twin bees gun—are of this type. The next simplest true-period glider guns have periods 22 and 45, and are similarly of this type.



(a) A true-period 22 glider gun that consists of two copies of Jason's p22 oscillator (highlighted in yellow and aqua).



(b) A true-period 45 glider gun that consists of a custom p45 oscillator and a pentadecathlon (highlighted in yellow and aqua, respectively).

Figure 8.19: Two true-period glider guns. The (a) period 22 gun was found by David Eppstein later on the same day that Jason Summers found the oscillator (August 23, 2000), and the (b) period 45 gun was found by Matthias Merzenich in April 2010, with subsequent improvements by Adam P. Goucher, Dave Greene, and Tanner Jacobi.

The true-period 45 gun displayed in Figure 8.19(b) works by colliding the sparks of a custom period 45 oscillator with the edge of a pentadecathlon, so that their interacting sparks create a glider. Similarly, the true-period 22 gun displayed in Figure 8.19(a) works by positioning two copies of the very sparky Jason's p22 oscillator (refer back to Figure 3.7(c)) next to each other. Unfortunately, no filter is capable of doubling the period of this gun (see Exercise 8.6) and none of the other glider deletion techniques that we discussed earlier work at period 22, so it is not clear whether or not this

[14] Herschel tracks can be used to create true-period guns of large period (period 62 and larger, in particular).

[15] It probably will not be long before this list of true-period guns becomes out of date (7 new periods were discovered while this book was being written!), so a perhaps more up-to-date list of known true-period gun periods can be found at `conwaylife.com/wiki/Gun` and `conwaylife.com/wiki/LifeWiki:Game_of_Life_Status_page#Glider_gun_true-periods`.

[16] Technically, the Gosper glider gun does not work by colliding *sparks*, since the colliding debris would form a beehive if left uninterrupted, rather than dying. Nonetheless, the idea behind it is the same.

Free PDF. See conwaylife.com/book for pattern files and print version.

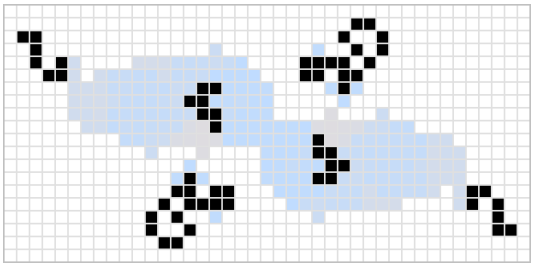
Period	Year	Original Discoverer	Example and Notes
20	2013	Matthias Merzenich Noam Elkies	• upcoming Figure 8.22(b) • smaller ones now known
21, 42	2021	Tanner Jacobi David Raucci	Exercise 8.13
22	2000	David Eppstein Jason Summers	upcoming Figure 8.19(a)
24, 48	1997	Noam Elkies	upcoming Figure 8.22(a)
28	2020	Matthias Merzenich Paul Callahan	upcoming Figure 8.23
30	1970	Bill Gosper	Gosper glider gun (Figure 1.18)
32	2021	Matthias Merzenich David Raucci	• smallest known is in the upcoming Figure 8.24 • first known was found just one month earlier
33	2018	Arie Paap Matthias Merzenich	upcoming Figure 8.21(b)
36	2004	Jason Summers	upcoming Figure 8.20(b)
40	2013	Adam P. Goucher Matthias Merzenich Jason Summers	• a small one (Figure 8.22(b)) uses a blocker to filter the p20 gun • first known (not displayed) was found 3 months earlier
44	1992	David Buckingham	improved a bit in 1997 by Paul Callahan (Figure 8.25)
45	2010	Matthias Merzenich	upcoming Figure 8.19(b)
46	1971	Bill Gosper	twin bees gun (Figure 1.23)
49	2021	Tanner Jacobi "hotcrystal0" "wwei23"	• see Exercise 8.17 • the p51 gun uses the same base reaction
50	1996	Dean Hickerson Noam Elkies	• Figure 8.26 shows nearly the smallest known • first known (not displayed) used Exercise 3.25
51	2021	Luka Okanishi	see Exercise 8.17
52	2018	Dave Greene Chris Cain Adam P. Goucher Matthias Merzenich	• first known (not displayed) was about $800 \times 1\,000$ • smaller one in Figure 8.32(a) was constructed two months later with help from ConwayLife.com forums user "Entity Valkyrie"
54	1998	Dietrich Leithner	• smallest p54 and p55 guns built by David Raucci
55	1998	Stephen Silver	• medium p54–56 guns are in Figure 8.32(c–e)
56	1998	Dietrich Leithner	• first known p54–56 guns (not displayed) were huge
57	2016	Matthias Merzenich	• first known (not displayed) was large (about 330×350) • a smaller one is in Figure 8.32(b) • an even smaller one was built in April 2021
58	2016	Maia Thunkies Matthias Merzenich	upcoming Figure 8.36
59	2009	Adam P. Goucher Jason Summers	• first known (not displayed) was huge ($\sim 4\,000 \times 3\,000$) • smallest known is in upcoming Figure 8.28
60	1970	Bill Gosper	two Gosper glider guns (Figure 8.2(a))
61	2016	Luka Okanishi	upcoming Figure 8.34
62+	1996	David Buckingham	Herschel tracks (some periods known earlier)

Table 8.1: A summary of when the first true-period glider gun of each known period was first discovered. The first discoverer credited above put the gun itself together, while other contributors who provided important reactions or ideas used in the construction are listed alphabetically by their last name.

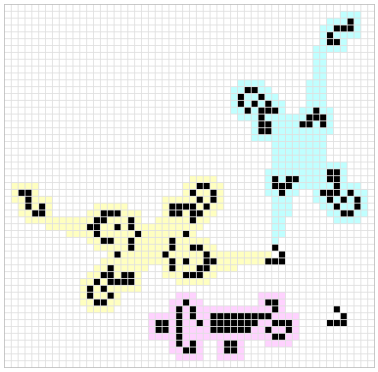
Free PDF. See conwaylife.com/book for pattern files and print version.

gun can be easily modified to create a true-period 44 gun. However, we will see a different method of making a true-period 44 gun shortly, in Section 8.5.2.

There are also glider guns of periods 33 and 36 that are mostly of this type—they work by colliding together large sparks from two copies of very sparky oscillators of periods 33 and 36, respectively. However, these guns also require a few extra stabilizing components to clean up the glider-producing reaction. For example, the period 36 oscillator in Figure 8.20(a) can be paired up with itself to create the true-period 36 gun in Figure 8.20(b), but it requires an extra heavyweight emulator to clean up a block that is left behind by the sparks that would otherwise destroy the gun between the first and second glider that it creates.



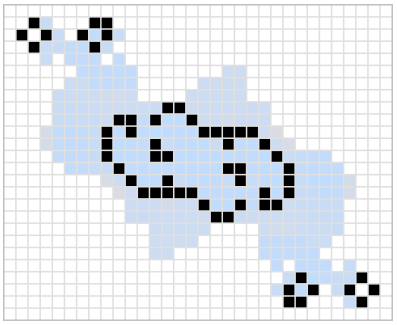
(a) A period 36 oscillator called **Jason's p36** that works by using jams and eater 1s to hassle two B-heptominoes. It was found by Jason Summers in July 2004.



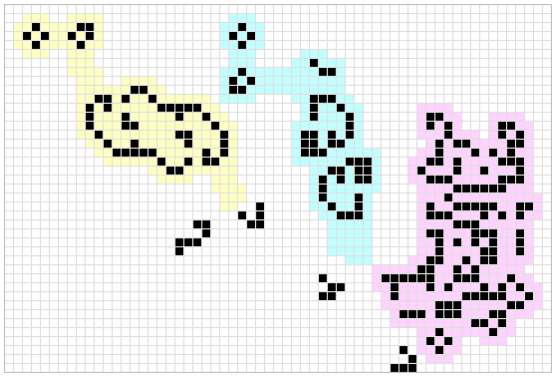
(b) A true-period 36 gun.

Figure 8.20: A (b) true-period 36 gun (found by Jason Summers in July 2004, with improvements by Adam P. Goucher and Scot Ellison) that works by colliding two copies of (a) Jason's p36 (highlighted in aqua and yellow). The heavyweight emulator (highlighted in magenta) is needed to clean up the reaction so that the gun doesn't destroy itself after creating its first glider.

Similarly, the period 33 oscillator in Figure 8.21(a) can be paired with itself, once we remove the boat and tub that stabilize it on one end, to create the true-period 33 gun that is displayed in Figure 8.21(b). However, an extra eater 1 is needed to help produce the output glider, and a custom period 3 oscillator is needed to clean up the reaction so that the gun does not collide with its own glider stream, resulting in self-destruction shortly after the creation of its first glider.



(a) A period 33 oscillator called **Jason's p33**. Found by Jason Summers in August 2000 and shrunk somewhat by Matthias Merzenich in 2013.



(b) A true-period 33 gun.

Figure 8.21: A (b) true-period 33 gun (found by Arie Paap and Matthias Merzenich in October 2018) that works by colliding two copies of (a) Jason's p33 (highlighted in aqua and yellow). The eater 1 cleans up the reaction so as to actually create the output glider, while the custom period 3 oscillator (highlighted in magenta) is needed to clean up the reaction so that the gun doesn't destroy itself after creating its first glider.

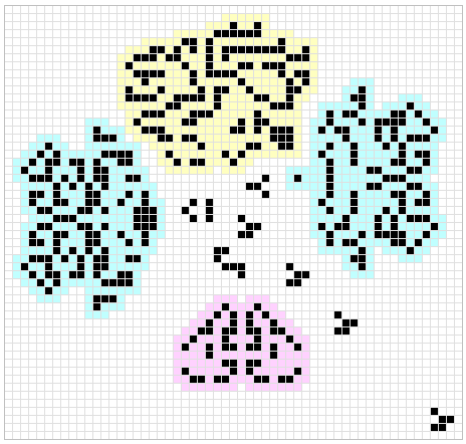
Free PDF. See conwaylife.com/book for pattern files and print version.

8.5.2 Hasslers: True-Period 20, 21, 24, 28, 32, 40, 42, 44, 48–51, 54–56, and 59 Guns

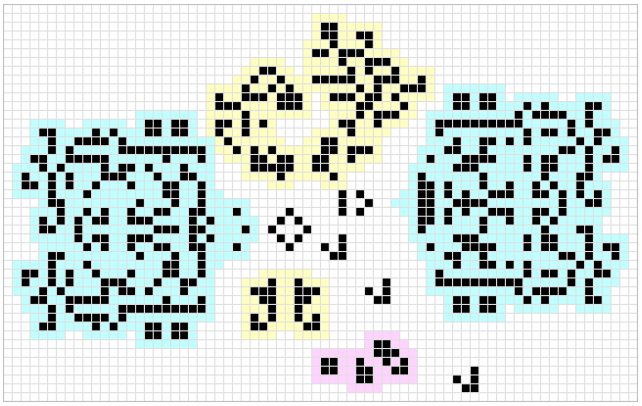
Quite a few true-period guns work by hassling a commonly occurring unstable object like a T-tetromino, pi-heptomino, B-heptomino, or pre-honey farm, much like the oscillators that we saw back in Section 3.4. The difference here is that the hassling reaction has to be chaotic enough that a glider can be extracted from the debris that it produces.

For example, we can construct true-period 20, 24, 40, and 48 glider guns by using the T-tetromino hassling reactions that we introduced back in Figure 3.23. In particular, notice that the period 24 oscillator from Figure 3.23(b) emits a large and somewhat long-lasting spark near its bottom-right corner—if we could hit this large spark with another spark just right, it seems believable that a glider might be formed.[17] There is indeed a (somewhat complicated) combination of sparks that works to collide two T-tetrominoes together to give us what we want, and the resulting true-period 24 glider gun is displayed in Figure 8.22(a). Since we can use Rich's p16 to double the period of period 24 glider streams as in Figure 8.7(b), this also gives us a true-period 48 glider gun almost for free.

By using the same T-tetromino collision, together with the p20 hassling reaction from Figure 3.23(a), we also get a true-period 20 glider gun for not too much extra effort. The biggest hurdle to overcome when constructing the p20 gun is the fact that there is so much going on in such a small space that it is difficult to generate the necessary sparks without the sparkers overlapping with each other. One configuration that works is displayed in Figure 8.22(b).[18] Furthermore, we can use a blocker to filter this p20 stream, as in Figure 8.7(a), to quickly and easily turn this into a true-period 40 gun.



(a) A true-period 24 (and 48) glider gun, based on the p24 oscillator from Figure 3.23(b). Superfountains (see Figure 3.13(d)) are highlighted in aqua and a custom p4 sparker is highlighted in yellow. Rich's p16 (highlighted in magenta) is used to filter the p24 stream into a p48 one, as in Figure 8.7(b).



(b) A true-period 20 (and 40) glider gun. Middleweight supervolcanoes (see Figure 3.13(e)) are highlighted in aqua and other oscillators (the bottom of which is a fumarole and the top of which is a custom p4 domino and thumb sparker) are highlighted in yellow. A blocker (highlighted in magenta) is used to filter the p20 stream into a p40 one, as in Figure 8.7(a).

Figure 8.22: True-period (a) 24, 48, (b) 20, and 40 glider guns. They were found by (a) Noam Elkies in 1997 (with subsequent improvements to decrease its size by Karel Suhajda) and (b) Matthias Merzenich in 2013 (also with improvements by Karel Suhajda).

The only known true-period 28 glider gun is also based on hassling a T-tetromino, but this one

[17] This line of thinking led Bill Gosper to guess that a gun would be made from his p24 T-tetromino oscillator within about 24 hours. It turned out to be trickier than he expected though, and actually took a bit longer than 2 years to complete.

[18] This is the smallest period for which a true-period glider gun is currently known.

Free PDF. See conwaylife.com/book for pattern files and print version.

uses components that we have not yet seen: a massive p7 ultrafountain,[19] as well as two eater 1s and another custom still life, which cause a single T-tetromino to release a glider. This gun is displayed in Figure 8.23, and it can be turned into a true-period 56 gun by adding a filter to it (see Exercise 8.14).

Pre-honey farm hasslers are similarly the only known method of constructing true-period 32 glider guns. The reaction at the core of the p32 oscillator from Figure 3.18(d) was used to construct the first such gun, a smaller version of which is displayed in Figure 8.24. Pre-honey farm hasslers were also used to create the smallest-known true-period 54 and 55 guns,[20] but they are not displayed here (we use another method to construct somewhat larger guns of these periods in Section 8.5.3 instead).

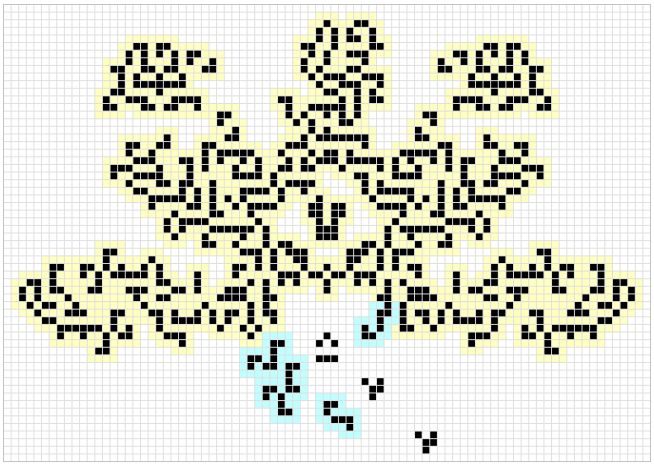


Figure 8.23: A true-period 28 glider gun that works by using a p7 ultrafountain (highlighted in yellow) to hassle a T-tetromino so that it releases a glider via a custom conduit (highlighted in aqua). Constructed by Matthias Merzenich in December 2020.

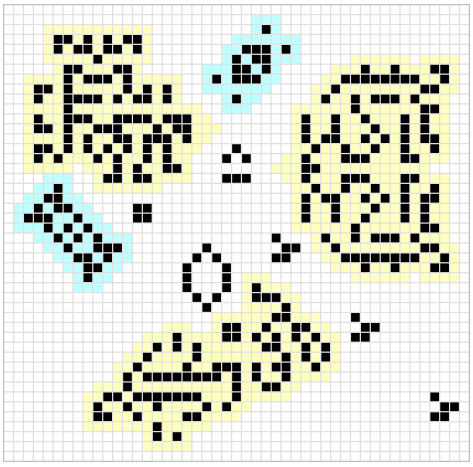


Figure 8.24: A true-period 32 glider gun that works by using some p4 and p8 oscillators (highlighted in yellow and aqua, respectively) to hassle a pre-honey farm and a T-tetromino. Built by Martin Grant in October 2021.

The only known true-period 21 and 42 glider guns are based on B-heptomino hasslers. However, the details of their construction are reasonably similar to the other guns that we have already explored in this section, so we leave them to Exercise 8.13.

Finally, the period 44 pi-heptomino hassler from Exercise 3.15 is the basis of several true-period 44, 50, and 59 glider guns. For example, the true-period 44 glider gun works simply by replacing one of its blocks (which are used to suppress the debris from the pi-heptominoes) on the side of the oscillator with a custom conduit that converts that debris into a glider. This conduit consists of a block and two eater 1s,[21] and the resulting gun is displayed in Figure 8.25.[22]

By swapping out the heavyweight emulators on the sides of this gun for an arrangement of two blocks and a p10 oscillator called **Bullet's p10**,[23] its period increases to 50 generations. The resulting true-period 50 glider gun is displayed in Figure 8.26.[24]

This pi-heptomino hassling reaction can be extended to period 59 by carefully hitting it with

[19] Just like a fountain or volcano releases its spark one row of dead cells away from the oscillator itself, and a superfountain or supervolcano releases its spark two rows of dead cells away, an *ultra*fountain provides three rows of dead cells of clearance between their spark and the oscillator itself.

[20] Constructed in September 2021 by David Raucci. See conwaylife.com/wiki/Period-54_glider_gun and conwaylife.com/wiki/Period-55_glider_gun.

[21] The third eater 1 is not part of the conduit, but just replaces one of the other stabilizing blocks for spacing reasons.

[22] The first true-period 44 glider gun was found by David Buckingham in 1992. He found this smaller version in 1996.

[23] Found by ConwayLife.com forums user "Bullet51" in December 2018.

[24] Constructed by Tanner Jacobi in March 2019. Its size was subsequently reduced slightly by Matthias Merzenich.

Free PDF. See conwaylife.com/book for pattern files and print version.

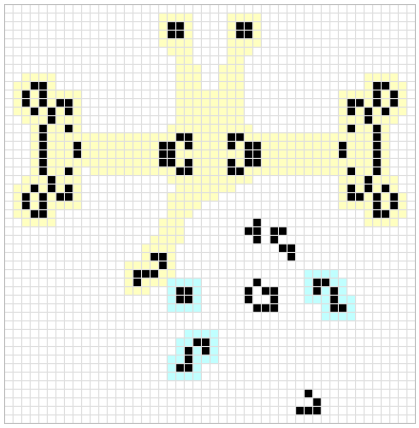


Figure 8.25: A true-period 44 glider gun that works by funnelling some of the debris from the p44 pi-heptomino hassler (highlighted in yellow) through a custom conduit (highlighted in aqua) so as to transform it into a glider.

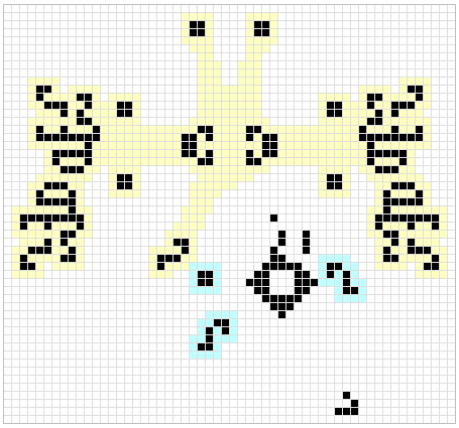


Figure 8.26: A true-period 50 glider gun that uses the same pair of pi-heptominoes and glider-extraction conduit as in the p44 gun from Figure 8.25, but stabilized by **Bullet's p10** instead of heavyweight emulators.

gliders. In particular, the reaction displayed in Figure 8.27 features two copies of the pi-heptomino reaction and converts 5 input gliders into 6 output gliders every 59 generations (two of the output gliders are created via the same conduit as in the period 44 and 50 guns, and the other four gliders are produced at the center where the two copies of the pi-heptomino reaction meet each other).[25]

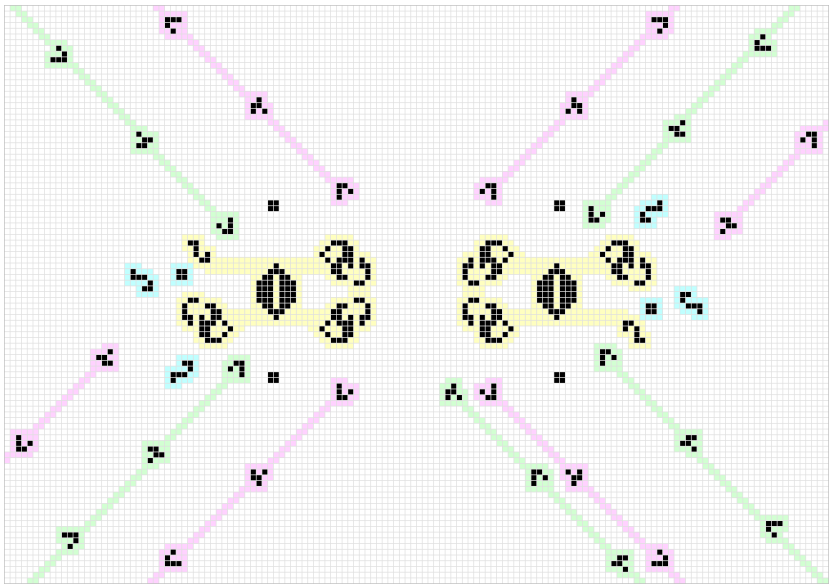


Figure 8.27: A period 59 reaction that takes in 5 gliders (highlighted in green) and spits out 6 gliders (highlighted in magenta). The reaction works by using two copies of the pi-heptomino reaction (highlighted in yellow) to create two gliders via a conduit (highlighted in aqua) and four more gliders when the debris from these reactions collide with each other at the center.

The reason that this reaction can be used to create a gun is that it produces more gliders than it destroys—we simply need to redirect 5 of the output gliders back into the appropriate input positions so as to continuously fuel the reaction, leaving the 6th output glider as the output of the gun. The

[25] This reaction was found by Jason Summers in December 2009.

Free PDF. See conwaylife.com/book for pattern files and print version.

smallest-known true-period 59 glider gun that can be constructed via this method is presented in Figure 8.28, which uses numerous (sometimes welded) Snarks to reflect most of the output gliders from this reaction right back into itself, thus "fueling" the creation of the single excess glider.[26]

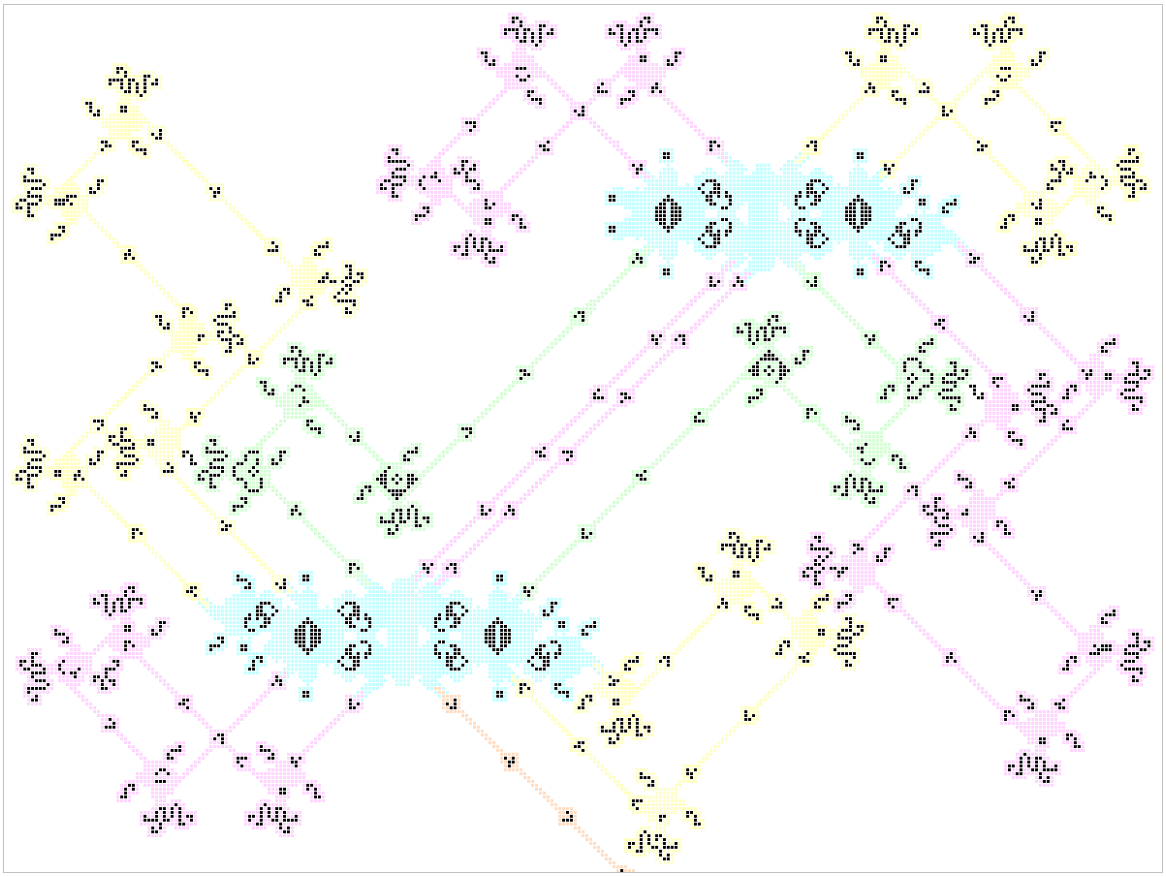


Figure 8.28: A true-period 59 glider gun that was constructed by ConwayLife.com forum user "Entity Valkyrie" in October 2018. It works by using two copies of the 5-to-6 glider reaction from Figure 8.27 (outlined in aqua) and several Snarks to bounce the output gliders of those reactions back into themselves (the tracks that the gliders follow are highlighted in yellow, green, and magenta).

It is worth noting that the top half of this gun is almost identical to its bottom half, and only exists to take care of the two troublesome glider streams that are positioned so close to each other near the center of the gun. Snarks (and all other known reflectors that work at period 59) are too large to reflect one of those glider streams without interfering with the other one, so instead a second copy of the reaction is used so that those two tight glider streams feed each other's reactions.

Another reaction that similarly produces more output gliders than input gliders is provided in Exercise 8.17—it can be used to create true-period glider guns of period 49 and 51.

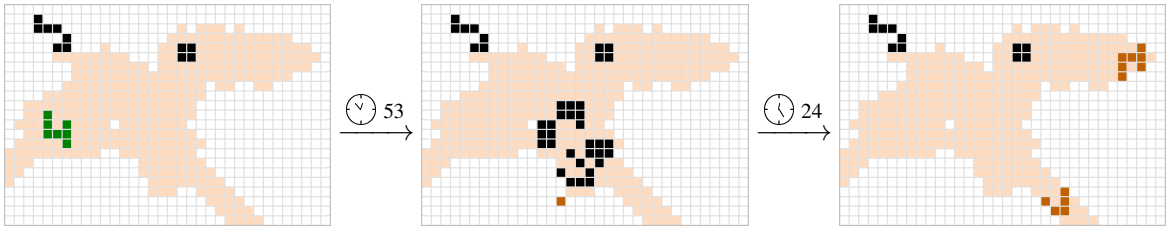
[26] The first true-period 59 gun used the same reaction and was built by Adam P. Goucher in December 2009. Since the Snark had not yet been discovered, it used huge reflectors based on Herschel tracks instead, and had more than 300 000 live cells.

Free PDF. See conwaylife.com/book for pattern files and print version.

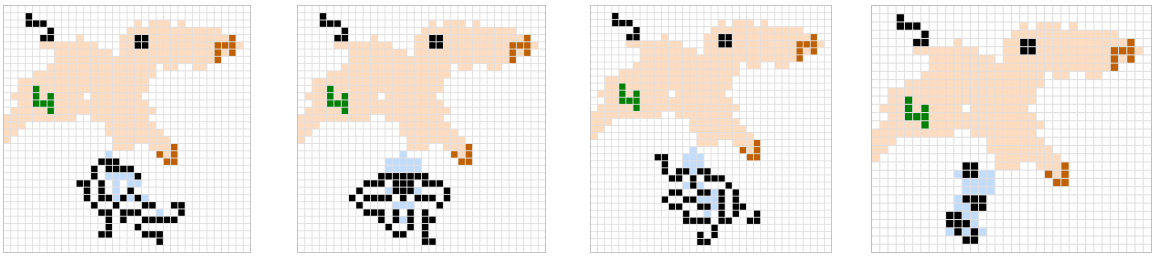
8.5.3 Quetzals: True-Period 52, 54–58, and 61 Guns

The remaining known true-period guns can be constructed using Herschel tracks. However, they make use of custom (typically oscillating) components to overcome the facts that a Herschel's first natural glider cannot be used as the output of a gun with period less than 69, since it would then collide with subsequent Herschels, and the L156 conduit can only release gliders in guns with period as low as 62 (see Exercise 7.37). A gun of this type is called a **quetzalcoatlus** (or simply **quetzal** for short) after the giant flying pterosaur, in reference to the fact that these guns are typically extremely large due to the limited components that we have to build the tracks. In contrast, oscillators of this type (such as the period 56 one from Figure 3.44) are called **emus**, in reference to the fact that they are "flightless" (i.e., they do not emit any gliders).

There are two key ideas that let quetzals work: we can use custom components to extract gliders from Herschels *other* than their first natural glider, and we can use periodic components to decrease the repeat time of some conduits by eating a Herschel's first natural glider before it has a chance to collide with the next Herschel. Some custom conduits that implement the first of these ideas (glider extraction) at some low periods are presented in Figure 8.29, and some additional methods of performing this same task are presented in Exercise 8.20. We note that this glider extraction technique, as well as the other upcoming conduit modifications based on oscillators, is applied to the Fx77 Herschel conduit since it has a much lower repeat time than most other known conduits, and thus is easier to get working at these low periods.



(a) A spark (displayed in orange in the middle) can be used instead of a second eater 1 to extract a glider from the Fx77 Herschel conduit.

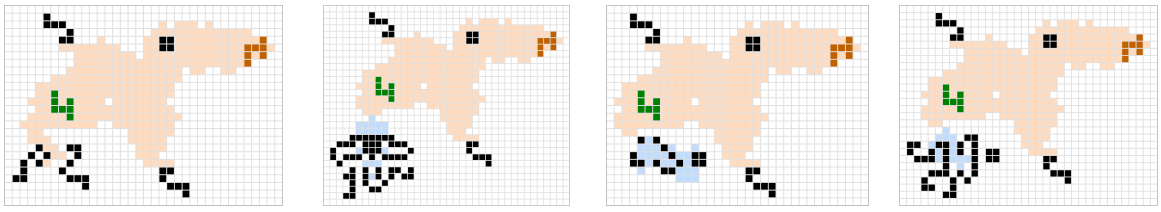


(b) p4 thumb sparker **(c)** p5 MW volcano **(d)** p6 pipsquirter **(e)** p8 blocker

Figure 8.29: A spark can be used to extract a glider from the Fx77 Herschel conduit. This spark can be provided by numerous different oscillators of several different periods.

For the second idea (decreasing a conduit's repeat time), there is a variant of eater 5 that can be used to eat a first natural glider quickly enough to let two Herschels follow each other by as little as 57 generations. In fact, this is exactly why we say that Fx77's repeat time is 57 generations—the first natural glider is its limiting factor. There are also some sparkers that are good enough at eating gliders to allow Herschels to follow each other every 52, 54, 55, or 56 generations (such components could theoretically exist to allow for gaps of 51 or 53 generations as well, but no examples are currently known, and as a result no true-period guns of period 53 are currently known). These various fast eaters are displayed on the Fx77 conduit in Figure 8.30.

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) stable eater 5 variant, repeat time 57 **(b)** p5 MW volcano, repeat time 55 **(c)** p8 blocker, repeat time 56 **(d)** p9 thumb sparker, repeat time 54

Figure 8.30: The repeat time of the Fx77 Herschel conduit can be reduced by quickly and cleanly eating each Herschel's first natural glider. The repeat time can be made as low as (a) 57 via stable components, or (b–d) 51 via oscillating components (though the oscillating components shown here only go as low as 54).

One final component that is extremely useful in these low-period situations is **Jormungant's G-to-H**,[27] which is the conduit displayed in Figure 8.31 that quickly converts two gliders into a Herschel. While a syringe (refer back to Figure 7.8) has a repeat time of 78 generations, which is too large for our purposes here, this two-glider variant of it uses a second glider to clean up some of the glider-to-Herschel reaction, resulting in a significantly smaller repeat time of just 47 generations. The downside of this conduit, however, is that it requires two input gliders rather than just one—it is a 2G-to-H conduit rather than just a G-to-H.

With these mechanisms in hand, we are now in a position to construct true-period glider guns with periods 52, 54, 55, 56, and 57—see Figure 8.32. In particular, we use Jormungant's G-to-H to create a Herschel, which we move along a track via periodic conduits that quickly clean up its first natural glider so as to not disturb the next Herschel, and we extract three gliders from this Herschel: two that will be reflected back into Jormungant's G-to-H, and one that becomes the output of the gun. Since each of the periods that we now tackle have small prime factors, we can implement these reactions via the low-period conduits that we saw earlier (for example, $56 = 2^3 \times 7$, so the period 56 gun can make use of p4, p7, and p8 components).



(a) helper glider from the southeast **(b)** helper glider from the northwest

Figure 8.31: Two versions of a conduit called **Jormungant's G-to-H**, which very quickly and cleanly turn a pair of gliders into a Herschel. They each have a repeat time of 47 generations, but are useful in different situations due to their second input glider coming from different directions.

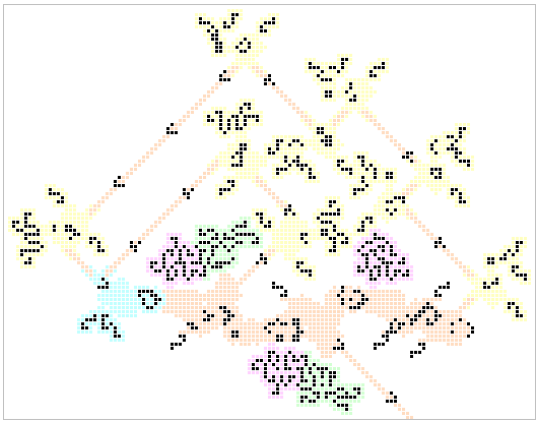
The only remaining periods that are currently known to be attainable by true-period guns are 58 and 61. However, these periods are somewhat more challenging to construct guns for than the previous ones, since they are not divisible by any of the periods for which we have periodic Herschel conduits (like 4, 5, 6, 8, and 9). For this reason, we must entirely make use of stable components, and the only Herschel conduits that have repeat time low enough to function at these periods are Fx77 (57 generations), L112 (58 generations), L200 (see Exercise 7.1(d)—59 generations), and R64

[27] Found by Louis-François Handfield, who goes by the online pseudonym "Jormungant", in April 2018.

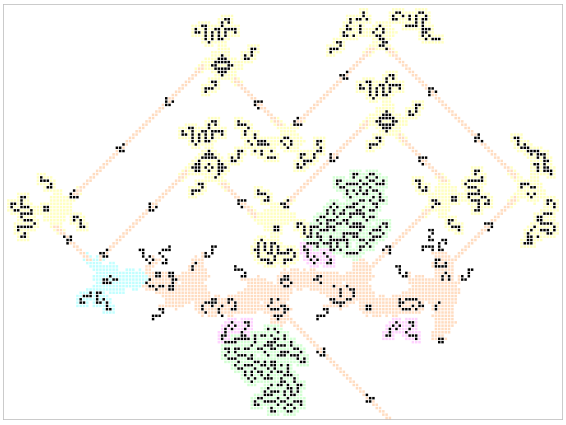
Free PDF. See conwaylife.com/book for pattern files and print version.

(61 generations).

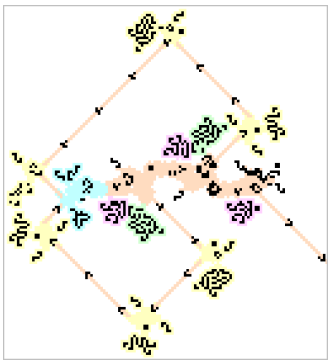
To extract a glider from a Herschel track made up of these conduits, we make use of two reactions—one that uses an extra glider to duplicate a Herschel on the track, and then another one that turns that extra Herschel into two output gliders (see Figure 8.33). It is important that *two* gliders are produced from that duplicated Herschel so that one of them can be used to fuel the earlier Herschel duplication reaction, while the other can be used as the output of the gun.[28]



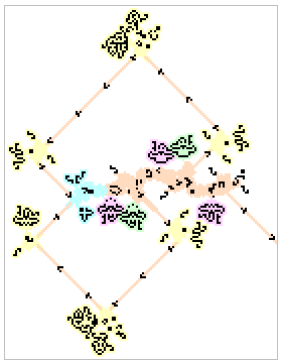
(a) A true-period 52 gun that uses an unnamed p13 oscillator to eat first natural gliders and p4 thumb sparkers to extract other gliders from Herschels.



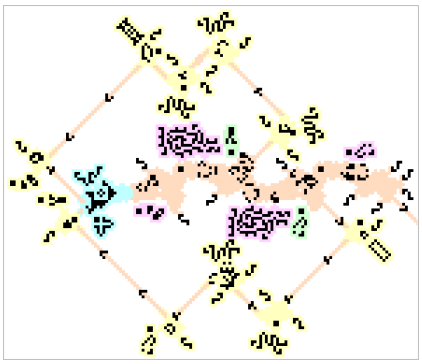
(b) A true-period 57 gun that uses eater 5s to eat first natural gliders and an unnamed p3 oscillator to extract other gliders from Herschels.



(c) A true-period 54 gun that uses p9 thumb sparkers and p6 pipsquirters.



(d) A true-period 55 gun that uses middleweight volcanoes.



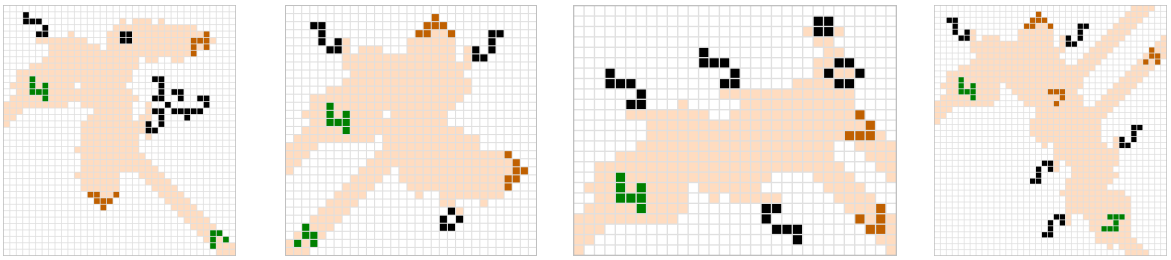
(e) A true-period 56 gun that uses blockers and p7 pipsquirters to eat first natural gliders and extract other gliders.

Figure 8.32: True-period glider guns of periods (a) 52, (b) 57, (c) 54, (d) 55, and (e) 56 that were constructed in 2018 by (a) ConwayLife.com forums user "Entity Valkyrie", (b) Luka Okanishi, and (c–e) Chris Cain. These guns all work by using Jormungant's G-to-H (highlighted in aqua) to create a Herschel, which is fed along a track of Fx77s and oscillators to eat its first natural gliders faster than stable components can (as in Figure 8.30, highlighted in magenta). That Herschel is converted into three gliders—two of those gliders are extracted by oscillators (as in Figure 8.29, highlighted in green) and the third is extracted by a stable H-to-G conduit. Two of those gliders are then reflected (by the reflectors highlighted in yellow) back into Jormungant's G-to-H to complete the loop, while the third glider is the output of the gun.

One slight issue that arises with these particular reactions is that the two output gliders are very close to each other, so it is difficult to separate them. We saw some similar conduits that convert a Herschel into multiple (more easily separated) gliders back in Figures 7.5 and 7.7, but they all have

[28] It is worth comparing these conduits with the Herschel transceivers from Section 7.9. Sometimes it is quicker or easier to send a pair of tandem gliders rather than individual gliders.

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) How to quickly convert a Herschel and a glider into two Herschels or B-heptominoes. The output B-heptominoes can be converted into Herschels, if desired, by BFx59H (see Table 7.3).

(b) How to quickly convert a Herschel into two output gliders. In the conduit on the right, the northwest Herschel should be delayed from the southeast one by 38 generations.

Figure 8.33: Some conduits that transform various types of signals (gliders, Herschels, and B-heptominoes) into each other more quickly than standard conduits. Their low repeat times (see Exercise 8.18) are achieved at the expense of being slightly more complicated than other conduits that we have seen—they all have multiple inputs and/or outputs.

repeat times higher than 61 generations and thus cannot be used in this setting. One method that *does* work at period 61 and higher to separate these closely spaced output gliders is to reflect one of the gliders off of the corner of an L112 conduit as a Herschel passes by.[29] Using this trick lets us construct the true-period 61 glider gun displayed in Figure 8.34.[30]

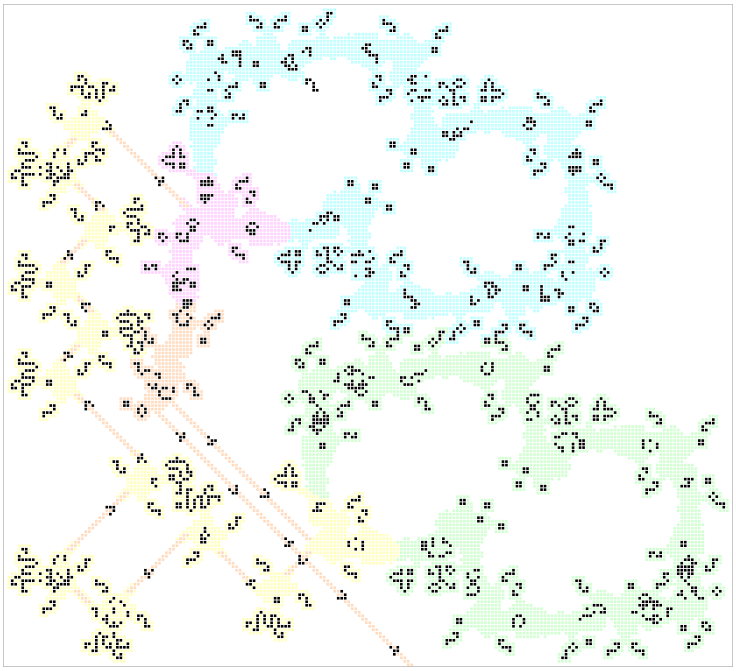


Figure 8.34: A true-period 61 glider gun. It uses one copy of a period 61 Herschel track made up of R64, L112, and L200 conduits (highlighted in aqua) to generate a Herschel via the trick of Figure 8.33(a) (highlighted in magenta). That Herschel is then converted into two gliders via one of the conduits from Figure 8.33(b) (highlighted in orange), one of which becomes the output of the gun and the other of which is reflected by the other copy of the p61 Herschel track (highlighted in green) back around to fuel the Herschel-generating reaction.

[29] This method does not work at period 60 or lower—see Exercise 8.21.

[30] The first true-period 61 gun was found by Luka Okanishi in April 2016. The smaller gun displayed here was constructed on the next day with the help of Chris Cain.

Free PDF. See conwaylife.com/book for pattern files and print version.

This same technique can be used to construct a true-period 58 glider gun, but the details are somewhat messier for a few reasons. First, we no longer have access to the R64 or L200 conduits (their repeat times are 61 and 59, respectively), so we must construct all Herschel tracks out of Fx77 and L112 conduits only. The smallest such track whose length is a multiple of the desired period of 58 makes use of 4 copies of L112 and 68 copies of Fx77, for a total length of $(4\times 112)+(68\times 77)=5\,684$ generations. Placing $5684/58=98$ Herschels on this track results in the period 58 oscillator displayed in Figure 8.35 that we can extract a glider from.

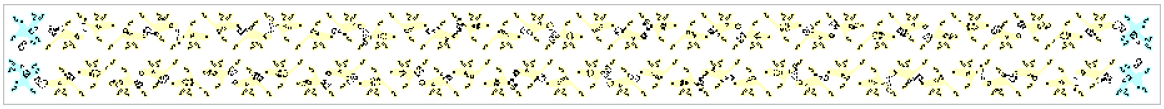


Figure 8.35: A period 58 oscillator that works by placing 98 Herschels on a track made up of 4 L112 conduits (highlighted in aqua) and 68 Fx77 conduits (highlighted in yellow).

The second issue that is somewhat messier to deal with in this period 58 case is that the method we used to separate the two closely spaced glider streams in Figure 8.34 does not work at period 58. Instead, we must use the second method in Figure 8.33(b) to create a different pair of glider streams. While this new pair of glider streams can be separated at period 58 via the same reflection reaction as in the p61 gun, we can avoid adding another large p58 Herschel loop by instead separating the streams with rephasers as in Exercise 4.11. Using two rephasers puts enough space between those streams that we can fit a Snark in between them, and from there we can inject one of those gliders back into the Herschel track just like we did in the p61 gun. The resulting p58 gun is displayed in Figure 8.36,[31] and it completes our collection of known periods that are attainable via true-period guns.

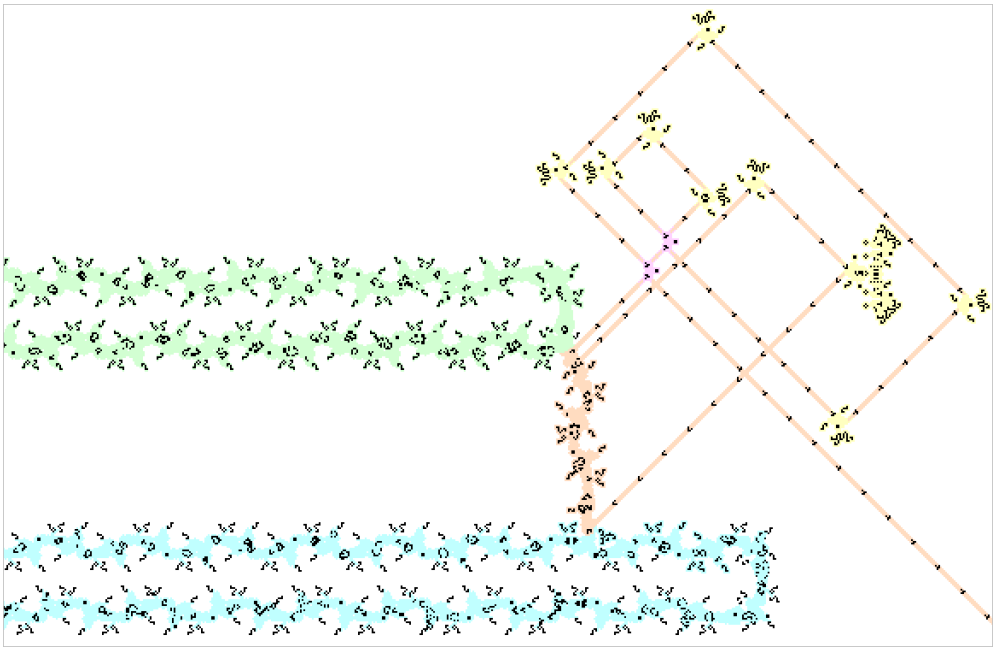


Figure 8.36: A true-period 58 glider gun. A glider is used to extract a Herschel from the period 58 Herschel track oscillator from Figure 8.35 (truncated and highlighted in aqua). That Herschel then collides with another copy of that p58 oscillator (highlighted in green) to create two gliders. Rephasers (highlighted in magenta) are then used to separate those two output glider streams so that one of them can be reflected (by reflectors highlighted in yellow) back into the Herschel-extracting reaction.

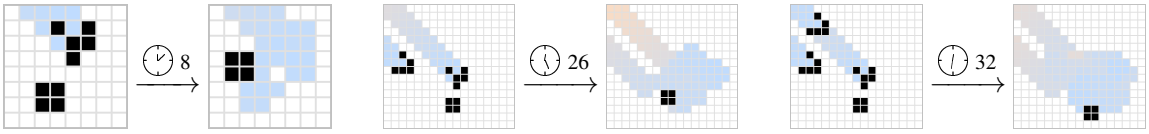
[31] This gun was built by Maia Thunkies and Matthias Merzenich just one day after the p61 gun.

Free PDF. See conwaylife.com/book for pattern files and print version.

8.6 Slide Guns

Now that we know how to construct glider streams with essentially any spacing of our choosing on a single lane, we switch focus a bit and ask how we can reposition those gliders so that a single gun (or group of guns) can create gliders traveling on a wide variety of different lanes. The simplest way to accomplish this task is to use an **elbow**, which is a small and simple still life (typically a block) or constellation (like a honey farm) that can be moved to any point in the Life plane by gliders, and which can also turn gliders.

We already saw several different ways of using gliders to move a block around the Life plane when we investigated slow-salvo synthesis back in Section 5.7.1. However, those recipes can be made slightly more efficient if we do not require them to be slow—that is, if we allow subsequent gliders to collide with the block before the reaction from the previous gliders settles down. For example, we can push a block diagonally away by 1 cell via just 3 synchronized gliders, whereas we required 6 gliders in a slow salvo to achieve the same result back in Figure 5.21(b). Some reasonably simple block-moving recipes of this type are displayed in Figure 8.37.



(a) left 1, up 2 **(b)** left 1, up 1 **(c)** right 1, down 1

Figure 8.37: Some reactions that use gliders to move a block around the Life plane. By combining these reactions with each other, the block can be moved to any location. The reaction (a) is the (2,1) block pull from Figure 2.20, while (b) was found by Dean Hickerson no later than March 1990 and (c) was found by Paul Chapman in 2004. Notice that (c) is exactly the same as (b) except with an additional input glider that changes the direction in which the block is moved.

There are also numerous ways of colliding gliders with blocks so as to create a glider travelling perpendicular to the input gliders without destroying (but potentially moving) the block. A brief selection of glider-reflecting reactions of this type are displayed in Figure 8.38. It is worth comparing these reactions with the one-time turners that we saw in Section 5.7.2—the key difference here is that the block is not destroyed during the reflection. In fact, the reactions in parts (e) and (f) of Figure 8.38 explicitly make use of one-time turners: the first three gliders move the block and create a boat which is used as a one-time turner (and thus destroyed).[32]

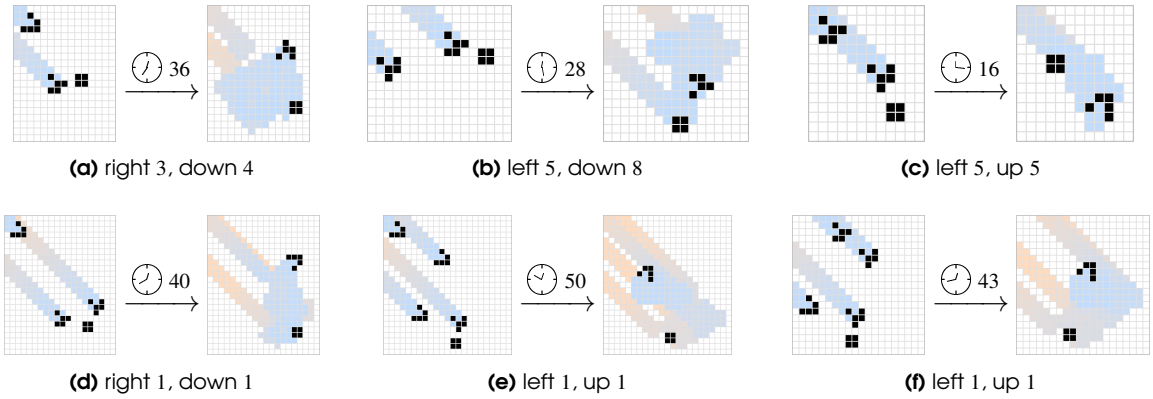
Reflection reactions of this type are particularly useful as they let us use guns along just a finite number of lanes to create gliders traveling along any lane of our choosing, without the need for permanent fixtures like Snarks or glider guns in the locations of those output lanes. For example, if we use one glider to shift a block as in Figure 8.37(a) and then two more gliders to reflect a glider (while moving the block) as in Figure 8.38(a), then the result is that the block is moved a total of $3-1=2$ cells right and $4-2=2$ cells down (i.e., perfectly diagonally) and a single glider is output in a perpendicular direction. By repeating this reaction (by producing those 3 input gliders via glider guns, for example), every perpendicular glider is released 4 lanes farther away than the one that came before it. We call the resulting gun a **slide gun**, and one example is displayed in Figure 8.39.[33]

Numerous other slide guns and variants of slide guns can also be created by combining these block-moving and glider-reflecting reactions in various ways. For example, the reaction from Figure 8.38(d) can be used to create a slide gun that has gliders separated by 2 lanes instead of 4 (see Exercise 8.22). We could similarly create a "slide gun" that doesn't slide at all by choosing a block-

[32] We first saw that a boat can be used as a one-time turner in Exercise 5.31.

[33] The first slide gun was constructed by Dietrich Leithner in July 1994 using only the reaction from Figure 8.38(d).

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) right 3, down 4 **(b)** left 5, down 8 **(c)** left 5, up 5

(d) right 1, down 1 **(e)** left 1, up 1 **(f)** left 1, up 1

Figure 8.38: Some ways of colliding gliders coming from one direction with a block so as to produce a glider traveling perpendicularly. The block is not destroyed, but it may be moved—it can then be moved back, if desired, by the reactions shown in Figure 8.37. Reactions (e) and (f) were found by Paul Chapman in 2004 and make use of the same first two gliders as the block mover from Figure 8.37(b). Their final input glider can be delayed by any amount, and their output gliders have opposite colors.

moving combination of gliders that returns the block to its initial position after reflecting a glider. One possible way to achieve this effect is to use a total of 7 gliders: the reflection from Figure 8.38(a) followed by the move of Figure 8.37(a) and then the move of Figure 8.37(b) twice (see Exercise 8.27).

We can even perform different block-reflecting operations like this in sequence. For example, we could create a slide gun that fires *two* gliders on each of its output lanes by using a 10-glider recipe: 7 gliders to create a perpendicular glider without moving the block, and then 3 more gliders to create another perpendicular glider while moving the block. The resulting "double" slide gun is displayed in Figure 8.40—it is made up of 10 glider guns (instead of just 3 like the regular slide gun from Figure 8.39), and due to the tight arrangement of 10 gliders that it must fire, it makes heavy use of the glider-insertion technique that we saw back in Figure 8.8(b).

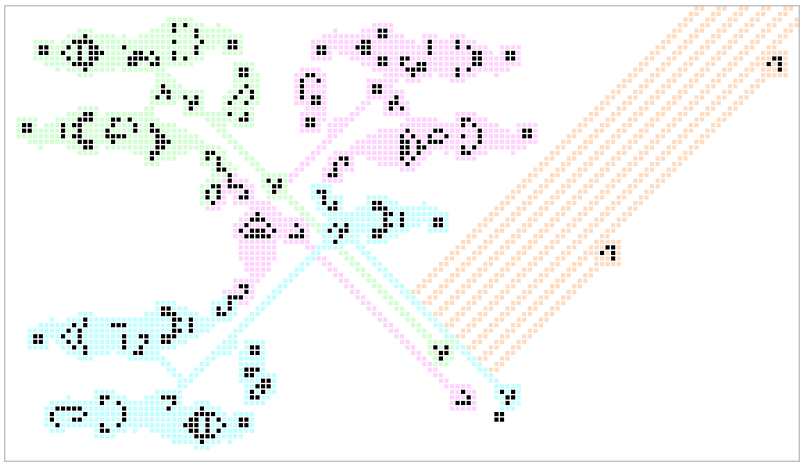


Figure 8.39: The glider highlighted in aqua moves the southeastern block and then the gliders highlighted in magenta and green create a perpendicular glider (while again moving the block). The total resulting movement of the block is 2 cells southeast, so this **slide gun** fires gliders on infinitely many different lanes (highlighted in orange). The three constituent p120 glider guns each consist of the p60 gun from Figure 8.2(a) together with a blocker to filter its output (and some buckaroos to reposition their glider streams).

Free PDF. See conwaylife.com/book for pattern files and print version.

Theoretically, this method could be used to create any sequence of gliders appearing on any sequence of lanes of our choosing, though the details become increasingly cumbersome as the desired pattern of output gliders increases in complexity. We will return to this topic in Chapter 11, where we discuss how to use these ideas to build a pattern that can create copies of any pattern that can be synthesized by gliders (including itself).

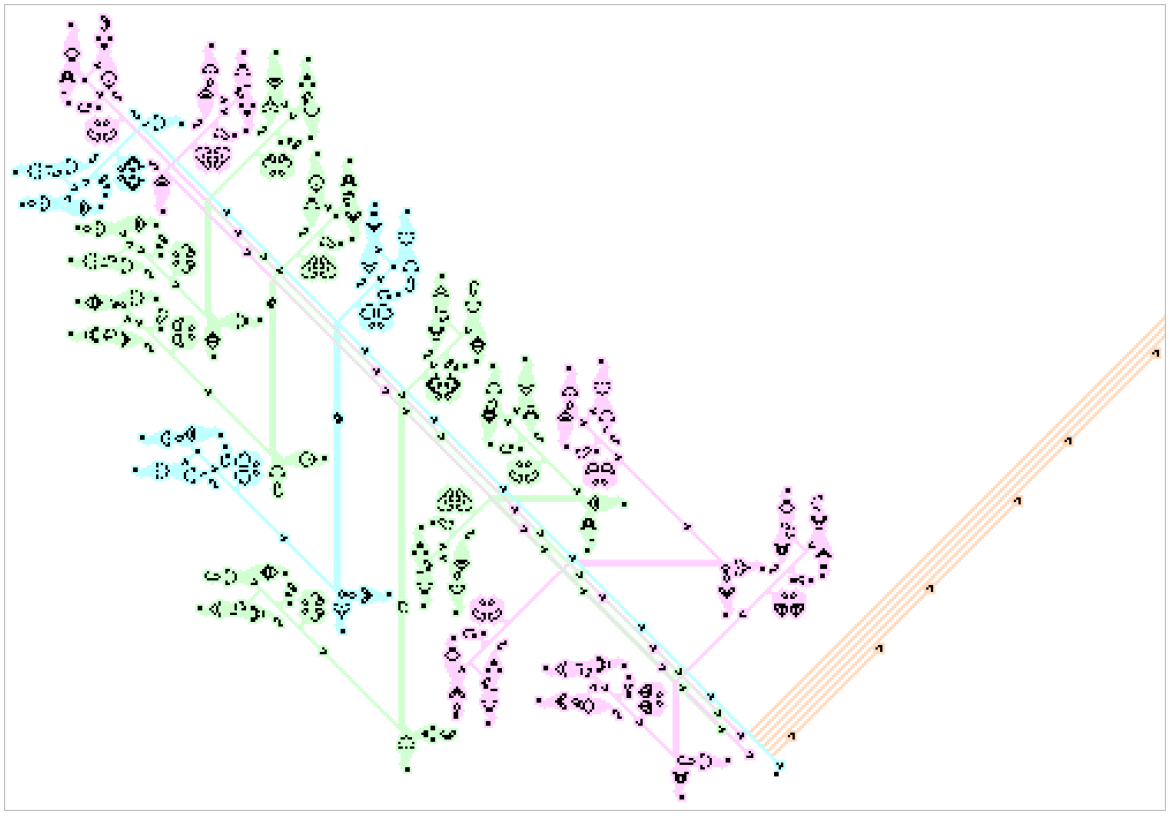


Figure 8.40: A slide gun that fires two gliders on each of its output lanes. The numerous p240 guns that make up this slide gun are each the same as the p120 guns from Figure 8.39, but with Rich's p16 attached to filter the output. The 10 gliders used in the block-pushing recipe consist of two copies of the single-glider block mover from Figure 8.37(a) (highlighted in aqua), two copies of the two-glider block mover from Figure 8.37(b) (highlighted in green), and two copies of the two-glider reflector from Figure 8.37(a) (highlighted in magenta).

8.7 Armless Guns

It is very useful to be able to fire gliders along arbitrary lanes via guns that work on just a fixed set of lanes, and the block-pushing reactions of the previous section give us one way of doing so. However, perhaps a simpler way of achieving the same effect is to collide antiparallel gliders with each other so as to create a perpendicular glider. We saw that this can be achieved via the "tee" collisions of Table 5.2, but those reactions require us to be able to generate input gliders on 3 different lanes, including two lanes that are quite close together. In fact, since a given tee reaction always produces gliders of the same color, we would need two separate tee reactions and a total 6 different input lanes if we wanted to use this method to generate perpendicular gliders of both colors.

Free PDF. See conwaylife.com/book for pattern files and print version.

8.7.1 Monochrome Armless Guns

Luckily there is a known component, which we saw in Exercise 7.36, that converts a single Herschel directly into a pair of gliders that can be used in one of the tee reactions. This means that we won't need three guns to fire a perpendicular glider via a tee—we can instead get away with just two, by having one of them generate glider pairs via this H-to-2G component while the other produces the corresponding antiparallel gliders.

If we use loop guns (like the adjustable one from Figure 7.17) to produce these gliders, we can then adjust the lane that the output glider is produced on simply by adjusting the relative spacing of the gliders in those loops. However, these guns are necessarily **monochrome**—they can only produce gliders of a single color. In particular, delaying a glider in one of the loops by 8 generations (i.e., 2 full diagonals) will shift the corresponding output glider from the tee reaction by one full diagonal, but there is no way to adjust the gliders in the loops so as to shift the output glider by just a half diagonal. To get around this problem, we can double the number of loop guns from two to four: one pair to produce gliders of one color, and another pair to produce gliders of the other color, as illustrated in Figure 8.41.

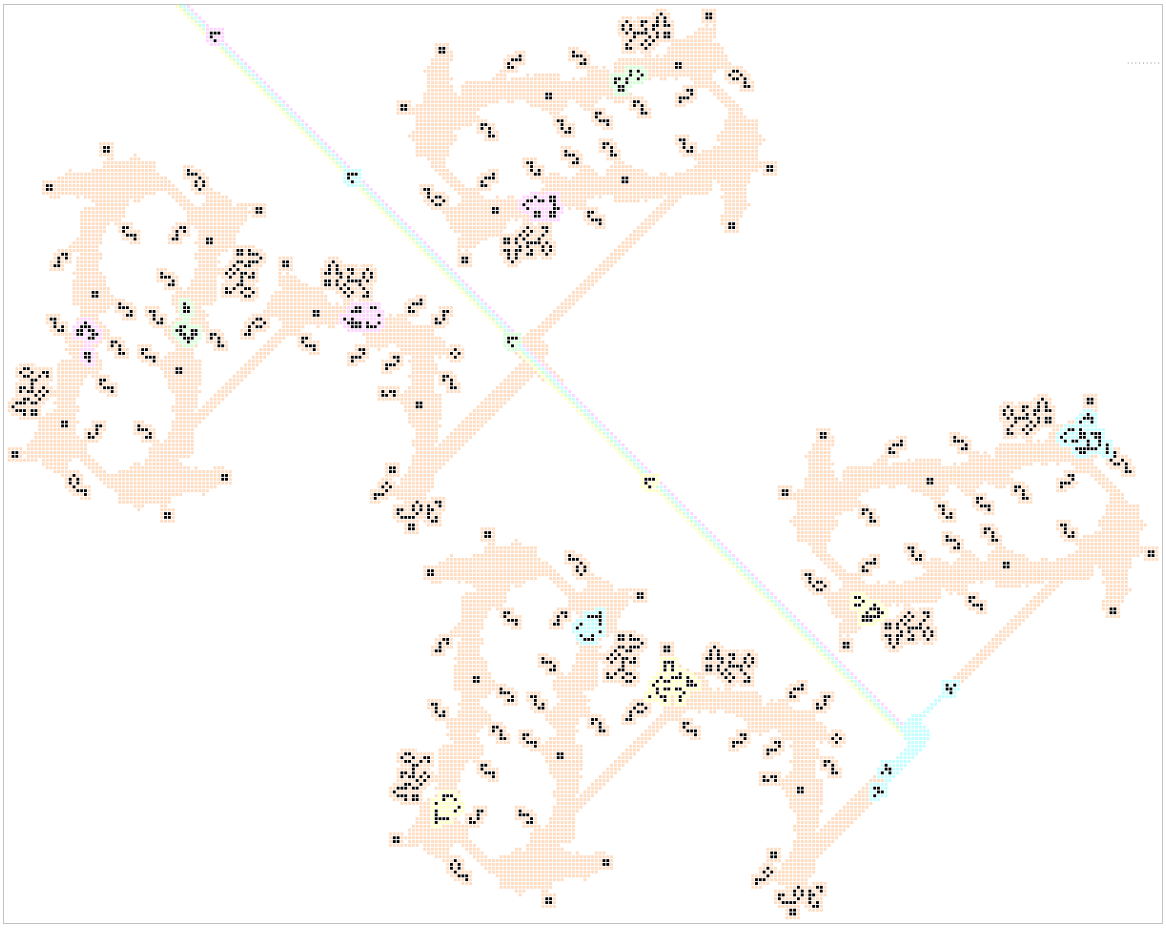


Figure 8.41: A gun that fires gliders to the northwest on four consecutive lanes (highlighted in order from right-to-left in magenta, aqua, green, and yellow). The first and third gliders are created by the northwest pair of loop guns, while the second and fourth gliders (of the other color) are created by the southeast pair of loop guns. The output lanes can be adjusted simply by adjusting the relative spacing of the gliders in the loops. The loop guns on the southwest feed into the SW1T43 H-to-2G conduit from Exercise 7.36 so that they produce 2 of the 3 gliders needed in the tee reaction at once.

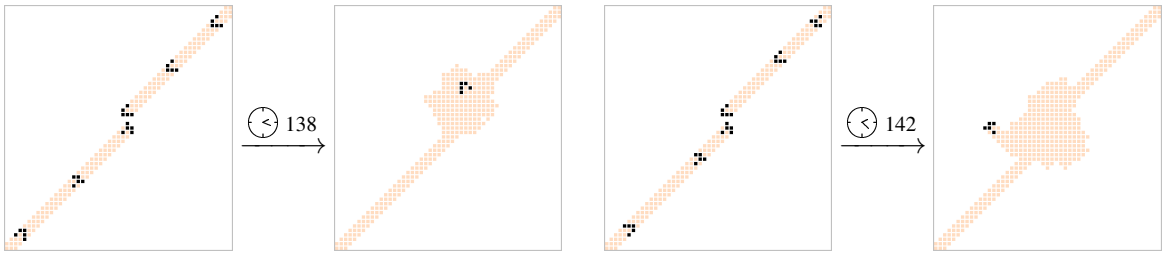
Free PDF. See conwaylife.com/book for pattern files and print version.

8.7.2 Multicolor Armless Guns

It is somewhat awkward to have to use a separate pair of loop guns to generate gliders of each of the two colors—it results in more glider streams to maintain, and additional constraints come into play because the gliders of one color have to cross the construction area for the gliders of the other color. This raises a natural question: is there a way to encode output gliders of both colors in a single pair of loop guns?

There is in fact a known way to do this. Instead of using 3 synchronized gliders to create a perpendicular one, as in tee reactions, we can use slow antiparallel glider collisions that only make use of 2 gliders at a time, always arriving from the same two antiparallel lanes. This gives us the maximum amount of flexibility in storing the gliders in a loop, and extracting copies—we only need to ensure that the time between adjacent gliders is at least the repeat time of the circuitry used to build the loop. For many syringe-based reflectors and splitters, this minimum repeat time is 78 generations (but it can be overclocked down to 74 generations).

The simplest known reactions of this type are displayed in Figure 8.42, which make use of 3 antiparallel 2-glider collisions to generate a perpendicular output glider of either color. Slowing down the input gliders in one half of these reactions by $8n$ generations (i.e., moving them backward by $2n$ full diagonals) will move the output glider in that direction by n full diagonals. We can thus use these two 6-glider recipes to create an output glider on any perpendicular lane of our choosing.



(a) output glider of one color **(b)** output glider of the other color

Figure 8.42: A way of colliding 3 single-lane antiparallel pairs of gliders so as to create perpendicular gliders of either color. The collisions are **slow** and p2 (some of the collisions create intermediate blinkers), so the glider pairs can be delayed relative to each other by any even number of generations without affecting the overall reaction.

The wonderful thing about not having to worry about the timing between the glider pairs in these reactions, and only having to use a single lane, is that we can now use just two loop-based guns to generate gliders on any sequence of lanes of our choosing. For example, we can now create a gun like the one from Figure 8.41 that cycles between firing gliders on four consecutive lanes, but which only uses two loop guns. The trade-off is that instead of each loop gun having 2 gliders on its loop (for a total of $2 \times 4 = 8$), we now need $4 \times 3 = 12$ gliders in each loop (for a total of $12 \times 2 = 24$).

To actually build this gun, we need to figure out which antiparallel glider pairs we need to fire at each other, and then we place those pairs into two loop guns that are aimed at each other. In particular, for each output glider that we wish to generate, we first choose which collision from Figure 8.42 we should use based on the color of that glider, and then we delay the 3 gliders on one half of that collision by 8 generations (i.e., 2 full diagonals) at a time until the output glider is produced on the correct lane. The resulting gun is displayed in Figure 8.43.

8.7.3 Armless Construction

This armless method of generating gliders is particularly useful for implementing slow salvo syntheses. The primary method that we saw earlier for generating a slow salvo was to use numerous carefully

Free PDF. See conwaylife.com/book for pattern files and print version.

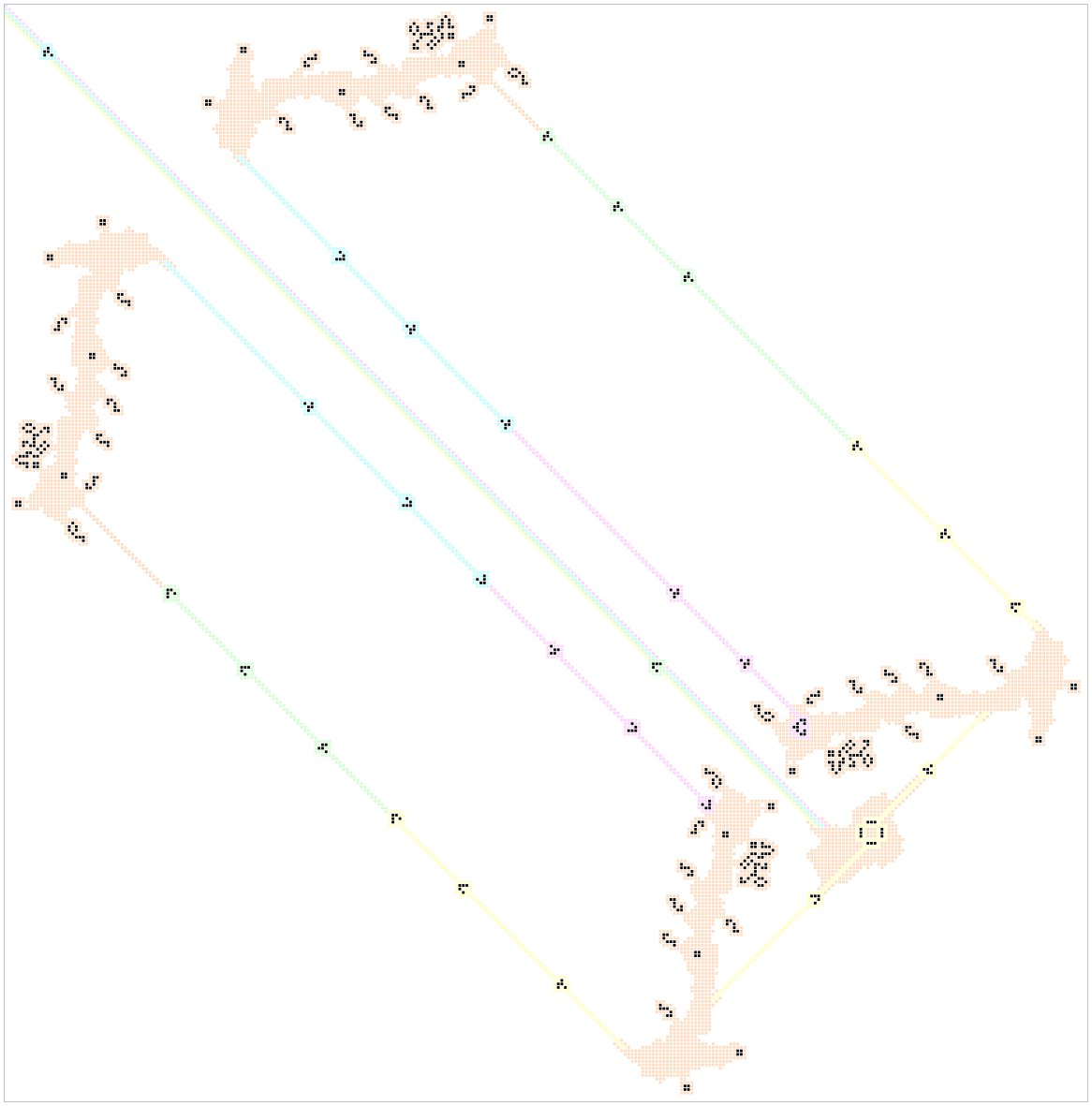


Figure 8.43: A gun that fires gliders to the northwest on four consecutive lanes (highlighted in order from right-to-left in magenta, aqua, green, and yellow). The first and third gliders are created by the reaction from Figure 8.42(a), while the second and fourth gliders of the other color are created by the reaction from Figure 8.42(b).

placed high-period guns (many of which would have to be edge shooters). However, if a salvo of gliders takes up too many lanes, then a conventional edge shooter may not have enough clearance to place a glider in one of the central lanes from either side. This is where armless guns shine: they can reach across any number of lanes, as long as the input gliders do not interfere with perpendicular output gliders that have already been placed in the salvo.

For this reason, we sometimes encode slow salvos in the sequences of gliders in pairs of (quite large) loop guns, which fire at each other to create perpendicular gliders on whichever lanes are needed in the slow salvo synthesis. This method of implementing a slow salvo synthesis is called **armless** construction, in contrast with **armed** construction that uses a block (or other still life or constellation) to position the slow gliders as in Section 8.6.

Free PDF. See conwaylife.com/book for pattern files and print version.

Unfortunately, slow-salvo syntheses of even moderately sized objects like Corderships tend to require hundreds of gliders, so huge and unwieldy loop guns would be needed to generate those syntheses. However, if we are willing to use 2-direction slow syntheses instead, then the glider recipes shrink considerably. That is, we use **slow glider pairs**: the two gliders in each pair have to be precisely synchronized with each other, but an arbitrarily long delay can be added between any two pairs of gliders. By using this technique, the slow salvo recipe for a 2-engine Cordership decreases from 112 unidirectional slow gliders to just 17 slow glider pairs, as illustrated in Figure 8.44.[34]

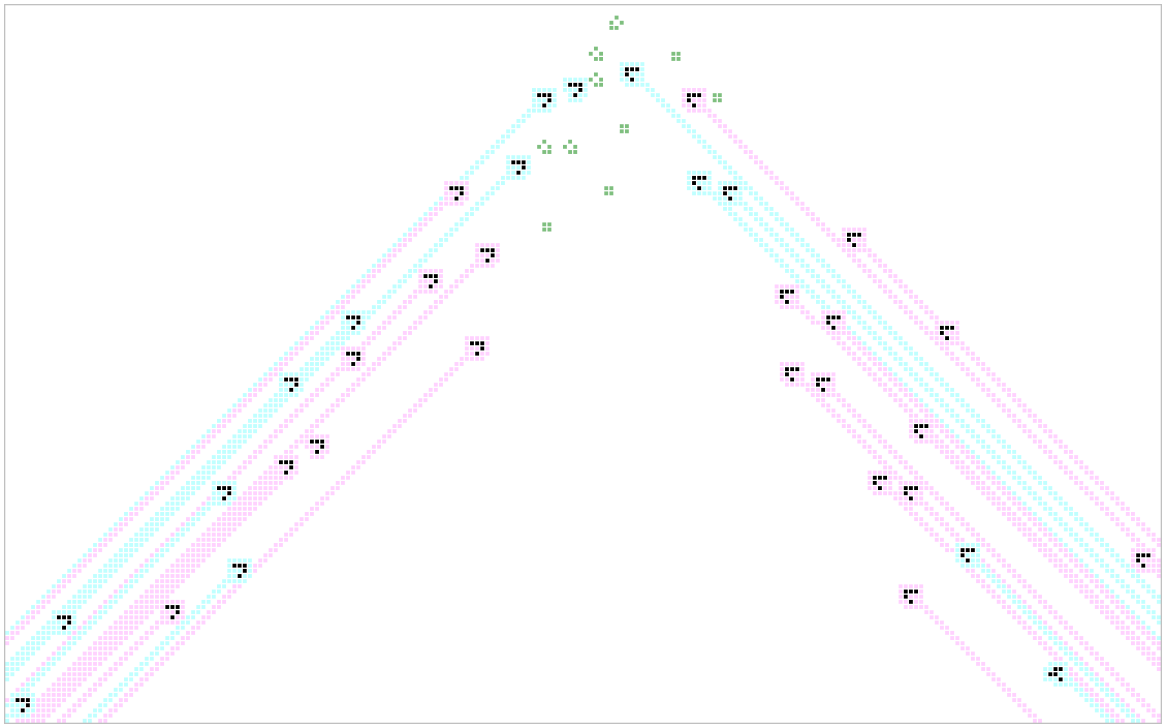


Figure 8.44: A way of synthesizing a 2-engine Cordership via 17 slow glider pairs. The first 16 pairs synthesize a seed (displayed in green), while the final pair turns that seed into the 2-engine Cordership. Gliders of one color are highlighted in aqua, while gliders of the other color are highlighted in magenta.

The two salvos that make up these slow glider pairs each contain gliders of both colors, so if we are to use the monochrome guns based on tees to create them, as described in Section 8.7.1, we need a total of 8 glider loops: 2 for each color in each salvo. Those loop guns will create the 17 slow glider pairs via a total of 68 gliders in the 8 different loops (2 gliders in the loops for each of the 34 gliders in the 17 pairs). The 2-engine Cordership gun that results from this construction is displayed in Figure 8.45.

If we instead make use of the multicolor armless techniques that were described in Section 8.7.2, the same 17 glider pairs in the Cordership recipe from Figure 8.44 can be created by a total of just four loop guns instead of eight, as illustrated in Figure 8.46. However, the trade-off is that each loop gun requires far more gliders in its loop in order to encode the synthesis. In particular, the loops each contain $3 \times 17 = 51$ gliders, for a total of $4 \times 51 = 204$: exactly three times as many as the 68 that were used in the loops of the monochrome gun from Figure 8.45).

[34] Both of these slow syntheses work by first building the seed for the 2-engine Cordership that we constructed in Exercise 5.44. The 112-glider unidirectional recipe is given in Exercise 11.20.

Free PDF. See conwaylife.com/book for pattern files and print version.

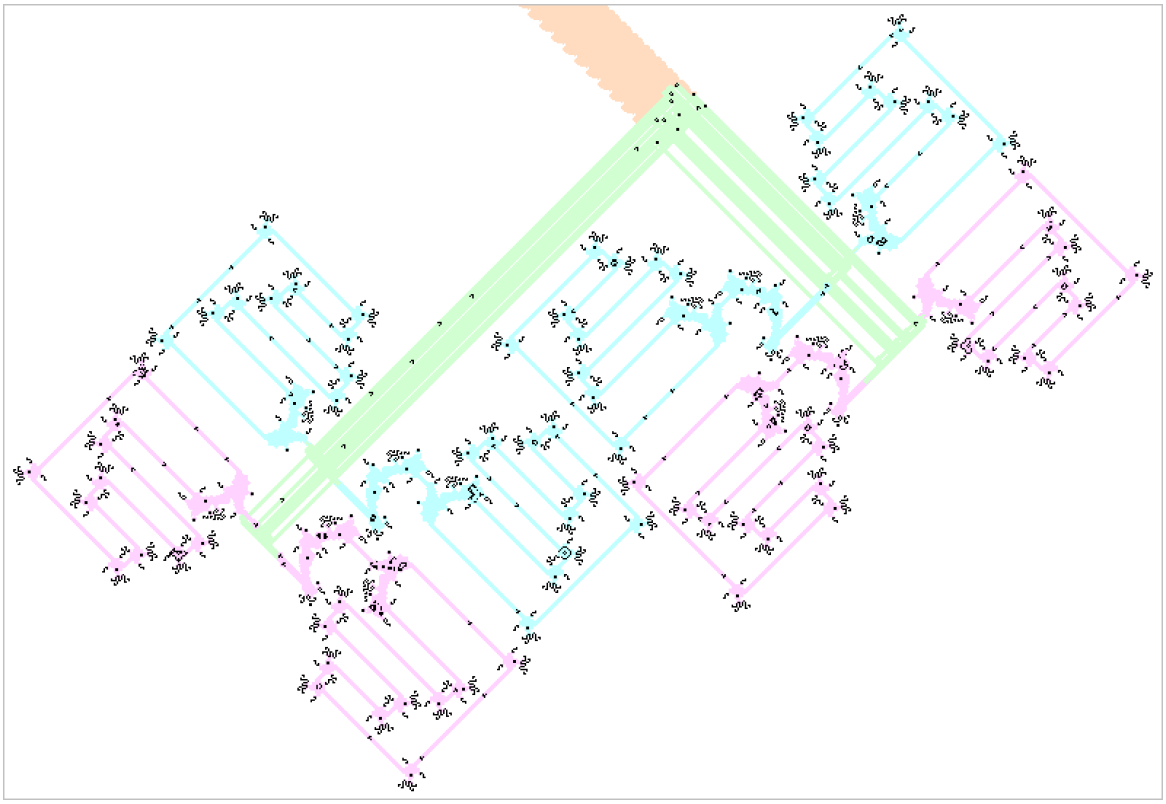


Figure 8.45: A 2-engine Cordership gun that works by using 8 loop guns (highlighted in aqua and magenta, and whose tracks are "folded" via Snarks just to reduce their size) to create gliders of different colors. Those glider pairs travel along various lanes (highlighted in green) that implement the 17-glider-pair slow salvo synthesis of a 2-engine Cordership from Figure 8.44.

8.7.4 Comparison of Methods

There are advantages and drawbacks to each of the monochrome (i.e., tee-based) and multicolor armless construction techniques, which we now compare and contrast. Some reasons that armless construction based on monochrome tee-based guns is sometimes advantageous include:

- Their total population will be lower, since only two stored loop gliders are needed per output glider, whereas multicolor armless guns need six.
- Their repeat time will be smaller, since the glider loops contain fewer gliders and can thus be smaller.
- They are somewhat easier to adjust: to change a glider's output lane by n cells, it is only necessary to move a single glider by $2n$ full diagonals in one of the loops, whereas adjustments would have to be made to three gliders in the multicolor design.

However, multicolor armless designs have other compensating advantages that sometimes make them preferable:

- They have half as many glider loops.
- Interleaving different-colored gliders is relatively trivial, with no need to worry about gliders from the back pair of guns interfering with the forward guns' insertion reaction (or vice versa).
- The glider loops that generate antiparallel colliding gliders can be made to be Spartan, and thus easy to synthesize via slow salvos. In contrast, the conduit from Exercise 7.36 that serves as

Free PDF. See conwaylife.com/book for pattern files and print version.

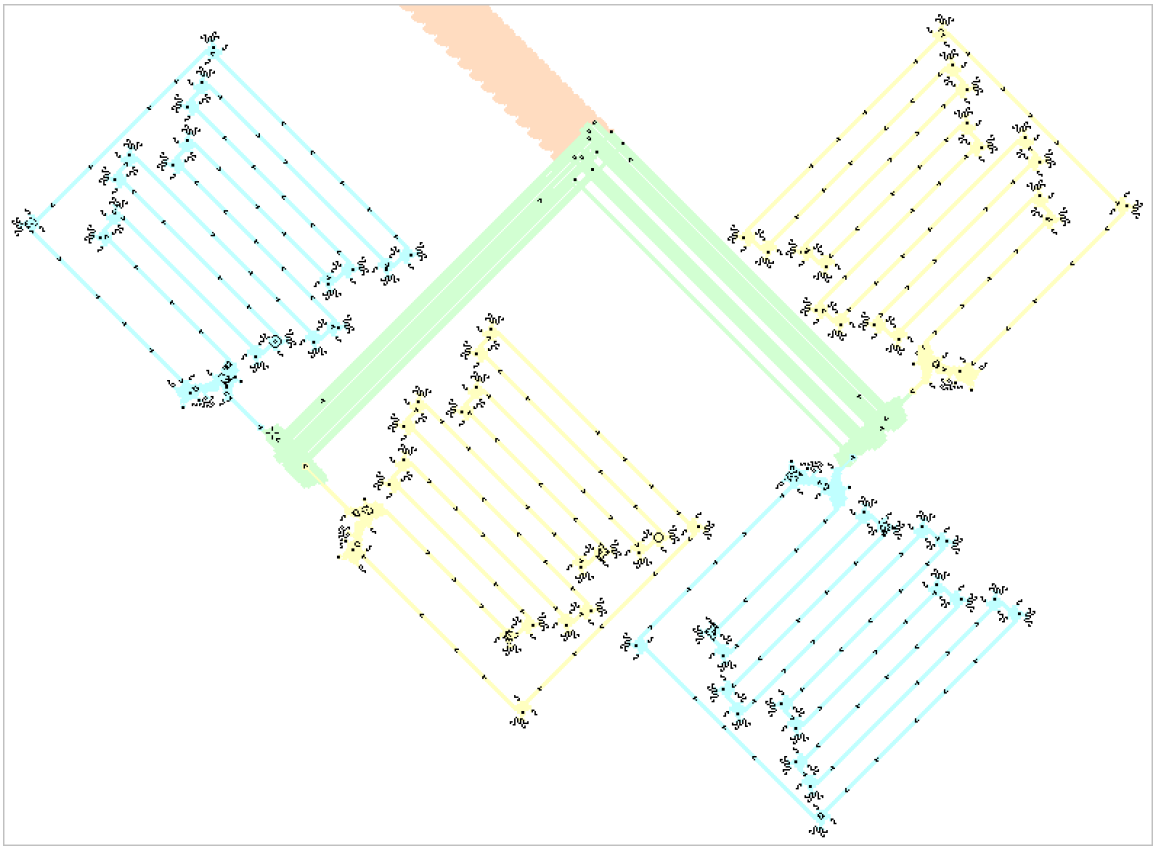


Figure 8.46: A 2-engine Cordership gun that works by using 4 loop guns (highlighted in aqua and yellow) that each contain 51 gliders along their tracks (which are "folded" via Snarks just to reduce their size). Those loop guns create 17 slow glider pairs along various lanes (highlighted in green), which synthesize the Cordership.

the basis for monochrome tee-based armless construction is not Spartan, so it would not be a reasonable part of any circuitry that must itself be constructed by gliders.

We have now seen four different explicit constructions of a gun that fires 2-engine Corderships: we saw how to use its 9-glider synthesis to construct such a gun "directly" in Exercise 6.20, we saw how to transform a glider into a 2-engine Cordership (and thus any sufficiently high-period glider gun into a 2-engine Cordership gun) in Figure 7.16, and we saw two different ways of using armless construction to fire 2-engine Corderships using loop guns in Figures 8.45 and 8.46. Even though the guns based on armless construction are much larger, and perhaps seem more complicated at first, they have the advantage that they can be more easily modified to synthesize other types of objects.

Indeed, the biggest advantage of armless construction in general, whether monochrome or multicolor, is that the general form of the gun would not significantly change if we wanted it to synthesize a completely different spaceship like a loafer or an ecologist. All that we would have to change is the sequences of gliders running through the glider loops. This trick of encoding patterns in sequences of gliders, rather than in the positions of various complicated guns, will be extremely useful once we investigate universal construction in Chapter 11.

Free PDF. See conwaylife.com/book for pattern files and print version.

8.8 Slow and Irregular Guns

We have spent most of this chapter looking at methods of constructing fixed-period guns with evenly spaced output. We now switch gears slightly and construct some guns that fire gliders more and more slowly, thus giving us our first examples of patterns with sub-linear (but non-zero) growth rates.

8.8.1 Sqrtguns

One method of creating a gun that gets slower and slower is to use a Herschel and/or glider track that becomes longer and longer. This can be achieved by making use of the block-pushing reaction that we saw back in Figure 7.40—every time the 3 input gliders hit the block it is pushed farther away, thus increasing the time that it takes for the next 3 input gliders to hit the block. If we can feed the 2 tandem output gliders into a circuit that recreates the 3 input gliders and also creates an output glider for the gun to fire, we will be done.

Fortunately, this is a relatively straightforward exercise involving the circuits that we have seen. We can convert the tandem gliders from the block-pushing reaction into a Herschel via the Herschel receiver that we saw in Figure 7.39. That Herschel can then be converted into 4 gliders via stable conduits like the ones that we saw in Section 7.2, and 3 of those gliders can be redirected back toward the block so as to start the process all over again. The resulting pattern is a gun that emits each glider 80 generations more slowly than its previous one, so its n-th glider is released in a generation that is asymptotically proportional to n^2 (specifically, in generation $40n^2 + 2029n - 969$). Such a gun is called a **sqrtgun** (pronounced "squirt gun"), in reference to the fact that its population in generation t is $\Theta(\sqrt{t})$[35], and one example is displayed in Figure 8.47.[36]

8.8.2 Caber Tossers and Logarithmic Growth

We now construct a gun that fires gliders even more slowly than a sqrtgun. The idea that we now use is to bounce a glider back and forth between a receding spaceship and a stationary object that reflects and duplicates the glider. Because the spaceship gets farther and farther away, the time that it takes for the bouncing glider to make a round-trip and get duplicated increases by some multiplicative factor every loop (rather than by some additive factor, as in sqrtguns), so the resulting growth rate is logarithmic.

Guns of this type are called **caber tossers**, and the simplest ones to construct make use of a single glider bouncing off of the back of a Cordership (we already saw how to bounce gliders off of the back of Corderships in Figure 4.24 and Exercise 4.20, for example). The only somewhat tricky part of their construction is the object that duplicates and reflects the bouncing glider back toward the Cordership. We saw several conduits that could carry out this task in Chapter 7, but we will instead make use of a periodic circuit that is somewhat smaller. In particular, we aim one glider gun in the correct positioning and timing as to be reflected by the receding Cordership, but we place another glider gun so as to block it. Then we just fiddle with the timing and spacing until we find a configuration in which the returning glider blocks the blocking stream as in an inverter,[37] thus letting a single glider from each stream escape and repeat the process. One configuration that works is displayed in Figure 8.48.[38]

For this particular caber tosser, the bouncing glider takes twice as long to make a round trip every time a glider is released, so each glider takes twice as long to be released as the one before it. In

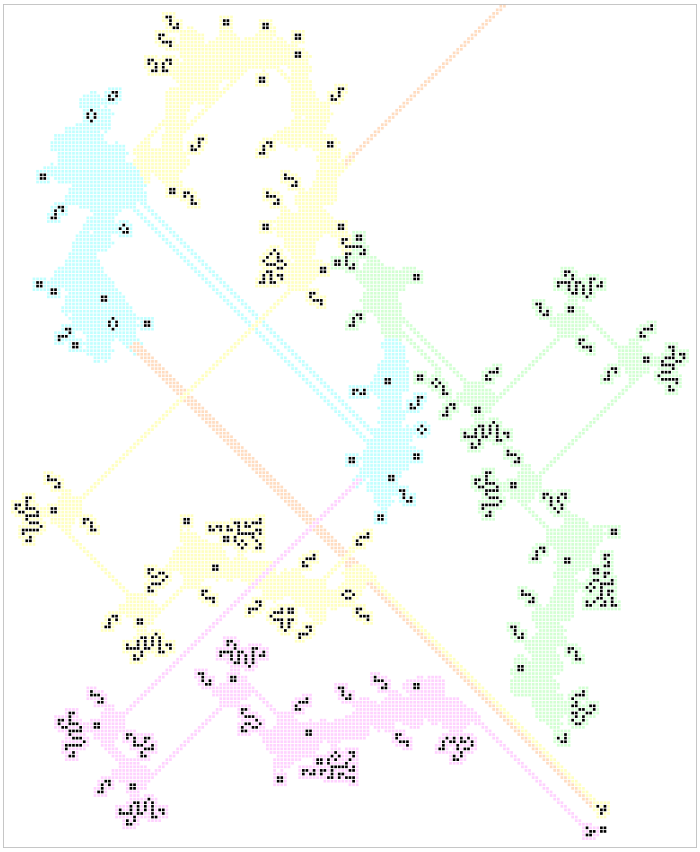
[35] Refer to Appendix A.3 if you are unfamiliar with this "big theta" notation.

[36] The first sqrtgun was constructed by Dean Hickerson sometime around 1991.

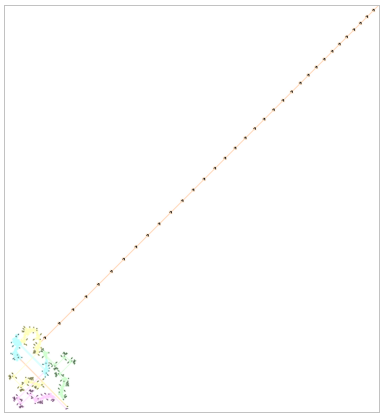
[37] This configuration can be found by hand relatively straightforwardly—recall that roughly one third of all two-glider collisions result in the gliders destroying each other, and one of those is what we want here.

[38] The first caber tosser was built by Dean Hickerson in May 1991 using this same arrangement of guns, but a slightly different glider reflection involving the 13-engine Cordership.

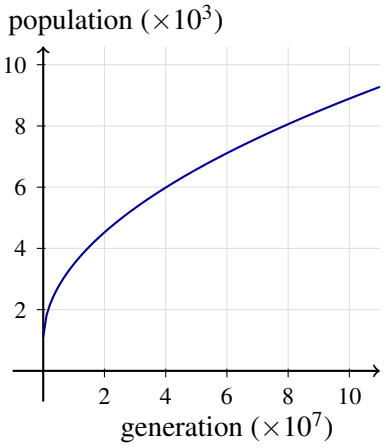
Free PDF. See conwaylife.com/book for pattern files and print version.



(a) Three gliders (highlighted in yellow, green, and magenta) push a block, thus extending the length of the track and delaying the next glider that the gun releases by 80 generations. Makes use of two Herschel transceivers (highlighted in aqua) and otherwise standard Herschel conduits.



(b) by generation 115000 we can see wider gaps between later gliders



(c) population versus time

Figure 8.47: A **sqrtgun** that fires gliders slower and slower. Its population in generation t is asymptotically proportional to $\sqrt{t}$. Note that in (b), the spacings between objects are to scale, but the objects themselves have been magnified (the bounding box at that point is roughly 30000×30000, so the objects would be too small to see otherwise).

particular, its n-th glider is released in generation $480 \times 2^n - 727$, so its growth rate is logarithmic (i.e., its population in generation t is $\Theta(\log(t))$).[39]

8.8.3 Recursive Filters and Slower Growth Rates

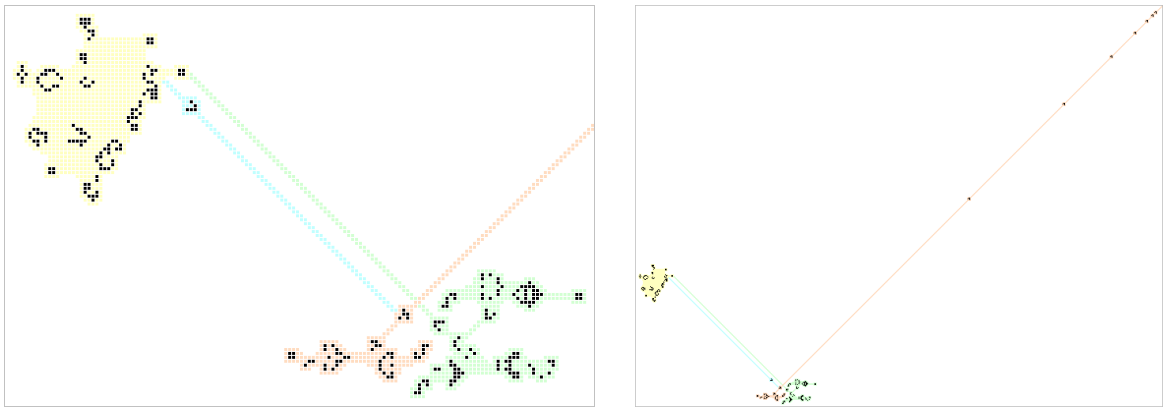
Now that we know how to create patterns that grow logarithmically, it seems natural to ask "How slow can you grow?" That is, are there patterns that grow even slower than logarithmically, and is there an asymptotically slowest-growing pattern? The answers to these questions are "yes" and "no", respectively, and they both follow from the existence of devices that can be attached to the output of a glider gun so as to create an asymptotically slower glider gun.[40]

One such device is displayed in Figure 8.49. This object is called a **recursive filter**, and when it

[39] All logarithms that we consider in this chapter are base-2. However, it does not matter what base we use in this particular case.

[40] In this section, we are discussing the growth rate of the *population* of a pattern, which can be made slower than any computable function. On the other hand, there *is* a slowest possible growth rate of the *bounding box* of a pattern, and we will see a pattern attaining it in Section 9.7.2.

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) a caber tosser firing gliders northeast **(b)** by generation 121 920 it has emitted 7 gliders

Figure 8.48: A **caber tosser** with logarithmic growth rate. A period 60 glider gun (highlighted in green) fires gliders at a 2-engine Cordership (highlighted in yellow) to be reflected back, but is blocked by a period 30 glider gun (highlighted in orange). When that returning glider arrives at the pair of guns, it allows them to each release a single glider, which restarts the process. Since the Cordership is constantly moving away from the guns, the round trip taken by the triggering glider takes longer and longer (in fact, exactly twice as long each time) so the output gliders have logarithmic spacing. In (b), the spacings between objects are to scale, but the objects themselves have been magnified.

receives a glider from whatever gun is attached to its input[41] (typically a slow gun like a caber tosser, but "standard" guns work fine too), it releases a pair of lightweight spaceships to the east. Then one of three things happens:

1) If this was the first pair of lightweight spaceships to be released, they hit the sparks at the back of the unnamed $c/3$ eastbound spaceship, creating a loaf.
2) If a loaf is present in the path of the lightweight spaceships, they collide so as to pull the loaf westward (i.e., closer to the recursive filter) by 6 cells.[42]
3) If the loaf has been pulled all the way back to the recursive filter, it is destroyed and the filter then releases a single glider.

The result of this behavior is that if the $c/3$ spaceship is $2n$ cells away from the recursive filter when its first input glider releases the first pair of lightweight spaceships, they will make the first loaf about $12n$ generations later when the $c/3$ spaceship is $6n$ cells away. It will then take $6n/6 = n$ additional input gliders to pull that loaf back to the recursive filter, at which point its first output glider will finally be released.

For example, if we attach a caber tosser to this recursive filter then it takes about $480 \times 2^{n+1}$ generations for those $n+1$ total input gliders to come in, so it takes that same amount of time for the recursive filter to release its first output glider. In particular, if the caber tosser starts in the phase displayed in Figure 8.48(a) then it takes about 500 generations for the first glider from the caber tosser of Figure 8.48 to release the recursive filter's first pair of lightweight spaceships, at which time the $c/3$ spaceship is 208 cells away from the recursive filter. We thus expect the filter's first output glider to appear after $208/2 = 104$ additional input gliders come in, at approximately generation

$$480 \times 2^{104+1} \approx 1.947 \times 10^{34}.$$

[41] The input glider must be aligned to period 120. The caber tosser from Figure 8.48 thus works fine for generating input to this recursive filter, but if we wanted to use the sqrtgun from Figure 8.47, for example, then we would have to insert a universal regulator in between.

[42] This loaf-pulling reaction is called a **tractor beam**, and it can be used to create many other patterns whose populations grow in strange ways—see Exercise 8.47.

Free PDF. See conwaylife.com/book for pattern files and print version.

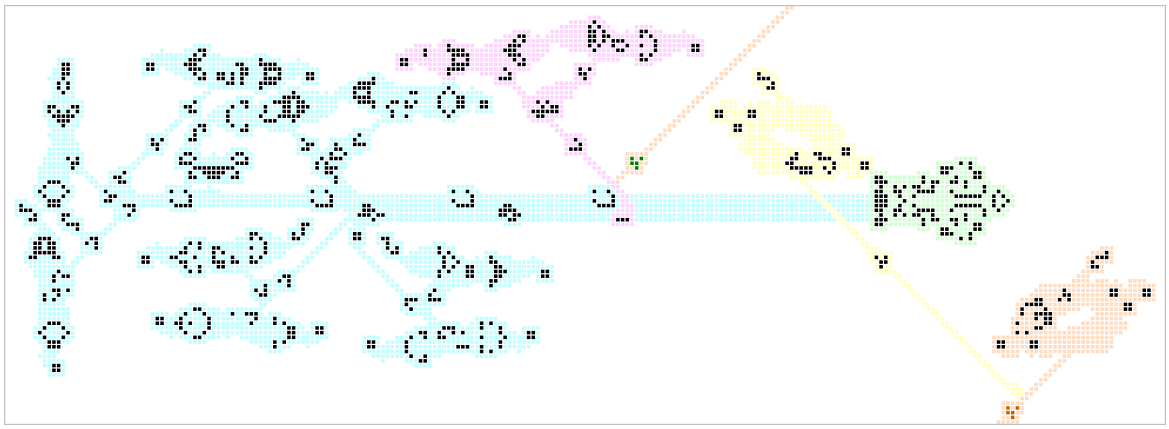


Figure 8.49: A **recursive filter** that receives input gliders from a gun that is aligned to p120 and emits output gliders much more slowly. The guns highlighted in aqua on the left create a stream of LWSS pairs that are blocked by the central gun highlighted in magenta, while the p120 gun highlighted in yellow similarly blocks the output gun at the southeast. When an input glider is received from the northeast, it blocks the magenta gun, thus letting an LWSS pair proceed to the east and collide with the unnamed $c/3$ spaceship highlighted in green, creating a loaf. Future LWSS pairs then pull this loaf westward so that the yellow stream is eventually blocked by that loaf, allowing an output glider (highlighted in orange) to escape to the southwest. Constructed by Alexey Nigin in July 2015.

Indeed, we see in Figure 8.50 that after 1 920 generations the first LWSS pair has created the first loaf, which is $6 \times 104 = 624$ cells away, and the first output glider finally appears at exactly generation

$$19\,471\,113\,219\,505\,603\,606\,989\,361\,234\,575\,175 \approx 1.947 \times 10^{34}.$$

To analyze when future gliders appear, we just focus on the asymptotic behavior of this recursively filtered caber tosser. Using the same argument as in the previous paragraph, if the $c/3$ spaceship is $2n$ cells away from the recursive filter when the first pair of lightweight spaceships is released, it will take $\Theta(2^n)$ generations for the recursive filter to release its first output glider. However, during that time while we waited for the first output glider, the $c/3$ spaceship also moved out to a distance of $\Theta(2^n)$ cells away from the recursive filter. We thus now have to wait for the caber tosser to release $\Theta(2^n)$ more input gliders before the recursive filter releases its second output glider, which takes $\Theta(2^{2^n})$ generations. Indeed, the pattern in Figure 8.50 doesn't release its second output glider until approximately generation

$$480 \times 2^{19471132195056036069893612345751 75/6+1},$$

which is a number with roughly 10^{33} digits—just writing down the decimal expansion of such a number would use more computer storage space than exists on the entire planet.

Repeating this argument shows that the third output glider takes $\Theta\left(2^{2^{2^n}}\right)$ generations to produce, and in general the k-th output glider takes

$$\Theta\Big(\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}}_{k \text{ levels}}\Big)$$

generations to produce. It follows that the population of this pattern in generation t grows asymptotically more slowly than $\log(t)$, $\log(\log(t))$, $\log(\log(\log(t)))$, and so on, regardless of how many times the logarithm is composed with itself. Its exact asymptotic growth rate is $\Theta(\log^*(t))$, where

Free PDF. See conwaylife.com/book for pattern files and print version.

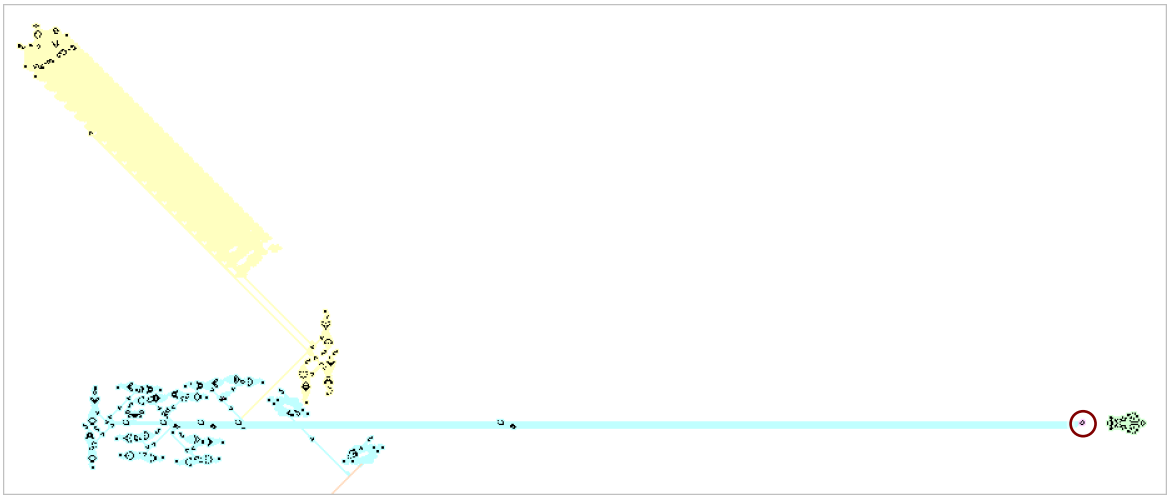


Figure 8.50: If we use a caber tosser (highlighted in yellow) as input to a recursive filter (highlighted in aqua), the result is a gun whose population grows extraordinarily slowly. The first glider from the caber tosser creates a loaf (circled in red) 624 cells away from the recursive filter, thus requiring $624/6 = 104$ additional input gliders from the caber tosser before the first output glider is created on the path highlighted in orange—this takes more than 10^{34} generations.

$\log^*$ is the **iterated logarithm** that counts how many times the (base 2) logarithm must be applied to a number to get a result no larger than 1:[43]

$$\log^*(t) = \begin{cases} 0 & \text{if} \quad t \le 1, \\ 1 + \log^*(\log(t)) & \text{if} \quad t > 1. \end{cases}$$

As if that weren't slowly growing enough, we could even use recursive filters in series to create guns that fire gliders even more slowly. For example, if we attach a second recursive filter to the output of the gun from Figure 8.50, we get a pattern whose population in generation t is $\Theta(\log^{**}(t))$, where $\log^{**}$ is the **doubly iterated logarithm** that counts how many times the iterated logarithm $\log^*$ must be applied to get a result no larger than 1:[44]

$$\log^{**}(t) = \begin{cases} 0 & \text{if} \quad t \le 1, \\ 1 + \log^{**}(\log^*(t)) & \text{if} \quad t > 1. \end{cases}$$

Such a gun won't emit even a single glider until somewhere around generation $2^{2^{\cdot^{\cdot^{2}}}}$, where there are roughly 10^{34} levels in that power tower.

8.8.4 Fermat Primes and Unknown Growth

We now go one step beyond constructing patterns that just grow slowly, and construct a pattern for which we do not even know whether its population grows without bound at all. That is, we construct a pattern whose population might tend to infinity as time goes on, or might stay bounded below some fixed value as time goes on, and we do not know how to determine which of these two possibilities is correct.

[43] Equivalently, $\log^*(t)$ is the number of levels needed in a power tower of 2's to get a result at least as large as t, just like $\lceil \log(t) \rceil$ is the number of times 2 must be multiplied by itself to get a result at least as large as t.

[44] To give a rough idea of how slowly this function grows, we note that if t is an integer with a trillion digits then $\log^{**}(t) = 4$.

Free PDF. See conwaylife.com/book for pattern files and print version.

One way to construct such a pattern is to encode an unsolved mathematical problem into the evolution of the pattern. Since there are numerous unsolved problems involving prime numbers, we use the prime-number-generating gun that we constructed in Figure 6.22 as our starting point. For example, it is currently unknown whether or not there are infinitely many **Fermat primes**: prime numbers of the form $2^n + 1$, where n is a non-negative integer. In fact, the only known Fermat primes are 3, 5, 17, 257, and 65 537, corresponding to $n = 1$, 2, 4, 8, and 16,[45] and it is known that these are the only Fermat primes below

$$2^{2^{33}} \approx 9.63 \times 10^{2\,585\,827\,972}.$$

In order to tweak the primer so that it computes just Fermat primes (rather than all primes), we can use the caber tosser. For example, we could use some p240 guns to destroy each LWSS that the primer outputs, but then aim the caber tosser so as to block those p240 guns, thus allowing lightweight spaceships corresponding to numbers of the form $2^n + 1$ to pass (see Exercise 8.39). The result would be a gun that fires lightweight spaceships whose positions correspond to Fermat primes.

However, the population of such a gun would still tend to infinity regardless of how many Fermat primes exist, since the primer itself continues to construct more and more Gosper glider guns as time goes on. In order to get around this problem, we rig this Fermat prime calculator to self-destruct if a sixth Fermat prime is ever found. In particular, in that case we have the calculator send a signal to the front of the primer that causes an explosion to block it from expanding any farther.

One mild hiccup that arises when constructing such a pattern is that the primer travels at a speed of $c/2$ to the north and to the east, so the self-destruct signal that we send cannot be a spaceship (recall from Theorem 4.1 that no spaceship travels faster than this). Instead, we use wickstretchers to create wicks along the sides of the primer, and we send the self-destruct signal in the form of a fuse burning through those wicks at a speed exceeding $c/2$. One configuration that works, displayed in Figure 8.51, makes use of the $4c/5$ fuse from Exercise 4.28 and the beehive wickstretcher from Exercise 4.29. This pattern grows without bound if there are only 5 Fermat primes, whereas it has a bounded population otherwise.

For spacing and timing reasons, this particular pattern starts with the Fermat prime 5 (not 3), and gliders corresponding to the Fermat primes 5, 17, 257, and 65 537 each destroy a single tub near the southwest corner of the pattern. If a sixth Fermat prime is ever found, the resulting glider is fed into a glider duplicator that ignites both fuses and leads to the primer's self-destruction. However, such self-destruction will not happen until at least generation

$$120 \times 2^{2^{33}} \approx 1.16 \times 10^{2\,585\,827\,975},$$

so evolving this pattern via computer software is not actually an effective method of checking whether or not a sixth Fermat prime exists.

While it is not possible to run this pattern long enough to see it self-destruct, we can force its self-destruction by manually erasing one or more of the glider-absorbing tubs at its southwest corner. If we erase all 4 of those tubs, the glider corresponding to the Fermat prime 5 almost immediately triggers the primer's self-destruction, whereas deleting a smaller number of tubs delays its self-destruction somewhat. If we remove just 1 tub, the self-destruct fuse is ignited after roughly 8 million generations (by the glider corresponding to the Fermat prime 65 537) and the primer finally stabilizes after roughly 21 million generations.

[45] It is known that if $2^n + 1$ is prime then n must be a power of 2, so Fermat primes must actually be of the form $2^{2^n} + 1$.

Free PDF. See conwaylife.com/book for pattern files and print version.

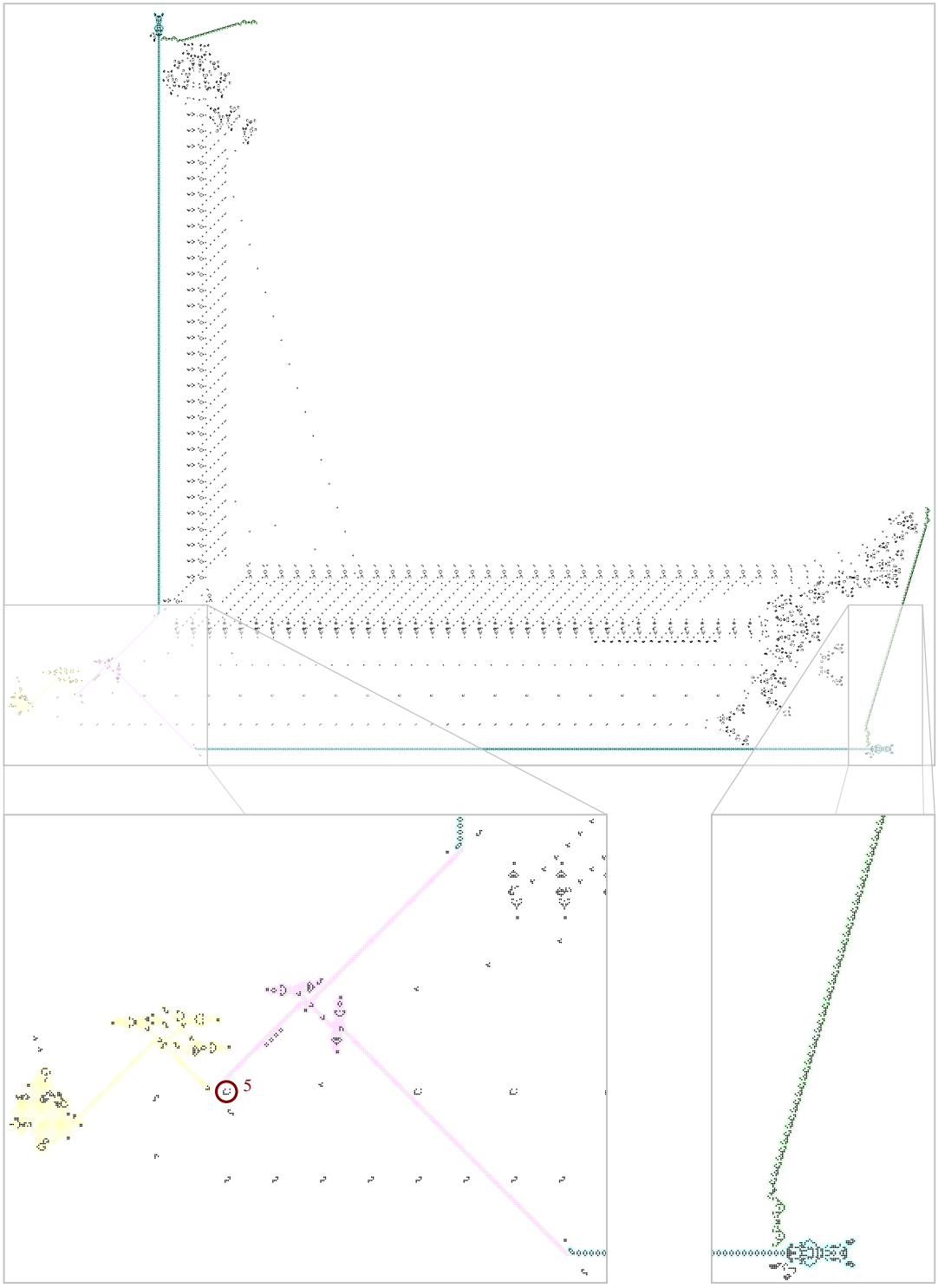


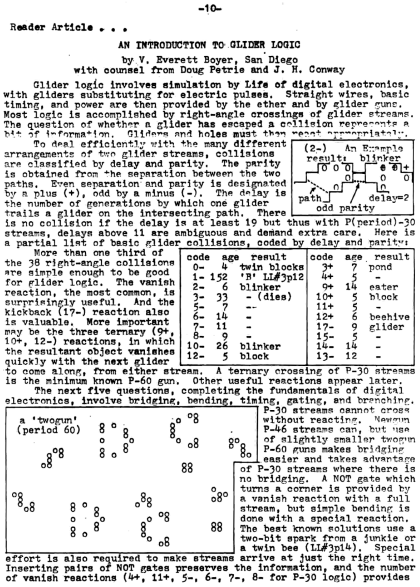
Figure 8.51: A Fermat prime calculator that self-destructs and has bounded population if a sixth Fermat prime exists, but grows without bound otherwise. A caber tosser (highlighted in yellow) fires gliders at the lightweight spaceships that the primer creates corresponding to integers of the form 2^n+1. If that LWSS is present (i.e., 2^n+1 is prime) then the glider is reflected and destroys one of the four tubs to its northeast (they are destroyed by the known Fermat primes 5, 17, 257, and 65 537). If another Fermat prime is found, the glider goes through a glider duplicator (highlighted in magenta) and ignites two $4c/5$ beehive wicks that are laid by beehive puffers (highlighted in aqua). Those wicks eventually cause two extensible $c/2$ spaceships (highlighted in green) to explode, blocking the primer.

Free PDF. See conwaylife.com/book for pattern files and print version.

8.9 Notes and Historical Remarks

Many of the ideas discussed early in this chapter were investigated right from the early days of Life. For example, many of the glider deletion tricks of Section 8.1 were already known in the early 1970s and were used to create the same period 60 gun that we constructed in Figure 8.2(a)—see Figure 8.52. Indeed, the idea of encoding information in gliders and then using collisions between gliders to encode logical operations was already well-established at that time, and many useful interactions of this type were simply found by hand.

Other than guns resulting from these simple glider logic tricks though, not many new guns were found after the p30 Gosper glider gun and the p46 twin bees gun in the years following 1971. Bill Gosper made a p1100 gun sometime around 1984, Dean Hickerson constructed a p94 gun in 1990, and David Buckingham had constructed guns of period 44 and $136+8n$ ($n \geq 1$), mostly based on his early work with Herschel tracks, by the early 1990s. However, progress in this area was relatively stagnant until he revealed Herschel track technology in full in 1996, and most of the earlier guns were so large and unwieldy that they were made almost instantly obsolete at that time.



-10-

Reader Article . . .

AN INTRODUCTION TO GLIDER LOGIC

by V. Everett Boyer, San Diego
with counsel from Doug Petrie and J. H. Conway

Glider logic involves simulation by Life of digital electronics, with gliders substituting for electric pulses. Straight wires, basic timing, and power are then provided by the ether and by glider guns. Most logic is accomplished by right-angle crossings of glider streams. The question of whether a glider has escaped a collision represents a bit of information. Gliders and holes must then react appropriately.

To deal efficiently with the many different arrangements of two glider streams, collisions are classified by delay and parity. The parity is obtained from the separation between the two paths. Even separation and parity is designated by a plus (+), odd by a minus (-). The delay is the number of generations by which one glider trails a glider on the intersecting path. There is no collision if the delay is at least 19 but thus with P(period)-30 streams, delays above 11 are ambiguous and demand extra care. Here is a partial list of basic glider collisions, coded by delay and parity:

(2-) An Example
result: blinker
path
delay=2
odd parity

code	age	result	code	age	result
0-	4	twin blocks	9+	7	pond
1-	152	'B' LL#3p12	4+	5	-
2-	6	blinker	9+	14	eater
3-	33	- (dies)	10+	5	block
5-	7	-	11+	5	-
6-	14	-	12+	6	beehive
7-	11	-	17-	9	glider
8-	9	-	15-	9	-
10-	26	blinker	16-	14	-
12-	5	block	13-	12	-

More than one third of the 38 right-angle collisions are simple enough to be good for glider logic. The vanish reaction, the most common, is surprisingly useful. And the kickback (17-) reaction also is valuable. More important may be the three ternary (9+, 10+, 12-) reactions, in which the resultant object vanishes quickly with the next glider to come along, from either stream. A ternary crossing of P-30 streams is the minimum known P-60 gun. Other useful reactions appear later.

The next five questions, completing the fundamentals of digital electronics, involve bridging, bending, timing, gating, and branching.

a 'twogun' (period 60)

P-30 streams cannot cross without reacting. Newsom, P-46 streams can, but use of slightly smaller twogun P-60 runs makes bridging easier and takes advantage of P-30 streams where there is no bridging. A NOT gate which turns a corner is provided by a vanish reaction with a full stream, but simple bending is done with a special reaction. The best known solutions use a two-bit spark from a junkie or a twin bee (LL#3p14). Special effort is also required to make streams arrive at just the right time. Inserting pairs of NOT gates preserves the information, and the number of vanish reactions (4+, 11+, 5-, 6-, 7-, 8- for P-30 logic) provides

Figure 8.52: A page from the September 1973 issue (Volume 11) of *Lifeline* that discussed glider logic and how it can be used to make a p60 glider gun (shown at the bottom left).

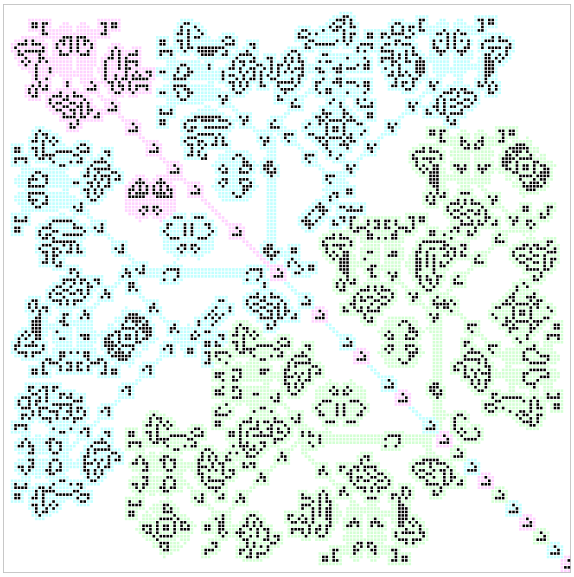


Figure 8.53: A tightly packed p16 glider gun that uses a p48 gun (highlighted in magenta) and two p48 glider insertions (highlighted in aqua and green).

After Herschel tracks hit the mainstream, which allowed for the systematic creation of glider guns of arbitrary large periods, a collaborative effort among many Life enthusiasts between then and 2003 (initiated by Dietrich Leithner and Peter Rott, and mainly led by Jason Summers and Karel Suhajda) produced explicit glider guns of all periods from 14 to 1000. The guns in this collection have since been shrunk down considerably via newer technology like Snarks and syringes, and at least a few of them are typically updated whenever a new sparky oscillator is discovered (as long as that oscillator can be used as a filter or reflector, anyway).[46]

As a result of these repeated optimizations, most of the guns in this collection are now very cleverly and tightly packed—much moreso than the p14 gun that we constructed in Figure 8.18.

[46] This collection of small glider guns is now being maintained by Adam P. Goucher as part of the Catagolue project. Up-to-date links to the smallest known guns of each period can be found at catagolue.hatsya.com/census/b3s23/synthesis-costs/gun.

Free PDF. See conwaylife.com/book for pattern files and print version.

Indeed, that gun had lots of empty whitespace that could be reduced by customizing the four glider insertion mechanisms that it uses, rather than using the exact same track for all four of them. For example, a particularly compact p16 gun is displayed in Figure 8.53.[47] This gun uses a small p24 glider gun (one that is slightly smaller and newer than the one we saw in Figure 8.22(a)), which is then filtered to p48 by Rich's p16, and then sped up to p16 via two copies of the glider insertion reaction from Figure 8.10(b).

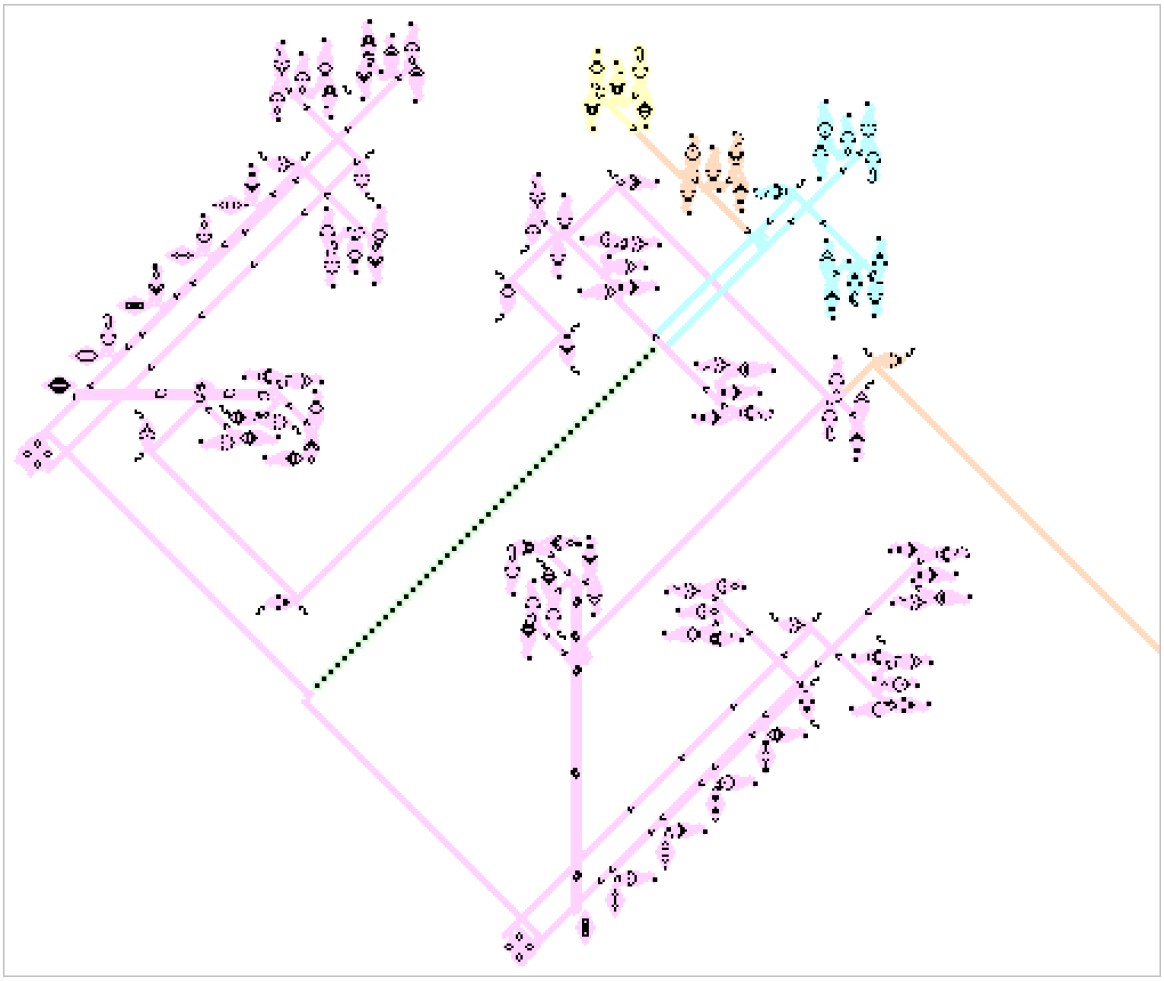


Figure 8.54: A **quadratic filter**, which reduces the growth rate of a gun quadratically. It works by using a pair of gliders (highlighted in aqua) to pull a block from an ever-receding trail (highlighted in green), similar to the loaf-pulling reaction used in the recursive filter of Figure 8.49. The rest of the device (highlighted in magenta) extends the back end of the block trail. Here, the filter is attached to a p90 glider gun (highlighted in yellow), resulting in a sqrtgun. Constructed by Gabriel Nivasch in June 2006.

Patterns that decrease the asymptotic growth rate of a gun were known prior to the recursive filter of Figure 8.49, but their effects were less pronounced. The first such pattern was a **quadratic filter**, which is displayed in Figure 8.54. This pattern produces its n-th output glider after receiving its $n(n+1)/2$-th input glider,[48] and thus decreases the growth rate of glider guns quadratically. In particular, appending this pattern to a gun whose population in generation t is $\Theta(f(t))$ results in a gun whose population is $\Theta(\sqrt{f(t)})$. For example, if we append n of these quadratic filters in series after a

[47] However, this is not quite the smallest known p16 gun—see Exercise 8.42.

[48] This device relies on p90 circuitry, so the input gliders have to be aligned to that period.

Free PDF. See conwaylife.com/book for pattern files and print version.

gun whose population grows linearly (i.e., a "standard" glider gun), we get a gun whose population is $\Theta(t^{1/2^n})$.

The quadratic filter works by performing a block-pulling reaction whenever an input glider is received. Once a certain threshold is passed, the block is deleted, an output glider is released, and a farther-away block is created. If we rearrange the circuitry so that the location of the new block is pushed away *every* time an input glider is received, rather than only when an output glider is produced, the filtering becomes even more pronounced. In fact, we get an **exponential filter**—a pattern that can be attached to the output of a glider gun so as to decrease its growth rate exponentially.

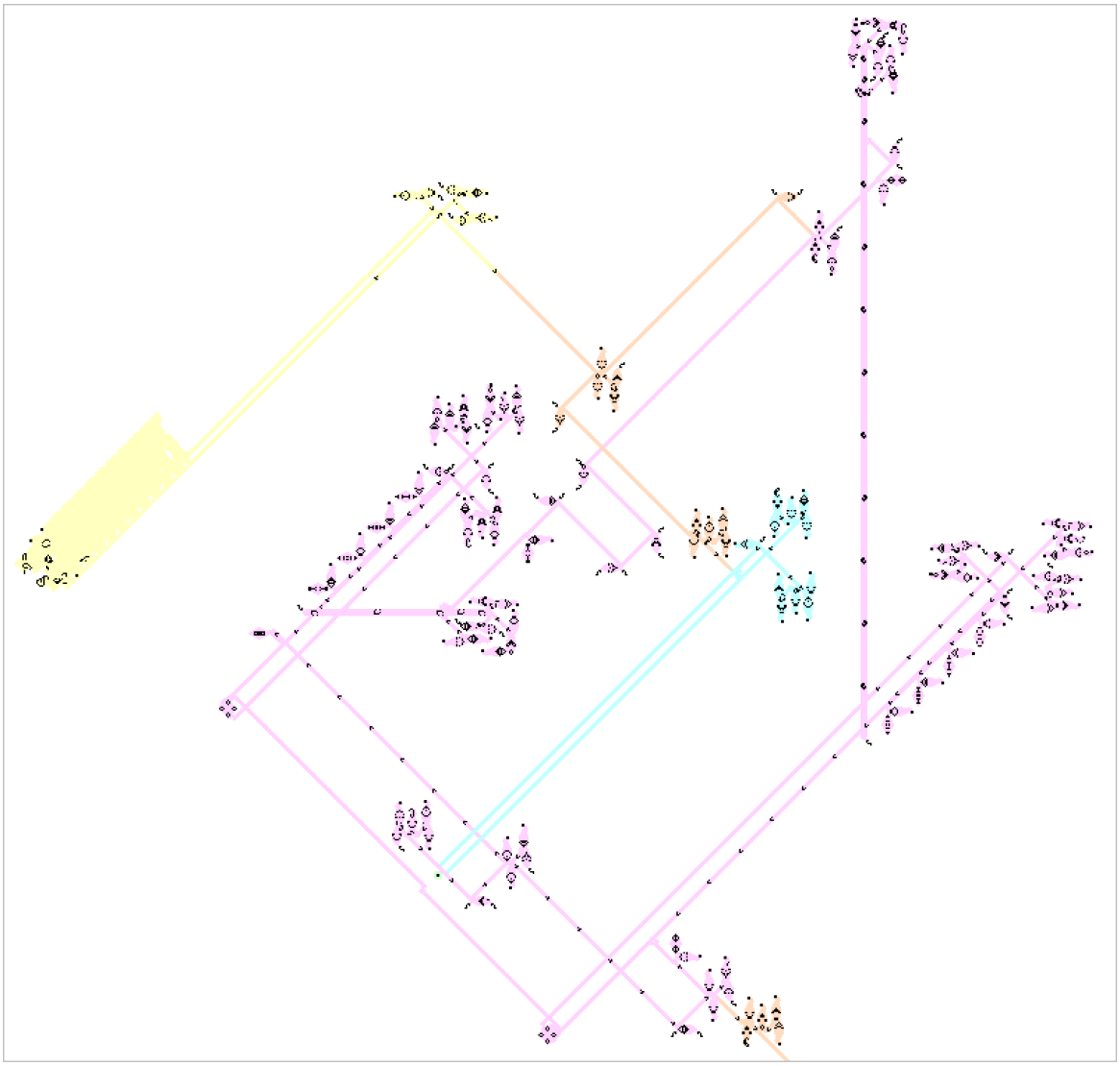


Figure 8.55: An **exponential filter**, which reduces the growth rate of a gun exponentially, attached to a caber tosser (highlighted in yellow) so that its population in generation t is $\Theta(\log(\log(t)))$. It works via most of the same mechanisms as the quadratic filter of Figure 8.54, and was also constructed by Gabriel Nivasch in June 2006.

The particular exponential filter that is displayed in Figure 8.55 produces its n-th output glider after receiving $5 \times 2^n - 4$ input gliders, so appending it to a gun whose growth rate is $\Theta(f(t))$ results in a gun with the slower growth rate $\Theta(\log(f(t)))$. For example, feeding the output of a caber tosser into this exponential filter, as in Figure 8.55, results in a gun whose population in generation t is

Free PDF. See conwaylife.com/book for pattern files and print version.

$\Theta\big(\log(\log(t))\big)$.

We could also apply several of these filters in sequence, producing guns with population

$$\Theta\big(\log(\log(\cdots(\log(t)))))\big),$$

where the logarithms are nested as many times as we like. However, even just a single usage of the recursive filter as in Figure 8.50 results in a slower gun.

Exercises

solutions to starred exercises on page 463

∗**8.1** [1/5] The glider collision in Figure 8.1 can only be used to double the period of glider streams of period 29 or higher. Use the collision in Table 5.1 that produces an eater 1 to double the period of a period 28 glider stream.

8.2 [1/5] We used the reaction in Figure 8.1 to turn two Gosper glider guns into a period 60 gun. Use that reaction multiple times to turn four Gosper glider guns into a period 120 gun.

∗**8.3** [2/5] Use the reaction from Figure 8.5(b) to create a period 322 glider gun.

8.4 The stream inverter-and-duplicator from Figure 6.9 is quite useful in situations where we want to duplicate a glider stream that took a lot of effort to create.

- **(a)** [1/5] Use three (not four!) Gosper glider guns and a single extra block to create a gun that fires two p60 streams of gliders.
- **(b)** [2/5] Use four (not six!) Gosper glider guns and two extra blocks to create a gun that fires three p60 streams of gliders.

∗**8.5** [2/5] Use Rich's p16 (and potentially other sparkers as well) as a filter to turn the true-period 20 glider gun into a true-period 80 glider gun.
[Hint: If you're lucky, you can get a single copy of Rich's p16 to delete *two* consecutive gliders.]

∗**8.6** [2/5] We saw in Figure 8.14(a) that a filter can be used to double the period of any stream of lightweight spaceships with period $4n+2$ ($n \geq 3$). Explain why there cannot exist a filter capable of doubling the period of every stream of *gliders* with period $4n+2$ ($n \geq 3$).

8.7 [2/5] Find an oscillator that can be used to double the period of any LWSS stream with period $6n+3$ ($n \geq 2$).

8.8 [4/5] Recall the reactions from Figures 8.15 and 8.16 that can be used to create p18 xWSS streams.

- **(a)** Use (potentially multiple copies of) the LWSS insertion reaction to create a p18 LWSS gun.
- **(b)** Use the LWSS-to-MWSS reaction to turn your LWSS gun from part (a) into a p18 MWSS gun.
- **(c)** Use the MWSS-to-HWSS reaction to turn your MWSS gun from part (b) into a p18 HWSS gun (which is the smallest period possible for such a gun!).

8.9 [4/5] The conduit from Figure 8.12 can be used to create smaller stable glider-to-LWSS and glider-to-MWSS converters than we have seen so far.

- **(a)** Use that conduit to build a smaller stable glider-to-LWSS converter than the one that we constructed in Figure 7.12(b).
  [Hint: Be careful—do not use periodic reflectors like we did in Figure 8.12, and make sure that the glider collides with the *same* Herschel, rather than the *next* one like in that figure.]
- **(b)** Use that conduit to build a stable glider-to-MWSS converter.

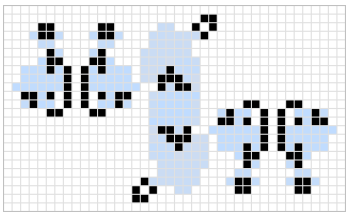
8.10 [3/5] Adjust the reflectors in the gun from Figure 8.17 to create a gun that emits 2 out of 3 gliders in a p29 (instead of p28) stream.

8.11 [2/5] Adjust the reflectors in Figure 8.12(a) to make the reaction work with Herschel streams of period 180 (instead of 156).

8.12 [2/5] Show how to stabilize Jason's p36 (see Figure 8.20(a)) with caterers instead of jams.

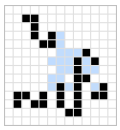
Free PDF. See conwaylife.com/book for pattern files and print version.

∗**8.13** A common method of creating a true-period glider gun is to replace a stabilizing component of a hassler, whose purpose is to absorb some debris, with another one that converts that debris into a glider. Consider the following period 21 B-heptomino hassler:[49]



(a) [1/5] If one of its boats is removed, what common object forms at the edge of the oscillator, causing it to self-destruct?

(b) [3/5] Show how the following period 7 oscillator can be used to convert the object from part (a) into a glider, thus creating a true-period 21 glider gun:



(c) [2/5] Create a true-period 42 glider gun.

8.14 [2/5] Construct a true-period 56 gun that is smaller than the one from Figure 8.32(e).
[Hint: Just add a filter to some other gun.]

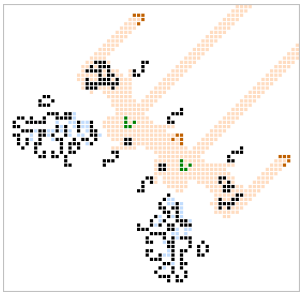
8.15 [4/5] By using (multiple copies of) the p28 glider gun from Figure 8.23, construct a period 14 glider gun that is smaller than the one that we built in Figure 8.18.

8.16 [3/5] The p56 glider gun displayed in Figure 8.32(e) uses three reflectors at its top and at its bottom, instead of just one, to make its bounding box slightly smaller. You can similarly "fold in" the top and bottom corners of some other guns to make their bounding boxes smaller.

(a) Modify the p54 glider gun from Figure 8.32(c) so that its bounding box has at least 300 fewer cells in it.

(b) Modify the p55 glider gun from Figure 8.32(d) so that its bounding box has at least 700 fewer cells in it.

8.17 [3/5] The following reaction converts 2 input gliders into 3 output gliders:[50]



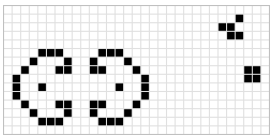
(a) Use (potentially multiple copies of) this reaction to create a true-period 51 gun.

(b) Replace the p3 sparkers with the following p7 sparker, and then create a true-period 49 gun:



8.18 [1/5] Determine the repeat times of the four conduits from Figure 8.33.

∗**8.19** [3/5] Another method of stabilizing one side of the pi-heptomino reaction used in the true-period 44 and 50 glider guns is displayed below.



Use this reaction to create a true-period 50 glider gun that is different than the one from Figure 8.26. In particular, use two mirror-image copies of the glider-extracting conduit and then feed one of those gliders back into the reaction via...

(a) two Snarks.

(b) two bumpers (what is the only bumper period that we have seen that works with period 50 mechanisms?).

8.20 [2/5] Show how to extract a glider from an Fx77 conduit like we did in Figure 8.29 via the following oscillators:

(a) a unix (p6),

(b) a pentadecathlon (p15), and

(c) Beluchenko's p7 (Figure 3.13(c)).

[49] Found by David Raucci in August 2021. The period 21 gun that results from this exercise was found by Tanner Jacobi later on the same day.

[50] This base reaction was found by "hotcrystal0" and "wwei23" in November 2021, and the resulting p51 and p49 guns were first constructed shortly thereafter by Luka Okanishi and Tanner Jacobi, respectively.

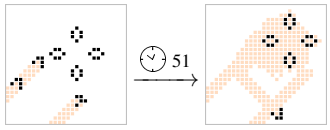
Free PDF. See conwaylife.com/book for pattern files and print version.

8.21 [2/5] Explain why the method of separating the two close glider streams in the p61 gun from Figure 8.34 (by reflecting one of the gliders off of the corner of an L112 conduit) does not work at p58, p59, or p60, despite L112 having a repeat time of 58 generations.

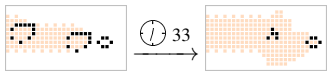
8.22 [3/5] Use the reaction from Figure 8.38(d) to create a slide gun that fires gliders separated by 2 lanes (instead of 4 lanes, like the slide gun from Figure 8.39).

8.23 [3/5] Slide guns can be made to work by pushing any stable object, not just a block. Use the following reactions[51] to create slide guns:

(a)

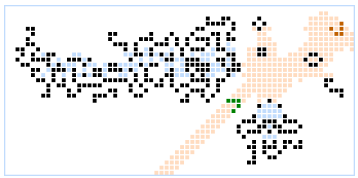


(b)



8.24 [3/5] Use the small p120 glider gun from Exercise 7.27 to rebuild the slide gun from Figure 8.39 within an 85×70 bounding box.

8.25 The device below[52] can be attached to the output of any gun with period $5n$ ($n \geq 8$) to create an edge-shooting gun of the same period:



(a) [3/5] Use this reaction to create an edge-shooting p240 gun.

(b) [4/5] Use 10 copies of the gun from part (a) (and perhaps the p120 glider gun from Exercise 7.27) to rebuild the slide gun from Figure 8.40 in a much smaller way. In particular, this edge shooter lets you avoid using the glider insertion mechanism from Figure 8.8(b).

8.26 [4/5] Create a slide gun that fires gliders separated by just 1 lane.
[Hint: This is tricky! The reactions like Figure 8.38(d) that move a block 1 cell diagonally result in slide guns with a separation of 2 lanes.]

8.27 [2/5] Create a "slide" gun that fires 7 gliders at a block so as to create a perpendicular glider without moving the block at all. That is, create a slide gun that doesn't slide.

8.28 [4/5] Create a slide gun that fires 3 gliders on each of its output lanes (instead of 2, like in Figure 8.40).
[Hint: Using an edge-shooting gun like the one from Exercise 8.25 will help keep the size of your slide gun down.]

8.29 [3/5] Decrease the bounding box of the four-lane gun from Figure 8.43 so that it is no larger than 275×275, by using Snarks to wrap the glider loops around like we did in Figures 8.45 and 8.46.

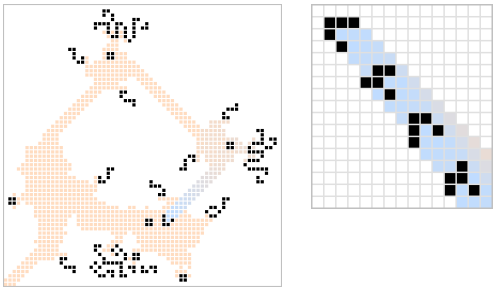
8.30 Armless glider production is a useful and highly adjustable mechanism, but some adjustments have consequences that may not be intuitively obvious.

(a) [2/5] What behavior appears if you try to produce perpendicular output gliders via two loop guns whose periods differ by a multiple of 8?

(b) [4/5] Make a design for a "drifting" armless gun that would allow loop guns of different periods, but avoid the problem that was alluded to in part (a).

8.31 In Section 8.7, we only constructed armless guns that fire slow salvos. However, we can also construct ones that fire synchronized salvos.

(a) [4/5] Use (several copies of) the p387 loop gun displayed below on the left, along with the H-to-2G conduit that we used in our tee-based armless guns, to create a 4-consecutive-lane armless gun that fires the tight arrangement of gliders displayed below on the right.



(b) [2/5] Adjust your gun from part (a) so that it fires that configuration of gliders with period 129.
[Hint: $129 \times 3 = 387$.]

(c) [3/5] Adjust your gun from part (a) so that it fires that configuration of gliders with period 87 (which is minimal).
[Hint: $387 + 6 \times 8 = 435 = 87 \times 5$, so first adjust each loop to be 6fd bigger.]

[51] Both found by Jason Summers in 1999. The reaction in part (a) was used in the quadratic and exponential filters of Figures 8.54 and 8.55.

[52] Found by Arie Paap and Tanner Jacobi in late 2018 in a period $30n$ ($n \geq 2$) form, and adjusted to the period $5n$ form shown here in late 2019.

Free PDF. See conwaylife.com/book for pattern files and print version.

8.32 In this exercise, we explore some methods that can be used to decrease the repeat time of an armless gun.

- **(a)** [3/5] The tee-based armless Cordership gun of Figure 8.45 has already been adjusted to close to its recipe's minimum repeat time of 2061 generations. Explain how additional loop guns could be added to allow a repeat time of less than 2000 generations.
- **(b)** [5/5] How could the repeat time of this gun be reduced *without* increasing the number of loop guns?

8.33 In a tee-based armless gun that implements 2-directional slow synthesis, the gliders in the loop guns are synchronized in groups of four. Adjustments to the four gliders in a group can make a collision happen in a different place, or at a different time.

- **(a)** [2/5] Find a case where a glider follows another one by just 74 generations in the tee-based armless Cordership gun of Figure 8.45. Then find the other three gliders that are synchronized with that glider, and delay all four gliders by one generation. What happens?
- **(b)** [2/5] Return the four gliders to their original positions, and instead of delaying them all, advance them all by one generation. What happens?
- **(c)** [3/5] Find the four gliders that collaborate to produce the southernmost block in the Cordership seed. How would you move those gliders to build the block two cells directly south of its current position, or one cell directly southeast or southwest?

8.34 It is somewhat easier to change the period of a multicolor armless gun than the monochrome one, since they have fewer glider loops to adjust. The multicolor armless Cordership gun of Figure 8.46 has period 5603.

- **(a)** [3/5] A syringe followed by NW31 (i.e., the conduit of Figure 7.5(b)) makes a stable glider reflector. Add one of these reflectors to one of the loop guns from the multicolor armless Cordership gun. How does its period change?
  [Hint: Start by moving some Snarks a long distance away to make space.]
- **(b)** [3/5] Adjust that glider loop so that it has period $2^{13} = 8192$.
  [Hint: You cannot just adjust the trombone slides, since the mod-8 length of that loop is not right. Either add more syringe + NW31s or add another reflector from Table 7.2.]
- **(c)** [4/5] Make the same adjustments to the other three loop guns to produce a working period 8192 Cordership gun.

8.35 [2/5] Explain why we cannot rewind the caber tosser from Figure 8.48(a) so that the triggering glider in aqua is traveling northwest before it is reflected by the caber tosser.

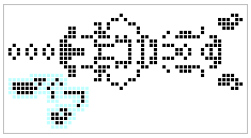
8.36 [3/5] Construct a caber tosser that uses the same Cordership and glider bouncing off of it as in Figure 8.48(a), but uses stationary circuitry (e.g., syringes and Herschel conduits) in place of the glider guns.

8.37 [2/5] If a caber tosser makes use of a receding spaceship with speed s_1 (instead of a Cordership) and bouncing spaceship with speed s_2 (instead of a glider), what is the ratio of the gaps between the gliders that it emits? For example, in the caber tosser of Figure 8.48(a) we had $s_1 = c/12$, $s_2 = c/4$, and a ratio of 2 (each glider took twice as long to be emitted as the previous one).

8.38 [4/5] Rebuild the sqrtgun from Figure 8.47 without any Herschel tranceivers—use syringes (and other "modern" Herschel conduits) instead.

8.39 [2/5] Construct a gun that emits lightweight spaceships corresponding to Fermat primes (much like the primer itself emits lightweight spaceships corresponding to all primes), rather than a pattern that self-destructs if it finds a sixth Fermat prime like the one we made in Figure 8.51.

8.40 [2/5] The beehive puffer that is used at the bottom-right and top-left of the Fermat prime calculator (see Figure 8.51) is displayed below.



This puffer still works even if the cells highlighted in aqua are deleted. What is their purpose in the Fermat prime calculator?

8.41 [3/5] Modify the Fermat prime calculator of Figure 8.51 so that it computes **Mersenne primes**: primes of the form $2^n - 1$ for some integer n.

8.42 Recall the small p16 oscillator from Figure 3.43 called Rob's p16.

- **(a)** [2/5] Show how Rob's p16 can be used to filter a p24 LWSS stream, creating a p48 stream.
- **(b)** [2/5] What LWSS stream periods can be filtered by Rob's p16?
- **(c)** [4/5] Use the filtering reaction from part (a) to reduce the bounding box of the p16 glider gun that we saw in Figure 8.53 by at least 2 rows or columns.

8.43 [3/5] Construct a glider gun whose population in generation t is $\Theta(t^{1/4})$.

Free PDF. See conwaylife.com/book for pattern files and print version.

8.44 [3/5] Attach a quadratic filter to the output of a caber tosser. What is the growth rate of the population of the resulting gun?

8.45 [4/5] Attach an exponential filter to the output of a sqrtgun. What is the growth rate of the population of the resulting gun? Is it the same as the growth rate of the gun constructed in Exercise 8.44?
[Hint: Be careful—the exponential filter of Figure 8.55 is aligned to period 90 and thus cannot accept all input gliders from the sqrtgun of Figure 8.47. Either use a universal regulator between these two mechanisms or use a different sqrtgun.]

8.46 [5/5] Construct a **cubic filter**: a pattern that can be appended to a glider gun whose population in generation t is $\Theta(f(t))$ so as to get a gun whose population is $\Theta(\sqrt[3]{f(t)})$.

8.47 [3/5] A pattern whose population grows without bound, but also falls below some fixed value infinitely often, is called a **sawtooth**.[53]

(a) How can you modify the quadratic filter from Figure 8.54 to make a sawtooth?

(b) Use the loaf-pulling tractor beam from the recursive filter in Figure 8.49 to create another sawtooth.

[53] This name comes from the shape of the graph of their population versus time, which goes up and down over and over, like a sawtooth wave.

Free PDF. See conwaylife.com/book for pattern files and print version.

III Constructions

9 Universal Computation 271
9.1 A Computer in Life
9.2 A Compiled APGsembly Pattern: Adding Registers
9.3 Multiplying and Reusing Registers
9.4 A Binary Register
9.5 A Character Printer
9.6 A π Calculator
9.7 A 2D Printer
9.8 Notes and Historical Remarks
Exercises

10 Self-Supporting Spaceships 311
10.1 The Silverfish
10.2 The Caterpillar
10.3 Oblique Helices and the Waterbear
10.4 Caterloopillars
10.5 Notes and Historical Remarks
Exercises

11 Universal Construction 345
11.1 Gemini and Geminoids
11.2 Single-Channel Glider Synthesis with a 90-Degree Elbow
11.3 Single-Channel Glider Synthesis with a Zero-Degree Elbow
11.4 Duplicating and Reflecting Single-Channel Recipes
11.5 A Slow Demonoid
11.6 A Middling Demonoid
11.7 A Fast Demonoid
11.8 Notes and Historical Remarks
Exercises

12 The 0E0P Metacell 385
12.1 Other 2D Cellular Automata
12.2 Rule Emulation
12.3 Overview of the Metacell
12.4 The Shell and Lane-Switching Circuitry
12.5 The Kernel
12.6 The Nucleus
12.7 Construction and Self-Destruction
12.8 A Complete Metageneration
12.9 Notes and Historical Remarks
Exercises

Free PDF. See conwaylife.com/book for pattern files and print version.



9. Universal Computation

Becoming sufficiently familiar with something is a substitute for understanding it.

John H. Conway

At the end of Chapter 6, we briefly introduced the concept of **computational universality**—the ability of Life (or other data manipulation rules in general) to simulate any computation that a computer can accomplish. It has been known since the early days of Life that it is computationally universal (also sometimes referred to as "Turing complete") [Wai74, BCG82]. However, the constructions that were used to originally demonstrate this fact were monstrously large and only *mostly* pieced together. Actually constructing an explicit pattern that works as a universal computer that can be simulated in standard Life software still requires some additional work.

In this chapter, we dive into the details of one particularly flexible computation toolkit[1] that can be used to construct patterns that perform arbitrary computations, and even print the output of those computations in a font made up of blocks. For example, one of the major results of this chapter will be a pattern in Section 9.6 that computes and prints the decimal digits of the mathematical constant π.

9.1 A Computer in Life

What exactly does it mean to build a computer in the form of a Conway's Life pattern, anyway? To create a workable computational model in the form of a Life pattern, we need to construct patterns that implement a few different things:

1) We need mechanisms that can store information, preferably in the traditional binary form—zeroes and ones. We *could* perfectly well break with tradition and build, say, a trinary computer based on memory mechanisms with three possible states, or a native decimal computer based

[1] Developed by Adam P. Goucher in 2009 and 2010 [Gou10].

Free PDF. See conwaylife.com/book for pattern files and print versio

on switches with ten different states, but binary two-state switches are much easier to design and to work with.

2) We need to be able to write programs made up of simple steps, or "instructions". Each step should have a specific effect on data stored in memory, and values stored in memory should be able to affect the flow of the program. After a conditional instruction, for example, if a given memory location contains "1" instead of "0", a completely different instruction might be the next to be executed.

3) Finally, we also need mechanisms that can represent arbitrarily complex programs. We have to store both the individual instruction steps, as well as the order in which they'll be executed (including the conditional "jump" instructions that depend on the current data stored in memory).

All of this may seem like a big leap in complexity from what we have covered so far, but most of the mechanisms that we need have already appeared in previous chapters. Our job in this chapter is to tie these known mechanisms together in the form of a working Life computer.

9.1.1 A Sliding Block Register

The first logic component that we construct will allow us to store and manipulate the value of a non-negative integer. While we could store the value of this integer in binary (and we will do exactly this in Section 9.4), it will be easier for us to simply store it in the position of a single block. Indeed, we can move a block forward and backward via the block-moving operations that we used to make slide guns in Section 8.6, so we can use those operations to adjust the value that is stored in this device.

It is straightforward to use the techniques of Chapter 7 to build a stable conduit that turns a glider into the arrangement of three gliders from Figure 8.37(c) that pushes a block diagonally away by 1 cell. We call this an `INC` operation, since it corresponds to increasing the value that the block represents by 1. We can similarly construct a conduit that turns a glider into the arrangement of two gliders from Figure 8.37(b) that pulls the block diagonally closer by 1 cell—a `DEC` operation that decreases the value of the block by 1.

However, in order for this mechanism to actually be useful in a computer, we need a way of reading its value (i.e., checking where the block is without disturbing it). To this end, we note that it suffices to be able to check whether or not the block is in the "zero" (`Z`) position,[2] since we could repeatedly decrease the value of the block and perform that test after every decrement until we get a positive answer, and then increment the block back to its original position. Once we solve this problem of testing whether or not a block is in the "zero" position, we will have a memory device called a **sliding block register (SBR)**.

The trick that we will use for performing this test is based on the $(2,1)$ block pull reaction that we first saw way back in Figure 2.20. Indeed, this reaction is the one that happens when a glider just barely grazes a block, so we can position that glider so that it performs this $(2,1)$ block pull if the sliding block is in the "zero" position, and it simply passes by the block without any interaction at all if it is in any other position. In the latter case, we can simply use the unaffected glider as the "non-zero" (`NZ`) output of our test-if-zero (`TEST`) circuit. However, if the glider was destroyed by the $(2,1)$ block pull reaction (since the sliding block was in the "zero" position), we have some additional work to do: we have to arrange some circuitry so as to create a "zero" (`Z`) output signal and also restore the moved sliding block to its original "zero" position.

Restoring the moved sliding block is simple enough—we just send another glider from the opposite direction so as to perform the opposite (2,1) block pull. However, we only want to send that glider if

[2] We could equally well call this the "one" position and store positive integers instead of non-negative integers, but it's more traditional to have the block's starting position be zero.

Free PDF. See conwaylife.com/book for pattern files and print version.

we failed to see an NZ output coming out of this TEST circuit. We thus need a second signal, split off from the TEST input, that is suppressed by a NZ output, but otherwise produces a Z output as well as a glider that resets the sliding block to the correct position. This kind of signal-logic thinking takes a while to get used to, but it's less complicated than it sounds. Figure 9.1 displays a schematic that illustrates one way of handling the glider paths needed for this sliding block test-if-zero circuit and its two possible results.

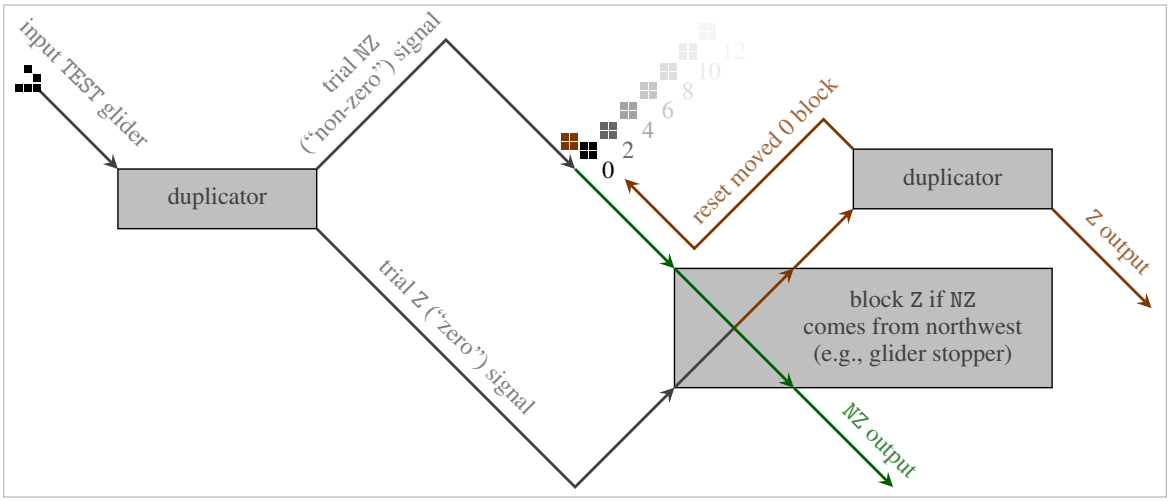


Figure 9.1: A schematic of a TEST circuit that tests whether or not a sliding block is in the "zero" (Z) position. If the block is in a non-zero (NZ) position then the trial NZ glider passes by without affecting it at all. Otherwise, the zero block is shifted via the $(2,1)$ block pull reaction, the trial NZ glider is destroyed, and the Z glider performs another $(2,1)$ block pull to put that zero block back in its original place.

We could then create a complete sliding block register by adding two additional inputs beyond just the TEST input:

- a DEC input that produces two gliders that are aimed at the sliding block as in Figure 8.37(b), so as to pull the block one cell closer, and
- an INC input that produces the three gliders from Figure 8.37(c) (two of which are the same as the gliders produced for the DEC operation), so as to push the block one cell farther away.

A sliding block register with these three inputs could certainly be used in a Life computer, but we would have to be very careful never to write a program that might accidentally send a DEC signal when the block is already at the "zero" position. After all, if that ever happened then the TEST mechanism would fail catastrophically, because the glider that's supposed to graze the block so as to perform the (2,1) block pull reaction would instead run right into it.[3]

Our program could avoid this danger by always calling TEST first, and then only calling DEC if an NZ output comes back from TEST. However, a better option is to instead bake this idea right into the sliding block register itself, rather than relying on the programs that we write to do so. That is, we adjust the design of the sliding block register so that a TEST always happens just before a DEC. The circuit itself will then ignore the DEC input if the block is already at the Z position.

This slight redesign gives us an equally useful (and much safer!) sliding block register that has only two inputs: INC and TEST-then-DEC, which we abbreviate as TDEC. In this register, which is illustrated in Figure 9.2, there is no longer a way to send in a series of inputs that causes a catastrophic

[3] By contrast, INC operations will never cause problems—it's always possible to move the block out by one more step to store the integer $n+1$ instead of n, so only DEC instructions are potentially dangerous.

Free PDF. See conwaylife.com/book for pattern files and print version.

failure. Note that we have made the output Z and NZ lanes of this particular sliding block register transparent so that multiple registers can easily be placed side-by-side.

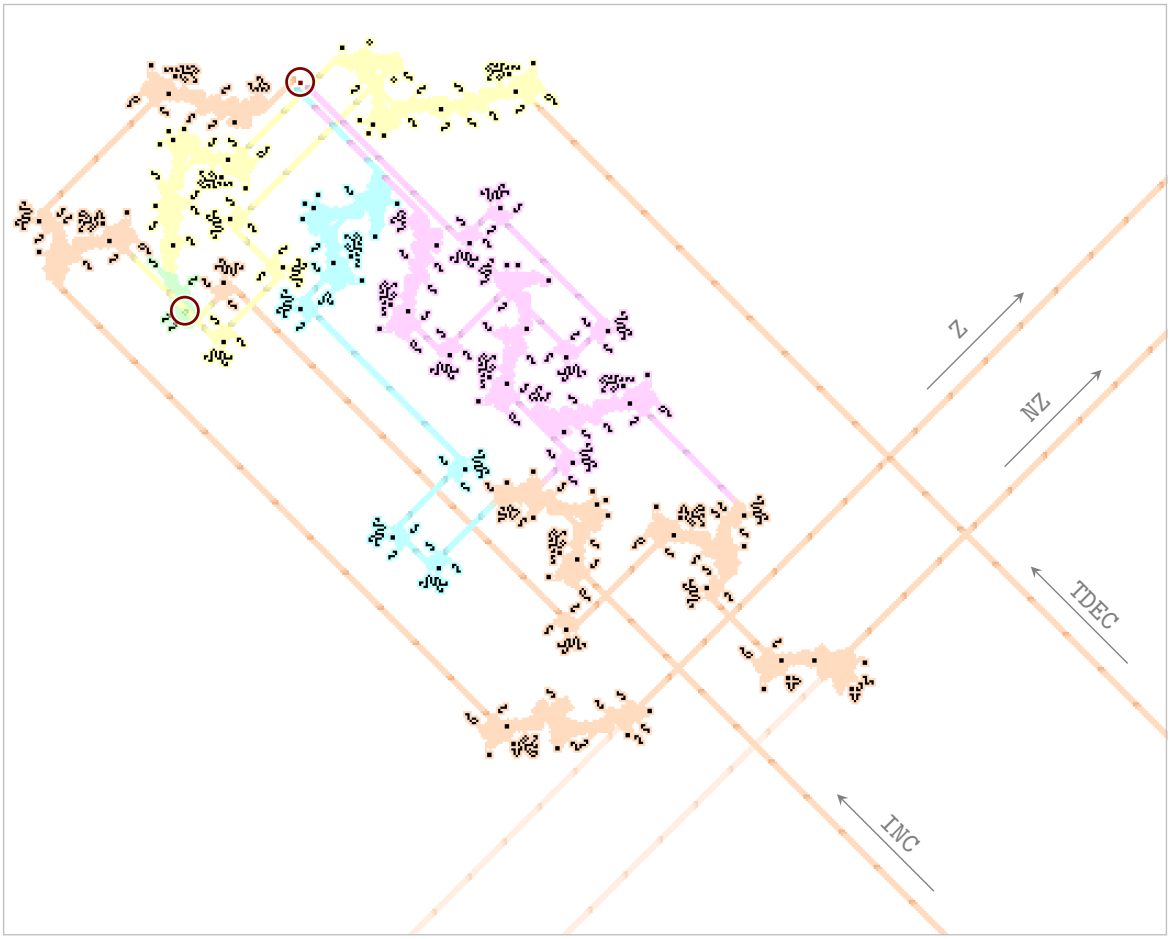


Figure 9.2: A sliding block register. If a glider enters on the INC lane, the aqua and magenta conduits split it into three gliders that push the sliding block (circled in red near the top) northwest by 1 cell. If a glider enters on the TDEC lane, it enters the TEST circuit (the first half of which is highlighted in yellow). That TEST circuit uses a demultiplexer (highlighted in green) to switch the path of the glider—if the block is in the "zero" position then the glider passes through the demultiplexer to the northwest on the Z output path, and otherwise the demultiplexer creates a boat (circled in red near the top-left) that redirects the glider to the northeast on the NZ output path. In the latter case, the NZ output glider is also fed into the magenta conduit that creates two gliders to pull the sliding block 1 cell southeast.

9.1.2 Assembly Code for a Finite-State Machine

We now have a mechanism that can store integer values, with two simple inputs (INC and TDEC) and two simple outputs (Z and NZ). Before delving into the details of how to position these registers on the Life plane to carry out computations, we first spend some time thinking about how to perform computations that only involve the operations corresponding to their input lanes—increasing or decreasing the value of some register, and checking whether or not a register is currently equal to zero.

To give a rough idea of how we could carry out a computation of this type, consider the task of just checking which of two registers contains a smaller value. We can solve this problem by repeatedly decrementing the value of the registers, and stopping once either of them is storing a value of 0. Whichever one hits 0 first is determined to be the register that started off storing the smaller value.

Free PDF. See conwaylife.com/book for pattern files and print version.

Pseudocode that implements this algorithm is provided by Pseudocode 9.1.

When we write pseudocode like this, we use labels like U0, U1, U2, ... to denote the various registers used by the computer program.[4] Also, for this particular pseudocode it is intended that the computer starts by executing the first line of code, then automatically moves to the following line and executes that, and so on—unless it encounters an instruction that tells it to HALT or jump to some other line of the program.

Pseudocode 9.1 Test which of the registers U0 or U1 contains a smaller value.

```
1: if U0 = 0, jump to 6
2: if U1 = 0, jump to 8
3: decrement U0
4: decrement U1
5: jump to 1
6: OUTPUT "U0 <= U1"
7: HALT
8: OUTPUT "U1 < U0"
9: HALT
```

Our next goal is to simplify and standardize our pseudocode so as to make it simpler to implement as a Life pattern. With this in mind, a **finite-state machine** is a model of computation in which the computer can be in one of a finite number of states at any given time, and its state can change based on inputs that it receives. We can implement a finite-state machine with n possible states via a Life pattern that has a single glider traversing a set of n parallel lanes connected by other circuitry. At any given computer clock tick, the glider will be on exactly one of the "state" lanes. It will then run through structures that send gliders to various inputs of various register circuits, and also send a glider to the lane representing the next state in the program.

In order for our pseudocode to more accurately reflect this finite-state machine that our Life program will implement, we require 2 additional things of all pseudocode that we write from this point forward:

1) We include a "jump" instruction at the end of every single line. This corresponds to specifying exactly what state transition happens at each computer clock tick in the finite-state machine that we are implementing.[5]
2) All conditional statements that we include are conditioned on whether the last output value received from a register was Z or NZ. We thus have to be careful to make sure that we always perform some task (like a TDEC) that returns a Z or NZ value immediately before jumping to a conditional statement.

By making the changes outlined in points (1) and (2) above, our original Pseudocode 9.1 is transformed into Pseudocode 9.2. These pseudocodes implement the same computational task (determining which of the two registers U0 and U1 is storing a smaller value), but the new pseudocode will be simpler for us to implement as a Life pattern. In particular, each line in Pseudocode 9.2 now corresponds to a state in the finite-state machine that we will implement. This machine will start in the state of Line 1, and will move from state to state (i.e., line to line) until the algorithm has completed

[4] The "U" stands for the word **unary**, since these sliding block registers stores integers in unary (i.e., the base-1 numeral system). We will see a different type of register that instead stores integers in binary a bit later, in Section 9.4.

[5] Our pseudocode, and the programming language that we will develop based on it, is thus one of the few examples of a language where execution does *not* automatically proceed from one line to the next. For a more "real-world" example of such a language, see for example RPC-4000 machine code, as described in The Story of Mel: catb.org/jargon/html/story-of-mel.html

Free PDF. See conwaylife.com/book for pattern files and print version.

its work. Eventually either one output or the other will be sent out, and the program will reach its HALT state.

Pseudocode 9.2 Test which of the registers U0 or U1 contains a smaller value—second version.

```
1: TDEC U0; jump to 2
2: if return=Z then jump to 5, otherwise jump to 3
3: TDEC U1; jump to 4
4: if return=Z then jump to 6, otherwise jump to 1
5: OUTPUT "U0 <= U1"; jump to 7
6: OUTPUT "U1 < U0"; jump to 7
7: HALT
```

We now refine this pseudocode even further and standardize it into what we call **APGsembly code**—the programming language that we will use to write and compile computer programs in Life patterns.[6] In order to make the conditional statements of our pseudocode even easier to implement, we split each state of the finite state machine (i.e., our "computer") into two substates: one corresponding to the most recent return value being Z, and one corresponding to the most recent return value being NZ. These substates may result in completely different actions being taken and may cause the program to subsequently jump to completely different states (just as in a regular conditional statement).[7]

Once we make the above change, we no longer really need logical instructions in our pseudocode at all: every program can be specified by listing what actions should be taken (e.g., INC or TDEC) when a particular state and substate is encountered, along with which state should then be jumped to next. See APGsembly 9.1 for our first concrete example of APGsembly code—it implements the same "does U0 contain a smaller value than U1?" test that we saw in Pseudocode 9.2.

APGsembly 9.1 APGsembly code to test which of the registers U0 or U1 contains a smaller value. An output value of 0 indicates that U0 <= U1, while an output value of 1 indicates that U1 < U0.

```
# State    Input    Next state    Actions
# ---------------------------------------
INITIAL;   ZZ;      ID1;          TDEC U0
ID1;       Z;       ID2;          OUTPUT 0, HALT_OUT
ID1;       NZ;      ID2;          TDEC U1
ID2;       Z;       ID1;          OUTPUT 1, HALT_OUT
ID2;       NZ;      ID1;          TDEC U0
```

On its surface, this APGsembly code looks quite different from the pseudocode that we saw earlier, so it is a good idea to trace through the program flow carefully until you are comfortable with the fact that it really does implement the exact same algorithm that we already saw. We now describe how APGsembly code is written, and how it differs from the pseudocode that we saw earlier:

- Line numbers from our pseudocode have been replaced by paired ID labels. Each state can have any alphanumeric ID label of our choosing, with the exception of the first, which is always called INITIAL.[8] Using IDs like this instead of line numbers makes it much easier to manage

[6] The name "APGsembly" is a play on the word "assembly" (low-level code for programming computer instructions) and the initials of its author, Adam P. Goucher. Indeed, APGsembly was used by Goucher to construct the original π calculator in 2010.

[7] In the actual Life implementation of the finite-state machine, this will mean that a glider will appear on one of two different parallel lanes during any given state, depending on whether the last output value received from a register was Z or NZ.

[8] Also, the INITIAL state should never be returned to later in a program's execution. It should be the first state, and *only* the first state.

Free PDF. See conwaylife.com/book for pattern files and print version.

our code—if we were to continue using line numbers instead of labels then we would have to update every single one of our "jump" instructions if we ever inserted a new line of code.

- There are no longer any explicit "if" statements in the code, since each state (pair of lines sharing an ID label) acts implicitly as an "if" statement. *If* the return value (from the previous action) is `Z`, perform the actions given on the first (`Z`) line; *otherwise* perform the actions given in the second (`NZ`) line.
- Sometimes, a particular state can only ever be reached by a `Z` input, so it does not need a corresponding `NZ` substate. In this case, the `NZ` substate can be omitted from the APGsembly code, and the `Z` substate is instead denoted by `ZZ` so that the compiler knows that the `NZ` omission was intentional. The `INITIAL` state is always assumed to have a `Z` input (just because the computer has to start in *some* substate) and thus the first line of APGsembly code always starts with `INITIAL;ZZ`.[9]
- Each line contains four items, separated by semicolons: state ID, input value, next state ID, and a comma-separated list of actions. The "next state" ID tells the program which state ID to jump to after performing the actions listed on the current line.
- To make code easier to read, we can add comments to our code via lines that start with `#` (so the first two lines of APGsembly 9.1 do not actually affect what the code does, but just help us keep track of the four items on each line of code). For a similar reason, we can include any white space of our choosing between pieces of code, and empty lines between states if desired.
- The `OUTPUT 0` and `OUTPUT 1` actions tell the Life computer to print out either a `0` or a `1` on the Life plane, in a font made up of blocks. We will see how this printing is actually done in Section 9.5, but for now we note that by default this action can only print the digits `0`–`9` and periods (`.`), not more complicated expressions like "`U0<=U1`".[10] The `HALT_OUT` action stops the computer[11] (presumably because we are done our desired computation) and emits an output glider that could then be used by another pattern on the Life plane to detect that the computation finished.

There are a few restrictions on what combinations of actions can be performed in a single line of APGsembly code. One of the reasons for this is that all actions on a particular line of APGsembly code are performed simultaneously, rather than sequentially. We should thus avoid single-line lists of actions like "`TDEC U0, INC U0`", for example, since the result of the `TDEC` may depend on whether or not the `INC` has already been completed, which is unpredictable. If the order of actions matters, they should be split into states on multiple different lines.

Furthermore, because of how these actions will be implemented in our Life computer (which we explore in the upcoming Section 9.2), it is not possible to perform the same action more than once in a single substate (i.e., line of APGsembly). For example, if we want to increase the value of the register `U0` by `2`, it is tempting to use the list of actions "`INC U0, INC U0`" in a single line of APGsembly. However, this must be avoided—we should instead perform the first `INC U0`, then jump to another state, and then perform the next `INC U0`.

One final restriction comes from the fact that the return value from one line of APGsembly code dictates which substate of the next state is executed, so each line of APGsembly must contain exactly

[9] Similarly, if a state performs the exact same actions and jump operation regardless of whether it receives a `Z` or `NZ` return value, we can just list a single line for that state with `*` as its input value, instead of two otherwise identical lines with `Z` and `NZ` as their input values. We will not see an example of this in the main text, but it appears in Appendix B.8.

[10] However, we will see in Section 9.5 that this printing can, with slightly more work, be extended to any set of characters of our choosing.

[11] For this reason, the "next state" provided in lines containing a `HALT_OUT` action does not actually matter, since the computer never actually proceeds past that point.

Free PDF. See conwaylife.com/book for pattern files and print version.

one action that produces a return value. If multiple actions were to produce a return value then it would not be clear which substate should be jumped to next, and if no action produced a return value then no substate would be jumped to and the computer would stop running altogether.[12] So far, TDEC is the only action that we have introduced that produces a return value. Future sections will introduce some additional logic components that include value-returning actions, and a summary of these components and actions can be found by jumping ahead to Table 9.1. But first, let's take a look at what a compiled Life pattern arising from APGsembly code actually looks like.

9.2 A Compiled APGsembly Pattern: Adding Registers

APGsembly code is specifically designed so that it can straightforwardly be compiled into a Life pattern.[13] We now illustrate how this is done by constructing a Life computer that adds the values of two sliding block registers.

Our first step toward building such a computer is to write APGsembly code that implements the desired computational task. The code in APGsembly 9.2 does the job nicely—it works by repeatedly decreasing the value of one register until it reaches a value of 0, while simultaneously increasing the value of the other register every time. The only pieces of this APGsembly that we have not yet seen are the "#COMPONENTS" and "#REGISTERS" lines in the APGsembly's header. These lines are used to tell the APGsembly compiler which registers are used in the code,[14] and what the initial values of the registers should be, respectively. If a register is not initialized via the #REGISTERS line, it starts with a value of 0.

APGsembly 9.2 APGsembly code to add the value of U0 to U1, and zero out U0. The #REGISTERS line pre-loads the registers with the values 7 and 5, respectively (so that after the computation completes, we will have U0 = 0 and U1 = 12).

```
#COMPONENTS U0-1,HALT_OUT
#REGISTERS {'U0':7, 'U1':5}
# State    Input    Next state    Actions
# ---------------------------------------
INITIAL;   ZZ;      ID1;          TDEC U0
ID1;       Z;       ID1;          HALT_OUT
ID1;       NZ;      ID1;          TDEC U0, INC U1
```

While this code is quite simple (it's only 3 lines long!), it illustrates an important oddity of APGsembly that we must keep in mind. Since TDEC is short for "test and *then* decrement", it has the somewhat counterintuitive feature of giving a Z or NZ return value that is based on the value that was contained in it *before* it was decremented. In particular, if we use a TDEC to decrement a register from 1 to 0, the return value will be NZ, not Z!

For this reason, APGsembly loops based on the TDEC action often require one more iteration than might be expected at first. If we want to decrement the value of a register from n down to 0, we have to call TDEC n times. However, we then have to call it 1 more time (which does not actually affect the value of the register, since it cannot decrease below 0) in order to get a return value of Z instead of NZ and break out of the loop.

[12] As a technical note, the HALT_OUT action does not actually produce a return value (it does not need to, since we *want* the computer to stop when it encounters HALT_OUT). Instead, it is just a special action that bypasses the need for every line to have a return value.

[13] Links to an APGsembly compiler, and some emulators that can be used to help debug APGsembly programs, can be found at conwaylife.com/wiki/APGsembly.

[14] "U0-1" means that the code uses registers U0 and U1. The "-" indicates a range (so "U0-2" would refer to registers U0, U1, and U2, for example).

Free PDF. See conwaylife.com/book for pattern files and print version.

Since APGsembly code requires an action in the `INITIAL` state anyway before any loops are entered, we often put the extra `TDEC` command there (we did this in each of APGsembly 9.1 and 9.2). The resulting code has the effect of decreasing the value of the register from n to $n-1$, then looping from $n-1$ to -1, but with the register getting stuck at 0 instead of actually decrementing all the way down to -1.

An actual Life pattern that implements the computation described by APGsembly 9.2 is presented in Figure 9.3. There is a lot going on in this pattern, so we now describe how it is built in some detail. This computer (and Life computers built from APGsembly in general) consists of three main parts: a **computer** (in the southeast), a **component stack** (in the northwest), and a **clock gun** (in the north). We describe these three pieces one at a time.

9.2.1 The Computer

The computer is an implementation of a finite-state machine, so when it is at rest it is always in one of a finite number of different states, represented by a pair of demultiplexers that have a boat. Either a `Z` or an `NZ` signal will head southeast from the clock gun as a result of the computation performed in the computer's previous state. That glider will hit either the `Z` or the `NZ` demultiplexer and be reflected toward the southwest on a lane corresponding to exactly one specific line of APGsembly code.

If the computer is currently in state `ID1`, for example, then the two demultiplexers representing the `ID1` state both contain boats. If a `Z` signal comes in, the `Z` boat turns a glider onto the `ID1;Z` lane, and the `NZ` boat is cleaned up by a following glider. Conversely, if an `NZ` signal comes in, the `NZ` boat turns a glider onto the `ID1;NZ` lane, and the `Z` boat gets cleaned up instead.

The glider heading southwest then passes through one or more splitters—glider duplicators that create a perpendicular glider while also sending another glider to continue along the exact same lane. As many splitters as we desire can thus be added along these lanes without changing the final destination of the southwest-travelling glider. Each splitter sends a glider on an output lane corresponding to an action from the current line of APGsembly code: `TDEC U0`, `INC U1`, and so on.

After going through the sequence of splitters, the southwest-bound glider hits a merge circuit—a reflector that is transparent to signals passing through it on its output lane. This implements the "jump to" part of each APGsembly line, and the transparency allows multiple lines of APGsembly to jump to the same state. Indeed, multiple merge circuits on the same northwest-to-southeast diagonal all produce gliders on the same lane, which are routed around to trigger the pair of demultiplexers for the next target state.[15] The two boats then wait in the demultiplexer for the next `Z` or `NZ` signal from the clock.

As a technical implementation note, recall that the `INITIAL` state is always assumed to have a `Z` input. That `Z` input is hard-coded into the Life pattern in order to start the computation. The computer displayed in Figure 9.3 does not even have an `INITIAL;NZ` substate, since it would be impossible to reach anyway—recall that this is why the `INITIAL;Z` substate was listed as `INITIAL;ZZ` in APGsembly 9.2.

9.2.2 The Component Stack

In these calculator patterns, information is stored separately from the computer, in an array of "components". These components each have a finite number of inputs (called "actions" in APGsembly code) that can be used to manipulate or retrieve that information, and they potentially provide a single

[15] The computer in Figure 9.3 only has one path around its southern corner along which gliders can be reflected back into the demultiplexers. This is because every single line of APGsembly 9.2 that generated that computer tells it to jump to the same state (`ID1`). We will see other computers shortly that can jump to multiple different states and thus have more glider paths down there.

Free PDF. See conwaylife.com/book for pattern files and print version.

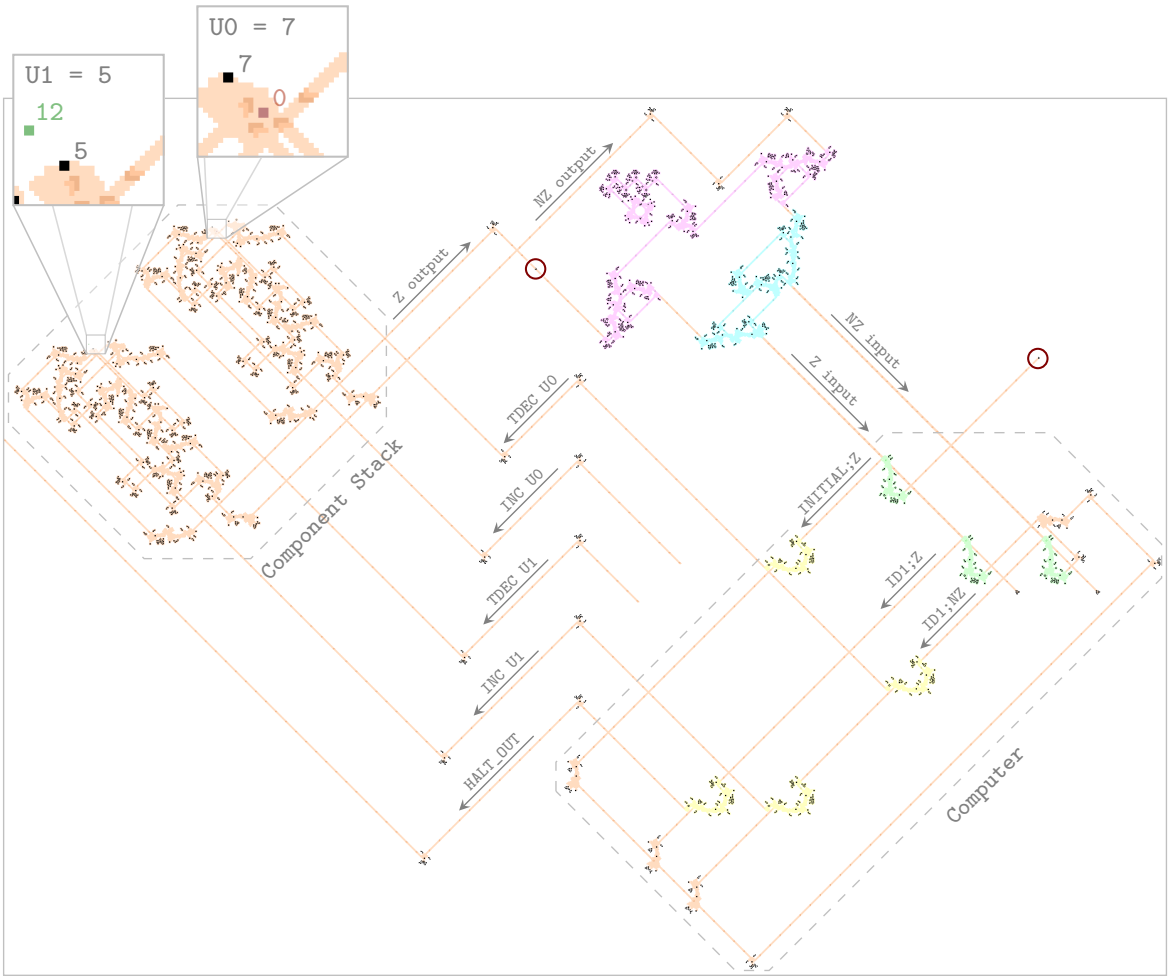


Figure 9.3: A compiled Life pattern that adds the values of the registers `U0` and `U1` (which have been pre-loaded with the values `7` and `5`, respectively), as in APGsembly 9.2. The registers at the west feed their output into the period 2^{20} clock gun (highlighted in magenta) with two separate universal regulators—one for the `Z` path and another for the `NZ` path). The clock gun feeds a glider into a demultiplexer (highlighted in green) corresponding to the next substate of the computer (i.e., line of APGsembly). That demultiplexer then feeds into splitters (highlighted in yellow) corresponding to the actions in that line of APGsembly (the `INC U0` and `TDEC U1` lanes are disconnected from the computer because those actions are not called by any line of APGsembly 9.2). Finally, the duplicated gliders heading to the northwest feed back into the registers, and the duplicated glider heading to the southwest is reflected around the southeast end of the computer so as to activate the demultiplexer for its next state. The conduit highlighted in aqua puts the `Z` or `NZ` glider coming from the clock gun on the correct input lane, and then produces an extra cleanup glider on each computer input lane so as to reset the demultiplexers. The two gliders circled in red start the computation by activating the `INITIAL;Z` demultiplexer and then feeding a glider into it.

`Z`/`NZ` output bit. The first example that we have seen is the sliding block register, which has two action inputs (`INC Un` and `TDEC Un`) and the two standard outputs (`Z` and `NZ`). Since their output lanes are transparent, any number of sliding block registers `U0`, `U1`, `U2`,... can be stacked side-by-side, and APGsembly code can `INC` or `TDEC` any one of them. If there are a lot of registers, the computer part of the pattern has to be stretched a little wider to accommodate spaces for more splitters. The computer has to be wide enough to hold two splitters for each sliding-block register—one for each `INC` action and one for each `TDEC` action.

Free PDF. See conwaylife.com/book for pattern files and print version.

Other components may have more or fewer action inputs. For example, we will see a binary register in Section 9.4 that has four actions: `INC Bn`, `TDEC Bn`, `READ Bn`, and `SET Bn`. New components could be designed to perform any logical function or store any amount of information that we might want; we will see an example of this in Section 9.7.2, with the two-dimensional memory storage "`B2D`" component.

No matter how many components are added to the stack, they're all called in the same way, by a single glider travelling northwest on an "action" lane. Multiple actions can be triggered simultaneously, though each component may be activated at slightly different times depending on where it is placed in the stack. Since the precise timing of the actions depends heavily on how the computer and component stack are wired together, it is strongly recommended that a single line of code in APGsembly never triggers more than one action in a particular component. While it does not always cause problems, calling two simultaneous actions would cause many of the components that we consider here to fail. This problem can be avoided simply by calling the two incompatible actions from separate states (i.e., lines of APGsembly code).

9.2.3 The Clock Gun

Once a `Z` or `NZ` output signal is generated by the component stack, it is fed back into the computer in order to jump to the next state. However, we have to be slightly careful about when this glider arrives at the computer—we have to be sure that it does not arrive before the next state's demultiplexers are set in the computer (i.e., before the computer's glider has had a chance to go through the merge circuits and find its way all the way around the southern perimeter of the computer part of the pattern).[16]

To delay the component stack's output glider, we could simply extend the length of the path that it must follow back to the computer. However, we instead make use of a very high-period universal regulator. The advantage of this method is that several pieces of the pattern then repeat predictably at the period of the universal regulator, which lets Life simulation software evolve it much quicker than it could if the glider timing was less regular. In particular, if we choose the universal regulator to align to some high power-of-two period, then Golly's *HashLife* algorithm is able to evolve these patterns extremely quickly.[17]

The universal regulator that we use[18] is conceptually very simple. A stream of gliders coming from a gun of any (sufficiently large) period of our choosing comes in from the northeast and is split into two streams that destroy each other. However, if a glider comes in from the northwest then it is fed into a syringe and then one of the Herschel-to-boat factories from Figure 7.30(a). The resulting boat suppresses a single glider from the gun's duplicated stream, letting a glider that is aligned to the period of that gun escape, as illustrated in Figure 9.4.

The gun that we attach to this universal regulator has period 2^{20}, which we choose simply because it is a power of 2 that is large enough to handle most of the calculators that we will construct in this chapter, but small enough that it can be simulated quickly in software like Golly. To actually construct this gun, we just do exactly what we did in Section 7.6: we attach period multipliers to another gun. This particular gun (displayed in Figure 9.5) uses quadri-Snarks (see Exercise 7.23) attached to a p256 machine gun, but semi-Snarks could have been used instead at the expense of the gun being slightly larger.

[16] This is not much of a concern in the pattern from Figure 9.3 since the path around the computer is so short, but it becomes an issue for patterns with larger computers (i.e., patterns generated by more lines of APGsembly).

[17] HashLife is an algorithm for simulating Life that was developed by Bill Gosper in 1984 [Gos84]. It works by creating a lookup table that keeps track of how the $2^n \times 2^n$ center of certain repetitive $2^{n+1} \times 2^{n+1}$ chunks of a pattern evolve over 2^{n-1} generations. Since nothing outside of that $2^{n+1} \times 2^{n+1}$ square can affect its $2^n \times 2^n$ center within those 2^{n-1} generations, the results of that computation can simply be reused whenever that same square is encountered in the future.

[18] Constructed by Louis-François Handfield in April 2020.

Free PDF. See conwaylife.com/book for pattern files and print version.

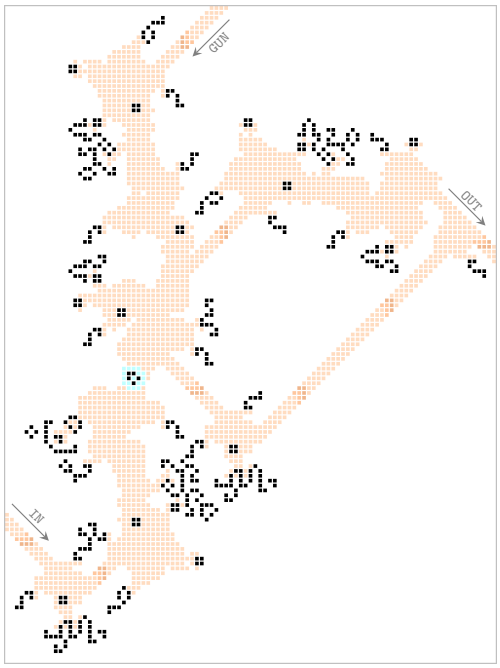


Figure 9.4: A stable universal regulator that works by having the input glider create a boat (highlighted in aqua), which suppresses the glider on the southeast path and allows the northeast glider to escape.

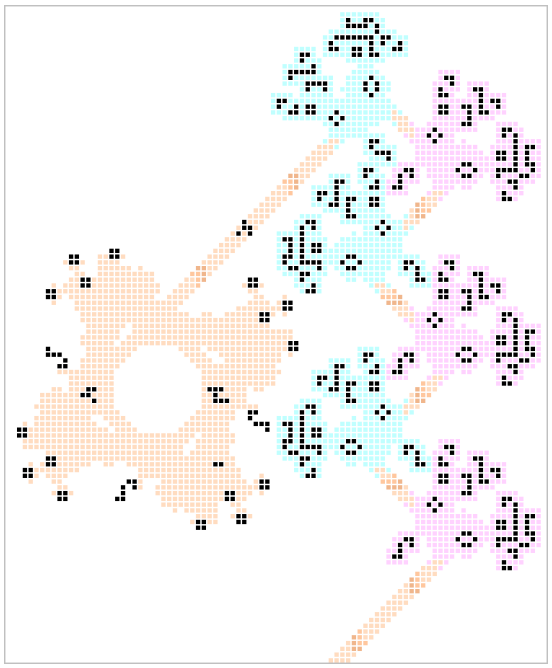


Figure 9.5: A period 2^{20} gun that works by using 6 quadri-Snarks (highlighted in aqua and magenta) to repeatedly quadruple the period of a p256 machine gun.

The high-period universal regulator that results from stitching together Figures 9.4 and 9.5 is called the **clock gun**, and it determines the speed at which the pattern computes whatever it has been programmed to compute. We think of the period of this clock gun as the clock speed of the computer, and one period of the regulator as one clock tick or computational cycle.[19]

In summary, the calculator patterns that we compile from APGsembly work as follows:

- The computer in the southeast corner keeps track of the state and sends action commands to the component stack.
- The action commands alter the contents of the components in the component stack, which serve as the computer's memory. Exactly one of these components then sends an output signal (glider) to the clock gun.
- The clock gun regulates the glider signal coming from the component stack and feeds it back into the computer.

9.3 Multiplying and Reusing Registers

Now that we understand how to *add* the value of two sliding block registers, we ramp up to the problem of *multiplying* their values. One seemingly simple method of multiplying `U0` by `U1` and storing the result in `U2` would be to repeatedly add the register `U1` to `U2` a total of `U0` times. However, there is a slightly problem with this idea—our method of adding two registers from APGsembly 9.2

[19] Some computations may take longer than one clock tick to complete, but that's okay. For example, if a sliding block register is storing an extremely large value then its sliding block will be extremely far away from the computer, so it will take a long time to `INC` or `TDEC`. In this case, the clock gun simply waits for one or more clock ticks until an output signal is received from the component stack.

Free PDF. See conwaylife.com/book for pattern files and print version.

zeroes out one of the registers while adding it to the other. Indeed, the only method that we have of looping over one register is based on TDECing it until it hits 0, thus erasing the value that register contained.

To get around this problem, we introduce a temporary register into which we copy the value of one of the registers that we wish to loop over, and then we loop over that temporary register instead. In this particular case of setting U2 = U0 * U1, every time we add U1 to U2, we do so by first copying U1 to a new temporary register U3 (zeroing out U1 in the process), and then looping over U3 so as to add it to each of U1 *and* U2. Pseudocode for carrying out this task, as well as the corresponding APGsembly code, is presented in APGsembly 9.3.[20]

APGsembly 9.3 Pseudocode (left) and APGsembly code (right) to set U2 = U0 * U1 and zero out U0. The register U3 is used just temporarily (it starts and ends at the value of 0) to store the value of U1. After this computation completes, the registers will have the values U0 = 0, U1 = 5, U2 = 35, and U3 = 0.

```
 1: TDEC U0
 2: if return value = Z:
 3:     # Loop U0 times, then halt.
 4:     HALT
 5: else:
 6:     # Add U1 to U2 without erasing U1
 7:     set U3 = U1
 8:     set U1 = U3, U2 = U2 + U3
 9: end if
10: jump to 1
```

```
#COMPONENTS U0-3,HALT_OUT
#REGISTERS {'U0':7, 'U1':5}
# State    Input    Next state    Actions
# ---------------------------------------
INITIAL;   ZZ;      ID1;          TDEC U0

# Loop over U0, TDECing it until it hits 0, and then halt.
ID1;       Z;       ID1;          HALT_OUT
ID1;       NZ;      ID2;          TDEC U1

# Copy U1 into U3 while setting U1 = 0.
ID2;       Z;       ID3;          TDEC U3
ID2;       NZ;      ID2;          TDEC U1, INC U3

# Loop over U3, adding its value to U1 (restoring it) and U2.
ID3;       Z;       ID1;          TDEC U0
ID3;       NZ;      ID3;          TDEC U3, INC U1, INC U2
```

A Life pattern that was compiled from APGsembly 9.3 is displayed in Figure 9.6. This multiplication pattern has the same general shape and structure as the addition pattern from Figure 9.3, but with slightly more of everything—4 sliding block registers instead of 2, 7 substates in the computer (corresponding to the 7 lines of APGsembly code) instead of 3, and so on. Perhaps the most notable change in this calculator pattern is that there are now 3 glider lanes looping around the southern perimeter of the computer, whereas there was only 1 such lane in Figure 9.3. These extra lanes correspond to the fact that there are lines of APGsembly code in the multiplication program that jump to each of three different states (ID1, ID2, and ID3), whereas every line in the addition program jumped to the same state (ID1).

Just like we can multiply two registers via repeated addition, we can divide two registers via repeated subtraction. In particular, to compute the integer part of U0 / U1 we can repeatedly subtract U1 from U0 until U0 = 0. The number of times that we were able to completely subtract U1 is the integer part of U0 / U1, and the value that was contained in U0 when we started our final subtraction is the remainder of that division. Implementing this division-by-subtraction algorithm in APGsembly is very similar to the multiplication-by-addition APGsembly 9.3, so we leave it to Exercise 9.4.

In fact, sliding block registers and the computational framework that we have introduced so far are already Turing complete—they can compute anything that can be computed. However, they are somewhat unwieldy to actually program to perform non-trivial computations, and they are not particularly interesting to look at. The remainder of this chapter is devoted to additional components that make computations in our Life patterns easier to implement, faster to perform, or simply more interesting to watch.

[20] After this code is executed, the value of U1 is preserved, but the value of U0 is not. If we want to preserve the value of U0, we can use *another* temporary register (see Exercise 9.3).

Free PDF. See conwaylife.com/book for pattern files and print version.

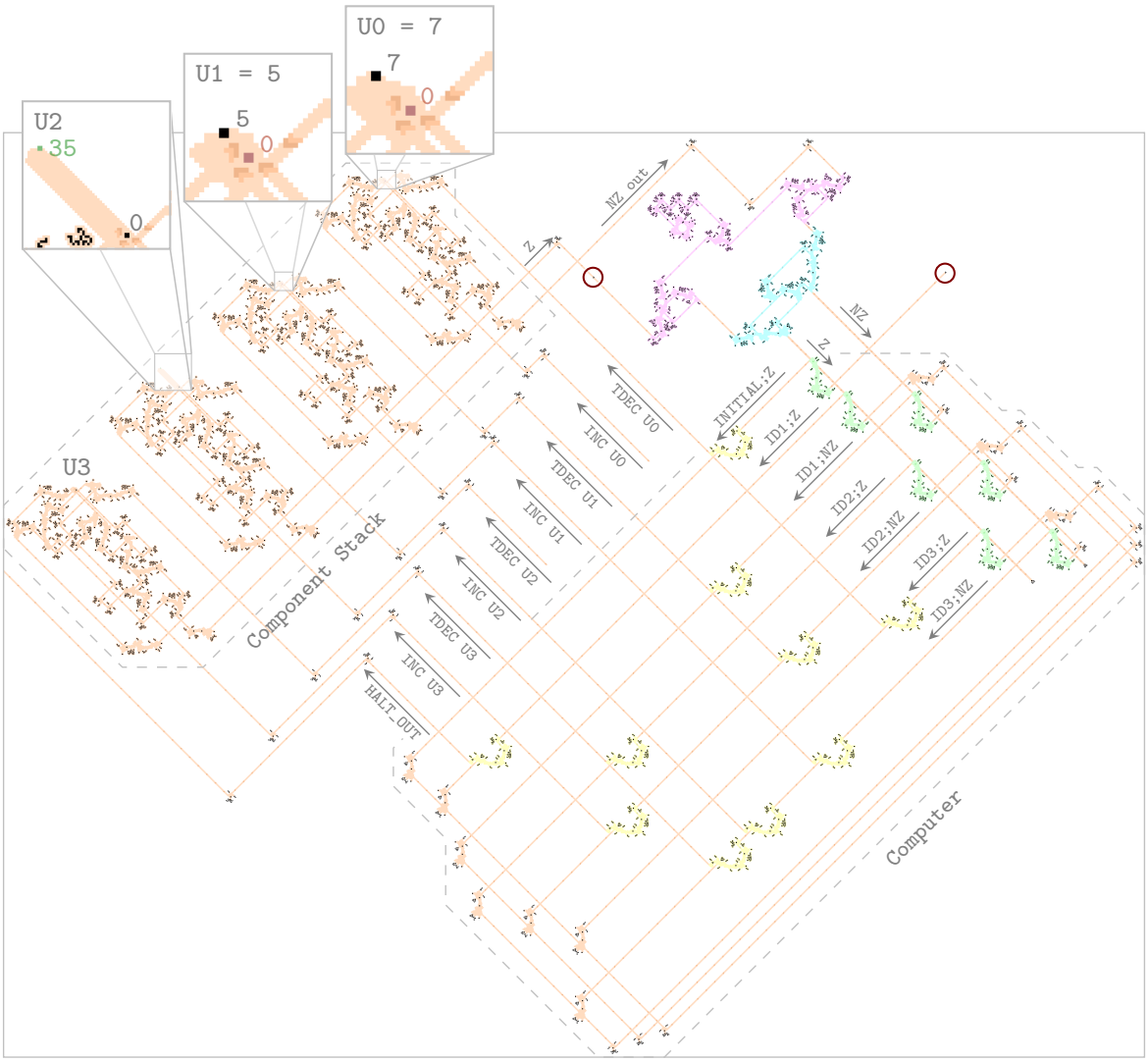


Figure 9.6: A compiled Life pattern that multiplies the values of the registers `U0` and `U1` (which have been pre-loaded with the values `7` and `5`, respectively) and stores the result in `U2`, as in APGsembly 9.3. The color scheme and general structure of this pattern is the same as in Figure 9.3.

9.4 A Binary Register

One of the unfortunate features of sliding block registers is that the only actions they have available to them are addition and subtraction by `1`, so arithmetic operations involving large numbers stored in these registers take a long time to complete. For example, the addition code of APGsembly 9.2 takes `U0+1` clock ticks to add the value of `U0` to `U1`, and the multiplication code of APGsembly 9.3 takes roughly `2 * U0 * U1` clock ticks to multiply `U0` by `U1`.[21]

While larger inputs leading to longer times to perform arithmetic operations is inevitable, we can save a lot of time by representing non-negative integers in a more efficient way than just as the position of a sliding block. While a sliding block register can be thought of as storing a non-negative integer in **unary** (i.e., the base-1 numeral system, in which each number is represented by tally marks or a distance from a fixed point), the register that we now introduce instead encodes non-negative

[21] In fact, even the subtraction by `1` operation (i.e., `TDEC`) takes longer to complete if the value of the register is extremely large, since the sliding block takes longer to interact with and produce its return value when it is far away.

Free PDF. See conwaylife.com/book for pattern files and print version.

integers in **binary** (i.e., the base-2 numeral system). Performing operations on such a register is somewhat more complicated, but it has the advantage of being much quicker because, for example, the non-negative integer n can be stored in $\Theta(\log(n))$ space instead of $\Theta(n)$ space.

We call a register that stores a non-negative integer in this way a **binary register**, and the one that we make use of is displayed in Figure 9.7. It uses block-moving shotguns similar to the ones in sliding block registers, except that they move a block 6 cells diagonally at a time instead of just 1. This wider spacing leaves room for additional reactions to place boats on one side of the sliding block. Every 6 cells along the sliding block's path is a designated bit location, which can contain either an empty space or a single boat, corresponding to the bits 0 and 1, respectively.

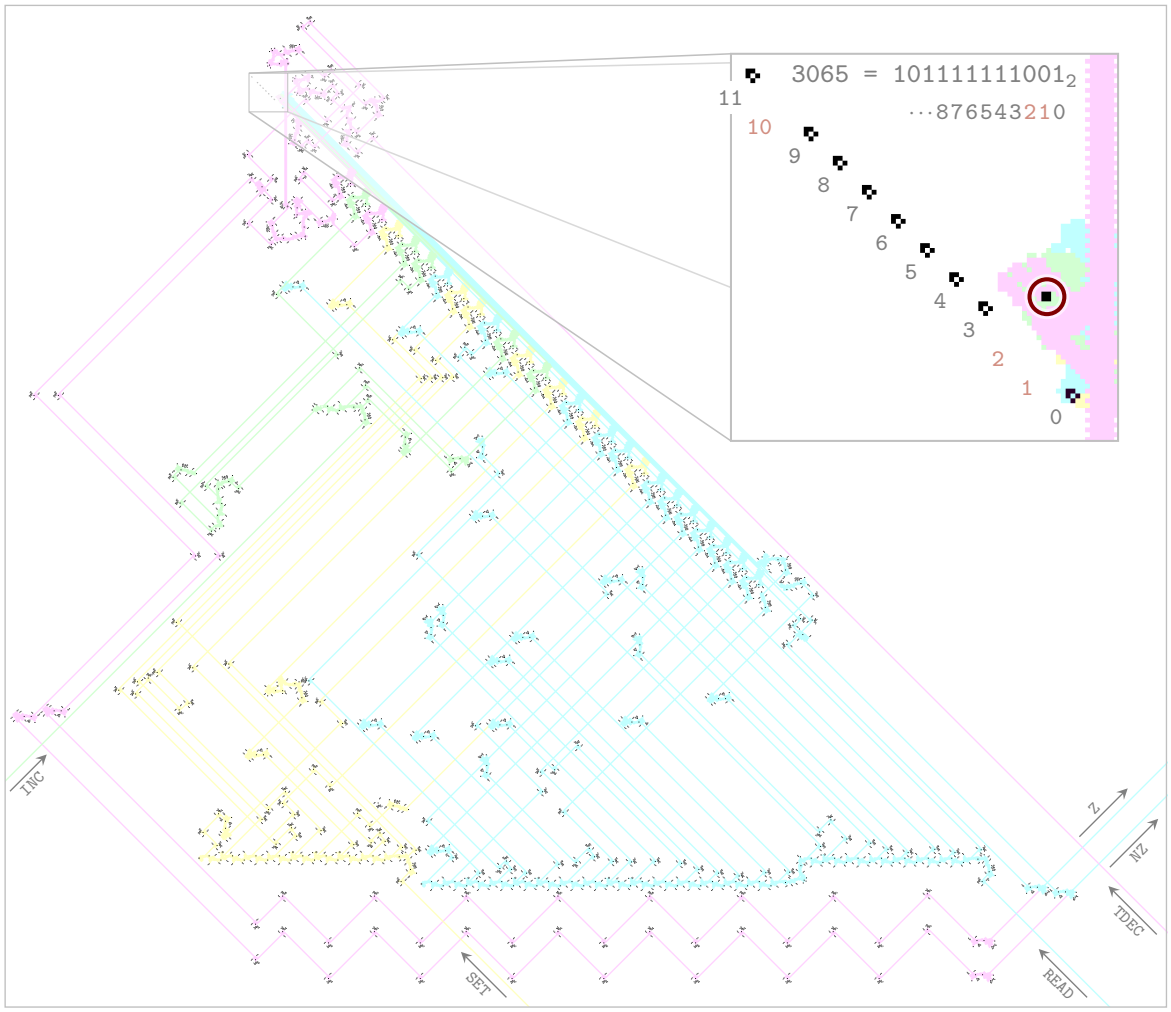


Figure 9.7: A binary register that uses a sliding block to keep track of the location of the read head (circled in red in the zoom box at the "0" location) and boats to represent bits (shown here storing the value $3065 = 101111111001_2$). The sliding block and boat bits are manipulated by four different slow salvos corresponding to the register's four actions `INC`, `TDEC`, `READ`, and `SET`, which are generated by the circuitry highlighted in green, magenta, aqua, and yellow, respectively.

The four actions available to this binary register are a bit more abstract than the actions of the sliding block register, since they work on the individual bits of the register rather than the value that those bits represent:

- An `INC` action moves the pointer block (called the **read head** of the register) to the next bit

Free PDF. See conwaylife.com/book for pattern files and print version.

location, one step farther away.

- A `TDEC` action moves the read head to the previous bit position (or keeps it in the same place if it's already at the zero position). This action also returns either `Z` or `NZ`, depending on whether or not the read head was already at the location of the least significant bit when this action was called.[22]
- A `READ` action sets the value of the bit at the current read head location to "`0`". This action returns either `Z` or `NZ`, depending on the original value of that bit.
- Finally, a `SET` action places a "1" bit at the current read head location.

Unlike a simple sliding block register, where there's no way to damage the mechanism by sending action signals to it, there are indeed ways to program a binary register that will cause it to explode catastrophically. In particular, if you send a `SET` action while there's already a `1` stored at the current read head location, it will cause irrecoverable damage. There's no built-in safety mechanism to prevent this, but this problem can easily be avoided by always sending a `READ` signal just before any `SET` signal.

We use `Bn` to denote the `n`-th binary register (where we start counting at `n = 0`), just like we use `Un` to denote the `n`-th sliding block register. To illustrate how to make use of these actions to perform bitwise operations, APGsembly 9.4 shows how to store the value $139 = 10001011_2$ in a binary register `B0`.[23] Another way of storing the value 139 in a binary register would be to change the appropriate line of the APGsembly header to `#REGISTERS {'B0':[0,'11010001']}`. However, the header-based method only works if you want to set `B0` to 139 at the start of the computation and not partway through it.

APGsembly 9.4 APGsembly code for storing the value 139 in the binary register `B0`, and then returning its read head to the least significant bit.

```
#COMPONENTS B0,NOP,HALT_OUT
#REGISTERS {}
# State    Input    Next state    Actions
# ---------------------------------------
INITIAL;   ZZ;      ID1;          SET B0, NOP
ID1;       ZZ;      ID2;          INC B0, NOP
ID2;       ZZ;      ID3;          SET B0, NOP
ID3;       ZZ;      ID4;          INC B0, NOP
ID4;       ZZ;      ID5;          INC B0, NOP
ID5;       ZZ;      ID6;          SET B0, NOP
ID6;       ZZ;      ID7;          INC B0, NOP
ID7;       ZZ;      ID8;          INC B0, NOP
ID8;       ZZ;      ID9;          INC B0, NOP
ID9;       ZZ;      ID10;         INC B0, NOP
ID10;      ZZ;      LSB1;         SET B0, NOP

# Move B0's read head back to its least significant bit.
LSB1;      ZZ;      LSB2;         TDEC B0
LSB2;      Z;       LSB2;         HALT_OUT
LSB2;      NZ;      LSB2;         TDEC B0
```

This APGsembly also contains one new action that we have not yet seen: `NOP`. This is an old

[22] This behavior is in complete analogy with the `TDEC` action for the sliding block register—it tests and *then* it decrements (if possible). In fact, the `INC` and `TDEC` portions of the binary register from Figure 9.7 alone make up a sliding block register (they just manipulate the read head, not the boats that represent bits).

[23] The subscript "2" indicates that we are writing the number in binary.

Free PDF. See conwaylife.com/book for pattern files and print version.

standard programming abbreviation, short for "**N**o **OP**eration"; it tells the computer not to change anything right now, but emit a `Z` return value anyway. It is typically used in conjunction with other actions that don't have a return value (`SET B0` and `INC B0` in this case), since exactly one action in every state's list of actions must return a value of either `Z` or `NZ`.[24]

9.4.1 A Binary Ruler

To illustrate how to perform some basic arithmetic operations with our binary register, consider the simple problem of increasing the value (i.e., the number that is stored in binary) of the register by `1`. Unlike the sliding block register, there is no single action that performs this operation—we can only perform actions on one bit at a time, but increasing the value of a binary register by `1` may affect several of its bits due to carries.

Fortunately, since we are just increasing the value of the register by `1`, the carry bits are not difficult to take care of: we look for the least significant `0` bit and set it to `1`, and we set all bits that are less significant than it (which are necessarily currently equal to `1`) to `0`. Slightly more explicitly, we can increase the value of a binary register by `1` via the following procedure:

1) Start with the read head at the least significant bit and then proceed to step (2) below.
2) Read the value of the current bit (setting it equal to `0` in the process). If it equaled `0`, set it to `1` and then return the read head to the least significant bit. If it equaled `1`, move the read head to the next most significant bit (i.e., increase its position) and then return to step (1).

This method is implemented in APGsembly 9.5, which counts in binary by repeatedly adding `1` to the value of a binary register `B0`.

APGsembly 9.5 APGsembly code for a **binary ruler**—a pattern that counts in binary.

```
#COMPONENTS B0,NOP
#REGISTERS {}
# State    Input    Next state    Actions
# ---------------------------------------
INITIAL;   ZZ;      CHECK1;       READ B0

## Determine whether the current bit equals 0 or 1.
# If it equals 0, set it to 1 and go to the least significant bit.
# If it equals 1, set it to 0 and go to the next most significant bit.
CHECK1;    Z;       LSB1;         SET B0, NOP
CHECK1;    NZ;      CHECK2;       INC B0, NOP
CHECK2;    ZZ;      CHECK1;       READ B0

# Move B0's read head back to its least significant bit.
LSB1;      ZZ;      LSB2;         TDEC B0
LSB2;      Z;       CHECK1;       READ B0
LSB2;      NZ;      LSB2;         TDEC B0
```

One interesting feature of the pattern that results from compiling this APGsembly code (see Figure 9.8) is that it exhibits a new type of slow growth that we have not yet seen. We explored

[24] It may be tempting to merge multiple lines of APGsembly 9.4 together so as to get rid of the `NOP` actions. For example, we might want to have an action list for a particular state that says something like `INC B0, INC B0, SET B0, NOP`. However, there are two problems with this: (1) we cannot call the same action twice in a single line of APGsembly (clock tick), and (2) it is not a good idea to perform two actions on the same register (`B0`) in the same clock tick. Indeed, it is not clear which one will be performed first (or even worse, they might interfere with each other and cause the register to self-destruct).

Free PDF. See conwaylife.com/book for pattern files and print version.

patterns with slowly growing *population* in Section 8.8, but this pattern instead has a slowly growing *bounding box*. In particular, its **diameter** (i.e., its longest bounding box side length) in generation t is $\Theta(\log(t))$, since this is the rate at which new most significant bits are added to the end of the binary register.

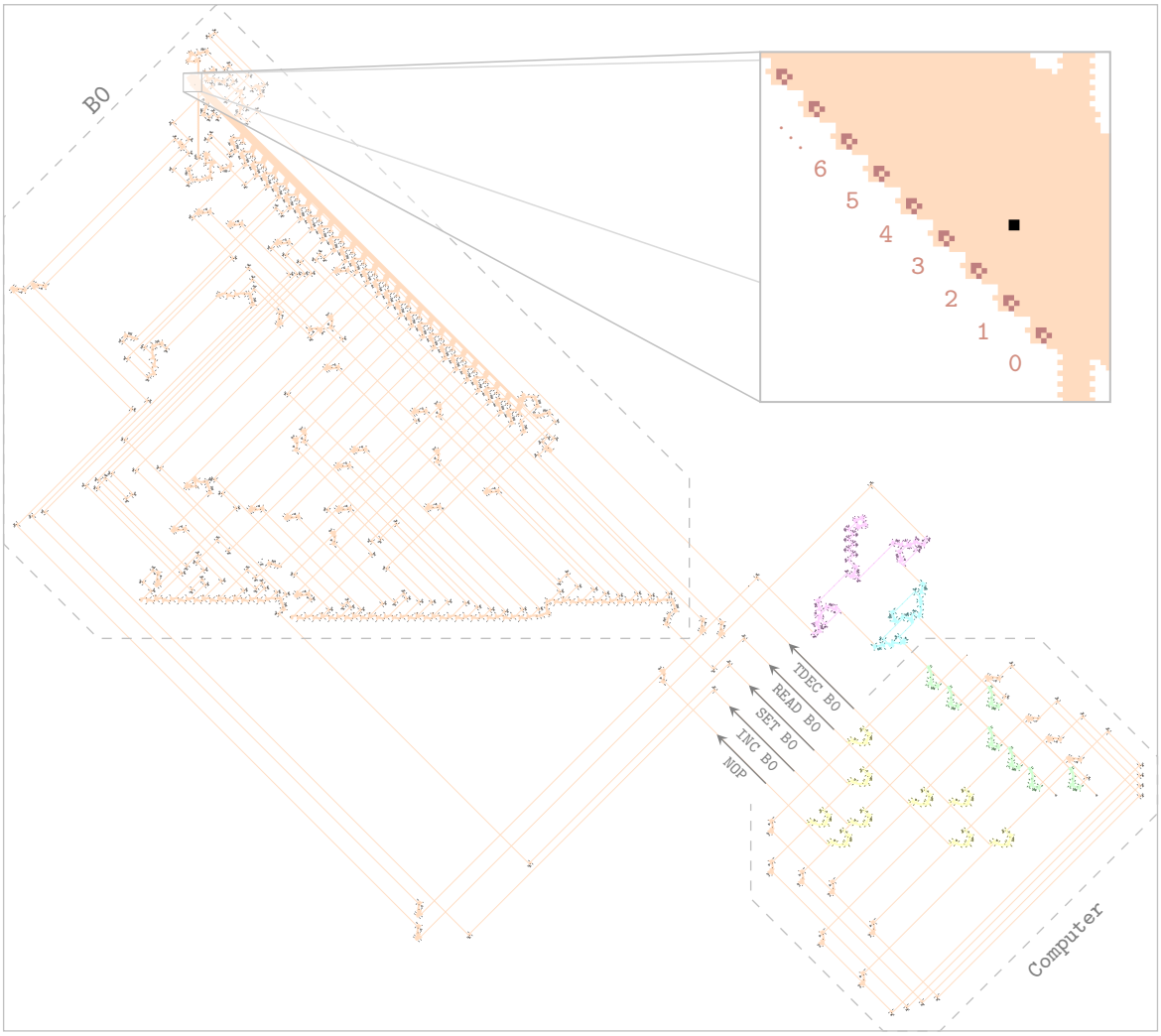


Figure 9.8: A **binary ruler** that counts in binary via APGsembly 9.5. As a result, it grows very slowly, with its diameter in generation t being $\Theta(\log(t))$. Its component stack consists of just a single component—the binary register `B0`, which places boats that act as bits in the locations highlighted in red.

This diametric growth rate is much slower than any other unbounded pattern that we have seen so far (all of which have been $\Theta(t)$). We will return to this idea of patterns with slowly growing bounding boxes, and push it to its ultimate limit, in Section 9.7.2.

9.4.2 Addition, Subtraction, and Multiplication by 10

While addition and subtraction of values that are stored in sliding block (unary) registers is reasonably straightforward (refer back to Section 9.2), these operations are more complicated for binary registers. Even just adding 1 to the value of a binary register and then resetting its read head makes use of seven lines of code (refer back to APGsembly 9.5). The reason for the increase in complexity in this case is that when we add or subtract two binary numbers, we have to keep track of bits that are carried from

Free PDF. See conwaylife.com/book for pattern files and print version.

one bit to the next, possibly many times in a row.

In order to make these arithmetic operations easier to perform, we introduce additional components that are custom-made for storing carry bits. In particular, the ADD and SUB components displayed in Figures 9.9 and 9.10, respectively, perform bitwise operations to assist in the addition and subtraction of binary registers.[25] The ADD component performs the addition of two bits, outputting the least significant bit of the addition and storing the carry bit in its own internal memory, and the SUB component similarly performs the subtraction of two bits while keeping track of the borrow bit.[26]

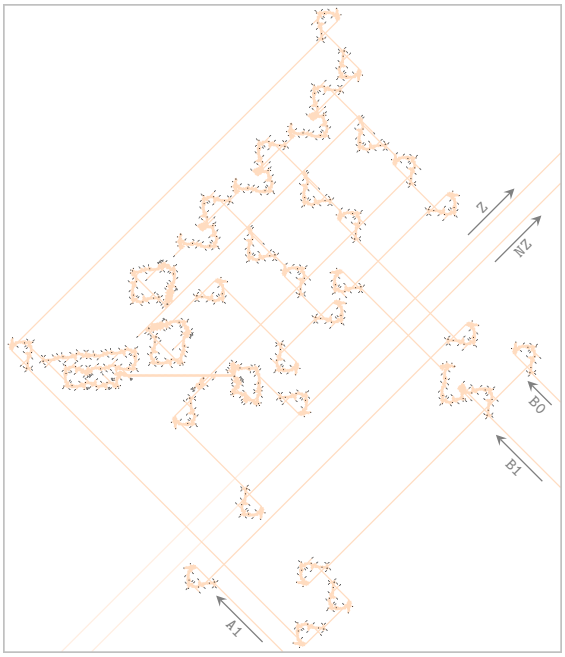


Figure 9.9: The ADD component. To add up two bits x and y while respecting carry bits, input A1 if x = 1 (provide no input if x = 0) and then By.

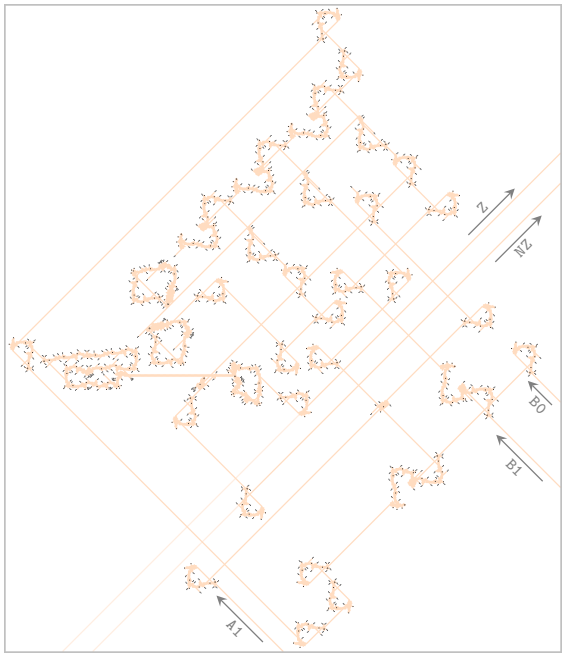


Figure 9.10: The SUB component. To subtract a bit y from x while respecting carry bits, input A1 if x = 1 (provide no input if x = 0) and then By.

More specifically, if we use z to denote the bit that is currently stored in the internal memory of the ADD component, then feeding two bits x and y into it returns a value of (x + y + z) mod 2 and updates its memory to the value (x + y + z)/2 (where we mean integer division here, so that this memory value is either 0 or 1). This ADD component can then be used to add up the values stored in binary registers by repeatedly calling upon it to add up the least-significant bits of those registers, then their next-least-significant bits, and so on until all bits have been added.

However, this procedure only works if we know where the most significant bit is, so that we know when we can stop performing this bitwise addition. The binary registers themselves provide no such functionality—even if we find a million "0" bits in a row, there could always be a more significant "1" bit waiting to be found further on. For this reason, when performing bitwise arithmetic operations like addition and subtraction, we need to make use of a helper sliding block register that tells us (an upper bound of) how many bits the binary registers are making use of. The contents of this sliding block register needs to be updated as our computation proceeds to make sure that it always allocates enough

[25] These components were designed in 2009 or so, before the discovery of the Snark or the syringe. If desired, they could be rebuilt much smaller nowadays with modern Life technology.

[26] Unlike the sliding block and binary registers, we only ever need one ADD or SUB component. In fact, every component that we see from this point on is limited to just a single copy per Life pattern—the sliding block and binary registers are the only components that we can use multiple copies of.

Free PDF. See conwaylife.com/book for pattern files and print version.

bits of memory to the binary registers.

The way that the bit addition `x + y` is executed via APGsembly is that the pair of commands `ADD Ax` and then `ADD By` must be called (where we replace `x` and `y` by their actual binary values, as in `ADD B1`, for example).[27] The `ADD Ax` command does not return an output (it just adds the input bit to the internal memory), but the `ADD By` command does (so it should always be used second). Furthermore, the `ADD A0` command is not actually implemented, since it does nothing—it does not return a value and also does not alter the internal memory.[28]

For example, to add binary registers containing the numbers $104 = 1101000_2$ and $57 = 111001_2$, we could perform the following sequence of actions in APGsembly (where we perform the "`A`" action in the top row and then the "`B`" action directly below it before moving left-to-right):

```
                                ADD A1                ADD A1    ADD A1
ADD B1,   ADD B0,   ADD B0,   ADD B1,   ADD B1,   ADD B1,   ADD B0,   ADD B0.
```

Notice that these `ADD` actions are performed "backwards": they start from the least significant bits of the numbers that we are adding. Also, we need to append an extra `ADD B0` command at the very end to account for the fact that the sum may have one more bit than the summands.[29] These actions would return the outputs

```
NZ,   Z,   Z,   Z,   Z,   NZ,   Z,   NZ
```

corresponding to the fact that $104 + 57 = 161 = 10100001_2$. Complete APGsembly that implements this addition of two binary registers is provided in APGsembly 9.6.

APGsembly 9.6 APGsembly code for adding (left) or subtracting (right) the binary register `B1` to/from `B0`. In both cases, the new value is stored in `B0` while the value of `B1` is unaffected. The number of bits allocated to the binary registers is stored in `U0`, and the registers `U1` and `U2` are only used temporarily (they start at, and are returned to, a value of `0`).

```
#COMPONENTS B0-1,U0-2,ADD,NOP,HALT_OUT
#REGISTERS {'B0':[0,'0001011'], 'B1':[0,'100111'], 'U0':8}
# State    Input    Next state    Actions
# ---------------------------------------
INITIAL;   ZZ;      ADD1;         TDEC U0

# Copy U0 into U2, with the help of U1
ADD1;      Z;       ADD2;         TDEC U1
ADD1;      NZ;      ADD1;         TDEC U0, INC U1
ADD2;      Z;       ADD3;         TDEC U2
ADD2;      NZ;      ADD2;         TDEC U1, INC U0, INC U2

# Loop over U2 to add B1 to B0, one bit at a time.
ADD3;      Z;       ADD7;         TDEC B0
ADD3;      NZ;      ADD4;         READ B0
ADD4;      Z;       ADD5;         READ B1
ADD4;      NZ;      ADD5;         READ B1, ADD A1
ADD5;      Z;       ADD6;         ADD B0
ADD5;      NZ;      ADD6;         ADD B1, SET B1
ADD6;      Z;       ADD3;         TDEC U2, INC B0, INC B1
ADD6;      NZ;      ADD6;         SET B0, NOP

# Move the B0 and B1 read heads back to least significant bit.
ADD7;      Z;       ADD8;         TDEC B1
ADD7;      NZ;      ADD7;         TDEC B0
ADD8;      Z;       ADD8;         HALT_OUT
ADD8;      NZ;      ADD8;         TDEC B1
```

```
#COMPONENTS B0-1,U0-2,SUB,NOP,HALT_OUT
#REGISTERS {'B0':[0,'0001011'], 'B1':[0,'100111'], 'U0':8}
# State    Input    Next state    Actions
# ---------------------------------------
INITIAL;   ZZ;      SUB1;         TDEC U0

# Copy U0 into U2, with the help of U1
SUB1;      Z;       SUB2;         TDEC U1
SUB1;      NZ;      SUB1;         TDEC U0, INC U1
SUB2;      Z;       SUB3;         TDEC U2
SUB2;      NZ;      SUB2;         TDEC U1, INC U0, INC U2

# Loop over U2 to subtract B1 from B0, one bit at a time.
SUB3;      Z;       SUB7;         TDEC B0
SUB3;      NZ;      SUB4;         READ B0
SUB4;      Z;       SUB5;         READ B1
SUB4;      NZ;      SUB5;         READ B1, SUB A1
SUB5;      Z;       SUB6;         SUB B0
SUB5;      NZ;      SUB6;         SUB B1, SET B1
SUB6;      Z;       SUB3;         TDEC U2, INC B0, INC B1
SUB6;      NZ;      SUB6;         SET B0, NOP

# Move the B0 and B1 read heads back to least significant bit.
SUB7;      Z;       SUB8;         TDEC B1
SUB7;      NZ;      SUB7;         TDEC B0
SUB8;      Z;       SUB8;         HALT_OUT
SUB8;      NZ;      SUB8;         TDEC B1
```

Since the `ADD` component is somewhat technical and basically only has a single use, not much is lost if we do not actually understand its inner workings and just accept APGsembly 9.6 as a black

[27] Be careful: this means that `B0` and `B1` can be both components (as in `INC B0` or `READ B1`) and actions (as in `ADD B0`).

[28] This contrasts with the command `ADD B0`, which also does not alter the internal memory, but *does* return a value. In particular, `ADD B0` returns the contents of the internal memory and then resets that memory to `0`.

[29] Another reason that it is a good idea to call `ADD B0` after adding two registers is that it clears the internal memory of the `ADD` component.

Free PDF. See conwaylife.com/book for pattern files and print version.

box that adds up two binary registers. Similarly, we do not go into any great depth in explaining the inner working of the SUB component. Instead, we just note that it does for subtraction exactly what the ADD component does for addition—it keeps track of the borrow bit (which is analogous to the carry bit for addition) and thus lets us subtract binary registers from each other one bit at a time. The resulting APGsembly code for subtracting one binary register from another one is also provided in APGsembly 9.6. Note that this code for subtracting B1 from B0 assumes that B1 <= B0; if you are unsure of which binary register is the smaller one then you should first compare them (see Exercise 9.9).

Now that we know how to add and subtract binary components, we can also multiply and divide them via the same tricks that we used for sliding block registers in Section 9.3 and Exercise 9.4. That is, we simply add or subtract repeatedly (see Exercise 9.7). However, doing so can be quite time-consuming when the registers store large values, so we now introduce one additional custom binary register arithmetic component—one that helps us multiply a binary register by 10.[30]

This final new component is called MUL, which stores *four* carry bits so that we can multiply a binary register by 10 one bit at a time, just like the ADD and SUB components made use of a single memory bit to let us add and subtract binary registers one bit at a time.[31] The MUL component is displayed in Figure 9.11, though it is not very enlightening to look at.[32]

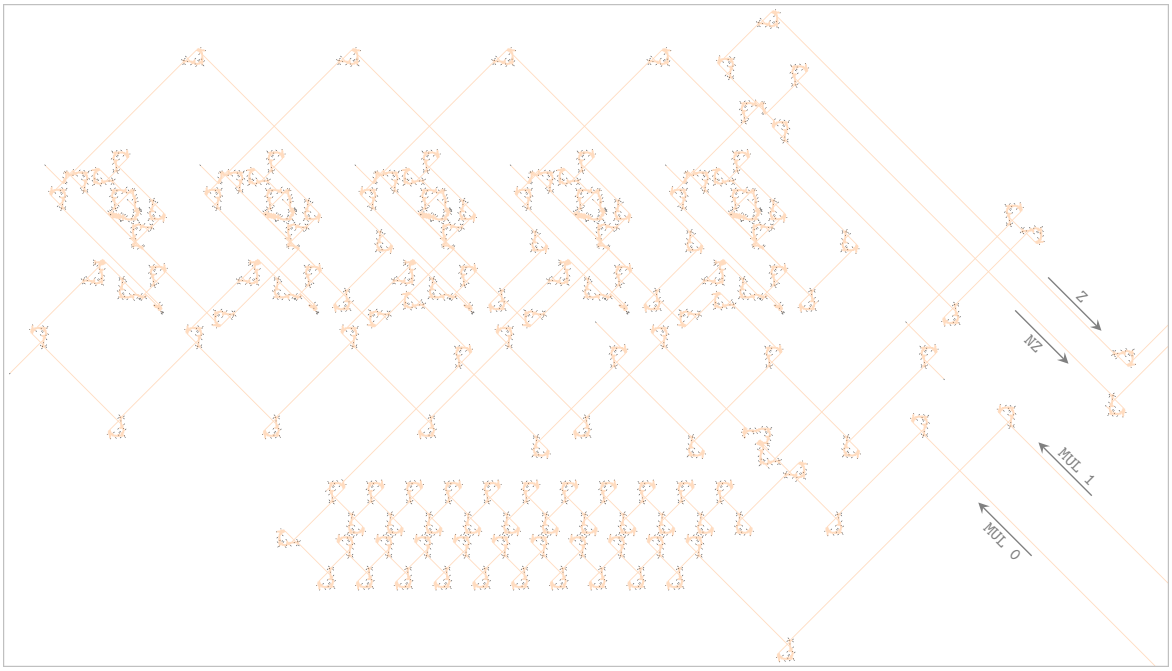


Figure 9.11: The MUL component, which keeps track of multiple carry bits so as to help us easily multiply a binary register by 10.

[30]The reason for us making 10 easier to multiply by (as opposed to any other fixed integer) is that multiplication by 10 is a common operation when working with decimal numbers. In particular, we will make use of this multiplication-by-10 component to help us extract digits in the upcoming π calculator in Section 9.6.

[31]The internal memory of these ADD, SUB, and MUL components is actually slightly more complicated than we have indicated. For example, the ADD component has *two* bits of internal memory since between the Ax call (if any) and the By call, it tracks the carry bit and the A input value separately. Those values cannot be combined and stored in a single one-bit memory cell, because an internal A0+(0 carry bit) results in a different internal state than A1+(1 carry bit), even though they both have a Z output.

[32]Even moreso than the binary register and ADD and SUB components, the MUL component could be made significantly smaller via modern machinery like Snarks and syringes.

Free PDF. See conwaylife.com/book for pattern files and print version.

This component is called in APGsembly via the commands `MUL 0` and `MUL 1`, where the number after the name of the component refers to the current bit of the binary register that we are multiplying by 10. For example, to multiply the number $26 = 11010_2$ by 10, we could perform the following sequence of actions in APGsembly:

`MUL 0, MUL 1, MUL 0, MUL 1, MUL 1, MUL 0, MUL 0, MUL 0, MUL 0.`

These `MUL` actions also start from the least significant bit of the number that we are multiplying by 10, just like the `ADD` and `SUB` actions. Also, we need to append enough `MUL 0` commands in order to retrieve the extra bits that the output will have. Four extra `MUL 0` commands always suffices, and these actions would return the outputs

`Z, Z, NZ, Z, Z, Z, Z, Z, NZ,`

corresponding to the fact that $26 \times 10 = 260 = 100000100_2$.

We can thus now multiply a binary register by 10 via approximately the same number of lines of APGsembly code as were required to add and subtract the values of binary registers. Complete APGsembly code for carrying out this multiplication is provided in APGsembly 9.7.

APGsembly 9.7 APGsembly code for multiplying the binary register `B0` by 10. The number of bits allocated to `B0` is stored in `U0`, and the registers `U1` and `U2` are only used temporarily (they start at, and are returned to, a value of 0). Compare with APGsembly 9.6 for addition and subtraction.

```
#COMPONENTS B0,U0-2,MUL,NOP,HALT_OUT
#REGISTERS {'B0':[0,'01011'], 'U0':9}
# State    Input    Next state    Actions
# ---------------------------------------
INITIAL;   ZZ;      MUL1;         TDEC U0

# Copy U0 into U2, with the help of U1
MUL1;      Z;       MUL2;         TDEC U1
MUL1;      NZ;      MUL1;         TDEC U0, INC U1
MUL2;      Z;       MUL3;         TDEC U2
MUL2;      NZ;      MUL2;         TDEC U1, INC U0, INC U2

# Loop over U2 to multiply B0 by 10, one bit at a time.
MUL3;      Z;       MUL6;         TDEC B0
MUL3;      NZ;      MUL4;         READ B0
MUL4;      Z;       MUL5;         MUL 0
MUL4;      NZ;      MUL5;         MUL 1
MUL5;      Z;       MUL3;         TDEC U2, INC B0
MUL5;      NZ;      MUL5;         SET B0, NOP

# Move the B0 read head back to its least significant bit.
MUL6;      Z;       MUL6;         HALT_OUT
MUL6;      NZ;      MUL6;         TDEC B0
```

9.5 A Character Printer

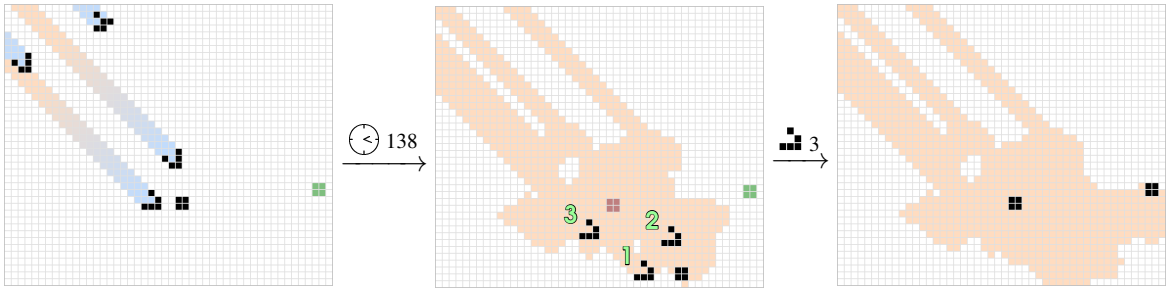
We now demonstrate how to construct a component that is capable of printing any characters (e.g., digits or letters) of our choosing on the Life plane, in a font made up of blocks. Fortunately, we have already seen most of the reactions that we need to assemble a printer of this type. Very much along

Free PDF. See conwaylife.com/book for pattern files and print version.

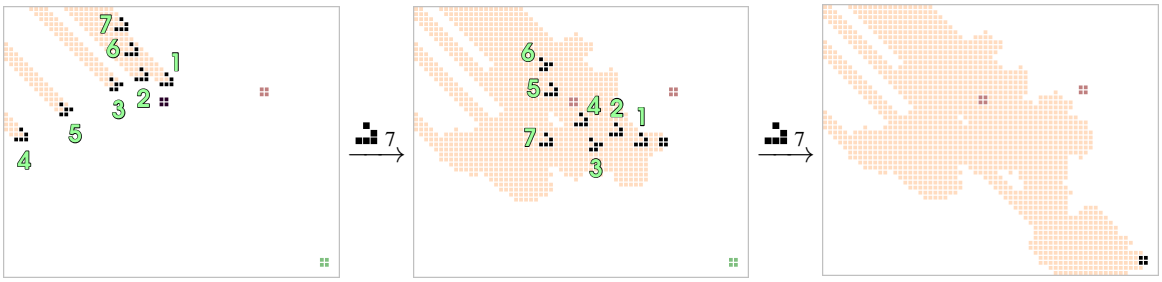
the same lines as the binary register of the previous section, our character printer will use a variant of slide gun technology to move a block that marks the current cursor position. We will also need a reaction that can use that cursor block to place another block nearby to serve as the pixels in printed characters (much like the read head block in the binary register can be used to place boats nearby to serve as bits in the register's memory).

In order to make our printer somewhat simpler to construct, we will have it move and position blocks via slow salvos like the ones that we introduced back in Section 5.7. Doing this lets us not have to worry about the precise timing of the gliders that we use to move and duplicate blocks—we just have to position them correctly (which is trivial via an array of edge-shooting Herschel conduits). This is the same technique that we used to construct the binary register in Figure 9.7, which has a long diagonal line of 26 edge shooters along its northeast edge.

One particular salvo for placing a pixel block is displayed in Figure 9.12(a). While this salvo is not quite slow (the first 3 of its 7 gliders must be synchronized, and the 4th glider must have the correct mod 8 timing), it has the added benefit of being unidirectional.[33] Similarly, Figure 9.12(b) shows a 14-glider unidirectional slow salvo that pushes the cursor block 32 full diagonals farther away from the glider source. Importantly, this cursor-pushing reaction can be used regardless of whether or not the pixel-placing reaction was used, since the pixel is placed in a location that is never touched by the cursor-pushing reaction.



(a) A 7-glider salvo that uses 4 synchronized gliders and then 3 slow gliders to create a far-away block, while only temporarily disturbing the target (cursor) block.



(b) A 14-glider p2 slow salvo that pushes the target (cursor) block 32 full diagonals away, while avoiding the location of the hypothetical printed pixel from (a).

Figure 9.12: Glider salvos that can be used to print block pixels at a separation of 32 full diagonals.

To create the salvos that implement these pixel-printing and cursor-moving operations, we just use stable circuitry like Snarks, syringes, and the edge-shooting Herschel conduits that we saw in Chapter 7. If we want the next pixel in the current row to be empty, we just push the cursor block by creating the glider salvo from Figure 9.12(b). If we want to print the next pixel in the current row, we

[33] We have already seen the slow portion of this salvo—the first slow glider is the $(2, 1)$ block pull from Figure 2.20, and the second and third gliders duplicate the block as in Figure 5.20. The first four (synchronized) gliders just reposition the cursor block so that these later (slow) gliders, when duplicating the block, move it back to where it started.

Free PDF. See conwaylife.com/book for pattern files and print version.

instead create the salvo from Figure 9.12(a) and *then* trigger the cursor-pushing mechanism. A device called a **row printer** that implements both of these operations is displayed in Figure 9.13.

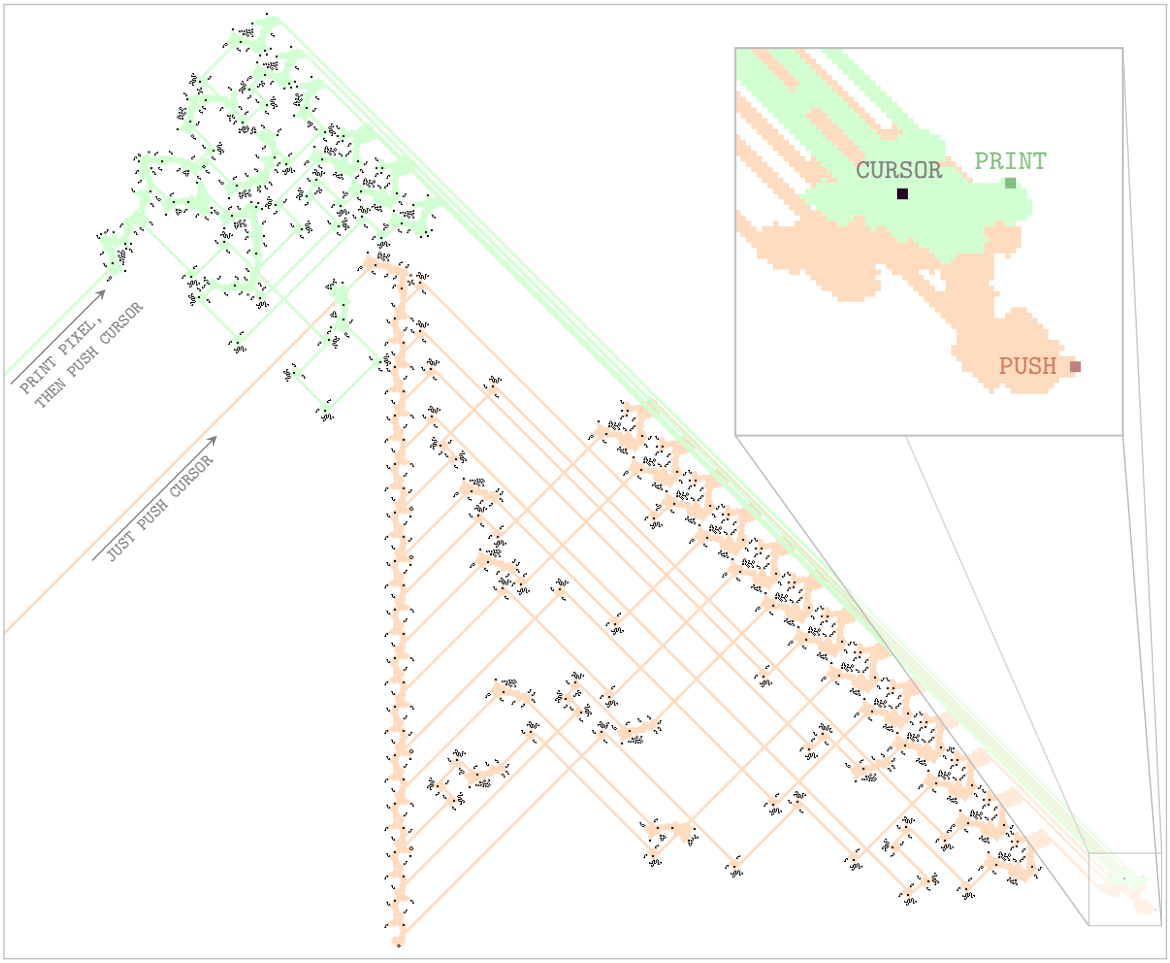


Figure 9.13: A **row printer** that can be used to either push a cursor block forward by 32 full diagonals (highlighted in orange), or print a pixel block (highlighted in green) and *then* push the cursor block.

This row printer lets us easily print an arbitrary sequence of pixels (blocks) along a single (northwest-to-southeast diagonal) row. To construct a more general printer that lets us print characters that are several pixels tall, we simply use multiple row printers—one cursor block and one row printer per pixel of height in the characters. For example, if we wish to print characters that are 10 pixels tall, we use 10 cursor blocks that are offset from each other diagonally in the southwest-to-northeast direction, and we aim a separate row printer at each of them.

The end result of using multiple row printers in this way is that each row always advances its cursor block by the same distance (32 full diagonals), regardless of whether or not a pixel has been placed in that row. Instructions for which pixels to print in each row (and thus which characters to print overall) can be encoded straightforwardly in circuitry that sends a glider into the appropriate input slot of each row printer. To illustrate this idea, Figure 9.14 presents a pattern that is capable of printing the characters `A`, `B`, `C`, `D`, and `R` in a font made up of blocks, and which has had 11 glider signals fed into it, instructing it to print the word "`ABRACADABRA`".

This very simple printer design adapts readily to any character height or width, and any number of

Free PDF. See conwaylife.com/book for pattern files and print version.

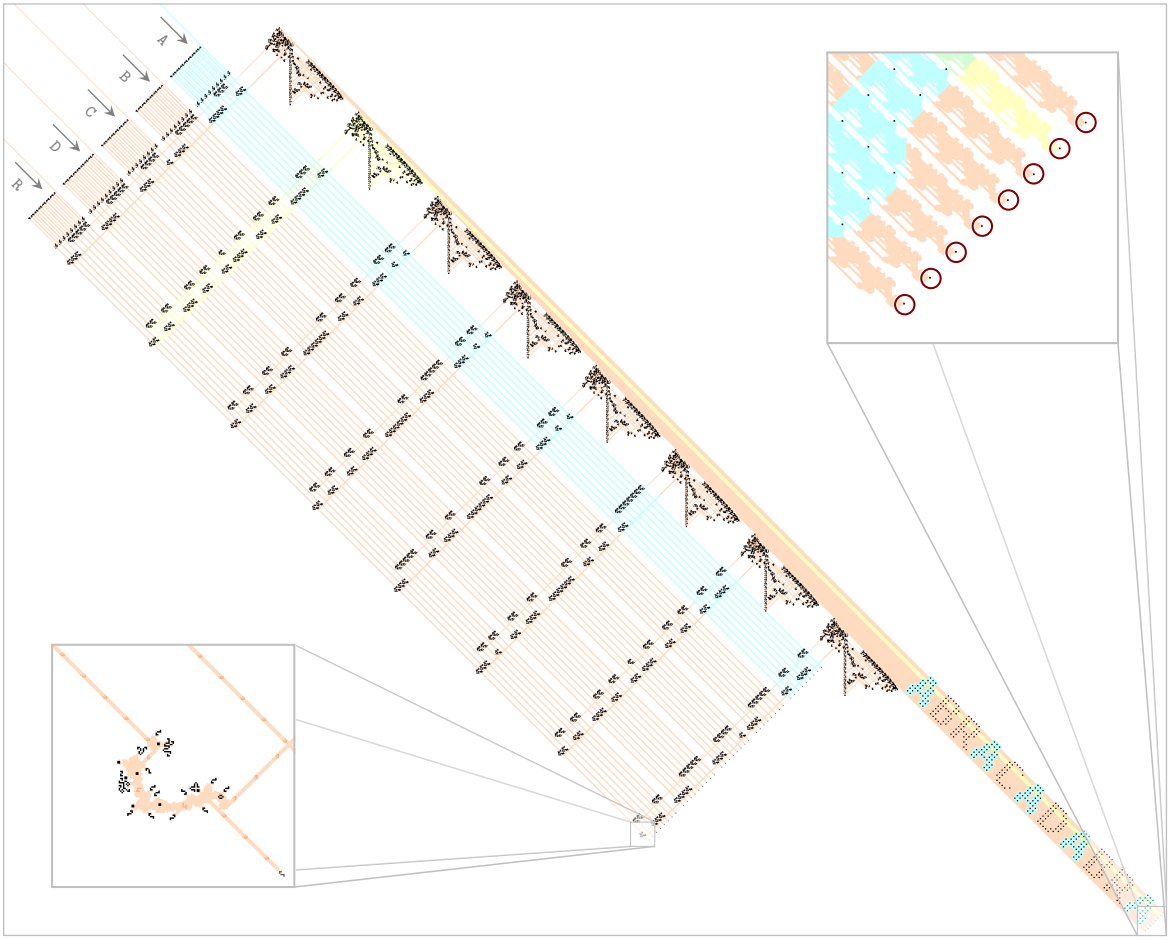


Figure 9.14: A printer with five input glider lanes that direct it to print one of the letters `A`, `B`, `C`, `D`, or `R` in a font made up of blocks, shown here having already printed the word "`ABRACADABRA`". The font used is 8 pixels (blocks) high, so 8 row printers and 8 cursor blocks (circled in red) are used. The row printer highlighted in yellow takes care of all printing on the second-from-the-top row, for example. The rectangular array of splitters below the row printers encodes which pixels should be printed in each row. For example, the arrangement of splitters highlighted in aqua encodes which pixels should be printed by each row printer when an "`A`" input is received.

characters.[34] There is no requirement that all characters have the same width. They are technically all the same height, though they may appear to be different heights if some characters are padded with empty pixels at the top or bottom.

While APGsembly requires us to explicitly specify which digit we want to print (e.g., we cannot print the contents of a sliding block register `U0` via the command `OUTPUT U0`), we can get around this limitation by repeatedly `TDEC`ing that register to determine its value, and then printing out that value. This idea is made explicit by APGsembly 9.8. Note that this APGsembly requires that `U0` is storing a value between 0 and 9, inclusive, but it could be modified straightforwardly to handle any fixed upper bound just by adding additional `TDEC` statements. It is even possible to print the contents of a sliding block register regardless of how many decimal digits it has, but this is much more complicated—see Exercise 9.22.

[34] A script that builds a printer pattern for any specified set of characters can be downloaded from conwaylife.com/forums/viewtopic.php?p=93420#p93420.

Free PDF. See conwaylife.com/book for pattern files and print version.

APGsembly 9.8 APGsembly code for printing the contents of the sliding block register U0, which is assumed to contain a value between 0 and 9 inclusive, and setting U0 = 0 at the same time.

```
#COMPONENTS U0,OUTPUT,HALT_OUT
#REGISTERS {'U0':7}
# State    Input    Next state    Actions
# ---------------------------------------
INITIAL;   ZZ;      OUT0;         TDEC U0
OUT0;      Z;       OUT0;         OUTPUT 0, HALT_OUT
OUT0;      NZ;      OUT1;         TDEC U0
OUT1;      Z;       OUT1;         OUTPUT 1, HALT_OUT
OUT1;      NZ;      OUT2;         TDEC U0
OUT2;      Z;       OUT2;         OUTPUT 2, HALT_OUT
OUT2;      NZ;      OUT3;         TDEC U0
OUT3;      Z;       OUT3;         OUTPUT 3, HALT_OUT
OUT3;      NZ;      OUT4;         TDEC U0
OUT4;      Z;       OUT4;         OUTPUT 4, HALT_OUT
OUT4;      NZ;      OUT5;         TDEC U0
OUT5;      Z;       OUT5;         OUTPUT 5, HALT_OUT
OUT5;      NZ;      OUT6;         TDEC U0
OUT6;      Z;       OUT6;         OUTPUT 6, HALT_OUT
OUT6;      NZ;      OUT7;         TDEC U0
OUT7;      Z;       OUT7;         OUTPUT 7, HALT_OUT
OUT7;      NZ;      OUT8;         TDEC U0
OUT8;      Z;       OUT8;         OUTPUT 8, HALT_OUT
OUT8;      NZ;      OUT8;         OUTPUT 9, HALT_OUT
```

9.6 A π Calculator

We now put all of the pieces that we have developed so far in this chapter together to create one of the most remarkable patterns that has ever been constructed in Life: one that computes and prints the decimal digits of the mathematical constant $\pi = 3.14159\ldots$.

Before we can start writing APGsembly code for our calculator, we have to choose which algorithm we will use to compute π. There are hundreds of known algorithms for this purpose, but because of how APGsembly works, we would like one that has the following two features:

- It makes use entirely of integer arithmetic, rather than floating-point arithmetic.[35] This is important because the computational mechanisms that we have developed only work with non-negative integers.
- It is "streaming": after producing a particular digit of π, it can carry on to produce its next digit without having to start over from scratch. This is important because we want our calculator to just keep on printing out digits of π forever, without us having to specify how many digits we want ahead of time.

We now describe one algorithm for computing the digits of π (originally developed in [Gib06]) that satisfies the criteria outlined above. This algorithm is based on the following infinite series representation for π (see Exercise 9.16 for an explanation of where this series comes from):

$$\pi = 2\left(1+\frac{1}{3}\left(1+\frac{2}{5}\left(1+\frac{3}{7}\left(\cdots\left(1+\frac{k}{2k+1}\left(\cdots\right)\right)\right)\right)\right)\right). \tag{9.1}$$

[35] We can turn floating-point arithmetic into integer arithmetic by multiplying all numbers involved by large powers of 10, but we then have to be very careful to deal with numerical accuracy concerns, and the integers we would have to use would be monstrously large. It will be better to use an algorithm that is *designed* to only use integer arithmetic.

Free PDF. See conwaylife.com/book for pattern files and print version.

If we truncate the above series after a finite number of additions, we get better and better approximations of π. For example,

$$\begin{aligned} 2 &= 2.0000, & 2\left(1+\frac{1}{3}\right) &\approx 2.6667, \\ 2\left(1+\frac{1}{3}\left(1+\frac{2}{5}\right)\right) &\approx 2.9333, & 2\left(1+\frac{1}{3}\left(1+\frac{2}{5}\left(1+\frac{3}{7}\right)\right)\right) &\approx 3.0476, \end{aligned} \tag{9.2}$$

and so on, with these terms getting closer and closer to $\pi \approx 3.14159\ldots$.

As-written, using these approximations to compute digits of π satisfies neither the streaming property that we wanted—it is not clear how to compute one of the approximations based on previous approximations without starting over from scratch—nor the property that it only uses integer arithmetic. To fix these problems and get an algorithm that is reasonable to implement in APGsembly, we now introduce a slightly different way of computing these exact same approximations of π.

To this end, we define the following sequence of 2×2 matrices:

$$A_0 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad A_k = \begin{bmatrix} k & 2k+1 \\ 0 & 2k+1 \end{bmatrix} \quad \text{for all} \quad k \geq 1. \tag{9.3}$$

For example,

$$A_1 = \begin{bmatrix} 1 & 3 \\ 0 & 3 \end{bmatrix}, \; A_2 = \begin{bmatrix} 2 & 5 \\ 0 & 5 \end{bmatrix}, \; A_3 = \begin{bmatrix} 3 & 7 \\ 0 & 7 \end{bmatrix}, \quad \text{and} \quad A_4 = \begin{bmatrix} 4 & 9 \\ 0 & 9 \end{bmatrix},$$

and so on. In particular, the entries of each A_k are all non-negative integers.

Next, for each integer $n \geq 1$ define B_n to be the product of the first $n+1$ of the A_ks: $B_n = A_0A_1A_2\cdots A_n$.[36] For example,

$$\begin{aligned} B_1 &= A_0A_1 = \begin{bmatrix} 2 & 6 \\ 0 & 3 \end{bmatrix}, & B_2 &= A_0A_1A_2 = \begin{bmatrix} 4 & 40 \\ 0 & 15 \end{bmatrix}, \\ B_3 &= A_0A_1A_2A_3 = \begin{bmatrix} 12 & 308 \\ 0 & 105 \end{bmatrix}, \quad \text{and} & B_4 &= A_0A_1A_2A_3A_4 = \begin{bmatrix} 48 & 2880 \\ 0 & 945 \end{bmatrix}. \end{aligned}$$

Again, each B_n is an integer matrix with a 0 in its bottom-left corner. Importantly, each B_n can be easily obtained from the previous one, since $B_n = B_{n-1}A_n$ for all $n \geq 2$.

If we let q_n and r_n denote the top-right and bottom-right entries of B_n, respectively, then q_n/r_n is exactly the n-th approximation of π from Equation (9.2) (a fact that we will prove in Exercise 9.17). For example,

$$\begin{aligned} \frac{q_1}{r_1} &= \frac{6}{3} = 2.0000, & \frac{q_2}{r_2} &= \frac{40}{15} \approx 2.6667, \\ \frac{q_3}{r_3} &= \frac{308}{105} \approx 2.9333, \quad \text{and} & \frac{q_4}{r_4} &= \frac{2880}{945} \approx 3.0476. \end{aligned}$$

This gives us a way of computing better and better approximations of π using just integer arithmetic (except for the very last step, where we divide q_n by r_n), and which is streaming (since each B_n can be computed straightforwardly from B_{n-1}).

[36] Here we are using matrix multiplication, which works via the formula $\begin{bmatrix} a & b \\ 0 & c \end{bmatrix}\begin{bmatrix} d & e \\ 0 & f \end{bmatrix} = \begin{bmatrix} ad & ae+bf \\ 0 & cf \end{bmatrix}$.

Free PDF. See conwaylife.com/book for pattern files and print version.

Since $k/(2k+1) \approx 1/2$ when k is large in the series (9.1), the distance between π and our approximation q_n/r_n decreases by roughly a factor of 2 every time we increase n by 1. We thus need to iterate the above procedure an average of $\log_2(10) \approx 3.3219$ times for each decimal place of accuracy that we would like. It follows that if we want n decimal places of π, it suffices to use the approximation q_{4n}/r_{4n} based on B_{4n} (we rounded 3.3219 up to 4 for simplicity).[37] For example,

$$B_4 = \begin{bmatrix} 48 & 2880 \\ 0 & 945 \end{bmatrix}, \qquad \frac{q_4}{r_4} = \frac{2880}{945} \approx \mathbf{3}.0476,$$

$$B_8 = \begin{bmatrix} 80640 & 108103680 \\ 0 & 34459425 \end{bmatrix}, \qquad \frac{q_8}{r_8} = \frac{108103680}{34459425} \approx \mathbf{3.1}371,$$

$$B_{12} = \begin{bmatrix} 958003200 & 24835120128000 \\ 0 & 7905853580625 \end{bmatrix}, \qquad \frac{q_{12}}{r_{12}} = \frac{24835120128000}{7905853580625} \approx \mathbf{3.14}14,$$

are approximations of π that are accurate to at least 1, 2, and 3 decimal places, respectively.

We can now put all of this together into the reasonably APGsembly friendly Pseudocode 9.3 for computing and printing the decimal digits of π. In this pseudocode, the registers `U0`, `U1`, `U2`, `U3`, `B0`, `B1`, and `B2` store the following quantities:

- `U0`: top-left corner of the A_n matrix
- `U1`: top-right and bottom-right corners of the A_n matrix
- `U2`: the current digit being computed
- `U3`: the index of the current digit (`U3` $= 0$ for "3", `U3` $= 1$ for "1", `U3` $= 2$ for "4", and so on)
- `B0`: top-left corner of the B_n matrix
- `B1`: top-right corner of the B_n matrix ($= q_n$)
- `B2`: bottom-right corner of the B_n matrix ($= r_n$)

The reason that we use the "`B`" labels for the entries of B_n and "`U`" labels for other quantities is that the entries of B_n grow quite large as n increases, so they should be stored in binary registers, while all other quantities used in this algorithm are comparatively small, so they can simply be stored in sliding block (unary) registers.

We have seen how to implement most of the pieces of Pseudocode 9.3 in APGsembly already. Adding fixed values to sliding block (unary) registers is as straightforward as using their `INC` command the corresponding number of times. Looping over a code section four times can be done by setting a register's value to 4 and then using a `TDEC` command to decide whether to restart the loop or exit it. Finally, adding binary registers together and multiplying them by unary registers can be done via the code that we saw in APGsembly 9.6 and Exercise 9.7, respectively.

However, one piece of Pseudocode 9.3 that is tricky to implement is the digit extraction at line 15. To carry out this step using just integer arithmetic, we do the following variant of integer division:

1) Set $s = 0$.
2) Subtract `B2` from `B1` until we cannot do so anymore. The number of subtractions that we performed is the s-th digit of `B1 / B2` after its decimal point. If this is the digit that we are currently interested in, we can stop. Otherwise, we proceed to step (3) below.
3) Multiply `B1` by 10, increment s by 1, and then return to step (2) above.

[37] There is an extraordinarily small chance that this algorithm computes an incorrect digit of π at some point. The argument we just gave guarantees that we extract its n-th digit from an approximation that is within about 2^{-4n} of the true value of π. Since $2^{-4n} = 16^{-n} < 10^{-n}$, the n-th digit of this approximation matches the n-th digit of π as long as π does not have an exceptionally long string of consecutive 0s in its decimal expansion significantly before we would expect to see one (since that could cause a long string of 9s in one approximation that turns into a string of 0s in the next approximation and π itself). However, over 62 000 000 000 000 decimal places of π are known, and this algorithm computes them all correctly.

Free PDF. See conwaylife.com/book for pattern files and print version.

Pseudocode 9.3 Pseudocode for computing and printing the decimal digits of π. Any variable that is used before it is set is assumed to start at a value of 0.

```
# Initialize the A and B matrices.
set U1 = 1, B0 = 2, B2 = 1

# Never stop computing and printing digits.
loop forever:
    # Iterate 4 times before printing a digit.
    loop 4 times:
        # Update the A matrix.
        set U0 = U0 + 1, U1 = U1 + 2

        # Update the B matrix.
        set B0 = U0 * B0
        set B1 = U1 * (B0 + B1)
        set B2 = U1 * B2
    end loop

    # Compute and print the desired digit of B1 / B2.
    set U2 = the U3-th digit after the decimal point of B1 / B2
    OUTPUT U2

    # Print a decimal point after the 0-th digit (3), and loop.
    if U3 = 0 then OUTPUT .
    set U3 = U3 + 1
end loop
```

In order to implement step (2) above in APGsembly, we need to know how to subtract the value of one binary register from another, and how to compare the values of two binary registers (so that we know whether or not we can do the subtraction in the first place). Fortunately, we saw how to do the former of these tasks in APGsembly 9.6, and the latter task is somewhat simpler and left to Exercise 9.9. Similarly, we saw how to do the multiplication by 10 that step (3) requires us to do back in APGsembly 9.7.

We have thus seen all of the pieces that we need to finally construct our π-calculating Life pattern. The APGsembly code that implements Pseudocode 9.3 is quite long, so we leave it to Appendix B.8. The resulting calculator that is compiled from that APGsembly code is presented in Figure 9.15.

9.6.1 Computing Other Constants

This same algorithm that we used in our π calculator can also be used to compute many other well-known mathematical constants. In particular, we can use that same method to construct patterns that print the digits of any number that can be represented by a series of the following form, where a, b, p, q, r, and s are non-negative integers:[38]

$$\begin{aligned}&\frac{a}{b}\sum_{k=0}^{\infty}\frac{(p+q)(2p+q)\cdots(kp+q)}{(r+s)(2r+s)\cdots(kr+s)}\\&\qquad=\frac{a}{b}\left(1+\frac{p+q}{r+s}\left(1+\frac{2p+q}{2r+s}\left(1+\frac{3p+q}{3r+s}\left(\cdots\left(1+\frac{kp+q}{kr+s}\left(\cdots\right)\right)\right)\right)\right)\right).\end{aligned}\tag{9.4}$$

For example, the π series (9.1) arises in the case when $a=2$, $b=1$, $p=1$, $q=0$, $r=2$, and $s=1$.

[38] If $k=0$ then the term $\frac{(p+q)(2p+q)\cdots(kp+q)}{(r+s)(2r+s)\cdots(kr+s)}$ is an "empty product", which equals 1.

Free PDF. See conwaylife.com/book for pattern files and print version.

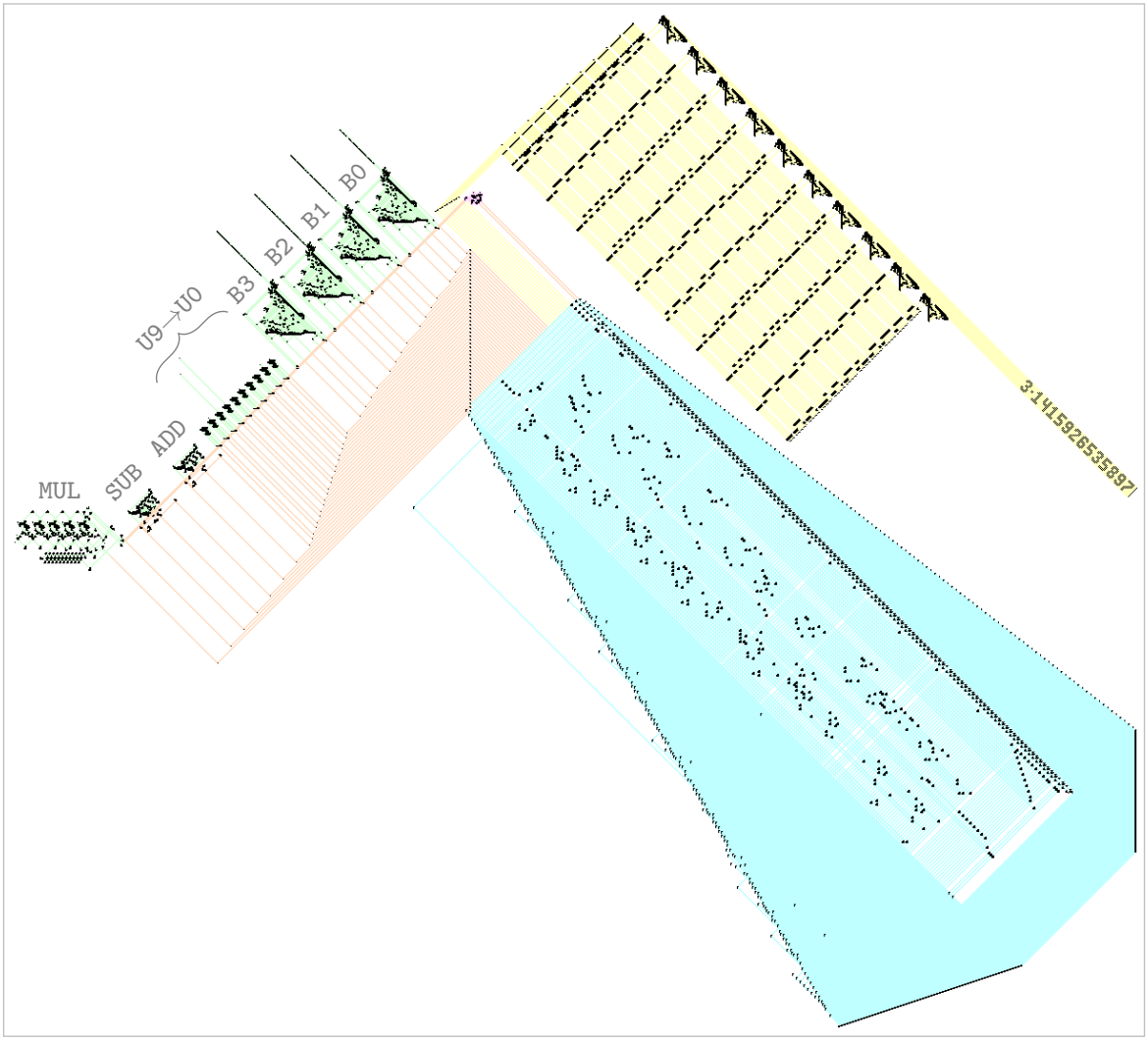


Figure 9.15: A π calculator that uses a massive computer (highlighted in aqua) based on the APGsembly code from Appendix B.8. It feeds into a component stack (highlighted in green) that contains every component that we have seen so far: 10 sliding block registers, 4 binary registers, and one each of the `ADD`, `SUB`, and `MUL` components. It also feeds into a character printer (highlighted in yellow) which, as of generation 8.3×10^{13}, has printed the first 14 digits of π: 3.1415926535897.

All that changes in the algorithm and APGsembly code for computing a constant of this more general form, rather than π itself, is that the A_k matrices from Equation (9.3) should instead be defined by

$$A_0 = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \quad \text{and} \quad A_k = \begin{bmatrix} kp+q & kr+s \\ 0 & kr+s \end{bmatrix} \quad \text{for all} \quad k \geq 1.$$

The computation of the B_n matrices from the A_ks, as well as the remainder of the algorithm and pseudocode, stays the exact same.

To illustrate how to make this change to construct a calculator for a number other than π, consider the mathematical constant $e = 2.71828\ldots$, which makes frequent appearances in calculus and statistics.

Free PDF. See conwaylife.com/book for pattern files and print version.

It has the well-known series representation

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} = 1 + \left(1 + \frac{1}{2}\left(1 + \frac{1}{3}\left(1 + \frac{1}{4}\left(\cdots\right)\right)\right)\right), \tag{9.5}$$

corresponding to the values $a = 1$, $b = 1$, $p = 0$, $q = 1$, $r = 1$, and $s = 0$ in the general series (9.4). We can thus make the following two extremely minor changes to Pseudocode 9.3 for computing π to turn it into an algorithm for computing e:

- Replace line 2 with "`set U0 = 1, B0 = 1, B2 = 1`".
- Replace line 8 with "`set U1 = U1 + 1`".

More generally, to turn Pseudocode 9.3 into pseudocode for computing a number via the series (9.4), we make the following changes based on the values of a, b, p, q, r, and s:

- Replace line 2 with "`set U0 = q, U1 = s, B0 = a, B2 = b`".
- Replace line 8 with "`set U0 = U0 + p, U1 = U1 + r`".

Modifying the APGsembly code for the π calculator from Appendix B.8 is similarly straightforward. It can be turned into APGsembly for an e calculator by just changing two or three lines of code: the `#REGISTERS` line, the `ITER6;NZ` line, and optionally the `ITER7;ZZ` line (see Exercise 9.18). Other constants like $\sqrt{2}$ can similarly be computed by making analogous minimal changes (see Exercise 9.19).

9.7 A 2D Printer

While the character printer that we introduced in Section 9.5 is quite flexible in that we can re-tool it to print any finite set of digits of our choosing, it is still limited by the fact that it can only print them linearly. The final component that we introduce, which we denote by `B2D`, solves this problem by being able to print an arbitrary pattern in an infinite 2D quadrant of the Life plane. Alternatively, this component can be thought of as a 2D generalization of the binary register from Section 9.4: while that component kept track of an infinite 1D *list* of bits, this one keeps track of an infinite 2D *array* of bits.[39]

The way that this 2D printer works is almost the exact same as the binary register, but with two perpendicular sliding blocks to read and write the desired bits instead of just one. In particular, those sliding blocks are used to fire gliders at each other so as to synthesize or destroy boats that are laid out in a diagonal grid with a separation of 16 full diagonals. These boats can either be interpreted as bits in some sort of computation (with a boat representing a `1` and an empty space representing a `0`), or they can be thought of as pixels in an image that can be viewed by zooming far out in Life simulation software.

In order to actually print those pixels (i.e., boats), the 2D printer can be called via APGsembly actions that are almost identical to those of the binary register, but with the minor extra complexity that we can increment and decrement the location of the 2D printer's read head in two different perpendicular directions. We say that the "x" direction of this grid is the diagonal running from southwest to northeast, and the "y" direction is the one running from southeast to northwest (so that the coordinate system for this printer is just the usual Cartesian coordinate system, but rotated counter-clockwise by 45 degrees). Only points with non-negative x and y coordinates are accessible to the printer, so it can print on the top-central quadrant of the Life plane.

The APGsembly actions themselves are listed below, and since these are the last components and actions that we will make use of in this computational toolkit, this is a natural time to summarize them all. This summary is provided in Table 9.1.

[39] This interpretation explains the abbreviation `B2D` for this component; it stands for "binary 2-dimensional".

Free PDF. See conwaylife.com/book for pattern files and print version.

- `INC B2DX` and `INC B2DY`, which move the pointer blocks of the read head one step farther away from the printer itself (northeast or northwest, respectively).
- `TDEC B2DX` and `TDEC B2DY`, which move the pointer blocks to the previous location in the x or y direction (i.e., southwest or southeast), respectively, or keep them in the same place if they're already at the zero position. These actions also return either `Z` or `NZ`, depending on whether or not the x or y read head was already at the `0` location.
- `READ B2D`, which sets the value of the bit at the current read head location to "`0`". This action returns either `Z` or `NZ`, depending on the original value of this bit.
- `SET B2D`, which places a "`1`" bit at the current read head location.[40]

The 2D printer is extremely large, so we do not display it here, but we will see it in the upcoming Figure 9.17. It is worth noting that this component is so slow and large that we actually have to use a slightly slower clock gun whenever it is present—one with period at least 2^{22} (instead of period 2^{20}, like we have used until now). Indeed, if we use a clock gun that is too fast then a return value from another component might arrive back at the clock gun, starting the next clock tick, while the `B2D` component is still working on its previous action. If another action is then sent to the `B2D` component, it could lead to an unwanted collision.

9.7.1 Printing a Fractal

To illustrate the flexibility of this 2D printer, we now demonstrate how it can be used to print a fractal. Slightly more accurately, we use it to print better and better approximations of a fractal, which of course is the best we could hope for—our printing capabilities are limited by the fact that we are working on the Life plane and thus must make our images out of pixels.

The way that we construct the fractal is to imagine walking along some path in the plane, printing a pixel after every step that we take. We always step a distance of 1 in one of the eight orthogonal or diagonal directions (in the Moore neighborhood sense, so increasing both the x- and y-coordinates by 1, for example, is a valid move). The path that we walk along is illustrated in Figure 9.16, and is determined as follows:

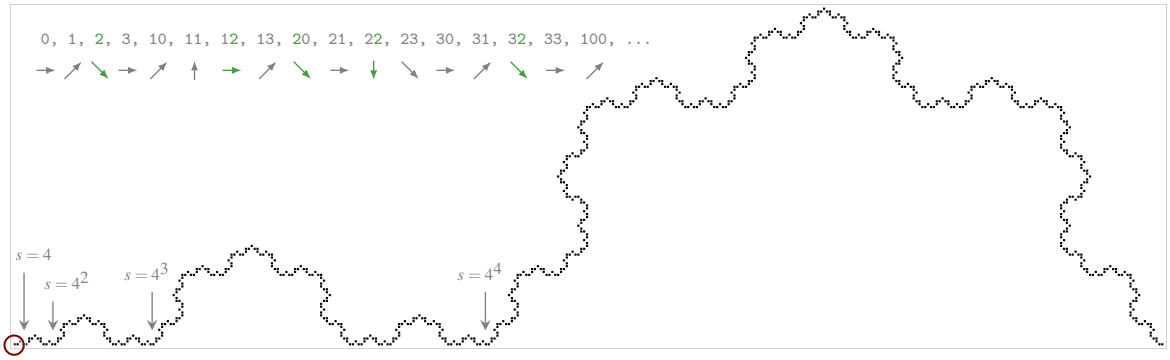


Figure 9.16: The (leftmost portion of the) image that results from starting at the bottom-left corner (circled in red) and printing pixels according to a path-following procedure based on counting in base 4 (in the variable s). These bulbous shapes are better and better approximations of the top half of an 8-pointed version of the Koch snowflake fractal.

[40] Just like the binary register's `SET` action, this mechanism will self-destruct if a pixel is `SET` when a boat is already present at the current read head location.

Free PDF. See conwaylife.com/book for pattern files and print version.

Component	Actions	Functionality and return value
`Un`: sliding block register stores a non-negative integer in unary	`INC Un`	increases the value of the register by `1`, no return value
	`TDEC Un`	returns `Z` if `Un = 0` and `NZ` otherwise, and *then* decreases the value of the register by `1` if `NZ`
`Bn`: binary register stores a non-negative integer in binary	`INC Bn`	increases the position of the read head, no return value
	`TDEC Bn`	returns `Z` if read head is at least significant bit and `NZ` otherwise, and *then* decreases position by `1` if `NZ`
	`READ Bn`	returns the bit (`Z = 0` or `NZ = 1`) at the read head, and then sets it equal to `0`
	`SET Bn`	set the bit at the read head to `1`, no return value, breaks if that bit already equals `1`
`B2D`: 2D binary register stores a 2D array of bits	`INC B2DX`	increases the X position of the read head, no return value
	`INC B2DY`	increases the Y position of the read head, no return value
	`TDEC B2DX`	returns `Z` if X read head is at least significant bit and `NZ` otherwise, and *then* decreases X position by `1` if `NZ`
	`TDEC B2DY`	returns `Z` if Y read head is at least significant bit and `NZ` otherwise, and *then* decreases Y position by `1` if `NZ`
	`READ B2D`	returns the bit (`Z = 0` or `NZ = 1`) at the read head, and then sets it equal to `0`
	`SET B2D`	set the bit at the read head to `1`, no return value, breaks if that bit already equals `1`

`ADD`: binary adder	`ADD A1`
	`ADD B0`
	`ADD B1`
		helps us add one binary register to another one, as in APGsembly 9.6

`SUB`: binary subtractor	`SUB A1`
	`SUB B0`
	`SUB B1`
		helps us subtract one binary register from another one, as in APGsembly 9.6

`MUL`: binary multiplier	`MUL 0` `MUL 1`	helps us quickly multiply a binary register by `10`, as in APGsembly 9.7
`OUTPUT`: digit printer	`OUTPUT x`	prints `x` in a font made up of blocks, no return value, `x` must be one of `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, or `.`
`NOP`: no operation	`NOP`	returns `Z` and does nothing else
`HALT_OUT`	`HALT_OUT`	halts the entire computation and emits a glider

Table 9.1: A summary of the standard components that APGsembly can make use of and the actions that they can perform.

Free PDF. See conwaylife.com/book for pattern files and print version.

1) Start at the coordinate $(x,y) = (0,0)$, facing right (toward the coordinate $(1,0)$).
2) Color in the pixel at the current location and then walk forward one step.
3) Let s be the total number of steps that we have taken so far.
   - If s, when represented in base 4, has more "2" digits than $s-1$, turn clockwise by 90 degrees.
   - Otherwise, turn counter-clockwise by 45 degrees.
4) Return to Step (2).

The above procedure results in a path that, on average, moves straight to the right. After all, every time we increase the total number of steps s by one, exactly one digit rolls over to either 1, 2, or 3. Two of those three possibilities result in our orientation rotating 45 degrees counter-clockwise, and the other one of those three possibilities results in our orientation rotating 90 degrees counter-clockwise, for a total average of no rotation at all.

The fractal-like behavior of this path comes from the repetitive nature of digit strings that are encountered when counting. Indeed, if we ever encounter a particular (base 4) string of digits in s when counting, we will encounter that exact same string of digits infinitely many times later on too, as additional leading digits are added to s. The resulting fractal is the top half of an 8-pointed version of a well-known fractal called the **Koch snowflake** (which is usually 6-pointed).

APGsembly code that implements this algorithm is displayed in APGsembly 9.9. Although it is somewhat long, it is reasonably straightforward and makes use of 5 sliding block (unary) registers and one binary register to keep track of the following quantities:

`U0`: stores the x-direction to move (0 = none, 1 = right, 2 = left)
`U1`: stores the y-direction to move (0 = none, 1 = up, 2 = down)
`U2`: a value that determines which of the 8 possible directions to move in at the next step (0 = right, 1 = up-right, 2 = up, ..., 7 = down-right)—`U0` and `U1` are computed based on this
`U3`: a temporary helper register
`U4`: the amount to add to `U2` after we encounter an extra "2" when counting in base 4 (stays constant at 6, corresponding to a clockwise turn by 90 degrees)
`U5`: how much to mod `U2` by after adding to it (stays constant at 8)
`B0`: keeps track of the number of steps that we have taken in base 4 ($= s$)

Since the 2D printer component prints on a diagonal grid, the Life pattern that implements this APGsembly code[41] prints the image from Figure 9.16 rotated counter-clockwise by 45 degrees. We leave the actual compilation of this Life pattern to Exercise 9.20.

9.7.2 An Extremely Slowly Growing Pattern

We now return to the problem of creating a pattern with a very slowly growing bounding box, which we first considered via a binary ruler in Section 9.4.1. In an $n \times n$ bounding box, there are n^2 different cells that can each be in one of 2 states, for a total of 2^{n^2} possible different patterns. It follows that if a pattern exhibits infinite growth then it cannot stay within an $n \times n$ bounding box past generation $t = 2^{n^2}$, since after that point in time its phases would necessarily start repeating.

If we flip this argument around and solve for n in terms of t, we see that in generation t, the bounding box of an infinitely growing pattern can be no smaller than $\sqrt{\log_2(t)} \times \sqrt{\log_2(t)}$.[42] We now construct a pattern that attains this minimum possible bounding box growth rate, at least asymptotically—its diameter in generation t is $\Theta(\sqrt{\log(t)})$.

[41] Originally constructed by Michael Simkin in 2019.

[42] This contrasts with our results of Section 8.8.3: even though there is no slowest *population* growth rate, there is a slowest *bounding box* growth rate.

Free PDF. See conwaylife.com/book for pattern files and print version.

APGsembly 9.9 APGsembly code for the 8-pointed Koch snowflake printer.

```
#COMPONENTS U0-5,B0,B2D,NOP
#REGISTERS {'U4':6, 'U5':8}
# State   Input   Next state   Actions
# ---------------------------------------
INITIAL;  ZZ;     DIR0;        TDEC U2

# Update U0 and U1, based on U2, so that we walk in the correct
# direction. Also set U3 = U2 and then U2 = 0.
DIR0;     Z;      RESETU2;     TDEC U3, INC U0
DIR0;     NZ;     DIR1;        TDEC U2, INC U3
DIR1;     Z;      RESETU2;     TDEC U3, INC U0, INC U1
DIR1;     NZ;     DIR2;        TDEC U2, INC U3
DIR2;     Z;      RESETU2;     TDEC U3, INC U1
DIR2;     NZ;     DIR3;        TDEC U2, INC U3
DIR3;     Z;      DIRX;        INC U0, INC U1, NOP
DIR3;     NZ;     DIR4;        TDEC U2, INC U3
DIR4;     Z;      DIRX;        INC U0, NOP
DIR4;     NZ;     DIR5;        TDEC U2, INC U3
DIR5;     Z;      DIRXY;       INC U0, INC U1, NOP
DIR5;     NZ;     DIR6;        TDEC U2, INC U3
DIR6;     Z;      DIRY;        INC U1, NOP
DIR6;     NZ;     DIRY;        INC U0, INC U1, INC U3, NOP
DIRX;     ZZ;     RESETU2;     TDEC U3, INC U0
DIRY;     ZZ;     RESETU2;     TDEC U3, INC U1
DIRXY;    ZZ;     RESETU2;     TDEC U3, INC U0, INC U1

# Restore U2 from value in temporary register U3, and then draw
# the pixel (boat) at the current bit.
RESETU2;  Z;      DRAWX1;      TDEC U0, SET B2D
RESETU2;  NZ;     RESETU2;     TDEC U3, INC U2

# Update the B2D read head location based on U0 and U1.
DRAWX1;   Z;      DRAWY1;      TDEC U1
DRAWX1;   NZ;     DRAWX2;      TDEC U0
DRAWX2;   Z;      DRAWY1;      TDEC U1, INC B2DX
DRAWX2;   NZ;     DRAWX3;      TDEC B2DX
DRAWX3;   *;      DRAWY1;      TDEC U1
DRAWY1;   Z;      RESETB0;     TDEC B0
DRAWY1;   NZ;     DRAWY2;      TDEC U1
DRAWY2;   Z;      RESETB0;     TDEC B0, INC B2DY
DRAWY2;   NZ;     DRAWY3;      TDEC B2DY
DRAWY3;   *;      RESETB0;     TDEC B0

# Move B0 read head to least significant bit.
RESETB0;  Z;      INCB0A;      READ B0
RESETB0;  NZ;     RESETB0;     TDEC B0

# Add 1 to B0. If we introduce a new "2" in its base-4
# representation, go to INCU2A, otherwise add 1 to U2.
INCB0A;   Z;      INCB0B;      SET B0, NOP
INCB0A;   NZ;     INCB0E;      INC B0, NOP
INCB0B;   ZZ;     INCB0C;      INC B0, NOP
INCB0C;   ZZ;     INCB0D;      READ B0
INCB0D;   Z;      MOD8A;       TDEC U5, INC U2
INCB0D;   NZ;     MOD8A;       TDEC U5, INC U2, SET B0
INCB0E;   ZZ;     INCB0F;      READ B0
INCB0F;   Z;      INCU2A;      TDEC U4, SET B0
INCB0F;   NZ;     RESETB0;     INC B0, NOP

# Add U4 (= 6) to U2 with the help of U0, without clearing U4.
INCU2A;   Z;      INCU2B;      TDEC U0
INCU2A;   NZ;     INCU2A;      TDEC U4, INC U0, INC U2
INCU2B;   Z;      MOD8A;       TDEC U5
INCU2B;   NZ;     INCU2B;      TDEC U0, INC U4

# Set U2 = U2 mod (U5 = 8), with the help of U0 and U1.
MOD8A;    Z;      RESET3;      TDEC U1
MOD8A;    NZ;     MOD8B;       TDEC U2, INC U0
MOD8B;    Z;      RESET1;      TDEC U0
MOD8B;    NZ;     MOD8A;       TDEC U5, INC U1

# Reset registers and restart.
RESET1;   *;      RESET2;      TDEC U0
RESET2;   Z;      RESET3;      TDEC U1, INC U5
RESET2;   NZ;     RESET2;      TDEC U0, INC U2
RESET3;   Z;      RESET4;      TDEC U0
RESET3;   NZ;     RESET3;      TDEC U1, INC U5
RESET4;   Z;      DIR0;        TDEC U2
RESET4;   NZ;     RESET4;      TDEC U0
```

The basic idea behind this slowly growing pattern is the same as it was for the binary ruler from Section 9.4.1, which had diameter $\Theta(\log(t))$. We count in binary, except instead of placing the bits of the resulting number in a single 1D row via a binary register, we arrange them in a 2D triangular array via a 2D printer. In particular, we place the least significant bit of the number in one row, the next 2 least significant bits in the next row, the next 3 least significant bits in the next row, and so on. The APGsembly code that performs this task is presented in APGsembly 9.10.

This APGsembly code is quite a bit longer, but not much more complicated, than APGsembly 9.5 for the binary ruler. The only extra complexity in this code is that we need some unary registers to help us keep track of how many bits in total we should print in the current row, and how many bits are left before we reach the end of the current row. In particular, it makes use of three sliding block registers as follows:

`U0`: the number of bits left to print before reaching the end of the current row
`U1`: the total number of bits to print in the current row
`U2`: a helper register used to copy `U1` into `U0` when we start a new row

The pattern that results from compiling this APGsembly code is displayed in Figure 9.17. To give an idea of how slowly the diameter of this pattern grows, note in the 7.5×10^{11} generations after printing its first bit (i.e., boat), its computer goes through roughly 44 700 clock ticks and the pattern's bounding box increases in size by only 64 cells in one direction and 96 cells in the other.[43] We could slow down this pattern even more by decreasing the speed of its clock gun, but that would not change the asymptotic growth rate of its diameter.

[43] The boats that the 2d printer prints are offset from each other by 16 full diagonals.

Free PDF. See conwaylife.com/book for pattern files and print version.

APGsembly 9.10 APGsembly code for a pattern whose diameter (i.e., longest bounding box side length) in generation t is $\Theta(\sqrt{\log(t)})$—the smallest unbounded diametric growth rate possible.

```
#COMPONENTS B2D,NOP,U0-2
#REGISTERS {}
# State    Input    Next state    Actions
# ---------------------------------------
INITIAL;   ZZ;      CHECK;        READ B2D

## Determine whether the current bit equals 0 or 1.
# If it equals 0, set it to 1 and go to the least significant bit.
# If it equals 1, set it to 0 and go to the next most significant bit.
CHECK;     Z;       LSB1;         SET B2D, NOP
CHECK;     NZ;      NSB1;         TDEC U0

## The LSB states move the B2D read head to (0,0) and set U0 = U1 = 0.
LSB1;      ZZ;      LSB2;         TDEC B2DX
LSB2;      Z;       LSB3;         TDEC B2DY
LSB2;      NZ;      LSB2;         TDEC B2DX
LSB3;      Z;       LSB4;         TDEC U0
LSB3;      NZ;      LSB3;         TDEC B2DY
LSB4;      Z;       LSB5;         TDEC U1
LSB4;      NZ;      LSB4;         TDEC U0
LSB5;      Z;       CHECK;        READ B2D
LSB5;      NZ;      LSB5;         TDEC U1

## The NSB states move the B2D read head to the next most significant bit.
# If U0 = 0 then we are at the end of the current X row, so start the next one.
# If U0 > 0 then go to the next position in this X row and read the bit.
NSB1;      Z;       NSB3;         TDEC B2DX
NSB1;      NZ;      NSB2;         INC B2DX, NOP
NSB2;      ZZ;      CHECK;        READ B2D

# When going to the next X row, increase U1 (the length of the current X row).
NSB3;      Z;       NSB4;         INC B2DY, INC U1, NOP
NSB3;      NZ;      NSB3;         TDEC B2DX

# Copy U1 into U0, with the help of U2. Then read the current bit.
NSB4;      ZZ;      NSB5;         TDEC U1
NSB5;      Z;       NSB6;         TDEC U2
NSB5;      NZ;      NSB5;         TDEC U1, INC U2
NSB6;      Z;       CHECK;        READ B2D
NSB6;      NZ;      NSB6;         TDEC U2, INC U0, INC U1
```

9.8 Notes and Historical Remarks

Conway's and Gosper's groups showed in the early 1970s that it was theoretically possible to assemble a Life pattern to complete any computational task that a standard programmable computer could accomplish [Wai74, BCG82]. However, it wasn't until almost three decades later that Life technology advanced far enough to allow for universal computer patterns that were small enough to be simulated on a desktop computer.

Paul Rendell developed the first such device in April 2000 [Ren16]—a Turing machine based on period 30 circuitry (stable circuitry was not nearly as well-developed at the time). He then extended

Free PDF. See conwaylife.com/book for pattern files and print version.

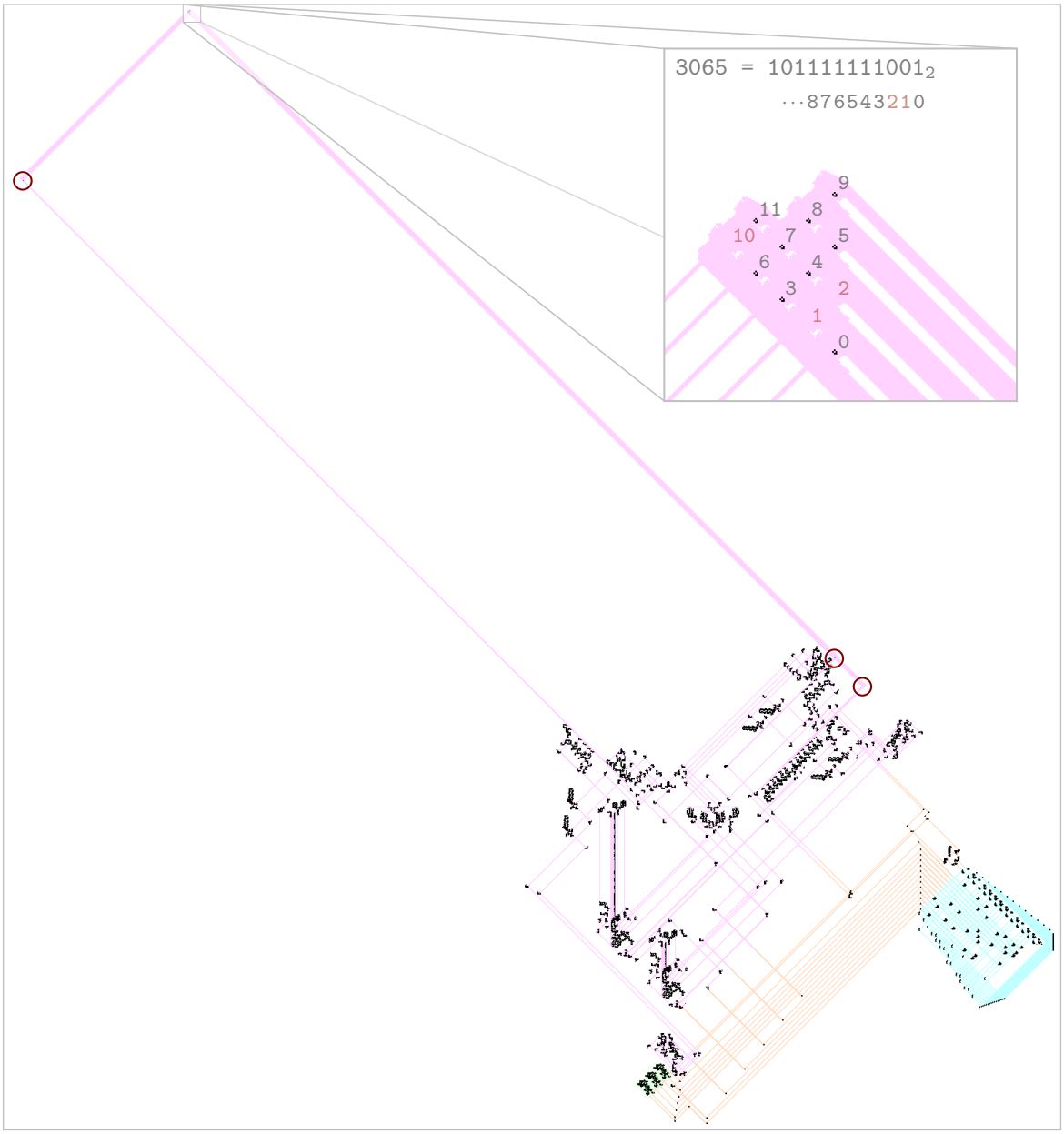


Figure 9.17: A pattern with minimal asymptotic diametric growth rate $\Theta(\sqrt{\log(t)})$, compiled from APGsembly 9.10. The computer at the bottom-right, highlighted in aqua, feeds into three sliding block registers (highlighted in green) and a 2D printer (highlighted in magenta). The 2D printer makes use of three sliding blocks (circled in red) to create and read a 2D array of boats at the top that can in principle extend arbitrarily far to the northwest and northeast. This particular computer counts in binary and instructs the 2D printer to arrange the bits of the current number as boats in a triangle (which gets larger as the number of bits increases). After 7.5×10^{11} generations, it has counted to $3065 = 101111111001_2$.

this to a *universal* Turing machine (i.e., a device capable of emulating any Turing machine) in February 2010. While these devices are computationally universal, and their input and output are reasonably straightforward to interpret, Turing machines themselves are rather abstract. For example, programming this device to compute (let alone display) the digits of π would be a monumental task.

Paul Chapman constructed the next universal computer in 2002—a register machine that, at a

Free PDF. See conwaylife.com/book for pattern files and print version.

high level, functions much like the universal computation toolkit that we introduced in this chapter.[44] Indeed, it made use of sliding blocks to build unary registers (just like we did), and switched between different states by virtue of actions to those unary registers returning zero or non-zero values. However, it used lightweight spaceships to transmit information instead of gliders, it made use of period 30 circuitry instead of stable circuitry, and it did not have any of the other components (e.g., binary registers or character printers) that we introduced.

The next universal computation toolkit to be developed was the one that we explored in this chapter, which was completed by Adam P. Goucher in 2010 [Gou10]. Finally, Nicolas Loizeau created an 8-bit programmable computer in 2016.[45] While it is less versatile than the toolkit from this chapter (e.g., it cannot compute quantities larger than $2^8 - 1 = 255$), it is significantly simpler to program. For example, just 8 relatively easy-to-understand lines of code suffice to program it to compute the sequence of Fibonacci numbers (up to the term 233, which is the largest one it can handle), whereas a few dozen lines of somewhat more complicated APGsembly code would be required to compute that sequence (see Exercise 9.26).

The original π calculator that Adam P. Goucher built in February 2010 had a bounding box that was roughly 12 times as large as the one that we displayed in Figure 9.15, since many of the small components that we used (like Snarks and syringes) were not available at that time. However, the actual functionality of our calculator is almost identical to that of the original one, as is the speed at which it runs.

About one week prior to building his π calculator, Goucher built a similar calculator for the mathematical constant $\varphi = (1+\sqrt{5})/2 = 1.61803\ldots$. While the techniques of Section 9.6.1 are general enough to build a φ calculator, the original one used a much simpler algorithm that caused it to run significantly faster—it could compute the first n digits of φ in $\Theta(n^3)$ generations, whereas the π calculator requires $\Theta(n^6)$ generations for the same task.

Exercises

solutions to starred exercises on page 464

9.1 [2/5] Add some additional lines of APGsembly code to APGsembly 9.1 so as to make it have 3 possible outputs instead of just 2: one corresponding to `U0 < U1`, one to `U1 < U0`, and one to `U0 = U1`.

9.2 [2/5] Modify APGsembly 9.1, with the help of additional registers, so that the `U0` and `U1` registers return to their original values at the end of the program. How many additional registers do you need to accomplish this task?

9.3 [2/5] Modify APGsembly 9.3 for multiplying two sliding block registers so as to complete the same computation without erasing the value of `U0`.
[Hint: Add another temporary register.]

9.4 [3/5] In this exercise, we implement the division-by-subtraction algorithm that was described at the end of Section 9.3.

- **(a)** Write pseudocode (similar in style to the pseudocode on the left of APGsembly 9.3) that computes and stores the integer part of `U0 / U1` in `U2` and the remainder of that division in `U3`. You may use as many temporary registers as you like.
- **(b)** Write APGsembly code that implements your pseudocode from part (a).
- **(c)** What happens in your code if `U1 = 0` and you try to divide by it? Modify your code, if necessary, so that it halts and provides some sort of error code in this case (e.g., maybe it could set some designated "error register" `U4` to 1).
- **(d)** Use the APGsembly compiler that is linked from `conwaylife.com/wiki/APGsembly` to compile a Life pattern that implements your APGsembly code from part (b).

[44] See `www.igblan.free-online.co.uk/igblan/ca/` for pattern files and a description of this machine.
[45] See `conwaylife.com/wiki/8-bit_programmable_computer` for download links and usage instructions.

Free PDF. See conwaylife.com/book for pattern files and print version.

9.5 [2/5] Describe how the `READ` and `SET` actions of the binary register from Figure 9.7 are implemented. For example, what glider salvos implement these operations, and how are those salvos constructed?

9.6 [1/5] Write APGsembly code that stores the value 241 in a binary register `B0` (using `INC` and `SET` actions as in APGsembly 9.4, not via the `#REGISTERS` header).

9.7 [3/5] Suppose that `U0` and `B0` are a sliding block and binary register, respectively, storing some values. Write APGsembly code that computes `U0 * B0` and stores it in `B0`.

9.8 In this exercise, suppose that `U0` is a sliding block register whose value has already been set.

- **(a)** [2/5] Write APGsembly code that stores the value $2^{\texttt{U0}}$ in the binary register `B0`.
- **(b)** [3/5] Write APGsembly code that stores the value $10^{\texttt{U0}}$ in the binary register `B0`.
- **(c)** [4/5] Write APGsembly code that stores the value $3^{\texttt{U0}}$ in the binary register `B0`.

9.9 [4/5] Write APGsembly code that determines which of two binary registers `B0` and `B1` is storing a larger value (i.e., code that outputs different results depending on whether `B0 <= B1` or `B1 < B0`, analogous to APGsembly 9.1 for unary registers).

***9.10** [3/5] The binary ruler displayed in Figure 9.8 has a bounding box that is roughly $3\,600 \times 3\,100$ cells. At approximately what generation will the size of its bounding box first exceed $10\,000 \times 10\,000$? [Hint: Do not try to run it long enough to find out—do a calculation instead.]

9.11 [3/5] The extremely slowly growing pattern displayed in Figure 9.17 has a bounding box that is roughly $24\,300 \times 26\,500$ cells. At approximately what generation will the size of its bounding box first exceed $30\,000 \times 30\,000$?

9.12 [2/5] Label the northeast–southwest lines in the computer portion of the binary ruler from Figure 9.8 (like we labelled those lines in the multiplication computer of Figure 9.6). Which substate do each of these lines correspond to in APGsembly 9.5?

9.13 [5/5] The `ADD`, `SUB`, and `MUL` components of Figures 9.9–9.11 were designed over a decade ago, and could be made much smaller via modern components like Snarks and syringes.

- **(a)** Rebuild the `ADD` component, without sacrificing or changing any of its high-level functionality, so that it fits within an 800×800 bounding box.
- **(b)** Rebuild the `SUB` component so that it fits within an 800×800 bounding box.
- **(c)** Rebuild the `MUL` component so that it fits within a $1\,500 \times 1\,500$ bounding box.

9.14 [2/5] In the green pixel-printing portion of the row printer from Figure 9.13, there are only 6 Herschel edge-shooters, yet they produce all 7 of the pixel-printing gliders from Figure 9.12(a). Explain where the extra glider comes from.

***9.15** [2/5] In the π calculator of Figure 9.15, one of the computer's merge circuits (i.e., transparent reflectors) is significantly farther west than any of the others. Why? Which substate (i.e., line of APGsembly code from Appendix B.8) does this correspond to?

***9.16** [4/5] In this exercise, we explore where the series (9.1) for π comes from.

- **(a)** Explain why $\pi = 4 - 4/3 + 4/5 - 4/7 + 4/9 - \cdots$. [Hint: We did this already in Section 6.6.]
- **(b)** Manipulate the series from part (a) to show that
$$\pi = 2 + \left(\frac{4}{1 \cdot 3} - \frac{4}{3 \cdot 5} + \frac{4}{5 \cdot 7} - \frac{4}{7 \cdot 9} + \frac{4}{9 \cdot 11} - \cdots \right).$$
[Hint: Write each term $4/(2k+1)$ as $2/(2k+1) + 2/(2k+1)$ and regroup parentheses.]
- **(c)** Use the same method on the parenthesized terms from part (b) to rewrite that series as
$$\pi = 2 + \frac{2}{3} + \left(\frac{8}{1 \cdot 3 \cdot 5} - \frac{8}{3 \cdot 5 \cdot 7} + \frac{8}{5 \cdot 7 \cdot 9} - \cdots \right).$$
- **(d)** Repeat this method over and over again to rewrite this series as
$$\pi = 2 \left(1 + \frac{1!}{3} + \frac{2!}{3 \cdot 5} + \frac{3!}{3 \cdot 5 \cdot 7} + \frac{4!}{3 \cdot 5 \cdot 7 \cdot 9} + \cdots \right).$$
- **(e)** Finally, repeatedly factor the series from part (d) to obtain the series (9.1).
[Side note: This method of converting an alternating series into one with non-negative terms is called the **Euler transform**.]

Free PDF. See conwaylife.com/book for pattern files and print version.

***9.17** [5/5] We claimed in the text that if the matrix A_k is as defined in Equation (9.3), $B_n = A_0A_1A_2\cdots A_n$, q_n is the top-right entry of B_n, and r_n is the bottom-right entry of B_n, then

$$\frac{q_n}{r_n} = 2\left(1+\frac{1}{3}\left(1+\frac{2}{5}\left(1+\frac{3}{7}\left(\cdots\left(1+\frac{n-1}{2n-1}\right)\right)\right)\right)\right),$$

so the entries of B_n can be used to approximate π. Prove this formula. [Hint: You can find fairly explicit formulas for the entries of B_n.]

***9.18** [3/5] Modify just 2 or 3 lines of the APGsembly code from Appendix B.8 to turn the π calculator into an e calculator.
[Hint: We described the changes that need to be made in Section 9.6.1.]

9.19 In this exercise, we use the following series to tweak the π calculator to instead compute

$$\sqrt{2} = \sum_{k=0}^{\infty}\frac{(2k+1)!}{2^{3k+1}(k!)^2} = \frac{1}{2}+\frac{3}{8}+\frac{15}{64}+\frac{35}{256}+\frac{315}{4096}+\cdots.$$

- **(a)** [2/5] The above series representation of $\sqrt{2}$ can be put in the general form (9.4). What are the values of a, b, p, q, r, and s?
- **(b)** [3/5] Change just a few lines of the APGsembly for the π calculator (from Appendix B.8) to account for these different values of a, b, p, q, r, and s.
  [Hint: Be careful to allocate enough memory to the various registers—the amount used by the π calculator is not quite enough.]

9.20 Recall APGsembly 9.9 for printing the Koch snowflake fractal.

- **(a)** [2/5] Use a compiler that is linked from `conwaylife.com/wiki/APGsembly` to compile this APGsembly code into a Life pattern.
- **(b)** [3/5] The `B2D` component of this pattern can print pixels much closer to the rest of the circuitry than the `B2D` component in the extremely slowly growing pattern of Figure 9.17. Explain why.

9.21 [4/5] Write APGsembly code that determines how many decimal digits the integer stored in a sliding block register has.

9.22 [5/5] We saw APGsembly code for printing the contents of a sliding block register, as long as that register contains a value no larger than 9, in APGsembly 9.8. Now write APGsembly code that prints the contents of a sliding block register, regardless of how many decimal digits it has.
[Hint: Make use of the code from Exercise 9.21]

9.23 [4/5] The e calculator from Exercise 9.18 can be sped us so as to compute digits considerably quicker, since the series (9.5) for e is so much simpler and converges so much quicker than the series (9.1) for π.

- **(a)** In the π series (9.1), each term is roughly half as large as the term that came before it, so we need to iterate on average $\log_2(10) \approx 3.3219$ times per decimal place of π. How many times do we need to iterate per decimal place of e?
- **(b)** In the e calculator, you do not actually need the `B0` register that was used in the π calculator. Why not?
- **(c)** In the e calculator, the binary registers `B1` and `B2` do not require as many bits of memory as in the π calculator (i.e., `U6` does not need to be as large). Why? Roughly how many bits of memory are needed after n iterations?
- **(d)** The clock period of the e calculator can be decreased from 2^{20} generations to 2^{18} generations without introducing any errors. Create a period 2^{18} universal regulator that can replace the e calculator's period 2^{20} universal regulator.[46]
- **(e)** Use the simplifications and speed-ups suggested by parts (a–d) to compile an e calculator that prints `2.718` before generation 2×10^{10} (instead of around generation 10^{12}, like the e calculator from Exercise 9.18).

9.24 [4/5] Write APGsembly code for a Life pattern that prints all positive integers, one after another, separated by dots. That is, its output should begin

```
1.2.3.4.5.6.7.8.9.10.11.12.13.14.15.
```

[Hint: Be careful with integers that have 2 or more digits. Make use of the code from Exercise 9.22.]

9.25 [5/5] Write APGsembly code for a Life pattern that prints the prime numbers, separated by dots. That is, its output should begin

```
2.3.5.7.11.13.17.19.23.29.31.37.41.
```

[Hint: Modify the code from Exercise 9.24 so that an integer n is not printed if the integer division n/k has a remainder of 0 for some $2 \le k \le n-1$.]

9.26 [4/5] Write APGsembly code for a Life pattern that prints the Fibonacci numbers, separated by dots. That is, its output should begin

```
0.1.1.2.3.5.8.13.21.34.55.89.144.
```

with each integer being the sum of the previous two.

[46] Even though decreasing the clock period decreases the number of Life generations needed to perform a calculation, it typically does not decrease the real-world time that it takes Life software like Golly to evolve the pattern to the point of having performed that calculation. The reason for this is that the HashLife algorithm used by Golly is able to quickly skip over "wasted" generations where the clock is just waiting, since so little of the Life plane changes during those generations.

Free PDF. See conwaylife.com/book for pattern files and print version.



10. Self-Supporting Spaceships

> Figuring out how to achieve a particular behavior makes a pleasant puzzle; actually building the thing is mostly tedious.
>
> Dean Hickerson

Universal computation, covered in the previous chapter, is the first of two major types of universality that often show up in discussions of Conway's Game of Life, and cellular automata in general. The second type of universality is universal construction. Whereas universal computation is the ability to "compute anything that can be computed", universal construction could be loosely summarized as the ability to "construct anything that can be constructed".

We will cover this topic in much more depth in Chapter 11, but this chapter provides a good preview of the general idea. Indeed, in this chapter we will build spaceships that work by reaching out into a region of empty space in front of themselves and constructing something there. More specifically, self-supporting spaceships (the topic of this chapter) work by manipulating a reaction that moves along a track so as to construct the track in front of itself.

10.1 The Silverfish

Most self-supporting spaceships rely on a core reaction that moves an object forward with the help of another object that is in its path. One such reaction (simply called the **31c/240 reaction**) is displayed in Figure 10.1, in which a Herschel collides with a block in such a way that it moves forward by 31 cells over the course of 240 generations.[1] At the same time, the block is moved back by 22 cells, a second block is created, and two gliders are released.

The first major goal of this chapter is to use this reaction to create a $31c/240$ spaceship, which we call the **silverfish**. The major obstacles that we will have to overcome are cleaning up the extra

[1] We saw another such reaction back in Figure 4.55. We explore the consequences of this reaction a bit later, in Section 10.2, since building a spaceship out of it is somewhat more technical than the reaction that we use here.

Free PDF. See conwaylife.com/book for pattern files and print versio

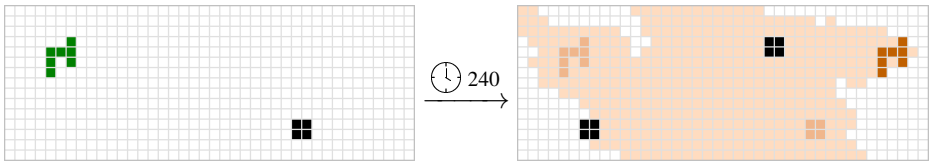


Figure 10.1: The **31c/240 reaction**. A Herschel collides with a block so as to move forward by 31 cells (and move the block back by 22 cells) over the course of 240 generations. This reaction also produces a second block (at the top-center) and two gliders as by-products.

objects (blocks and gliders) left behind by the Herschel as it moves, and using the Herschel to create a block in front of itself before it gets there.

10.1.1 Herschel Crawlers and Rakes

If we place blocks in a straight line with a spacing of 31 cells, a Herschel is able to crawl along them (via the $31c/240$ reaction of Figure 10.1) at a speed of $31c/240$. Furthermore, if we place two of these tracks next to each other, we can use one of the output gliders from one of the Herschels to cleanly erase the extra block that is produced by the other Herschel, as illustrated in Figure 10.2. When we do this, we get a track for two Herschels called a **reburnable wick**: a wick that is not used up (but is potentially repositioned and/or rephased) after its fuse (the pair of Herschels) burns through it.

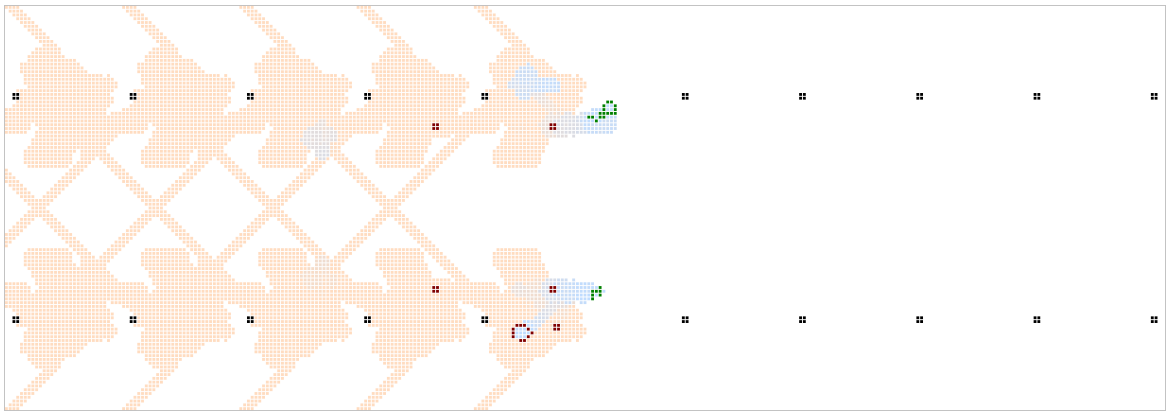


Figure 10.2: A reburnable block wick, along which a pair of Herschels (highlighted in green) can move at a speed of $31c/240$. The spark and blocks highlighted in red are only temporary—they are destroyed as the Herschels move farther down the track.

While a single Herschel pair changes the positions of the blocks in the reburnable wick, a sequence of 31 Herschel pairs would leave them all exactly where they would be if no Herschels burned through them at all.[2] This gives us an infinitely long pattern that moves at a speed of $31c/240$. We now focus on converting this pattern into a *finite* configuration (i.e., a spaceship) that moves at the same speed.

To this end, we need a way of constructing the blocks in front of the Herschel pairs that burn through them. Fortunately, blocks are fairly easy to construct—we can collide spaceships like gliders together so as to synthesize them. Furthermore, the Herschel pairs create gliders as they move, so we "just" need to redirect those gliders in front of the Herschels so as to synthesize the blocks.

Unfortunately, getting gliders (or any other spaceships) in front of the Herschel pair is quite tricky, as it does not seem like it should be possible to reflect the gliders that the Herschel pair emits without

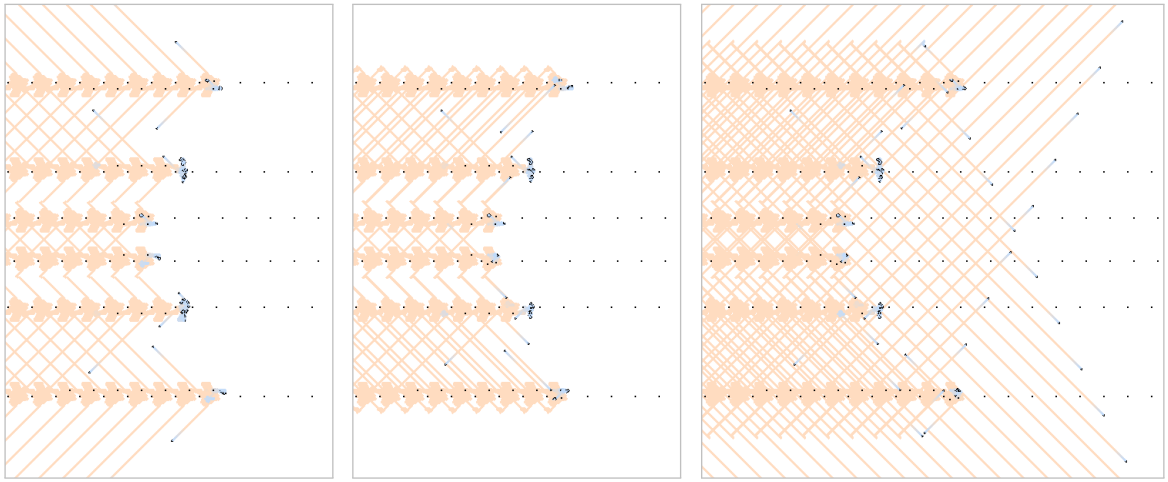
[2] After the 31 Herschel pairs burned through them, each block would actually be moved back $22 \times 31 = 682$ cells. However, since the wick is infinitely long and the blocks are spaced 31 cells apart, this makes no difference.

Free PDF. See conwaylife.com/book for pattern files and print version.

making use of stationary components like Snarks or other small spaceships (none of which travel at $31c/240$). One technique that works to at least let us send gliders forward (instead of backward, as in Figure 10.2) is to use the two-glider kickback reaction from Table 5.1. Unfortunately, to make use of this reaction, we have to place even more of these tracks next to each other. One configuration that works is displayed in Figure 10.3(c)—it makes use of six Herschels and six block tracks instead of just two.

We call this pattern a **forward rake** even though it is not *technically* a rake due to its reliance on the supporting block tracks. Since firing gliders backward via these Herschels is so straightforward (they fire gliders backward all on their own, after all), it is not difficult to construct the **backward rake** in Figure 10.3(a) that works on this same set of six block tracks (instead of the one that works on two block tracks, which we saw in Figure 10.2).

These rakes have one big limitation, however—we can only use them to place an output glider on every 31st lane. If we want to make sure that a glider is fired on a *particular* lane, we need a mechanism for moving the block track (and thus any subsequent rakes) slightly forward or backward. Fortunately, this is also straightforward—the **rephaser** displayed in Figure 10.3(b) moves the block tracks backward by 22 cells (or equivalently, forward by 9 cells) simply by having a Herschel burn through each track in such a way that their output gliders annihilate one another. Importantly, since $\gcd(22,31) = 1$, we can use multiple copies of this rephaser so as to make subsequent rakes output gliders on any lanes of our choosing.



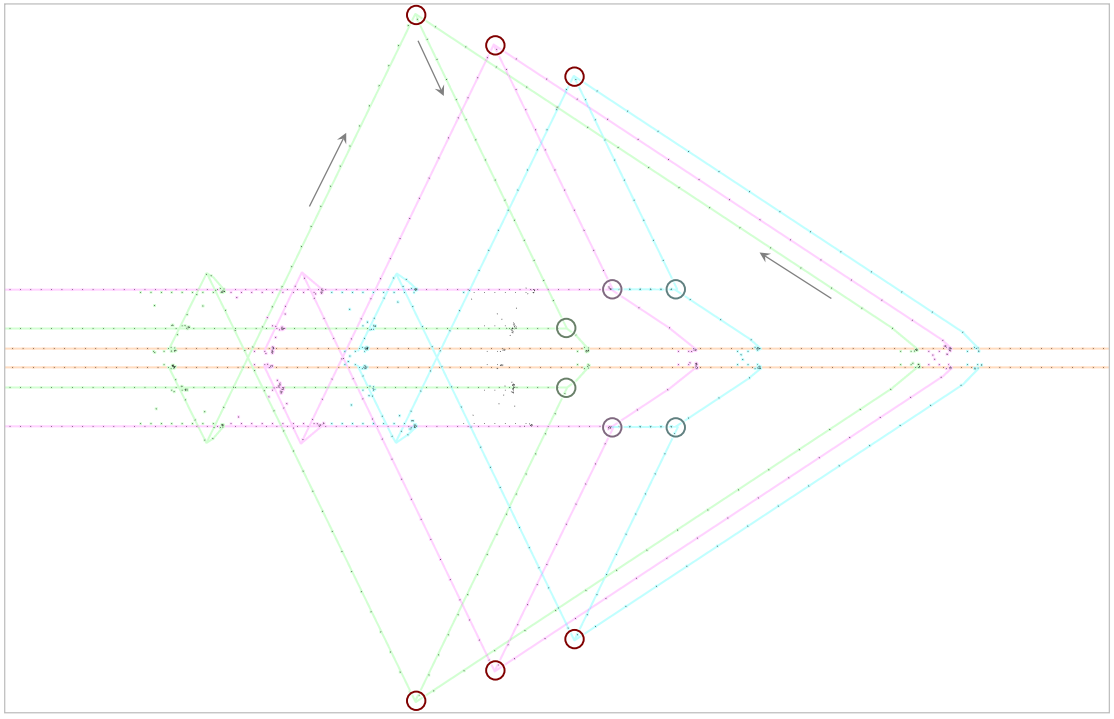
(a) A backward rake. **(b)** A rephaser. **(c)** A forward rake.

Figure 10.3: Rakes and rephasers that crawl along 6 reburnable block wicks at a speed of $31c/240$. In each case, the 6 Herschels release 12 gliders every 240 generations, and 6 of those gliders are used to destroy the excess blocks left behind by each Herschel. In (a) the backward rake, 4 of the gliders collide so as to destroy each other and the other 2 are released backward. In (b) the rephaser, 2 gliders are destroyed in a kickback reaction and then the other 4 are destroyed by colliding into each other. In (c) the forward rake, 4 kickback reactions are used to destroy 4 of the gliders, leaving the remaining 2 gliders to escape to the front.

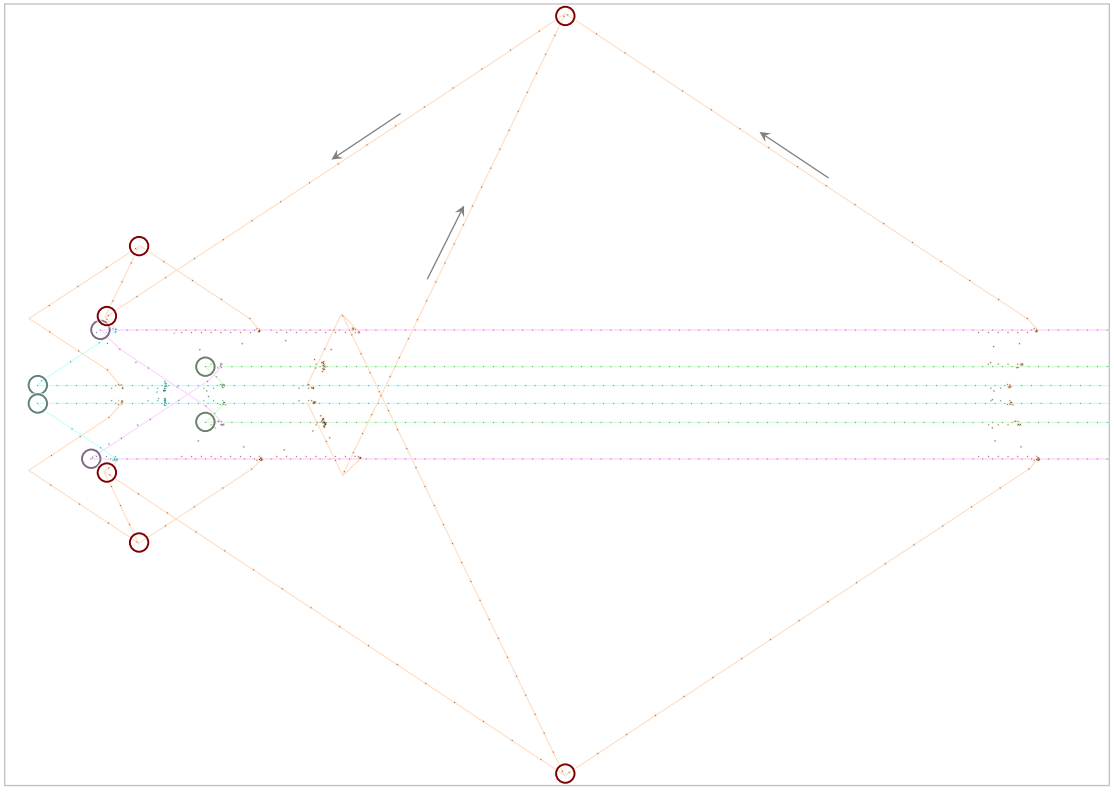
10.1.2 Synthesizing and Destroying Block Tracks

Fortunately, even though we now have 6 block tracks, the problem of constructing the blocks in front of the Herschels is not much trickier than it was when we just had 2 block tracks. Indeed, we can construct 4 of these block tracks simply via these rakes and some additional kickback reactions, as illustrated in Figure 10.4(a).

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) A configuration of rakes that turns a 2-block track (highlighted in orange) into a 6-block track.



(b) A configuration of rakes that destroys the 6-block track. Destroying the tracks is actually straightforward, but multiple kickback reactions must be used to clean up the leftover gliders (highlighted in orange).

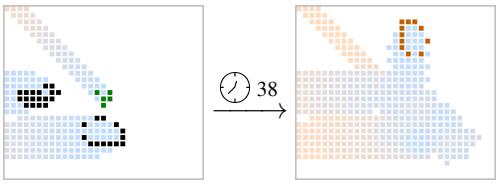
Figure 10.4: A configuration of rakes that uses multiple kickback reactions (circled in red) to position gliders so that they can either (a) synthesize block tracks ahead of the rakes, or (b) destroy the block tracks behind the rakes. The locations where gliders synthesize or destroy the block tracks are circled near the center.

Free PDF. See conwaylife.com/book for pattern files and print version.

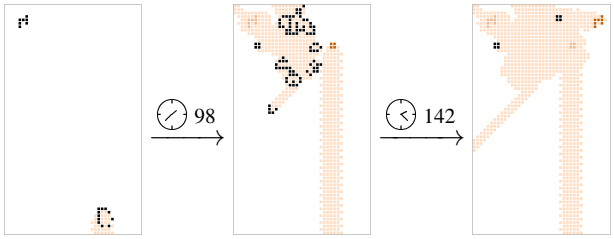
Lining up these kickback reactions and block syntheses is somewhat tricky due to the difficulty of controlling which lanes we produce gliders on when using these tracks. Fortunately, we have complete control of the *timing* of the gliders on these lanes. By delaying Herschels appropriately (and perhaps adding a single row of blocks to a track so as to change the color of the gliders that a Herschel produces) we can create any 2-glider collision that we like. With this technique we can clean up these six block tracks, just by using even more kickback reactions as illustrated in Figure 10.4(b).

However, we still do not have a method of constructing the two central block tracks at the front of Figure 10.4(a), since there is no way to arrange kickback reactions so as to put gliders in front of the frontmost rakes. While this may seem like an insurmountable problem at first, one solution is to synthesize light, middle, and/or heavyweight spaceships that travel in the same direction as the Herschels, and then bounce gliders off of those rows of xWSSes.

While this approach works, there is another method that results in the silverfish spaceship being significantly smaller. Instead of bouncing a glider back toward the front of the Herschels so as to synthesize the block track, we convert that glider into a middleweight spaceship via the reaction illustrated in Figure 10.5(a). The reason for this is that a middleweight spaceship can stabilize the front of the Herschel track in the exact same way as a block (in fact, one of the Herschel's sparks simply converts the MWSS into a block in the correct position), as illustrated in Figure 10.5(b).



(a) A way of colliding a glider with two heavyweight spaceships so as to make a perpendicular middleweight spaceship.



(b) An MWSS stabilizing a Herschel crawler.

Figure 10.5: Some reactions that can be used to stabilize the front of the $31c/240$ Herschel crawler, as long as we can somehow create a parallel double stream of heavyweight spaceships.

The last remaining question before we can piece together all of these reactions is how to create the pair of heavyweight spaceships that are used in Figure 10.5(a). The key insight that makes this possible is the fact that we can fire a glider at an HWSS so as to create some debris without affecting the HWSS (after all, heavyweight spaceships are nice and sparky), and then we can fire additional gliders at that debris so as to create another HWSS on the same lane.[3]

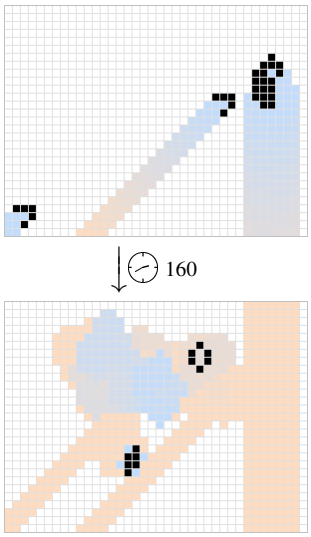
One reaction that implements the first half of this process, by creating a toad and a beehive a safe distance from the HWSS when a pair of gliders hits it (without disturbing the HWSS[4]), is displayed in Figure 10.6(a). An unfortunate feature of this reaction is that it relies on two synchronized gliders, and synchronizing the precise time and (especially) position of multiple gliders that are fired from these block tracks is quite tricky. Nevertheless, this synchronization is possible via kickback reactions, and one configuration of rakes that produces the exact glider pair that we need is displayed in Figure 10.6(b).

There are numerous different ways that we could fire gliders at that toad and beehive so as to turn them into a heavyweight spaceship on the correct lane. For the sake of simplicity, we use a slow salvo, since synchronizing the precise time and (especially) position of multiple gliders that are fired from these block tracks is quite tricky (as we saw for just two gliders in Figure 10.6(b)). We furthermore

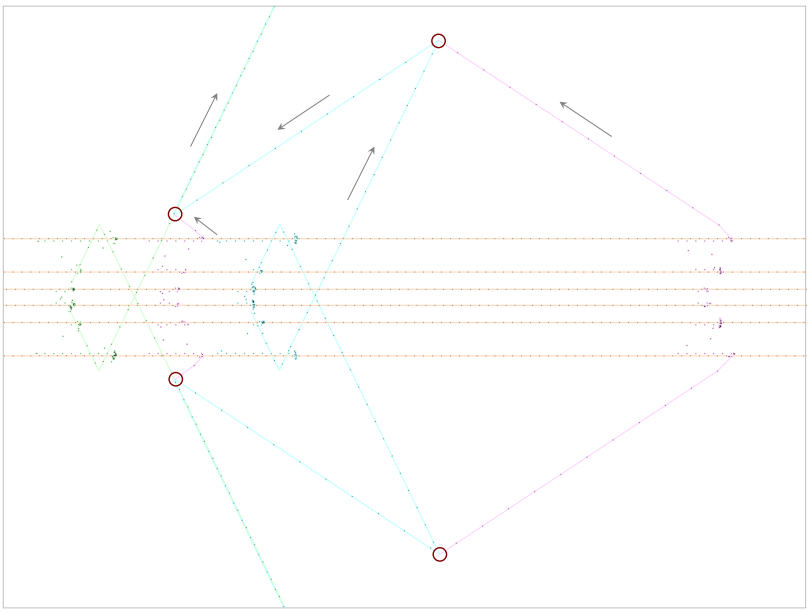
[3] In a sense, we are using a heavyweight spaceship to synthesize itself. It is this self-creation that is really what makes it possible to turn the $31c/240$ reaction into a spaceship.

[4] That is, this reaction is a Heisenburp.

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) A way of colliding two gliders with an HWSS so as to make a beehive and a toad away from the HWSS stream, and without affecting the HWSS.



(b) A configuration of block-crawling rakes that uses some kickback reactions (circled in red) to create the synchronized 2-glider paired needed in (a).

Figure 10.6: The configuration of rakes in (b) sends out a pair of gliders that, when they collide with a heavyweight spaceship as in (a), produce a beehive and a toad. That beehive and toad can then be used as a seed for a slow-salvo synthesis of that heavyweight spaceship.

just use gliders coming from one of the four possible directions, since (a) the gliders are fired from the 6-block track, which restricts us to just two directions (i.e., from forward and backward rakes on the block track), and (b) if we further restrict to gliders just coming from forward rakes then it makes the resulting construction of the $31c/240$ spaceship somewhat simpler and more uniform.[5] One particular slow salvo of this type that works is displayed in Figure 10.7.[6]

10.1.3 Glider Lanes and More Rakes

At this point, we have essentially everything that we need to construct a $31c/240$ spaceship. However, the spaceship that results from combining the various reactions and patterns that we have developed in this section ends up being monstrously large, so it is worthwhile to spend some time thinking about how we can reduce its size prior to assembling it.

The majority of this spaceship's length will come from the multi-step process needed to synthesize the heavyweight spaceship pair on each of its sides via the slow salvo of Figure 10.7.[7] Indeed, implementing this synthesis requires 13 gliders and thus 13 forward rakes. Furthermore, we need to use between 0 and 30 rephasers before each of those rakes to make sure the gliders that they emit are positioned on the correct mod 31 lane.

[5] Point (b) here is somewhat subjective—allowing gliders from backward rakes would decrease the number of gliders required, but perhaps increase the spacing needed between the rakes and make their placement somewhat more complicated.

[6] This recipe was produced by a long-running breadth-first search using a script that was written by ConwayLife.com forums user "oblique"—see conwaylife.com/forums/viewtopic.php?t=1296. It was one of the first options to show up that produced an upward HWSS with enough clearance that other HWSSes could safely travel past all the intermediate stages of the slow-salvo construction. It is not necessarily the smallest unidirectional salvo that exists, but it is probably fairly close.

[7] Similarly, the majority of its width comes from how close to the front of the spaceship we can put the first forward rake.

Free PDF. See conwaylife.com/book for pattern files and print version.

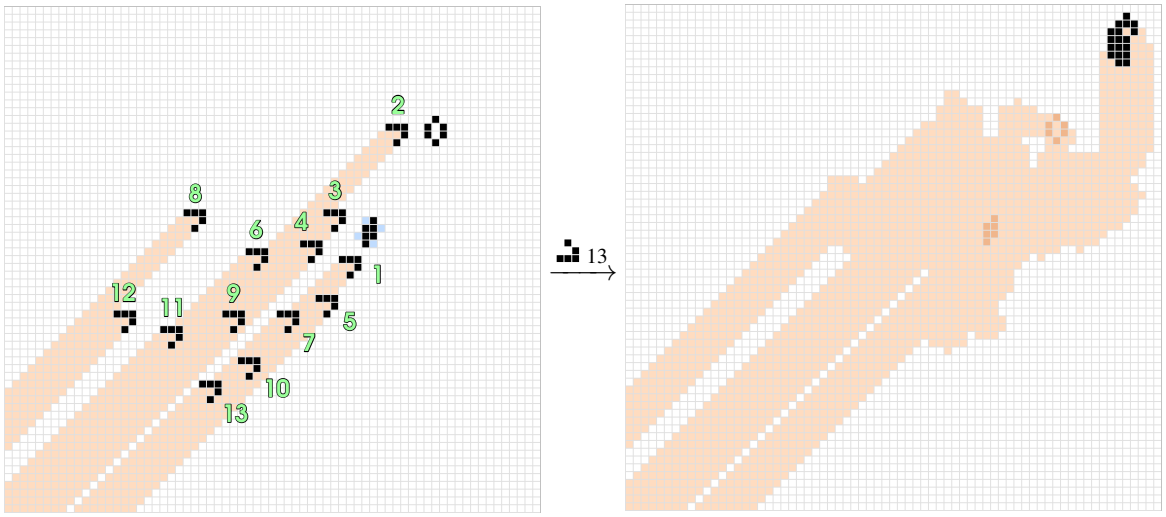


Figure 10.7: A 13-glider slow salvo that turns the configuration of a beehive and toad from Figure 10.6(a) back into an HWSS on the same lane as the HWSS that was used to create them.

In order to reduce the number of rephasers needed, and thus reduce the spaceship's length, we now introduce several other forward rakes that emit gliders on different lanes. These rakes, which are displayed in Figure 10.8,[8] work in one of three different ways:

- by using multiple copies of the two-glider kickback reaction (as in Figure 10.8(c)),[9]
- by using the two-glider synthesis of a traffic light and a glider from Table 5.1, and then using another glider or two to clean up the traffic light (as in Figure 10.8(e)), or
- by using two gliders to synthesize a small object that can be used to release a glider on another lane when it is destroyed (as in Figures 10.8(a) and (b)).

In order to help us keep track of where each of these rakes emits a glider, we name them according to what mod 31 lane they fire a glider on (relative to the original forward rake that we saw in Figure 10.3(c), which we say fires on lane 0) and the number of Herschels that are used on each block track. Specifically, their names have the form

R<number of Herschels>L<lane number>,

so that the original rake from Figure 10.3(c), for example, is called R1L0 (it uses a single Herschel on each block track and produces a glider on lane 0).[10]

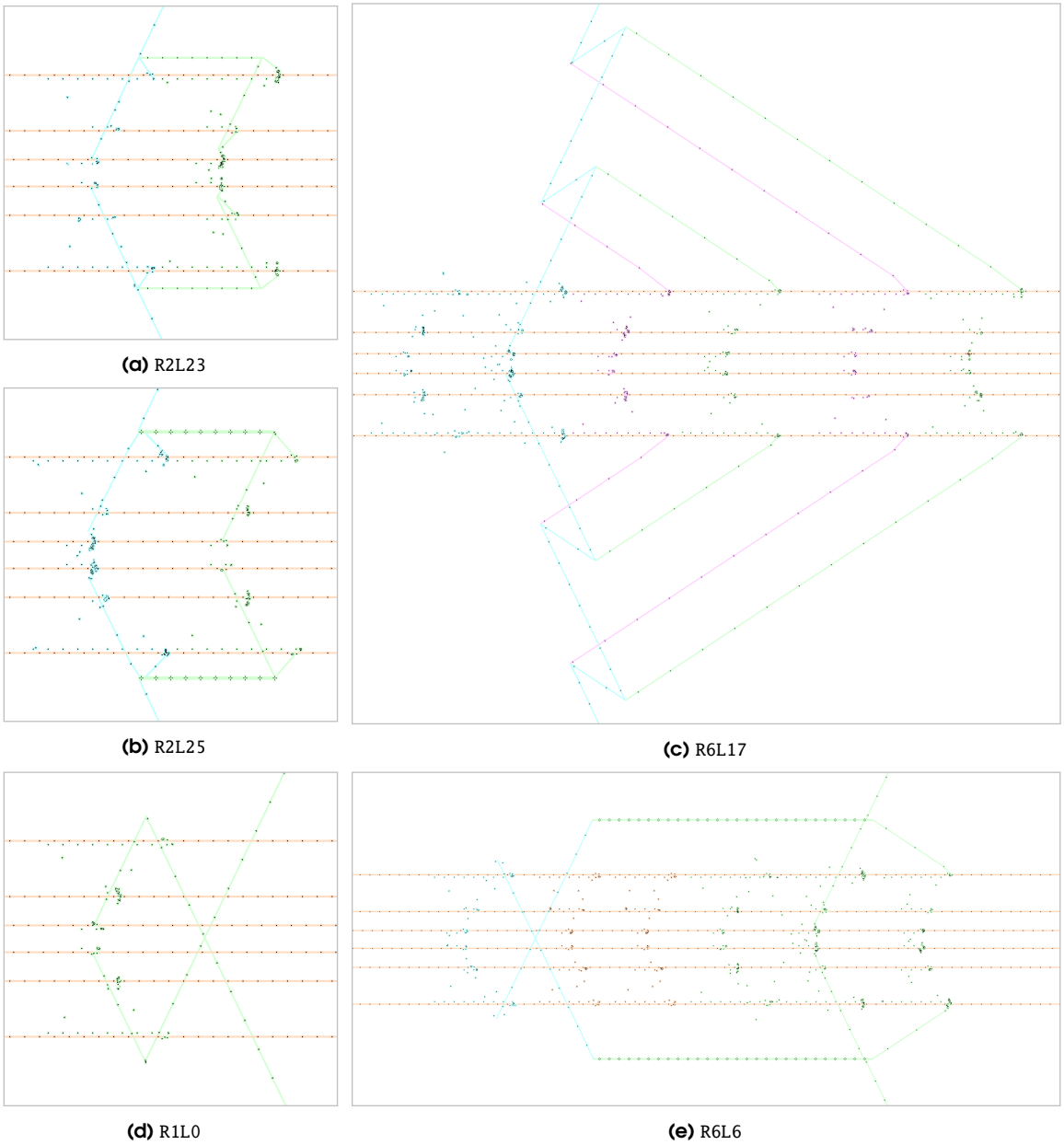
The "R" piece of this naming scheme is useful both as a measure of how long the rake is (e.g., it tells us that R6L17 is roughly three times as long as R2L25) and also for computing the mod 31 output glider lane of subsequent rakes. Indeed, a single rephaser (i.e., one Herschel on each block track) changes the output lane of subsequent gliders by -22 (or equivalently, by $+9$) mod 31, so a rake with name R<m>L<n> adjusts the output lane of subsequent gliders by $+9m$ (mod 31). For example, R6L21 increases the output lane of subsequent gliders by $9 \times 6 = 54 \equiv 23 \pmod{31}$.

[8] This collection of rakes is by no means exhaustive. See Exercises 10.2–10.6, and there are many others known as well (and probably quite a few others waiting to be found). The rakes given in Figure 10.8 are pretty good at hitting each mod 31 lane efficiently, though.

[9] Each pair of kickback reactions that we use in this way has the net effect of adding 2 rephaser lengths to the rake and increasing its output lane number by 16 (mod 31)—see Exercise 10.2.

[10] The "R" in this naming scheme stands for the fact that the rake has approximately the same length as this number of **r**ephasers, and the "L" stands for "**l**ane".

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) R2L23

(b) R2L25

(c) R6L17

(d) R1L0

(e) R6L6

Figure 10.8: Several forward rakes that crawl along a six-block track and emit gliders on different lanes mod 31. The forward rake from Figure 10.3(c) is the one displayed here at the bottom-left, called R1L0.

With these rakes in hand, we can fire gliders on particular mod 31 lanes much more efficiently than we could previously. For example, placing a glider on lane 22 (when the block track is currently aligned to lane 0) via just the rakes and rephaser from Figure 10.3 would require 30 rephasers followed by the forward rake R1L0. However, a much more efficient way of firing a glider on lane 22 is to use just 4 rephasers and then the forward rake R6L17 (for a total length of $4+6=10$ Herschels per track, instead of $30+1=31$ Herschels per track). A summary of the most efficient rake to use to place a glider on each mod 31 lane is provided in Table 10.1.

Free PDF. See conwaylife.com/book for pattern files and print version.

Lane	Cheapest Rake	Lane	Rake	Lane	Rake	Lane	Rake
0	R1L0	9	1 + R1L0	18	2 + R1L0	27	3 + R1L0
1	1 rephaser + R2L23	10	2 + R2L23	19	3 + R2L23	28	4 + R2L23
2	3 rephasers + R6L6	11	4 + R6L6	20	5 + R6L6	29	6 + R6L6
3	1 rephaser + R2L25	12	2 + R2L25	21	3 + R2L25	30	4 + R2L25
4	2 rephasers + R6L17	13	3 + R6L17	22	4 + R6L17		
5	4 rephasers + R1L0	14	5 + R1L0	23	R2L23		
6	R6L6	15	1 + R6L6	24	2 + R6L6		
7	7 rephasers + R6L6	16	8 + R6L6	25	R2L25		
8	5 rephasers + R2L25	17	R6L17	26	1 + R6L17		

Table 10.1: A summary of the shortest forward rakes that can be used to put a glider on a given lane (mod 31). For example, the shortest configuration of rephasers and forward rakes that can put a glider on lane 22 (when currently aligned to lane 0) consists of 4 rephasers followed by R6L17. Indeed, that configuration has a total length that is roughly the same as that of $6+4=10$ rephasers, and no other way of using rephasers and forward rakes is shorter.

10.1.4 Completing the Silverfish

We now proceed with construction of the silverfish. At the front of the spaceship we use the track-building component that we built in Figure 10.4(a) (with the understanding that we will still have to construct the two central block tracks somehow), and at the back of the spaceship we use the track-destroying component that we built in Figure 10.4(b). Right behind the track-building component, we place a forward rake that is aimed at a parallel stream of heavyweight spaceship pairs as in Figure 10.5(a), which create middleweight spaceships to stabilize the two central block tracks as in Figure 10.5(b).

The remainder of the silverfish is made up of forward rakes that synthesize the stream of heavyweight spaceship pairs, using the mechanisms of Figures 10.6 and 10.7. Indeed, after using the double gun from Figure 10.6(b) to create a beehive and toad, we want to create 13 gliders on the following sequence of lanes, as indicated by Figure 10.7:[11]

```
0, 11, 8, 7, 29, 13, 1, 19, 8, 0, 14, 22, 2.
```

We can place gliders on these lanes by repeatedly using Table 10.1 and keeping track of what lane the block tracks are currently synchronized to. After the two-glider rake from Figure 10.6(b), the block tracks are synchronized to lane 7. We then place 13 more rakes as follows:

- First, we want to fire a glider 24 (mod 31) lanes up from the current lane, on lane 0. Table 10.1 tells us that the most efficient way to do this is to use 2 rephasers and then R6L6. Since this places 8 Herschels on each track, it synchronizes the block tracks to lane $7+(9\times 8)=79\equiv 17$ (mod 31).
- Next, we want to fire a glider 6 lanes down (i.e., 25 lanes up) from here on lane 11 mod 31. Table 10.1 tells us that the most efficient way to do this is to use R2L25. Since this places 2 Herschels on each track, it synchronizes the tracks to lane $17+(9\times 2)=35\equiv 4$ (mod 31).
- Next, we want to fire a glider 4 lanes up from here on lane 8. Table 10.1 tells us that the most efficient way to do this is to use 2 rephasers followed by R6L17. Since this places 8 Herschels on each track, it synchronizes the tracks to lane $4+(9\times 8)=76\equiv 14$ (mod 31).

[11] We actually have some flexibility in which lanes we use here—see Exercise 10.7.

Free PDF. See conwaylife.com/book for pattern files and print version.

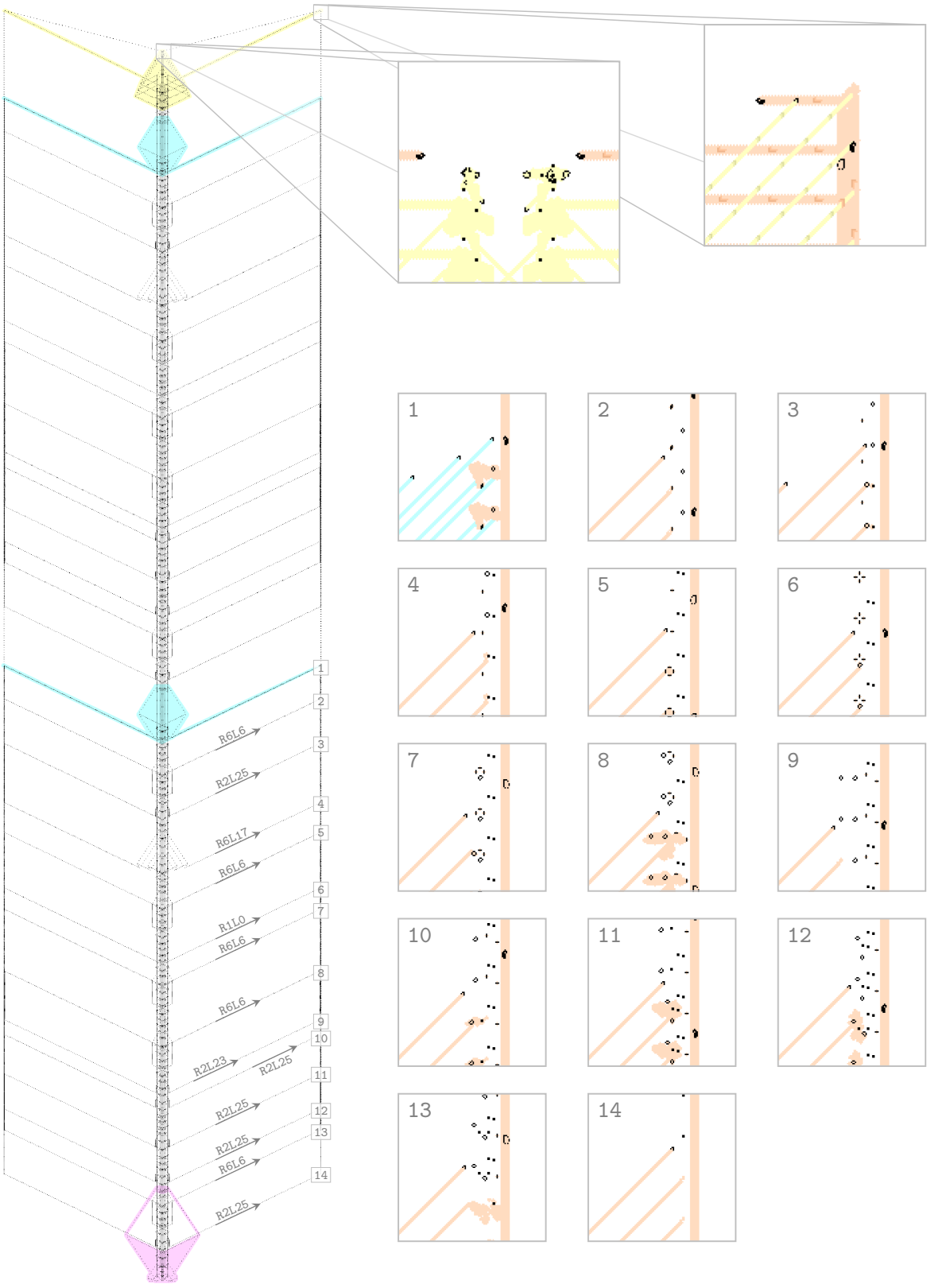


Figure 10.9: The **silverfish**: a $31c/240$ orthogonal spaceship oriented so that it travels up. The track-building mechanism from Figure 10.4(a) is highlighted in yellow, a track-destroying mechanism like the one from Figure 10.4(b) is highlighted in magenta, and the double-glider forward rakes from Figure 10.6(b) are highlighted in aqua. The remaining un-highlighted rakes implement the 13-step slow salvo synthesis of a heavyweight spaceship from Figure 10.7.

Free PDF. See conwaylife.com/book for pattern files and print version.

If we continue in this way, we quickly arrive at the following sequence of rakes and rephasers that should be placed along the 6-block track so as to synthesize one of the heavyweight spaceships that we need beside it:[12]

```
double rake, 2 rephasers, R6L6, R2L25, 2 rephasers, R6L17, 2 rephasers, R6L6,
    4 rephasers, R1L0, R6L6, 3 rephasers, R6L6, 1 rephaser, R2L23, R2L25,
      4 rephasers, R2L25, 3 rephasers, R2L25, 1 rephaser, R6L6, R2L25.
```

If we use two copies of this entire sequence of rakes and rephasers (since we need to synthesize two heavyweight spaceships on each side of the block track), we finally get a complete $31c/240$ orthogonal spaceship,[13] which is displayed in Figure 10.9. Despite its massive size of 215 338 live cells and $11\,970 \times 48\,047$ bounding box, it is the smallest known $31c/240$ spaceship.[14]

10.2 The Caterpillar

We saw another reaction that can be used to construct a self-supporting spaceship way back in Figure 4.55: a blinker can be used to move a pi-heptomino forward by 17 cells over the course of 45 generations (while just repositioning, not destroying, the blinker). By placing a row of blinkers with a spacing of 17 cells as in Figure 10.10, we thus get a reburnable blinker wick that a pi-heptomino can burn through.

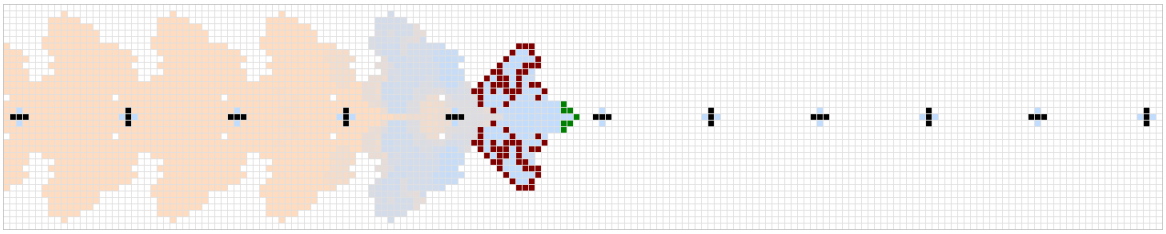


Figure 10.10: A reburnable blinker wick, along which a pi-heptomino fuse (highlighted in green) can move at a speed of $17c/45$. The cells displayed in red make up a large spark that simply dies without affecting anything else.

A single pi-heptomino shifts the blinker wick backward by 6 cells and rephases it,[15] so if we were to place 34 pi-heptominoes on this reburnable wick then it would be moved back by $6 \times 34 = 204$ cells, which, due to the spacing of the blinkers, leaves them all exactly where they would be if no pi-heptominoes burned through them at all.[16] This gives us an infinitely long pattern that moves at a speed of $17c/45$. Our goal is to turn this into a finite configuration (i.e., a spaceship) called the **caterpillar** that moves at the same speed, just like we turned the Herschel-crawling reaction of Figure 10.2 into the $31c/240$ silverfish spaceship.

To this end, we need a way of constructing the blinkers in front of the pi-heptominoes that burn through them, and also a way of cleaning them up behind those pi-heptominoes. Fortunately, most

[12] The timing of most of these rakes and rephasers does not matter, so we can simply pack them as tightly together as possible. However, the final R6L6 has to be delayed slightly so that the HWSS it created arrives at the front double rake at the correct time.

[13] Created by Chris Cain, Dave Greene, and Adam P. Goucher in May 2020.

[14] Actually, there are some modifications of it that can reduce its size a bit—see Exercise 10.3.

[15] Since the blinker wick is infinitely long, it is perhaps better to think of the pi-heptomino as shifting the wick forward by 11 cells and *not* rephasing it.

[16] If we used just 17 pi-heptominoes instead, the blinkers would return to their original positions, but in the opposite phases.

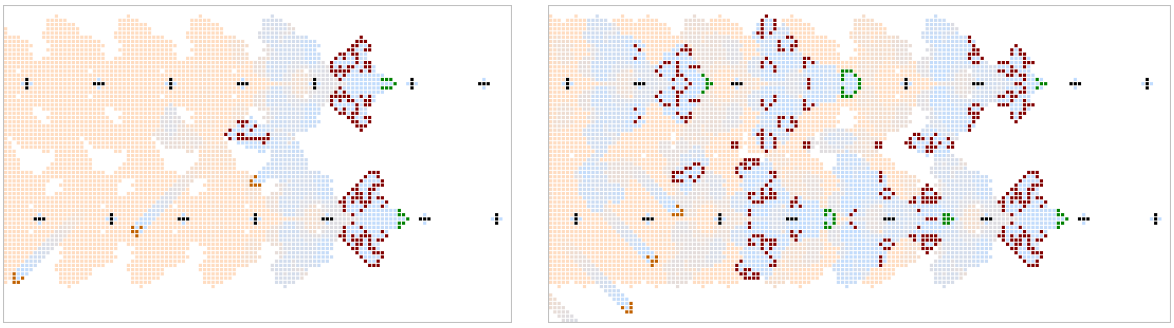
Free PDF. See conwaylife.com/book for pattern files and print version.

of the same ideas that we used when constructing the silverfish still apply: we can use multiple pi-heptominoes on multiple reburnable blinker wicks to create rakes, we can use gliders to clean up the blinker trails, and we can use parallel xWSS streams to help us synthesize the front ends of the blinker wicks.

Most of the details of this construction are quite similar to what they were for the $31c/240$ silverfish, so we omit many of them. Instead, we just focus on two parts of the construction where the details do change considerably.

10.2.1 Pi-Crawler Rakes

The first key difference between the Herschel crawler of Figure 10.2 and the pi crawler that we are now using is that the pi crawler does not emit any gliders on its own. Fortunately, it does create a large spark, and we can collide two of those sparks without much effort so as to create a backward rake, as illustrated in Figure 10.11(a). Creating a forward rake that travels along this same pair of blinker tracks is a bit trickier, but can be done via six pi crawlers (three per track), as in Figure 10.11(b).



(a) A backward pi-crawler rake.

(b) A forward pi-crawler rake.

Figure 10.11: A pair of p45 (a) backward and (b) forward rakes that crawl along a pair of blinker tracks that are separated by 32 cells.

While the blinker tracks that these rakes use have the same spacing as each other (32 cells), the relative phases of those blinker tracks are different. As a result, it is not possible to place a forward rake directly behind a backward rake, or vice-versa, unless we rephase one of the tracks. Fortunately, we can always properly rephase the tracks by simply inserting additional pi crawlers on one of them, since $\gcd(11,34) = 1$.[17]

With that technicality out of the way, we can then place a forward rake on the same pair of tracks behind a backward rate so that their gliders collide and synthesize other objects. For example, we can use these rakes to synthesize and/or destroy additional blinker tracks, as illustrated in Figure 10.12. By adjusting the number of pi rephasers that we use on each track, as well as the relative timing of the forward and backward rakes, we can adjust this reaction to even synthesize a pair of blinker trails that are 32 cells apart, thus allowing additional rakes to travel along them (see Exercise 10.10).

We can thus use a single pair of blinker tracks to create as many additional pairs of blinker tracks as we like. Furthermore, those tracks can all clean up after each other, by using rakes to fire gliders at each other so as to destroy the blinkers. All that remains is to synthesize the front of the very first pair of blinker tracks.

[17] This is directly analogous to how we could always properly rephase the six-block track in the silverfish since $\gcd(9,31) = 1$.

Free PDF. See conwaylife.com/book for pattern files and print version.

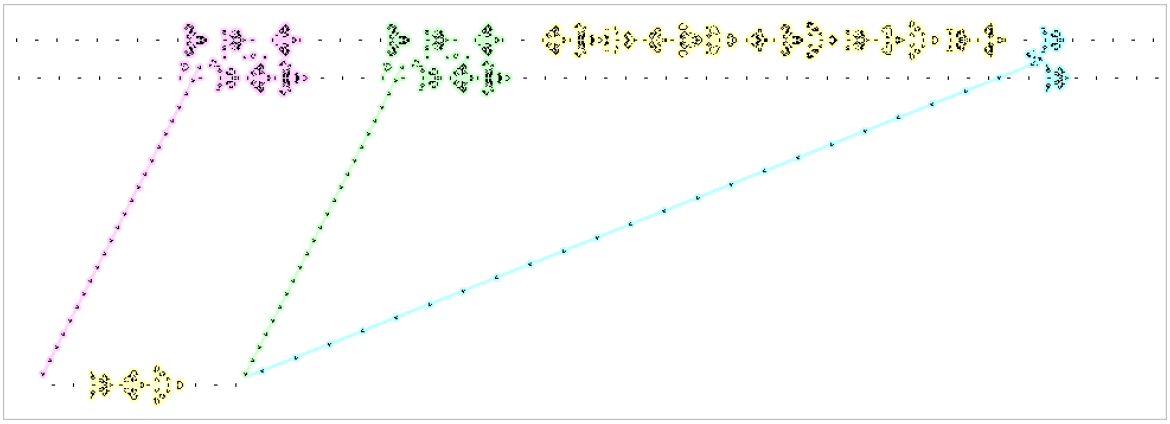
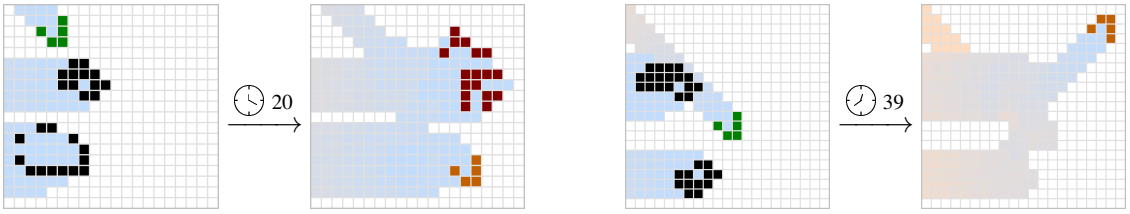


Figure 10.12: A backward rake (highlighted in aqua) followed by fourteen rephaser pis (highlighted in yellow) that allow a forward rake (highlighted in green) to use the same pair of blinker tracks. Those two rakes synthesize a parallel blinker track, which is then destroyed by a third rake (highlighted in magenta).

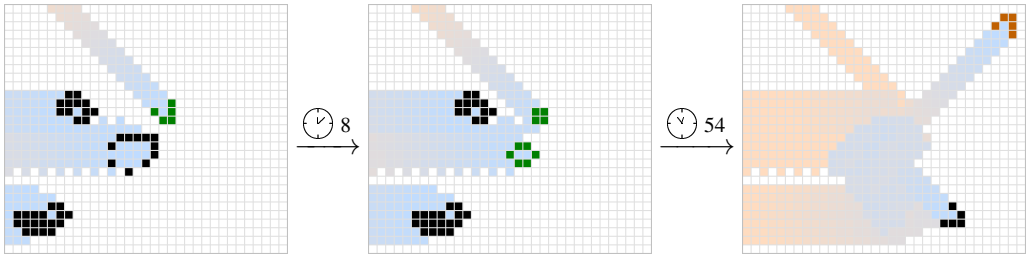
10.2.2 Helices

Just as was the case with the silverfish, synthesizing the front of the track is the trickiest part of the caterpillar. Our goal is to use rakes on blinker track pairs to synthesize xWSSes that reach out in front so as to then synthesize the front of the first pair of blinker tracks. However, there is one additional wrinkle here that was not present in the silverfish: because the $17c/45$ reaction that we are using travels faster than $c/4$,[18] we cannot fire gliders forward at those xWSSes to change their direction (we used this trick from Figure 10.5(a) at the top-right corner of the silverfish in Figure 10.9).



(a) Shifting a glider by $(10, 13)$ in 20 generations.

(b) Reflecting a glider in 39 generations.



(c) Reflecting a glider in 62 generations, while duplicating it. Delaying the final pair of xWSSes delays the output gliders by the same amount, without affecting their position.

Figure 10.13: Some methods of colliding a glider with xWSS flotillae so as to (a) move the glider, (b) reflect it, and (c) duplicate it. The spark displayed in red dies off in another 9 generations.

The way around this problem is to use a **helix**:[19] a configuration of xWSSes that has a burning

[18] Unlike the $31c/240$ reaction at the heart of the silverfish, which is slower than $c/4$.

[19] The name "helix" comes from the helical shape that these patterns often end up having—see the upcoming Figure 10.14.

Free PDF. See conwaylife.com/book for pattern files and print version.

front end that causes it to (a) travel at a speed slower than $c/2$ (in our case, we want a speed of $17c/45$), and (b) periodically fire gliders off to at least one side. One way to make a helix is to have a glider do the burning—the three reactions displayed in Figure 10.13 show how a glider can destroy and be repositioned (and possibly duplicated) by small xWSS flotillae.[20]

By stringing these reactions together, we can create helices that a glider burns through at any rational speed of our choosing (up to the $c/2$ speed limit established by Theorem 4.1). For example, the helix displayed in Figure 10.14 takes 270 generations to push a glider forward orthogonally by 102 cells, giving it a speed of $102c/270 = 17c/45$—exactly the speed that we need for the caterpillar spaceship.

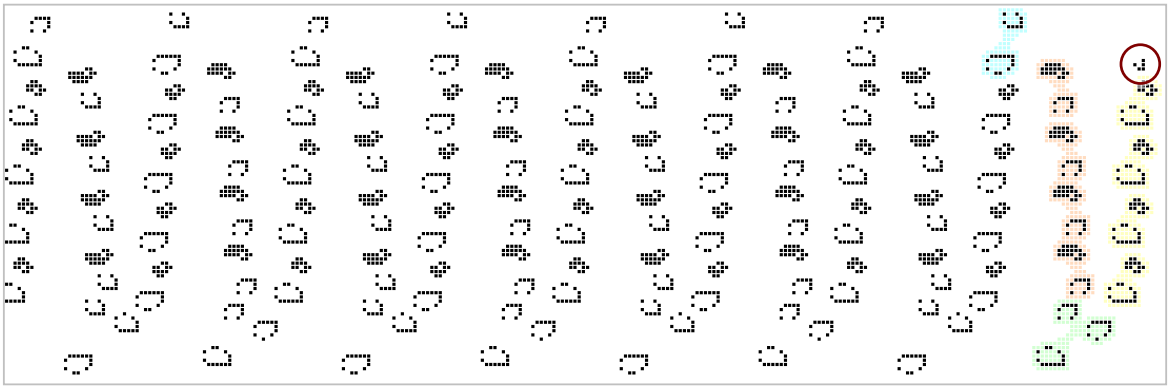


Figure 10.14: A $102c/270 = 17c/45$ helix that is used by the caterpillar spaceship. Four of the glider-shifting flotillae (highlighted in yellow) move the glider south, the flotilla highlighted in green reflects it and produces an extra glider to the southeast, four more of the glider-shifting flotillae (highlighted in orange) move the glider back north, and then the flotilla highlighted in aqua reflects it back to the south. This sequence of 21 xWSSes then repeats indefinitely.

It was not just a lucky coincidence that we were able to create a helix of the desired $17c/45$ speed via these three reactions—we can use them to make an orthogonal helix with the same basic shape as the one from Figure 10.14 that travels at any rational speed slower than $c/2$. To see this, suppose we delay the final xWSS pair in the reflection-and-duplication flotilla by n generations from what is shown in Figure 10.13(c), so that it takes $62+n$ generations for the output gliders to reach the indicated positions, and we use m of the glider-shifting flotillae in each direction (so that $m = 4$ in Figure 10.14). The resulting helix moves the glider forward orthogonally by $20m+20$ cells over the course of $40m+n+101$ generations:

- The top and bottom sides, taken together, move the glider forward by 20 cells over the course of $39+(62+n) = 101+n$ generations.
- The central glider-shifting flotillae each move the glider forward by 10 cells over the course of 20 generations. Since there are $2m$ such flotillae (m in each direction), they move the glider forward by $20m$ cells over the course of $40m$ generations.

This helix thus travels at a speed of

$$\frac{(20m+20)c}{40m+n+101}.$$

To create such a helix with speed $(p/q)c$, where p and q are arbitrary positive integers with $p/q < 1/2$,

[20] These reactions were found by Paul Callahan's *gencols* search program—see conwaylife.com/wiki/LifeWiki:Life_links#Downloadable_Computation_and/or_Search_Software.

Free PDF. See conwaylife.com/book for pattern files and print version.

we can simply choose

$$m = 4p - 1 \qquad \text{and} \qquad n = 80(q - 2p) - 61. \tag{10.1}$$

The fact that m is a non-negative integer is clear, and n being a non-negative integer follows from the fact that $p/q < 1/2$, so $q - 2p \geq 1$.[21] Some mildly ugly algebra also reveals that

$$\begin{aligned}\frac{20m+20}{40m+n+101} &= \frac{20(4p-1)+20}{40(4p-1)+(80(q-2p)-61)+101} \\ &= \frac{80p-20+20}{160p-40+80q-160p-61+101} = \frac{80p}{80q} = \frac{p}{q},\end{aligned}$$

as desired. For emphasis, we state what we have just demonstrated as a theorem:

Theorem 10.1 — Universality of Orthogonal Helices
The three reactions from Figure 10.13 can be used to create an orthogonal helix, which fires gliders forward on one side, with any rational speed slower than $c/2$.

It is worth noting that the values of m and n given by Equation (10.1) are typically much larger than necessary—we chose those *formulas* for m and n to be simple, but much smaller *values* of m and n can typically be found. For example, if $p = 17$ and $q = 45$ then those formulas give $m = 67$ and $n = 819$, which would result in a monstrously large helix consisting of 273 xWSSes. However, the smaller values $m = 16$ and $n = 159$ give a helix of the same speed consisting of "only" 69 xWSSes.

To decrease these values even further, simply notice that we can add[22] some uninterrupted glider travel time to the helix. If we let the glider travel, unaffected by xWSSes, for ℓ additional cells in each direction of the helix, then the helix's displacement is increased by 2ℓ cells, while its period is increased by $4(2\ell) = 8\ell$ generations. The speed of this new helix would then be

$$\frac{(20m+2\ell+20)c}{40m+8\ell+n+101},$$

and strategic choices of ℓ can lead to smaller helices. For example, picking $\ell = 1$, $m = 4$, and $n = 1$ gives us exactly the $17c/45$ helix from Figure 10.14 that consists of 21 xWSSes.

10.2.3 The Caterpillar Itself

Putting all of these ideas together finally gives us the caterpillar spaceship,[23] displayed in Figure 10.15, which has a very similar shape and basic structure to that of the silverfish. However, it is absolutely gargantuan, even compared to the silverfish—it is over 7 times as long and has over 50 times as many live cells. The primary reason for this drastic increase in size is that it has to synthesize the 21-xWSS helix from Figure 10.14 on its side, whereas the silverfish just needed to synthesize the 2-HWSS flotilla from Figure 10.5(a).

Actually, it's worse than this—because the caterpillar's helix has period $102 = 6 \times 17$, it seems like we should need 24 copies of this helix (6 on each side for each of the 2 blinker tracks that they need to synthesize). However, the caterpillar uses some glider-duplication tricks so that it "only"

[21] The inequality $p/q < 1/2$ is equivalent to $q - 2p > 0$. To get the (seemingly) stronger inequality $q - 2p \geq 1$, recall that p and q are integers.

[22] Or subtract—some uninterrupted travel time can be removed from the output gliders in the reactions from Figures 10.13(b) and 10.13(c).

[23] The components used in the caterpillar were found by Gabriel Nivasch, David Bell, and Jason Summers, and construction was completed in December 2004 by a computer program written by Nivasch (see gabrielnivasch.org/fun/life/caterpillar).

Free PDF. See conwaylife.com/book for pattern files and print version.

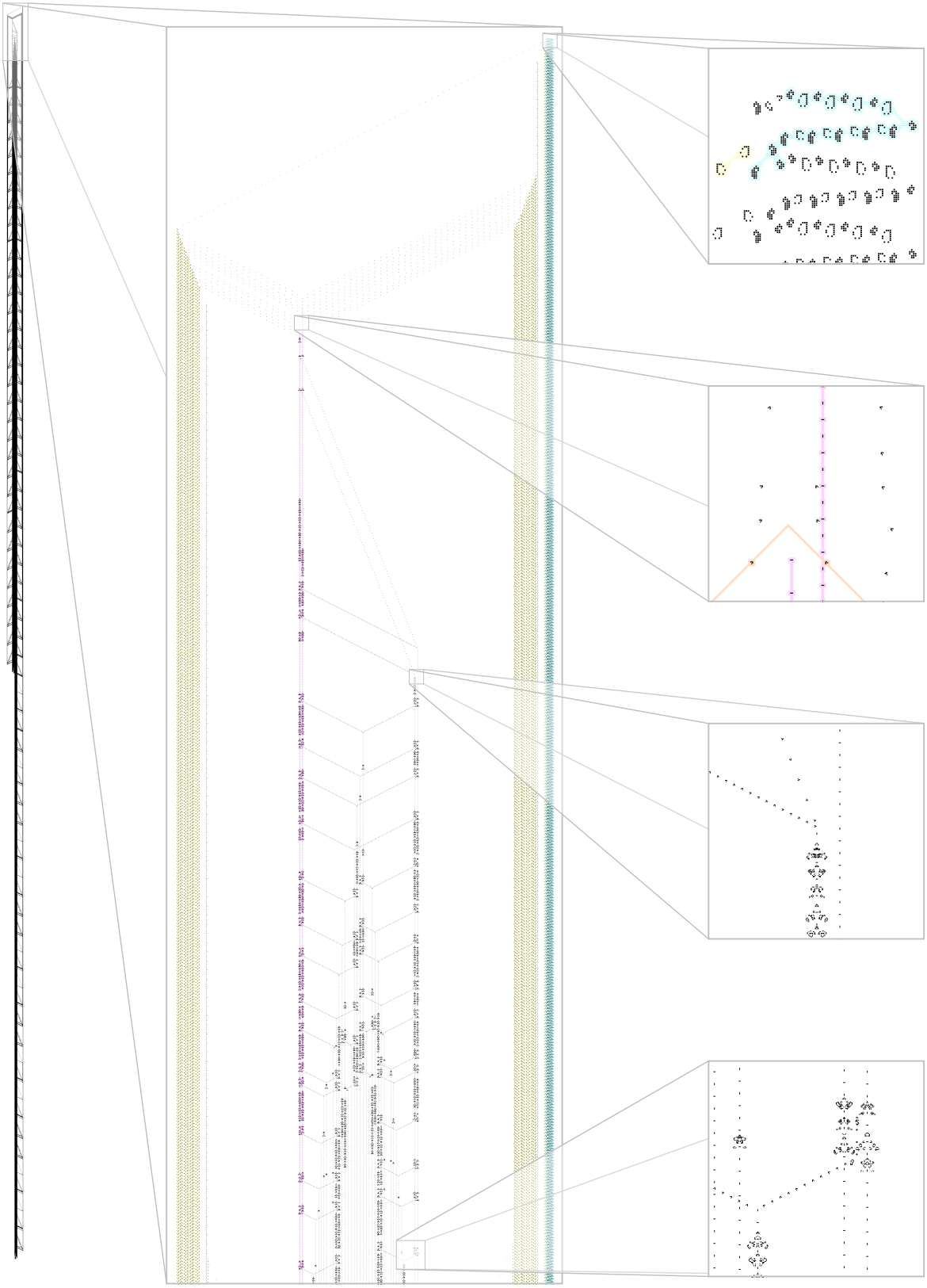


Figure 10.15: The $17c/45$ orthogonal **caterpillar** spaceship, travelling up. The helix along its right side (highlighted in aqua) periodically fires a glider that hits numerous glider duplicators (highlighted in yellow) so as to synthesize the frontmost pair of blinker tracks (highlighted in magenta). Those two blinker tracks then synthesize 38 more blinker tracks so that numerous pi-crawler rakes can travel on them and synthesize the afore-mentioned helix and glider duplicators.

Free PDF. See conwaylife.com/book for pattern files and print version.

needs an extra 30 or so xWSSes on each side (see Exercise 10.14). To synthesize this extreme number of xWSSes on its sides a bit more efficiently, the caterpillar uses several columns of rakes on parallel tracks to implement non-slow synthesis, whereas the silverfish used just a single column of rakes down its center to implement slow salvo synthesis.[24]

10.3 Oblique Helices and the Waterbear

Another self-supporting spaceship that works in much the same way as the silverfish and caterpillar is the **waterbear**. The reaction at the heart of this spaceship is displayed in Figure 10.16, and consists of a Herschel colliding with a glider so as to displace itself by $(23,5)$ over the course of 79 generations while duplicating the glider.

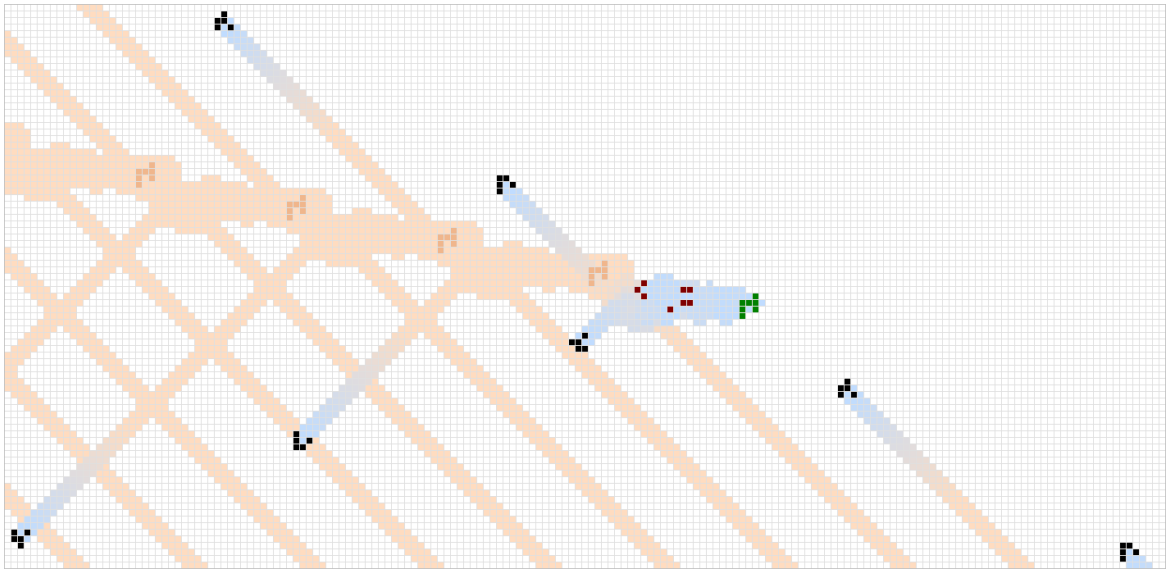


Figure 10.16: The **(23,5)c/79 reaction**, in which a Herschel (displayed in green) moves obliquely along a glider track while duplicating that track. The cells displayed in red simply die off in two more generations.

Much like the $31c/240$ reaction from Figure 10.1 and the $17c/45$ reaction from Figure 10.10, this $(23,5)c/79$ reaction is a prime candidate for the central spine of a self-supporting spaceship. We "just" need to find a way to use its output gliders to synthesize the necessary input gliders from the front. We do not explore most of the techniques that make this possible. However, we do note that, since this reaction moves faster than a glider (i.e., $c/4$), we again need to use the backward gliders to synthesize a helix that travels to the front of the spaceship.

Constructing a helix that travels at the correct speed might seem problematic, since xWSSes all move orthogonally, whereas we need the helix to move obliquely at $(23,5)c/79$. Remarkably, this is not actually a problem—the exact same reactions that we used in Section 10.2.2 to create orthogonal helices can in fact be used to create *oblique* helices that travel at the same speed and angle as any oblique spaceship.[25]

[24] Also, slow salvo synthesis was not as fully developed when the caterpillar was constructed, so non-slow synthesis was the only realistic option at the time.

[25] We know from Exercise 4.26 that a spaceship that displaces itself by (x,y) over the course of its period must have speed no larger than $(x,y)c/(2x+2y)$.

Free PDF. See conwaylife.com/book for pattern files and print version.

Theorem 10.2 — Universality of Oblique Helices
For any integers $x \geq y > 0$, the three reactions from Figure 10.13 can be used to create an oblique helix, which fires gliders forward on one side, with any rational speed that is no faster than $(x,y)c/(2x+2y)$. In particular, such helices exist that travel at the same speed and direction as any oblique spaceship.

Indeed, the only modification that we have to make to our orthogonal method of constructing helices is to use fewer glider-shifting flotillae on one half of the helix than on the other. For example, the helix displayed in Figure 10.17 uses two copies of that flotilla in one direction, but only one copy of it in the other, and ends up travelling obliquely at a speed of $(50,13)c/161$.

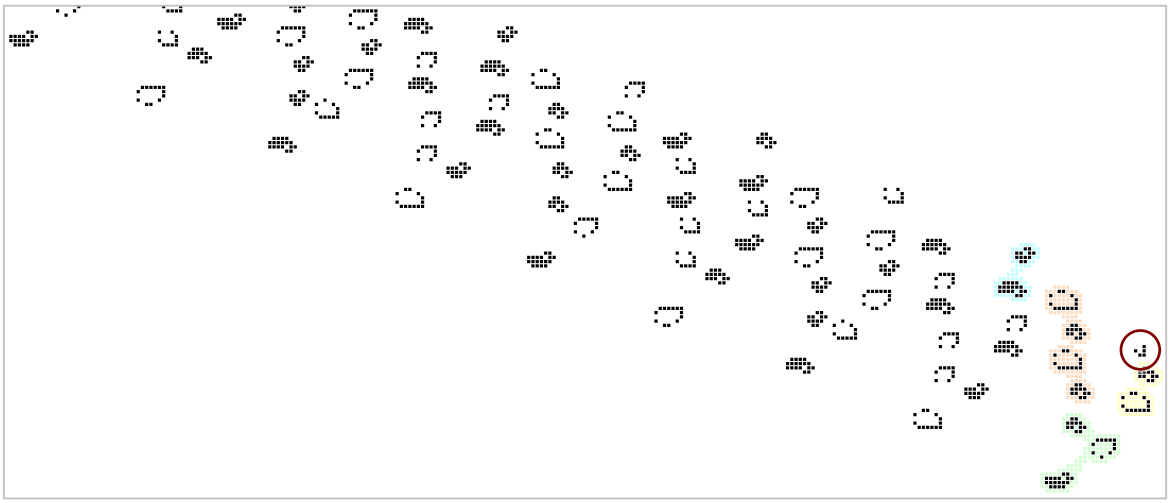


Figure 10.17: A $(50,13)c/161$ helix that is oblique since it uses fewer of the glider-shifting flotillae to move the glider south (highlighted in yellow) than it does to move the glider back north (highlighted in orange).

We should also note that, for oblique helices, there are two different "forward" directions (and two different "backward" directions) that a glider can be fired in. That direction is determined by which side of the helix has the glider-duplicating flotilla on it. For example, interchanging the green and aqua flotillae in Figure 10.17 results in a helix that travels at the same speed and slope, but fires forward gliders at a different angle.

Proof of Theorem 10.2. We proceed mostly as in the proof of the corresponding result for orthogonal helices (Theorem 10.1)—the main difference is that instead of using m of the glider-shifting flotillae in each direction of the helix, we use k of them in one direction and m of them in the other (so, for example, $k = 2$ and $m = 1$ in the oblique helix from Figure 10.17). This helix travels at a speed of

$$\frac{(10k+10m+20, 13k-13m)c}{20k+20m+n+101}, \tag{10.2}$$

where n is again the number of generations that the final xWSS pair is delayed in the glider duplicator from Figure 10.13(c).[26]

To create such a helix with speed $(x,y)c/q$, where x, y, and q are arbitrary positive integers with

[26]For example, if $k = 2$, $m = 1$, and $n = 0$ then we get a helix with speed $(20+10+20, 26-13)c/(40+20+101) = (50,13)c/161$, as in Figure 10.17.

Free PDF. See conwaylife.com/book for pattern files and print version.

$y/q \le x/q < 1/2$,[27] we can simply choose

$$k = 13x + 10y - 1, \qquad m = 13x - 10y - 1, \qquad \text{and} \qquad n = 260(q - 2x) - 61.$$

The fact that k is a non-negative integer is clear, m being a non-negative integer follows from the fact that $x \ge y \ge 1$, so $m = 13x - 10y - 1 \ge 3x - 1 \ge 2$, and n being a positive integer follows from the fact that $x/q < 1/2$ implies $q - 2x \ge 1$. Finally, direct (and more-than-mildly ugly) calculation reveals that

$$\begin{aligned}
&\frac{(10k + 10m + 20, 13k - 13m)c}{20k + 20m + n + 101} \\
&\quad = \frac{\big(10(13x + 10y - 1) + 10(13x - 10y - 1) + 20, 13(13x + 10y - 1) - 13(13x - 10y - 1)\big)c}{20(13x + 10y - 1) + 20(13x - 10y - 1) + (260(q - 2x) - 61) + 101} \\
&\quad = \frac{(130x + 100y - 10 + 130x - 100y - 10 + 20, 169x + 130y - 13 - 169x + 130y + 13)c}{260x + 200y - 20 + 260x - 200y - 20 + 260q - 520x - 61 + 101} \\
&\quad = \frac{(260x, 260y)c}{260q} = \frac{(x, y)c}{q},
\end{aligned}$$

as desired. ■

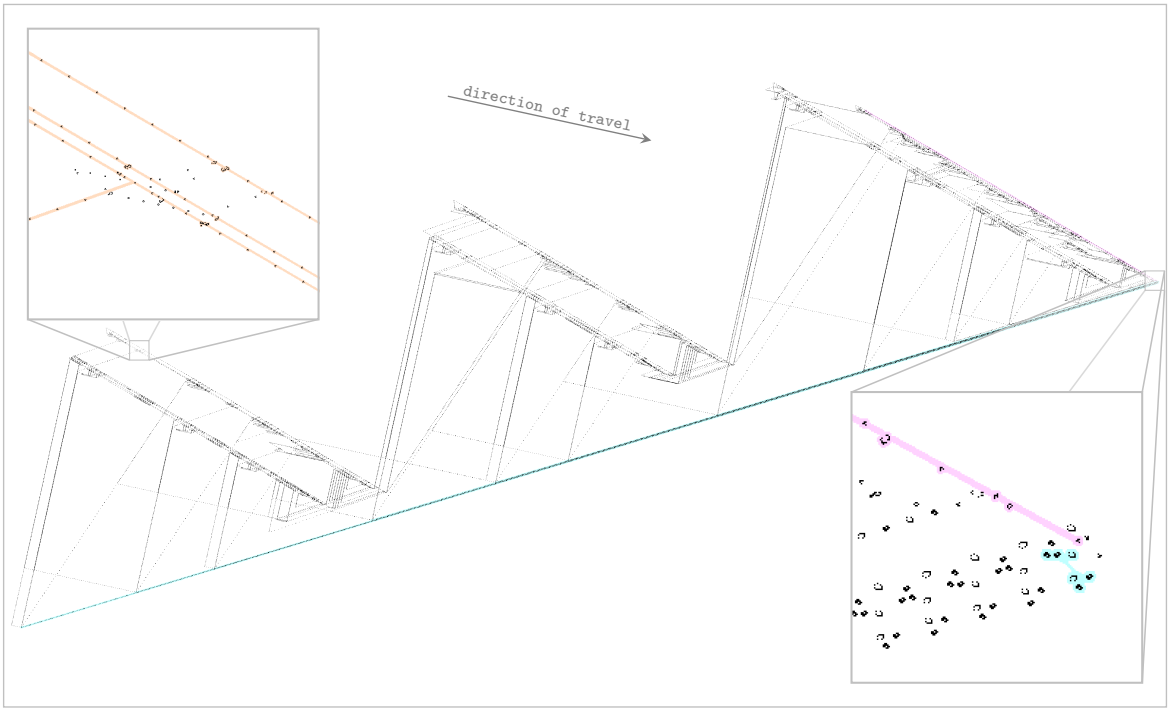


Figure 10.18: The $(23,5)c/79$ oblique **waterbear** spaceship. The helix along its southeast side (highlighted in aqua) periodically fires a backward glider that supports a Herschel via the $(23,5)c/79$ reaction on its northeast side (highlighted in magenta). The output gliders from that reaction are then used to synthesize additional tracks and the afore-mentioned helix. Constructed by Brett Berger in December 2014, based on numerous reactions (such as the helix) that were found by Ivan Fomichev.

[27] This is actually a stronger result than in the statement of the theorem. These helices can travel diagonally at speeds arbitrarily close to $c/2$, for example, despite the $c/4$ diagonal spaceship speed limit. The reason that this does not contradict Theorem 4.1 is that the glider is not travelling through empty space—it is burning through xWSSes.

Free PDF. See conwaylife.com/book for pattern files and print version.

The waterbear's helix makes use of slightly different components than the ones we introduced for use in the caterpillar (see Exercise 10.19 and Figure 10.18), so as to reduce its size down to just 6 xWSSes, and also so that it fires gliders backwards instead of forwards.[28] Fortunately, we can construct backwards-firing helices that travel at any speed and slope, just as we could with forwards-firing helices (see Exercise 10.17).

Unlike the caterpillar, in which the primary $17c/45$ reaction moved parallel to the helix, the waterbear's $(23,5)c/79$ reaction runs at a different angle than the helix. The result of this change is that, instead of being long and thin like the caterpillar, the simplest waterbear to construct is triangular, with the helix along one side of the triangle and the $(23,5)c/79$ reaction along another. To keep its size down a bit, the $(23,5)/79$ Herschel-and-glider track on one of the sides is "reset" at two points, so that the resulting waterbear looks like it is made up of 3 much smaller triangles instead. As a result, it has 197 896 live cells and fits in a $13\,295 \times 28\,010$ bounding box, making it comparable in size to the silverfish.

10.4 Caterloopillars

The final family of self-supporting spaceships that we look at are called **caterloopillars**. The basic idea behind these spaceships is the same as it was for the silverfish, caterpillar, and waterbear: many copies of a central moving reaction are used to produce gliders that collaborate so as to send signals forward and build their own support structures ahead of themselves. However, we now flip this idea on its head—what if instead of using an infinite trail of stable objects like blocks or blinkers to support a naturally moving object like a Herschel or pi-heptomino, we use an infinite trail of spaceships to move a stable object?[29]

10.4.1 Loaf Tracks

For a still-life-moving reaction to actually be usable at the core of a self-supporting spaceship, it also needs to produce some sort of output (ideally a glider) that can then be used to synthesize the spaceships that pull the stable object. One such reaction, which uses three xWSS flotillae to push a loaf forward while also generating a glider, is displayed in Figure 10.19.[30]

We will thus use a track of loaves as the central spine of a self-supporting spaceship, with this 9-xWSS flotilla being used to push it forward. As the flotilla pushes the loaves, it also releases gliders that can be used to synthesize supporting mechanisms via unidirectional slow salvo synthesis (in almost exactly the same way as the silverfish used unidirectional slow salvo synthesis to synthesize two HWSSes on each side).

Perhaps the simplest supporting mechanism to synthesize is an xWSS flotilla travelling in the opposite direction that *pulls* a second loaf track by the same amount that the first loaf track is *pushed*. By arranging these two flotillae so that they run on parallel tracks in opposite directions, we can have the loaf-pushing and loaf-pulling flotillae synthesize each other, as illustrated in Figure 10.20.[31]

[28] Again, for oblique helices there are two different "backwards" directions that gliders can be fired in.

[29] This basic idea for the construction of a caterloopillar spaceship was originally proposed by David Bell in October 2006. It wasn't until April 2016 that slow-salvo technology had advanced to the point that it was actually constructed.

[30] In contrast with the glider-manipulating flotillae that are used in helices (refer back to Section 10.2.2), these flotillae are *not* destroyed by the glider.

[31] The fact that these two flotillae synthesize each other is what gives caterloopillars their name (besides the obvious analogy with the caterpillar)—the xWSSes form a long loop that seems to be synthesizing itself. The name is also a reference to the term "strange loop", a concept explored in Douglas Hofstadter's famous book *Gödel, Escher, Bach*.

Free PDF. See conwaylife.com/book for pattern files and print version.

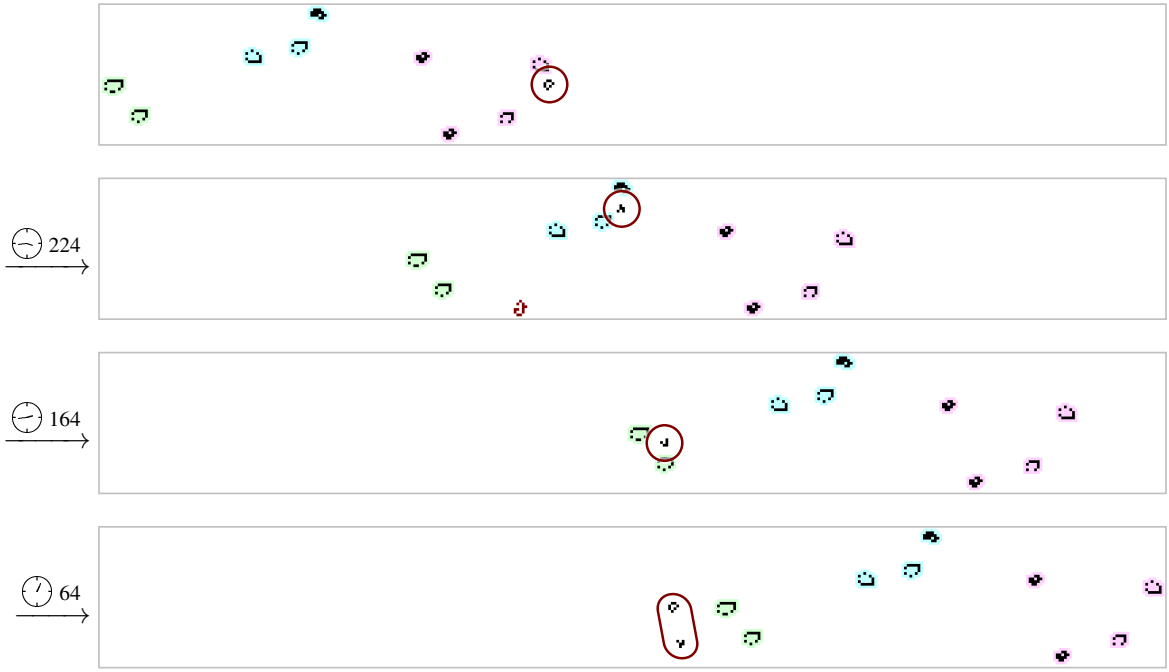


Figure 10.19: An arrangement of three xWSS flotillae that push a loaf forward by 46 cells and simultaneously create a glider. The first 4-xWSS flotilla (highlighted in magenta) turns the loaf into a glider, the next 3-MWSS flotilla (highlighted in aqua) reflects that glider, and finally the 2-xWSS flotilla (highlighted in green) creates a loaf from that glider (without destroying it). Moving the aqua flotilla northwest by n cells and the green flotilla west by $2n$ cells results in the loaf being pushed forward by $46+2n$ cells.

We just have three tasks that remain in order to actually construct a caterloopillar spaceship with this shape:

1) Build the loaf-pulling xWSS flotilla.
2) Build mechanisms for destroying the flotillae at the front and back of the spaceship.
3) Come up with unidirectional slow salvo syntheses for these flotillae, and encode them in the spacing between loaves.

We tackle these three issues one at a time. For (1), we proceed much like we did with the loaf-pushing flotilla that we saw in Figure 10.19: a loaf-*pulling* flotilla can be built out of smaller flotillae that turn the loaf into a glider, and then rebuild the loaf while duplicating the glider. The particular loaf-pulling flotilla that we use is displayed in Figure 10.21.

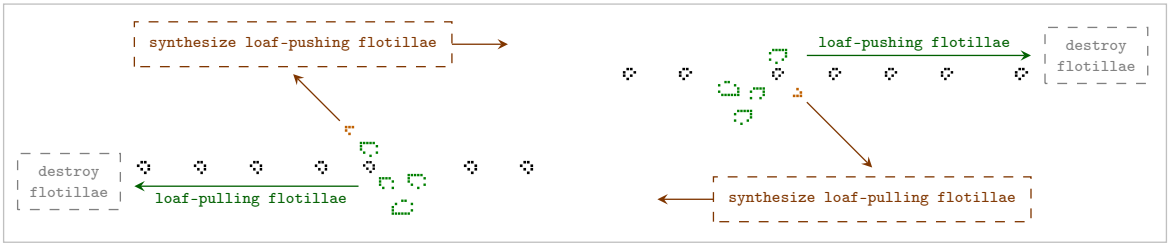


Figure 10.20: A schematic for a caterloopillar spaceship that travels to the right. Some loaf-pushing flotillae travel along a loaf track that encodes a unidirectional slow-salvo synthesis of a loaf-pulling flotilla. Those loaf-pulling flotillae travel backwards along a parallel loaf track that encodes a unidirectional slow-salvo synthesis of the loaf-pushing flotilla.

Free PDF. See conwaylife.com/book for pattern files and print version.



Figure 10.21: An arrangement of three xWSS flotillae that pull a loaf backward by 54 cells and simultaneously create a glider. The first 4-xWSS flotilla (highlighted in magenta) turns the loaf into a pair of gliders, the next 3-LWSS flotilla (highlighted in aqua) reflects one of those gliders, and the final LWSS (highlighted in green) turns that glider back into a loaf. Moving the aqua flotilla south by n cells and west by $3n$ cells, and the green flotilla west by $6n$ cells, results in the loaf being pulled backward by $54+2n$ cells.

While this reaction does not pull the loaf by the same number of cells that the reaction from Figure 10.19 pushes it, the spacing between the component flotillae that make up these reactions can easily be adjusted so as to fix this problem. In particular, for every integers $m, n \geq 0$, we can adjust the loaf-pushing and loaf-pulling flotillae to move the loaf by $46+2m$ and $54+2n$ cells, respectively. Choosing $m = n+4$ results in both flotillae moving the loaf track by $54+2n$ cells.

10.4.2 The Front and Back Ends

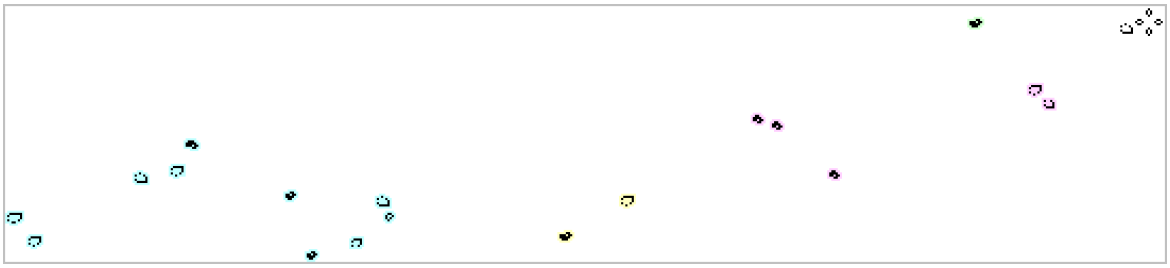
To take care of task (2) above—destroying the xWSS flotillae after they are done pushing and pulling the loaf tracks—we can place a constellation of still lifes at the ends of the tracks. The tricky part is moving that constellation down the track at the same speed as the loaves.

To solve this problem, we instead use a *single* still life along with additional xWSS flotillae. Those extra flotillae are arranged so as to convert the still life into a glider that then (a) reconstructs a copy of that single still life farther down the track, and (b) builds a constellation of still lifes that destroys the loaf-moving flotillae. A bit more specifically, we use additional xWSS flotillae that transform the still life as follows:

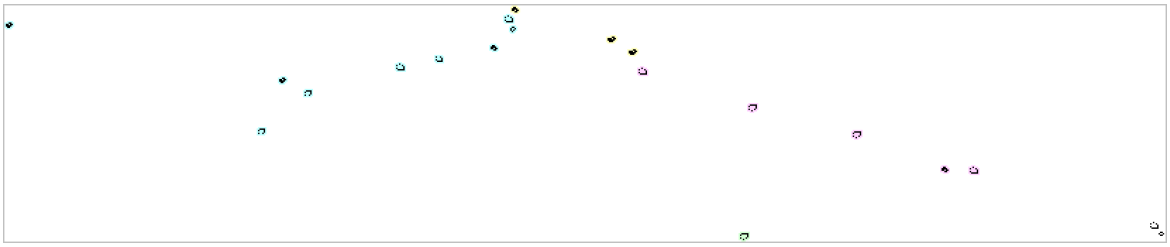
$$\text{still life} \xrightarrow{\text{xWSSes}} \text{glider} \xrightarrow{\text{xWSSes}} \text{two gliders} \begin{cases} \xrightarrow{\text{xWSSes}} \text{original still life, moved forward or backward} \\ \xrightarrow{\text{xWSSes}} \text{constellation of still lifes in front of the loaf-movers.} \end{cases}$$

Free PDF. See conwaylife.com/book for pattern files and print version.

Unlike the loaf-moving flotillae, we need the xWSSes in these flotillae to be destroyed as they manipulate the still life and gliders. However, this does not make it any harder to find flotillae that get the job done (in fact, we already saw many one-time glider manipulating flotillae like this when we investigated helices in Section 10.2.2). Particular flotillae that can be used at the front and back of the caterloopillar, to destroy the loaf-pusher and loaf-puller that we have already seen, are displayed in Figure 10.22.[32]



(a) A honey farm and 9-xWSS flotilla that can be placed in front of the loaf-pushing flotilla from Figure 10.19 (highlighted in aqua) so as to destroy it. With the spacing displayed here, the loaf and honey farm are each pushed forward by 62 cells. To increase that displacement to $62+2n$ cells, move the top-center MWSS (highlighted in green) west by $2n$ cells, the next five xWSSes (highlighted in magenta) southwest by n cells, and the next two MWSSes (highlighted in yellow) and loaf-pusher southwest by $2n$ cells.



(b) A beehive and 10-xWSS flotilla that can be placed in front of the loaf-pulling flotilla from Figure 10.21 (highlighted in aqua) so as to destroy it. With the spacing displayed here, the loaf and beehive are each pulled backward by 94 cells. To increase that displacement to $94+2n$ cells, move the bottom-center MWSS (highlighted in green) west by $6n$ cells, the next five xWSSes (highlighted in magenta) north by n cells and west by $3n$ cells, and the next three xWSSes (highlighted in yellow) and loaf-pusher north by $2n$ cells and west by $4n$ cells.

Figure 10.22: Some xWSS flotillae and still lifes that can be placed in front of the loaf-moving flotillae so as to destroy them. When adjusting these flotillae, also adjust the loaf-pushing and loaf-pulling flotillae as indicated in the captions of Figures 10.19 and 10.21, respectively.

10.4.3 Putting It All Together

All that is left now to complete the caterloopillar is task (3)—coming up with a unidirectional slow-salvo glider synthesis of the xWSS flotillae from Figure 10.22 that will travel antiparallel to each other. In fact, we even need this glider synthesis to be **monochrome**—consisting entirely of gliders of just one color. Indeed, the locations from which gliders are fired is determined by the spacing of the loaves on the loaf track, and that spacing is orthogonal (i.e., the loaves and hence gliders in the slow salvo are separated by full diagonals, not half diagonals).

Such a glider synthesis will consist of hundreds of steps, so actually coming up with it is best left to a computer script.[33] Once we have that glider synthesis, though, we are done, and get a completed caterloopillar like the one displayed in Figure 10.23.

[32] The flotilla from Figure 10.22(a) that destroys the loaf-pusher also releases a backwards glider—see Exercise 10.23.

[33] Such a script, which was made by Michael Simkin and used to construct the first caterloopillars in April 2016, is available at conwaylife.com/forums/viewtopic.php?p=30104#p30104.

Free PDF. See conwaylife.com/book for pattern files and print version.

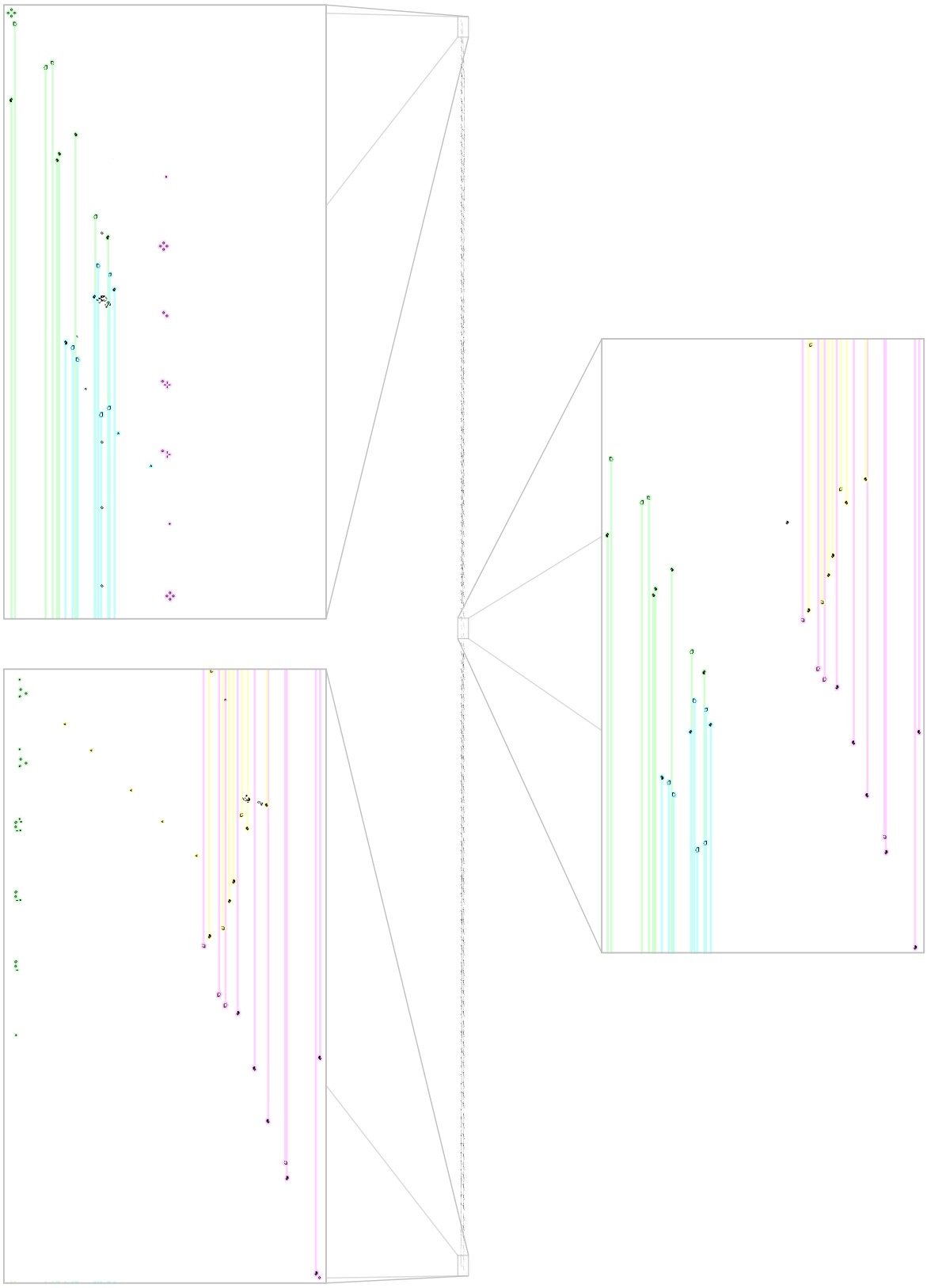


Figure 10.23: A $c/24$ orthogonal **caterloopillar** spaceship, travelling up. At the top, the loaf-pushing flotillae (highlighted in aqua) and their destroyers (highlighted in green) begin synthesizing the loaf-pulling flotillae (highlighted in yellow) and their destroyers (highlighted in magenta). At the bottom, the loaf-pullers begin synthesizing the loaf-pushers. In the middle, these flotillae simply pass by each other.

Free PDF. See conwaylife.com/book for pattern files and print version.

This particular caterloopillar pushes and pulls the loaf track by 98 cells per period, and the forward-travelling xWSS flotillae are separated by 1 078 cells, for a speed of $98c/(2(1078+98)) = 98c/2352 = c/24$.[34] Caterloopillars of many other speeds can be straightforwardly created from this one, though. For example, we could straightforwardly (though tediously!) move the flotillae farther apart, resulting in larger caterloopillars that travel much more slowly.

Similarly, we can also squeeze the flotillae closer together, resulting in slightly smaller and faster caterloopillars, but we are much more limited in this direction. For example, the forward flotillae used by this caterloopillar must be separated by at least 352 cells, so these simple adjustments cannot possibly produce caterloopillars that are any faster than $98c/(2(352+98)) = 49c/450$.

To create even faster caterloopillars, we have to increase how far the flotillae from Figure 10.22 push and pull their still lifes, which requires us to adjust the slow salvo syntheses that are used to construct those flotillae. We thus have to rebuild the caterloopillar almost entirely from scratch (or, rather, the computer script has to rebuild it from scratch). If we are willing to put in this extra bit of effort, then we can construct caterloopillars that travel at any rational speed slower than $c/4$:[35]

Theorem 10.3 — Caterloopillar Speeds
Caterloopillar spaceships can be constructed that travel at any rational speed slower than $c/4$.

Proof. For any integers $m, n \geq 0$, if the flotilla from Figure 10.22(a) is configured to push the loaf and honey farm forward by $100+2n$ cells, then we can separate subsequent copies of this flotilla by $400+2n+2m$ cells.[36] Similarly, if the flotilla from Figure 10.22(b) is configured to pull the loaf and beehive backward by $100+2n$ cells, then we can separate separate copies of it by $600+6n+2m$ cells.

It follows that the speed of the resulting caterloopillar is

$$\frac{(100+2n)c}{2\big((400+2n+2m)+(100+2n)\big)} = \frac{(50+n)c}{500+4n+2m}.$$[37]

To create a caterloopillar with speed $(p/q)c$, where $p, q > 0$ are integers with $p/q < 1/4$, it suffices to choose

$$n = 300p - 50 \qquad \text{and} \qquad m = 150(q-4p) - 150.$$

These are both non-negative integers, and straightforward algebra shows that

$$\frac{50+n}{500+4n+2m} = \frac{50+(300p-50)}{500+4(300p-50)+2(150(q-4p)-150)} = \frac{300p}{300q} = \frac{p}{q},$$

as claimed. ∎

It is worth noting that faster caterloopillars are typically larger than slower ones, not just because their component flotillae become more spaced out, but also because the glider syntheses used to construct those flotillae require more steps. *Roughly* the same synthesis recipes can be used in

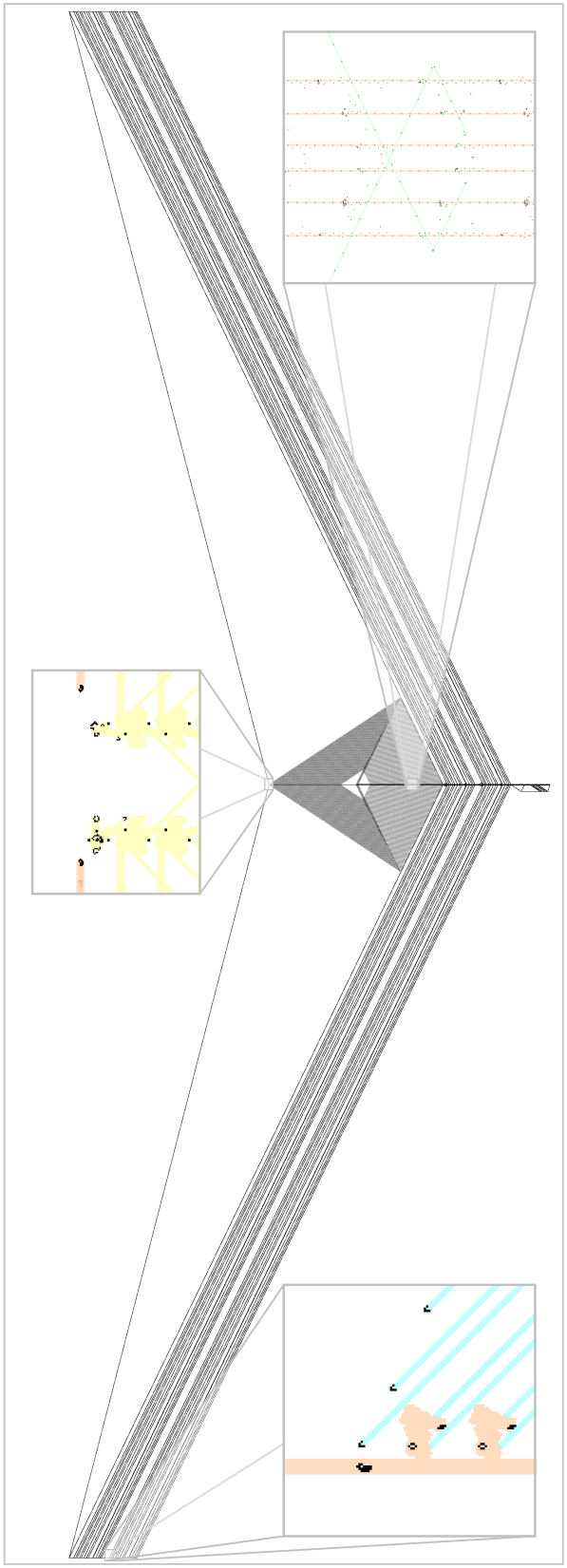
[34] The speed is $98c/(2(1078+98))$ instead of just $98c/(2\cdot 1078)$ because of a "doppler effect": the flotillae do not have to travel just 1 078 cells per period, but also the extra 98 cells that the still lifes were pushed.

[35] This $c/4$ speed limit should be fairly intuitive—as we increase the amount that the caterloopillar displaces its still lifes per period, more and more of its travel time is spent as a $(c/4)$ glider bouncing between the components of the flotillae from Figure 10.22. Reaching the theoretical speed limit of $c/2$ would require a major redesign.

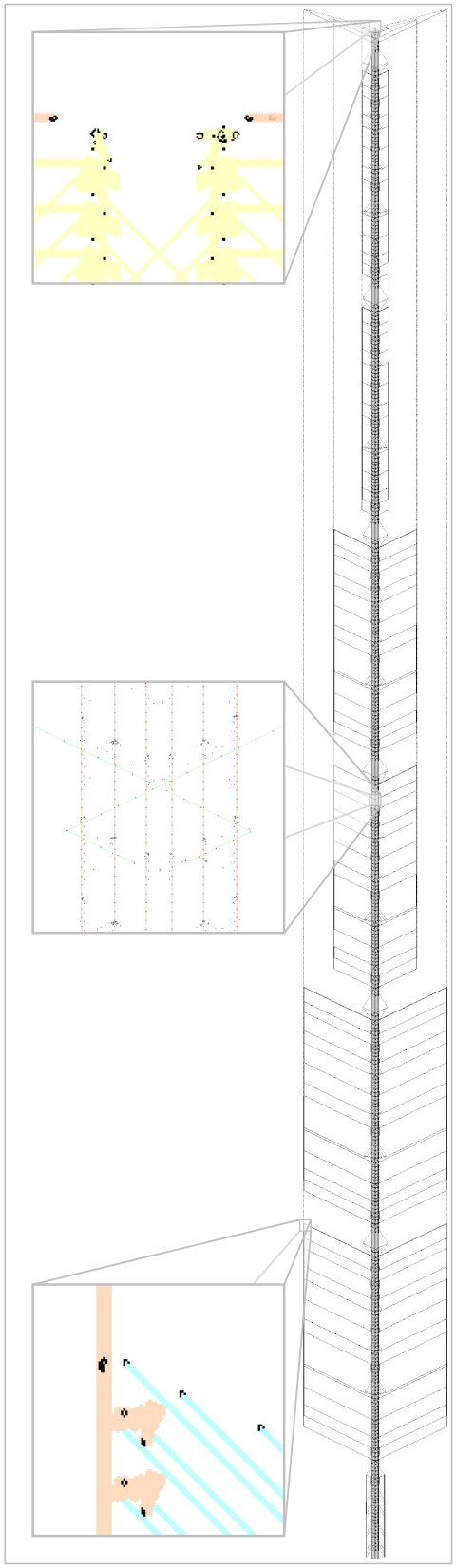
[36] That is, we can separate copies of these flotillae by any even number of cells that is at least as large as $400+2n$. Even though the width of that flotillae is actually $682+4n$ cells, they can "overlap" considerably, with one copy of the loaf pusher being nestled underneath the previous loaf-pusher-destroyer.

[37] We calculated this speed based on the loaf-pulling flotillae. If we instead calculated it based on the loaf-*pulling* flotillae, we would get the same answer: $(100+2n)c/\big(2((600+6n+2m)-(100+2n))\big) = (50+n)c/(500+4n+2m)$.

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) The **shield bug**, travelling to the left.



(b) The **centipede**, travelling up.

Figure 10.24: A pair of (huge) $31c/240$ spaceships based on the same Herschel-crawling reaction as the silverfish. They each use six-block tracks that carry Herschel-based rakes down their center, and the same double-HWSS slow salvo synthesis on their sides to stabilize the front end.

Free PDF. See conwaylife.com/book for pattern files and print version.

all caterloopillars, but additional seed-moving steps are required if the xWSSes that make up the loaf-moving flotillae are separated by large distances.

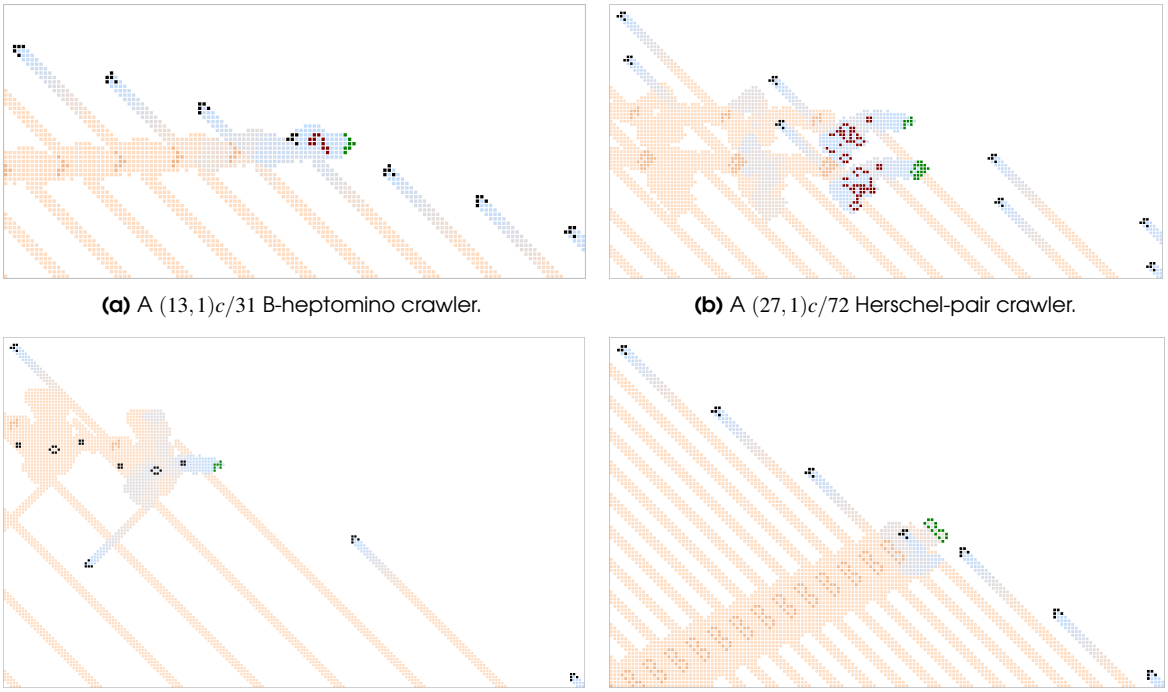
Because of some ugliness involving these unidirectional slow-salvo glider syntheses, caterloopillars have only been explicitly constructed for speeds of the form $(p/q)c$ (where p/q is in lowest terms and $p/q < 1/4$) when $q \not\equiv 2 \pmod 4$. For example, no one has constructed a caterloopillar with speed $c/18$ or $5c/18$, but caterloopillars have been constructed with speeds $c/15$, $c/16$, $c/17$, and $2c/18 = c/9$. Regardless of this limitation, caterloopillars of these troublesome speeds still *exist*, thanks to Theorem 10.3.

The fastest caterloopillar that has been explicitly constructed so far has speed $19c/80$. It is approximately 10 times as long and wide as the $c/24$ caterloopillar from Figure 10.23, and also has about 10 times as many live cells.

10.5 Notes and Historical Remarks

Despite its monstrous size, the silverfish displayed in Figure 10.9 is currently the smallest $31c/240$ orthogonal spaceship known in Life. About six years prior to its construction, two other spaceships of this same speed were built using most of the same reactions. These spaceships, called the **shield bug** and the **centipede**, are displayed in Figure 10.24. They were completed on the same day—September 4, 2014—by Dave Greene and Chris Cain, respectively, with help from Kiho Park, Ivan Fomichev and Adam P. Goucher.

When measuring size according to number of live cells, these spaceships are roughly 16 and 3 times as large as the silverfish, respectively. The reason that the silverfish could be so much smaller is that it uses a new front end (which was found by Chris Cain just 16 days after he completed the centipede) that is supported by just two heavyweight spaceships on each side instead of six.



(a) A $(13,1)c/31$ B-heptomino crawler.

(b) A $(27,1)c/72$ Herschel-pair crawler.

(c) A $(34,7)c/156$ Herschel crawler.

(d) A $(3,6)$ half-bakery crawler.

Figure 10.25: Some crawlers that could potentially be used to construct self-supporting spaceships.

Besides the $31c/240$, $17c/45$, and $(23,5)c/79$ reactions that we saw in this chapter, there are

Free PDF. See conwaylife.com/book for pattern files and print version.

numerous other known reactions that involve an object moving through a stable or glider-based wick—such reactions are called **crawlers** or **climbers**. Four crawlers are displayed in Figure 10.25, and each one of them could potentially be used to construct a self-supporting spaceship using the same techniques from this chapter.

Of these crawlers, the $(3,6)$ half-bakery reaction from Figure 10.25(d) is by far the easiest to work with, since a half-bakery is stable and thus the timing of the gliders is almost completely irrelevant. Indeed, this reaction has actually been used to create another self-supporting spaceship (essentially the only self-supporting spaceship other than the ones that we saw earlier in this chapter).

The way it works is based on the fact that two copies of this half-bakery reaction can be placed next to each other so as to produce a perpendicular glider (much like we placed two copies of the pi-crawler next to each other to create output gliders back in Figure 10.11(a)). We can thus use two gliders on a pair of half-bakery tracks to output sequences of gliders, and by using one more glider on a third parallel half-bakery track, we can reflect those gliders by 180 degrees. If we use even more parallel copies of these reactions (seven in total), we can have the triples of output gliders collide with each other in one of the tee reactions from Table 5.2, as illustrated in Figure 10.26.

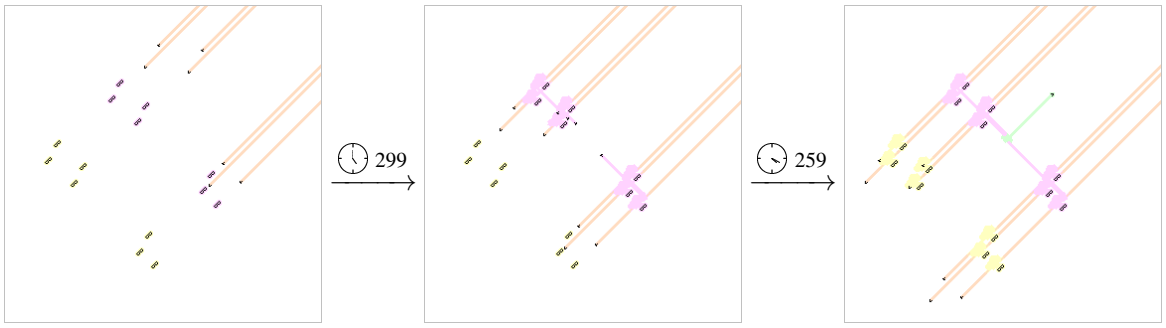


Figure 10.26: Some reactions in which gliders travelling southwest along half-bakery trails generate gliders travelling in any of the three other directions. In the northeast arrangement of half-bakeries (highlighted in magenta), sparks from the half-bakery crawlers collide with each other so as to generate gliders that then collide in a tee reaction (highlighted in green). In the southwest arrangement of half-bakeries (highlighted in yellow), the half bakeries are separated a bit more so that their sparks do not interfere with each other, thus producing no gliders.

These reactions give us the freedom to generate gliders travelling in any direction, and in particular they allow us to send gliders back to the start of the half-bakery trails. Those gliders can then use slow salvo synthesis to recreate the gliders that follow the half-bakery trails. In other words, we can use a handful of gliders travelling along half-bakery trails to implement armless construction (as introduced in Section 8.7) of that handful of gliders.

Spaceships that use these ideas based on the half-bakery crawler are called **half-baked knightships**, and the one displayed in Figure 10.27, which was constructed by Chris Cain in July 2014,[38] is called the **parallel half-baked knightship** (or **parallel HBK** for short).

The other three crawlers displayed in Figure 10.25 have not yet been turned into self-supporting spaceships. One of the main reasons for this is simply that constructing a self-supporting spaceship is a monumental task. However, they are also all slightly more difficult to work with than, for example, the $(23,5)c/79$ reaction from Figure 10.16 at the core of the waterbear:

- The $(13,1)c/31$ B-heptomino crawler from Figure 10.25(a) moves very fast: $13c/31 \approx 0.42c$ in one of the orthogonal directions, which is faster than any self-constructing spaceship that

[38] The first half-baked knightship was built by Adam P. Goucher just 5 days earlier, and was roughly 10 times as long, wide, and populous. Dave Greene and Ivan Fomichev helped come up with some of the component reactions and syntheses used in each of these knightships.

Free PDF. See conwaylife.com/book for pattern files and print version.

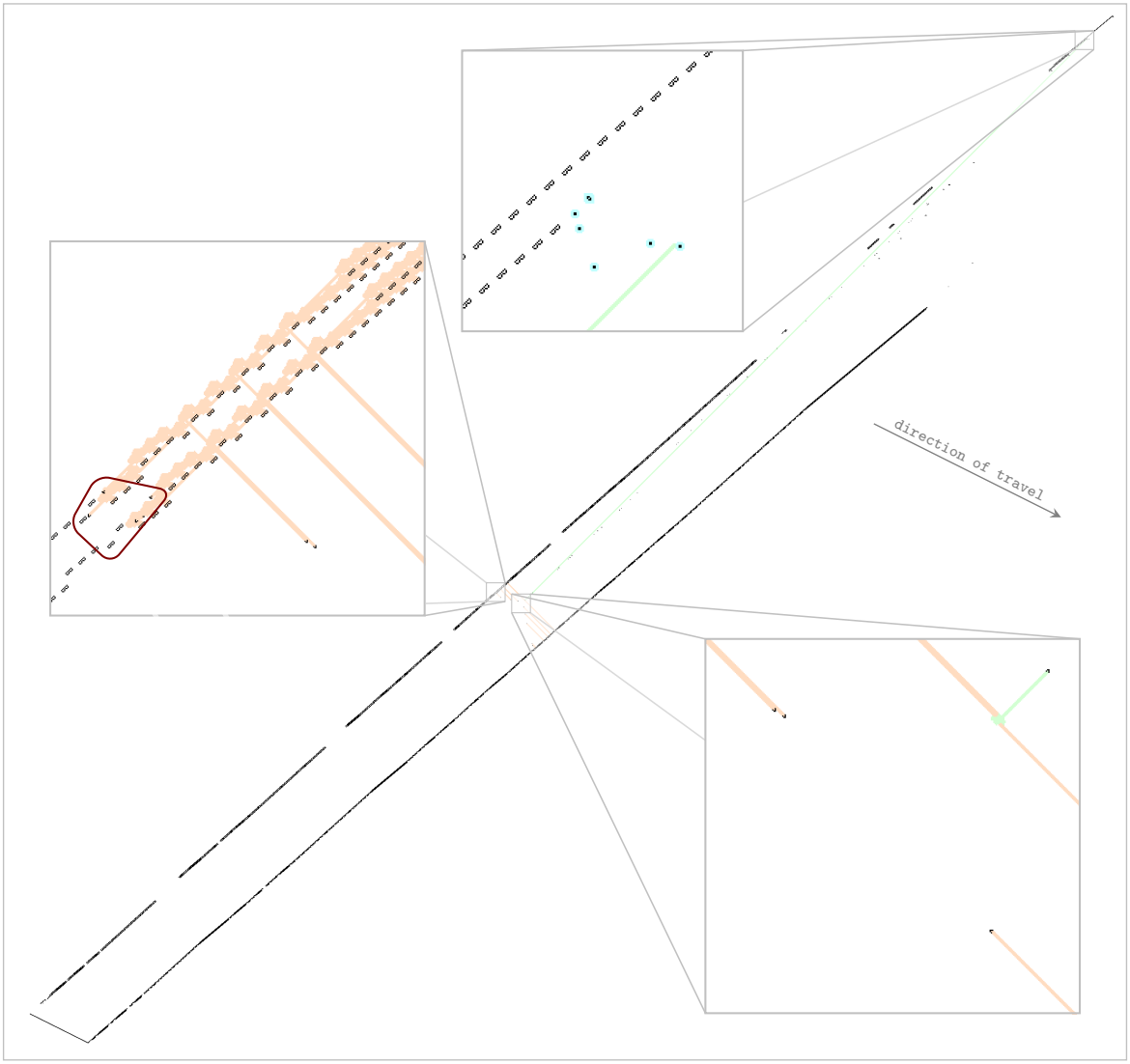


Figure 10.27: The **parallel half-baked knightship**. Its sides are tracks of half-bakeries that a total of 7 gliders follow along (4 on the northwest tracks, circled in red, and 3 on the southeast tracks). Those gliders move the half-bakeries by $(3,6)$ per period via the half-bakery crawler of Figure 10.25(d), while producing backward-firing gliders via the reactions of Figure 10.26 (highlighted in green). Those gliders go all the way back to the start of the half-bakery tracks and re-synthesize the original 7 gliders via a seed highlighted in aqua.

has ever been built as of this writing. As a result, all known helices that travel at this speed and slope are rather large, making any spaceship that would be built from this reaction also rather large (even by self-supporting spaceship standards). For example, any forward-firing helix of this speed and slope that is built out of the components from Figure 10.13 must contain at least 35 xWSSes (see Exercise 10.34).

- The $(27,1)c/72$ Herschel-pair crawler from Figure 10.25(b) does not produce an extra output glider that can be used to synthesize other components—only a pair of gliders that can be used by subsequent Herschel pairs. This problem can be fixed by placing multiple copies of this

Free PDF. See conwaylife.com/book for pattern files and print version.

crawler next to each other (just like we did with pi crawlers in Section 10.2.1),[39] but since it already runs on a two-glider track, these rakes quickly become very expensive and complicated to construct.

- The $(34,7)c/156$ Herschel crawler from Figure 10.25(c) produces some junk behind itself every period (two blocks and a beehive) that needs to be cleaned up. Fortunately, it also produces an extra output glider that can be used to do that cleanup (much like we did with $31c/240$ Herschel pairs in Figure 10.2). However, this again increases the complexity of the glider tracks (especially since it seems to require 4 tracks instead of just 2 to do all of the required cleanup), making the resulting spaceship much larger and more difficult to construct.

Exercises

solutions to starred exercises on page 466

10.1 [2/5] There are some non-highlighted cells near the center of the block-laying pattern displayed in Figure 10.4(a). What is their purpose?

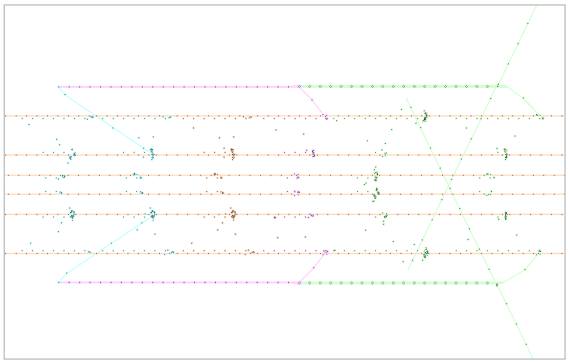
∗10.2 One way of creating rakes for Herschel crawlers (used by the silverfish) that output gliders on different lanes is to use a **double kickback** reaction: the two-glider kickback reaction twice. For example, the R6L17 rake from Figure 10.8(c) uses two double kickback reactions.

- **(a)** [2/5] Modify the rake from Figure 10.8(c) so as to use just *one* double kickback reaction.
- **(b)** [1/5] What is the name (in the R#L# format) of the rake that you created in part (a)?
- **(c)** [1/5] Explain why the rake that you constructed in part (a) is not useful when constructing the silverfish.

10.3 Two of the gliders from the slow salvo synthesis of Figure 10.7 are just used to destroy a beehive and a block.

- **(a)** [2/5] Adjust the synthesis so that those two gliders come from the northwest instead of the southwest.
- **(b)** [5/5] Construct a silverfish that is at least 100 rows shorter than the one from Figure 10.9 by using a backrake to produce one or both of the gliders that were repositioned in part (a).
  [Hint: A rephaser from near the front of the silverfish can be turned into a backrake whose stream crosses that of the forward rakes.]

10.4 [1/5] Another forward rake that crawls along 6 block tracks and could be used in the construction of the silverfish is displayed below.



- **(a)** What is the name (in the R#L# format) of this rake?
- **(b)** Explain why this rake is not useful when constructing the silverfish.

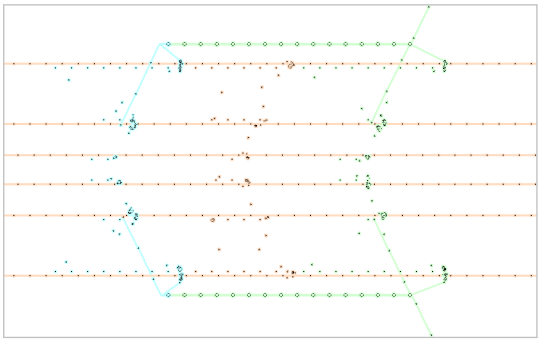
∗10.5 [3/5] In this exercise, we create another rake that crawls along 6 block tracks and could be used in the construction of the silverfish.

- **(a)** Adjust the positions of the two outermost Herschels in the rephaser from Figure 10.3(b) so that the gliders on the outside of the tracks do not annihilate each other. Doing so should create a forward-and-backward rake.
- **(b)** Place the forward rake R1L0 behind the double rake that you created in part (a) so as to eliminate its backward gliders.
- **(c)** What is the name (in the R#L# format) of the rake that you created in part (b)?
- **(d)** Explain why the rake that you constructed in part (b) is not useful when constructing the silverfish.
  [Hint: Be careful. The answer is slightly different than it was in Exercises 10.2 and 10.4.]

[39] This same problem also applies to the $(13,1)c/31$ B-heptomino crawler from Figure 10.25(a), but the solution is less expensive in that case since its base trail only involves a single glider.

Free PDF. See conwaylife.com/book for pattern files and print version.

10.6 Another forward rake that crawls along 6 block tracks and could be used in the construction of the silverfish is displayed below.



(a) [1/5] What is the name (in the R#L# format) of this rake?
(b) [2/5] Create an updated version of Table 10.1 that takes this rake into account. Which rake from Figure 10.8 is no longer useful in the construction of the silverfish?
(c) [3/5] On page 321, we gave a sequence of rakes that would fire 13 gliders as efficiently as possible using the rakes we had at that time. Update this sequence now that we have this new rake.
(d) [5/5] Reconstruct the silverfish according to the sequence of rakes that you compiled in part (c).
[Hint: Most of the silverfish from Figure 10.9 can be left as-is. All that needs to be changed is which rakes crawl along its central block tracks.]

∗**10.7** [3/5] On page 319 we listed a sequence of mod 31 lane numbers for implementing the slow salvo synthesis from Figure 10.7. The list of lanes that can be used is actually much more flexible than indicated there. For example, we can fire the first two gliders on lanes 0 and 11 in either order, since the first two gliders in that slow salvo do not interact with each other. Provide a complete description of *all* possible sequences of mod 31 lane numbers that implement that synthesis.

10.8 [2/5] Break down the forward rake from Figure 10.11(b), which is made up of six pi crawlers, into separate non-interacting pieces that each consist of fewer pi crawlers.

10.9 Use a backward and forward rake on a pair of blinker tracks (as in Figure 10.12), along with some rephasers, to synthesize a parallel track of...

(a) [2/5] blocks,
(b) [2/5] eater 1s, and
(c) [3/5] gliders, with the synthesized gliders coming back and destroying one of the blinker tracks.

∗**10.10** [4/5] In this exercise, we investigate how to synthesize blinker tracks like in Figure 10.12, but at any location and spacing of our choosing.

(a) If you move the forward rake back by 34 generations, how many cells farther away is the blinker track synthesized?
(b) Insert two additional pi rephasers on each blinker track between the backward and forward rakes, and then adjust the positioning of the rakes so that they again synthesize a blinker track. How much farther away is this blinker track than the one that is synthesized in Figure 10.12?
(c) Your solutions to parts (a) and (b) should be relatively prime. Explain why this means that we can synthesize a pair of blinker tracks with any spacing of our choosing (as long as they are far enough apart that their syntheses do not interfere with each other).
(d) Create an arrangement of rakes and rephasers that synthesize a pair of blinker tracks that are 32 cells from each other (and could thus be used by additional rakes).

∗**10.11** [3/5] Recall that the values of m and n from Equation (10.1) are typically not the smallest positive integers for which

$$\frac{(20m+20)c}{40m+n+101} = \frac{p}{q}.$$

However, they are *sometimes* the smallest such positive integers. Give an example (i.e., particular positive integers p and q with $p/q < 1/2$) for which these values of m and n are minimal.

10.12 Construct an orthogonal helix that fires gliders forward on one side and travels at...

(a) [2/5] $120c/301$,
(b) [3/5] $12c/31$, and
(c) [4/5] $124c/323$.

∗**10.13** [2/5] Compute the slopes of the lines of forward and backward gliders in the caterpillar (i.e., in Figure 10.15). Do not just read the slopes off of that figure – explain where they come from.

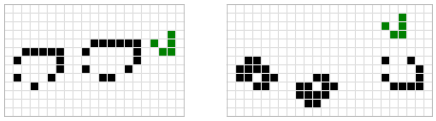
10.14 [2/5] In Figure 10.15, we highlighted in yellow a two-MWSS flotilla that can be used to duplicate a glider. Find at least 2 other xWSS flotillae that are used to duplicate gliders in the caterpillar, and explain why it is useful to have so many different flotillae that seem to perform the same task.

10.15 Construct an oblique helix that fires gliders forward on one side and travels at...

(a) [2/5] $(70,13)c/201$,
(b) [3/5] $(40,13)c/111$, and
(c) [4/5] $(13,2)c/35$.

Free PDF. See conwaylife.com/book for pattern files and print version.

10.16 [2/5] Two additional helix components are displayed below—the one on the left is a reflector, while the one on the right is a duplicator that fires a glider backwards.



(a) What useful feature does the reflector have that makes it more useful in some situations than the reflector from Figure 10.13(b)?
(b) Create a helix, of any speed of your choosing, that makes use of both of these components.

10.17 [4/5] Consider the two helix components from Exercise 10.16 and the glider-shifting component from Figure 10.13(a).

(a) Show that these three components can be used to construct orthogonal helices, which fire gliders backward on one side, with any rational speed slower than $c/2$.
(b) Show that these three components can be used to construct oblique helices, which fire gliders backward on one side, that travel at the same speed and direction as any oblique spaceship.

∗**10.18** [2/5] In the waterbear, what is the slope of the line of Herschels and gliders along its northeast edge (i.e., what is the slope of the magenta line in Figure 10.18)? In particular, explain why the slope is *not* $5/23$.

∗**10.19** [2/5] The helix used by the waterbear contains a 3-xWSS reflector. Find that reflector, and explain why it is used instead of either of the 2-xWSS reflectors that we saw in Figure 10.13(b) and Exercise 10.16.

∗**10.20** [3/5] Directly beside its 6-xWSS helix, the waterbear has a pair of xWSSes that convert the helix's backward glider into a backwards glider and a loaf. Explain why—what purpose does this loaf serve?

10.21 [2/5] Recall the loaf-moving xWSS flotillae from Figures 10.19 and 10.21.

(a) Adjust the loaf-pusher so that it pushes the loaf forward by 60 cells.
(b) Adjust the loaf-puller so that it pulls the loaf backward by 60 cells.

10.22 [2/5] Recall the xWSS flotillae from Figure 10.22 that push or pull a loaf and then self-destruct with the help of another still life.

(a) Adjust the flotilla from Figure 10.22(a) so that it pushes the loaf and honey farm forward by 100 cells.
(b) Adjust the flotilla from Figure 10.22(b) so that it pulls the loaf and beehive backward by 100 cells.

∗**10.23** [3/5] The flotilla from Figure 10.22(a) that destroys the loaf-pusher also releases a backward glider, whereas the flotilla from Figure 10.22(a) that destroys the loaf-puller does not. Explain why.
[Hint: Look at the front and back ends of the caterloopillar from Figure 10.23.]

∗**10.24** [2/5] The xWSS flotillae displayed in Figure 10.22 contain two different 2-xWSS flotillae that can reflect a glider while firing one backward. Explain why the 3-xWSS component from Exercise 10.16 that performs this same task would often be preferable when building a backward-firing helix.

∗**10.25** [2/5] In the zoomed-in middle section of caterloopillar in Figure 10.23, there is a single LWSS that is not highlighted (i.e., it is not part of either of the loaf-moving flotillae). What is its purpose?

10.26 [3/5] What happens to the caterloopillar from Figure 10.23 if you adjust one or two of the flotillae at its front so that they are slightly closer together? In particular, does its speed increase, decrease, or stay the same?

10.27 [3/5] In a caterloopillar spaceship, the displacement per period, separation of the loaf-pushing flotillae, and separation of the loaf-pulling flotillae are not independent of each other: given two of these quantities, we can compute the third.

(a) If a caterloopillar pushes itself forward by 400 cells per period, and its loaf-pushing flotillae are separated by 2000 cells, how many cells must the loaf-pulling flotillae be separated by?
(b) If a caterloopillar's loaf-pushing flotillae are separated by 1800 cells, and its loaf-pulling flotillae are separated by 2400 cells, what is the displacement of the caterloopillar per period?

10.28 [3/5] Use the computer script from `conwaylife.com/forums/viewtopic.php?p=30104#p30104` to create a $c/21$ caterloopillar spaceship.

10.29 [2/5] Explain why the $(27,1)c/72$ Herschel-pair crawler from Figure 10.25(b) uses a *pair* of gliders and Herschels, rather than just a single Herschel on a single glider track.

Free PDF. See conwaylife.com/book for pattern files and print version.

∗**10.30** [3/5] Recall the $(34,7)c/156$ Herschel crawler from Figure 10.25(c), in which a Herschel travels along a trail of gliders.

- **(a)** Show how the trail of gliders can be replaced by a trail of blocks.
- **(b)** Give a reason that you think it might be easier to build a self-supporting spaceship out of the Herschel crawler on a block track, and another reason that you think it might be easier to build one out of the Herschel crawler on a glider track. Which of your reasons do you think is more compelling (i.e., which of these spaceships do you think would be easier to build)?

10.31 [4/5] In Figure 10.2, we placed two $31c/240$ Herschel crawlers next to each other so as to clean up their extra leftover debris (i.e., blocks). Now similarly place multiple $(34,7)c/156$ Herschel crawlers (on block tracks, as in Exercise 10.30) next to each other so as to clean up their extra leftover debris (i.e., blocks and beehives).

10.32 [4/5] In Section 10.2.1, we constructed forward and backward rakes out of pi crawlers by placing them next to each other so that their sparks synthesized gliders. Now similarly construct forward and backward rakes out of multiple copies of the $(27,1)c/72$ Herschel-pair crawler from Figure 10.25(b).

∗**10.33** [2/5] Suppose you were to create a self-supporting spaceship from the three crawlers displayed in Figures 10.25(a–c). Which of those spaceships would require a helix?

∗**10.34** [4/5] Consider the problem of constructing a helix that travels at $(13,1)c/31$ (i.e., the same speed as the B-heptomino crawler from Figure 10.25(a)).

- **(a)** Show that any such helix consisting only of the three components from Figure 10.13 must contain at least 33 xWSSes.
  [Side note: No other known helix components, like the ones from Exercise 10.16, can be used to bring this number down.]
  [Hint: What value of $k+m$ is required for the speed (10.2) to be at least as fast as $(13,1)c/31$?]
- **(b)** Show that any such helix consisting only of those three components must actually contain at least 35 xWSSes.
- **(c)** Construct such a helix consisting only of those three components that contains fewer than 75 xWSSes.

10.35 [3/5] In Exercise 10.34, we constructed a helix that fires gliders forward on one side and travels at the same speed as the $(13,1)c/31$ B-heptomino crawler from Figure 10.25(a). Now construct such a helix that travels at the same speed as...

- **(a)** the $(27,1)c/72$ Herschel-pair crawler from Figure 10.25(b), and
- **(b)** the $(34,7)c/156$ Herschel crawler from Figure 10.25(c).

10.36 [2/5] The parallel HBK from Figure 10.27 has speed $(6,3)c/245912$. Adjust it so that it travels more slowly. Specifically, adjust it so that it travels at a speed of exactly $(6,3)c/300000 = (2,1)c/100000$.

10.37 Alter the parallel HBK from Figure 10.27 so as to turn it into a rake that emits a single glider per period. Keeping the orientation of the parallel HBK as it is in that figure, make the output gliders travel...

- **(a)** [3/5] northwest,
- **(b)** [3/5] southeast,
- **(c)** [4/5] northeast, and
- **(d)** [4/5] southwest.
  [Hint: This last case is a bit different from the previous ones, since the knightship already has seven gliders travelling southwest along the half-bakery trails. Instead of adding more glider-generating half bakeries, modify the knightship's cleanup mechanism so as to only destroy six of those seven gliders.]

Free PDF. See conwaylife.com/book for pattern files and print version.

Free PDF. See conwaylife.com/book for pattern files and print version.



11. Universal Construction

> The truth is you don't know what is going to happen tomorrow.
> Life is a crazy ride, and nothing is guaranteed.
>
> Eminem

In the previous chapter, we learned that we can often stabilize unstable reactions (like the $31c/240$ reaction of Figure 10.1, or the $17c/45$ reaction of Figure 4.55) by using them to create gliders, and then using those gliders to synthesize stabilizing components. In this chapter, we take this idea one step further and show that similar behavior can be achieved without the need for the initial unstable reaction—we can use gliders to create or move some component in the Life plane, while simultaneously moving or recreating those gliders so that they can be reused.

We refer to these techniques as **universal construction**, and a pattern that implements them as a **universal constructor**, with "universal" referring to the fact that they can build any Life pattern that is synthesizable via gliders.[1] Although most universal constructors only fire slow glider salvos, we know from Theorem 5.1 that this is enough to implement arbitrary glider syntheses. Most useful universal constructors are built out of simple stable components like blocks, beehives, and eater 1s, so that it is relatively straightforward to synthesize them via gliders, and thus they can even be used to build copies of themselves.

The applications of universal construction that we will present in this chapter are a wide variety of spaceships with speeds and slopes that we have not yet seen. Indeed, we start off by first showing how to construct spaceships with any rational slope (but not necessarily any rational speed), and then we flip things around so as to construct spaceships with any rational speed (but fixed at a diagonal slope of 1).[2]

[1] Some patterns, like Gardens of Eden and the grandparentless pattern from Figure 1.42, cannot be synthesized by gliders.

[2] Well, any rational speed below $c/4$, which is the diagonal speed limit from Theorem 4.1.

Free PDF. See conwaylife.com/book for pattern files and print versio

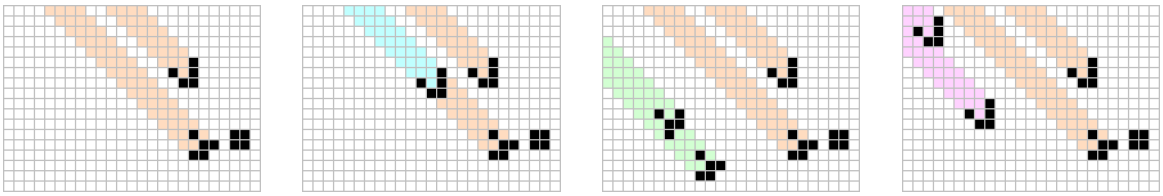
11.1 Gemini and Geminoids

We have seen the basic idea behind universal construction several times now, starting back in Section 8.6. By using a sliding block (or other small still life) as a construction elbow, we can fire gliders at any location in the Life plane, thus allowing a sequence of gliders to encode the construction or destruction of essentially any pattern (via the slow salvo techniques of Section 5.7, for example).

As our first illustration of the utility of universal computation, we now construct a spaceship called **Gemini** that is primarily made up of long sequences of gliders that encode its own construction. Much like the silverfish and caterpillar spaceships of Chapter 10, Gemini works by reaching out in front of itself to construct a track along which it can travel. This time, the track that it travels along is simply a series of Herschel tracks that manipulate the glider trails that then construct those Herschel tracks.

11.1.1 Elbow Operations

We already saw gliders salvos that push or pull a sliding block back in Figure 8.37, as well as salvos that can be used to fire perpendicular gliders of either color in Figure 8.38. These operations are the heart and core of the Gemini spaceship, and we summarize the ones that it makes use of in Figure 11.1.[3] Just these four operations are enough for us to be able to implement any unidirectional slow salvo of our choosing.[4]



(a) `PULL` **(b)** `PUSH` **(c)** `FIRE WHITE` **(d)** `FIRE BLACK`

Figure 11.1: A collection of reactions that can be used to fire perpendicular gliders (to the southwest) along any lane of our choosing. All four reactions use the same initial pair of gliders (highlighted in orange).

By using some Herschel circuitry to create these glider configurations, we can straightforwardly construct the **construction arm** displayed in Figure 11.2 that takes an input glider on one of four different lanes and performs the corresponding operation on the sliding block. This particular construction arm uses some slightly old conduits like the Callahan G-to-H (refer back to Figure 7.38) rather than more recent conduits for two reasons:

- It will be useful for us to have this component built entirely out of pieces like blocks and eater 1s (and unlike Snarks) that are "simple" to synthesize.
- The Gemini spaceship that we will construct out of this component was built before many of the more efficient conduits that we have explored. For example, the Gemini spaceship that we will soon present was built in 2010, whereas the Snark was found in 2013 and the syringe was found in 2015.

By making use of this construction arm, we can use slow salvo syntheses in which gliders come in from just one or two out of four specified lanes to simulate general slow salvo glider syntheses (in which the gliders can come from any lane). For example, by aiming the output gliders of two

[3] The names "`FIRE WHITE`" and "`FIRE BLACK`" used for two of these reactions just refer to the fact that they fire perpendicular gliders of opposite colors (the absolute color names do not matter).

[4] Strictly speaking, we just need the three `PULL`, `PUSH`, and `FIRE WHITE` operations, since monochromatic (i.e., single-color) slow glider salvos are universal (i.e., can implement any glider synthesis). However, they are somewhat less efficient than their polychromatic counterparts, so we do not make use of them.

Free PDF. See conwaylife.com/book for pattern files and print version.

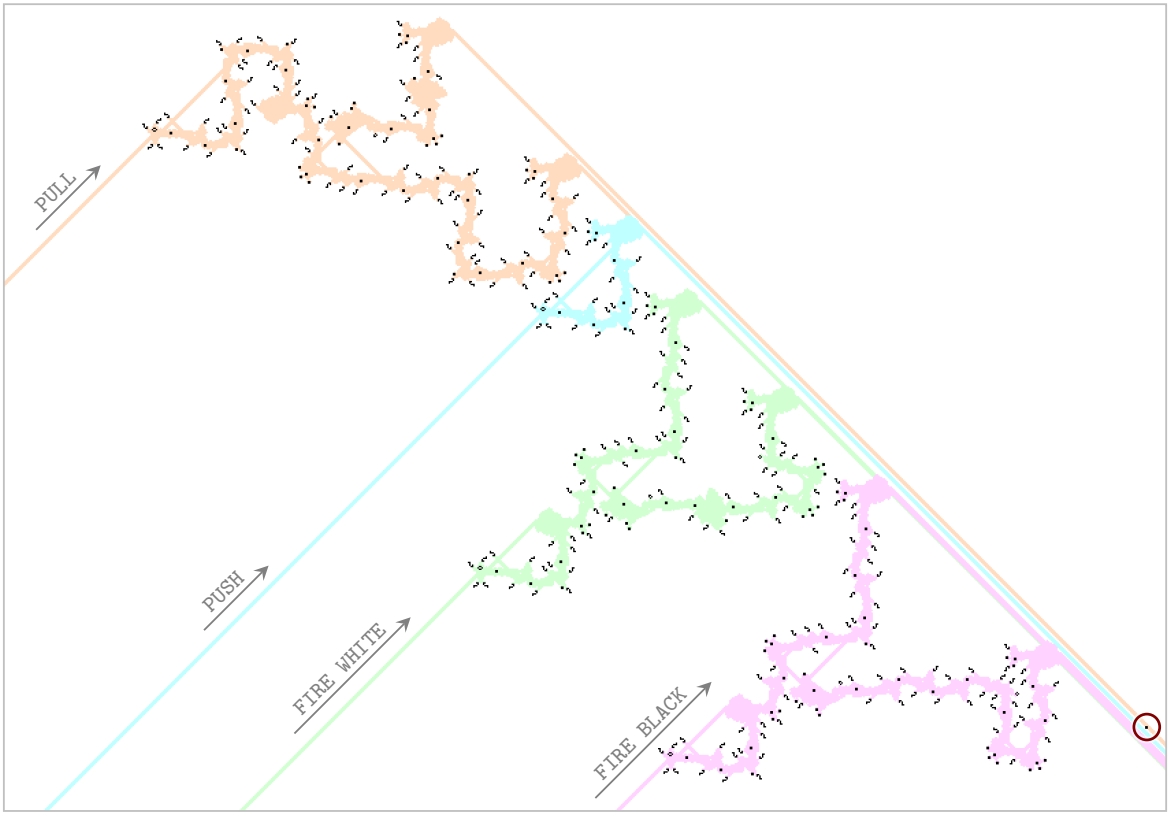


Figure 11.2: A **construction arm**, which takes a glider on one or two of four input glider lanes and as a result either pulls or pushes the sliding elbow block (circled in red), or uses it to fire a perpendicular glider of either color to the southwest. Note that all four actions require the PULL circuit to be activated, as it produces the frontmost pair of gliders in all four operations (see Exercise 11.2). Constructed by Paul Chapman and Dave Greene in 2004.

construction arms at each other, we can encode any two-direction slow salvo synthesis in the sequence of 4 input lanes that are used, much like we used armless construction to encode the synthesis of a 2-engine Cordership in some glider sequences back in Figure 8.46.

The pattern displayed in Figure 11.3 illustrates how this technique can be used to build a middleweight spaceship gun. While this gun is much larger and more complicated than other MWSS guns that we have seen, it has the advantage that the MWSS is encoded entirely in the input glider streams. To instead have this gun fire a different type of spaceship, or even produce essentially any arrangement of objects in the upper-central quarter of the Life plane, we do not need to change its circuitry at all—only the order in which slow gliders are fed into it (see Exercise 11.6).

11.1.2 The Gemini Itself

We can use arrangements of construction arms like the one in Figure 11.3 to construct anything that has a glider synthesis. In particular, because these construction arms are made up of such simple components (mostly blocks and eater 1s, with the most difficult piece to synthesize being a tub), they can even be used to synthesize copies of *themselves*. Indeed, this is the key idea behind the Gemini spaceship: use an extremely long chain of gliders to have two construction arms build copies of themselves somewhere else in the plane. To turn this idea into a *spaceship* though (i.e., to make the construction propagate from one location in the Life plane to another, without leaving anything behind itself), we need to make two additions:

Free PDF. See conwaylife.com/book for pattern files and print version.

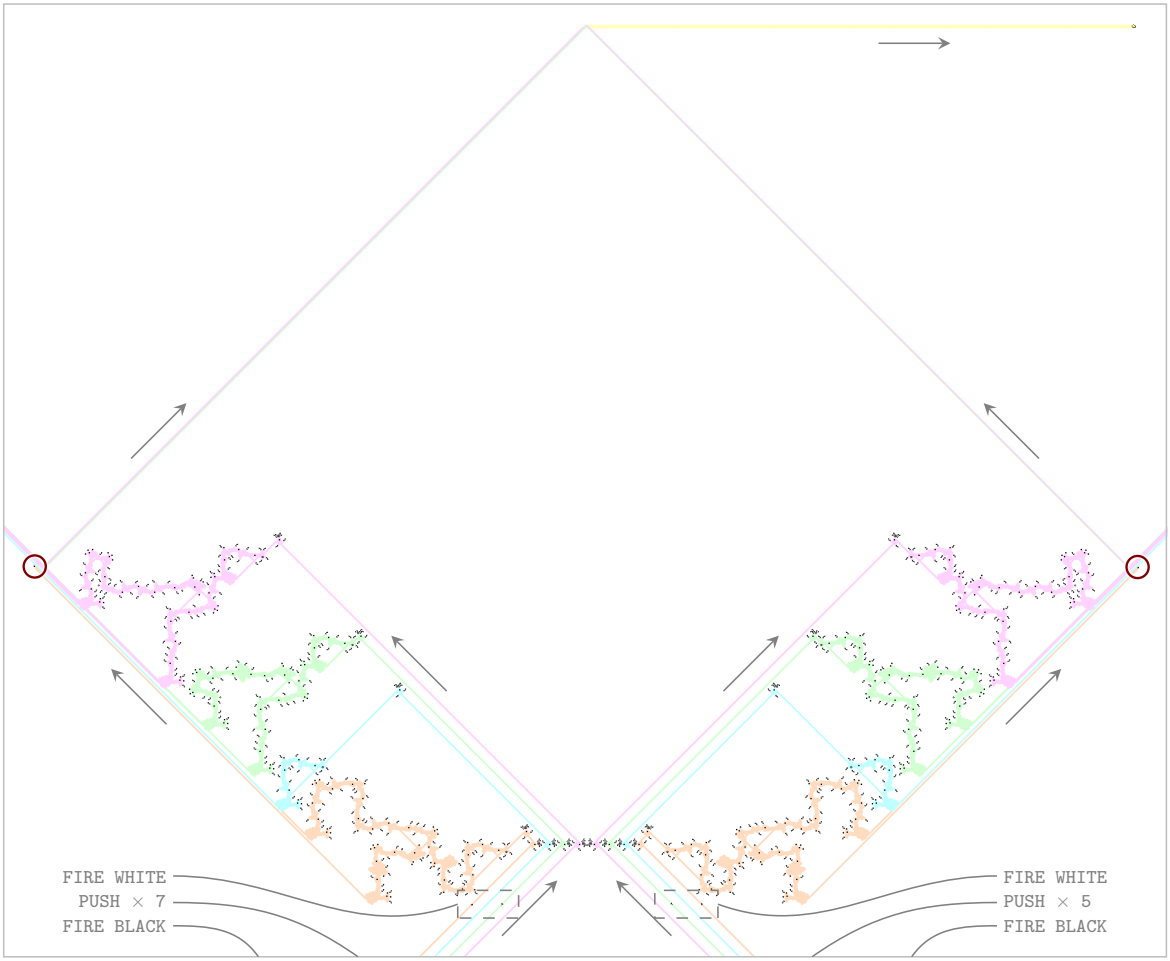


Figure 11.3: A pair of construction arms that are aimed at each other so they can implement two-direction slow salvo syntheses. Here, we first fire a glider from each of the construction arms so as to synthesize a blinker, and then we adjust the positions of the sliding blocks and fire another from each of the construction arms so as to turn that blinker into the middleweight spaceship highlighted in yellow (via the incremental synthesis of Table 5.4).

- We need to destroy the original pair of construction arms after they have constructed the new ones. To do this, we simply use a third construction arm (which we will instead call the **destruction arm**), which is constructed alongside the original pair. After all three arms are constructed, the first two start construction again while the third one starts destroying the no-longer-needed circuitry from the spaceship's previous period. Since objects like blocks and eater 1s can easily be destroyed by a single glider each, it is straightforward to find a glider recipe that destroys these old circuits.
- We need to be able to reuse the glider recipes that store the construction and destruction recipes. Since we cannot use a static loop (we need the glider recipes to move along with the construction arms), we instead have the glider recipes bounce back and forth between *two* copies of the entire three-arm circuit that we have described.[5]

After we put all of these ideas together, we get the Gemini spaceship that is displayed in Fig-

[5]In fact, the two ends of the spaceship that we construct will be exactly identical. This is the reason for its name "Gemini"—it is Latin for "twins".

Free PDF. See conwaylife.com/book for pattern files and print version.

ure 11.4.[6] It builds a copy of itself displaced by 5 120 cells in one direction and 1 024 cells in the other direction every 33 699 586 generations, giving its direction of travel a slope of $5120/1024 = 5$.

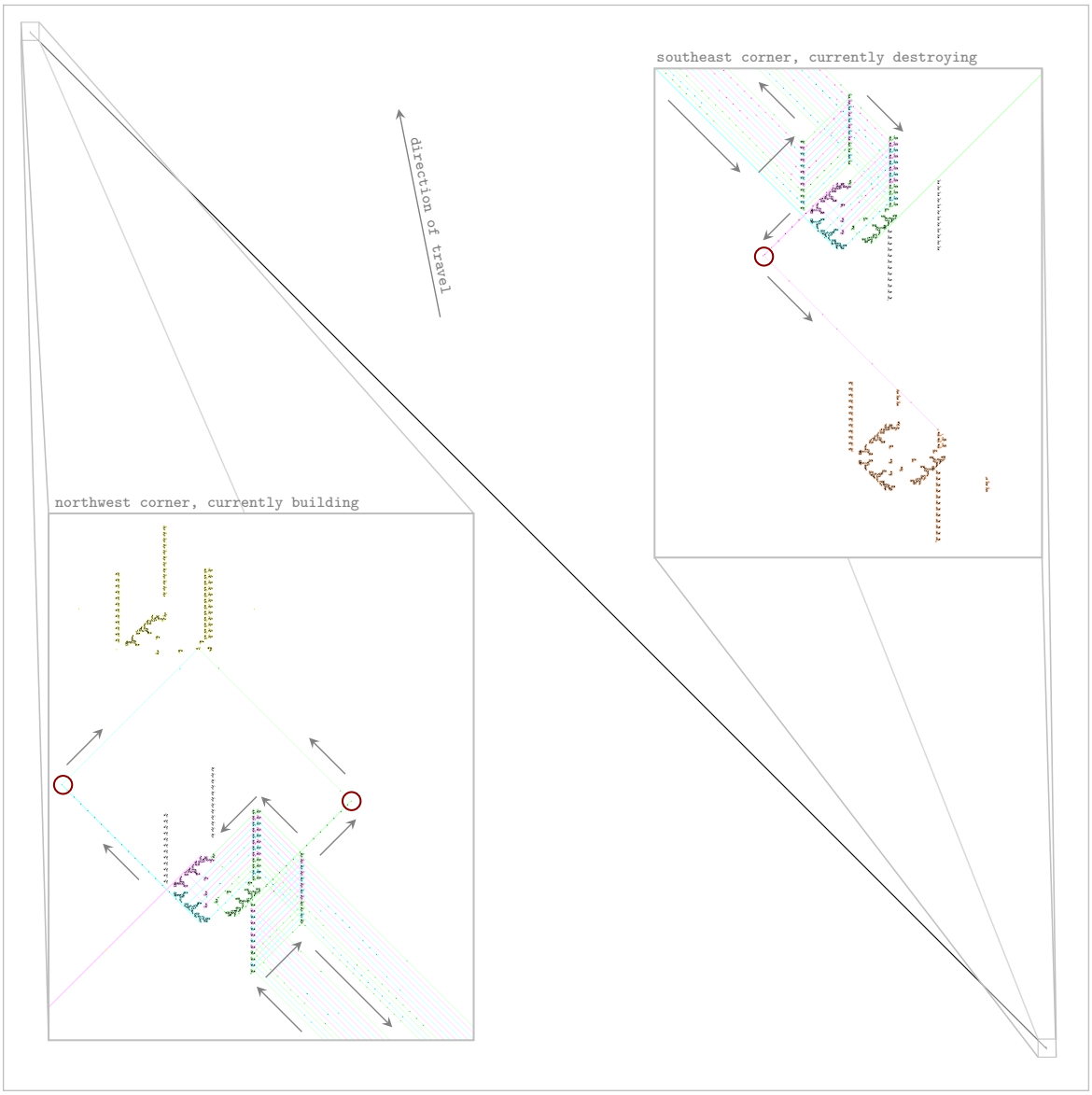


Figure 11.4: The self-constructing **Gemini** spaceship, which simply looks like a long, thin diagonal line when zoomed out far enough to see in its entirety. The bulk of the spaceship of made up of 24 parallel glider lanes (12 travelling in each direction), which carry a recipe for building the ship. The ends of the spaceship, which are zoomed in on here, are identical arrangements of stable glider reflectors, glider duplicators, and three construction/destruction arms whose sliding blocks are circled in red. The construction arms highlighted in aqua and green build the next copy of the spaceship (highlighted in yellow), while the destruction arm highlighted in magenta destroys the previous copy of the spaceship (highlighted in orange).

[6] Constructed by Andrew J. Wade in May 2010.

Free PDF. See conwaylife.com/book for pattern files and print version.

11.1.3 Geminoids

The Gemini spaceship travels at a speed of $(1024,5120)c/33699586$, but it can be modified in numerous ways to produce closely related spaceships, called **Geminoids**, that have a wide variety of different speeds and directions. We now explore some of these alterations, and discuss what their theoretical limits are.

The simplest way to change the speed of the Gemini is to change the distance between its two identical ends, without changing any of the glider recipes that bounce back and forth between them. Each additional cell (i.e., full diagonal) by which we separate the ends increases the travel time of each glider by 8 generations per period (4 generations in each direction). Since this change does not affect how far the Gemini travels over the course of its period, this lets us build Geminoids that are arbitrarily slow, having any speed of the form $(1024,5120)c/(33699586+8k)$, where $k \geq 0$ is an integer.[7] We can similarly squeeze the ends of the Gemini closer together, which *decreases* its period by 8 generation per full diagonal, but only by a bit. The glider sequence still has to fit between the two ends—if we squeeze them too close together then gliders will try to use circuitry before it is constructed (see Exercise 11.9).

We can also change the slope (i.e., the direction) that the Gemini travels at, just by changing how many PUSH operations are applied to the northwest construction elbow (i.e., the aqua elbow, in the orientation and coloring scheme of the Gemini displayed in Figure 11.4) before it starts constructing. In particular, if it is pushed out m cells before beginning construction, then the resulting displacement of the Geminoid will be $(m-2048,m+2048)$ cells per period.[8]

In the Gemini itself, the northwest construction elbow is pushed out by $m = 3072$ cells, for a displacement of $(m-2048,m+2048) = (1024,5120)$ cells per period, and a slope of $5120/1024 = 5$. We can change this into, for example, a knightship (i.e., a spaceship with slope 2) by increasing m so that $m+2048 = 2(m-2048)$ (i.e., $m = 6144$). The easiest way to do this is to insert another 3072 PUSH operations on the northwest construction elbow's path of the Gemini. This alteration produces the knightship Geminoid with $m = 3072+3072 = 6144$ and $n = 2048$ (i.e., a displacement of $(4096,8192)$ cells per period) that is displayed in Figure 11.5.[9]

We can achieve an even wider variety of Geminoid slopes by similarly pushing the northeast construction elbow (i.e., the green one in Figures 11.4 and 11.5) farther out before construction begins. We have to be slightly more careful with this elbow, though, for two reasons:

- To keep the two halves of the Geminoid lined up properly, we cannot just push the northeast construction elbow away by n cells—we also have to push the three rows of reflectors and glider duplicators on the northeast half of those ends away by $n/2$ cells.[10]
- The destruction elbow must be pushed out by the same amount as the northeast construction elbow: n cells. Furthermore, the destruction recipe itself must be changed slightly to accommodate the fact that the circuits that it is destroying have been separated by $n/2$ cells (see the above bullet).

If we (carefully, taking into account the technicalities mentioned above) push out the northwest and northeast construction elbows by m and n cells, respectively, the resulting Geminoid will have

[7] For some values of k, we will have to adjust the spacing of the gliders along the track. This is because one of the construction arms at the Gemini's southeast end crosses over the 24 glider lanes, which can lead to unwanted collisions if we are not careful. Any value of k that is a multiple of 144 is safe though, as that is the spacing between gliders on those lanes.

[8] When we talk about the northwest elbow being pushed by m cells before constructing, we really just mean that the displacement of the Geminoid over the course of one period is m full diagonals in the northwest direction (i.e., we are working in a coordinate system that is rotated 45 degrees from the usual one).

[9] This knightship was constructed by Dave Greene in June 2010.

[10] And n must therefore be even.

Free PDF. See conwaylife.com/book for pattern files and print version.

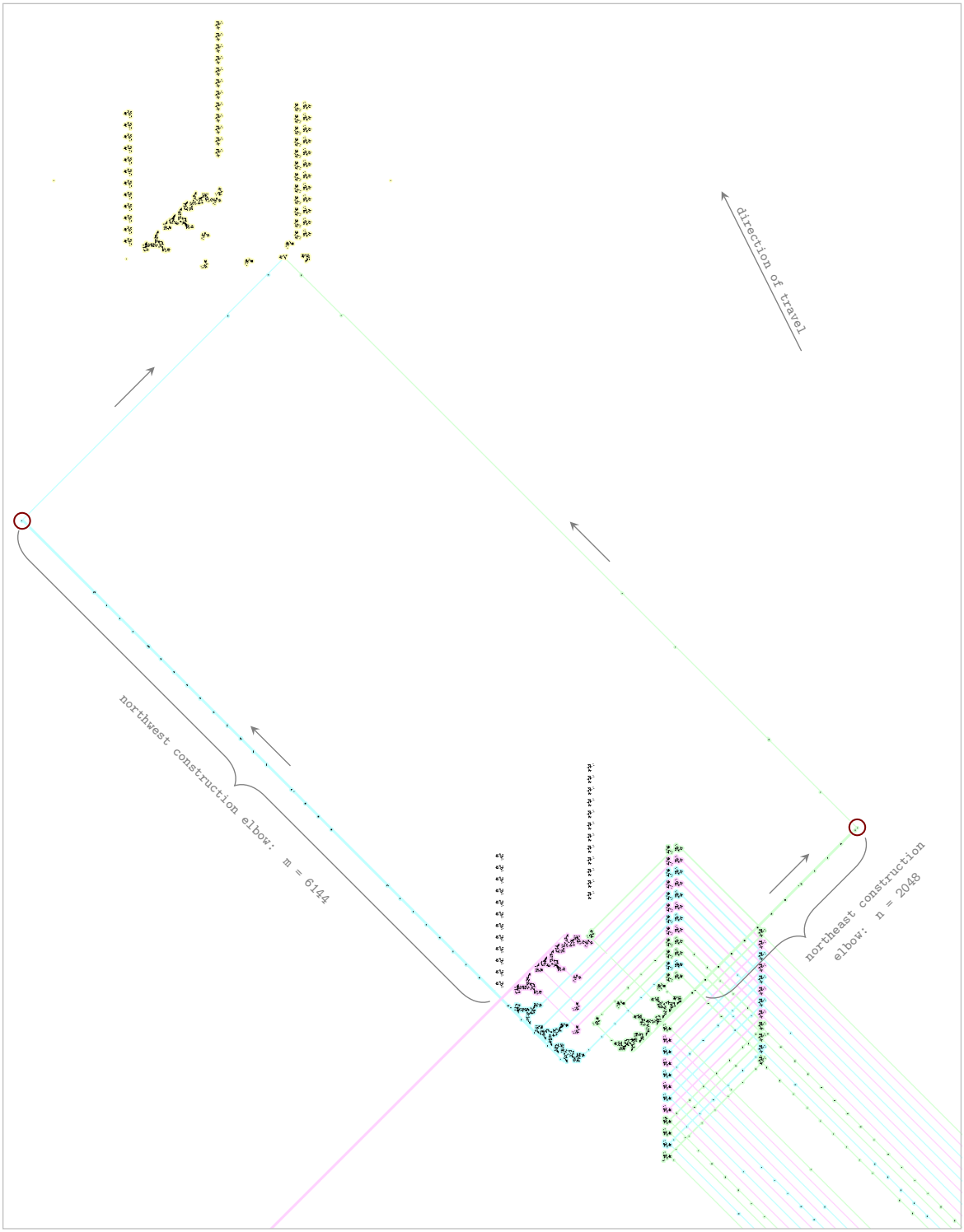


Figure 11.5: The northwest corner of a (slope 2) knightship Geminoid. It functions the exact same as the Gemini itself from Figure 11.4, except the construction elbow along the aqua path is pushed an extra 3072 full diagonals before beginning construction. It travels at a speed of $(4096, 8192)c/35567490$.

Free PDF. See conwaylife.com/book for pattern files and print version.

a displacement of $(m-n, m+n)$ cells per period. Its slope will thus be $(m+n)/(m-n)$,[11] so any rational slope $p/q \neq 1$ (i.e., diagonal) is attainable: we can get any rational slope p/q that is larger than 1 by choosing $m = 2p+2q$ and $n = 2p-2q$, so that

$$\frac{m+n}{m-n} = \frac{(2p+2q)+(2p-2q)}{(2p+2q)-(2p-2q)} = \frac{4p}{4q} = \frac{p}{q},$$

and we can achieve any rational slope that is between 0 and 1 by rotating one of these Geminoids by 90 degrees.

In addition to changing Gemini's slope, we can also use this method of inserting additional PUSH operations to increase its speed. For example, if we insert another 3072 PUSH operations on the Gemini's northwest construction elbow's path, and another 2048 PUSH operations on its northeast construction elbow's path (i.e., we increase the construction elbow displacements to $m = 6144$ and $n = 4096$), we get a Geminoid that travels exactly twice as far as the Gemini per period: $(2048, 10240)$. The exact speed of this new Geminoid depends a bit on some fine details like how much space we leave between the new PUSH-triggering gliders, and how tightly we push the ends of the Geminoid together, but it will be a bit under twice as fast as the original Gemini.

By adding more and more PUSH operations to each of the construction elbows, we can construct Geminoids that travel faster and faster. However, this speed increase is limited by the speed at which we can perform consecutive PUSH operations. Since the Gemini's circuitry contains a Callahan G-to-H (refer back to Figure 7.38), which has a repeat time of 575 generations, any PUSH-triggering gliders that we add to the Gemini must be separated by at least that much. Furthermore, since each PUSH operation pushes the elbow 1 full diagonal farther away, it adds an extra 4 generations to the Gemini's period, for a total of $575+4 = 579$ generations per PUSH. When we put all of these observations together, we arrive at the following theorem:

Theorem 11.1 — Geminoid Speed Limit
By changing the locations at which the Gemini spaceship constructs its circuitry (but without changing the components of that circuitry), we can construct Geminoid spaceships with any speed of the form $(x,y)c$, where $x, y < 1/579$ are rational numbers with $x \neq y$.

If we are willing to further tweak our Geminoids then we could make ones that are even faster, though the rebuilding effort would increase considerably. For example, we could replace the Callahan G-to-H that is used in the Gemini's construction arms with a Silver reflector, thus bringing its repeat time down to 497 generations, and its asymptotic speed limit up to $c/(497+4) = c/501$.

We could even go one step further and replace all reflectors that are used in the Gemini by Snarks, and all glider duplicators by syringe-based conduits (like the upcoming Scorbie splitter in Section 11.4.3). The resulting circuitry's repeat time would then be 90 generations, for an asymptotic speed limit of the Geminoid equal to $c/(90+4) = c/94$. However, a Geminoid with a speed anywhere close to this limit would have to be monstrously large due to the high number of PUSH operations used each period. Furthermore, it would require a complete rebuild of the Gemini, and extreme care would have to be taken to avoid unwanted collisions in the now more tightly packed crossing glider streams.[12]

[11] If $m = n$ then the Geminoid will travel straight up, orthogonally.

[12] These crossing glider streams make it rather fiddly to even reduce the spacing between gliders from 576 generations in the current Gemini to the theoretically possible 575 generations.

Free PDF. See conwaylife.com/book for pattern files and print version.

11.2 Single-Channel Glider Synthesis with a 90-Degree Elbow

While Geminoids let us construct spaceships that travel in any direction, they all travel quite slowly. We now switch gears and start developing the tools needed to construct a new type of spaceship that travels at any speed (but not any direction) of our choosing.

Recall from Section 5.7 that slow salvo synthesis is a method of encoding the construction of a pattern in the positions of a sequence gliders, with their timing playing no role (except that the gap between consecutive gliders must be sufficiently large that each glider's collision must settle down before the next glider arrives). We now flip this idea around and instead show how to encode the construction of a pattern in the *timing* of a sequence of gliders, with their *position* playing no role. In particular, we show how to encode glider synthesis via gliders that are all coming from the same direction on the same lane.

11.2.1 Creating Slow 90-Degree Gliders

As with many of our other universal constructions, we make use of a block that acts as an elbow in order to make these single-lane syntheses possible. In particular, we fire a stream of gliders at a block so as to first create a chaotic explosion (by turning the block into a pi-heptomino), and then clean up the resulting debris while creating a perpendicular glider. For example, Figure 11.6 displays a sequence of 13 gliders that, when it collides with a block, produces a single glider travelling in a different direction.

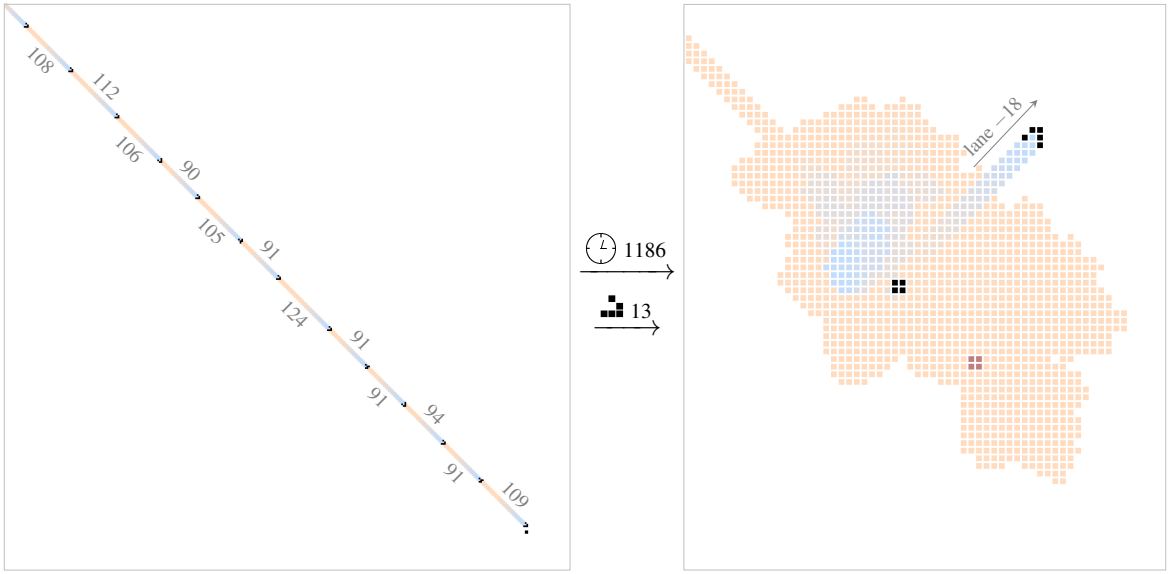


Figure 11.6: A sequence of 13 gliders travelling along the same lane that collide with a block so as to (a) produce a single perpendicular glider, and (b) recreate the target block along the same lane (but shifted northwest by 20 half-diagonals) so that it can be reused by other single-channel recipes. The numbers on the left indicate the number of generations between consecutive gliders.

Since all gliders in a single-channel recipe like this come from the same direction on the same lane, we can encode it simply via the number of generations between each consecutive pair of gliders. For example, the 13-glider recipe from Figure 11.6 could be encoded by the following sequence of 12 timing gaps:

`109, 91, 94, 91, 91, 124, 91, 105, 90, 106, 112, 108.`

Free PDF. See conwaylife.com/book for pattern files and print version.

Computer searches have been carried out[13] to generate single-channel glider recipes that, when aimed at a block as in Figure 11.6 (i.e., so that the first glider triggers a block-to-pi-heptomino explosion), a single output glider is produced on some perpendicular lane. A summary of some reasonably short recipes of this type that produce gliders on a variety of different output lanes (and in either of the two perpendicular output directions) is provided in Table 11.1.

Lane	Move	Timings
-18i	-20	109,91,94,91,91,124,91,105,90,106,112,108,(90)
-15i	-4	109,90,93,91,91,92,90,90,91,90,110,90,90,91,124,133,90,91,113,90,90,(90)
-7i	-25	109,91,94,91,91,136,91,90,91,104,90,90,90,110,90,90,98,(90)
1i	2	109,91,94,91,90,96,90,91,146,240,109,91,94,91,91,92,90,143,90,91,158,(90)
3i	-27	109,90,93,91,91,142,90,109,91,92,90,92,90,118,91,91,90,90,119,(90)
5i	8	109,91,94,91,91,136,91,90,91,139,98,90,156,(133)
8i	-8	109,90,95,245,90,126,208,128,90,96,91,90,90,91,91,91,91,100,90,90,(90)
19i	15	109,91,94,91,91,92,90,97,91,90,91,90,149,90,98,91,90,95,(90)
-21x	-37	109,91,93,90,155,106,90,90,92,91,109,90,93,91,90,100,124,(90)
-15x	-26	109,91,93,91,127,91,90,113,90,90,111,90,111,91,91,91,(90)
-11x	8	109,91,94,91,91,136,91,90,91,101,90,90,90,92,90,144,90,91,90,90,126,(90)
-9x	16	109,91,94,91,91,136,91,90,91,168,90,90,97,91,91,91,90,116,90,90,90,90,90,(90)
-3x	-15	109,91,93,91,97,91,91,106,91,90,90,90,90,90,91,163,90,90,104,(90)
-1x	-5	109,91,94,91,91,136,91,90,91,139,98,90,94,90,95,90,91,118,207,93,(90)
2x	3	109,91,94,91,91,92,90,143,90,91,158,(90)
10x	11	109,91,94,91,91,92,90,119,90,90,109,91,94,91,91,92,90,143,90,91,158,(90)

Table 11.1: Single-channel glider recipes that collide with a 90-degree elbow so as to produce a perpendicular output glider on a given lane. The first 8 rows give recipes that produce an "**i**nternal" glider (i.e., one that travels northeast when oriented as in Figure 11.6) and the last 8 rows give sequences that produce an "e**x**ternal" glider (i.e., one that travels southwest). The "move" column indicates how many half-diagonals the elbow is moved forward, and the "timings" column indicates the number of generations between consecutive gliders in the sequence. The final number in parentheses is the number of generations that must pass before it is safe to send subsequent glider recipes.

There are numerous points of clarification that we need to make about these recipes:

- The output lane number has either an "`i`" or "`x`" suffix, indicating in which of the two perpendicular directions the output glider travels. The suffixes stand for **i**nternal and e**x**ternal, which refer to the output glider travelling on the same side as the elbow block that the input glider sequence hits, or the opposite side, respectively.
- Some of these glider sequences, like the one with output on lane 2x, change the zero-degree elbow block into another small object like a boat, beehive, or pond. This is okay because those objects explode into a pi-heptomino when they are hit by a glider in the exact same way that a block does, and this is always the reaction that the first glider in these single-channel sequence triggers.
- The "move" column of Table 11.1 specifies how many half-diagonals farther away the zero-degree elbow is moved by the glider sequence. If this value is odd then the elbow is moved to the other side of the input glider stream. If this happens, the output lanes of all subsequent glider sequences are flipped from `i` to `x`, and vice-versa. For example, the sequence corresponding to

[13] Mostly by Simon Ekström, with some optimizations by Dave Greene, in 2017.

Free PDF. See conwaylife.com/book for pattern files and print version.

lane $2x$ has a "move" value of 3, which is odd. Thus if we want to output gliders on lanes $2x$ and then $-12x$, we would have to send the sequences corresponding to lanes $2x$ and then $-15i$ (not $-15x$).

- In most of these glider sequences, there are numerous gaps of 90 and 91 generations. The reason for this is that we want to be able to feed these single-channel glider recipes through standard components like Snarks and the easy-to-synthesize Lx200-assisted syringe from Exercise 7.10 (which has a repeat time of 90 generations).[14] Most of the gliders that have a timing of 90 or 91 generations could actually be moved much closer (i.e., the previous glider's collision settles down before they arrive), but only if they are used in conjunction with faster-recovering conduits. The reason that some gliders use a gap of 91 generations instead of just 90 is that their timing matters mod 2, due to p2 components like blinkers in an intermediate reaction.

11.2.2 Moving the Elbow

The sequences given in Table 11.1 can only be used to fire gliders on a small selection of lanes (8 in each direction). Furthermore, these sequences move the elbow forward and backward by some amount that we do not control—it is dictated by the output lane on which we wish to fire a glider. It is thus important to be able to move the elbow however far forward or backward we like, *without* firing a glider, so that the glider-firing sequences then produce a glider where we actually want it. For example, the following pair of elbow-moving sequences, which are displayed in Figure 11.7, move the elbow forward by 8 half diagonals or backward by 12 half diagonals, respectively.

```
push 8hd: 109, 91, 94, 91, 91, 92, 90, 119, 90, (90),
pull 12hd: 109, 91, 93, 91, 137, 91, 91, 125, 172, 108, 90, 109, 91, 101, 120, 90, (90).
```

A more complete summary of elbow-moving sequences is given in Table 11.2, which tells us how to move the 90-degree elbow in either direction by any number of half-diagonals up to 10. To move the elbow by a larger amount, we can simply repeat these sequences.

With these block-moving sequences at our disposal, we are now able to use gliders on a single lane to fire slow perpendicular gliders on any sequence of lanes of our choosing: by repeating the sequences from Table 11.2 we can move the elbow to any lane that we like, and we can then use a sequence from Table 11.1 to actually fire a perpendicular glider.

11.2.3 Creating and Using a Hand

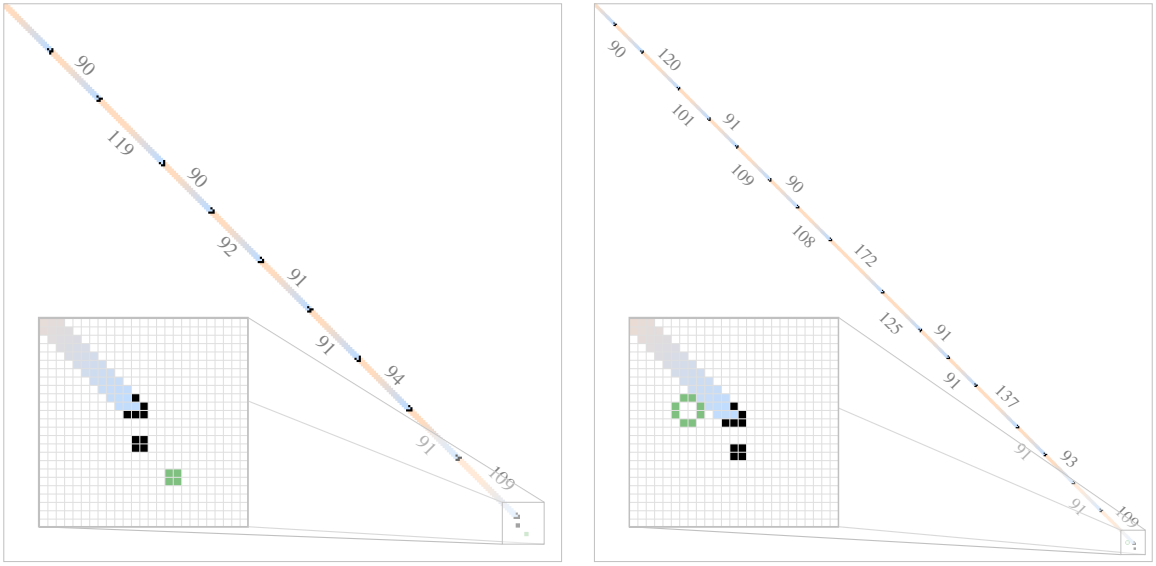
The only extra ingredient that we need in order to show that single-channel glider synthesis with a 90-degree elbow block is universal (i.e., can implement any glider synthesis) is a way of creating a target block (which we refer to as a **hand**[15]) that the perpendicular slow gliders will be fired at. One sequence of gliders that works to create such a hand block is

```
109, 91, 93, 90, 132, 115, 127, 91, 90, 91, 95, 90, 114,
        162, 233, 159, 90, 155, 126, 93, 118, 90, 91, 90, (90),
```
(11.1)

[14] Single-channel glider sequences with large timing gaps are desirable since they can make use of circuitry that has a higher repeat time. We could use sequences with gaps no smaller than 153 generations (such sequences are known to be able to emulate arbitrary glider syntheses), thus letting us feed these sequences through the syringe variant from Figure 7.8(b), for example. However, as the minimum timing gap increases, so does the length of the resulting glider sequences, so we stick with a minimum timing gap of 90 generations.

[15] It is connected to an elbow, after all.

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) push 8 half diagonals **(b)** pull 12 half diagonals

Figure 11.7: Single-channel glider sequences that (a) push or (b) pull an elbow. The location of the elbow after being pushed or pulled is marked in green.

which is illustrated as the first transition in Figure 11.8. Indeed, this recipe creates a hand block 86 half-diagonals away, on the internal side of the elbow block, while pushing that elbow block away by 17 half diagonals.[16]

Since we can now use single-channel glider sequences and a 90-degree elbow to create slow gliders on any lane, and we can also create a hand block for those gliders to collide with, we immediately see from Theorem 5.2 that these types of glider syntheses are universal. That is, we have demonstrated the following result:

Theorem 11.2 — Universality of Single-Channel Synthesis
Every pattern that can be constructed via glider synthesis can be constructed by a single-channel glider synthesis with a 90-degree elbow.

To illustrate how to implement slow salvo synthesis via single-channel synthesis, consider the problem of moving the hand block after we create it. We can use any of the slow salvos from Figures 5.21 and 5.22 to carry out this task—we will implement the 4-glider $(11,0)$ block pull from Figure 5.22(c), which uses gliders on lanes (relative to the initial position of the elbow block) $-6i$, $-6i$, $6i$, and $0i$, in that order.

To this end, we first use the hand-making sequence of gliders (11.1), which moves the block to position 17. We then fire the four slow gliders as follows:

- We want to fire a glider $17-(-6)=23$ lanes behind the elbow. Since the hand-making sequence of gliders flipped the orientation of the elbow block, we thus want to fire the first glider on lane $-23x$ (not $-23i$). This can be achieved by a -2 half-diagonal elbow move (from Table 11.2) followed by firing a glider on lane $-21x$ (from Table 11.1).
- Since firing a glider on lane $-21x$ moved the elbow by -37 half-diagonals (which is odd, so the elbow's orientation is flipped back to where it started), the elbow is now on lane

[16] In particular, since we have described the elbow block as moving by an odd number of half-diagonals, we know that it moves to the opposite side of the single-lane glider sequence.

Free PDF. See conwaylife.com/book for pattern files and print version.

Move	Timings
-10	109,91,94,91,91,96,90,97,91,91,130,94,90,105,90,95,111,(90)
-9	109,91,94,91,91,179,90,91,94,91,102,91,105,91,108,90,91,91,120,90,(90)
-8	109,90,93,91,90,95,91,90,91,91,90,90,91,90,90,99,90,90,91,90,94,90,(90)
-7	109,91,93,91,118,90,91,91,91,90,90,156,114,90,90,90,90,141,(90)
-6	109,91,93,91,156,91,91,94,90,91,140,91,103,91,91,132,(90)
-5	109,91,94,91,91,92,90,173,100,90,141,91,90,90,90,147,90,117,(192)
-4	109,90,93,91,91,90,90,92,90,95,91,170,90,90,91,91,98,91,91,(90)
-3	109,91,94,91,90,96,90,91,92,90,217,90,103,(90)
-2	109,91,94,91,91,136,90,90,91,171,100,118,90,(90)
-1	109,91,94,91,90,96,90,91,146,(240)
1	109,91,94,91,91,92,90,143,90,91,156,90,104,(164)
2	109,90,93,91,91,90,90,91,91,90,90,91,90,90,94,90,(90)
3	109,91,94,91,91,92,90,143,90,90,90,129,101,102,(90)
4	109,91,93,91,92,90,90,90,151,93,90,143,134,94,90,90,90,109,91,(90)
5	109,91,93,91,92,90,110,90,152,90,90,91,91,91,90,90,90,90,175,119,115,(193)
6	109,91,93,91,145,215,104,90,90,90,90,90,102,92,90,90,90,106,155,150,(90)
7	109,91,94,91,90,96,90,91,146,240,109,91,94,91,91,92,90,119,90,(90)
8	109,91,94,91,91,92,90,119,90,(90)
9	109,90,93,91,91,90,90,95,90,91,90,90,140,90,90,128,(93)
10	109,91,93,91,129,149,91,90,90,105,90,90,90,90,114,91,100,90,(90)

Table 11.2: Single-channel glider sequences that can be used to move the 90-degree elbow to any location along the input glider lane. The "move" and "timings" columns are as in Table 11.1.

$17-2-37=-22$. We want to fire a glider $-22-(-6)=-16$ lanes behind the elbow, which is lane $16i$. This can be achieved by pushing the elbow forward by 8hd and then firing on lane $8i$.

- Since firing on lane $8i$ moved the elbow by -8hd, it is now on lane $-22+8-8=-22$. We want to fire a glider $-22-6=-28$ lanes behind the elbow, which is lane $28i$. This can be achieved by pushing the elbow forward by 10hd, pushing it by 10hd again, and then firing on lane $8i$.[17]
- Finally, since firing on lane $8i$ moved the elbow backward by 8hd, it is now on lane $-22+10+10-8=-10$. We want to fire a glider $-10-0=-10$ lanes behind the elbow, which is lane $10i$. This can be achieved by a 2hd elbow move followed by firing a glider on lane $8i$.

When we put all of this together, we get the following single-lane sequence of 185 gliders (which we illustrate in Figure 11.8 and specify by 184 timings between those gliders) that first creates a hand block and then moves that hand block 11 cells via a sequence of 4 perpendicular slow gliders:[18]

```
make hand: 109, 91, 93, 90, 132, 115, 127, 91, 90, 91, 95, 90, 114, 162, 233, 159, 90, 155, 126, 93,
           118, 90, 91, 90, 90,
move -2hd: 109, 91, 94, 91, 91, 136, 90, 90, 91, 171, 100, 118, 90, 90,
```

[17] Be careful—we cannot just push the elbow forward by 9hd and then fire on lane $19i$, since the 9hd move would flip the orientation of the block, meaning you would have to fire on lane $19x$ instead.

[18] We did something slightly tricky with the two "move 10hd" sequences here—notice that the gap between their final pair of gliders are not the same as each other. See Exercise 11.15.

Free PDF. See conwaylife.com/book for pattern files and print version.

```
fire -21x: 109, 91, 93, 90, 155, 106, 90, 90, 92, 91, 109, 90, 93, 91, 90, 100, 124, 90,
move  8hd: 109, 91, 94, 91, 91, 92, 90, 119, 90, 90,
fire   8i: 109, 90, 95, 245, 90, 126, 208, 128, 90, 96, 91, 90, 90, 91, 91, 91, 91, 100, 90, 90, 90,
move 10hd: 109, 91, 93, 91, 129, 149, 91, 90, 90, 105, 90, 90, 90, 90, 114, 91, 100, 90, 90,
move 10hd: 109, 91, 93, 91, 129, 149, 91, 90, 90, 105, 90, 90, 90, 90, 114, 91, 100, 90, 91,
fire   8i: 109, 90, 95, 245, 90, 126, 208, 128, 90, 96, 91, 90, 90, 91, 91, 91, 91, 100, 90, 90, 90,
move  2hd: 109, 90, 93, 91, 91, 90, 90, 91, 91, 90, 90, 91, 90, 90, 94, 90, 90,
fire   8i: 109, 90, 95, 245, 90, 126, 208, 128, 90, 96, 91, 90, 90, 91, 91, 91, 91, 100, 90, 90.
```

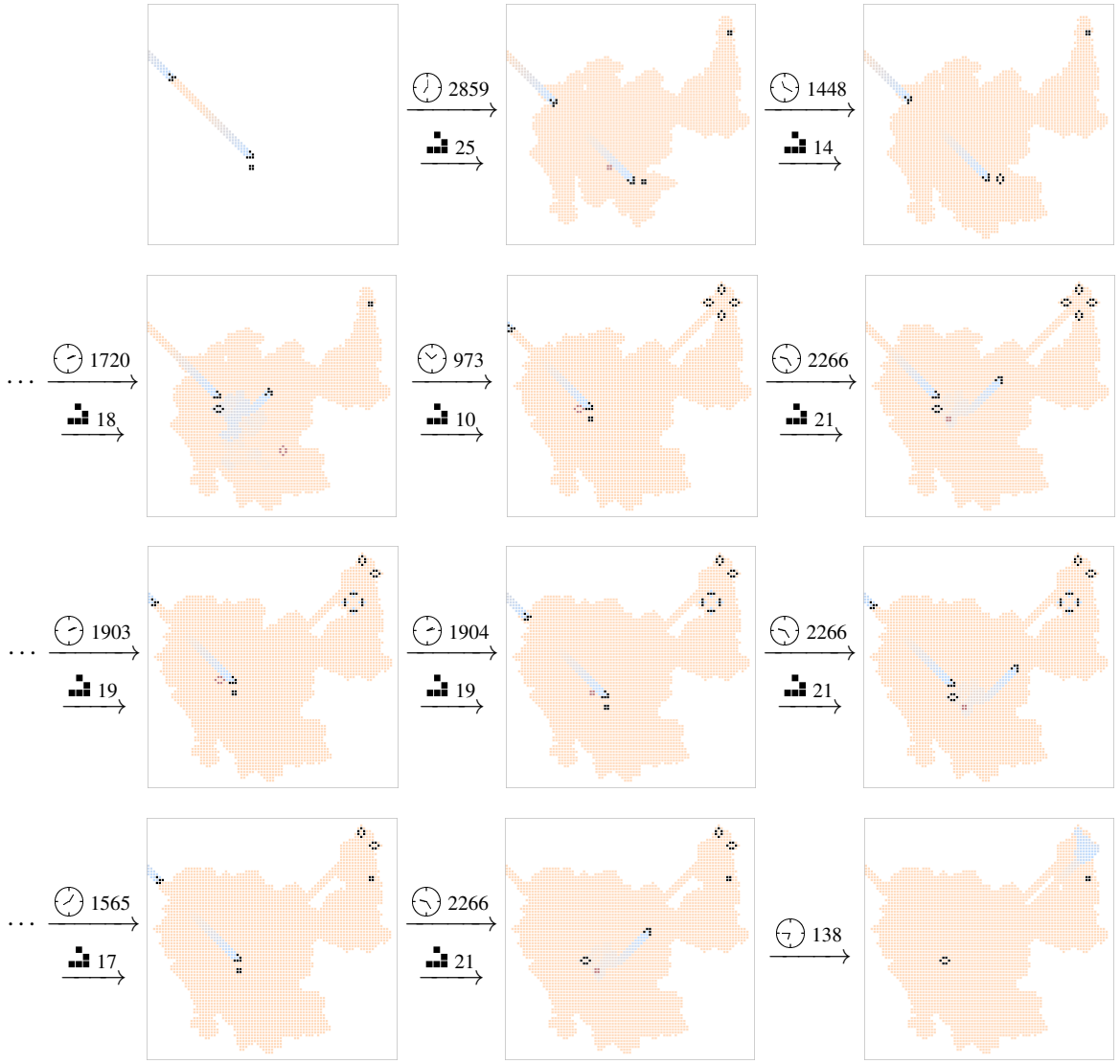


Figure 11.8: A single-lane sequence of 185 gliders creating a hand block and then firing 4 perpendicular slow gliders at that hand block so as to move it down by 11 cells. The perpendicular slow gliders are created on the correct lanes by repeatedly adjusting the position of the 90-degree elbow.

Free PDF. See conwaylife.com/book for pattern files and print version.

11.3 Single-Channel Glider Synthesis with a Zero-Degree Elbow

While the single-channel syntheses of the previous subsection are very useful, it is sometimes desirable to use single-channel glider recipes to build an object directly in front of them, rather than off to their side. For example, it will be useful for us to be able to use gliders from a single lane to construct a Snark that accepts gliders on that same lane as input. To make this possible, instead of using a 90-degree elbow we use a **zero-degree elbow**: a small stable object (like a block) that several gliders are fired at on a particular lane so as to produce a single glider going in the *same direction* but on a different lane.[19]

11.3.1 Creating Slow Zero-Degree Gliders

To illustrate how a zero-degree elbow works, consider the sequence of 14 gliders displayed in Figure 11.9 that, when it collides with a block, produces a single glider going in the same direction, but shifted by 8 lanes.

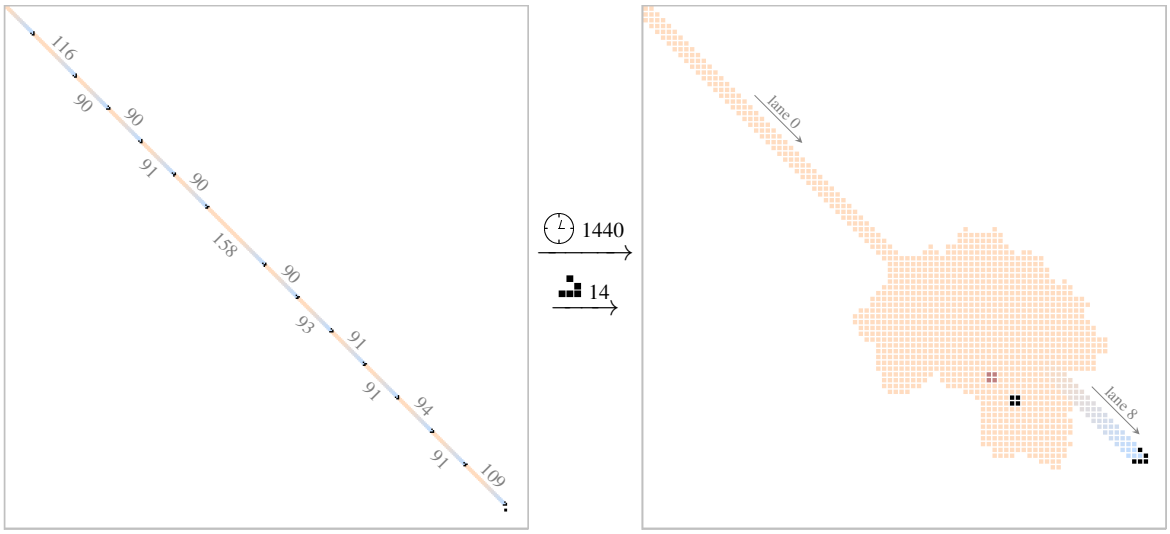


Figure 11.9: A sequence of 14 gliders travelling along the same lane that collide with a block so as to (a) produce a single glider travelling in the same direction, but shifted up by 8 lanes, and (b) recreate the target block along the same lane (but shifted southeast by 8 half-diagonals) so that it can be reused by other single-channel recipes.

As with 90-degree single-channel sequences, we encode zero-degree single-channel sequences simply by the number of generations between each consecutive pair of gliders. For example, we encode the 14-glider sequence from Figure 11.9 via the following sequence of 13 timing gaps:

`109, 91, 94, 91, 91, 93, 90, 158, 90, 91, 90, 90, 116, (104),`

with the final number in gray indicating how many generations must pass before another glider sequence can safely arrive.

In this zero-degree setting, it is much more important to be able to fire output gliders on a wide variety of different lanes than it was in the 90-degree setting. Indeed, when we worked with a 90-degree elbow we were able to simply move the elbow forward or backward to reach different output lanes, but doing so has no effect whatsoever on the output lane in the zero-degree setting.

[19] Here, "zero-degree" refers to the fact that the input glider stream is being "reflected" by zero degrees (i.e., not reflected at all, but just shifted to a different lane).

Free PDF. See conwaylife.com/book for pattern files and print version.

For this reason, extensive computer searches have been used to generate single-channel glider sequences of this type that produce a single output glider on any lane from -100 to $+100$.[20] A summary of sequences of this type for output lanes -22 through $+22$ is provided in Table 11.3.

As with 90-degree single-lane sequences, the value in the "move" column being odd means that the zero-degree elbow moves to the other side of the input glider stream. If this happens, the output lanes of all subsequent glider sequences are flipped from positive to negative, and vice-versa. For example, the sequence corresponding to lane 2 has a "move" value of -27, which is odd. If we want to output gliders on lanes 2 and then 3, we would thus have to send the sequences corresponding to lanes 2 and then -3 (not 3).[21]

11.3.2 Moving the Elbow

Unlike a 90-degree elbow, we typically do not care about the exact position of the zero-degree elbow.[22] Indeed, rather than having to align the elbow with a particular output lane, we just have to make sure that the elbow stays within an acceptable region—we do not want to move it so far forward that it crashes into whatever we are constructing, nor so far backward that it crashes into the source of the gliders. Fortunately, the elbow-moving sequences that we already saw in Table 11.2 (or even just the two sequences from Figure 11.7) can be used to adjust the positioning of the elbow if needed.

11.3.3 Creating and Using a Hand

Now that we can create output gliders on a wide variety of different lanes, we need to be able to create a target for those output gliders to hit. Once again, this target (typically a block) is called a **hand**, but this time we must build it in front of the elbow rather than off to its side.[23] One sequence that creates a suitable hand block is provided below and illustrated in Figure 11.10:

```
109, 90, 93, 91, 91, 90, 90, 100, 90, 90, 146, 96, 90, 90, 90, 92, 156, 144, (90).
```

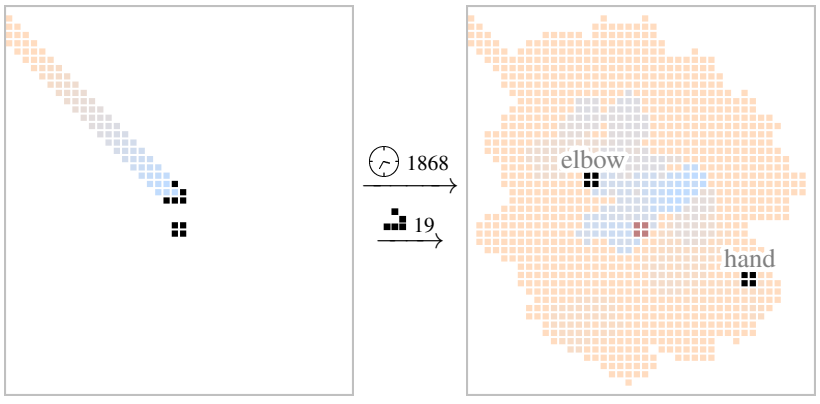


Figure 11.10: A single-lane sequence of 19 gliders creating a hand block for a zero-degree elbow.

[20] A complete collection of tens of thousands of these recipes can be downloaded from gitlab.com/apgoucher/slmake/-/tree/master/data/simeks. It can be useful to have multiple recipes for the same output lane, since they might move the zero-degree elbow by different amounts.

[21] Since the "move" value for the lane -3 sequence is also odd, the zero-degree elbow would then be back on its original side of the glider stream after this second sequence.

[22] This is one feature that makes zero-degree elbows easier to work with than 90-degree elbows.

[23] Technically, we *could* use the same hand block (11.1) that we used with the 90-degree elbow, since that hand is within the -100 to $+100$ lane range that we can fire output gliders on via the zero-degree elbow. However, it is much more convenient to use a more centered hand instead.

Free PDF. See conwaylife.com/book for pattern files and print version.

Lane	Move	Timings
-22	-20	109,91,94,91,91,136,90,90,91,114,90,91,90,99,90,97,90,90,90,126,90,137,90,(90)
-21	8	109,91,94,91,91,95,91,90,146,91,99,90,118,120,135,90,90,121,(90)
-20	-29	93,90,91,91,90,90,91,90,103,113,91,103,90,152,181,140,91,90,166,91,(106)
-19	-20	109,91,93,91,171,91,90,90,94,91,106,91,91,90,90,143,90,91,91,91,90,91,112,(90)
-18	8	109,90,93,91,91,128,91,91,90,97,90,99,90,139,91,91,117,134,92,90,90,90,(90)
-17	-22	109,91,93,91,123,90,129,90,90,111,142,91,90,120,91,142,(98)
-16	8	109,91,94,91,91,124,91,126,91,140,162,148,90,90,119,90,(90)
-15	13	109,91,93,90,156,91,91,94,91,90,147,117,91,144,90,91,128,100,91,90,105,91,(90)
-14	-39	109,91,93,91,155,106,91,91,96,90,90,91,108,90,156,90,90,120,90,112,91,99,(90)
-13	1	109,91,93,91,92,90,158,90,94,270,172,130,90,91,91,96,90,90,147,(90)
-12	-23	109,91,94,91,90,162,122,111,90,90,90,96,91,91,91,122,91,91,171,(90)
-11	-53	93,91,151,90,139,180,103,115,167,91,120,139,135,91,91,(169)
-10	15	109,91,93,91,97,91,90,91,120,90,95,91,143,90,90,90,90,(90)
-9	-15	109,91,94,91,91,90,91,91,90,158,90,91,90,90,101,90,107,90,90,90,(90)
-8	1	93,91,116,90,106,91,143,91,109,90,91,103,110,91,136,91,92,91,155,(199)
-7	-34	109,91,93,91,92,91,98,201,91,129,90,90,90,90,90,103,90,108,90,104,(90)
-6	-11	109,91,94,91,90,90,90,90,109,91,101,90,98,90,(90)
-5	-20	109,91,94,91,91,95,91,90,104,90,90,97,91,91,94,191,97,90,126,(90)
-4	-23	0,93,91,90,144,90,111,91,92,91,103,91,144,90,168,91,91,102,90,92,90,94,(90)
-3	-9	109,91,94,91,91,136,90,90,91,168,90,106,90,90,138,90,90,106,(90)
-2	7	109,91,93,91,92,91,90,90,162,91,91,90,129,91,113,90,90,90,(90)
-1	-9	109,90,95,245,90,95,90,123,91,90,115,142,(90)
0	-1	109,91,93,91,92,90,97,91,116,91,145,90,91,98,90,90,188,91,90,90,(90)
1	-24	109,91,94,91,91,93,90,95,90,113,90,99,90,156,90,90,90,138,(170)
2	-27	109,91,94,91,91,124,91,90,91,91,90,91,90,141,90,172,91,161,90,169,228,(90)
3	4	109,91,94,91,91,92,90,169,90,90,90,107,90,90,91,90,95,91,(90)
4	-56	109,91,94,91,91,92,90,169,91,90,116,90,161,91,104,(90)
5	-8	109,90,93,91,91,135,91,124,90,90,148,91,91,97,141,91,(90)
6	-15	93,90,90,90,91,90,91,136,155,98,120,90,90,91,92,90,97,161,161,(139)
7	-30	109,91,94,91,91,124,91,105,90,169,91,90,116,91,142,90,90,(90)
8	8	109,91,94,91,91,93,90,158,90,91,90,90,116,(104)
9	11	109,91,93,90,171,90,90,91,90,91,90,91,129,144,90,90,120,90,91,91,169,90,(90)
10	14	109,91,93,90,140,150,108,91,90,111,91,91,194,98,90,169,(90)
11	-5	109,91,94,91,91,92,90,146,90,90,90,91,135,91,152,(135)
12	8	91,94,91,91,121,90,90,90,90,90,90,99,90,165,119,90,106,90,90,(90)
13	-32	109,91,94,91,91,96,90,97,91,91,145,90,113,90,90,105,91,193,(90)
14	-16	109,91,93,91,129,149,91,90,90,142,219,90,99,91,109,115,92,185,(90)
15	9	109,90,93,91,91,158,94,113,91,90,91,96,90,142,(90)
16	8	109,91,94,91,91,95,91,90,93,218,172,90,90,90,116,112,341,107,106,90,163,91,(90)
17	-28	109,91,94,91,91,96,90,166,91,91,114,90,90,91,90,90,114,91,101,(90)
18	1	109,90,93,91,91,148,91,90,151,90,91,163,108,151,112,144,90,149,90,90,99,(90)
19	-19	109,91,93,91,115,107,90,90,90,90,90,90,90,103,99,118,91,130,(90)
20	12	109,91,93,90,169,90,91,103,91,133,90,90,91,91,90,110,91,93,90,112,171,(90)
21	-2	109,91,93,91,120,91,91,91,91,90,91,100,91,90,97,91,91,90,90,(160)
22	27	109,91,94,91,91,93,90,91,91,90,100,90,94,90,108,90,91,91,119,(90)

Table 11.3: Single-channel glider sequences that produce an output glider on a given lane (relative to the sequence itself, which is on lane 0). The "move" and "timings" columns are as in Table 11.1.

Free PDF. See conwaylife.com/book for pattern files and print version.

Since the hand that this glider sequence creates is fairly close to the zero-degree elbow, it is often a good idea to separate them a bit more by applying one of the elbow-pulling sequences that we have seen. We can use the 12hd pull from Figure 11.7(b), for example, repeatedly to create as large of a gap between the elbow and hand as we want.

With the glider sequences that we have now seen, it is straightforward to use a zero-degree elbow to synthesize any object that has a slow salvo synthesis no wider than 200 or so lanes.[24] For example, we implement the 4-glider $(11,0)$ block pull from Figure 5.22(c) via a zero-degree elbow much like we did for the 90-degree elbow. However, the details work out more simply since we do not have to worry about moving the elbow to a precise location after each slow glider is produced:

- We first use the hand-making sequence of gliders from Figure 11.10. We then want to fire gliders on lanes -1, -1, 11, and 5, in that order.
- To fire a glider on lane -1, we simply use the appropriate glider sequence from Table 11.3.
- To fire another glider on lane -1, we have to instead use the glider sequence corresponding to lane 1 (since the zero-degree elbow was moved to the other side of the glider sequence in the previous step, which has an odd "move" value of -9). However, before we can fire this glider, we have to pull the elbow block away from the hand block, or else they would collide with each other at this point. We thus apply the 12hd pull sequence from Figure 11.7(b) twice.
- To fire the third glider on lane 11, we use the glider sequence corresponding to lane -11 (since the elbow block is still on the side of the glider sequence opposite from where it started).
- Since the lane -11 sequence pulled the elbow block by 53 half-diagonals (which is odd), the zero-degree elbow has now been flipped back to its original side of the glider sequence. We can thus simply use the glider sequence corresponding to lane 5 at this point.

When we put all of this together, we get the following single-lane sequence of 118 gliders (which we illustrate in Figure 11.11 and specify by 117 timings between those gliders) that first creates a hand block and then moves that hand block 11 cells via a sequence of 4 parallel slow gliders:

```
make hand: 109, 90, 93, 91, 91, 90, 90, 100, 90, 90, 146, 96, 90, 90, 90, 92, 156, 144, 90,
 fire  -1: 109, 90, 95, 245, 90, 95, 90, 123, 91, 90, 115, 142, 90,
 move -12: 109, 91, 93, 91, 137, 91, 91, 125, 172, 108, 90, 109, 91, 101, 120, 90, 90,
 move -12: 109, 91, 93, 91, 137, 91, 91, 125, 172, 108, 90, 109, 91, 101, 120, 90, 90,
 fire   1: 109, 91, 94, 91, 91, 93, 90, 95, 90, 113, 90, 99, 90, 156, 90, 90, 90, 138, 170,
 fire -11: 93, 91, 151, 90, 139, 180, 103, 115, 167, 91, 120, 139, 135, 91, 91, 169,
 fire   5: 109, 90, 93, 91, 91, 135, 91, 124, 90, 90, 148, 91, 91, 97, 141, 91.
```

11.4 Duplicating and Reflecting Single-Channel Recipes

Now that we know how to build things with single-channel glider recipes, we would like to be able to manipulate them. In particular, it is going to be important that we are able to reflect these recipes and also duplicate them, so that we can use one copy of the recipe to build something while potentially saving the other copy for later use.

[24] It is actually possible to use a zero-degree elbow to implement *any* slow salvo synthesis (i.e., zero-degree elbows are universal, just like we saw 90-degree elbows are universal in Theorem 11.2). Indeed, we could use the zero-degree elbow to fire a slow salvo that duplicates the hand block, thus leaving us with *three* blocks: the zero-degree elbow block, which fires at a (slow, not single-lane) 90-degree elbow block, which fires at a target block. However, actually synthesizing anything in this manner is excruciatingly slow, so we do not make use of this technique.

Free PDF. See conwaylife.com/book for pattern files and print version.



Figure 11.11: A single-lane sequence of 118 gliders creating a hand block and then firing 4 parallel slow gliders at that hand block so as to move it up by 11 cells. The parallel slow gliders are created on the correct lanes by using the sequences from Table 11.3.

Thanks to the fact that these recipes are single-channel, a single Snark can be used to reflect the entire recipe. However, if there is not already a Snark at the location we would like the reflection to take place, we have to be a bit more clever. In this situation, we can use a single-channel recipe to build a Snark directly in the path of the recipe itself, then let an arbitrary number of gliders be reflected by the Snark, and then use another single-channel recipe to destroy the Snark once it is done being useful. The recipes that create and destroy this in-lane Snark are appropriately called the **Snarkmaker** and **Snarkbreaker**.

11.4.1 The Snarkmaker

The Snarkmaker can be constructed by first searching for a unidirectional slow-salvo recipe for a Snark, and then using the zero-degree elbow toolkit from the previous section to turn it into a single-lane recipe (at the expense of having many more gliders). One unidirectional slow salvo synthesis of a Snark that works is the 95-glider monstrosity displayed in Figure 11.12.[25]

We can turn this unidirectional slow salvo into a single-channel recipe by consulting a database of zero-degree single-channel glider sequences. For example, the first glider in the Snark-creating slow salvo is on lane 15 (relative to the input lane of the Snark), so it can be created by the lane 15 recipe from Table 11.3:

```
109, 90, 93, 91, 91, 158, 94, 113, 91, 90, 91, 96, 90, 142, (90).
```

[25]This salvo was found by Adam P. Goucher in March 2017.

Free PDF. See conwaylife.com/book for pattern files and print version.

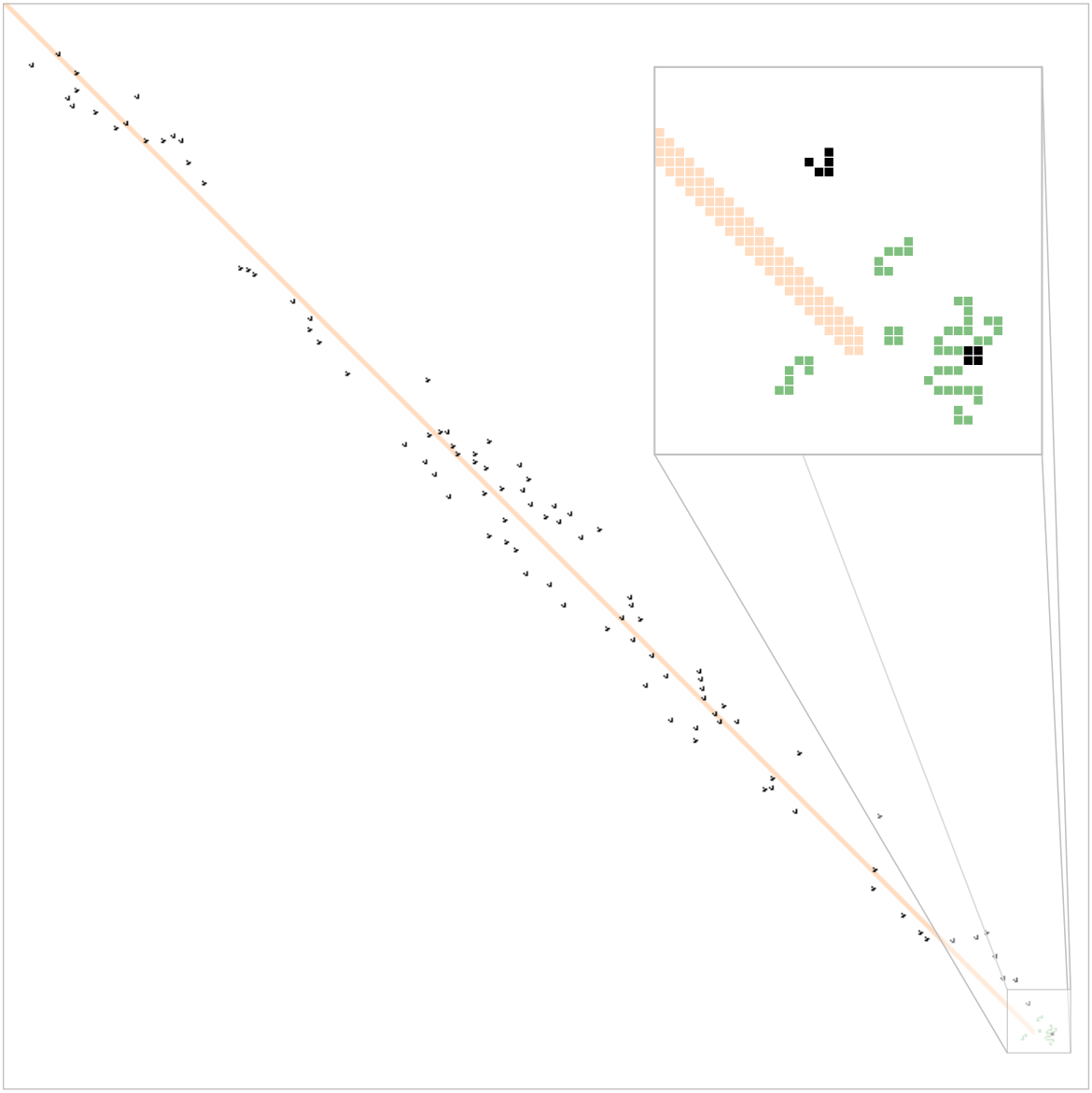


Figure 11.12: A 95-glider unidirectional p2 slow salvo synthesis of a Snark. The location of the to-be-constructed Snark is highlighted in green, and the lane that it can reflect gliders from is highlighted in orange.

The next glider is on lane 22, so we can again simply use the corresponding recipe from Table 11.3:

```
109, 91, 94, 91, 91, 93, 90, 91, 91, 90,
100, 90, 94, 90, 108, 90, 91, 91, 119, (90).
```

After stringing together 95 recipes of this type (one for each glider in the slow salvo),[26] we get a single-channel recipe consisting of 2 253 gliders that creates a Snark. However, there are still a few other zero-degree elbow recipes that we must prepend and append to make this Snarkmaker actually usable:

[26] This process is completely mechanical, but also extremely tedious, which means it is best done by a computer instead of a human. Also, the particular recipes that we used in the Snarkmaker do not always match up exactly with the recipes from Table 11.3—multiple recipes are known for most lane numbers.

Free PDF. See conwaylife.com/book for pattern files and print version.

- We first need to use a recipe that creates a block that is offset far to the side, so that the glider recipes that are reflected through the Snark have something to hit (i.e., we need to create a block that will become the 0-degree elbow after the Snark has been made). The 90-degree hand recipe (11.1) can carry out this task.
- We then need to use a recipe that creates the zero-degree hand that will be used as the seed for the synthesis of the Snark (i.e., the seed block in Figure 11.12). The recipe from Section 11.3.3 can carry out this task.
- We need a recipe that can destroy the original zero-degree elbow, since it serves no purpose after the Snark is constructed (and in fact just prevents gliders from entering the Snark). The following 8-glider recipe takes care of this task:

  `109, 91, 95, 113, 90, 134, 90, (90).`

- Finally, we can use the elbow-moving recipes of Table 11.8 to adjust the relative positioning of the three different elbows that are used throughout all of these operations.

After we put all of this together, we get a single-channel glider recipe containing a grand total of 2427 gliders that creates a Snark, and also moves the elbow to its far side so that subsequent glider recipes on that lane hit the elbow after passing through the Snark. This **Snarkmaker** is displayed in Figure 11.13, and the complete sequence of glider timings that make it up is provided in Appendix B.2.

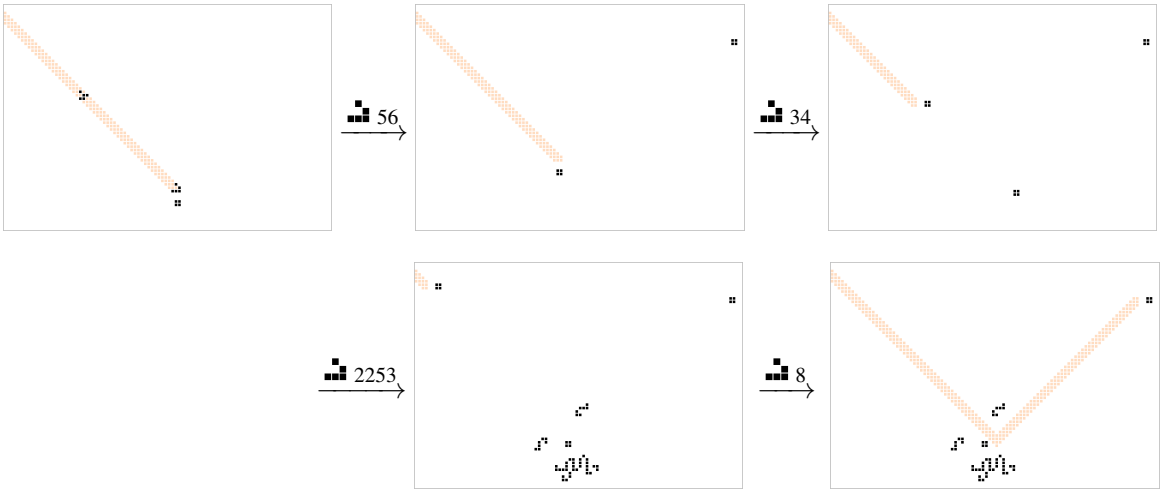


Figure 11.13: The Snarkmaker, which is a single-channel glider recipe that first creates the post-Snark elbow block, then creates a zero-degree elbow, then creates a Snark via that zero-degree elbow, and then finally destroys the zero-degree elbow. Not displayed are 76 (optional) final gliders that are typically appended at the end to push the post-Snark elbow far enough away from the Snark that another Snarkmaker recipe could be used.

Single-channel recipes are also known for creating a Snark in its three other possible orientations.[27] However, they are all roughly as large as the Snarkmaker, and the details of their construction are all very similar to that of the Snarkmaker, so we do not present them here.

11.4.2 The Snarkbreaker

The Snarkbreaker, which destroys a Snark that is in the path a single-channel glider stream, is *considerably* simpler than the Snarkmaker.[28] To create it, basically all we need is a way of colliding a

[27] These recipes were all found by Adam P. Goucher in March 2017—see conwaylife.com/forums/viewtopic.php?p=42092#p42092.

[28] In Life, just like in life, it is typically much easier to break something than it is to make it.

Free PDF. See conwaylife.com/book for pattern files and print version.

glider (or gliders) with the Snark so as to destroy it, and a single-channel recipe that creates those gliders. Recipes that accomplish these tasks, as well as their net effect of converting a Snark back into an elbow block, are displayed in Figure 11.14.

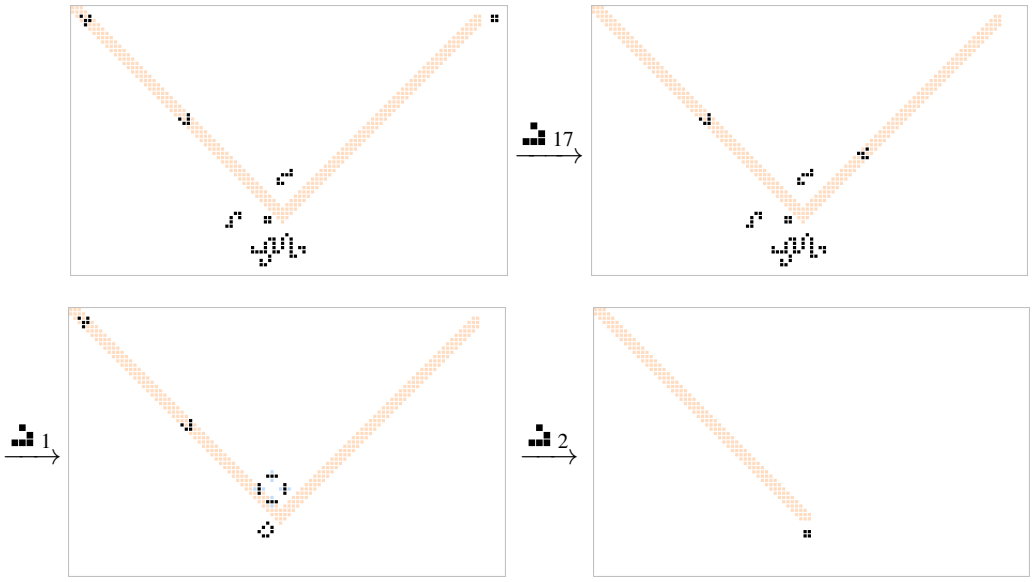


Figure 11.14: The **Snarkbreaker** is a 20-glider single-channel recipe that destroys a Snark and turns it back into an elbow block.

Altogether, the Snarkbreaker consists of 20 gliders, with the following timing gaps:

```
return glider: 93, 91, 118, 93, 151, 90, 99, 155, 120, 92, 108, 90, 102, 164, 90, 90,
Snark destroy: varies, 143, 97, (90).
```

The exact timing of the glider that is labelled "varies" depends on how far away the elbow block is from the Snark that is being used to destroy it. The "varies" glider must collide with the Snark at the same time as the glider that is fired backward from the elbow by the previous 17 gliders. It follows that if the elbow block is n cells away from the Snark, then the "varies" glider must come roughly $8n$ generations after the previous one.

11.4.3 The Scorbie Splitter

The final piece of machinery that we will need for our upcoming constructions is a method of making a single-channel recipe duplicate itself. We have seen numerous methods of duplicating glider streams (e.g., the periodic duplicator of Figure 6.10, and the stable duplicators of Figure 7.11), but we cannot assume that such mechanisms are already in the path of the single-channel recipe. Instead, we would like to use a single-channel recipe to build a duplicating mechanism in such a way that further recipes along the same lane can make use of it.

The simplest such mechanism to synthesize is called a **Scorbie splitter**,[29] which is displayed in Figure 11.15. This conduit uses the left half of the large easier-to-synthesize version of the syringe from Figure 7.8(b) to convert the input glider into a Herschel, and then another conduit to convert that Herschel into two output gliders (this is the exact same method that was used to create the syringe variant in Exercise 7.10, and it has the exact same repeat time of 90 generations as that pattern).

[29]Named after Dongook Lee, who uses the ConwayLife.com forums username "Scorbie", and who was partially responsible for the identification of this conduit and its prominent usage in self-constructing circuitry.

Free PDF. See conwaylife.com/book for pattern files and print version.

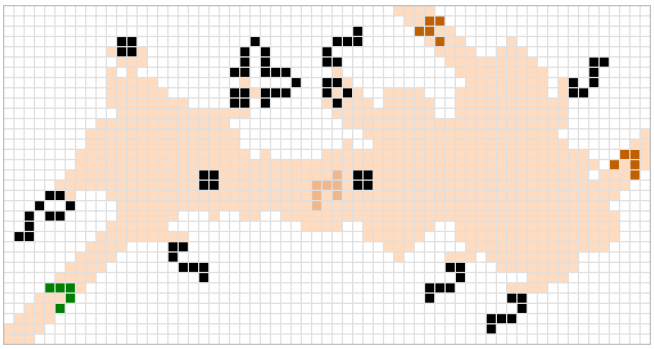


Figure 11.15: The **Scorbie splitter** is a stable glider duplicator that is made up of the left half of a syringe, followed by an H-to-2G conduit on the right that was found by user "praosylen" on the ConwayLife forums in January 2016.

Just like we can use a single-channel recipe to synthesize a Snark directly on its path, so too can we use such a recipe to synthesize a Scorbie splitter directly on its path. In principle, we could build a recipe to carry out this task by hand—we just need to make a slow salvo synthesis that builds each of the Scorbie splitter's still lifes, one at a time, and then convert that slow-salvo synthesis into a zero-degree single-channel recipe in the same way that we did with the Snarkmaker. However, the computer script *slsparse* (see `conwaylife.com/wiki/Slsparse` for tutorials and a download link) automates this procedure and is able to generate single-channel recipes for objects of this size and complexity in a minute or two. One such Scorbie-splitter-making recipe consists of 4 006 gliders, whose timing gaps are provided in Appendix B.3.

We can also use single-channel recipes to synthesize Scorbie splitters in any other orientation, or via a 90-degree elbow instead of a zero-degree elbow. In all cases, the method of construction is basically the same, and slsparse can deal with them straightforwardly—see Exercise 11.22. In fact, once we figure out how to synthesize an object from one orientation via a single-channel recipe, the other orientations come for free since we can just prepend as many Snarkmakers to the recipe as we need—see Exercise 11.29.

11.5 A Slow Demonoid

The downside of single-channel glider synthesis versus slow salvo synthesis is that the former requires somewhere around 25 to 40 times as many gliders to synthesize the same object (roughly the length of an average elbow-moving sequence followed by a glider-firing sequence). However, the upside of it is that the surrounding circuitry that makes use of the synthesis can be considerably simpler, since it just needs to be able to manipulate gliders along a single lane.

We now demonstrate the utility of single-channel glider synthesis by creating another self-constructing spaceship. This spaceship, which is called a **Demonoid**[30], is smaller than the Gemini by roughly one order of magnitude—it has approximately 7×10^4 live cells instead of 8×10^5, and its bounding box has a side length of approximately 3×10^5 cells instead of 4×10^6.

Much like the Gemini works by bouncing 12 glider streams back and forth between two identical ends that they construct and destroy, this Demonoid works by bouncing a single-channel recipe back and forth between two mirror-image ends that they construct and destroy. This time, the ends consist just of a Scorbie splitter (to duplicate the single-channel recipe, so that one copy can be used for construction purposes while the other copy is preserved) and a Snark (to send the preserved copy of the single-channel recipe back toward the other end of the Demonoid), as illustrated in Figure 11.16.

[30]The name "Demonoid" is a portmanteau of "diagonal" and "Geminoid".

Free PDF. See conwaylife.com/book for pattern files and print version.

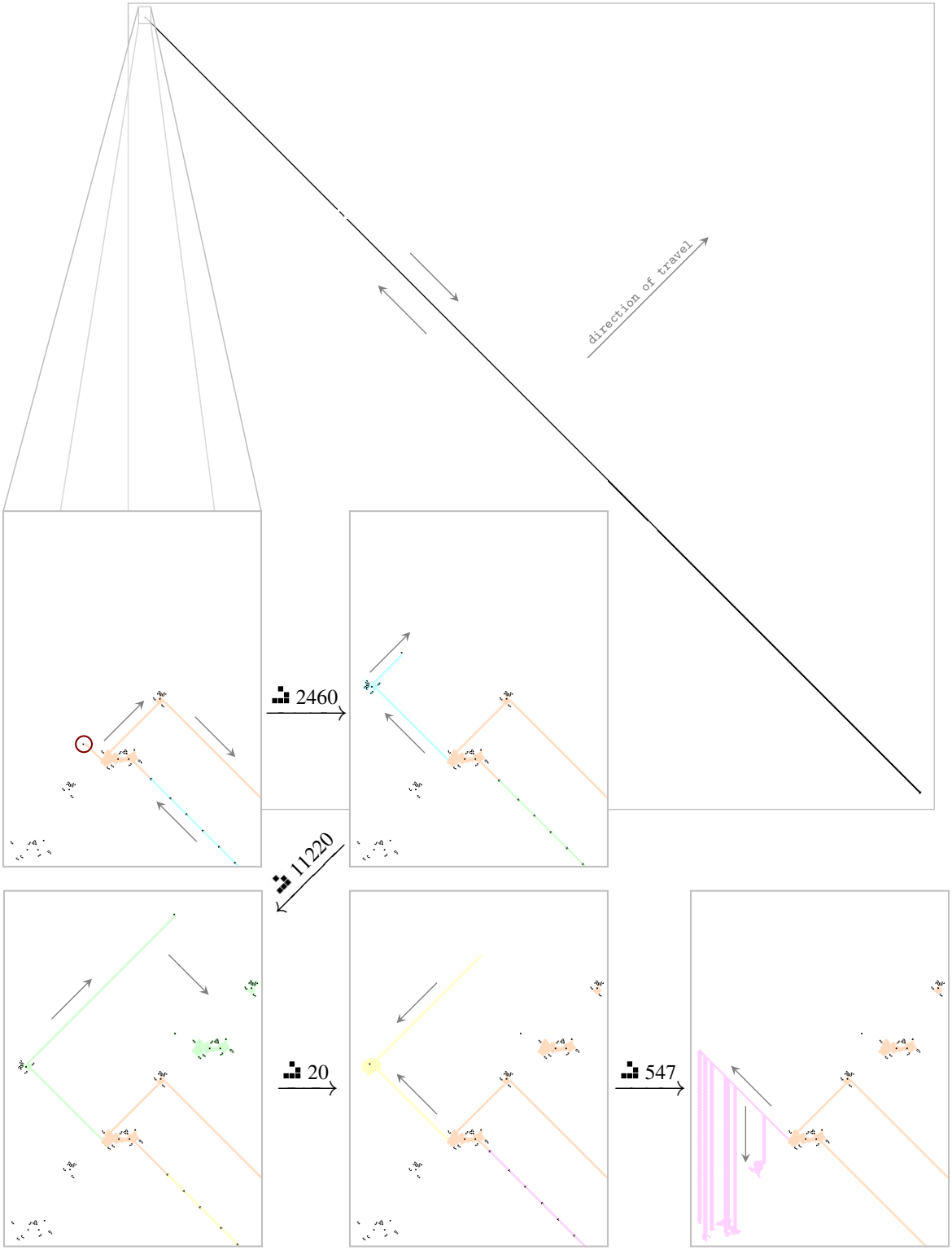
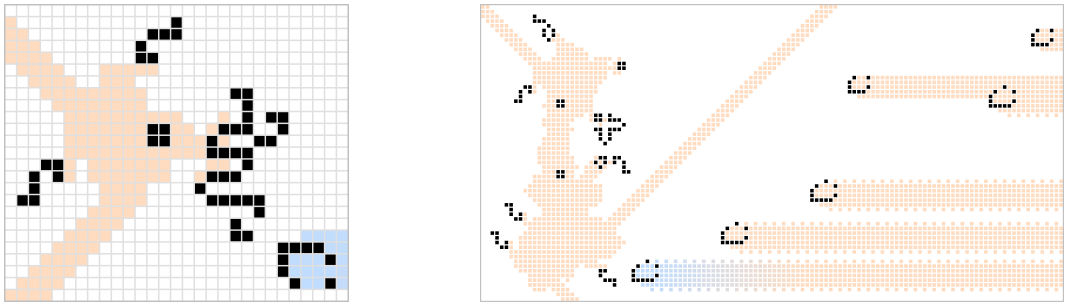


Figure 11.16: A $c/16384$ Demonoid spaceship that works by bouncing a (very long) single-channel glider recipe between two self-constructing ends that are made up of a Scorbie splitter and a Snark. The first 2460 gliders collide with an elbow block (circled in red) so as to build a temporary Snark (highlighted in aqua) to provide a better angle for construction. The next 11220 gliders use a 90-degree elbow to construct the next Scorbie splitter and Snark (highlighted in green). The next 20 gliders implement a Snarkbreaker that gets rid of the temporary Snark (highlighted in yellow). Finally, the last 547 gliders create some lightweight and middleweight spaceships that destroy the previous, no longer needed, Scorbie splitter and Snark (highlighted in magenta).

Free PDF. See conwaylife.com/book for pattern files and print version.

The single-channel recipe that this Demonoid uses consists of 14247 gliders that perform the following tasks:

- Construct the next Scorbie splitter and Snark 128 cells in front of the ones currently in use. This is done by using a Snarkmaker to bend the single-channel recipe around the existing Scorbie splitter and Snark, and then do the actual construction via a 90-degree elbow.
- Destroy the previous Scorbie splitter and Snark that are no longer needed. Since the Scorbie splitter and Snark (and pretty much any other object—see Exercise 5.3) can be destroyed by gliders, this task could be taken care of via another Snarkmaker and 90-degree elbow. However, it is much cheaper (costing just 547 gliders instead of 3000 or so) to instead perform the destruction via lightweight and middleweight spaceships. Figure 11.17 illustrates a method of using a single LWSS to destroy a Snark, as well as a slow salvo of 4 MWSSes and 2 LWSSes that destroys a Scorbie splitter. Appendix B.4 lists the timing gaps that specify a 461-glider single-channel recipe for producing this slow salvo – see Exercise 11.25 for a breakdown of what the different pieces of this recipe do.



(a) An LWSS about to cleanly destroy a Snark. **(b)** Six slow xWSSes about to cleanly destroy a Scorbie splitter.

Figure 11.17: A Snark and a Scorbie splitter (and pretty much any other object—see Exercise 11.28) can be destroyed by slow salvos of lightweight, middleweight, and heavyweight spaceships.

Altogether, this Demonoid has period $2^{21} = 2097152$ and speed $128c/2097152 = c/16384$, but those values can be improved slightly by moving its ends closer together—see Exercise 11.23. By adding some extra block-pushing recipes to it, so that the new Scorbie splitter and Snark are built more than 128 cells away, we can increase its speed by a considerably larger amount, just like we can use additional PUSH operations to increase the speed of Geminoids. The fastest block-pushing reaction that we currently know lets us make Demonoids with speeds arbitrarily close to, but not equal to, $44c/6379 \approx 0.0069c$ (see Exercise 11.27).

11.6 A Middling Demonoid

In order to construct a Demonoid that is faster than $44c/6379$ (or even one that is close to that speed without being an order of magnitude larger than the Demonoid from Figure 11.16), we need a faster way of moving a construction elbow forward than we have seen so far. One technique that works well (over long distances, at least) is to use the elbow block to synthesize a spaceship that moves in the desired direction, and then shoot it down, turning it back into a construction elbow some time later. By letting the spaceship travel a longer and longer distance before shooting it down, we can move the elbow at speeds closer and closer to that of the spaceship.

Perhaps the simplest elbow-moving spaceships to implement this idea with are Corderships, thanks to how straightforward they are to synthesize (compared to most other spaceships, anyway). Indeed, we saw a seed for the 2-engine Cordership back in Exercise 5.44, and by adding a one-time splitter and

Free PDF. See conwaylife.com/book for pattern files and print version.

some one-time turners to it, we can trigger it via just a single glider from behind (see Figure 11.18). We can build this seed via a unidirectional slow salvo synthesis consisting of 112 gliders, and that slow salvo synthesis can be emulated via a single-channel glider recipe made up of 2 393 gliders. We leave the exact timing gaps to Appendix B.5, and the slow salvo that they implement to Exercise 11.20.

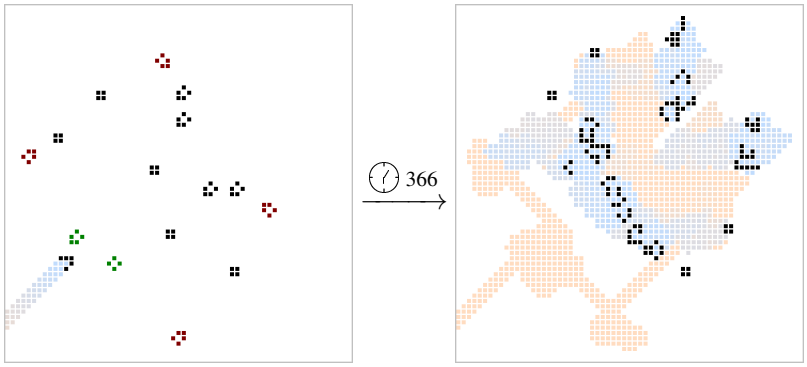


Figure 11.18: A seed that, when hit by a single glider, produces a 2-engine Cordership travelling in the same direction. It uses a one-time splitter (displayed in green) and some one-time turners (displayed in red) to convert the input glider into a pair of synchronized gliders that each synthesize a switch engine.

After the Cordership has been synthesized and travelled however far we would like it to move, it is comparatively straightforward to turn it back into an elbow block via a single-channel recipe. As illustrated in Figure 11.19, a single glider can be used to stop the Cordership, leaving behind just a few simple ash objects. A few more slow gliders can then be used to clean up that ash, leaving just an elbow block behind. Finally, that slow salvo can be converted into a single-channel recipe that uses a zero-degree elbow as usual, such as the following 306-glider recipe (specified by 305 timing gaps):[31]

```
109, 91, 93, 90, 140, 150, 108, 91, 90, 111, 91, 91, 194, 98, 90, 169, 90, 109, 91, 94, 91,
91, 93, 90, 125, 90, 170, 90, 90, 90, 169, 179, 91, 160, 91, 91, 90, 132, 91, 131, 90, 90,
124, 126, 90, 94, 90, 126, 128, 140, 115, 121, 142, 103, 91, 119, 214, 118, 91, 112, 170, 90,
90, 90, 91, 91, 91, 109, 91, 94, 91, 91, 179, 91, 90, 94, 91, 114, 90, 166, 90, 90, 90, 91,
117, 90, 96, 90, 90, 95, 91, 91, 109, 91, 93, 91, 123, 90, 129, 91, 90, 104, 157, 90, 171,
91, 90, 90, 90, 90, 164, 94, 109, 91, 93, 91, 115, 107, 90, 90, 90, 90, 90, 90, 90, 103, 99,
118, 91, 130, 91, 109, 91, 94, 91, 91, 95, 91, 90, 93, 218, 172, 90, 90, 90, 116, 112, 341,
107, 106, 90, 163, 91, 91, 109, 91, 93, 90, 140, 150, 142, 91, 90, 111, 91, 91, 193, 97, 91,
91, 155, 90, 98, 90, 90, 93, 91, 151, 90, 139, 180, 103, 115, 167, 91, 120, 139, 135, 91, 91,
170, 109, 91, 94, 91, 91, 92, 90, 169, 90, 90, 90, 107, 90, 90, 91, 90, 95, 91, 90, 109, 91,
93, 91, 92, 90, 162, 90, 129, 91, 91, 91, 90, 137, 99, 90, 90, 111, 91, 153, 90, 90, 91, 109,
91, 95, 125, 128, 90, 90, 90, 172, 90, 90, 90, 119, 91, 113, 247, 90, 144, 90, 140, 90, 109,
91, 93, 91, 92, 90, 97, 91, 116, 91, 145, 90, 91, 98, 90, 90, 188, 91, 91, 91, 90, 115, 91,
109, 91, 95, 125, 128, 90, 90, 90, 172, 90, 90, 90, 119, 91, 113, 247, 90, 144, 90, 140, 90,
109, 90, 93, 91, 90, 90, 90.
```

We can now place these elbow-to-Cordership and Cordership-to-elbow recipes into the slow Demonoid that we already constructed in order to speed it up. Note that we have to insert this pair of Cordership-based recipes not just in the construction half of the Demonoid (to push the elbow block far forward to construct the next Snark and Scorbie splitter), but also in the destruction half of the Demonoid (to push the elbow block far enough backward to destroy the previous Snark and Scorbie

[31] This recipe can be downloaded in RLE format from conwaylife.com/book. Alternatively, if you are reading this book in Adobe Acrobat Reader, you can click here.

Free PDF. See conwaylife.com/book for pattern files and print version.

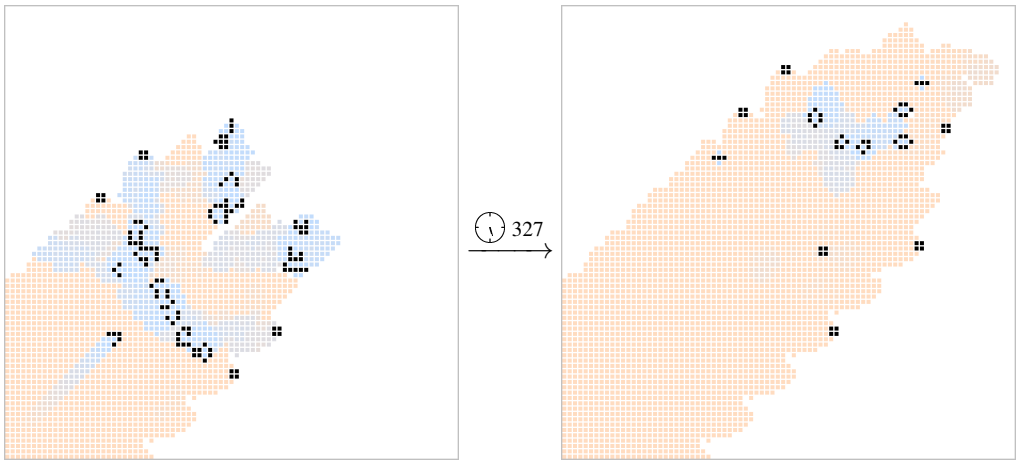


Figure 11.19: A 2-engine Cordership can be shot down from behind by a single glider that turns it into a mess of simple stable objects. Those leftover ash objects can subsequently be cleaned up by additional slow gliders, or converted back into a construction elbow block.

splitter). See Figure 11.20 for a completed Demonoid of this type. It displaces itself by $2^{14} = 16\,384$ cells over the course of $2^{22} = 4\,194\,304$ generations, for a speed of $16384c/4194304 = c/256$, making it a whopping 64 times as fast as our earlier Demonoid.

By waiting longer to shoot down the Cordership after it has been constructed, we can speed this Demonoid up considerably closer to the Cordership's speed of $c/12$. This design's speed limit is actually $c/14$ (not $c/12$) though, since the construction elbow must send back a backward glider as part of a Snarkbreaker once it is no longer needed, to switch the glider recipe over to the destruction elbow. That backward glider has to complete approximately half of its backward journey (i.e., it has to get back past the most recently constructed Scorbie splitter) before the next construction phase can begin. If the Cordership moved a total of n cells before being shot down (over the course of $12n$ generations) then it will take that backward glider $4(n/2) = 2n$ generations to move backward the requisite $n/2$ cells. The total speed of the Demonoid is thus slower than $nc/(12n+2n) = c/14$.[32]

11.7 A Fast Demonoid

We now push these ideas to their absolute limit, and demonstrate how to construct Demonoids with *any* rational speed below $c/4$ (rather than just speeds below $c/14$).[33] In order to make this possible, we have to modify the $c/256$ Demonoid from Figure 11.20 in two ways:

- We have to replace the $c/12$ elbow-carrying Cordership with a $c/4$ diagonal spaceship.
- We have to destroy any no-longer-needed circuitry without making use of Snarkbreakers (which are slow due to their usage of a long-distance backward glider on the construction lane).

We now discuss how to implement both of these changes, so that we can construct Demonoids of essentially any speed of our choosing.

11.7.1 A Faster Elbow-Carrying Spaceship

A rather large problem appears right away if we try to synthesize a $c/4$ diagonal spaceship that we later shoot down from behind by gliders—those gliders cannot possibly catch up to the elbow-carrying

[32] However, it is possible to slightly redesign this Cordership so as to avoid using a Snarkbreaker, thus pushing its not-quite-attainable speed limit up to $c/12$ from $c/14$—see the upcoming Section 11.7.2.

[33] Recall from Theorem 4.1 that no diagonal spaceship can go faster than $c/4$.

Free PDF. See conwaylife.com/book for pattern files and print version.

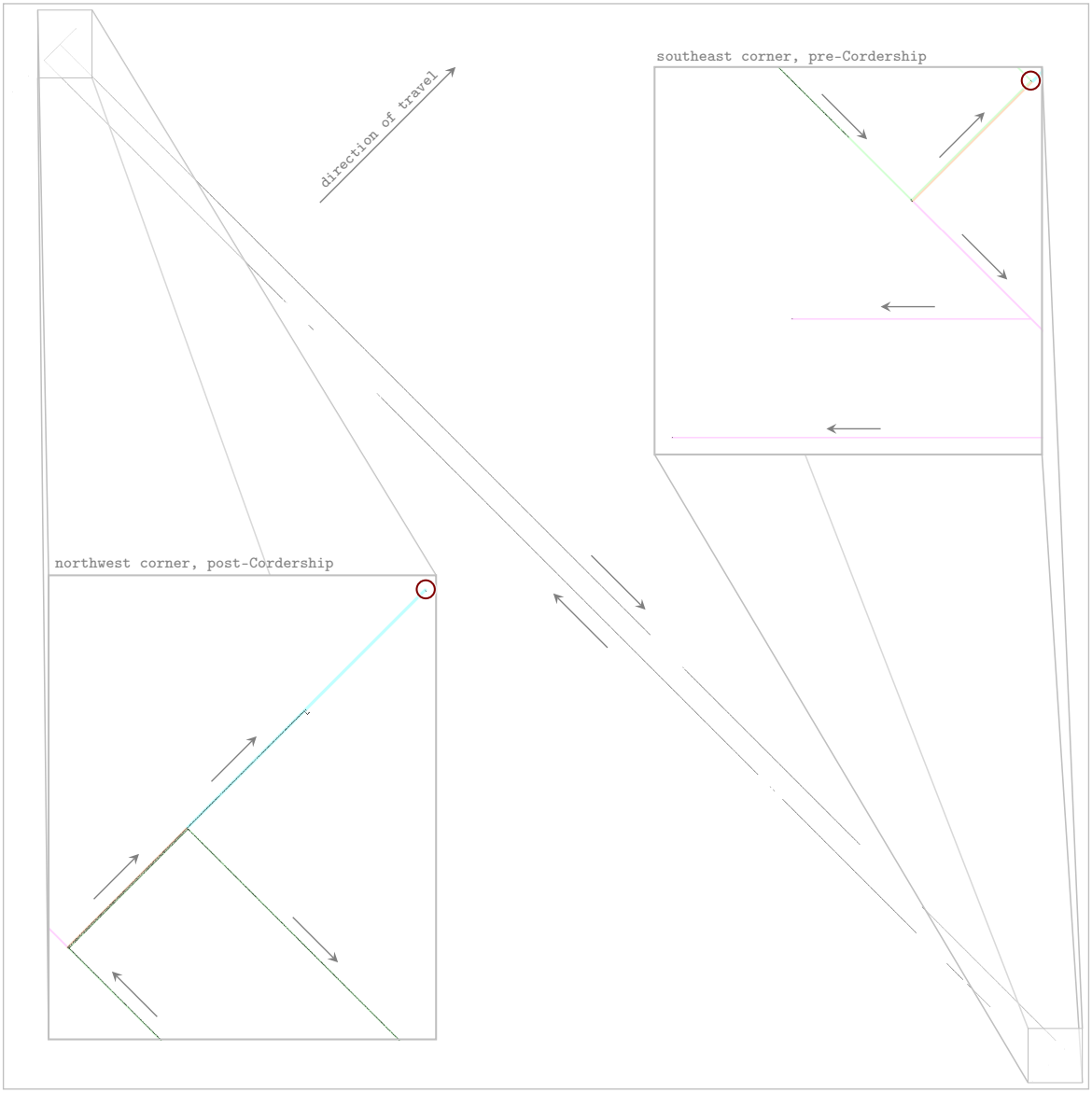


Figure 11.20: A $c/256$ Demonoid spaceship that works by bouncing a (very long) single-channel glider recipe (highlighted in green) between two self-constructing ends. In the phase shown here, the recipe is about to construct a Cordership in the southeast corner (at the location circled in red), which moves northeast along the path highlighted in aqua. The single-channel glider recipe then destroys the Cordership (at the location similarly circled in the northwest corner) so as to leave behind a construction elbow. That elbow is then used to build the Snarks and Scorbie splitters necessary to repeat this process. Finally, a single glider is sent backwards as part of a Snarkbreaker, letting the single-channel glider recipe follow the path highlighted in magenta, which is used to destroy old Snarks and Scorbie splitters that are no longer needed.

spaceship since they travel at the same speed. To get around this problem, we instead synthesize a $c/4$ diagonal *wickstretcher*, since we can burn wicks at speeds faster than $c/4$ and can thus shoot down a $c/4$ wickstretcher from behind.

We saw a synthesis for a $c/4$ diagonal boatstretcher based on the crab spaceship back in Exercise 5.29, and it is fairly routine at this point to turn that glider synthesis into a unidirectional slow synthesis, and then finally into a single-channel recipe. Indeed, Figure 11.21 shows a seed for this

Free PDF. See conwaylife.com/book for pattern files and print version.

boatstretcher that can be triggered by just a single glider from behind[34] (much like the single-glider seed for the Cordership from Figure 11.18). This particular seed also produces some small ash objects near the end of the boat wick (which is easily cleaned up by a few extra slow gliders—see Exercise 11.31), as well as a single glider that travels along the side of the boat wick, whose purpose we will see shortly.

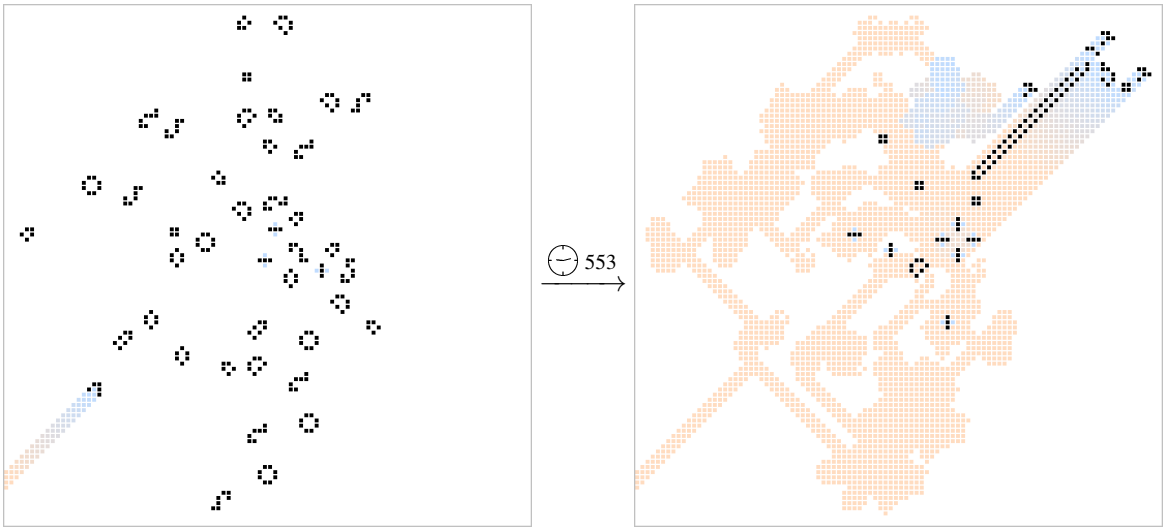


Figure 11.21: A seed that, when hit by a single glider, produces a crab-based boatstretcher travelling in the same direction. It uses some one-time splitters and one-time turners (see Exercise 11.32) to convert the input glider into multiple synchronized gliders that synthesize the boatstretcher.

To shoot down this boatstretcher, we fire a glider at the end of the wick (i.e., the boat that is being stretched) so as to ignite a fuse that travels faster than $c/4$, thus eventually catching up to the crab at the front and destroying it. A suitable technique for igniting a $c/3$ fuse is displayed in Figure 11.22, as is the resulting destruction of the crab boatstretcher.[35] This destruction technique is why we made sure that the seed from Figure 11.21 fires a glider next to the crab—when the fuse catches up to the crab, it just fizzles out, without actually destroying the crab itself. However, the presence of the glider changes that fizzle into an explosion, thus destroying the crab and letting us turn it back into an elbow block.

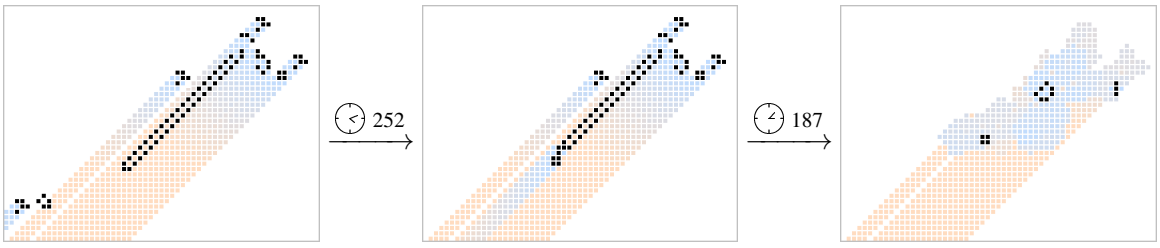


Figure 11.22: By colliding a glider with a boat near the end of a boat wick, we can ignite a $c/3$ fuse. That $c/3$ fuse eventually catches up with the crab and glider at the front, causing them to self-destruct back into some simple ash objects (which could then be converted back into an elbow block by subsequent gliders).

[34]This seed was constructed and heavily optimized via a collaborative effort involving Dave Greene, Tanner Jacobi, and ConwayLife.com forums users "Goldtiger997" and "toroidalet" in July and August 2020.

[35]There is also a common $2c/3$ fuse that can burn through this wick and destroy the crab—see Exercise 11.34.

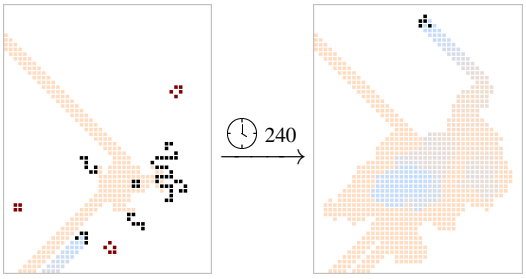
Free PDF. See conwaylife.com/book for pattern files and print version.

11.7.2 A Faster Destruction Method

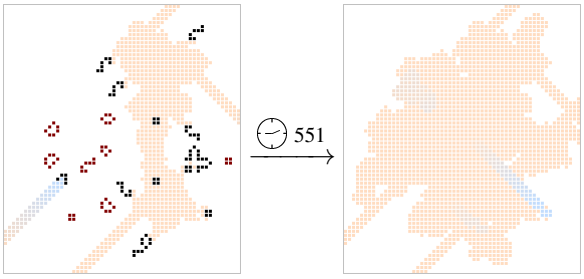
By making use of this wickstretcher instead of a Cordership for long-distance elbow moves, we can now construct Demonoids with speeds approaching $c/6$. However, we cannot yet get speeds arbitrarily close to $c/4$, due to the return glider that is needed for the Snarkbreaker that we discussed earlier (i.e., the same reason that the Cordership-based Demonoid of Section 11.6 can reach speeds close to $c/14$, but not $c/12$). To solve this problem, we introduce another method of destroying old, no-longer-needed circuitry: by placing other still lifes nearby that facilitate their destruction.

For example, Figure 11.23 illustrates how still lifes can be placed around a Snark and Scorbie splitter so that they can each be destroyed by a single glider.[36] Importantly, these still lifes do not interfere with the usual function of the circuit (which is to either reflect or duplicate a glider).

The circuit-destroying glider comes from the same direction as the usual input glider, but on a different lane. When destruction is complete, a glider is released, aimed at the next piece of circuitry that is due to be destroyed, with exactly the correct timing relative to the final recipe gliders. This allows circuit destruction to be triggered immediately, without having to wait for a return glider from far away as in a Snarkbreaker.



(a) A Snark, seeded with 3 extra still lifes, being destroyed by a single glider.



(b) A Scorbie splitter, seeded with 8 extra still lifes, being destroyed by a single glider.

Figure 11.23: Additional still lifes (displayed in red) can be placed near a circuit so as to make it simpler to destroy, without affecting its functionality. Here we use (a) 3 extra still lifes to turn a Snark into a one-time turner, and (b) 8 extra still lifes to turn a Scorbie splitter into a one-time splitter.

It's important to notice that these destruction recipes produce output gliders, so we don't need to generate the destruction-triggering gliders via single-channel recipes at all. Instead, we can just use a single glider that follows along behind (but slightly offset from) the single-channel recipe. This cleanup glider triggers the destruction of every single piece of circuitry in the Demonoid once it is no longer needed, in a continuous chain reaction. After one piece of circuitry is cleaned up, the glider is reflected toward the next piece of circuitry that needs cleaning up, and so on. Again, the timing has to be carefully controlled so that the cleanup glider neither catches up to the recipe nor falls behind it.

11.7.3 The Speed Demonoid Itself

Now that we have faster elbow-moving and circuit-destroying techniques, we can build a much faster Demonoid, appropriately named the **Speed Demonoid**.[37] The general layout of this Demonoid is very similar to that of the other Demonoids that we have seen—it is mostly made up of its single-channel glider recipe, which creates Snarks and Scorbie splitters that it zig-zags back and forth between.

The only additional wrinkle that we add to this Demonoid design is that we use the crab as a *double* boatstretcher instead of just as a single boatstretcher. That is, we will have the crab create and

[36] Still life configurations like this that aid the self-destruction of a given pattern can be found via Simon Ekström's *GoL-destroy* program, which is available at `github.com/simeksgol/GoL_destroy`.

[37] Constructed by Pavel Grankovskiy in September 2020.

Free PDF. See conwaylife.com/book for pattern files and print version.

stretch two boat wicks, as illustrated in Figure 11.24. The reason for making this change is that we want to use the crab to create two construction elbows, not just one—an elbow for making the next Scorbie splitter, and another elbow for making the next Snark. The two boat wicks can be burned independently so as to create those elbow blocks whenever we like (though we must be careful that the crab is only destroyed by the second fuse, not the first).

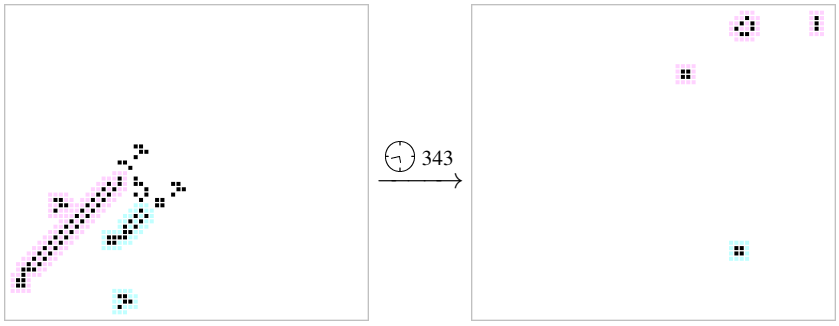


Figure 11.24: A double boatstretcher that can have its two wicks ignited independently, allowing the crab two carry two construction elbows instead of just one. Here, the first fuse (highlighted in aqua) creates a small explosion, without hurting the crab, that the nearby glider helps turn into the first elbow. The second fuse (highlighted in magenta) then destroys the crab, and the nearby glider helps turn it into the second elbow, as in Figure 11.22.

We similarly needed to make two construction elbows in each of our previous Demonoids. In the slow Demonoid from Figure 11.16, we first pushed the construction elbow to the correct location to build the next Scorbie splitter, and then once it was built we pushed the construction elbow forward again to the correct location to build the next Snark. In the $c/256$ Demonoid from Figure 11.20, the Cordership was shot down into an elbow at the location of the next Snark, and then a backward glider was shot back to create an elbow at the location of the next Scorbie splitter (alternatively, we could have used two Corderships). We could use similar tricks in this Speed Demonoid, but a double boatstretcher is a bit faster and more elegant than two single boatstretchers.

Rather than using zero-degree recipes to build the gliders that ignite the $c/3$ boat fuses, it will be slightly more efficient to place two isolated gliders on their own lanes, away from the single-channel recipe, so as trigger the ignitions. We can redirect these gliders, which we refer to as `TRIGGER1` and `TRIGGER2`, around the same path as the single-channel recipe via one-time turners and one-time splitters (rather than Snarks and Scorbie splitters). Furthermore, since the Snark and Scorbie splitter destruction technique of Figure 11.23 turns those circuits into one-time-turners and splitters, respectively, the `TRIGGER2` glider can naturally serve a secondary role as the one that destroys no-longer-needed Snarks and Scorbie splitters.

Putting all of this together finally gives us a completed Speed Demonoid, which is displayed in Figure 11.25.

11.7.4 Adjusting the Speed Demonoid

Now that we have constructed a single Speed Demonoid, it is straightforward to adjust its speed, even by hand. In the Speed Demonoid of Figure 11.25, there are two parameters that we are free to adjust:

1) The first parameter is the number of full diagonals separating the two mirror-image halves of the Demonoid, which we denote by m. We used a separation of $m = 418\,930$ full diagonals in Figure 11.25, and increasing m has the usual effect of increasing the Demonoid's period by 8 generations, without changing how far it moves over the course of its period.

Free PDF. See conwaylife.com/book for pattern files and print version.

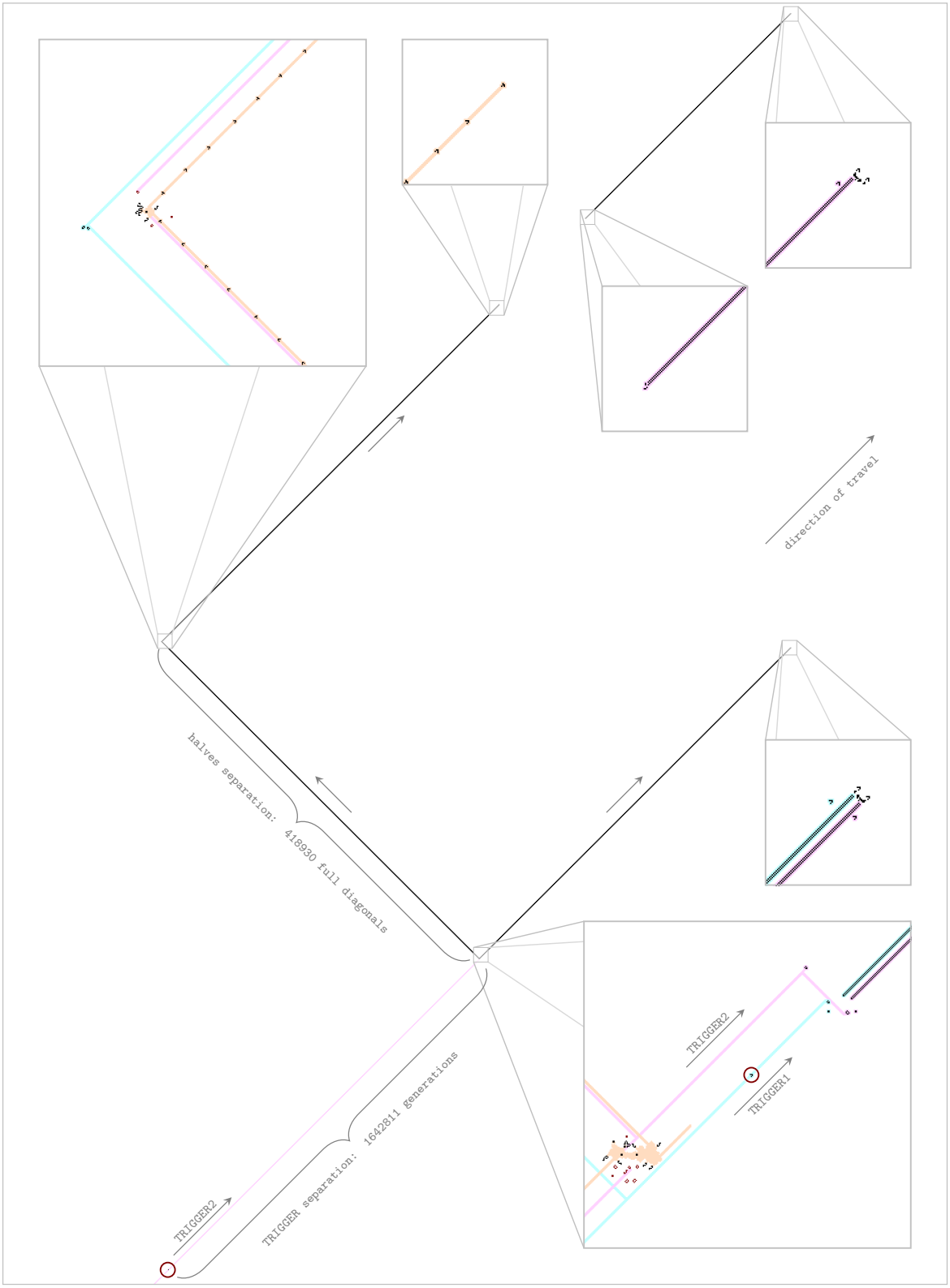


Figure 11.25: A Speed Demonoid spaceship that moves 3285622 cells diagonally every 16493928 generations, for a speed of $1642811c/8246964 \approx 0.1992c$. The single-channel glider recipe (whose path is highlighted in orange) constructs Snarks, Scorbie splitters, and crab-based double boatstretchers. The two TRIGGER gliders (circled in red and highlighted in aqua and magenta, respectively) each ignite one of the fuses behind the boatstretcher, and TRIGGER2 also destroys no-longer-needed Snarks and Scorbie splitters.

Free PDF. See conwaylife.com/book for pattern files and print version.

2) The second parameter is the number of generations between the isolated TRIGGER1 and TRIGGER2 gliders, which we denote by n. We used a separation of $n = 1\,642\,811$ generations in Figure 11.25, and we have to be slightly careful about increasing it. The separation between the two trigger gliders determines how far apart each Snark will be from the subsequent Scorbie splitter, so to keep the two halves of the Demonoid lined up, we need to similarly adjust the spacing between each Snark and the *preceding* Scorbie splitter. Thus if we increase the trigger separation n, we also have to move both trigger gliders back by that same amount.[38]

The trigger gap $n = 1\,642\,811$ that we used in the Speed Demonoid of Figure 11.25 is minimal—any smaller gap between the two trigger gliders would result in either TRIGGER1 attempting to light the fuse while some of the mechanism is still under construction, or TRIGGER2 crashing into a Snark while it's still under construction. Similarly, the value of $n+m = 2\,061\,741$ (which measures the total length of the track along which the single-channel recipe travels) that we used in Figure 11.25 is also minimal—any smaller value of $n+m$ would result in the single-channel recipe trying to use a Snark before it is done being built.

While we cannot decrease n or $n+m$,[39] we are free to *increase* n and m as much as we like, resulting in larger and larger Speed Demonoids. After adjusting the values of n and m to our liking, the resulting Speed Demonoid's speed will be

$$\frac{2nc}{8n+8m} = \left(\frac{n}{n+m}\right)\frac{c}{4}.$$

Every rational number between 0 and 1 can be written in the form $n/(n+m)$ for some integers $m, n > 0$, simply by first choosing n to get whatever numerator we want, and then choosing m to get whatever denominator we want. Furthermore, since $n/(n+m) = (kn)/(kn+km)$ for all $k > 0$, we can choose m and n to be as large as we like. In particular, we can choose $n \geq 1\,642\,811$ and $n+m \geq 2\,061\,741$ so that the resulting Speed Demonoid is actually constructible.

For example, to build a $3c/14$ Speed Demonoid, we want $n/(n+m) = 4(3/14) = 12/14$, so we could first choose $n = 12$ and then $m = 2$ (so that $n+m = 14$). However, these values of n and m are much too small for us to actually be able to build a Speed Demonoid with these parameters (we cannot separate the trigger gliders by just $n = 12$ generations, for example, without components crashing into each other), so we scale them up. That is, we replace n and m by $\widetilde{n} = kn$ and $\widetilde{m} = km$, respectively, where k is an integer chosen so that $kn \geq 1\,642\,811$ and $k(n+m) \geq 2\,061\,741$.[40] In this case, $k = 147\,268$ works, which gives $\widetilde{n} = 12 \times 147\,268 = 1\,767\,216$ and $\widetilde{m} = 2 \times 147\,268 = 294\,536$ (see Exercise 11.37).

This procedure works in general, and shows that Speed Demonoids can be adjusted to have any rational speed that is slower than $c/4$. In fact, there is a computer script[41] that carries out this adjustment automatically, and can construct a Speed Demonoid of any desired (rational, slower than $c/4$) speed.

11.8 Notes and Historical Remarks

While universal construction has been known to be possible since the early days of Life [Wai74, BCG82, Pou85], it took almost 40 more years of work for the Life community to streamline it to the point that it is actually small and fast enough to watch in action.

[38] In other words, to separate the trigger gliders by k more generations, we should move TRIGGER1 and TRIGGER2 back by k and $2k$ generations, respectively (not 0 and k generations, respectively).

[39] However, we can decrease m (the width of the Speed Demonoid) as long as we increase n by the same amount or more.

[40] In other words, $k \geq \max\{\lceil 1\,642\,811/n \rceil, \lceil 2\,061\,741/(n+m) \rceil\}$.

[41] Available for download at conwaylife.com/forums/viewtopic.php?p=105167#p105167.

Free PDF. See conwaylife.com/book for pattern files and print version.

Paul Chapman and Dave Greene built the first universal constructor in 2004 (see Figure 11.26), and it made use of the exact same construction arm that we saw in Figure 11.2. However, instead of encoding the `PULL`, `PUSH`, `FIRE WHITE`, and `FIRE BLACK` commands directly in the input stream of gliders (as was done with the Gemini spaceship of Figure 11.4), they were stored in unary as boats that were separated via sequences of zero, one, two, or three blocks. This resulted in an average of two and a half bits of static storage being used to encode two bits' worth of information—rather inefficient, but it was only intended as an initial prototype.

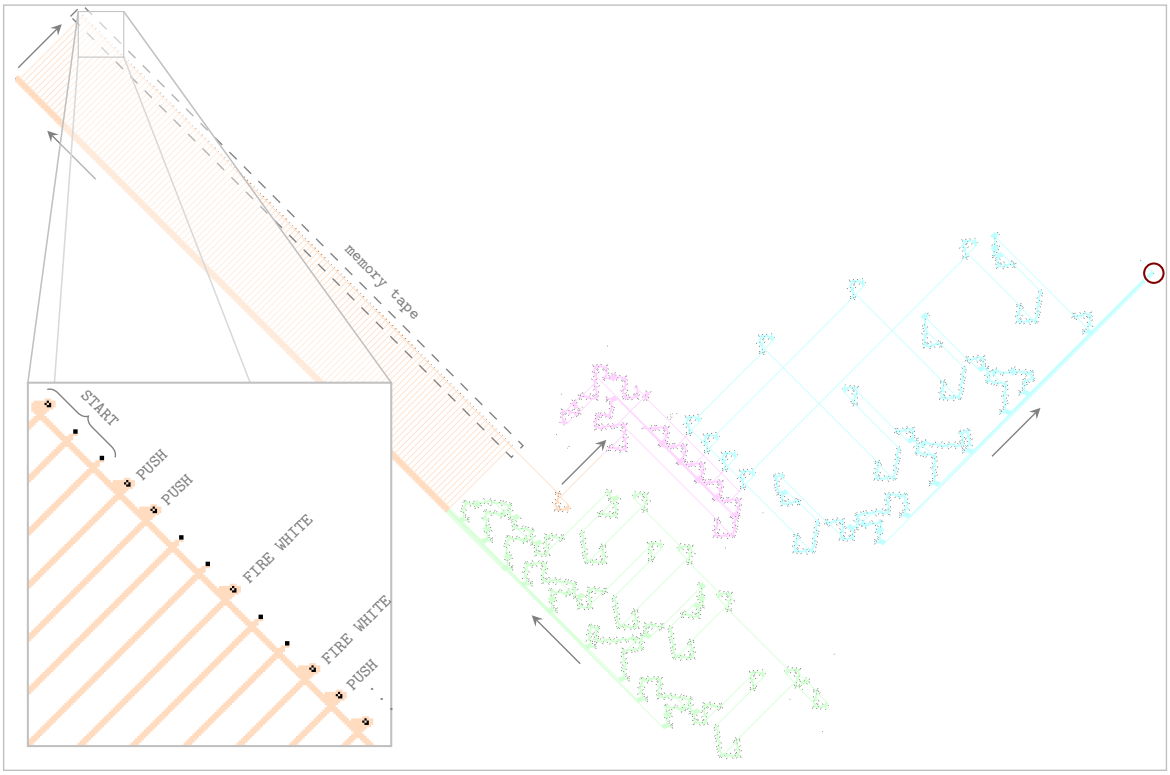


Figure 11.26: A universal constructor that was built by Paul Chapman and Dave Greene in 2004. The conduit highlighted in green fires 10-glider salvos toward the memory tape that is highlighted in orange. Those salvos read the memory tape by either detecting a boat and sending a glider back, or detecting a block and sending nothing back. If a boat was detected, the returned glider is fed into the magenta conduit, which keeps track of how many generations passed between receiving input gliders (i.e., how many blocks there were between boats on the memory tape). That generation gap determines which of four lanes is used to feed into the construction arm that is highlighted in aqua, and thus which of the four construction operations is performed.

The next major development in this area came in 2010 with Adam P. Goucher's universal computer-constructor. While we have not explicitly explored the "constructor" part of this device, it uses the same construction arm that we have already seen, in conjunction with the binary register from Section 9.4 for storing the construction instructions. This binary storage is more efficient than the unary storage of the Chapman–Greene prototype, and the fact that it comes as part of a programmable computer makes it more straightforward to actually make use of.

Both the Chapman–Greene constructor prototype, as well as the Goucher constructor, produced their output unidirectional slow glider salvos by reading a tape consisting of still lifes that represented sequences of glider-firing operations. The idea of encoding information via static objects like this was deeply ingrained in the Life community at the time, likely as a result of how information is similarly stored in actual computers. However, the sudden appearance of the Gemini in 2010 showed that this

Free PDF. See conwaylife.com/book for pattern files and print version.

method of storing data was just silly. Obviously (with hindsight), if you want to build some kind of memory storage system for a bunch of spacings between gliders, you just store a bunch of moving gliders at the spacing you want—there is no need to encode anything at all.

That one insight allowed Andrew J. Wade, who was a complete newcomer to the Life community, to build the first ever oblique and/or self-constructing spaceship, even using extraordinarily sub-optimal components. The Chapman–Greene prototype universal constructor was a first proof of concept, not really intended to be used anywhere since it could straightforwardly be made so much smaller. But Wade wasn't a stable circuitry expert, so he just used what was available: the Gemini spaceship included not just one, but three complete copies of the prototype construction arm, almost completely unmodified. All he did was throw away the hopelessly inefficient static storage system and substitute something better.

The Gemini's completion, and its accompanying philosophy of storing information in glider streams instead of sequences of still lifes, marked the start of a new era of Life construction. In addition to the Geminoids and Demonoids that we saw in this chapter, a flood of other self-constructing spaceships and bizarre patterns based on universal construction were built over the next eleven years. Some of the more notable examples include:

- In October 2010, Paul Tooke constructed numerous **pianola breeders**: breeders that work by using universal constructors to synthesize objects that then synthesize other objects. Because of the flexibility of universal construction, these breeders could be configured to behave in extremely unusual ways. For example, one of these breeders consists of a stationary object (a pair of universal constructors) that synthesizes stationary objects (slide guns), which then synthesize more stationary objects (lines of beehives).[42]
- In November 2013, Dave Greene built a pattern called a **linear propagator**[43] that repeatedly makes copies of itself—something that had been known to be possible since the early 1970s, but had eluded construction for the intervening four decades.
- In August 2014, Dave Greene constructed a pattern that grows outward in a spiral fashion by laying ever-expanding tracks around itself. The original version of this pattern had a bounding box with side length of approximately 1.6 million cells, but has since between reduced to approximately 2000 cells (see Figure 11.27 and Exercise 11.40).
- In November 2015, Chris Cain and Dave Greene completed the first Demonoid, which used somewhat more complicated circuitry than the Demonoids that we investigated in this chapter. Most notably, it used pairs of gliders separated by 10hd to implement slow glider syntheses instead of single-channel glider recipes, which doubled the amount of circuitry that had to be constructed at each end.
- In June 2017, Dave Greene constructed an **Orthogonoid** spaceship,[44] which works similarly to Demonoids, except that it uses middleweight spaceships (instead of gliders) to achieve a slow orthogonal (instead of diagonal) speed of travel. Adam P. Goucher then helped build faster Orthogonoids with speeds up to $65536c/951573 \approx 0.06887c$ in October 2018.
- In December 2018, Chris Cain completed a **camelship**[45] by modifying an earlier **volatility 1 oscillator**[46] design. The oscillator ensures that all its cells oscillate at the pattern's full period, by using single-channel glider recipes to destroy then rebuild its own circuitry. The camelship

[42] See conwaylife.com/wiki/Pianola_breeder, and contrast this with the other breeders that we have seen, which all require some of the synthesized objects to move.

[43] See conwaylife.com/wiki/Linear_propagator.

[44] See conwaylife.com/wiki/Orthogonoid.

[45] See conwaylife.com/wiki/Camelship.

[46] See conwaylife.com/wiki/Volatility.

Free PDF. See conwaylife.com/book for pattern files and print version.

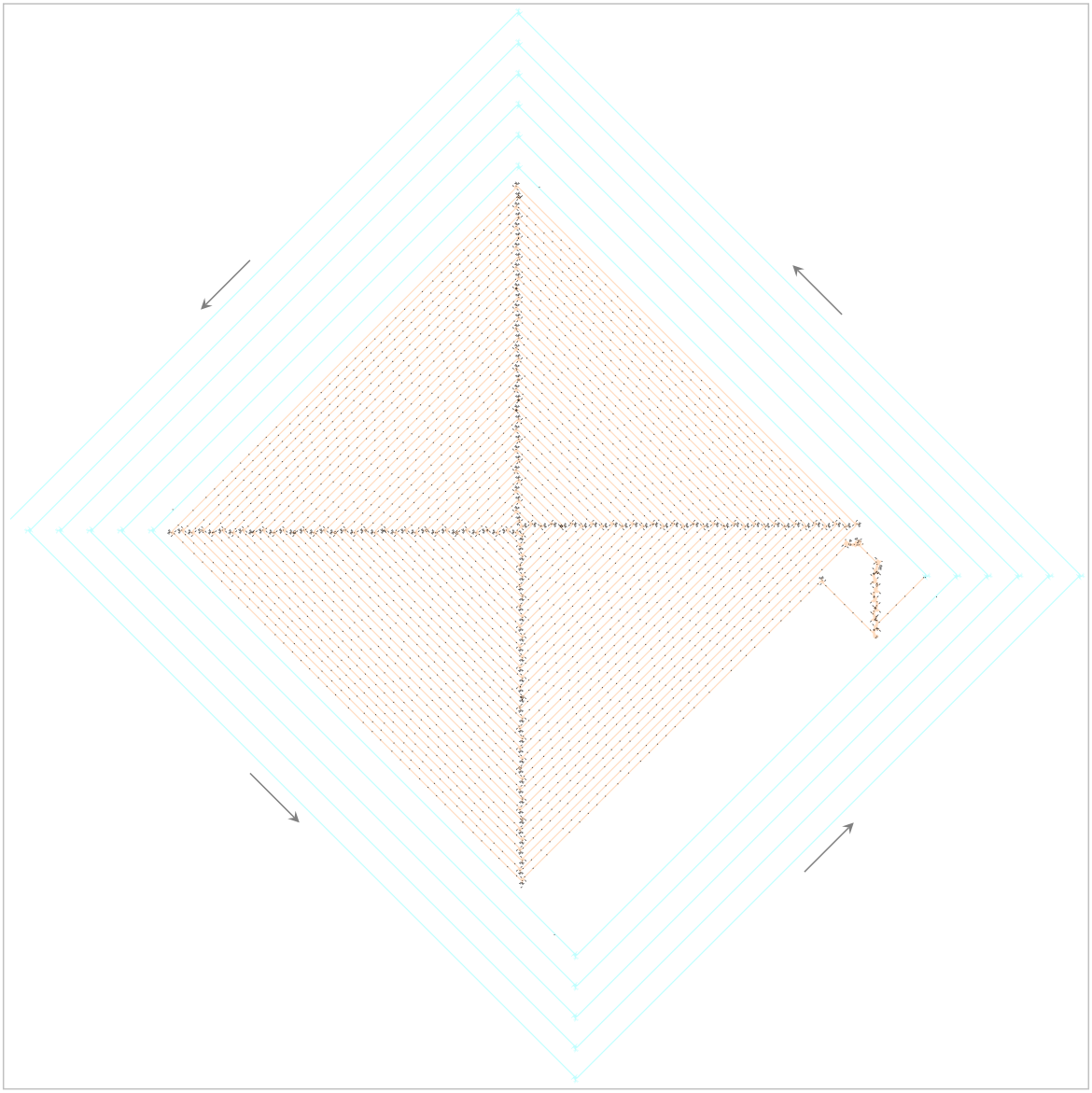


Figure 11.27: A spiral growth pattern that was constructed by Dave Greene in June 2017. A single-channel recipe that is primarily made up of a Snarkmaker bounces around the diamond-shaped track highlighted track in orange, creating Snarks that extend the track as indicated in aqua.

consists of a very similar single-channel signal loop, but the circuit reconstruction happens at a $(3,1)$ offset.

- In April 2019, Michael Simkin constructed **Remini**,[47] an oblique self-constructing puffer that is designed like a single-channel version of the Gemini, but which uses period 30 circuitry to perform the glider reflections and duplications instead of stable circuitry. The tricky thing about constructing this device was the fact that the single-channel recipe has to align with the p30 circuitry, so the only timing gaps that are allowed are multiples of 30.
- In August 2021, ConwayLife.com forums user "Goldtiger997" constructed a **spaceship made**

[47] Short for **R**etro **Gemini**, in reference to the fact that it uses old-style p30 circuitry instead of more modern stable circuitry. See conwaylife.com/wiki/Remini.

Free PDF. See conwaylife.com/book for pattern files and print version.

of spaceships—a spaceship that, in one of its phases, consists only of 144 221 gliders. They also constructed a **reflectorless rotating oscillator**—an oscillator that rotates around a point in the Life plane without the help of any fixed circuitry like Snarks. We discuss these types of patterns in more depth in Section 12.1.

- In December 2021, Pavel Grankovskiy completed a blueprint posted by ConwayLife.com forums user "googleplex" to produce the "QuickSilver" Demonoid, using 1990s-era circuitry (in particular, Silver reflectors) but 21st-century design techniques. Silver reflectors' recovery time of 497 ticks is ordinarily far too slow to support single-channel recipes, but universal construction still turned out to be achievable using overclocked Silver reflectors to reflect and duplicate a carefully chosen subset of known single-channel recipes.

The development of single-channel glider synthesis mostly took place in 2017, when another relative newcomer, Simon Ekström, cavalierly disregarded accumulated prejudices from the previous decade, tried a search, and found that it worked.[48] It was already known in 2015 how to emulate arbitrary glider synthesis via gliders on a single lane, but Ekström was the first to show that it is possible even when the gliders are far enough apart from each other that they can be fed through standard components like Snarks and syringes.

Exercises

solutions to starred exercises on page 467

11.1 [3/5] Rebuild the construction arm from Figure 11.2 via modern components like Snarks and syringes.

11.2 The construction arm of Figure 11.2 requires two synchronized input gliders to trigger each of the `PUSH`, `FIRE WHITE`, and `FIRE BLACK` operations—one to create the frontmost pair of gliders via the `PULL` input, and one to create the additional glider(s) via one of the other three inputs.

- **(a)** [2/5] Find the correct timing for the pair of input gliders to trigger each of the `PUSH`, `FIRE WHITE`, and `FIRE BLACK` operations.
- **(b)** [3/5] Add additional Herschel conduits to that arm (or the one from Exercise 11.1) so that each of the four operations can be triggered by just a single input glider.

11.3 [1/5] The arrangement of two gliders that the `FIRE WHITE` circuit in Figure 11.2 makes is slightly different than the pair of `FIRE WHITE` gliders highlighted in green in Figure 11.1(c). Why is this not a problem?

∗**11.4** [2/5] The two zoom boxes in Figure 11.4 each have two rows of unhighlighted reflectors that never reflect a single glider. What is their purpose?

11.5 [3/5] The pattern in Figure 11.3 uses two universal constructors to synthesize a middleweight spaceship.

- **(a)** Add a loop gun to each of the 8 input lanes, so that universal constructor *repeatedly* constructs middleweight spaceships.
- **(b)** Adjust the input glider recipes so that each middleweight spaceship is produced on the same lane as the previous one (thus making this pattern a middleweight spaceship gun).

11.6 Adjust the input glider sequences for the pattern from Figure 11.3 so that instead of constructing an MWSS, it constructs...

- **(a)** [3/5] a lightweight spaceship,
- **(b)** [3/5] a line of blocks (of any slope and spacing of your choosing), and
- **(c)** [5/5] a 2-engine Cordership.
  [Hint: You can use the 2-direction slow synthesis from Figure 8.46.]

11.7 [3/5] The 12 glider lanes in the Gemini spaceship from Figure 11.4 each implement one of the 4 basic operations from Figure 11.1 (i.e., `PULL`, `PUSH`, `FIRE WHITE`, and `FIRE BLACK`). Determine which lanes implement which of those operations.

11.8 [3/5] Move the two identical ends of the Gemini spaceship farther apart, so as to create a spaceship that travels slower than $(1024, 5120)c/50000000$.

[48] This is not unlike Wade's construction of the Gemini in 2010, or when ConwayLife.com forums user "zdr" found the copperhead spaceship in 2016. In all of these cases, a newcomer simply looked where the members of the Life community thought there wasn't anything to find.

Free PDF. See conwaylife.com/book for pattern files and print version.

∗**11.9** [4/5] By moving the two identical ends of the Gemini spaceship closer together, we can make it slightly faster.

- **(a)** Move the ends 1 full diagonal closer together. What is the speed of the resulting spaceship?
- **(b)** How many full diagonals closer together can the ends be moved without causing the Gemini to self-destruct? What is the speed of the resulting spaceship?

∗**11.10** [3/5] The slope-2 Geminoid from Figure 11.5 was created by just adding additional PUSH operations to the northwest construction elbow (i.e., we increased the value that we called "m" in that section, while leaving "n" fixed at $n = 2048$). What Geminoid slopes are attainable by making this type of change to Gemini? That is, what slopes can be attained by increasing m, but not changing n?

11.11 Recall that the Gemini itself travels at slope 5, and the Geminoid knightship travels at slope 2.

- **(a)** [2/5] To create a Geminoid that travels at a slope of 3, how many PUSH operations should we add to the path of the northwest construction elbow?
- **(b)** [5/5] Construct a Geminoid that travels at a slope of 3.

11.12 [2/5] String together multiple elbow-moving single-channel glider recipes from Table 11.2 so as to make a recipe that moves the elbow forward by exactly...

- **(a)** 11 half diagonals,
- **(b)** −25 half diagonals, and
- **(c)** 38 half diagonals.

11.13 [3/5] String together multiple single-channel glider recipes from Tables 11.1 and 11.2 so as to fire a single glider on output lane...

- **(a)** $3x$,
- **(b)** $0i$, and
- **(c)** $15x$.

11.14 [2/5] The single-channel elbow-pulling sequence displayed in Figure 11.7(b) turns an elbow block into a pond (rather than another block). Why is this not a problem?

∗**11.15** [2/5] The 185-glider single-lane sequence illustrated in Figure 11.8 makes use of two copies of the glider sequence that moves the elbow forward by 10 half-diagonals. In the first sequence the final glider arrives after 90 generations, whereas in the second sequence it arrives after 91 generations. Explain the reason for this discrepancy—why can't the final glider in the second sequence also arrive after just 90 generations?

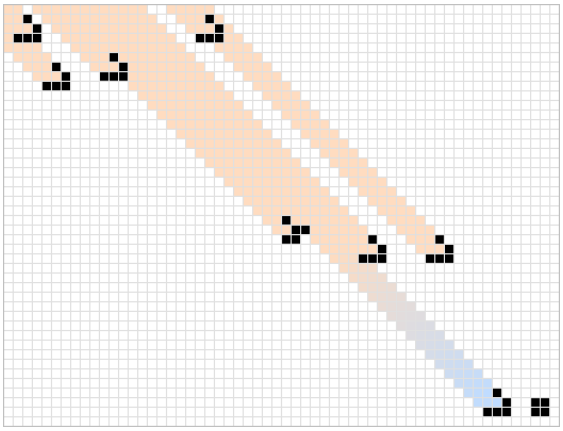
11.16 [2/5] The following single-lane glider recipe moves the elbow block:

```
109, 91, 93, 91, 145, 215, 106, 90, 90,
91, 91, 174, 90, 158, 90, 90, 90, 91,
137, 90, 91, 127, (90).
```

- **(a)** How many half-diagonals (and in which direction) does this sequence move the elbow?
- **(b)** Use this 23-glider sequence to rebuild the 185-glider sequence from Figure 11.8 to be at least 15 gliders shorter, while still performing the same task (i.e., building a hand block and then moving it south by 11 cells).

∗**11.17** [1/5] What is the "move" value of the zero-degree hand-making glider sequence from Figure 11.10 (i.e., how many half-diagonals is the elbow block moved)?

11.18 [3/5] An 8-glider unidirectional slow salvo that synthesizes a heavyweight spaceship[49] is displayed below:



- **(a)** Use the recipes of Tables 11.1 and 11.2 to emulate this slow salvo via a single-channel glider recipe and a 90-degree elbow.
- **(b)** Use the recipes of Table 11.3 to emulate this slow salvo via a single-channel glider recipe and a zero-degree elbow.

11.19 [2/5] The 95-glider slow salvo synthesis of a Snark from Figure 11.12 is an incremental synthesis. Break it down into steps like we did with the 94-glider synthesis from Figure 5.31.

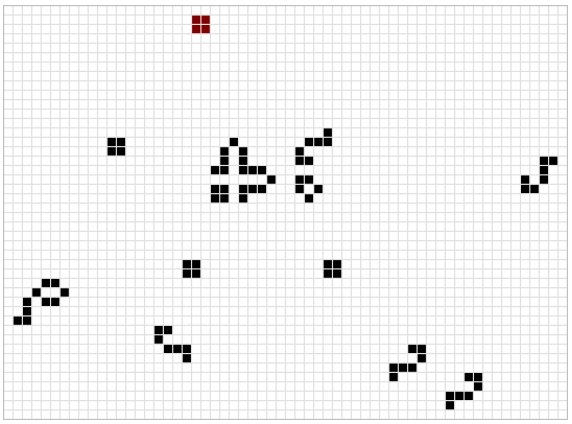
11.20 [3/5] Extract a 112-glider unidirectional slow-salvo synthesis of a 2-engine Cordership from the 2393-glider single-channel recipe that is provided in Appendix B.5.

[49] Found by Adam P. Goucher in December 2016.

Free PDF. See conwaylife.com/book for pattern files and print version.

∗**11.21** [3/5] Extract a unidirectional slow-salvo synthesis of a Scorbie splitter from the 4006-glider single-channel recipe that is provided in Appendix B.3. How many gliders does this slow salvo have?

11.22 [3/5] Use slsparse (`conwaylife.com/wiki/Slsparse`) to create a single-channel glider recipe that synthesizes a Scorbie splitter in the following orientation, starting with a zero-degree elbow block in the location indicated at the top-center:



∗**11.23** [2/5] The bounding box of the slow Demonoid from Figure 11.16 can be decreased considerably just by moving its two stabilizing ends closer together, and doing so also reduces its period and increases its speed.

Adjust that Demonoid so that the longest side of its bounding box is no more than 200000 cells long. What are its period and speed after you make this adjustment?

11.24 [2/5] Recall that the Gemini can be modified to travel in essentially any rational direction, whereas Demonoids can only travel diagonally with slope 1. Explain why—what key aspect of their designs differs?

11.25 [3/5] Recall the 461-glider single-channel recipe from Appendix B.4 that produces the 6-xWSS slow salvo of Figure 11.17(b) for destroying a Scorbie splitter.

- **(a)** Separate this single-channel recipe into smaller recipes that (i) move the elbow, (ii) synthesize lightweight spaceships, (iii) synthesize middleweight spaceships, and (iv) destroy the elbow once it is no longer needed.
- **(b)** In part (a), you should have found two different LWSS recipes, and also two MWSS recipes. Why do there have to be two of each? Would it be possible to create this destruction salvo with just one LWSS recipe and one MWSS recipe?

11.26 [3/5] Demonoid spaceships can be turned into Demonoid puffers simply by removing the destruction portion of their glider recipe. Use this technique to turn the slow Demonoid spaceship from Figure 11.16 into a puffer that leaves behind Snarks and Scorbie splitters.

∗**11.27** [4/5] The following single-lane glider recipe (called the **44hd elbow push**) is the fastest-known elbow-pushing recipe—it moves the elbow block forward by 44 half-diagonals in 2107 generations:

```
109, 90, 93, 91, 90, 95, 91, 91, 138, 157,
 96, 90, 120, 91, 97, 107, 90, 90, 93, (188).
```



- **(a)** Insert copies of this elbow push into the appropriate spots in the slow Demonoid from Figure 11.16 so that it moves forward 194 cells (i.e., full diagonals) throughout its period, instead of just 128 cells. What is the speed of this new, slightly faster Demonoid? [Hint: You will also have to slightly adjust the Snarkbreaker timing, as well as the position of the elbow during the Demonoid's destruction phase.]
- **(b)** Adding more and more of these 44hd elbow pushes makes the Demonoid faster and faster (but also larger and larger). Explain why only $2/3$ of these block pushes contribute to the speed of the Demonoid. That is, explain why the speed limit is much closer to $(2/3)22c/2107$ than $22c/2107$.
- **(c)** Explain why the exact speed limit of this Demonoid design is

$$\frac{(2/3)22c}{2107+(4/3)88} = \frac{44c}{6673}.$$

[Hint: The extra unexplained term in the denominator comes from the Snarkbreaker.]

- **(d)** This 44hd block push can be overclocked so that subsequent block pushes come in just 90 generations later instead of 188. Recompute the Demonoid's speed limit in light of this speed-up.

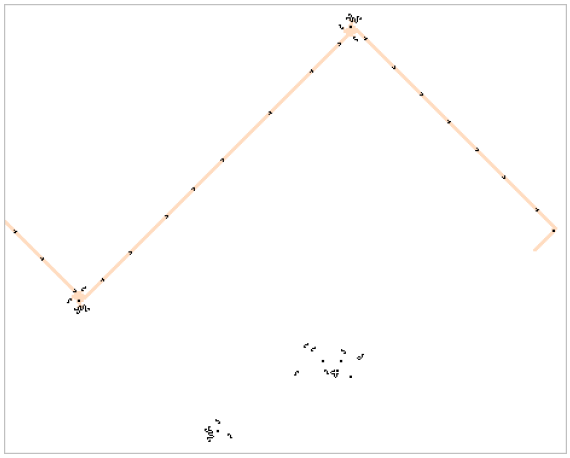
Free PDF. See conwaylife.com/book for pattern files and print version.

11.28 [3/5] The fact that Snarks and Scorbie splitters can be destroyed by xWSS slow salvos is not surprising—most objects can be similarly destroyed.

(a) Construct an xWSS slow salvo that destroys the syringe from Figure 7.8(a).

(b) Construct a single-channel glider recipe that synthesizes the slow salvo from part (a), and thus destroys a syringe.
[Hint: Use the LWSS- and MWSS-synthesizing recipes that you found in Exercise 11.25.]

∗**11.29** [3/5] When the $c/256$ Demonoid from Figure 11.20 synthesizes its Scorbie splitters, it uses two Snarkmakers and a 90-degree elbow to perform the synthesis via gliders coming from the northeast:



Explain why this method might be preferable or easier than, for example, just using a zero-degree elbow and no Snarkmakers to synthesize it via gliders coming from the northwest.

11.30 [4/5] By separating the Cordership-creation and Cordership-destruction portions of the single-channel glider recipe, make a version of the $c/256$ Demonoid from Figure 11.20 that advances by at least 2^{15} cells per period, instead of just 2^{14}. What is the speed of this new, faster Demonoid?
[Hint: You will have to adjust the timing of Corderships in both the construction and destruction portions of the recipe, and also the timing of some backward glider collisions (e.g., for a Snarkbreaker).]

11.31 [2/5] Show how some additional slow gliders coming from the southwest can be used to clean up the ash objects (i.e., the blinkers, blocks, and loaf) left behind by the boat-stretcher seed from Figure 11.21.

11.32 [1/5] Determine which of the still lifes in the boat-stretcher seed from Figure 11.21 are (a) one-time splitters, and (b) one-time turners. How many synchronized gliders do the one-time splitters produce, and what is their role (i.e., what object(s) do they help synthesize)?

11.33 [5/5] Rebuild the slow Demonoid from Figure 11.16 so that the Snarks and Scorbie splitters are destroyed by the seeding technique of Figure 11.23, rather than via Snarkbreakers and slow xWSS salvos.

11.34 [2/5] Recall the boat wick that was created by the one-glider seed in Figure 11.21.

(a) Find a $2c/3$ fuse that burns through this wick.
[Hint: This fuse is much more common than the $c/3$ fuse that we found in the main text—just look for it by hand.]

(b) Why do you think we used the $c/3$ fuse to build the Speed Demonoid instead of this $2c/3$ fuse?

11.35 [4/5] Use GoL-destroy (`github.com/simeksgol/GoL_destroy`) to find an arrangement of still lifes that can be placed around the color-changing reflector from Figure 7.9 so that it can be destroyed by a single glider.

11.36 [2/5] Find values of the separation parameters n and m that lead to a Speed Demonoid having speed...

(a) $6c/25$,

(b) $c/6$, and

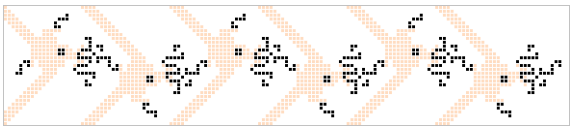
(c) $3c/100$.

11.37 [3/5] Construct a Speed Demonoid that has a trigger separation of $n = 1\,767\,216$ generations and a separation between its two halves of $m = 294\,536$ full diagonals.
[Side note: we explained in the text that such a Speed Demonoid would have speed $3c/14$.]

11.38 [2/5] The universal constructor displayed in Figure 11.26 has a dozen or so unhighlighted still lifes scattered around. What purpose do they serve?

11.39 [3/5] Program the universal constructor displayed in Figure 11.26 to synthesize a heavyweight spaceship (instead of an eater 1) via the slow salvo synthesis provided in Exercise 11.18.

11.40 [3/5] The spiral growth pattern of Figure 11.27 packs gliders in its central diamond together with 27hd spacing. Rebuild a smaller version of that pattern by using this Snark weld that allows for 23hd spacing:



[Hint: The glider recipe itself does not need to change at all. Just extract it from the existing spiral growth pattern, reconstruct the plus-sign-shaped arrangement of Snarks, and then feed the glider recipe back in.]

Free PDF. See conwaylife.com/book for pattern files and print version.



12. The 0E0P Metacell

> If you couldn't predict what [Life] did then probably that's because it's capable of doing anything.
>
> John H. Conway

In the previous chapter, we introduced universal construction and presented several adjustable spaceships based on it. In this chapter, we explore a more complex application of universal construction: a self-reproducing pattern that interacts with nearby copies of itself so as to emulate Conway's Game of Life, or a different cellular automaton, on a much larger and slower scale.

Specifically, we construct a pattern called the **0E0P metacell**,[1] which has the property that if we arrange copies of it on the Life plane then, at a zoomed-out macroscopic scale, it evolves in the same way that the corresponding arrangement of cells would evolve. For example, if we place five copies of this metacell in the Life plane in the formation of a glider, as illustrated in Figure 12.1, then we indeed get a spaceship that travels like a glider (but much more slowly).

A meta-fied pattern like this is $2^{18} = 262\,144$ times as large as the original pattern that it is based on, and it runs $2^{36} = 68\,719\,476\,736$ times as slowly. Indeed, the 0E0P metacell is so much larger and slower than any other pattern that we have seen in this book that evolving the metaglider from Figure 12.1 through four **metageneration**s (i.e., 4×2^{36} generations), to see that it really is a spaceship, would take a couple of years on a modern desktop computer via standard Life simulation algorithms.[2]

While emulating Life within Life perhaps does not seem as remarkable a feat as some of the other things we have achieved over the past three chapters, what really makes the 0E0P metacell shine is

[1] Constructed by Adam P. Goucher from 2014 to November 2018. The acronym "0E0P" stands for "(State) 0 Encoded by 0 Population", in reference to the fact that its "off" state is just a metacell-sized region of empty space—its most remarkable property, which is not shared by any metacell that came before it (see Section 12.9).

[2] The fastest "standard" Life simulation algorithm is **HashLife** [Gos84], which is fast enough to run any of the patterns that we saw earlier in this book—even huge ones like the π calculator and the Gemini spaceship. To run patterns that use huge streams of gliders (like the 0E0P metacell) more efficiently, Adam P. Goucher developed a new **StreamLife** algorithm, but even it would require several months to evolve a metaglider through four metagenerations.

Free PDF. See conwaylife.com/book for pattern files and print versio

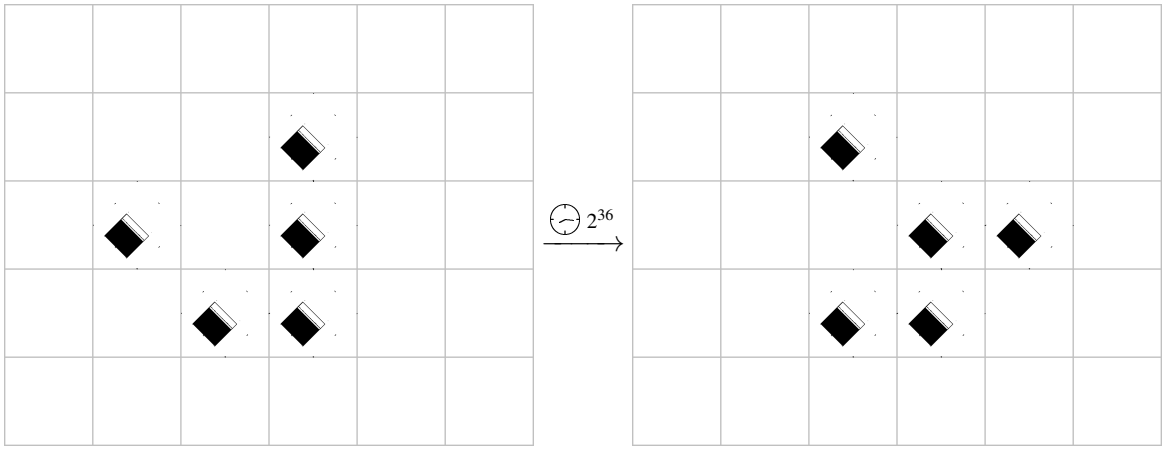


Figure 12.1: A **metaglider** made up of five copies of the 0E0P metacell, each of which is $2^{18} \times 2^{18}$ and logically makes up one "cell". It travels in much the same way as a glider, but 2^{36} times as slowly—and with a step size of 2^{18} cells diagonally instead of just 1 cell diagonally.

that it can actually emulate a huge variety of 2D cellular automata besides Life as well. As a result, any pattern from one of those other cellular automata can be straightforwardly "imported" into Life simply by meta-fying it, thus giving us exotic new types of patterns that were previously not known how to construct.

To illustrate this phenomenon, consider the cellular automaton that has all the same rules as Life, plus the additional rule that a dead cell comes to life if it has exactly 6 live neighbors (i.e., the Life-like cellular automaton with rulestring `B36/S23`). This cellular automaton is called **HighLife**, and it is interesting for the fact that it has a simple **replicator**: a pattern that produces arbitrarily many copies of itself, as illustrated in Figure 12.2.

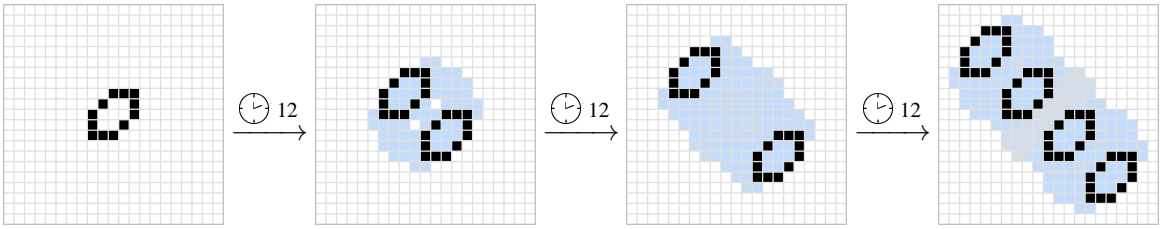


Figure 12.2: A **replicator** in the HighLife (`B36/S23`) Life-like cellular automaton that duplicates itself every 12 generations. Found by Nathan Thompson in February 1994.

After this replicator copies itself the first time, each of its copies attempt to do the same. However, since these copies are right next to each other, two of *their* copies attempt to occupy the same space, and instead annihilate each other (while their other copies are successfully created farther away). This behavior repeats forever: every 12 generations, if there is space for a copy of this replicator then it is made, but two copies being made at the same spot at the same time mutually annihilate. The result is that after n replication cycles (i.e., at generation $12n$), the number of replicators is exactly $2^{B(n)}$, where $B(n)$ is the **binary weight** of n: the number of "1" bits in its binary representation.

Since we're using HighLife's rules that require birth when a dead cell has 6 live neighbors, this is not a valid replicator in Life: when run, it merely degenerates into a configuration of eight blinkers. However, we can "import" its behavior into regular Life by programming the 0E0P metacell to emulate HighLife and then arranging 12 copies of that metacell in the same formation as the 12 cells that make

Free PDF. See conwaylife.com/book for pattern files and print version.

up the replicator from Figure 12.2. This meta-fied replicator is displayed in Figure 12.3.[3] While it is not the first replicator to be constructed in Life (that honor goes to the linear propagator that we mentioned in Section 11.8), this is just the tip of the monumental iceberg of what can be done with the 0E0P metacell.

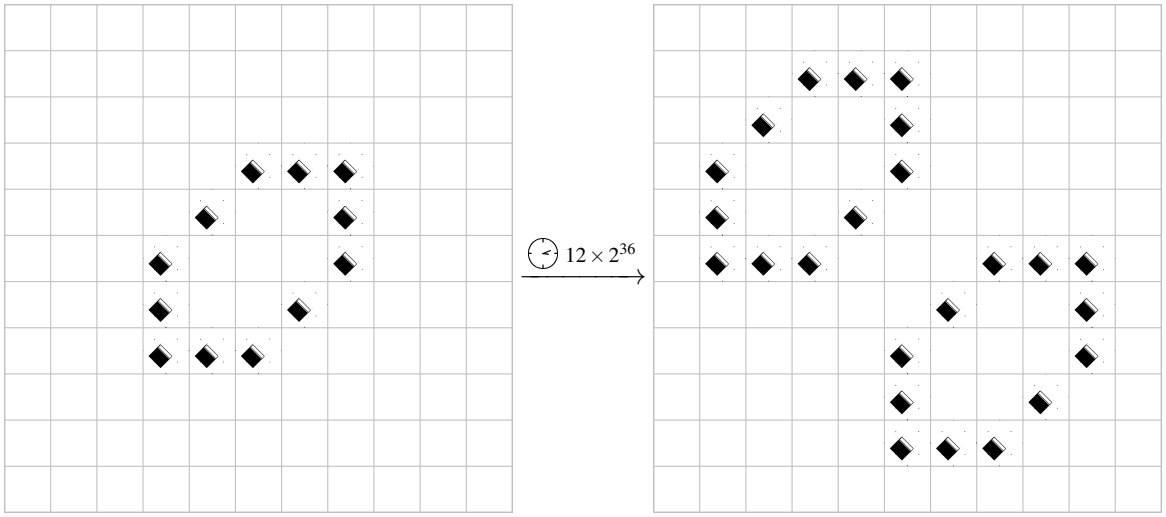


Figure 12.3: A replicator in Conway's Game of Life that is made up of twelve 0E0P metacells, each emulating the HighLife rule (`B36/S23`) from Figure 12.2.

In this chapter, we first explore a few other cellular automata that have exotic patterns that can be implemented in this way in Life via the 0E0P metacell, and then we describe the inner workings of the 0E0P metacell itself.

12.1 Other 2D Cellular Automata

While cellular automata (CA) can act on grids of any dimension and of a variety of different shapes, all of the ones that we consider (and all of the ones that can be emulated by the 0E0P metacell) act on a 2-dimensional square grid. Furthermore, in this section we only consider cellular automata that use two states, which we still refer to as "alive" and "dead", and the Moore neighborhood of Figure 1.1, though we will see in Section 12.2 that the 0E0P metacell can emulate some cellular automata without these two restrictions.

12.1.1 Life-Like (i.e., Outer-Totalistic) Cellular Automata

A 2-state cellular automaton is called **outer-totalistic** if the birth and death rules depend only on the state of the current cell, as well the number of live neighbors that it has.[4] That is, they are exactly the cellular automata that can be described by the `Bx/Sy` rulestring notation that was introduced earlier. An outer-totalistic cellular automaton is said to be **Life-like** if it furthermore satisfies all of the properties that we assumed at the start of this section (i.e., it acts on a 2-dimensional square grid and neighbors are counted according to the Moore neighborhood).

Some Life-like cellular automata have simple patterns that behave unlike any simple patterns that are known in Life itself, as evidenced by the replicator that we saw in Figure 12.2. While that pattern

[3] Slsparse (see `conwaylife.com/wiki/Slsparse`) comes with a Python script called `isotropic_metafier.py` that can metafy patterns like this automatically.

[4] In contrast with **totalistic** cellular automata, in which the birth and death rules depend only on the number of live neighbors including the cell itself.

Free PDF. See conwaylife.com/book for pattern files and print version.

replicates along a single line, there are also Life-like rules that give rise to simple replicators that replicate in multiple directions and fill the whole plane. In fact, in the appropriately named **replicator** rule (B1357/S1357), *every* pattern is a replicator that repeatedly produces copies of itself in all 8 orthogonal and diagonal directions—see Figure 12.4 and Exercise 12.1.

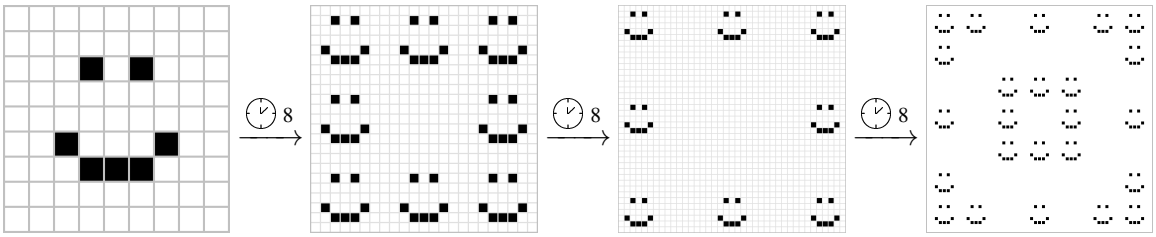


Figure 12.4: In the **replicator** rule (B1357/S1357), every pattern is a replicator that creates copies of itself in the 8 standard directions.

While the patterns of Figures 12.2 and 12.4 replicate in a sawtooth-like fashion—whenever two copies would be created in the same place, they cleanly destroy each other instead, so their populations repeatedly reach new heights and then jump back down below some fixed value—replicators need not behave in this way. For example, consider the rule B12345678/S012345678 in which a cell is born if it ever has at least one live neighbor, and then lives forever. In this rule, a single cell acts as a replicator that repeatedly births all cells in its Moore neighborhood, as illustrated in Figure 12.5.

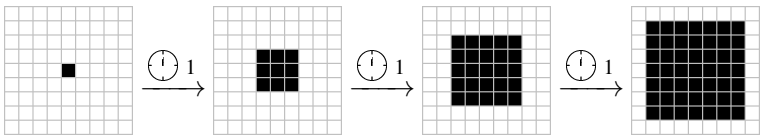


Figure 12.5: A single cell acts as a space-filling replicator in the rule B12345678/S012345678.

In fact, elementary replicators of various shapes and speeds are known in a few dozen different Life-like cellular automata,[5] giving us a wide variety of patterns of this type in Life now, thanks to the 0E0P metacell. Somewhat less common are patterns that grow in other ways, like the spiral-growth pattern for the rule B34568/S15678 that is displayed in Figure 12.6 (compare with the spiral growth pattern that we constructed in Life back in Figure 11.27).

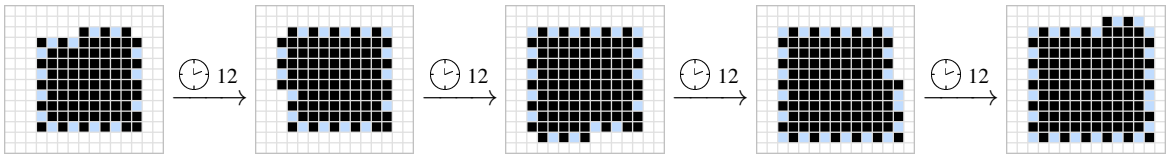


Figure 12.6: A pattern that exhibits spiral growth in the B34568/S15678 rule. Found by Dean Hickerson in June 2006.

12.1.2 Isotropic (but Non-Outer-Totalistic) Cellular Automata

A cellular automaton is called **isotropic** if the cell transition rules are invariant under rotations and reflections, and it is called **non-isotropic** otherwise. That is, an isotropic cellular automaton may take into account the *relative* positions of neighboring cells, but not their *absolute* positions. Every outer-totalistic cellular automaton is isotropic, but the converse is not true, as shown in Figure 12.7.

[5] See www.ics.uci.edu/~eppstein/ca/replicators/index.html for a partial list.

Free PDF. See conwaylife.com/book for pattern files and print version.



Figure 12.7: In an outer-totalistic cellular automaton, the central cells (displayed in orange) in the leftmost four configurations must all evolve in the same way, since they have the same number of live neighbors (2). In an isotropic cellular automaton, the central cells must evolve in the same way in only the *three* leftmost configurations, since their arrangements of neighbors are rotations and/or reflections of each other. In a non-isotropic cellular automaton, the central cells can evolve in different ways in all five configurations.

Because of the extra flexibility afforded by isotropic cellular automata (there are a whopping 2^{102} isotropic 2-state CA on a 2D square grid, versus "just" 2^{18} outer-totalistic ones), they often contain elementary patterns that appear completely alien when compared to those from Life. For example, consider the CA that has the same rules as Life, but with four isotropic modifications: two to its birth conditions and two to its survival conditions, as illustrated in Figure 12.8. This CA's rulestring is `B3-j6i/S23-c4i`, where the substrings "`-j`", "`6i`", "`-c`", and "`4i`" correspond to these four new rules (in the same order as they are displayed in Figure 12.8). The details of how to construct rulestrings for arbitrary isotropic cellular automata are somewhat involved, so we leave them to Appendix B.6.

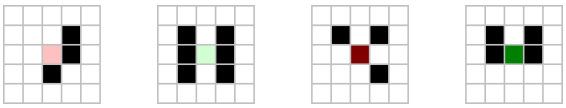


Figure 12.8: The four differences (up to rotation and reflection) between Life (`B3/S23`) and the isotropic cellular automaton `B3-j6i/S23-c4i`. The central cell in the first (i.e., leftmost) configuration is *not* born (whereas it would be in Life), the central cell is born in the next configuration, the central cell dies in the third, and the central cell survives in the fourth.

Our interest in this cellular automaton comes from the pattern displayed in Figure 12.9, which is an example of a **reflectorless rotating oscillator** (or **RRO** for short): an oscillator with the property that one of its phases is a rotation of another one, and two non-interacting copies of the oscillator can combine so as to produce an oscillator with period half as large.[6] Informally, RROs travel around in a loop like a spaceship that periodically turns a corner (they are even sometimes called **looping spaceships**). This is very unlike anything that we have seen in Life—all oscillators that we have seen either stay mostly stationary (like all of the oscillators in the first 8 or 9 pages of Chapter 3), or move around a path with the help of other stationary pieces (like glider loops, Herschel tracks, and the pi-heptomino hassler from Figure 3.19(b)).

Another exotic type of pattern that appears in isotropic cellular automata, but for which no elementary example is known in their Life-like brethren, is a **spaceship made of spaceships** (or **SMOS** for short): a spaceship that works by colliding two or more other spaceships with each other.[7] One of the simplest such patterns occurs in the (admittedly very *not* simple) cellular automaton

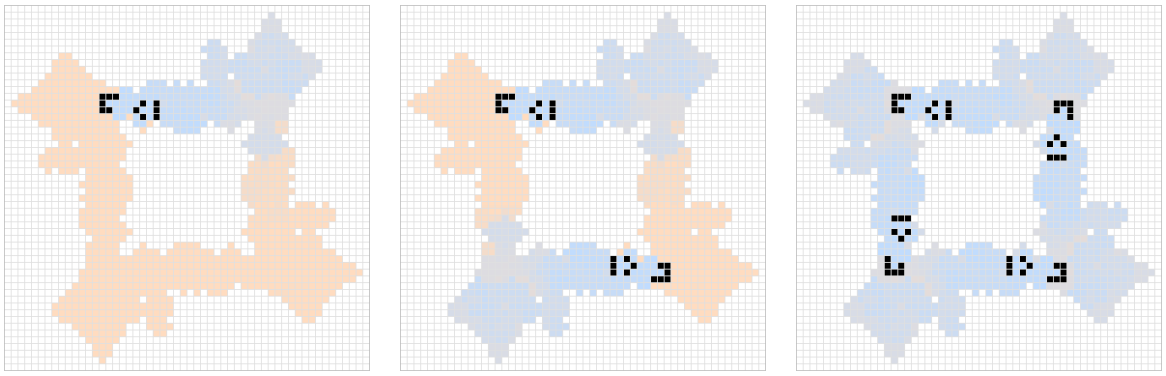
`B3acijn4jktwyz5ijr6-en7c/S2aen3-aceq4acijqty5cikr6ak7c`.

Under this rule, a glider functions as a $c/4$ diagonal spaceship in the exact same way as in Life, but if two gliders collide together just right then they push each other in such a way as to produce

[6] This final "two copies..." requirement enforces the "reflectorless" part of the name: a single glider in a square Snark loop is an oscillator whose phases are rotations of other phases, but two copies of this loop only produce an oscillator with half the period if we overlap the Snarks (thus violating the "non-interacting" part of the definition of an RRO).

[7] The Demonoids that we constructed in the previous chapter are *mostly* made up of gliders. However, they are not spaceships made of spaceships since there is no phase in which they consist *entirely* of gliders.

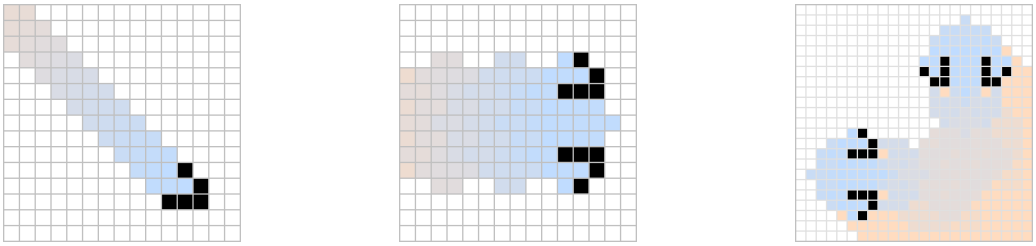
Free PDF. See conwaylife.com/book for pattern files and print version.



(a) One copy, period 200. **(b)** Two copies, period 100. **(c)** Four copies, period 50.

Figure 12.9: A **reflectorless rotating oscillator** in the isotropic cellular automaton `B3-j6i/S23-c4i`. Either one, two, or four copies of the oscillator can be placed along the same path, producing oscillators with periods 200, 100, and 50. Found by Justin Tang in January 2020.

a $4c/17$ orthogonal spaceship (see Figure 12.10). Even more remarkably, if two of these $4c/17$ spaceships collide together just right then they produce a $c/44$ diagonal spaceship—a **spaceship made of spaceships made of spaceships** (**SMOSMOS**).



(a) A ($c/4$ diagonal) glider. **(b)** A $4c/17$ orthogonal SMOS. **(c)** A $c/44$ diagonal SMOSMOS.

Figure 12.10: In the rule `B3acijn4jktwyz5ijr6-en7c/S2aen3-aceq4acijqty5cikr6ak7c`, (a) a glider functions as a $c/4$ diagonal spaceship in the usual way. However, when it collides with itself it turns into (b) a $4c/17$ orthogonal spaceship (found by ConwayLife.com forums user "Saka" in August 2017), and when *that* spaceship collides with itself it turns into (c) a $c/44$ diagonal spaceship (found by ConwayLife.com forums user "FWKnightship" in August 2019).

12.1.3 Non-Isotropic Cellular Automata

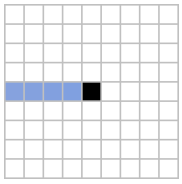
If we go one step further and consider non-isotropic cellular automata, we can make even just a single cell behave in an extraordinary variety of ways. For example, we can define a rule with the property that cells never survive, and they are only born if they have a single live cell directly to their left.[8] In this rule, a single cell is a spaceship that travels orthogonally at lightspeed (see Figure 12.11(a)).

We can also consider the slightly more exotic cellular automaton in which cells are only born if they have no neighbors or a single neighbor to their southeast, and they only survive if they have neighbors everywhere *except* for to their north and northwest.[9] In this rule, a single cell acts as a $(1,2)c/2$ knightship, as illustrated in Figure 12.11(b). It is similarly possible to construct rules in

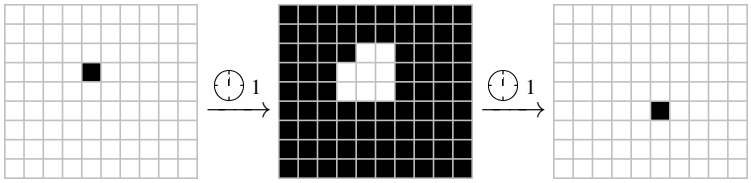
[8] This cellular automaton is described by the rulestring
`MAPAAAAAIAAAAA

which a single cell is a spaceship that travels at many other speeds and slopes—see Exercise 12.2.

By changing the evolution rule even more, a single cell can also act in a variety of other exotic ways. For example, consider the cellular automaton that is defined by all cells surviving, and cells being born if and only if they have exactly one live cell directly to their northeast or northwest, as in Figure 12.12(a).[10] In this rule, a single cell generates better and better approximations of the Sierpiński triangle, as illustrated in Figure 12.12(b).



(a) A single-cell lightspeed orthogonal spaceship.

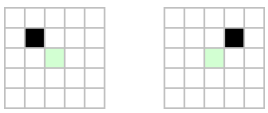


(b) A single-cell knightship travelling at $(1,2)c/2$. In this rule, a dead cell is born when it has no live neighbors, so the background array of cells alternates between dead and alive in even and odd generations.

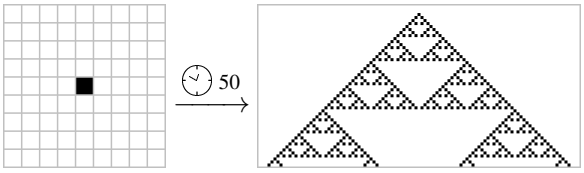
Figure 12.11: Non-isotropic rules are general enough that they can turn a single cell into a spaceship with a wide variety of speeds and slopes.

Indeed, if we restrict non-isotropic cellular automata to a single dimension (e.g., by having every cell survive and having births only occur to the south of live cells, as we did in Figure 12.12(b)) then we get exactly what are called **elementary cellular automata** [Wol02]. There are $2^8 = 256$ cellular automata of this type, and the Sierpiński-triangle-generating rule works by emulating the one called **Rule 18**.

There are 2^{512} different 2D not necessarily isotropic cellular automata, and the 0E0P metacell can emulate the 2^{511} of them that send a dead cell with no live neighbors to a dead cell. It can thus be used to embed all of the patterns from this section into Life, except for the knightship from Figure 12.11(b), though at a much larger and slower scale. Of particular note is the fact that this method gave the first explicit construction of a spaceship made of spaceships in Life (and thus the first SMOSMOS in Life as well). This method also gave the first reflectorless rotating oscillator in Life, but there are minor technicalities that must be overcome in this case (see Exercise 12.3).[11]



(a) Cells always survive, but they are only born in the two configurations shown here (in the central location marked in green).



(b) A single-cell pattern creating the Sierpiński triangle.

Figure 12.12: In the non-isotropic rule in which all cells survive, but are only born as in (a), a single cell produces the Sierpiński triangle as in (b).

12.2 Rule Emulation

While the 0E0P metacell can emulate any 2-state Moore-neighborhood cellular automaton in which a dead cell with all dead neighbors stays dead, it does not do so directly. Indeed, building a 0E0P

[10] This cellular automaton is described by the rulestring
`MAPAAD//wAA//+AAP//AAD//wAA//8AAP//AAD//wAA//+AAP//AAD//wAA//8AAP//AAD//wAA//8AAP//AAD//w`.

[11] However, ConwayLife.com forums user "Goldtiger997" used single-channel construction techniques to "directly" construct an SMOS and an RRO in Life a bit later, in August 2021.

Free PDF. See conwaylife.com/book for pattern files and print version.

metacell that directly emulates Life would be highly nontrivial for (at least) two reasons:

- **Quantity of neighbors**: each metacell would need to be able to construct a copy of itself in up to eight different locations. If both the north and east neighbors are present, for example, it may be difficult to maneuver a construction arm to build the northeast neighbor.
- **Survival**: a cell can live for multiple generations, so its logic circuitry would need to be reusable. Reusable circuitry is considerably more expensive than single-use circuitry: compare, for instance, the size of a boat and a Snark—the smallest known one-time turner[12] and reusable stable reflector, respectively.

To get around these two problems, the 0E0P metacell instead emulates an 8-state von-Neumann-neighborhood CA in which every cell dies in every generation (but new cells may be born, so the pattern as a whole need not die).[13] In a cellular automaton of this type, patterns evolve according to a checkerboard pattern—if they live entirely on one color of the checkerboard in one generation, then they will live entirely on its other color in the next (see Figure 12.13 and note that we rotated the square grid by 45 degrees so that its cells match the orientation of the diamond-shaped 0E0P metacell). This checkerboard pattern is important, as it ensures that the 0E0P metacell's four diagonal neighbors are dead (i.e., empty), giving it lots of space to copy itself if necessary.

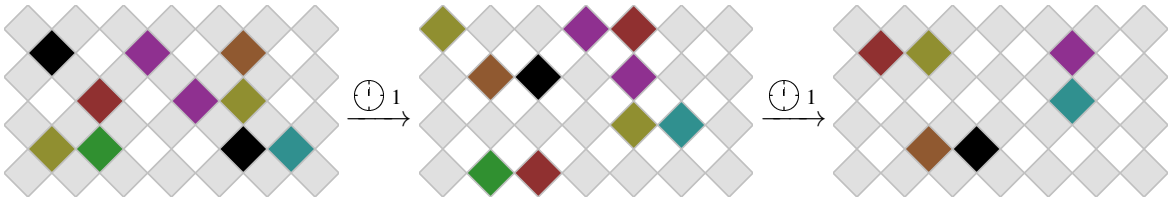


Figure 12.13: An 8-state cellular automaton using the von Neumann neighborhood that is of the type that the 0E0P metacell can emulate. The "dead" state is displayed in a white and light gray checkerboard pattern. The other 7 states are displayed in black and various other colors, which alternate being on the white and light gray portions of the checkerboard pattern from one generation to the next.

Despite the seemingly limited nature of these cellular automata, the fact that they have 8 states makes them general enough to emulate arbitrary 2-state Moore-neighborhood cellular automata at half speed. To see how this works, suppose we are given a particular 2-state CA that we want to emulate. If we represent the "dead" and "alive" cells of that CA by the numbers 0 and 7, respectively,[14] then we can encode the CA by a function

$$M : \{0,7\}^9 \to \{0,7\}$$

that describes how a cell changes from alive to dead, or vice-versa, depending on the state of itself and its 8 neighbors.

Our goal is to construct an 8-state von-Neumann-neighborhood CA, in which the central cell always dies, that emulates M at half speed. If we label the states of this cellular automaton by $\{0,1,2,3,4,5,6,7\}$ (where 0 is the background "dead" state, as usual), then this is equivalent to finding a function

$$N : \{0,1,2,3,4,5,6,7\}^4 \to \{0,1,2,3,4,5,6,7\}$$

[12] We originally showed that a boat is a one-time turner in Exercise 5.31(a).

[13] In other words, every pattern is a phoenix in these cellular automata.

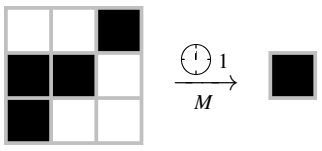
[14] Yes, 7 seems like a weird choice. It will make more sense shortly.

Free PDF. See conwaylife.com/book for pattern files and print version.

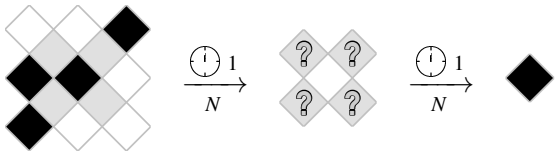
with the property that

$$M\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} = N\begin{pmatrix} N\begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} & N\begin{pmatrix} a_{1,2} & a_{1,3} \\ a_{2,2} & a_{2,3} \end{pmatrix} \\ N\begin{pmatrix} a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{pmatrix} & N\begin{pmatrix} a_{2,2} & a_{2,3} \\ a_{3,2} & a_{3,3} \end{pmatrix} \end{pmatrix} \tag{12.1}$$

whenever $a_{1,1}, a_{1,2}, \ldots, a_{3,3} \in \{0,7\}$. That is, we want this 8-state CA to have the property that, if we apply it to 2×2 grids of cells (i.e., rotated von Neumann neighborhoods) independently, then after 2 iterations we get exactly the same result as if we were to apply the original 2-state CA rule to 3×3 grids of cells (i.e., Moore neighborhoods), as in Figure 12.14.



(a) A 3×3 configuration that leads to the central cell surviving in Life.



(b) In the emulating CA, each 3×3 grid evolves in the same way over the course of 2 generations, by evolving 2×2 grids individually.

Figure 12.14: The 8-state CA that we construct will work by acting on 2×2 grids of cells in such a way that applying it twice results in the same evolution (at half speed) as applying the original 2-state CA to 3×3 grids of cells.

While there are many ways that we could construct N, one reasonably simple approach is to ensure that if all four of its inputs comes from $\{0,7\}$ then its output is in $\{0,1,2,3,4,5,6\}$, and vice-versa. That is, we design the 8-state CA so that patterns in even generations consist entirely of cells in the states 0 and 7, thus emulating the "dead" and "live" cells in a 2-state CA, whereas in odd generations they consist entirely of cells in the states 0 through 6, which are just used as helper states for reconstructing the proper configuration in even generations.

We first specify how N should act when all of the inputs that it receives are either 0 or 7. Since there are only $2^4 = 16$ possible combinations of inputs in this case, we can list how N acts on them explicitly:

$$\begin{aligned} &N(0,0,0,0) = 0, && N(0,0,0,7) = 1, && N(0,0,7,0) = 2, && N(0,0,7,7) = 3, \\ &N(0,7,0,0) = 4, && N(0,7,0,7) = 5, && N(0,7,7,0) = 6, && N(0,7,7,7) = 4, \\ &N(7,0,0,0) = 6, && N(7,0,0,7) = 4, && N(7,0,7,0) = 3, && N(7,0,7,7) = 0, \\ &N(7,7,0,0) = 1, && N(7,7,0,7) = 6, && N(7,7,7,0) = 5, && N(7,7,7,7) = 2. \end{aligned} \tag{12.2}$$

These 16 evolution rules are displayed in Figure 12.15, and they do not depend at all on the rule that we are trying to emulate (i.e., these input-output combinations of N are the same no matter what M is).

The reason for choosing these seemingly random transition rules is that they lead to the function

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \mapsto \begin{pmatrix} N\begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} & N\begin{pmatrix} a_{1,2} & a_{1,3} \\ a_{2,2} & a_{2,3} \end{pmatrix} \\ N\begin{pmatrix} a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{pmatrix} & N\begin{pmatrix} a_{2,2} & a_{2,3} \\ a_{3,2} & a_{3,3} \end{pmatrix} \end{pmatrix} \tag{12.3}$$

from $\{0,7\}^9$ to $\{0,1,2,3,4,5,6\}^4$ being injective (i.e., each output of the function corresponds to a *unique* input).[15] That is, given any 2×2 arrangement of states 0, 1, ..., 6, we can figure out which

[15]However, the choices we made in Equation (12.2) (or equivalently, Figure 12.15) are not the only ones that lead to this function being injective. This collection of outputs, and many others, can be found by computer search—see Exercise 12.9.

Free PDF. See conwaylife.com/book for pattern files and print version.

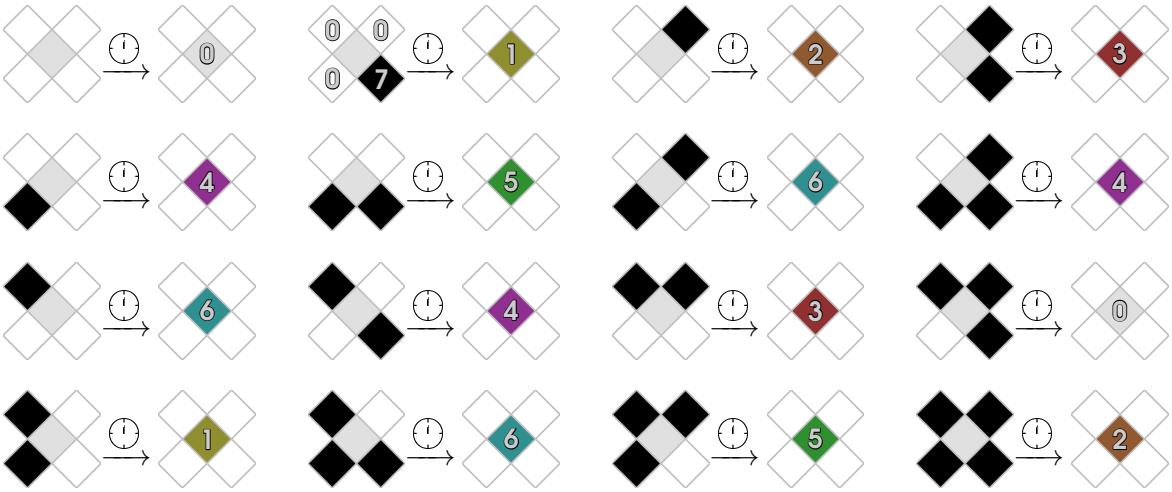


Figure 12.15: The rule for transitioning from even to odd generations in the 8-state von-Neumann-neighborhood CA from Equation (12.2). The states 0 and 7 correspond to "dead" and "alive" in Life and are thus displayed in white and black, respectively. The states 1, 2, 3, 4, 5, and 6 are displayed in yellow, orange, red, magenta, green, and aqua, respectively.

3×3 arrangement of 0 and 7 states (if any) gave rise to it. We can thus define N on inputs from $\{0,1,2,3,4,5,6\}^4$ so that Equation (12.1) holds (i.e., the cellular automaton defined by N emulates the cellular automaton defined by M at half speed).

The resulting function N is typically quite messy to write out in full detail (as it would just be an explicit list of how the $2^9 = 512$ possible arrangements of 2×2 grids of states 0, 1, ..., 6 should evolve), even if the rule M describes a relatively simple cellular automaton like Life. However, it can be constructed straightforwardly by a computer script,[16] and the emulation of a glider in this way is illustrated in Figure 12.16.

The 0E0P metacell can be programmed to emulate any of the 8^{8^4-1} possible zero-preserving 8-state von-Neumann-neighborhood cellular automata in which every cell dies every generation (not just the ones that emulate one of the 2^{2^9-1} zero-preserving 2-state Moore-neighborhood CAs that we are actually interested in). It takes 2^{35} generations for the 0E0P metacell to run one generation of the 8-state rule (emulated at a 45-degree angle, as in the figures of this section), and therefore 2^{36} generations to emulate one generation of the corresponding 2-state Moore-neighborhood rule (in the usual orientation).

12.3 Overview of the Metacell

The 0E0P metacell works by using the single-channel glider synthesis techniques of Chapter 11 to construct copies of itself in neighboring locations. It starts by constructing a copy to the southeast, northeast, northwest, and southwest (in that order, and only if no metacells are already present in those locations), and then it sends information about its current state to those neighboring metacells so that the proper rule is emulated.

In order to implement this behavior, the 0E0P metacell is made up of three components (see Figures 12.17 and 12.18), which are constructed by parent metacells one at a time:

- The outermost part is the **shell**, which directs information about the states of neighboring metacells. It has exact 90-degree rotational symmetry, consisting of four spiral arms which

[16] One such script is available at conwaylife.com/forums/viewtopic.php?p=36112#p36112.

Free PDF. See conwaylife.com/book for pattern files and print version.

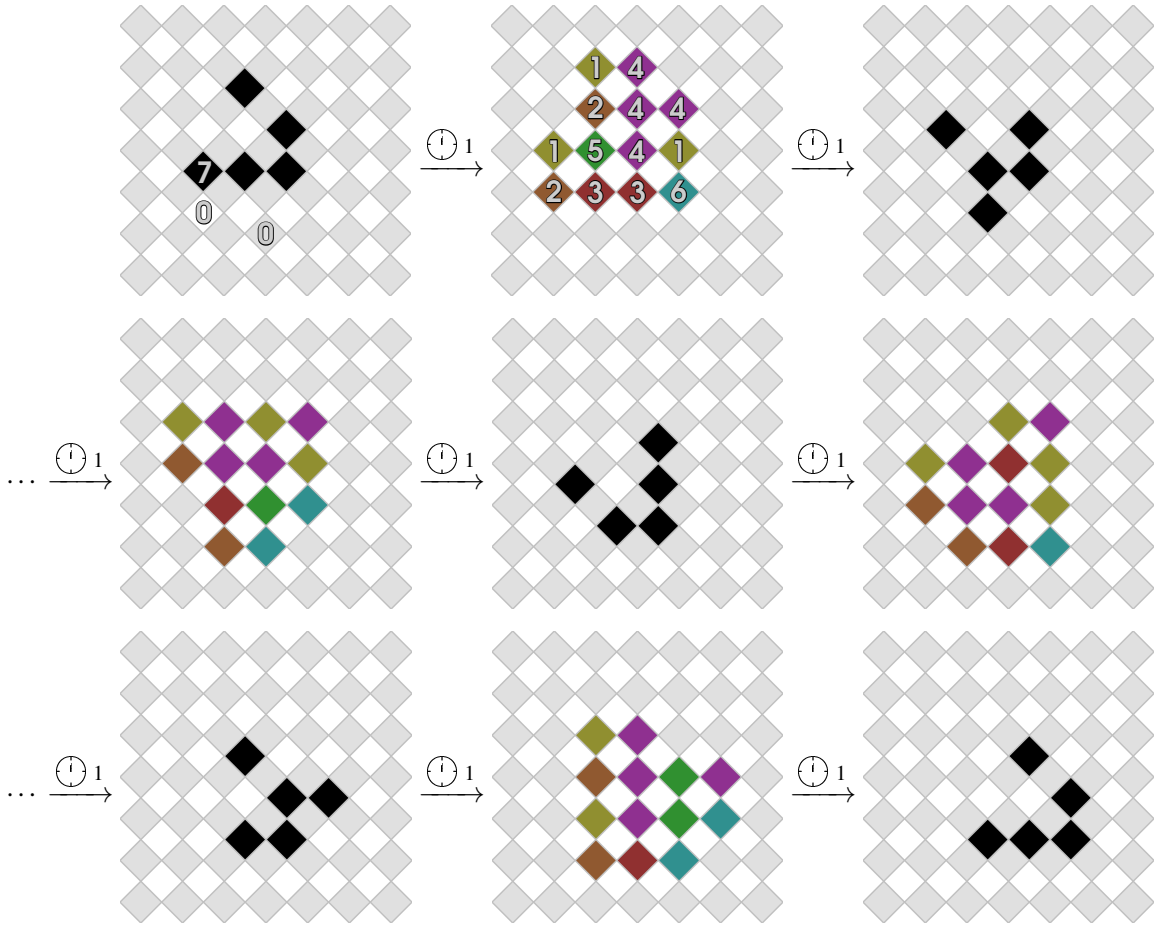


Figure 12.16: A glider from Life being emulated at half speed via an 8-state cellular automaton using the von Neumann neighborhood (rotated by 45 degrees). Uses the same state coloring as in Figure 12.15, which describes the transitions from even to odd generations. The transitions from odd to even generations are encoded as an explicit list of which 2×2 arrangements of states $0, 1, \ldots, 6$ should lead to their central cell being born.

propagate gliders inwards. Only one of these arms is actually used; the other three exist only to enforce the rotational symmetry.

- Inside the shell is the **kernel**, which does not have any symmetry constraints. The south corner of the kernel contains a clock gun for regulating the metacell's lifecycle and logic circuitry for computing the state of the metacell based on the states of its four neighboring cells. The kernel also contains an output path, shown in magenta in Figure 12.17, which can connect to one of the four construction arms (dashed) or to the input shell of one of the four neighbors.
- At its center is the largest region of the metacell: its **nucleus**, which is a huge glider loop with a period of exactly 2^{29}. It is populated by over three million gliders, consisting of a complete single-channel glider synthesis of the 0E0P metacell, as well as a state lookup table that encodes the rule that the metacell is emulating.

We describe how these three components of the metacell function throughout its lifecycle in more detail in the next three sections, and we then talk about how they are constructed by parent metacells in Section 12.7. Due to the complexity and size of the 0E0P metacell, it is extremely important that the reader does not just read through the next few sections, but also uses Life simulation software to

Free PDF. See conwaylife.com/book for pattern files and print version.

explore the metacell's different components as we describe them.[17]

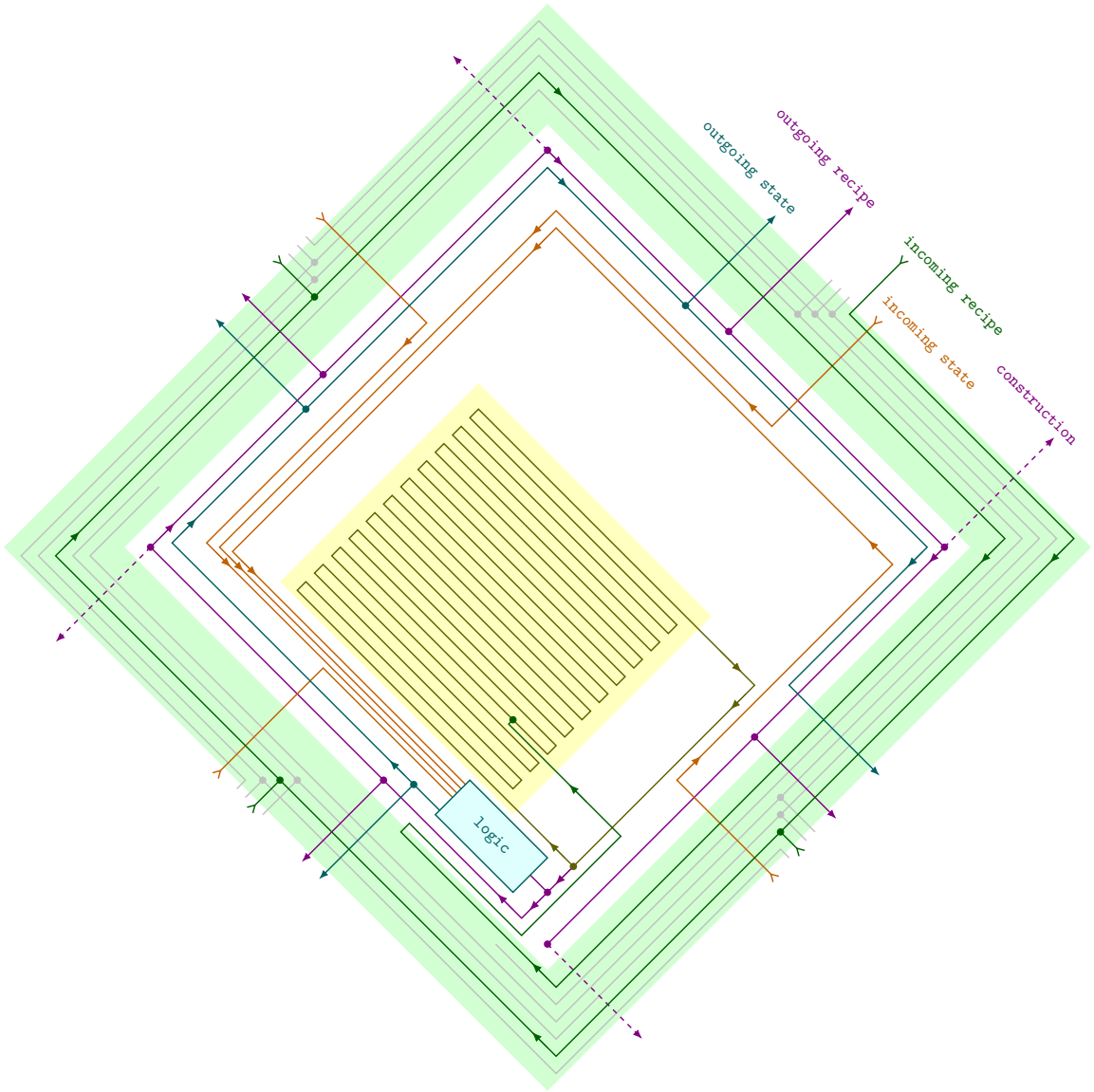


Figure 12.17: A schematic of the 0E0P metacell. The **nucleus**, highlighted in yellow, houses a single-channel glider sequence containing over 3 million gliders, which encode the rule being emulated as well as the construction of the 0E0P metacell itself. When the 0E0P is being constructed, its symmetrical outer **shell** (highlighted in light green) takes in the single-channel glider sequence from any of its neighboring parents (along the path displayed in dark green) and injects it into the nucleus. The unhighlighted region in the middle is the **kernel**, which contains an output path for a copy of the single-channel glider sequence (displayed in magenta), as well as input and output paths along which parent metacells can tell child metacells what state they are in (displayed in orange and aqua, respectively).

[17] Unfortunately, no Life simulation software is currently fast enough to run the entire 0E0P metacell "quickly". However, bits and pieces of it can be evolved to see how they behave, and numerous snapshots that show the metacell's state at different timestamps are provided in Section 12.8 and at conwaylife.com/book/0e0p.

Free PDF. See conwaylife.com/book for pattern files and print version.

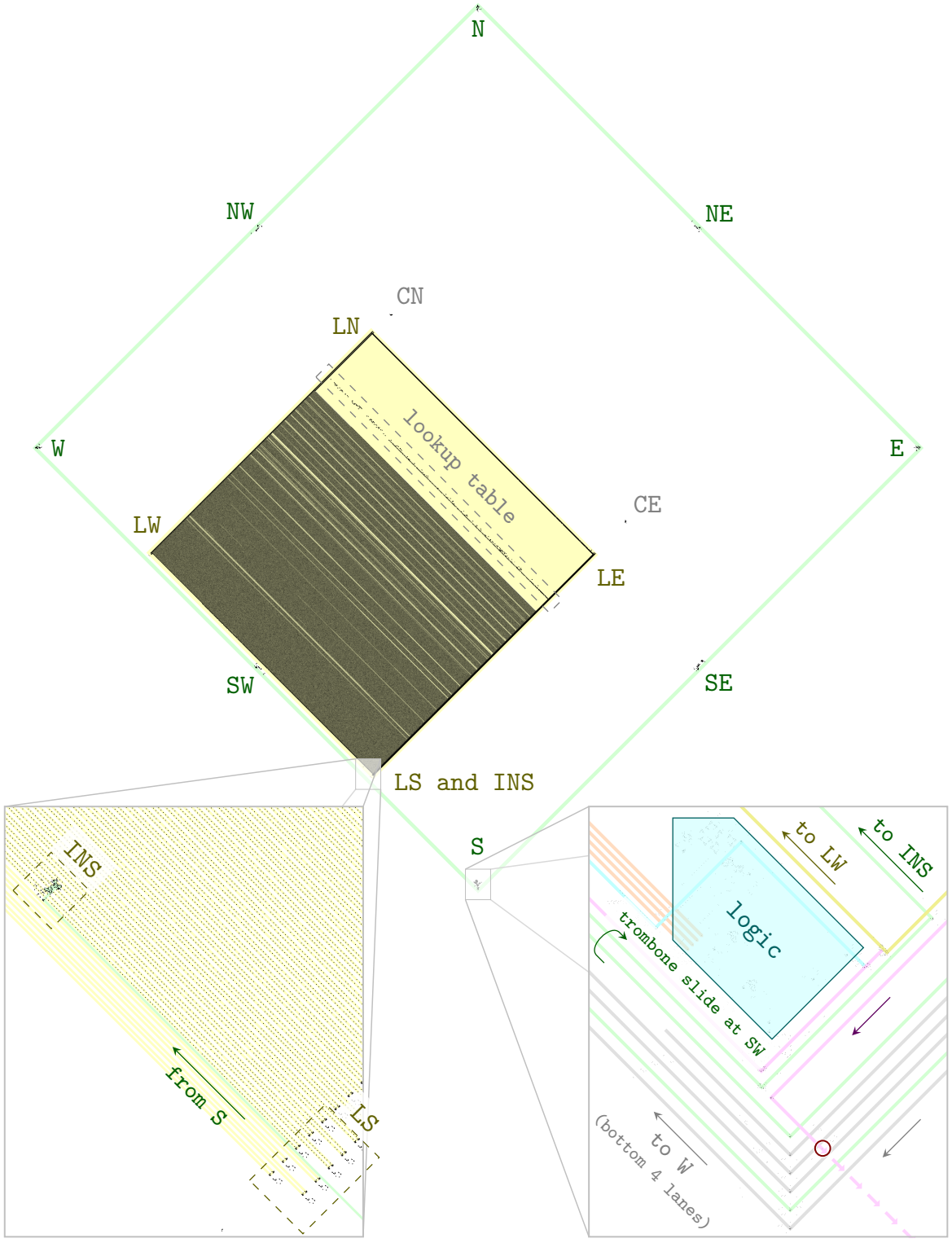


Figure 12.18: The 0E0P metacell itself, which implements the schematic of Figure 12.17. It contains a significant amount of empty space between 15 points of interest. The shell (highlighted in green) goes through points `SW`, `W`, `NW`, `N`, `NE`, `E`, `SE`, and `S`, as do the points where the glider paths are reflected in the kernel. The logic circuitry is also located at point `S`, and an elbow block that is used to construct a new metacell to the southeast is circled in red. The nucleus (highlighted in yellow) has corners at points `LW`, `LN`, `LE`, and `LS`. The point `INS` is used to insert gliders into the nucleus during construction. The points `CN` and `CE` are not displayed in the schematic of Figure 12.17—they contain temporary circuitry that is used to construct the walls of the nucleus.

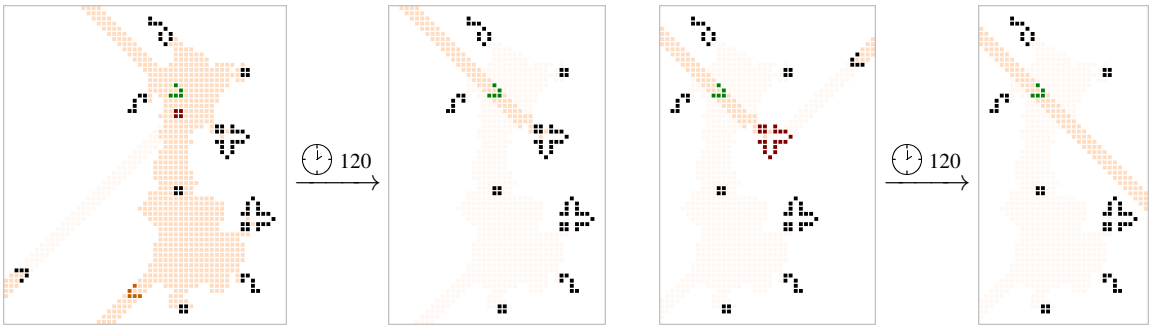
Free PDF. See conwaylife.com/book for pattern files and print version.

12.4 The Shell and Lane-Switching Circuitry

The rotationally symmetric shell that makes up the outermost edges of the 0E0P metacell is the simplest of its three components. At each of the locations marked `SW`, `NW`, `NE`, and `SE` in Figure 12.18, there are four recipe input lanes (in green and gray in Figure 12.17). However, only one of those four lanes is ever actually used (in green)—the one that connects to the recipe output lane coming in from the kernel of the neighboring metacell (in solid magenta), if it exists.[18] The remaining input lanes (in gray) are unused, and only exist to enforce the shell's fourfold rotational symmetry.

After the metacell has been constructed, these input lanes read in a copy of the 0E0P's single-channel construction recipe and state lookup table from its neighbors. That (extremely long) sequence of gliders revolves clockwise all the way around the spiral arm of the shell, until it is finally injected into the nucleus (at the location called `INS` in Figure 12.18) at the 0E0P's center.

To make sure that just one copy of the single-channel glider sequence is allowed to enter the shell, regardless of how many neighbors try to send in that glider sequence, the 0E0P metacell strategically deletes part of its own circuitry after the nucleus is populated (as well as at numerous other times throughout its lifecycle). By using an external glider to delete a single block from a syringe, as demonstrated in Figure 12.19(a), the 0E0P can change a syringe-based-reflector into a device that simply eats a glider stream. Another external glider could also be used to destroy the syringe's eater 2, thus unblocking the stream as demonstrated in Figure 12.19(b), if desired. When external gliders are needed for these kinds of switching and control operations, they are provided by the metacell's control clock gun, which we discuss in the upcoming Section 12.5.1.



(a) A single glider coming in from the southwest destroys a block, thus "shutting off" the syringe-based reflector.

(b) A glider coming in from the northeast destroys the eater 2, thus unblocking the glider stream.

Figure 12.19: Some ways of deleting pieces of a syringe (displayed in red) to change the path of a glider stream.

12.5 The Kernel

The asymmetric kernel contains the paths that are used to feed the 0E0P's single-channel recipe into its neighbors (in magenta in Figure 12.17). Along these paths, gliders spiral clockwise-and-outwards, and then bypass the shell and do one of two things: go into a construction lane (in dashed magenta) to construct a neighboring metacell, or go into a recipe output lane (in solid magenta) that connects to a recipe input lane of an already-constructed neighboring metacell.

One important aspect of how the recipe output and input lanes of neighboring metacells connect to each other is that, irrespective of the orientation of the neighbors that are communicating, the single-channel glider sequence performs exactly 6 spiral quarter-turns to get from the south corner

[18] This design is similar to that of a circuit on a motherboard—you can think of each side of the 0E0P metacell as having 8 "pins" (4 inputs pins and 4 output pins) that connect to the 8 pins of neighboring metacells.

Free PDF. See conwaylife.com/book for pattern files and print version.

of the parent metacell to the south corner of the child metacell. This is how child metacells always end up in the correct phase and orientation after they are constructed. There are also some trombone slides of different lengths on different edges of the kernel, designed to compensate for the "Olympic running track" effect—the fact that outer lanes of the spiral are slightly longer than inner lanes. These trombone slides ensure that the communication time from the parent to each of the four children is exactly the same.

In order to control which recipe output lane or construction lane the single-channel glider sequence enters, the 0E0P periodically destroys some of the kernel's Snarks. We saw one method of doing this back in Figure 11.23(a)—a single glider can be used to destroy a Snark (with the help of 3 extra pre-placed still lifes). The 0E0P metacell makes use of Snark-destroying reactions like this one[19] to redirect copies of its single-channel glider sequence to eight different output lanes over the course of its lifespan. In order,[20] they are:

1a) The southeast construction lane (in dashed magenta in Figure 12.17), to construct the child metacell to the southeast.

1b) The southeast recipe output lane (in solid magenta), to copy the glider sequence into the (now constructed) southeast child's nucleus.

2a) The northeast construction lane (in dashed magenta), to construct the child metacell to the northeast.

2b) The northeast recipe output lane (in solid magenta), to copy the glider sequence into the (now constructed) northeast child's nucleus.

3a) – 4b) The northwest construction lane, then the northwest recipe output lane, and so on counterclockwise.

12.5.1 The Control Clock Gun

The lane-switching mechanisms described earlier are implemented by a **control clock gun** that is located at the top-left corner of the "`logic`" component in Figures 12.17 and 12.18.[21] This gun sends out a single glider every 2^{29} generations, thus matching exactly the period of the glider loop contained in the nucleus. It is made up of a p256 gun attached to a sequence of 21 semi-Snarks, much like the clock gun of Figure 9.5 that was used by Chapter 9's universal computer (though that gun used quadri-Snarks instead).[22]

The gliders that are released by the control clock gun travel through sequences of one-time turners and splitters that are interspersed throughout the rest of the 0E0P's circuitry, so as to trigger the appropriate lane-switching mechanisms at each step of its lifecycle. This method is illustrated at a much smaller scale in Figure 12.20, where a clock gun is used to change what happens to the path of a single-channel glider sequence three times.

In the actual 0E0P, the control clock gun's output is redirected many more times than this, and the one-time circuitry involved is much more complex. Furthermore, it also has a period 2^{25} output lane that spends most of its time blocked, but is periodically unblocked so that it can send out a glider with different timing than the main period 2^{29} output lane. Jump ahead to Figure 12.24 to see what the period 2^{29} output lane actually looks like.

[19] But not *exactly* this one—see Exercise 12.10.

[20] The single-channel glider sequence really does have to be sent along these 8 paths one after another. If we just used glider duplicators to send it along all 8 paths at the same time, there could be unwanted collisions when constructing child metacells (e.g., if two parent metacells tried to create a child in the same location at the same time).

[21] There are actually four clock guns in the 0E0P metacell. This "control" clock gun is active for the entirety of the metacell's lifecycle, while the other three are all temporary.

[22] The 0E0P's clock guns use the color-changing semi-Snark of Figure 7.18(b), since it is Spartan and is thus easier to construct via a single-channel glider recipe than quadri-Snarks.

Free PDF. See conwaylife.com/book for pattern files and print version.

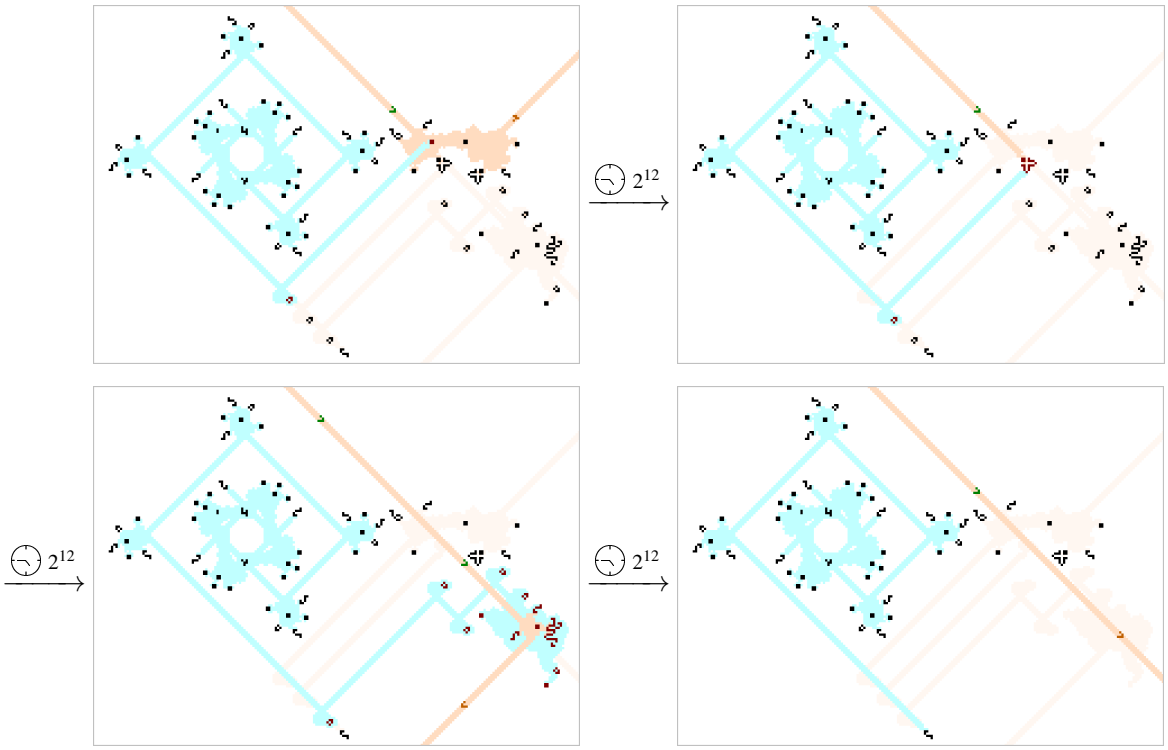


Figure 12.20: A period $2^{12} = 4096$ clock gun (highlighted in aqua) using one-time turners and the reactions from Figures 11.23(a) and 12.19 to change the path of the single-channel glider stream coming from the northwest. Initially, that stream is reflected to the northeast, but the first glider from the clock gun blocks the reflector. Its second glider unblocks it and redirects the stream to the southwest via a Snark. Its third glider destroys the Snark, so the stream passes straight through to the southeast.

12.5.2 Logic Circuitry and State Computation

The kernel also contains, at its southern corner (i.e., the location called S in Figure 12.18), the logic circuitry that is used to compute the metacell's state from the states of its four neighboring parents. Metacells' states are transmitted via sequences of gliders, and are stored via *absences* of blocks, as follows:

- **Transmission**: When a parent metacell sends its state to its children, it does so via a sequence of 0 or 2–8 gliders, representing state 0 or 1–7, respectively.[23]
- **Temporary storage**: When children receive their parents' states, the first glider (if it exists) is used to unblock their control clock gun, while the remaining 1–7 gliders destroy 1–7 blocks. The children thus temporarily store their parents' states in the *absence* of those blocks.
- **Internal transmission**: After a child computes its state from its parents, it initially appears as a unary sequence of 0–7 gliders. One-time circuitry is used to duplicate one of those gliders (if they exist), resulting in a sequence of 0 or 2–8 gliders.
- **Long-term storage**: Those 0 or 2–8 gliders are aimed at a sequence of 8 blocks at the western-most edge of the child's "S" corner (see the upcoming Figure 12.47 for the precise location). Its state is thus stored in the *absence* of those blocks, with state 0 corresponding to 8 blocks and state $1 \leq s \leq 7$ corresponding to $7 - s$ blocks.

To compute its state from those of its parents, a child metacell uses a lookup table: a single-channel glider sequence consisting of $8^4 - 1 = 4095$ subsequences of 0–7 gliders each, arranged in

[23] That is, it represents its state in unary, but with an extra glider if it is in any non-zero state.

Free PDF. See conwaylife.com/book for pattern files and print version.

1 024-generation chunks, specifying what the new state should be given any possible combination of not-all-zero parent states. This lookup table is stored in the nucleus, and we can use the one-time switching techniques of Section 12.4 to extract a single 1 024-generation chunk from it. To extract the *correct* chunk, however, we have to delay those one-time switching operations to the correct moment in time, based on the absence of the blocks representing parent metacells' states. To this end, the child metacell uses a **state-lookup clock gun** like the one illustrated in Figure 12.21.

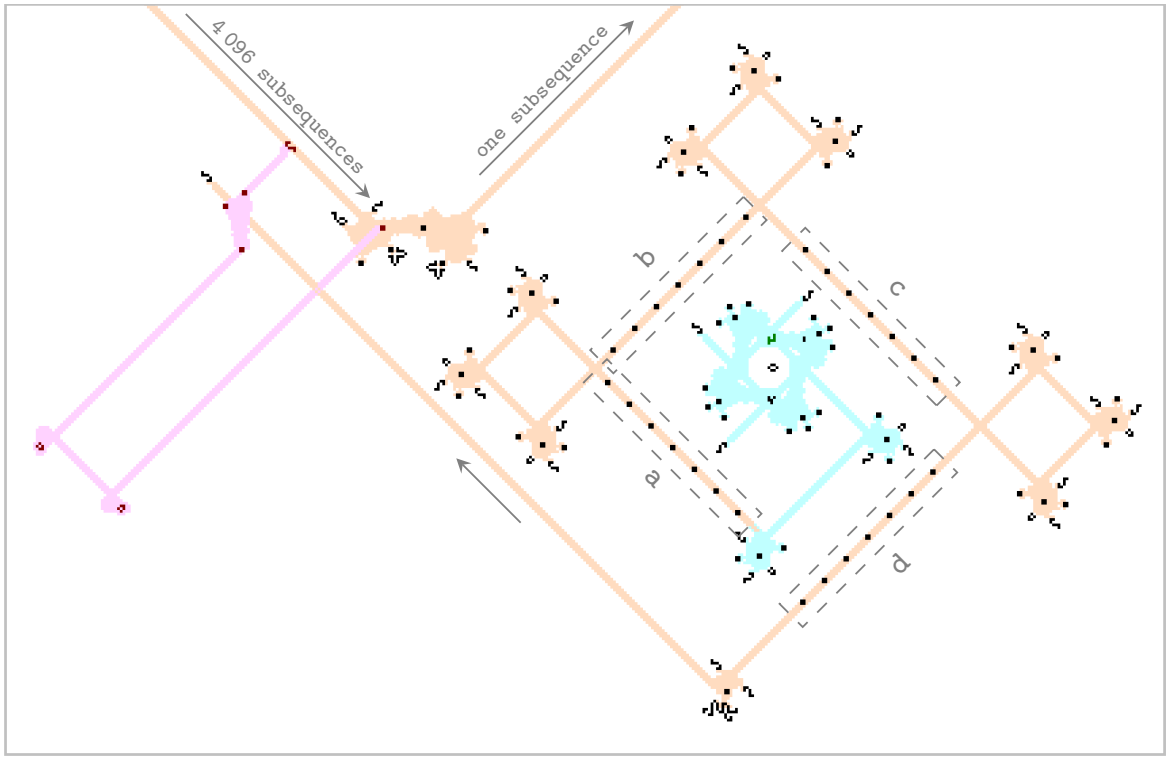


Figure 12.21: A period 1024 clock gun (highlighted in aqua) that has blocks along its path, delaying its first output glider. If there are a blocks along the southwest edge, b along the northwest edge, c along the northeast edge, and d along the southeast edge, then the clock gun's first escaping glider is produced by the Herschel that is present at generation $1024(a+8b+64c+512d)$. That glider hits one-time circuitry (highlighted in magenta) that unblocks and quickly re-blocks a glider stream, allowing roughly 900 generations of its $(1+a+8b+64c+512d)$-th 1 024-generation chunk to escape.

This mechanism works by placing blocks and semi-Snarks along a period 1 024 clock gun's output path: each block destroys a single glider, thus delaying the glider that will trigger the one-time switching circuitry. However, if a semi-Snark is placed between the clock gun and the block, then *two* gliders will be destroyed: one by the block and then another one by the semi-Snark. More generally, placing k semi-Snarks between the clock gun and the block results in the clock gun's first 2^k output gliders being destroyed.

It follows that if we place $0 \leq a,b,c,d \leq 7$ blocks along the gun's output lane, with 3 semi-Snarks between each of those groups of blocks, then

$$a+(2^3)b+(2^6)c+(2^9)d = a+8b+64c+512d$$

gliders will be absorbed before the first one escapes.[24] That first escaping glider then hits one-time

[24] This is similar to the method that we saw in Figure 7.20 for blocking a specific number of gliders via semi-Snarks, but is easier to synthesize and configure.

Free PDF. See conwaylife.com/book for pattern files and print version.

circuitry that unblocks a copy of the state lookup table, re-blocks it just over 900 generations later,[25] and causes the state-lookup clock gun to self-destruct.

12.6 The Nucleus

The 0E0P metacell's nucleus is a massive boustrophedonic loop[26] that houses roughly 3.6 million gliders. Its corners are located at the positions marked `LS`, `LW`, `LN`, and `LE` in Figure 12.18, and its walls between `LW` and `LN`, and between `LE` and `LS`, are each made up of $2^{10} = 1\,024$ two-Snark (i.e., 180-degree) reflectors. The walls of the nucleus are separated by roughly $2^{16} = 65\,536$ full diagonals, so that altogether it takes a glider exactly

$$2^{16} \times 2^{10} \times 4 \times 2 = 2^{29}$$

generations to loop through it.

However, a small fraction of these 2^{29} generations are spent with gliders outside of the nucleus. After cycling through the nucleus's Snark walls, gliders are directed southeast past the corner at `LE` to the locations marked `SE`, then `S`, and then back into the nucleus at `LW`. The important part of this process is the time spent with the logic circuitry at `S`, where the contents of the nucleus can be duplicated and redirected so as to construct a child metacell, or the metacell's state can be extracted from a copy of the nucleus's lookup table.

On the other hand, when a metacell's nucleus is being populated for the first time, gliders are instead inserted into it at the location marked `INS` in Figure 12.18, which is several rows ahead of the start of the loop at `LW`. This "head start" compensates for the fact that this first copy of the nucleus's contents is coming from farther away—the metacell that constructed it—and ensures that the memory loops of the parent and child metacells end up in exactly the same phase.

12.7 Construction and Self-Destruction

Copies of the 0E0P metacell construct each other via the single-channel glider synthesis techniques of Chapter 11. Indeed, the vast majority of the contents of the nucleus are a single-channel glider recipe that carries out this task. For the most part, this construction is straightforward, with long-distance elbow pushes via Corderships (as we described back in Section 11.6) being one of its most complicated pieces.[27]

However, there are two particular problems that arise when trying to construct child metacells that are worth clarifying and focusing on:

- **Child orientation**: We need all metacells to be constructed in the same orientation, regardless of whether they are being constructed by a parent to the northwest, southwest, southeast, or northeast. This is ensured by first building the child's shell (which is impossible to get wrong, since it is rotationally symmetric) and then sending the remainder of the single-channel recipe through the correct recipe input lane so as to build the kernel in the correct orientation. If we'd counterfactually sent the recipe down one of the other three input lanes, it would end up in a different one of the four spiral arms and the resulting metacell would be constructed in the wrong orientation.

[25] The sequences of 0–7 gliders in the state lookup table are separated by 1 024 generations (the period of the state-lookup clock gun). However, roughly 100 of those generations are unusable and must remain empty so that the glider stream does not interfere with the unblocking and re-blocking reactions.

[26] That is, a loop made up of numerous antiparallel lanes.

[27] The 0E0P metacell uses 3-engine Corderships for these long-distance elbow pushes, since the single-channel recipe for the 2-engine Cordership from Figure 11.18 and Appendix B.5 was not known at the time of its completion in 2018. The faster crab-based elbow-pushing method of Section 11.7.1 was also not yet known.

Free PDF. See conwaylife.com/book for pattern files and print version.

- **Nucleus walls**: Each of the 180-degree reflectors along the walls of the nucleus is quite expensive, requiring somewhere between 2^{20} and 2^{21} generations for a single-channel glider recipe to construct. It follows that a recipe for the $2 \times 2^{10} = 2\,048$ reflectors used by the nucleus would be more than $2 \times 2^{10} \times 2^{20} = 2^{31}$ generations long, and would thus not fit inside the period 2^{29} nucleus.
  We could try getting around this problem by adding more rows to the nucleus, but this does not work: adding another 180-degree reflector on each of its northwest and southeast sides increases the length of the glider sequence that it can store by approximately 2^{19} generations, but increases the length of its construction recipe by more than 2×2^{20} generations. It thus becomes necessary to increase the *width* of each row of the nucleus so that it takes 2^{21} generation for gliders to travel from one wall to the other, making it eight times as wide as in the actual 0E0P: $2^{19} = 524\,288$ full diagonals instead of $2^{16} = 65\,536$.

In order to avoid the aforementioned factor-of-eight width increase, the 0E0P metacell only stores a construction recipe for a single 180-degree reflector, and builds temporary circuitry (which we call a **subroutine loop**) that allows that reflector-building recipe to be copied and used over and over again. Due to parity constraints, the reflectors along the northwest and southeast walls have a different shape, so it is actually necessary to have *two* period 2^{21} subroutines: one for the northwest wall, and one for the southeast wall. In the terminology of Figure 12.18, these subroutine loops are located at points `CN` and `CE`, respectively.

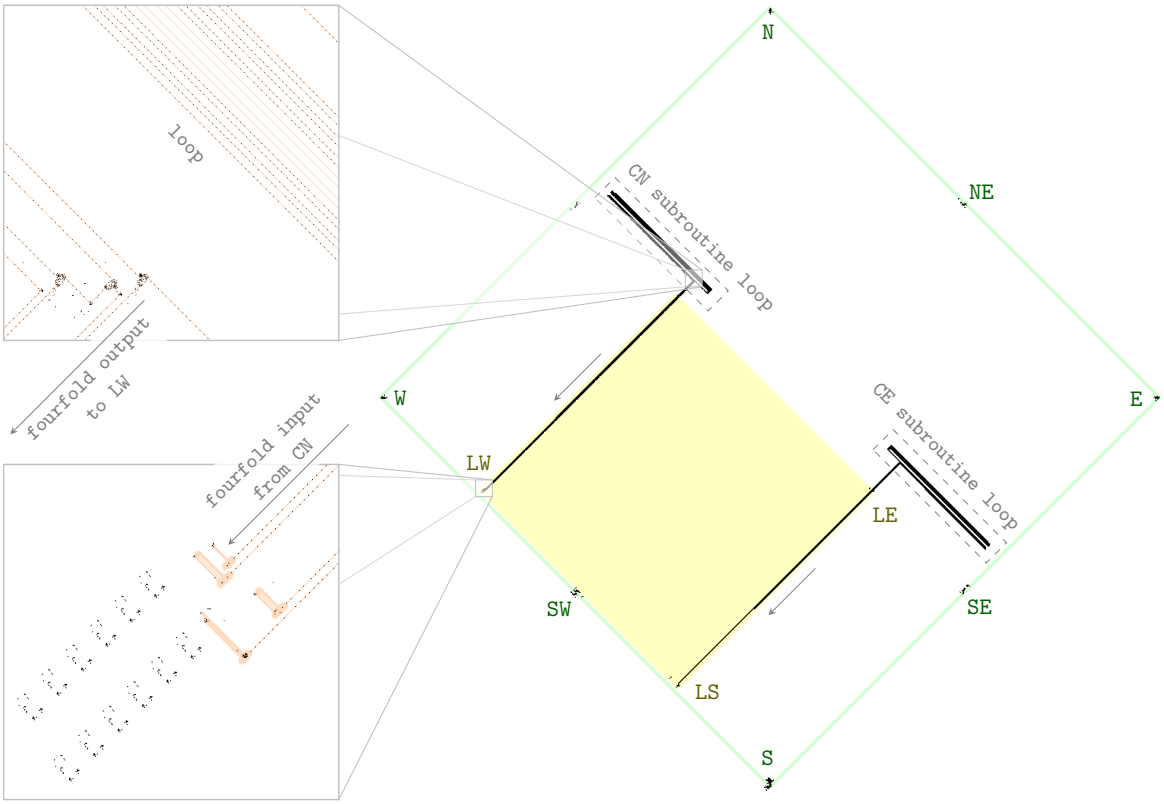


Figure 12.22: An 0E0P metacell that is currently using its pair of subroutine loops to construct the walls of its nucleus. Eight reflectors are constructed simultaneously, since the two loops run in parallel, and each loop uses glider duplicators to send four copies of its reflector-construction recipe southwest at a time.

This technique reduces the problem of constructing an immense (2^{29}-generation) glider loop to the problem of constructing the two smaller (2^{21}-generation) subroutine loops. Fortunately, the

Free PDF. See conwaylife.com/book for pattern files and print version.

period 2^{21} loops are small enough that they can be built without any special tricks, along with the rest of the metacell's kernel. In order to make them each copy the subroutine recipe the correct number of times, each subroutine loop is each equipped with its own clock gun that triggers its self-destruction 2^{29} generations after it is built. Since the subroutine loop has period 2^{21}, it runs for exactly $2^{29}/2^{21} = 2^8 = 256$ iterations. Each subroutine loop furthermore feeds into a fourfold fanout (see Figure 12.22), so that it constructs a total of $256 \times 4 = 1\,024$ reflectors, as desired.

The building the nucleus walls by the subroutine loops is the final step in the 0E0P metacell's construction process. After that, its nucleus is filled with a copy of its own construction recipe. It then goes through its various stages of communicating with and constructing neighboring metacells, and finally it self-destructs.

This self-destruction works in much the same way as that of the Speed Demonoid from Section 11.7.2: every single piece of circuitry in the 0E0P metacell has extra still lifes placed nearby that make it easy to destroy via a single glider. We saw how to do this explicitly with Snarks and Scorbie splitters in Figure 11.23, and none of the 0E0P's circuitry is any more complicated than that. Perhaps the only other object that it is useful to know how to destroy in this way is a semi-Snark. We illustrate a particularly efficient semi-Snark self-destruction trick that is used frequently by the 0E0P metacell in Figure 12.23.

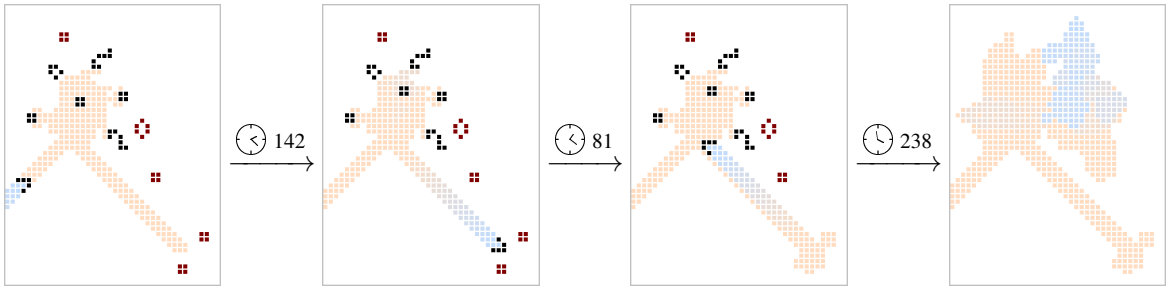


Figure 12.23: Additional still lifes (displayed in red) can be placed near a semi-Snark so that its output glider bounces back, cleanly destroying itself. The self-destruction also releases another output glider towards the southwest on exactly the correct lane so that this reaction can be chained together to destroy multiple semi-Snarks.

12.8 A Complete Metageneration

So far, we have described how the 0E0P metacell works at a fairly high level: every 2^{29} generations, the control clock gun fires a glider that potentially changes what happens throughout the metacell. Some of these gliders direct the single-channel glider sequence from the nucleus to start constructing a neighboring metacell, some of them trigger a signal that sends the metacell's state to its children, and some of them trigger the self-destruction of the entire metacell.

If we break the 2^{35}-generation lifecycle of the metacell down into $2^6 = 64$ "stages" of 2^{29} generations each, then they behave as follows:

- Stage 0: The metacell constructs its southeast neighbor.
- Stage 1: It duplicates the contents of its nucleus into that of the southeast neighbor.
- Stage 2: It constructs its northeast neighbor.
- Stage 3: It duplicates the contents of its nucleus into that of the northeast neighbor.
- Stages 4–7: Repeat stages 0–3 for the northwest and then southwest neighbors.
- Stages 8–11: Do nothing (just spend some time "looking like a cell").

Free PDF. See conwaylife.com/book for pattern files and print version.

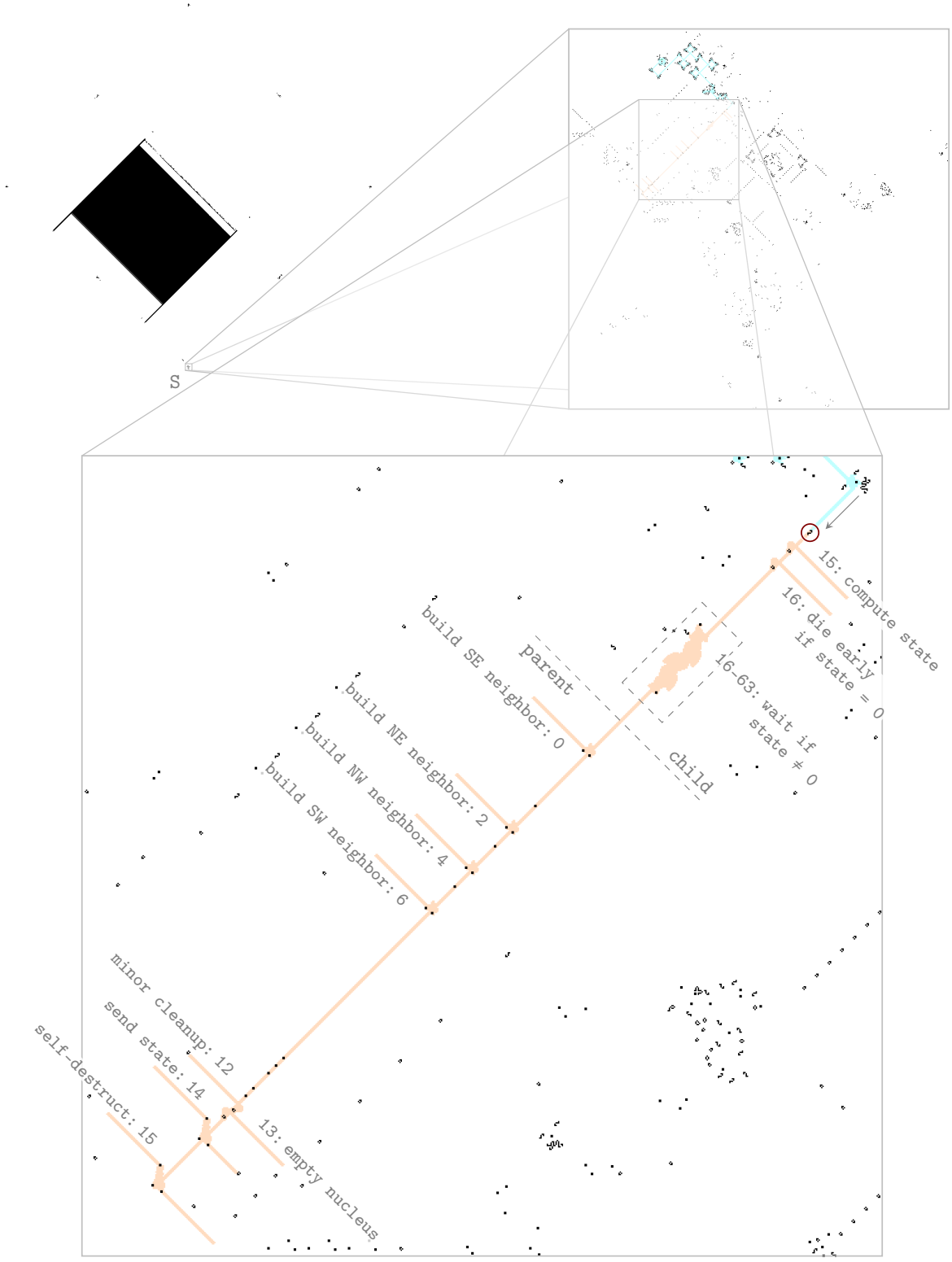


Figure 12.24: One-time circuitry is used to direct the output of the metacell's period 2^{29} control clock gun (highlighted in aqua) to trigger various stages of its lifecycle. The stages begin when the eater 1 circled in red is removed. After that, the metacell computes its state and either dies (if its state is 0) or does nothing for a long time (if its state is non-zero). Stages 16–63, where child cells do nothing, are implemented by an obstruction along the output lane that eats 48 gliders before being destroyed. It then begins acting as a parent metacell rather than as a child: it starts building its own children in stage 0, it starts dying off in stage 12, and it sends its state to its children in stage 14.

Free PDF. See conwaylife.com/book for pattern files and print version.

- Stage 12: Minor cleanup (erase a single Snark).
- Stage 13: Empty the single-channel glider stream out of the metacell's nucleus.
- Stage 14: The parent metacell sends its state to all four of its children.
- Stage 15: The parent metacell self-destructs while the child metacells compute their states.
- Stages 16–63: Child metacells either self-destruct (if their state is 0) or do nothing (if their state is non-zero, they just spend time "looking like a cell").[28]

These stages are initiated by one-time circuitry that is placed along the period 2^{29} output lane of clock gun, as illustrated in Figure 12.24. We now describe what happens throughout the 0E0P metacell's lifecycle in much more specific detail, and highlight key timestamps where interesting things happen or change in its circuitry.

12.8.1 The First 0.26×2^{29} Generations: Construction of the Southeast Child's Shell

We start with an 0E0P metacell that is beginning to act like a parent: it is at the start of stage 0, and is ready to start constructing its children. We say that generation $t = 0$ is the one displayed in Figure 12.25: the metacell has a Herschel in its control clock gun that will create a glider that escapes past all of the semi-Snarks, hits a one-time turner on the period 2^{29} output lane, and is redirected so as to remove an eater 1 that is blocking the clock gun's period 2^{25} output lane.

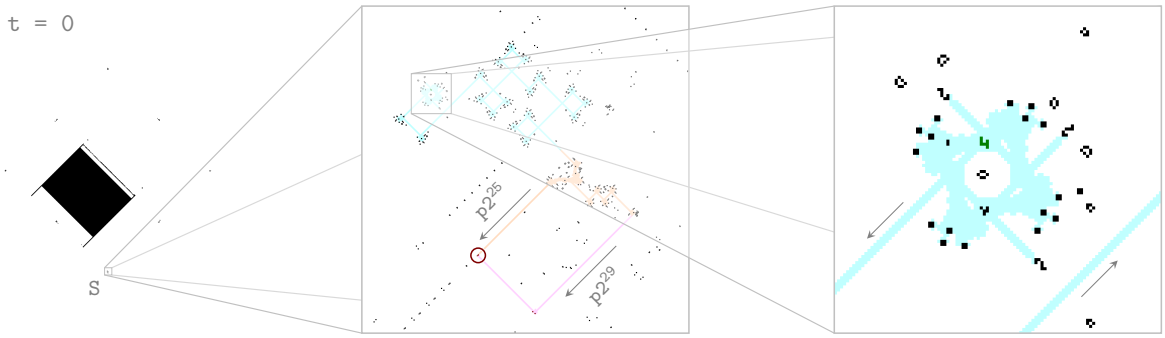


Figure 12.25: A Herschel in the parent metacell's control clock gun (highlighted in aqua) is about to create a glider that will make it out of its period 2^{29} branch. That glider will destroy an eater (circled in red) so that the next glider can escape along that path 2^{25} generations later.

A glider escapes via that new path 2^{25} generations later, at $t = 33\,554\,432 = 0.0625 \times 2^{29}$, and removes an eater 2 from a syringe (see Figure 12.26). This unblocks a copy of the single-channel glider recipe, which now spirals outward through the kernel until it reaches the construction elbow at the metacell's south corner. That construction elbow is then pushed southeast, and is used to begin constructing the southeast child metacell's symmetric shell.

Construction of the child's symmetric shell continues in several stages. A stage often begins with the construction of a Cordership, followed by a slow salvo to shoot down the Cordership once it has travelled the right distance, so that it can be turned into a new target "hand" block at the correct location. Another slow salvo recipe then follows, to build whatever is needed at the location of that new hand block. There are many visible gaps in the 0E0P's single-channel recipe, and most of them correspond to the flight time of a Cordership, while the recipe waits to shoot it down after it has travelled the right distance.

[28] This wait time is only present so that patterns made up of metacells spend a decent amount of time looking, from a distance, like the patterns that they are emulating. When building the 0E0P metacell, Goucher underestimated how long it would take to run in Life simulation software—it could be modified to run 4 times as quickly by removing this wait time.

Free PDF. See conwaylife.com/book for pattern files and print version.

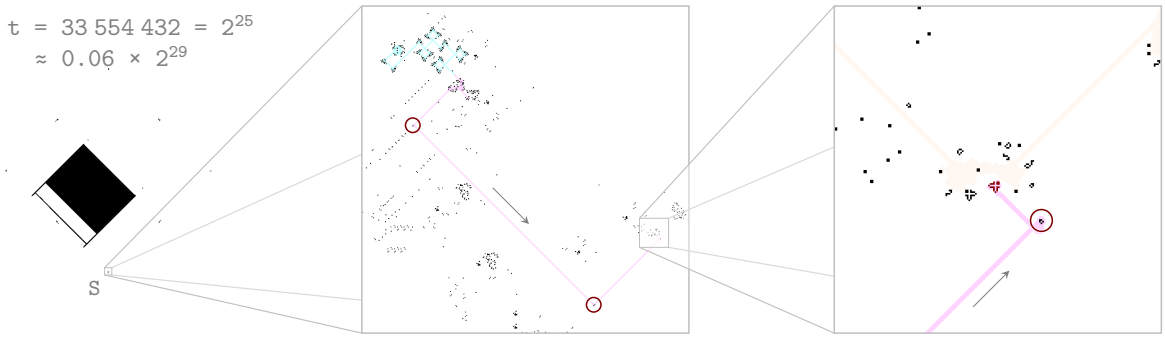


Figure 12.26: A Herschel in the parent metacell's control clock gun (highlighted in aqua) is about to create a glider that will escape via the path that was cleared in Figure 12.25. It will use some one-time turners (circled in red) and then destroy an eater 2 from a syringe, which will allow a copy of the single-channel glider recipe to escape from the nucleus and start constructing the southeast child metacell, starting with its shell.

The first such Cordership launch happens at generation $t = 50\,921\,682 \approx 0.0948 \times 2^{29}$ (see Figure 12.27), where a Cordership is constructed at the child's south corner and is launched northeast to its east corner.

By generation $t = 139\,291\,154 \approx 0.2594 \times 2^{29}$, construction of the final south corner of the symmetric shell is just about complete—the final construction glider has been produced, and a long-range backwards elbow move (**LRBEM**) recipe is processed immediately after this. At $t = 140\,354\,185 \approx 0.2614 \times 2^{29}$, the return glider from this LRBEM recipe has made a new elbow, and the single-channel recipe then shoots a single glider sideways from the new elbow position (at $t = 140\,355\,701$), removes the elbow (at $t = 140\,356\,370$), and then uses five gliders to clean up some leftover junk from the LRBEM (at $t = 140\,880\,132$), far away in the northeast corner.[29] That single extra glider that was shot sideways by the temporary elbow will be used shortly to destroy some of the parent metacell's circuitry, triggering construction of the southeast child metacell's kernel.



Figure 12.27: A glider (circled in red) at the south corner of the child metacell's shell, about to synthesize a Cordership that does a long-distance elbow push to its east corner.

12.8.2 Generations 0.26×2^{29} to 0.80×2^{29}: Construction of the Southeast Child's Kernel

At generation $t = 141\,139\,967 \approx 0.2629 \times 2^{29}$ the extra glider has traveled through several one-time turners to the circuitry at the parent's southeast edge. It then cleanly destroys a Snark from the single-channel glider recipe's path (see Figure 12.28), so that the recipe is directed into the child's

[29] This out-of-order cleanup is just the way that slsparse automatically compiled these LRBEMs in 2018. There are more efficient recipes known now.

Free PDF. See conwaylife.com/book for pattern files and print version.

newly constructed symmetric shell. The design of the shell allows the interior part of the metacell to always be constructed in the same orientation, no matter which direction the construction recipe is coming from, thanks to this step where it is redirected into the correct recipe input lane.

The kernel takes a bit more than twice as long to construct as the shell, and also makes use of numerous long-distance Cordership-based elbow pushes. This construction is heavily unoptimized, as it was generated automatically by slsparse using known standard methods—fire a Cordership towards a location, build something there, fire a glider back, and repeat.

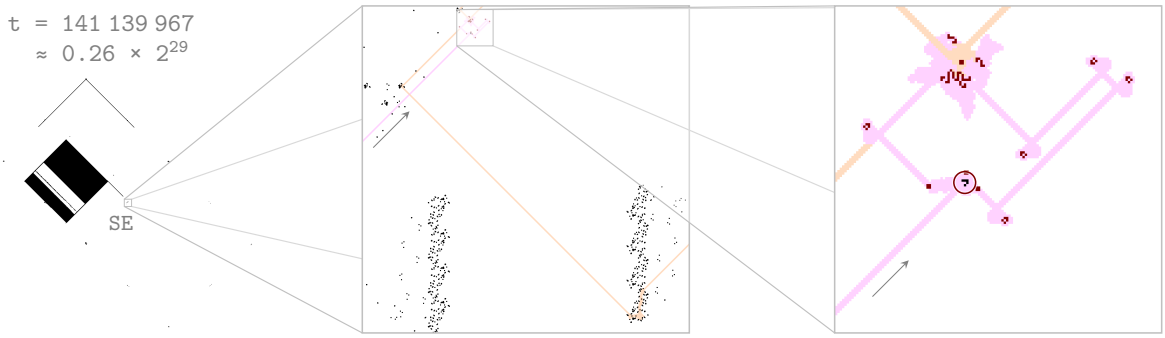


Figure 12.28: The southeast child's shell has been constructed. A single external glider (circled in red) is about to delete a Snark, allowing the rest of the construction recipe to enter the correct lane of the child's shell. This ensures that the rest of that child is constructed in the proper orientation.

However, this could be made much less expensive. For example, instead of using a long-distance Cordership-based elbow push to start construction of the CN construction site (which is done at $t = 181\,162\,958 \approx 0.3374 \times 2^{29}$), it would have been possible to aim gliders at an outlying still life in the unused circuitry near the construction point along the northwest edge, produce a new target object, then rebuild the temporarily sacrificed still life. The 0E0P metacell could be reduced in size and made to run several times faster with these kinds of optimizations—but it would have been far more difficult to complete a working pattern in the first place.

The last part of the kernel to be constructed is its south corner, which begins at generation $t = 367\,748\,078 \approx 0.6850 \times 2^{29}$ (see Figure 12.29). The entirety of the child's control clock gun and logic circuitry will been constructed at this point. The final glider used in the south corner's construction is emitted from the 90-degree elbow block at $t = 430\,473\,285 \approx 0.8018 \times 2^{29}$, followed by a Snarkmaker recipe that is used to redirect the rest of the single-channel glider recipe towards the nucleus.

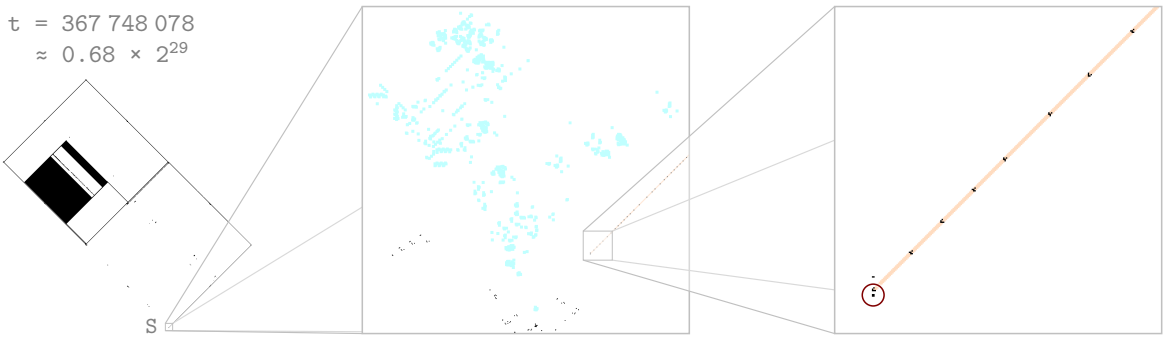


Figure 12.29: Construction of the child's south corner begins. The logic circuit (highlighted in aqua, including the control clock gun) will now be constructed by the elbow block circled in red.

Free PDF. See conwaylife.com/book for pattern files and print version.

12.8.3 Generations 0.80×2^{29} to 0.82×2^{29}: CN and CE Loop Initialization

A single "trigger" glider passes through the Snark that was just created at the south end of the kernel, and is redirected towards a temporary Snark that it passes through at generation $t = 431\,312\,741 \approx 0.8034 \times 2^{29}$ (see Figure 12.30). That glider then travels northeast and then northwest so as to initialize the CN subroutine loop. The single-channel glider recipe that will fill that loop will follow the same path shortly afterwards.

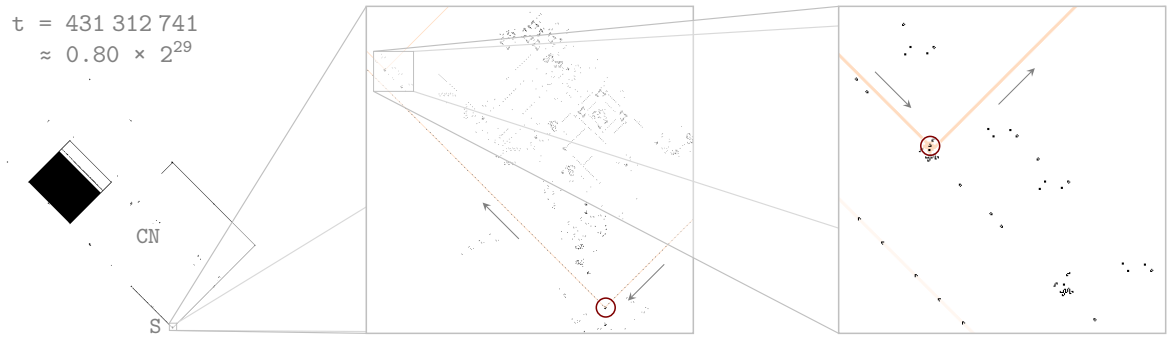


Figure 12.30: The child's kernel has been constructed. A trigger glider goes through the final Snark that was constructed at the south corner of its kernel (circled in red in the middle) and then is redirected through another temporary Snark (circled in red on the right) that sends it to CN, which will initiate construction of the nucleus's northwest wall.

By $t = 431\,969\,616 \approx 0.8046 \times 2^{29}$, the leading trigger glider reaches a temporary Snark in the CN subroutine storage loop (see Figure 12.31). It then goes through a splitter just 214 generations later, at $t = 431\,969\,830$ and initializes the period 2^{29} clock gun there at $t = 431\,971\,416$, which will destroy the period 2^{21} storage loop after it has been used $2^{29}/2^{21} = 256$ times.

The single-channel recipe following the trigger glider then goes through the same splitter, to be duplicated several times into four streams heading southwest that construct the northwest wall of the nucleus. It is also copied into the subroutine glider loop, where it will eventually re-enter this same splitter again, 255 more times. The final glider in that portion of the single-channel recipe enters the loop at $t = 433\,582\,348 \approx 0.8076 \times 2^{29}$ (see Figure 12.32).

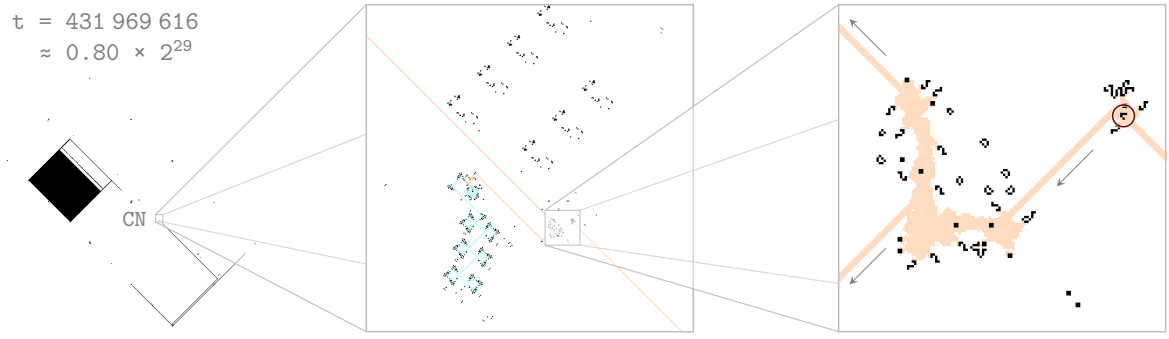


Figure 12.31: The trigger glider from Figure 12.30 (circled in red) has reached CN and is about to start the clock there (highlighted in aqua). That clock will trigger CN's self-destruction 2^{29} generations later, after the period 2^{21} CN subroutine loop has been used $2^8 = 256$ times.

That loop of gliders can be seen having constructed the first set of four 180-degree reflectors (i.e., eight Snarks) along the northwest wall of the nucleus by generation $t = 434\,341\,913 \approx 0.8090 \times 2^{29}$ in Figure 12.33. This eight-Snark construction is repeated a total of 256 times along this northwest wall (for a total of 1 024 reflectors made up of 2 Snarks each).

Free PDF. See conwaylife.com/book for pattern files and print version.

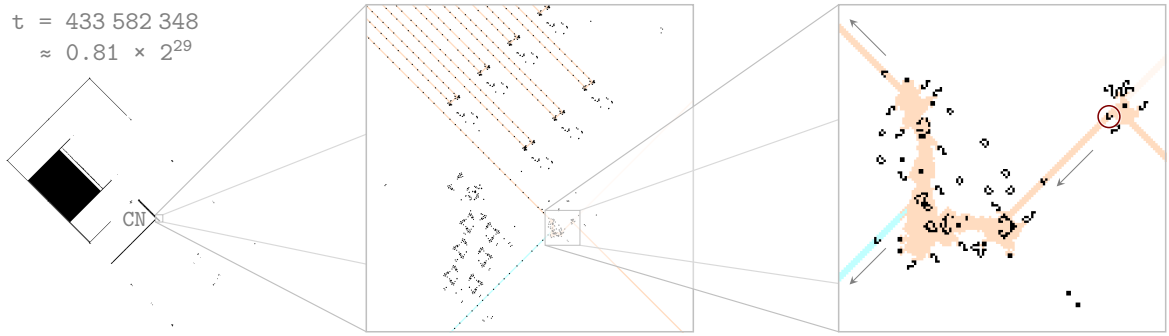


Figure 12.32: The CN subroutine glider loop at CN is now filled, since its final glider (circled in red) just passed through the same temporary Snark as in Figure 12.31, which will soon be destroyed by the glider from Figures 12.30 and 12.31. The gliders leaving from the aqua lane of the splitter are duplicated three more times and then head to LW to construct the northwest wall of the nucleus.

After the subroutine glider loop is filled back in Figure 12.32, the input Snark from that figure must be removed because it's directly in the way of the circulating recipe in the loop. The trigger glider does this, and the Snark is completely erased as of $t = 434\,725\,275 \approx 0.8097 \times 2^{29}$. The trigger glider then travels southeast, and then southwest, back to the child metacell's south corner, where it will destroy and be destroyed by the Snark from Figure 12.30 that sent it to the CN storage loop in the first place. This final act of the trigger glider takes place at $t = 434\,724\,270 \approx 0.8097 \times 2^{29}$ (see Figure 12.34).

The entire procedure used to initialize the CN subroutine loop, as described by Figures 12.30–12.34, then repeats for the CE subroutine loop. At $t = 435\,925\,955 \approx 0.8120 \times 2^{29}$, the trigger glider for the CE subroutine recipe passes through the location where the Snark from Figure 12.34 used to be, and instead goes into a Snark that directs it to CE (see Figure 12.35).

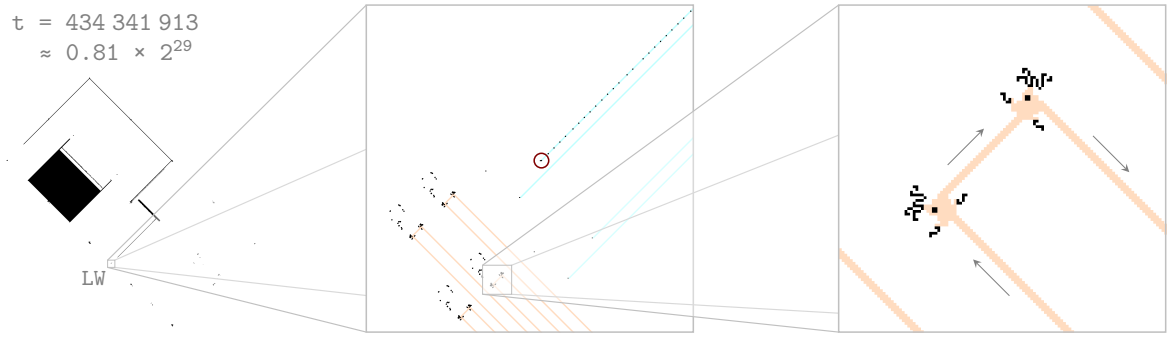


Figure 12.33: The subroutine glider loop has been used once at LW to construct the first set of Snarks along the nucleus's northwest wall (the last elbow operation from that loop is about to pull the elbow block circled in red by 14fd). The trigger glider from Figure 12.30 is between CE and SE, on its way back to S.

That trigger glider will then activate the clock gun located at CE at $t = 436\,382\,282 \approx 0.8128 \times 2^{29}$, in the exact same way as happened at CN in Figure 12.31. The subroutine portion of the single-channel recipe then fills the loop, with its final glider entering the loop at $t = 437\,805\,347 \approx 0.8155 \times 2^{29}$. Once this glider passes through this temporary Snark, the full construction recipe of the 0E0P metacell has been completely processed.

The trigger glider then destroys the CE subroutine loop's input Snark, and is then redirected along a self-destruct chain toward the child's south corner, removing reflectors in four more locations along the way. Finally, it destroys (and is destroyed by) the Snark that directed it to CE in the first place.

Free PDF. See conwaylife.com/book for pattern files and print version.

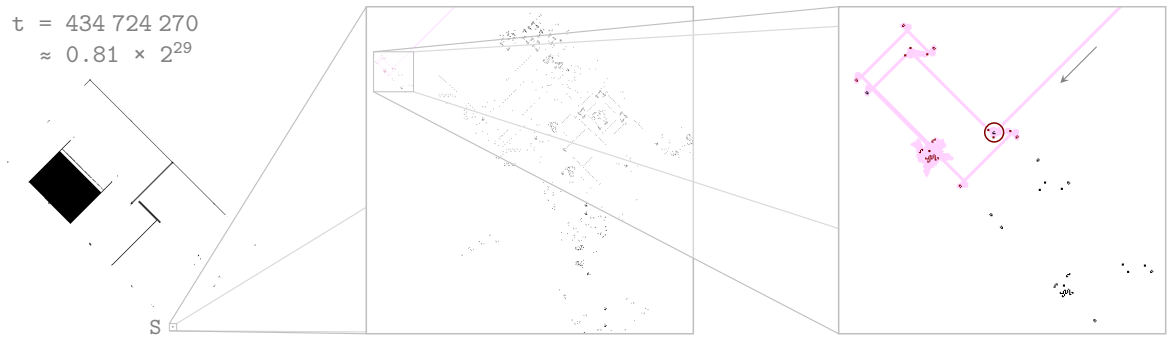


Figure 12.34: The trigger glider from Figure 12.30 makes it back to S, where it destroys (and is destroyed by) the Snark from that figure. This opens a path toward the CE construction location, which will be used soon.

This final usage of the trigger glider occurs at $t = 438\,935\,522 \approx 0.8176 \times 2^{29}$ (see Figure 12.36).

Thanks to the removal of this Snark, the *next* copy of the single-channel glider recipe that comes into this child metacell will be fed through the location that was marked INS in Figure 12.18, and thus into the Snarks that make up the nucleus's walls, rather than to the CN or CE subroutine loops. Those nucleus walls are still being constructed by the subroutine loops, which won't finish for roughly 2^{29} more generations. However, the nucleus construction process moves at roughly the same speed as the approaching recipe, so the construction will safely complete before the recipe gets there.

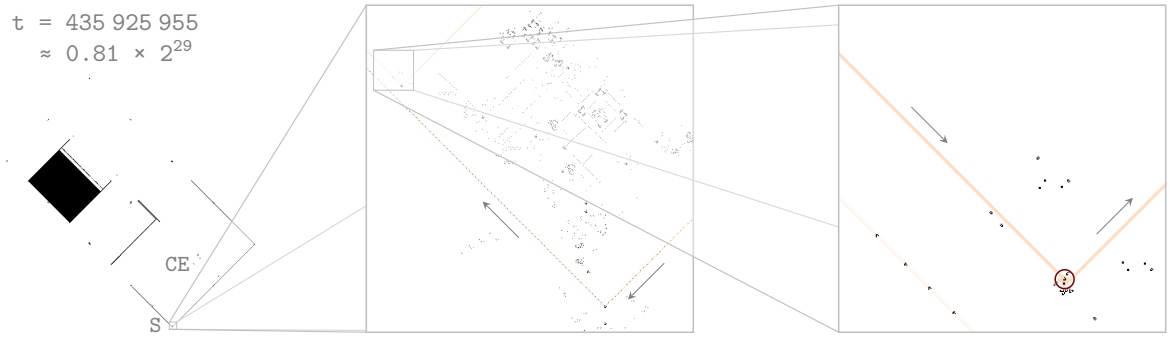


Figure 12.35: A trigger glider goes through a Snark at the child's south corner that leads it to CE, which will initiate construction of the nucleus's southeast wall (just like the CN trigger glider from Figure 12.30). The same things as in Figures 12.31–12.33 then happen, but along that southeast wall instead of the northwest one.

12.8.4 Generations 0.82×2^{29} to 2×2^{29}: Activation of the Southeast Child

Now both subroutine loops of the child metacell are filled and are building the repetitive Snark chains that will hold its copy of the single-channel glider recipe. When construction is done there will be 256 copies of eight Snarks in long chains along the northwest and southeast walls of the nucleus. These sets of eight Snarks and their associated self-destruct circuitry are not quite mirror images of each other, so different recipes are needed in the CN and CE subroutine loops.

This construction and filling of the nucleus goes on for quite a while. The subroutine storage loops are period 2^{21}, and they have to cycle 256 times each before being shut down by their period $256 \times 2^{21} = 2^{29}$ clock guns. Splitters near the storage units duplicate the recipe several times, so that construction actually proceeds on four Snark pairs simultaneously. In the configuration of eight Snarks that is constructed 256 times, Snark pairs are offset from each other in a zig-zag pattern so that

Free PDF. See conwaylife.com/book for pattern files and print version.

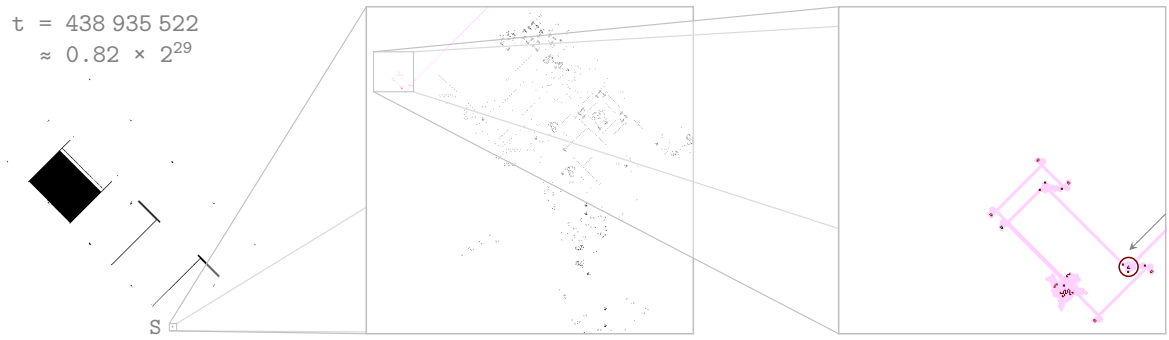


Figure 12.36: The trigger glider from Figure 12.35 makes it back to S, where it destroys (and is destroyed by) the Snark from that figure. A complete copy of the construction recipe from the parent metacell has now been used—the next copy will be directed into the nucleus of this child.

the simultaneous constructions don't get in each other's way (refer back to Figure 12.22).

At generation $t = 968\,843\,520 \approx 1.8046 \times 2^{29}$, the northwest wall of Snarks is complete, and a Herschel is present in the CN subroutine clock's gun that will produce the glider that makes it through all of the twenty-one semi-Snarks in the loops below (see Figure 12.37). That glider destroys a block in a syringe in that subroutine loop, so that all further recipe gliders in the loop will be harmlessly absorbed by the syringe's eater 2—no more construction will occur.

This same first output glider (which was produced 2^{29} generations after the CN subroutine loop was activated) is also duplicated, and its copy is redirected so as to start the gradual process of dismantling the CN subroutine clock gun and subroutine loop. Specifically, the duplicated glider destroys three of the clock gun's semi-Snarks via the reaction of Figure 12.23. This makes the clock gun 8 times faster, and subsequent output gliders act similarly. The result of this is that the subroutine clock gun did nothing for a long time (2^{29} generations), but will now start dismantling itself faster and faster.

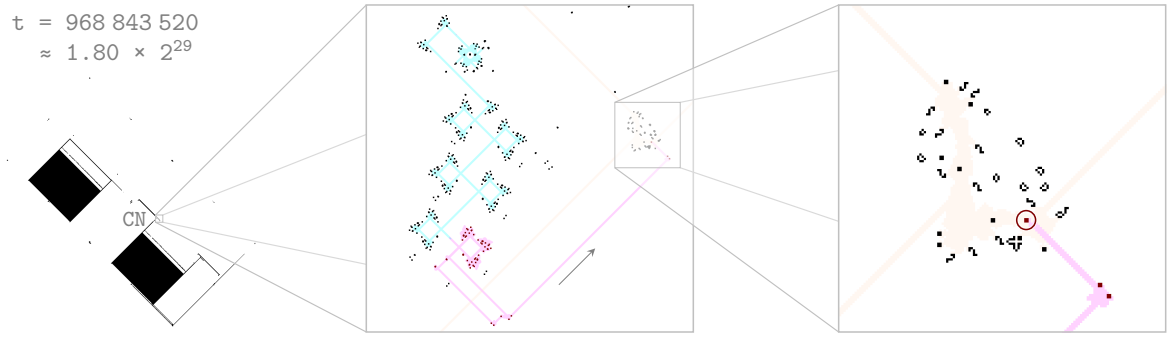


Figure 12.37: A Herschel is present in the CN clock gun that will produce the glider that makes it through all of the twenty-one semi-Snarks in the loops below. That signal destroys a block in a syringe (circled in red), so that the CN storage loop stops building the northwest wall of the nucleus. This same glider also destroys the three outermost semi-Snarks, making the clock gun 8 times faster and initiating its self-destruct process.

For example, the *next* three semi-Snarks are destroyed just 2^{26} generations later, by the Herschel that appears at $t = 1\,035\,952\,384 \approx 1.9296 \times 2^{29}$, and the clock gun vanishes completely just a bit more than 2^{24} generations after that, at $t = 1\,054\,065\,485 \approx 1.9633 \times 2^{29}$. The entirety of the CN subroutine loop is similarly gone just a few thousand generations later, at $t = 1\,054\,073\,568$. The CE subroutine clock and storage loop self-destruct in the exact same way shortly thereafter, and are gone as of $t = 1\,058\,486\,020 \approx 1.9716 \times 2^{29}$.

Back in time slightly at $t = 974\,368\,517 \approx 1.8149 \times 2^{29}$, after construction of the Snark walls

Free PDF. See conwaylife.com/book for pattern files and print version.

finished but before the subroutine storage loops self-destructed, the final glider in the single-channel recipe is about to enter the child's nucleus at INS (see Figure 12.38). Once this insertion is complete, the child's nucleus will be fully populated with a copy of the parent's DNA.

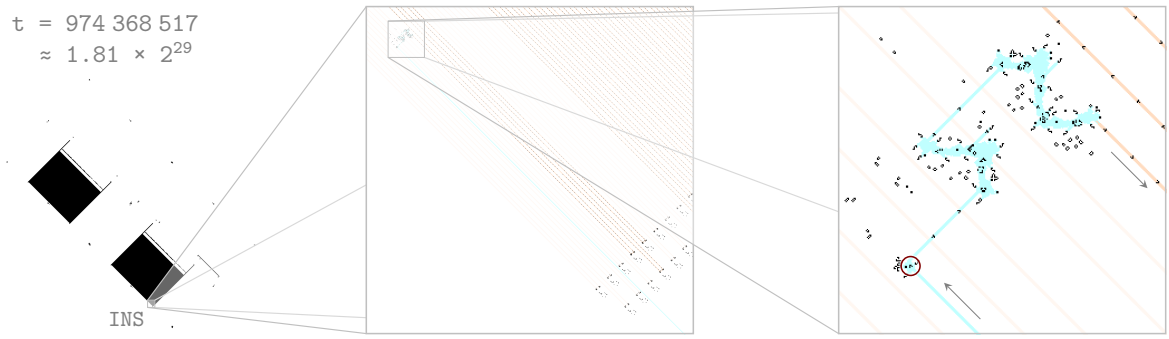


Figure 12.38: The final glider in the 0E0P's single-channel glider recipe is injected into the child's nucleus, at the location circled in red.

After the first glider in the child's nucleus makes it all the way through the long back-and-forth Snark chains for the first time, it arrives at a splitter at the south corner at $t = 1\,073\,734\,584 \approx 2.0000 \times 2^{29}$. It then uses one-time circuitry to start up the child's control clock gun (see Figure 12.39), aligned exactly with the timing of the parent's control clock gun. However, the clock gun does not actually do anything until much later, when an eater is removed from its period 2^{29} output path.

At this point, the southeast child is fully constructed and simply waits until the parent metacell's other children are constructed before really doing anything. The single-channel glider recipe will cycle through its nucleus several times, completely unchanging, before it is actually made to do anything.

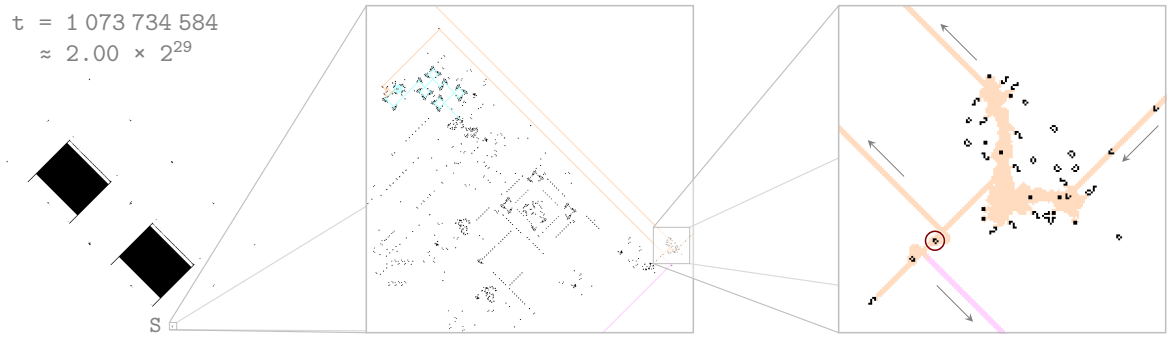


Figure 12.39: The first gliders in the child's single-channel glider recipe complete their first journey through its nucleus. The very first of these gliders hits a one-time reflector (circled in red) that redirects it to start up that child metacell's clock gun (highlighted in aqua).

12.8.5 Generations 2×2^{29} to 8×2^{29}: Construction of the Other Children

Back in the parent metacell, at $t = 1\,073\,741\,824 = 2 \times 2^{29}$, a Herschel is present in the control clock gun. This Herschel will produce a glider that removes an eater 1 blocking the period 2^{25} branch of the clock gun (see Figure 12.40). Exactly 2^{25} generations later, at $t = 1\,107\,296\,256$, a Herschel in the same location uses one-time circuitry to switch over to building the northeast child metacell.

At $t = 1\,610\,612\,736 = 3 \times 2^{29}$, the next "$2^{29}$" Herschel appears in the clock gun. However, despite the resulting glider making it past all 21 semi-Snarks in the clock gun, it does not actually do anything of note—there is a block along the period 2^{29} output lane that simply absorbs it. This block causes another 2^{29}-generation cycle to occur, so that the single-channel glider recipe is sent to the northeast

Free PDF. See conwaylife.com/book for pattern files and print version.

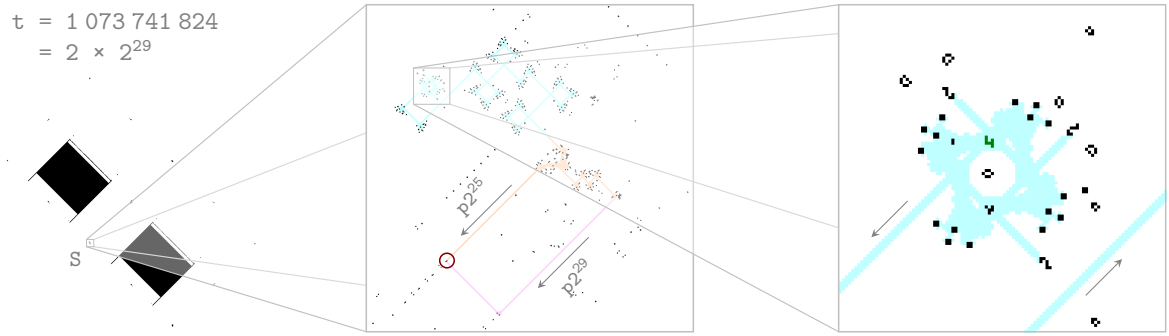


Figure 12.40: A Herschel is present in the control clock gun of the parent metacell (highlighted in aqua), which releases a glider that uses a one-time turner to destroy an eater 1 (circled in red), thus initiating construction of the northeast child.

child metacell twice: once to do the construction, and once to populate the new child's nucleus with a copy of the recipe. Blocks along the control clock gun's output lane will similarly be used to absorb the output gliders that are produced by Herschels at $t = 5 \times 2^{29}$, and $t = 7 \times 2^{29}$.

By generation $t = 2\,147\,483\,648 = 4 \times 2^{29}$, the northeast child's nucleus has been filled and its control clock gun has started up. At the same time, a Herschel is present in the parent's control clock gun that will produce a glider that removes an eater 1 from its period 2^{25} output lane. Exactly 2^{25} generations later, at $t = 2\,181\,038\,080$, another Herschel is present in the parent's control clock gun that will use one-time circuitry to switch over to construction of the northwest child metacell (see Figure 12.41).

By generation $t = 3\,221\,225\,472 = 6 \times 2^{29}$, the northwest child has been constructed and had its nucleus filled, and another Herschel is present in the parent's control clock gun that leads to the construction of the southwest child. Finally, all four child metacells have been constructed and had their nuclei filled by $t = 4\,294\,967\,296 = 8 \times 2^{29}$.

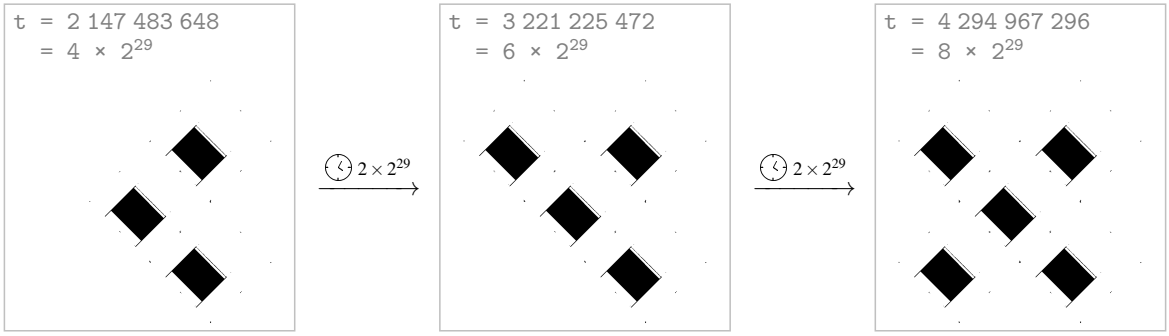


Figure 12.41: The parent metacell constructs its children in the following order: southeast, northeast, northwest, southwest. It takes 2×2^{29} generations to build a single child and populate its nucleus.

12.8.6 Generations 8×2^{29} to 14×2^{29}: Preliminary Clean-Up

Nothing of importance happens in any of the parent or child metacells between generations 8×2^{29} and 12×2^{29}—they all just loop the single-channel recipe through their nuclei a few times. In the parent metacell, the output glider of the control clock gun hits a sequence of four blocks along its period 2^{29} output lane (which can be seen near the southwest end of that lane in Figure 12.24), simply causing it to do nothing for these generations. In the child metacells, the output glider of the control clock gun is still being absorbed by an eater 1 (which is circled in red at the northeast end of the

Free PDF. See conwaylife.com/book for pattern files and print version.

output lane in Figure 12.24).

At $t = 6\,442\,450\,944 = 12 \times 2^{29}$, the parent metacell's control clock gun is holding a Herschel that will release a glider, which destroys a single Snark elsewhere at its south corner. This Snark was used by the metacell when it was a child, after computing its state, to store that state in a sequence of up to 8 blocks.[30]

More interesting things finally start happening 2^{29} generations later: at $t = 6\,979\,321\,856 = 13 \times 2^{29}$, a Herschel is present in the parent's control clock gun that will release a glider, which will use one-time circuitry to destroy a block from a syringe in the part of the nucleus memory loop that extends through the metacell's south corner (see Figure 12.42). As a result, the parent metacell's entire single-channel glider recipe and state lookup table will be absorbed by that syringe's eater 2 on their next cycle through the nucleus. All of those gliders will thus gradually empty out of the nucleus over the course of those next 2^{29} generations, with the final glider being absorbed at $t = 7\,516\,189\,238 \approx 14.0000 \times 2^{29}$, leaving the parent's nucleus completely empty.

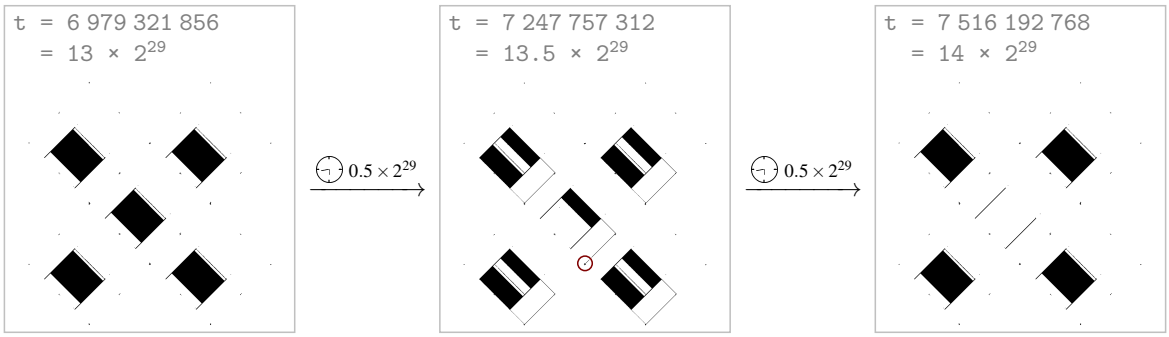


Figure 12.42: A Herschel is present in the parent metacell's control clock gun that is used to erase a block from a syringe, so that the contents of its nucleus are absorbed over the next 2^{29} generations at the location circled in red.

12.8.7 Generations 14×2^{29} to 15×2^{29}: State Information is Sent to Children

After its nucleus is empty, the parent metacell sends its state to its children, so that those children will be able to compute their own states. This process is initiated at $t = 7\,516\,192\,768 = 14 \times 2^{29}$, when a Herschel is present in the parent metacell's control clock gun that will fire 8 gliders at the 0–6 or 8 blocks that store its state. As a result, 0 or 2–8 gliders will escape (representing the metacell's state 0 or 1–7), as illustrated in Figure 12.43. In the example that we are illustrating here, the parent metacell is in state 7, so there are 0 blocks, and all 8 gliders will escape.

Those gliders follow a spiral path through the kernel into the south "`logic`" area of each of the children (and this always happens, regardless of whether or not this particular parent metacell built that child). When the state gliders arrive at a child's "`logic`" area, the first glider (if it exists) hits one-time circuitry that erases the eater 1 that has been blocking the control clock gun's period 2^{29} output lane, while the remaining 0–7 gliders are used to destroy 0–7 blocks along the output path of the state-lookup clock gun, thus storing the parent's state in the child. The state gliders arrive at the four child metacells at slightly different times, in the following order:

- **SW child**: at generation $t = 7\,517\,755\,585 \approx 14.0029 \times 2^{29}$ (see Figure 12.44).
- **NE child**: just 2052 generations later, at $t = 7\,517\,757\,637$.

[30] This Snark is destroyed now, instead of during the "main" self-destruction stage at $t = 15 \times 2^{29}$, just as a minor convenience—the 0E0P metacell could be rewired to destroy it later instead.

Free PDF. See conwaylife.com/book for pattern files and print version.

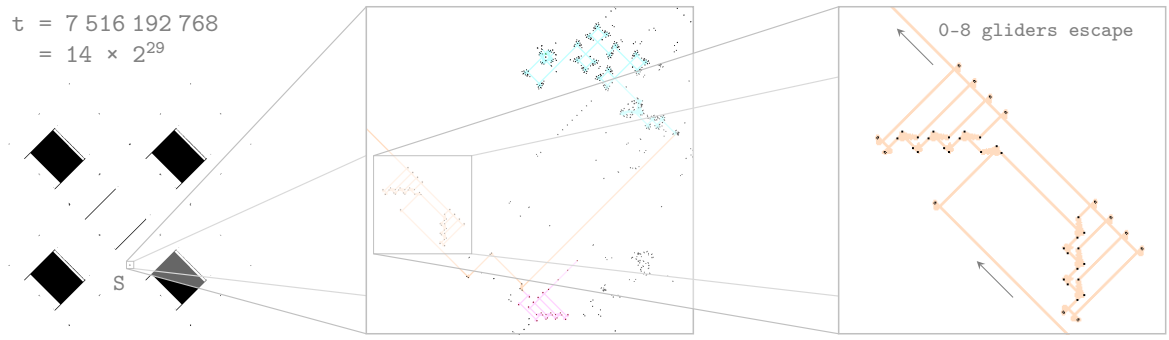


Figure 12.43: A Herschel is present in the control clock gun of the parent metacell (highlighted in aqua). It releases a glider that uses one-time circuitry to send its state, which is currently stored in a sequence of 0–8 blocks, to its children as a sequence of 0–8 gliders. Some mild cleanup also happens at this stage—the four state-reading boats that this metacell used back when it was a child (if any of them still exist, highlighted in magenta) are destroyed.

- **NW child**: at $t = 7\,518\,791\,855 \approx 14.0048 \times 2^{29}$.
- **SE child**: just 2 892 generations later, at $t = 7\,518\,794\,747$.

Since every child metacell has a parent, a non-zero state will be sent in from at least one the four possible directions, so the eater 1 blocking the control clock gun's period 2^{29} output lane will always be destroyed. Also, state information arrives from the (up to) four parents on slightly different lanes, so that state gliders from different parents destroy different blocks in the state-lookup clock gun.

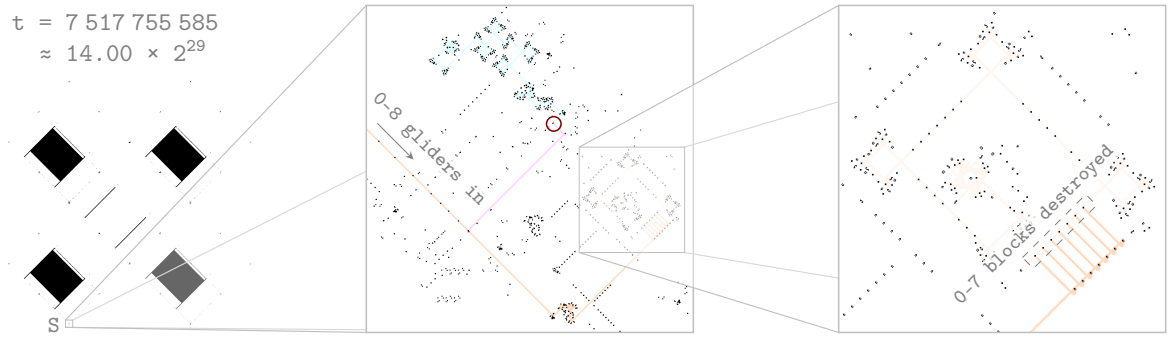


Figure 12.44: State gliders from the parent metacell arrives at its southwest child. The first of these gliders hits a one-time turner (a boat, mentioned as being destroyed in the parent metacell in the caption of Figure 12.43) and then destroys the eater 1 blocking the period 2^{29} output lane of that child's control clock gun (circled in red). The other 0–7 state gliders destroy 0–7 blocks along the path of the state-lookup clock gun (here, the parent metacell was in state 7, so 7 gliders destroy 7 blocks).

12.8.8 Generations 15×2^{29} to 16×2^{29}: Child State Computation and Parent Self-Destruction

Now that child metacells have recorded the states of their parents (in the absence of certain blocks in their state-lookup clock guns), it is time for them to compute their own states. To do this, a Herschel is present in each child's control clock gun at $t = 8\,053\,063\,680 = 15 \times 2^{29}$ that triggers its state-lookup clock gun to start. That state-lookup gun will be ready to compute the metacell's state 8 846 generations later, at $t = 8\,053\,072\,526$ (see Figure 12.45).

This period 1 024 state-lookup gun works as described in Section 12.5.2: if we label the states of the northwest, southeast, southwest, and northeast parents by s_{NW}, s_{SE}, s_{SW}, and s_{NE}, respectively, then

$$(7 - s_{\mathrm{NW}}) + 8(7 - s_{\mathrm{SE}}) + 64(7 - s_{\mathrm{SW}}) + 512(7 - s_{\mathrm{NE}})$$

Free PDF. See conwaylife.com/book for pattern files and print version.

gliders are absorbed by blocks and semi-Snarks before any escape.[31] That first escaping glider is thus generated by the Herschel that is present in it at generation

$$t = 15 \times 2^{29} + 8846 + 1024\big((7 - s_{\mathrm{NW}}) + 8(7 - s_{\mathrm{SE}}) + 64(7 - s_{\mathrm{SW}}) + 512(7 - s_{\mathrm{NE}})\big).$$

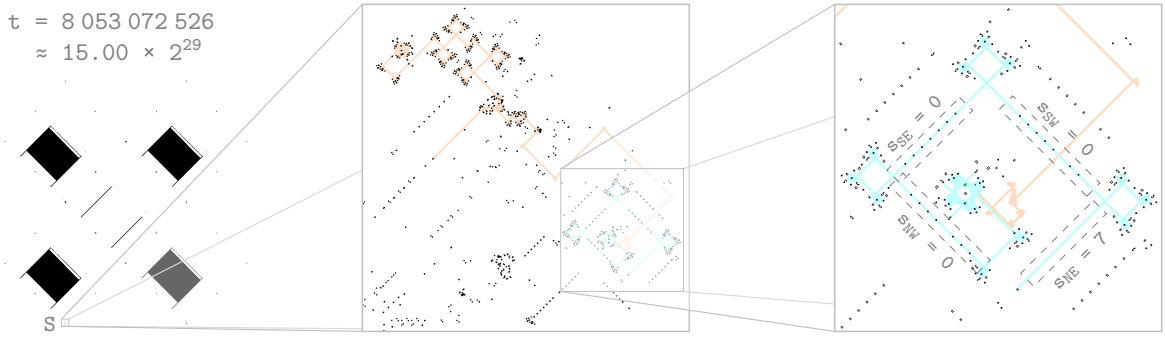


Figure 12.45: The state-lookup clock gun (highlighted in aqua) has been started in the child metacells. In the southwest child metacell (magnified here), the parent metacell to the northeast had state 7, which is encoded in the fact that 7 blocks are missing along one of the clock gun's edges. The leftover arrangement of blocks make it so that the central period 1024 gun releases $(7 - s_{\mathrm{NW}}) + 8(7 - s_{\mathrm{SE}}) + 64(7 - s_{\mathrm{SW}}) + 512(7 - s_{\mathrm{NE}}) = 511$ gliders before one escapes.

For example, consider the southwest child of the metacell that we have been tracking. It has a northeast parent in state 7, and all other parents in state 0 (since they do not exist), so the first glider to make it out of its state-lookup gun does so via the Herschel that is present at generation

$$\begin{aligned} t &= 15 \times 2^{29} + 8846 + 1024\big((7 - 0) + 8(7 - 0) + 64(7 - 0) + 512(7 - 7)\big) \\ &= 8\,053\,595\,790 \approx 15.0010 \times 2^{29}. \end{aligned}$$

That glider escaping the state-lookup gun causes it to self-destruct and then emit a pair of gliders: one that destroys an eater at $t = 8054286989 \approx 15.0023 \times 2^{29}$, unblocking the single-channel glider stream,[32] and one that destroys a block in a syringe just 972 generations later, re-blocking the stream. The result is that an approximately 900-generation chunk of the 0E0P's state lookup table is copied—a portion containing 0–7 gliders that describe the new state of this child metacell (see Figure 12.46).

The state-lookup gun of this southwest child thus extracts the 512th chunk of the lookup table, which contains 2 gliders, indicating that its state is 2. This agrees with the list of state transitions for the first half-metageneration that we saw back in Figure 12.15: a lone parent in the 2-state Moore-neighborhood CA creates a child to the southwest in the 8-state von-Neumann-neighborhood CA with state 2. The other three children look at different chunks of the lookup table and thus extract different states, all of which also coincide with the state transition rule described by Figure 12.15:

- **NE child**: The state-lookup clock gun extracts the

$$1 + (7 - 0) + 8(7 - 0) + 64(7 - 7) + 512(7 - 0) = 3\,648\text{th}$$

chunk of the lookup table, which contains 4 gliders, so its state is 4.

[31] In the notation of Section 12.5.2, we have $a = 7 - s_{\mathrm{NW}}$, $b = 7 - s_{\mathrm{SE}}$, $c = 7 - s_{\mathrm{SW}}$, and $d = 7 - s_{\mathrm{NE}}$. This particular ordering of which neighbors corresponds to "a", "b", "c", and "d" is unimportant—it's just a result of how it was easiest to wire the 0E0P metacell.

[32] This timestamp is specific to this southwest child. Children with different parent state configurations will have this happen slightly earlier or later.

Free PDF. See conwaylife.com/book for pattern files and print version.

- **NW child**: Extracts the $1+(7-0)+8(7-7)+64(7-0)+512(7-0)=4\,040$th chunk of the lookup table, which contains 1 glider, so its state is 1.
- **SE child**: Extracts the $1+(7-7)+8(7-0)+64(7-0)+512(7-0)=4\,089$th chunk of the lookup table, which contains 6 gliders, so its state is 6.

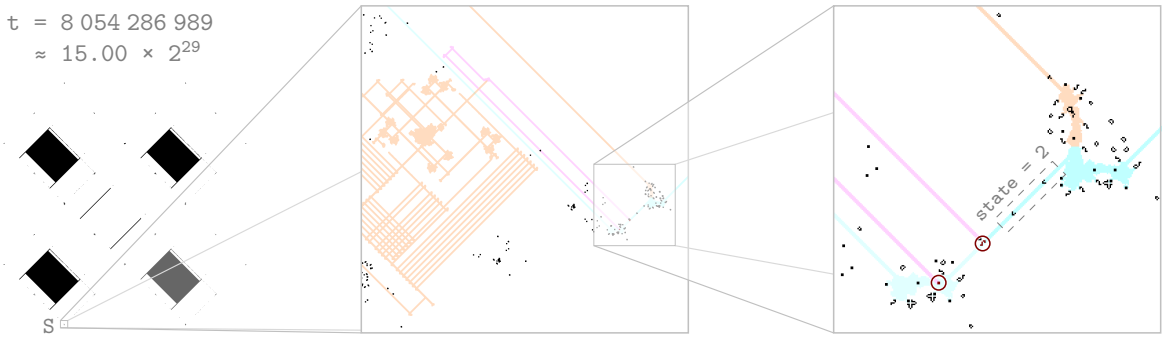


Figure 12.46: The southwest child's state (2) being extracted from the lookup table. A glider destroys an eater 1 (at the top-right location circled in red), unblocking a portion of the lookup table, and another glider destroys a block (at the bottom-left location circled in red), re-blocking it 972 generations later.

After the children's states are extracted in the form of 0–7 gliders, the first glider (if it exists) is duplicated, resulting in a sequence of 0 or 2–8 gliders, and erases some one-time circuitry that would otherwise cause the metacell to die early. The resulting sequence of gliders is then redirected toward a sequence of 8 blocks, so that the state can be stored for a long period of time in a pattern of missing blocks. In the southwest child, this happens at $t=8\,054\,294\,330$ (see Figure 12.47).

Meanwhile, the parent metacell has served its purpose, so it is free to die. The Herschel that is present in *its* control clock gun at generation $t=8\,053\,063\,680=15\times2^{29}$ trigger this self-destruction. Much like the destruction of CN clock gun that was described in Figure 12.37, this self-destruction takes place over multiple stages, with some semi-Snarks being destroyed in each stage so that subsequent stages occur more rapidly.

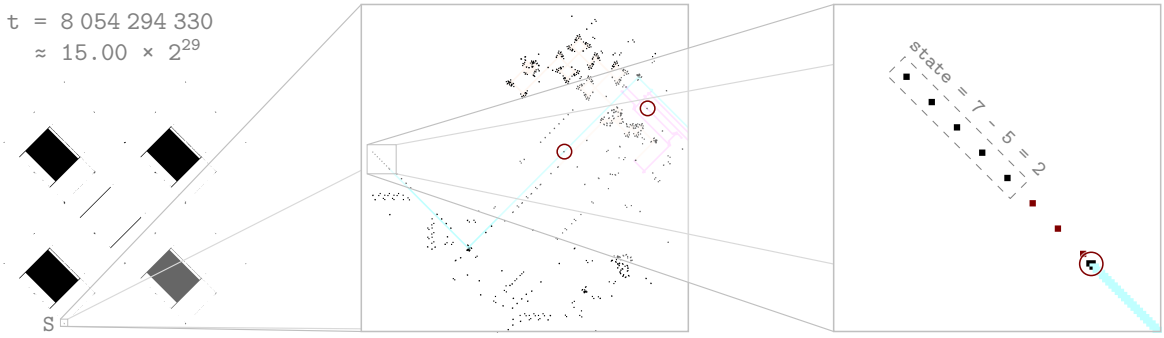


Figure 12.47: The southwest child's 2 state gliders erasing one-time circuitry that would otherwise cause the metacell to die early (highlighted in magenta), and storing their state in the 8-block sequence at the west edge of the south corner. The first of those 2 state gliders was duplicated, so 3 gliders (circled in red) will hit the blocks, leaving $8-3=5$ blocks behind, corresponding to the child's state $7-5=2$.

During one of these intermediate stages, at $t=8\,124\,366\,848\approx15.1328\times2^{29}$, a Herschel is present in the control clock gun that will produce a glider that escapes northwest towards LS, triggering the destruction of the entire southeast wall of the nucleus, as well as a few remaining pieces of the CE construction site (see Figure 12.48). It then continues travelling clockwise around the metacell, destroying the circuitry at SW, the northwest wall of the nucleus (and some remaining pieces of CN),

Free PDF. See conwaylife.com/book for pattern files and print version.

W, NW, N, NE, E, and SE, before finally being stopped by a block back at S at $t = 8\,126\,433\,672 \approx 15.1367 \times 2^{29}$.

Self-destruction continues through a few more stages past this point, with it being completely done at generation $t = 8\,129\,508\,130 \approx 15.1424 \times 2^{29}$, after which the parent metacell is gone, and only child metacells remain. The final clean-up step that occurs is a second glider escaping to the northwest from the control clock, which destroys the nucleus insertion location INS.

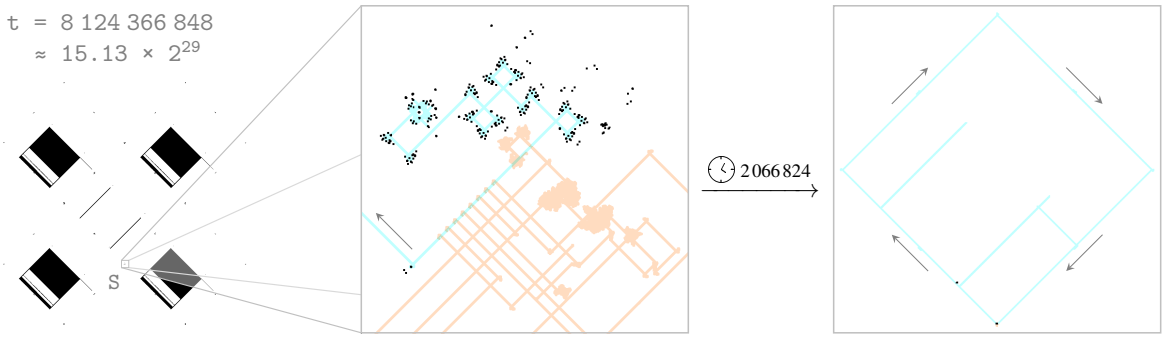


Figure 12.48: The parent's control clock gun has destroyed most of its southern corner. The Herschel that is now present in it will send a glider counter-clockwise around its outer edge, destroying almost every other piece of the metacell over the next 2 million generations.

12.8.9 Generations 16×2^{29} to 128×2^{29}: Waiting and Repeating

If a child metacell's state is non-zero, it now spends an extraordinarily long time doing nothing of note—from generation $t = 16 \times 2^{29}$ until $t = 64 \times 2^{29}$ the gliders from the period 2^{29} control clock are blocked by an obstruction, so the contents of its nucleus simply cycle 48 times without really doing anything (see Figure 12.49).

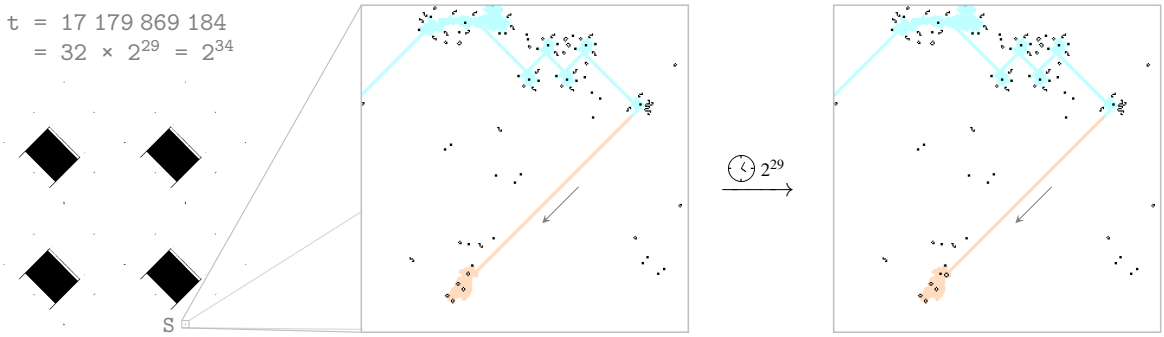


Figure 12.49: One of the blocks on the control clock gun's period 2^{29} output lane eats 48 gliders, and thus halts the metacell for 48×2^{29} generations. In the phase shown here, it has already absorbed 16 gliders, it is about to absorb another one, and it will then absorb 31 more much later.

On the other hand, if a child's state equals 0, then the Herschel that is present in its south control clock at $t = 16 \times 2^{29}$ releases a glider that uses one-time circuitry to (a) destroy the aforementioned slow-salvo seed, and (b) empty the contents of its nucleus. The result of (b) is that this metacell will not construct any children of its own, and the result of (a) is that it will self-destruct 47×2^{29} generations sooner than it would otherwise (i.e., it acts as a "dead" cell). In the example that we are illustrating, none of the child metacells have state 0, so this early death does not occur. However, we will see it occur in the next half-metageneration in Figure 12.50, 2^{35} generations from now.

Free PDF. See conwaylife.com/book for pattern files and print version.

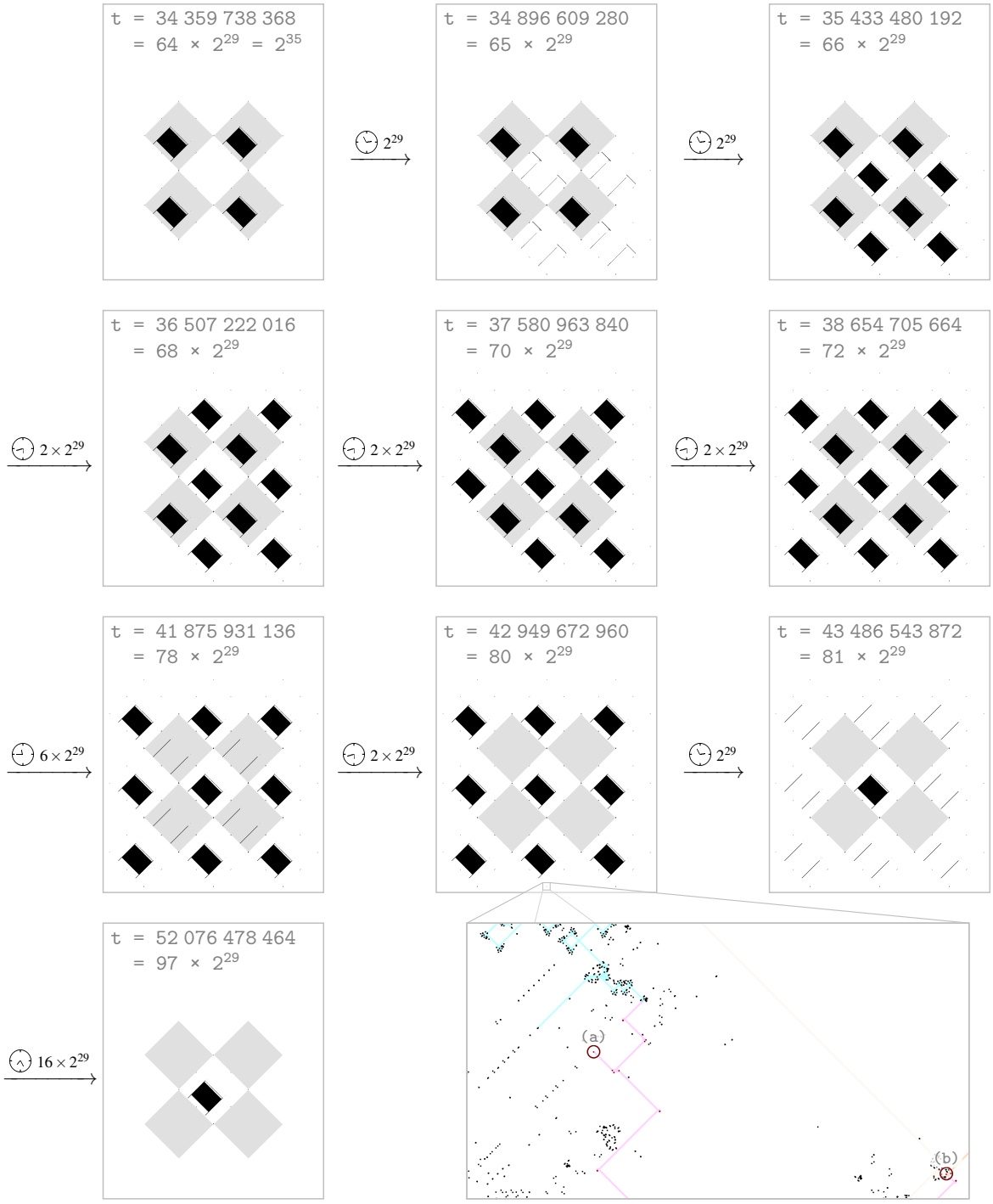


Figure 12.50: The four children are now parents, so they construct their own children via the exact same stages that their parents used. In this example, 8 of the 9 grandchildren have state 0, so they die early thanks to the Herschels that are present in their control clock guns at generation 80×2^{29}. Those Herschels create gliders that destroy (a) blocks that would otherwise cause the grandchildren to wait for 48×2^{29} generations, and (b) blocks in syringes, so that the contents of their nuclei are absorbed by those syringes' eater 2s.

Free PDF. See conwaylife.com/book for pattern files and print version.

At generation $t = 34\,359\,738\,368 = 64 \times 2^{29} = 2^{35}$, one half of a metageneration of the Moore-neighborhood cellular automaton (or equivalently, a full metageneration of the von-Neumann-neighborhood cellular automaton) has been completed, and child metacells are now in the same phase that the parent metacells were in at $t = 0$. All of the steps that we have seen so far now repeat, as these child metacells now start acting like parents (see Figure 12.50).

Only two aspects of this second half-metageneration differ from what we saw in the first half-metageneration:[33]

- **Multiple parents**: Some children are now located in positions that are adjacent to two or more parents. To prevent a parent from trying to construct a child that has already been constructed by another parent, the shell of each metacell simply has an eater in the path of incoming construction recipes, preventing them from doing anything.
- **Early death**: Some children are constructed with state 0, so they die early.

After all of this is complete, at $t = 68\,719\,476\,736 = 128 \times 2^{29} = 2^{36}$, a complete metageneration of the Moore-neighborhood rule is complete. In the particular rule that we are emulating here, a single live cell simply lives from one generation to the next, so we are left with a single copy of the 0E0P metacell that is identical to the one that we had at $t = 0$ (see Figure 12.51).

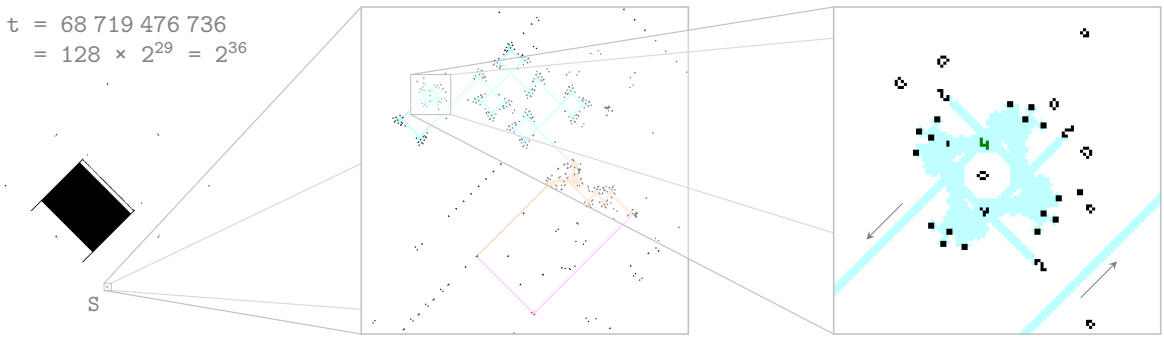


Figure 12.51: A full metageneration of the 2-state Moore-neighborhood rule has passed. In the particular rule being emulated here, a single cell lives from one generation to the next, so this snapshot is identical to the one from Figure 12.25: a Herschel is present in the clock gun which will release a glider that will destroy an eater, eventually leading to the construction of its southeast child.

[33] Both of these differences *can* happen in the first half-metageneration, but did not for us since we are only emulating a single cell.

Free PDF. See conwaylife.com/book for pattern files and print version.

12.9 Notes and Historical Remarks

Many **metacells**—patterns of size larger than 1×1 that emulate the behavior of a single cell—were constructed prior to 0E0P. The first one was the **p5760 metacell**, which was constructed by David Bell in January 1996. This metacell is much smaller and faster than 0E0P, with a period of just 5 760 generations and a bounding box size of just 500×500 (see Figure 12.52). However, there are numerous trade-offs that make this metacell less useful:

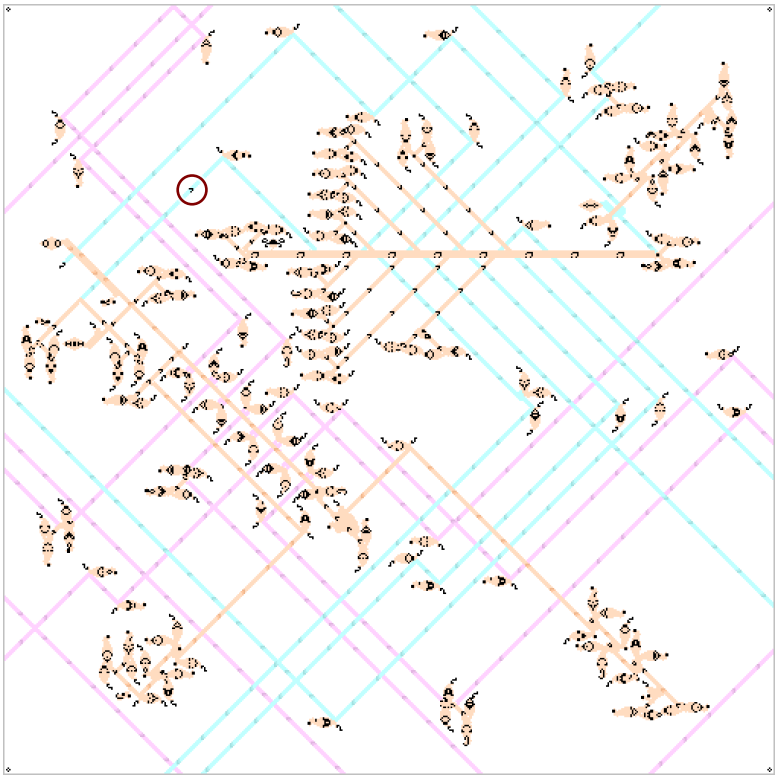


Figure 12.52: The **p5760 metacell**. Whether the cell is considered "alive" or "dead" is determined by the presence or absence of a glider at the location circled in red. That glider, if present, is duplicated 8 times and sent to its neighbors along the 8 output paths highlighted in aqua, signaling to them that it is alive. Similarly, neighboring alive cells send their signals to this one along the input paths highlighted in magenta.

- **Rule emulation**: It is hard-wired to emulate Life, and cannot easily be modified to emulate most of the other 2^{511} different zero-preserving 2-state Moore-neighborhood cellular automata.
- **Visualization**: It is not easy to tell at a glance which "cells" are alive and which are dead—it is determined by the presence or absence of a single extra glider in the cell—and it is thus not interesting to look at from a far-out zoom level.
- **Background grid**: "Dead" metacells must be placed on the Life plane, which means that, for example, spaceships cannot be emulated by this metacell unless infinitely many copies of it are used.

The first two of these problems were solved by the **OTCA metapixel**, which was constructed by Brice Due from late 2005 to mid-2006. While this metacell is a bit larger and slower than the first metacell (it is 2048×2048 and has period 35 328), it can be used to emulate any of the 2^{18} different outer-totalistic (i.e., Life-like) cellular automata that were described in Section 12.1.1.[34] Indeed, built

[34] This property is what gives it its name: it is the **O**uter-**T**otalistic **C**ellular **A**utomata metapixel.

Free PDF. See conwaylife.com/book for pattern files and print version.

into its circuitry is an easily adjustable array of eaters that determine how many live neighboring OTCA metapixels should lead to the birth or survival of the current metacell (see Figure 12.53).

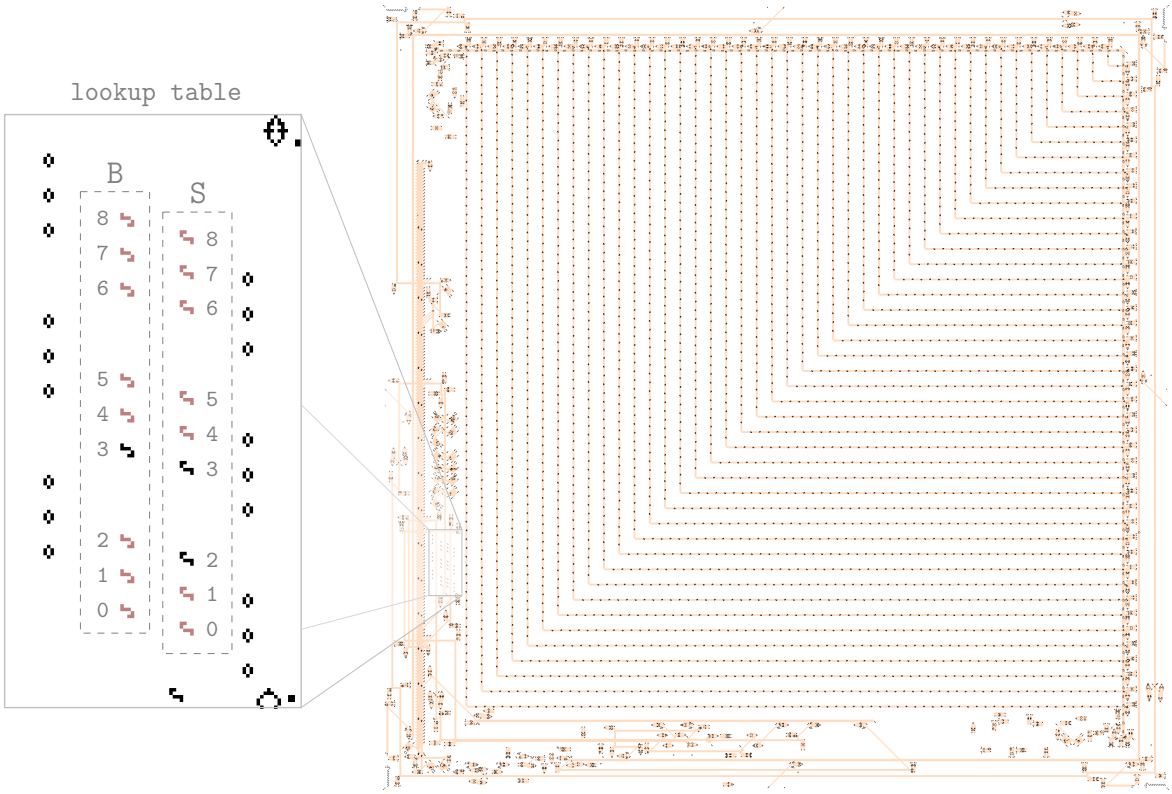


Figure 12.53: The **OTCA metapixel**, with orthogonal rows of lightweight spaceships filling up its central region, making it look "alive". By adjusting which eaters are present on its west edge, it can be made to emulate any Life-like cellular automaton. The eaters displayed in light red are not actually present—they are just displayed so as to show their potential locations—so this metacell is emulating the rule B3/S23 (i.e., Life).

The OTCA metapixel also has the remarkable feature that its alive and dead states look, from a distance, like alive and dead cells. This feature is achieved by the "alive" version of the cell releasing 43 pairs of perpendicular lightweight spaceship streams that mutually annihilate each other, thus partially filling in the otherwise empty center of the metacell. For example, arranging a 1×3 row of "alive" metapixels (and a suitably large "dead" array of metapixels around its edges) results in a pattern that looks and evolves like a blinker, but 2048 times as long and wide and 35328 times as slow (see Figure 12.54).[35]

The next notable metacell to be constructed was the **p1 megacell**, by Adam P. Goucher in 2008. This metacell was, again, larger and slower than the metacells that came before it, with a bounding box of $2^{15} \times 2^{15}$ and a period of $2^{24} = 16777216$. The new features this time that warranted the extra size and delay were twofold:

- **Stable construction:** This metacell was built entirely out of stable (p1) components like Herschel tracks, with the exception of a single period 2^{24} gun used to regulate its timing.[36] This

[35] Golly (golly.sourceforge.net/) comes with a Lua script called `metafier.lua` that automatically compiles metapatterns like these out of OTCA metapixels.

[36] This gun could be swapped out for a gun of another period, but periods that are multiples of 2 help the pattern run quicker under the HashLife algorithm in Life simulation software like Golly. This is the same reason that most of the 0E0P metacell's mechanisms are aligned to periods that are powers of 2.

Free PDF. See conwaylife.com/book for pattern files and print version.

contrasts with the OTCA metapixel, which is based almost entirely on p46 circuitry.

- **Non-outer-totalistic rules:** The p1 megacell can emulate all 2^{512} Moore-neighborhood 2-state cellular automata on a square grid, versus the 2^{18} outer-totalistic cellular automata that can be emulated by the OTCA metapixel.[37] It encodes these rules in a sequence of up to 512 eaters along one of its edges, with the presence or absence of those eaters indicating which of its 512 possible neighborhoods should lead to the megacell being "alive" in the next generation (see Figure 12.55).

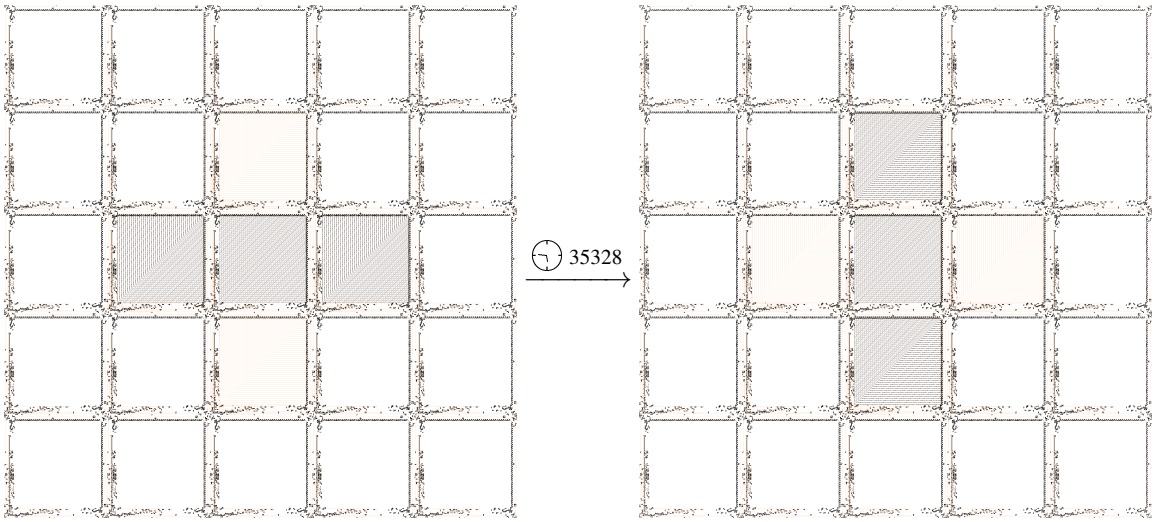


Figure 12.54: A **metablinker**: an arrangement of OTCA metapixels that emulates a blinker, but with period 2×35328.

When the diamond-shaped p1 megacell emulates another cellular automaton, it does so rotated 45 degrees clockwise (this is similar to how the 0E0P metacell emulates a von-Neumann-neighborhood CA, but *not* how it emulates a Moore-neighborhood CA).[38]

Finally, Adam P. Goucher spent 2014 through 2018 designing and constructing the 0E0P metacell, which solved the final problem described earlier—it does not require a background grid of "dead" cells to be placed on the Life plane. It achieves this extra feat by not just sending its state to its neighbors and then computing its own, but by also using universal construction to *build* those neighbors in the first place. Methods of universal construction made significant progress throughout the 0E0P metacell's development, and the advent of single-channel glider synthesis in 2017 provided the key breakthrough that made its completion possible.

[37] The p1 megacell can even emulate rules in which an all-dead neighborhood leads to a birth, unlike the 0E0P metacell. However, this requires that the entire Life plane be tiled with "dead" p1 megacells.

[38] A Lua script (for use in Golly) that "metafies" a pattern via the p1 megacell can be found at conwaylife.com/forums/viewtopic.php?p=40105#p40105.

Free PDF. See conwaylife.com/book for pattern files and print version.

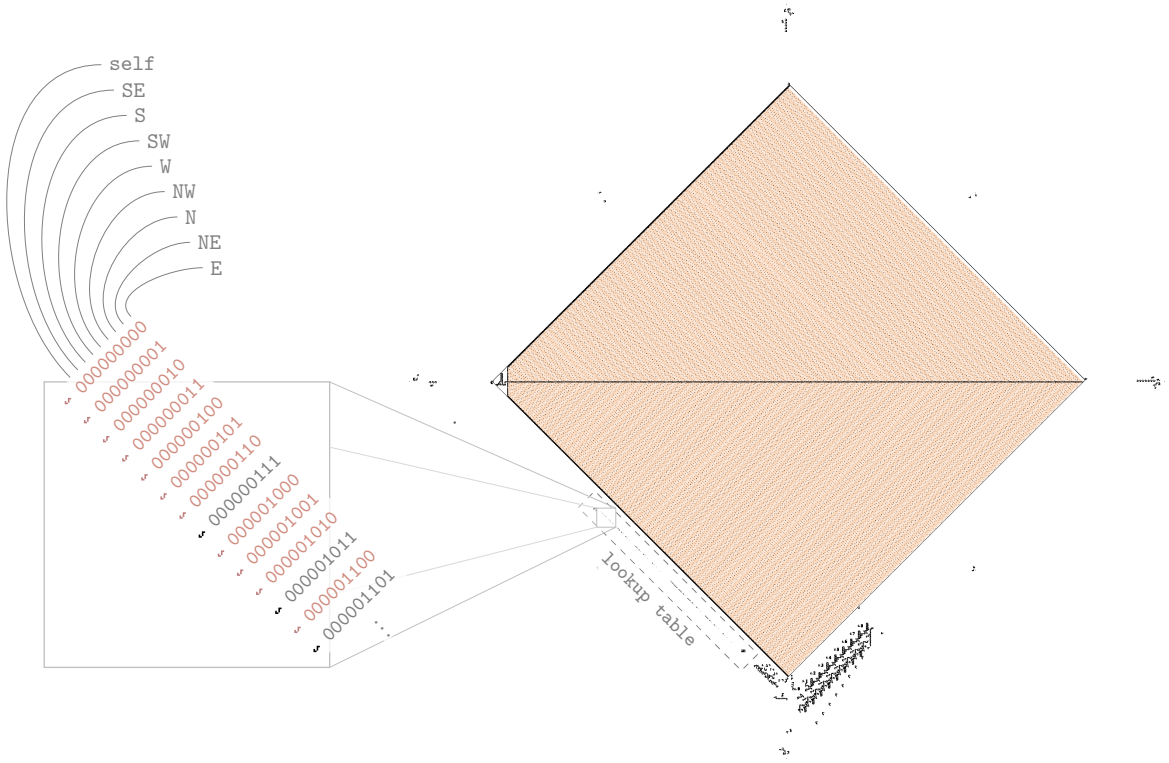


Figure 12.55: The **p1 megacell**, with numerous rows of gliders filling in a diamond-shaped region, making it look "alive". By adjusting which eaters are present on its southwest edge, it can be made to emulate any (not necessarily isotropic) 2-state Moore-neighborhood cellular automaton. The eaters displayed in light red are not actually present—they are just displayed so as to show their 512 potential locations, which are indexed by 9-bit bitstrings. Each bit corresponds to the state of a neighboring metacell, with the first (i.e., most significant) bit corresponding to its own state, the second bit corresponding to the southeast neighbor's state, and so on clockwise. The presence of an eater 1 indicates that the metacell will be "alive" in the next generation if its neighborhood states match that bitstring. Here, the eaters are encoding Life, and the first eater that is present at the "000000111" location indicates that the metacell will be born if its north, northeast, and east neighbors are the only ones that are alive.

Exercises

solutions to starred exercises on page 468

12.1 Recall the replicator rule B1357/S1357 that was illustrated in Figure 12.4.

- **(a)** [2/5] If the longest side of a pattern's bounding box is n cells long, how many generations would it take to copy itself for the first time in this rule?
- **(b)** [5/5] Prove that every pattern in this rule really does replicate, as we claimed.
  [Hint: First, prove (via induction, perhaps) that a single cell replicates. Then show that evolving a multi-cell pattern in this rule is equivalent to evolving each cell individually and XOR-ing the results together.].

12.2 Modify the rule used in Figure 12.11(b) so as to construct a (non-isotropic) 2-state cellular automaton on a square grid in which a single cell is a spaceship with speed...

- **(a)** [2/5] $c/2$ orthogonal;
- **(b)** [2/5] c diagonal;
- **(c)** [2/5] $c/2$ diagonal; and
- **(d)** [4/5] $(1,3)c/4$ or any other non-orthogonal, non-diagonal, and non-knightship slope.
  [Hint: Use two generations to transform the cell into a domino, and two more generations to transform it back into a single cell.]

Free PDF. See conwaylife.com/book for pattern files and print version.

12.3 [3/5] Copies of the 0E0P metacell can be placed on the Life plane so as to evolve in the same way as the pattern from Figure 12.9(a).

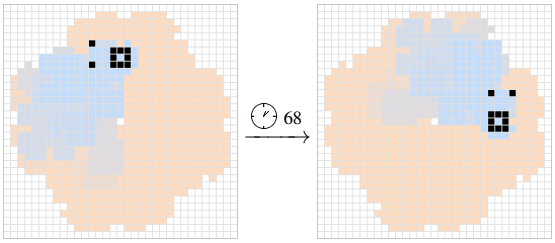
- **(a)** Explain why this pattern is *not* a reflectorless rotating oscillator in Life, despite being a RRO in the rule that the 0E0P is emulating.
- **(b)** Explain how you could use even more 0E0P metacells to emulate multiple copies of the pattern from Figure 12.9(a) so as to create an actual RRO in Life.

12.4 The 0E0P metacell from Figures 12.25–12.51 is emulating the isotropic rule

`B2-an3-eiky4aiqw5ijnr6ak`
`/S012ik3-acir4kqw5acq6ack7c8`.

- **(a)** [2/5] Find an RRO in this rule.
- **(b)** [3/5] Describe how you could use multiple copies of the 0E0P metacell (each emulating this rule) so as to create an RRO in Life.
  [Hint: Be careful and recall the solution of Exercise 12.3.]

∗**12.5** [3/5] There are currently no elementary RROs known in any Life-like cellular automaton. The period 272 oscillator from the rule `B02348/S0123` that is displayed below is *almost* an RRO, but is not quite. Explain why not.



∗**12.6** [3/5] What is the non-isotropic rulestring (in the sense of Appendix B.7) for Conway's Game of Life?

12.7 [3/5] Create a von-Neumann-neighborhood cellular automaton on a 2D square grid with at most 8 states, where every cell dies in every generation (so it is of the type described in Section 12.2 that can be emulated by the 0E0P metacell), in which two single-cell spaceships can collide so as to create a 2-cell SMOS.
[Hint: Make a cell in one state move south, a cell in another state move east, and something else happen when they collide.]

12.8 [4/5] We saw in this chapter that cellular automata using the von Neumann neighborhood can sometimes emulate cellular automata using the Moore neighborhood. In this exercise, we show that the converse is also true.

- **(a)** Explain how (2-state, on a 2D square grid) isotropic cellular automata that use the Moore neighborhood can emulate arbitrary (2-state, on a 2D square grid) outer totalistic cellular automata that use the von Neumann neighborhood.
- **(b)** Provide the isotropic rulestring of a Moore-neighborhood CA that emulates the von-Neumann-neighborhood rule `B23/S2V`.[39]

12.9 [5/5] The transition rule described by Figure 12.15 (or equivalently Equation (12.2)) is not the only one that leads to the function from Equation (12.3) being injective, and thus not the only one that can be used to emulate 2-state Moore-neighborhood CA via 8-state von-Neumann-neighborhood CA.

- **(a)** Find another transition rule that could be used instead, which cannot be obtained from the one in Figure 12.15 via rotation, reflection, and/or renumbering the states.
  [Hint: Write a computer script to help you.]
- **(b)** How many distinct (i.e., inequivalent up to rotation, reflection, and renumbering the states) transition rules are there that lead to the function from Equation (12.3) being injective?
- **(c)** Show that we really needed 8 states: show that if another transition rule is used in place of the one from Figure 12.15 that only has output states 0–5, then the function from Equation (12.3) is not injective.

∗**12.10** [2/5] Recall that a single glider can destroy a Snark, as long as we place some extra still lifes near it as in Figure 11.23(a).

- **(a)** Find a similar configuration of still lifes that is used in the 0E0P metacell so as to allow a single glider to destroy a Snark.
- **(b)** Explain why the configuration of still lifes from part (a) might be preferable to the one from Figure 11.23(a), despite being larger.

12.11 In this exercise, we explore how to use the semi-Snark-deletion technique of Figure 12.23.

- **(a)** [3/5] Modify the path-switching clock gun of Figure 12.20 so that, after the three path switches occur, the semi-Snarks all self-destruct.
- **(b)** [4/5] Modify that path-switching clock gun further so that, after the three path switches occur, *everything* other than the gliders from the single-channel glider stream is destroyed.
  [Hint: You can find configurations of still lifes and gliders that destroy the clock gun and syringe-based reflector in the 0E0P metacell.]

[39] The "`V`" at the end of this rulestring just indicates that the CA uses the von Neumann neighborhood.

Free PDF. See conwaylife.com/book for pattern files and print version.

12.12 Recall the state-lookup clock gun from Figure 12.21, which extracts two gliders from the single-channel glider sequence in its current configuration.

- **(a)** [2/5] If you remove a single block from the "a" side of the gun, how many gliders are extracted from the single-channel glider sequence?
- **(b)** [2/5] If you remove two, three, four, five, six, or all seven blocks from that same "a" side, how many gliders are extracted? How could you have predicted these answers without actually removing those blocks?
- **(c)** [2/5] Which configuration of blocks can you remove to extract the next earliest glider subsequence, which contains zero gliders?
- **(d)** [3/5] Which configuration of blocks can you remove to extract the 3 427th glider subsequence?
- **(e)** [3/5] Alter the single-channel glider sequence so that its 1 838th glider subsequence contains 7 gliders instead of just 1, and then alter the configuration of blocks so that the gun extracts those 7 gliders.

12.13 [2/5] In Figure 12.24, we can see that stages 1, 3, 5, and 7 of a metacell's lifecycle (i.e., the stages where its newly constructed children's nuclei are filled) are not initiated by the $p2^{29}$ clock gun—there are simply blocks in the output lane at those positions that absorb the output glider.
Explain what *other* mechanism is used to initiate those stages and redirect a copy of the single-channel glider recipe into the children's nuclei.

12.14 [3/5] Determine at which generation in an 0E0P metacell's lifecycle the following events occur (using the same choice of generation $t = 0$ as in Section 12.8).

- **(a)** The northeast child metacell's `CN` subroutine clock gun is initialized by a trigger glider.
- **(b)** The northwest child metacell's kernel finishes construction.
- **(c)** The final glider in the single-channel glider recipe enters the southwest child's nucleus via its `INS` location.
- **(d)** The northeast (instead of southwest) child metacell of Figure 12.47 stores its state by destroying 5 of its 8 state-storing blocks.

∗12.15 [2/5] In Figure 12.39, there is a one-time-turner that emits a glider to the south (first to the southeast, but it is then redirected to the southwest). What does this glider do, and what purpose does it serve?

∗12.16 [3/5] Suppose a newly constructed child 0E0P metacell has parent neighbors to the northeast, southeast, southwest, and northwest in states 2, 3, 4, and 5, respectively. Which 0–7 glider subsequence of its lookup table contains its state?

∗12.17 [3/5] Suppose a newly constructed child 0E0P metacell extracts its state from the 1 141th subsequence of its lookup table. What were its four parent neighbors' states?

12.18 [4/5] How many gliders would be present in the lookup table (i.e., all 4 095 subsequences together) of an 0E0P metacell that is programmed to emulate HighLife (i.e., B36/S23)?

12.19 [2/5] In Figure 12.52, the p5760 metacell is shown with 8 (aqua) output lanes, but more than 8 (magenta) input lanes. Explain this apparent discrepancy.

12.20 [3/5] Describe how the OTCA metapixel works. In particular, explain the mechanisms that it uses to send its state ("alive" or "dead") to its neighbors, and explain how those neighbors use the incoming states and the lookup table of Figure 12.53 to compute their states.

12.21 [4/5] Describe how the p1 megacell works. In particular, explain the mechanisms that it uses to sends its state ("alive" or "dead") to its neighbors, and explain how those neighbors use the incoming states and the lookup table of Figure 12.55 to compute their states.

12.22 [3/5] If you were to modify the p1 megacell's lookup table from Figure 12.55 so as to emulate HighLife (i.e., B36/S23) instead of Life, how many eater 1s would be present in it?

12.23 [3/5] Describe at least three different ways that the 0E0P metacell could be modified so as to be smaller and/or faster, without sacrificing any of its functionality.

Free PDF. See conwaylife.com/book for pattern files and print version.

Free PDF. See conwaylife.com/book for pattern files and print version.



Appendices and Supplements

A Mathematical Miscellany 431

A.1 Modular Congruence
A.2 Greatest Common Divisor and Least Common Multiple
A.3 Big-Θ Notation

B Extra Details 437

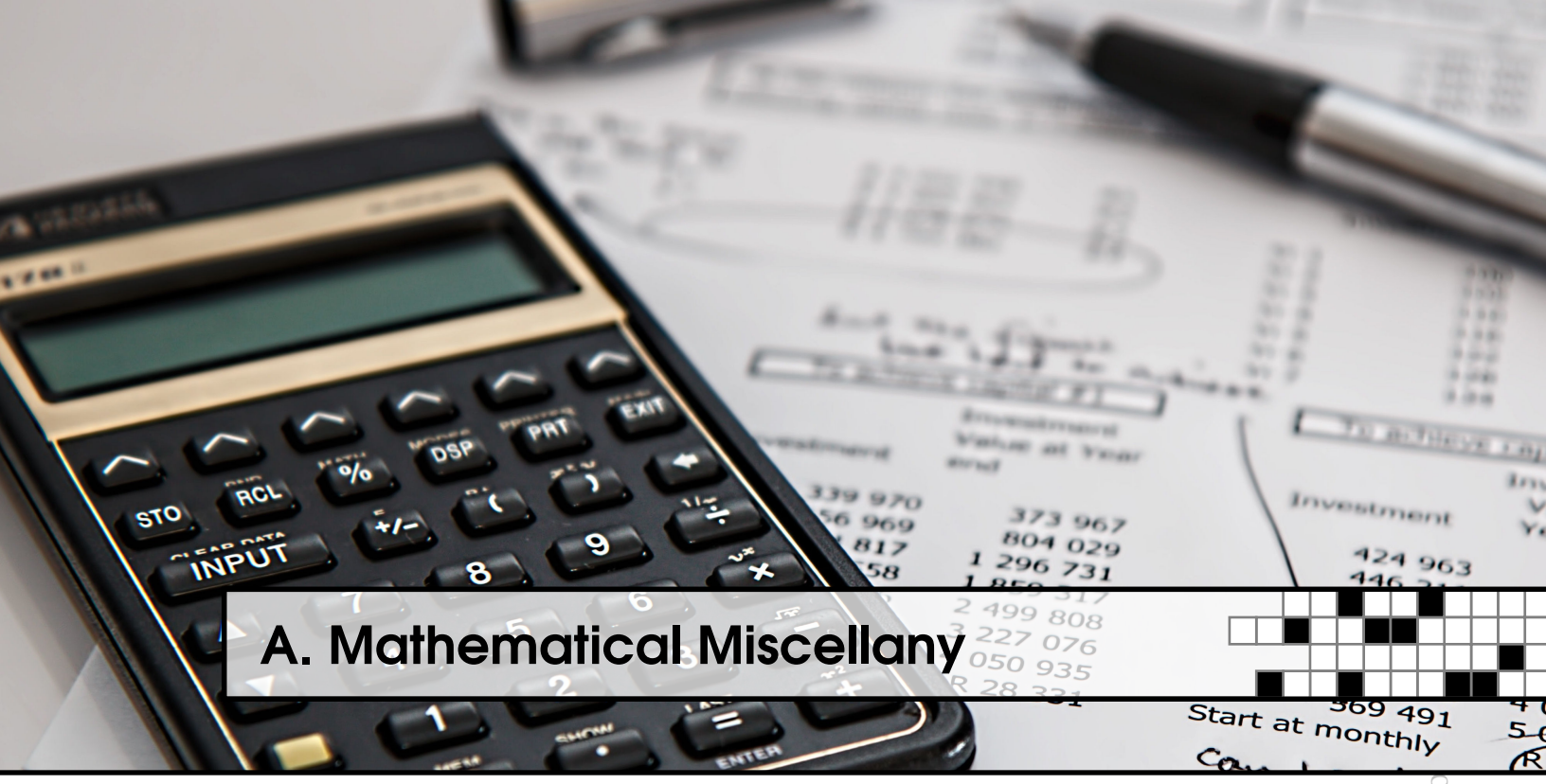
B.1 Universality of the Clock Inserter
B.2 Snarkmaker Timings
B.3 Scorbie Splitter Maker Timings
B.4 Scorbie Splitter Destroyer Timings
B.5 Cordership Maker Timings
B.6 Isotropic Rulestrings
B.7 Non-Isotropic Rulestrings
B.8 π Calculator APGsembly Code

C Solutions to Selected Exercises 451

Bibliography 469

Index 473

Free PDF. See conwaylife.com/book for pattern files and print version.



A. Mathematical Miscellany

In this appendix, we present some of the miscellaneous bits of mathematical knowledge that we need in order to discuss certain concepts or to prove certain theorems regarding the Game of Life in the main text. While none of these topics are particularly deep, they are often taught at scattered places throughout one's math education (or not taught at all), so we summarize them here for ease of reference.

A.1 Modular Congruence

When we divide two integers by each other, we get a **quotient** (i.e., the integer result of division) and a **remainder** (i.e., a piece that is left over from the integer division). For example, dividing 7 by 3 gives the quotient $q = \lfloor 7/3 \rfloor = 2$ with a remainder of $r = 1$. The number that we divide by (3 in this example) is called the **modulus**, and the remainder is always between 0 (inclusive) and the modulus (exclusive).

Modular congruence is a way of grouping numbers together based on their remainder upon division by a given modulus $n \geq 2$. For example, if the modulus is $n = 3$ then we say that 1, 4, 7, 10, and so on are all **congruent modulo 3**, since they all have the same remainder (1) upon division by 3. Equivalently, we say that two integers a and b are **congruent modulo n** if $a - b$ is an integer multiple of n, and in this case we write

$$a \equiv b \pmod{n}.$$

For example, $7 \equiv 1 \pmod{3}$ since $7 - 1 = 6$ is a multiple of 3.

Free PDF. See conwaylife.com/book for pattern files and print versio

Congruence modulo n provides a way partitioning the set of integers into n disjoint sets, called **congruence classes**, each consisting of all integers that are congruent to each other. For example, if $n = 2$ then there are two congruence classes: the set of even integers and the set of odd integers. If $n = 3$ then there are three congruence classes:[1]

$$\{\ldots,-6,-3,0,3,6,9,\ldots\}, \quad \{\ldots,-5,-2,1,4,7,10,\ldots\}, \quad \text{and} \quad \{\ldots,-4,-1,2,5,8,11,\ldots\}.$$

Since every integer a belongs to (exactly) one of the congruence classes, if we are given another member b of that congruence class then we can find an integer k so that $a = kn + b$.

One of the most prominent uses of modular congruence in the Game of Life arises when making timing and spacing adjustments in Life circuitry. For example, if we know of some way to delay a signal by 3 generations, then we can repeat that reaction n times in order to delay the signal by $3n$ generations. If we also know of methods of delaying that signal by, say, 5 and 7 generations, then we can delay it by any (sufficiently large) amount of our choosing: every mod-3 congruence class contains either 3, 5, or 7, so every integer can be written in the form $3n$, $3n+5$, or $3n+7$.

A slightly more realistic scenario that occurs in the Game of Life makes use of mod-8 arithmetic. It is often easy to delay a glider by any multiple of 8 generations, so to be able to implement arbitrary delays we "just" need to find ways of delaying it by amounts that cover the other 7 congruence classes. We collect glider-delaying reactions like this in two different places in the main text: Tables 5.5 and 7.2.

A.2 Greatest Common Divisor and Least Common Multiple

Suppose that we know of some repeatable way of delaying a signal in a Life circuit by 3 generations, and another (also repeatable) way of delaying it by 7 generations. Just these two delays are enough to implement a delay of *any* sufficiently large[2] number of generations. Indeed, every mod-3 congruence class contains either 0, 7, or 14, so we know from Section A.1 that we can write every integer in the form $3m + 7n$ for some integers m and n (and we can even choose $0 \le n \le 2$).

On the other hand, if we only know repeatable methods of delaying a signal in a Life circuit by 6 or 10 generations, for example, then we can only implement delays of an *even* number of generations. The problem here is that 6 and 10 are each even, so every number of the form $6m + 10n$ (where m and n are integers) is even as well. There was no such limitation when the delays were 3 and 7 generations, since there is no common number larger than 1 that they are each multiples of.

This line of thinking leads naturally to the **greatest common divisor** of two non-zero integers a and b: the largest positive integer that both a and b are integer multiples of, which we denote by $\gcd(a,b)$. For example, $\gcd(6,10) = 2$, since 6 and 10 are each multiples of 2, but they are not each multiples of any larger integer, while $\gcd(3,7) = 1$. The following theorem tells us that what we informally observed earlier really is true: the integers of the form $am + bn$ are exactly the multiples of $\gcd(a,b)$.[3]

[1] Negative integers can be congruent to each other, and do belong to congruence classes. For example, $7 \equiv -5 \pmod 3$ since $7 - (-5) = 12$ is a multiple of 3, so 7 and -5 belong to the same mod-3 congruence class.

[2] Specifically, 12 generations or more.

[3] We might have to choose m and/or n to be negative to achieve certain multiples of $\gcd(a,b)$, though. For example, no *positive* integer values of m and n give $3m + 7n = 11$: we need to allow one of them to be negative, as in the solution $m = 6$, $n = -1$.

Free PDF. See conwaylife.com/book for pattern files and print version.

Theorem A.1 — Bézout's identity
Let a, b, and c be non-zero integers. Then there exist integers m and n such that

$$am + bn = c$$

if and only if c is a multiple of $\gcd(a,b)$. Furthermore, if a and b have opposite signs then m and n can be chosen to both be positive.

Proof. To prove the "only if" direction of the theorem, note that a and b each being multiples of $\gcd(a,b)$ implies that am and bn are also multiples of $\gcd(a,b)$ for all integers m and n, so $am+bn=c$ is also a multiple of $\gcd(a,b)$.

To prove the "if" direction of the theorem, we suppose that c is a multiple of $\gcd(a,b)$, and we will show that there exist integers m and n such that $am+bn=c$. To this end, let m' and n' be integers that make $am'+bn'$ as small (but positive) as possible. For convenience, define $s = am'+bn'$ to be this minimal value. When dividing a by s, the remainder r is also of the form $r = am+bn$ since it is obtained by subtracting a multiple of $s = am'+bn'$ from a. Since the remainder satisfies $0 \le r < s$, and s is the smallest positive number of this form, the only possibility is that $r = 0$. In other words, s evenly divides a (and a similar argument shows that s evenly divides b).

It follows that c/s is an integer (since c is a multiple of $\gcd(a,b)$, which is a multiple of s), so choosing $m = m'(c/s)$ and $n = n'(c/s)$ gives $am+bn = (am'+bn')(c/s) = s(c/s) = c$, as desired.

To prove the final claim that m and n can be chosen to be positive when a and b have opposite signs, note that if m and n are integers such that $am+bn=c$ then for any integer k it is also the case that

$$a\left(m + \frac{kb}{\gcd(a,b)}\right) + b\left(n - \frac{ka}{\gcd(a,b)}\right) = c.$$

By taking k large enough and positive (if $a<0$ and $b>0$) or large enough and negative (if $a>0$ and $b<0$), this gives us a positive solution to the original equation $am+bn=c$. ■

In the special case when $\gcd(a,b) = 1$ (in which case a and b are said to be **relatively prime**), Theorem A.1 tells us that *every* integer c can be written in the form $c = am+bn$ for some integers m and n. Equivalently, every mod-b congruence class contains a multiple of a, and every mod-a congruence class contains a multiple of b.

Complementary to the greatest common divisor is the **least common multiple** of two positive integers a and b: the smallest positive integer that is a multiple of each of a and b, which we denote by $\operatorname{lcm}(a,b)$. For example, $\operatorname{lcm}(3,7) = 21$ and $\operatorname{lcm}(6,10) = 30$. It is straightforward to verify that $\operatorname{lcm}(a,b) = ab/\gcd(a,b)$ for all integers a and b.

Free PDF. See conwaylife.com/book for pattern files and print version.

A.3 Big-Θ Notation

When describing the long-term behavior of a Life pattern, we often just want to focus on the "big picture", while suppressing the "fine details". For example, the pattern displayed in Figure A.1 grows extremely quickly, filling the entire Life plane with zebra stripes as its four corners expand outward. Patterns that fill the plane like this are called **spacefillers**, and this one is called **max**.[4]

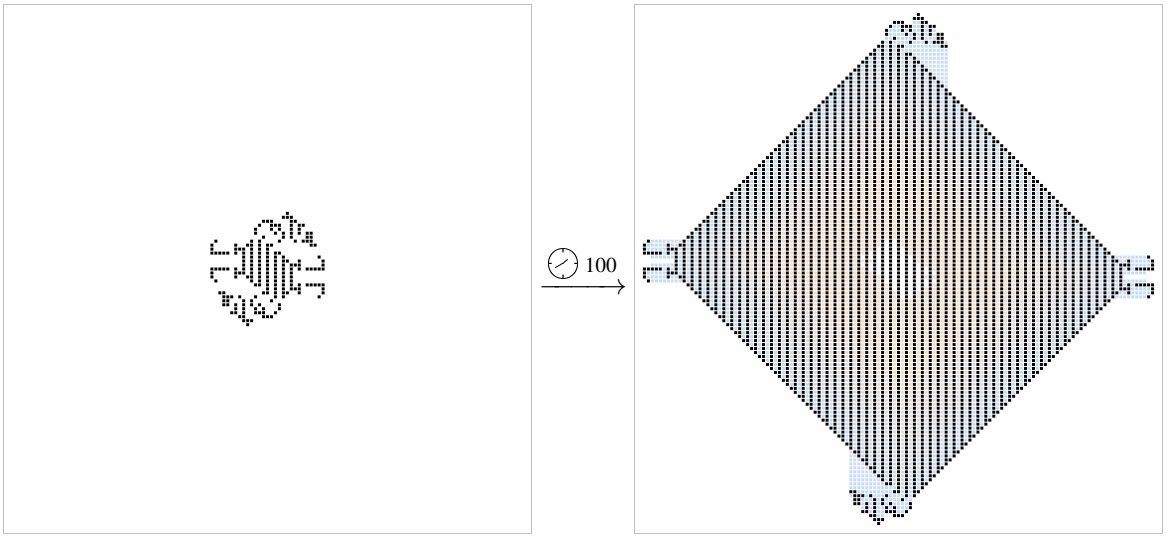


Figure A.1: The **max** spacefiller. Constructed by Tim Coe in October 1995.

To communicate how quickly this pattern grows, we could of course give an explicit formula for its population $p(t)$ in generation t, which has the following form:

$$p(t) = \frac{t^2}{4} + \begin{cases} 21t/2 + 209, & \text{if } t \equiv 0 \pmod 4 \\ 21t/2 + 215, & \text{if } t \equiv 2 \pmod 4 \\ 10t + 923/4, & \text{otherwise.} \end{cases} \tag{A.1}$$

However, this formula is quite technical and has many details that we typically do not actually care about. Its "most important" piece is the $t^2/4$ term at the front—in the long run (i.e., when t is large), that term has the biggest effect on the population. For this reason, we would typically just say that $p(t)$ "grows like $t^2/4$", or even that $p(t)$ "is proportional to t^2". Big-Θ notation provides a way of making this terminology precise:

Definition A.1 — Big-Θ Notation
Suppose f and g are real-valued functions. We say "$f(x)$ is $\Theta(g(x))$" if there exist positive scalars c, C, and N such that

$$cg(x) \le f(x) \le Cg(x) \quad \text{whenever} \quad x \ge N.$$

Informally, the phrase "$f(x)$ is $\Theta(g(x))$" means that f and g have the same rate of growth as x gets large, ignoring things like scalars and low-order terms. For example, for the population formula $p(t)$ given in Equation (A.1), we can say that $p(t)$ is $\Theta(t^2)$. This is hopefully somewhat intuitive, but it

[4] Its name is a reference to the fact that its growth rate is maximal—no pattern can expand faster than $c/2$ in each direction (see Theorem 4.1), and no pattern can tile the plane with a stable configuration that is more dense than zebra stripes (see Theorem 2.2).

Free PDF. See conwaylife.com/book for pattern files and print version.

can be made precise by choosing $c = 1/4$, $C = 1/2$, and $N = 57$ in Definition A.1—see Figure A.2.[5]

Some other examples of potential usage of big-Θ notation include:

- If $f(x) = 4x^3 + 3x^2 - 7x + 5$ then $f(x)$ is $\Theta(x^3)$. In fact, if f is *any* degree-n polynomial with positive leading coefficient then $f(x)$ is $\Theta(x^n)$.
- If $f(x) = x^3$ then $f(x)$ is $\Theta(4x^3 + 3x^2 - 7x + 5)$. However, this type of usage is extremely uncommon, if not completely non-existent. We typically (always?) want the function within the big Θ to be as simplified as possible.
- All logarithmic functions have the same growth rate as each other, in the sense that $\log_a(x)$ is $\Theta(\log_b(x))$ for all $a, b > 1$. Indeed, $\log_a(x) = (\log_a(b))\log_b(x)$, so logarithms only differ by the multiplicative scalar $\log_a(b)$, which is absorbed by the big Θ. For this reason, we typically just write $\Theta(\log(x))$ when discussing logarithmic growth rates, as the base makes no difference.

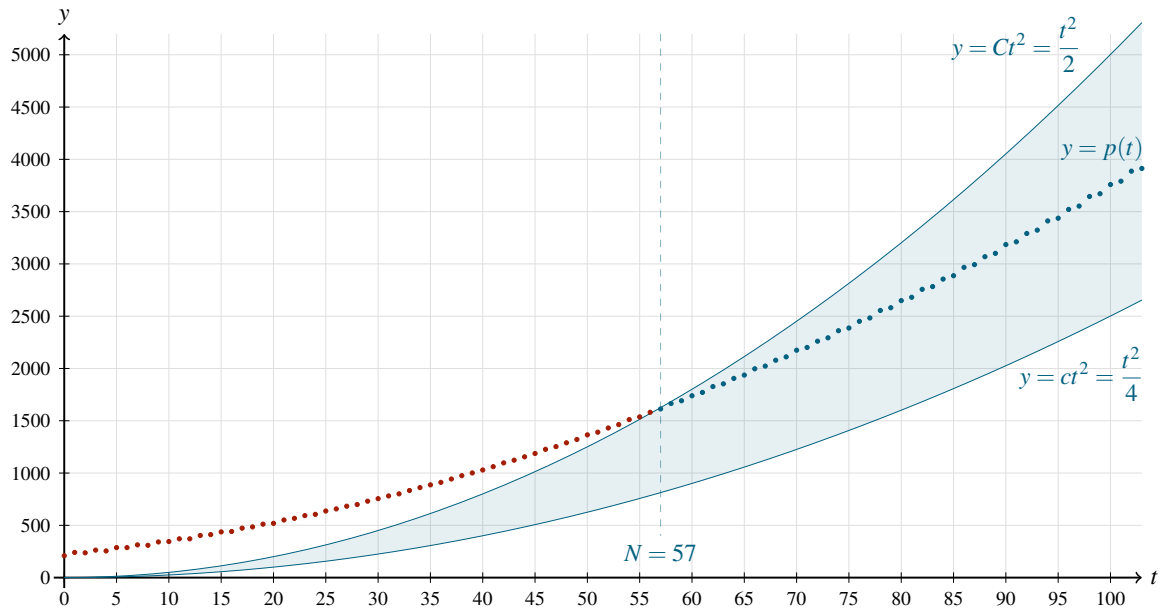


Figure A.2: A plot of the population $p(t)$ of the max pattern from Figure A.1 in generation t. Since $p(t)$ is between $t^2/4$ and $t^2/2$ when $t \geq 57$, we say that $p(t)$ is $\Theta(t^2)$.

[5]Many other choices of c, C, and N are possible here as well. For example, we could choose any smaller value of c and/or any larger value of C or N. We could even choose a smaller value of C as long as we choose a larger value of N to compensate.

Free PDF. See conwaylife.com/book for pattern files and print version.

Free PDF. See conwaylife.com/book for pattern files and print version.



B. Extra Details

In this appendix, we provide some details that makes theorems or pattern from the main text more precise, but whose details are so long, ugly, or technical that we prefer to hide them away.

B.1 Universality of the Clock Inserter

In Section 5.7, we showed that we can use a p1 slow salvo to simulate any glider synthesis in which the gliders are not "too close" together, and we suggested that the clock inserter from Figure 5.28 could be used to remove that final spacing restriction. Here we demonstrate the nitty-gritty details needed to show that the clock inserter really can create any tight packing of gliders that we desire, thus showing that *all* glider syntheses can be implemented via a p1 slow salvo.

B.1.1 Timing of Tight Glider Salvos

Recall the definitions of glider lanes and timing from Section 4.1.2, which give us a way to discuss how close gliders are to each other in both perpendicular directions. We now investigate exactly how tightly gliders in a salvo can be packed. For example, we mentioned in Section 3.5 that two gliders in the same lane must be at least 14 generations apart from each other or else they will collide. Similarly, it is straightforward to check that gliders that differ by $0, 1, 2, 3, 4, 5$, or 6 lanes must have their timings offset by at least $14, 14, 14, 13, 11, 9$, or 7 generations, respectively, as demonstrated in Figure B.1. On the other hand, gliders that differ by 7 lanes or more will never collide with each other, regardless of their timings, since the Moore neighborhoods of those lanes do not overlap enough to cause an unexpected birth between the gliders.

B.1.2 Timing of the Clock Inserter

The basic idea behind proving universality of the clock inserter is to show that we can use it to place gliders next to each other in each of the tight configurations from Figure B.1. However, this alone is not enough to prove the theorem, since a glider salvo might have dozens of tightly packed gliders, and a priori it's not obvious that being able to place any pair of gliders means that we can place any

Free PDF. See conwaylife.com/book for pattern files and print versio

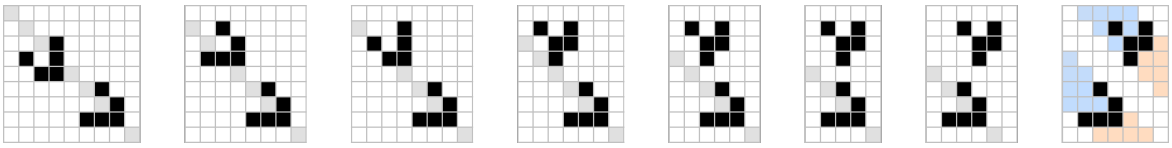


Figure B.1: The closest timings that are possible between two gliders in nearby lanes without colliding with each other. From left to right, these gliders differ by $0, 1, 2, 3, 4, 5$, and 6 lanes, and their timings differ by $14, 14, 14, 13, 11, 9$, and 7 generations. The rightmost image shows that gliders that differ by 7 lanes can never collide with each other.

complicated configuration of multiple gliders—it could be that no matter which order we place the gliders pairs in, we eventually get blocked off from placing one of the gliders that has other gliders on all sides of it.

To get around this problem, we establish a precise ordering that we will use to construct the salvo. Since the clock inserter is good at placing a glider closely in front of another glider, but bad at placing a glider closely behind another glider, we build the salvo from back-to-front. Specifically, we say that a glider's **rank** is its lane number plus its timing (see Figure B.2), and we construct the salvo in order from the lowest-ranked glider to the highest-ranked glider (breaking ties arbitrarily).[1]

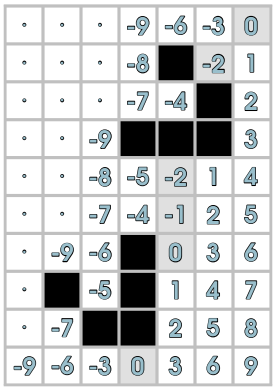


Figure B.2: Two gliders that both have a rank of 0. The numbers displayed on the grid show the rank of a glider when its leading cell (in the phase of the glider at the top) is at that location. The glider at the bottom has rank 0 since it will be in that phase in 2 generations, and its leading cell in that phase will be at a location labelled "2". Rank increases by 3 every cell to the right and by 1 every cell down, so gliders with constant rank lie on lines of slope 3 in the Life plane.

We need to show that if we insert gliders into the salvo according to their rank, then the (already-placed) low-rank gliders do not collide with the clock inserter reaction as we place the high-rank gliders. To this end, suppose that the glider we are inserting via the clock inserter is in lane 0 and has timing 0 (and thus rank 0). We will prove the result by contradiction, so let's assume that there is a glider in the salvo that has already been placed that collides with the clock inserter reaction. We derive the contradiction by splitting into two cases, based on the lane of the glider that collides with the clock inserter. Note that all lane and timing (and thus hence rank) numbers that we describe for this interfering glider are relative to the rank 0 glider that we are inserting.

Case 1: The colliding glider is in one of the lanes $-6, -5, \ldots, 5, 6$.

If the timing of the glider in a particular lane is small enough (i.e., far enough below 0), it will be

[1] While the lane and timing of a glider are concepts that are regularly used in the Life community, we have introduced the rank of a glider very specifically for this proof.

Free PDF. See conwaylife.com/book for pattern files and print version.

far to the top-left of the clock inserter reaction and thus not collide with it. It is straightforward (albeit somewhat tedious) to calculate the smallest possible timing of a glider in each lane $-6, -5, \ldots, 5, 6$ that *does* collide with the clock inserter, and these values are presented in Table B.1.

Lane:	0	1	2	3	4	5	6
Min. timing:	-13	-13	-13	-12	-10	-8	-6

Table B.1: The minimal timing that a glider in lanes 0 through 6 can have if it collides with the clock inserter reaction. The minimal timing for a glider in lane $-n$ in all of these cases is the same as the timing for a glider in lane n.

Since any pair of gliders with timing differences as in Table B.1 would also collide with each other (refer back to the timings listed in Figure B.1), we conclude that if a glider in one of these lanes collides with the clock inserter, then it would also collide with the output glider and thus those two gliders could not be part of the same salvo in the first place.

Case 2: The colliding glider is in one of the lanes $\ldots, -9, -8, -7$ or $7, 8, 9, \ldots$.

In a similar vein to that of case 1, we can calculate the minimum possible timing (and thus minimum rank) of a glider that collides with the clock inserter in each of these lanes, and these values are presented in Table B.2. In this case, however, we are interested in the rank of the colliding glider rather than its timing, and we notice that in each case, regardless of the glider's lane, its rank is strictly positive, which contradicts the fact that we are inserting gliders in order of increasing rank, and we are currently inserting a glider with rank 0.

Lane:	$-n$	-11	-10	-9	-8	-7	7	8	9	10	11	n
Min. timing:	$2n-5$	17	15	13	11	9	-3	15	17	19	21	$2n-1$
Min. rank:	$n-5$	6	5	4	3	2	4	23	26	29	32	$3n-1$

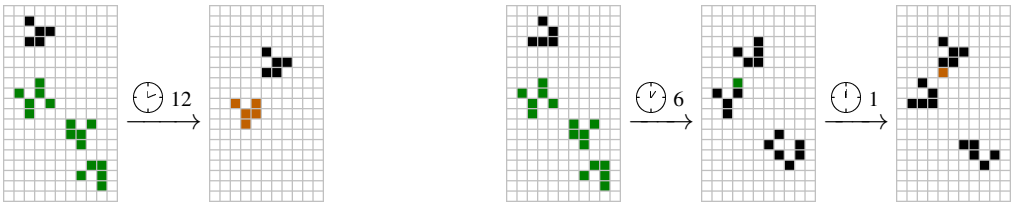
Table B.2: The smallest possible timing and rank that a glider in lanes $\ldots, -9, -8, -7$ or $7, 8, 9, \ldots$ can have if they collide with the clock inserter reaction. Importantly, the rank of the colliding glider is positive in all cases.

While this contradiction completes the proof of universality of the clock inserter, it is worth focusing on some of the values in Table B.2 and clarifying some points about the rank of a glider:

- First, the obvious pattern in Table B.1 of the minimal timing increasing by 2 generations every time the glider moves farther out by 1 lane does indeed continue forever—the collision that happens in these lanes occurs with one of the input gliders to the clock inserter rather than with the clock itself, which explains the regular pattern.
- Second, lane 7 is somewhat of an anomaly in Table B.2, with the minimal timing of the glider in that lane being much lower than in the other lanes. The reason for this oddity is that the clock in the clock inserter sticks one lane farther to the right the output glider does, which causes early collisions in this lane, and this lane only. This minimum-timing lane-7 glider collision is displayed in Figure B.3.

In fact, the anomaly in lane 7 is exactly why we insert gliders according to their rank rather than (for example) their timing. If we inserted gliders in order from low timing to high timing (i.e., according to diagonal lines of slope 1 in the Life plane) then we can run into problems when trying to insert gliders that are 7 lanes apart from each other. However, our choice of defining a glider's rank to

Free PDF. See conwaylife.com/book for pattern files and print version.



(a) Lane 7, timing −4. **(b)** Lane 7, timing −3.

Figure B.3: A clock inserter attempting to insert a glider in lane 0 at timing 0 when there is already a glider in lane 7 with (a) timing −4 and (b) timing −3. In (a), the new glider is successfully inserted, but in (b), a cell from the clock (shown in green in generation 6) interferes with the glider from lane 7, causing an extra cell to be born (shown in orange in generation 7).

equal its lane number plus its timing is by no means the only way of ranking gliders that works. Many other choices are possible, and each ranking function corresponds to lines of different slopes in the Life plane (see Exercise 5.43).

B.2 Snarkmaker Timings

In Section 11.4.1, we described how to build a single-channel recipe called a Snarkmaker that constructs a Snark along its path, while simultaneously moving the construction elbow to its far side. This recipe consists of 2427 gliders (specified by 2426 glider spacings), and is presented here:

109, 91, 93, 90, 132, 115, 127, 91, 90, 91, 95, 90, 114, 162, 233, 159, 90, 155, 126, 93, 118, 90, 91, 90, 1000, 109, 91, 94, 91, 91, 92,
90, 169, 91, 90, 116, 90, 113, 1000, 109, 91, 93, 91, 156, 91, 91, 94, 90, 91, 140, 91, 103, 91, 91, 132, 1000, 109, 90, 93, 91, 91, 90,
90, 100, 90, 90, 146, 96, 90, 90, 90, 92, 156, 144, 1000, 109, 91, 93, 91, 132, 115, 102, 90, 91, 91, 91, 90, 90, 154, 1000, 93, 91, 118,
91, 151, 90, 159, 91, 92, 90, 136, 90, 90, 154, 90, 101, 104, 165, 129, 91, 109, 91, 93, 91, 97, 90, 91, 111, 91, 116, 91, 94, 330, 91,
90, 95, 91, 90, 90, 91, 123, 90, 91, 152, 90, 90, 93, 91, 116, 91, 131, 91, 95, 188, 113, 91, 91, 147, 122, 91, 173, 91, 91, 133, 247, 92,
90, 109, 91, 93, 91, 129, 148, 91, 93, 154, 90, 134, 91, 91, 90, 91, 91, 111, 91, 91, 91, 91, 91, 109, 90, 93, 91, 91, 158, 94, 113, 91,
90, 91, 96, 90, 142, 91, 109, 91, 94, 91, 91, 179, 91, 90, 94, 91, 114, 90, 166, 90, 90, 90, 91, 117, 90, 96, 90, 90, 95, 91, 91, 109, 91,
93, 90, 156, 91, 91, 94, 91, 90, 147, 117, 91, 144, 90, 91, 128, 100, 91, 90, 105, 91, 91, 109, 91, 94, 91, 91, 124, 91, 105, 90, 169, 91,
90, 116, 91, 142, 90, 90, 91, 109, 91, 93, 91, 92, 91, 90, 90, 95, 102, 91, 91, 91, 130, 91, 90, 136, 91, 91, 119, 113, 90, 91, 114, 90,
109, 91, 94, 91, 91, 179, 91, 90, 94, 91, 102, 91, 151, 90, 90, 101, 90, 91, 125, 184, 90, 90, 90, 109, 91, 94, 91, 91, 179, 91, 90, 94,
91, 102, 91, 151, 90, 90, 101, 90, 91, 125, 184, 90, 90, 90, 109, 91, 93, 90, 140, 150, 132, 212, 103, 90, 98, 90, 148, 90, 90, 91, 91, 91,
119, 101, 108, 90, 91, 91, 119, 90, 109, 91, 94, 91, 90, 99, 90, 112, 90, 91, 105, 90, 121, 118, 103, 90, 144, 117, 95, 91, 109, 91, 93,
91, 92, 91, 90, 90, 95, 102, 91, 91, 91, 130, 91, 90, 136, 91, 91, 119, 113, 90, 91, 114, 91, 109, 90, 93, 91, 91, 181, 90, 95, 110, 114,
100, 160, 90, 143, 91, 119, 90, 106, 129, 109, 91, 93, 91, 92, 91, 90, 90, 162, 91, 91, 90, 129, 91, 113, 90, 90, 90, 90, 109, 91, 93, 90,
140, 150, 142, 91, 90, 111, 91, 91, 193, 97, 91, 91, 155, 90, 98, 90, 91, 93, 91, 151, 90, 139, 180, 103, 115, 167, 91, 120, 139, 135, 91,
91, 170, 109, 91, 93, 90, 155, 106, 91, 121, 90, 90, 91, 137, 90, 232, 90, 91, 91, 94, 90, 171, 90, 91, 103, 102, 109, 91, 93, 91, 137, 90,
166, 91, 102, 90, 104, 91, 96, 96, 91, 90, 90, 90, 166, 90, 90, 93, 90, 91, 109, 91, 94, 91, 91, 124, 91, 105, 91, 119, 91, 132, 99, 90,
90, 90, 150, 160, 116, 91, 91, 91, 90, 96, 90, 90, 109, 91, 93, 90, 171, 90, 90, 91, 90, 91, 90, 91, 129, 144, 90, 90, 120, 90, 91, 91,
169, 90, 91, 109, 91, 93, 91, 118, 90, 91, 91, 91, 104, 219, 91, 135, 105, 154, 90, 91, 164, 91, 132, 90, 90, 140, 94, 93, 90, 96, 90, 90,
91, 149, 90, 90, 161, 100, 109, 91, 93, 91, 92, 91, 90, 90, 124, 91, 142, 90, 90, 91, 91, 112, 90, 102, 102, 103, 90, 90, 90, 117, 112,
90, 189, 90, 90, 109, 91, 93, 91, 92, 91, 90, 90, 162, 91, 91, 90, 129, 91, 113, 90, 90, 90, 109, 91, 93, 91, 92, 90, 97, 91, 116, 91,
145, 90, 91, 98, 90, 90, 188, 91, 91, 91, 90, 115, 91, 109, 91, 93, 91, 97, 91, 90, 91, 120, 91, 117, 91, 123, 90, 118, 91, 146, 110, 160,
90, 109, 91, 93, 90, 129, 148, 90, 93, 90, 143, 96, 92, 90, 165, 90, 118, 90, 90, 91, 91, 109, 91, 94, 91, 91, 93, 90, 158, 90, 91, 90, 90,
116, 104, 109, 91, 94, 91, 91, 167, 90, 90, 91, 95, 90, 90, 148, 90, 151, 90, 90, 136, 134, 155, 115, 103, 91, 109, 91, 93, 90, 155, 106,
91, 121, 90, 90, 91, 137, 90, 232, 90, 91, 91, 94, 90, 171, 90, 91, 103, 101, 109, 91, 94, 91, 91, 136, 91, 91, 90, 168, 90, 90, 110, 90,
90, 93, 91, 111, 91, 91, 90, 132, 91, 91, 93, 91, 118, 90, 137, 91, 173, 93, 158, 90, 90, 90, 118, 90, 91, 90, 151, 154, 167, 91, 133, 90,
119, 178, 155, 90, 90, 90, 109, 91, 94, 91, 91, 95, 91, 90, 93, 218, 142, 90, 91, 161, 90, 138, 90, 162, 91, 90, 140, 95, 109, 109, 91, 93,
91, 92, 91, 98, 201, 91, 129, 90, 90, 90, 90, 90, 103, 90, 108, 90, 104, 90, 109, 91, 93, 90, 129, 148, 90, 93, 90, 143, 96, 92, 90, 165,
90, 118, 90, 90, 91, 91, 109, 91, 94, 91, 91, 179, 91, 90, 94, 91, 111, 90, 90, 90, 171, 91, 110, 91, 154, 90, 132, 91, 109, 91, 94, 91,
91, 124, 90, 144, 90, 90, 90, 165, 119, 90, 104, 90, 100, 90, 90, 91, 109, 91, 93, 91, 92, 91, 90, 90, 162, 91, 91, 90, 129, 91, 113, 90,
90, 90, 91, 109, 91, 94, 91, 91, 95, 91, 90, 93, 218, 172, 90, 90, 90, 116, 112, 341, 107, 106, 90, 163, 91, 90, 109, 91, 93, 90, 169, 90,
91, 103, 91, 133, 90, 90, 91, 91, 90, 110, 91, 93, 90, 112, 171, 90, 109, 91, 94, 91, 91, 171, 91, 90, 113, 90, 97, 114, 90, 105, 90, 139,
90, 113, 90, 106, 98, 121, 90, 109, 91, 94, 91, 91, 124, 90, 142, 90, 90, 146, 91, 153, 90, 102, 91, 152, 108, 97, 91, 109, 91, 94, 91, 91,
124, 90, 170, 90, 90, 91, 90, 99, 91, 90, 91, 110, 121, 161, 117, 115, 137, 90, 91, 90, 109, 90, 93, 91, 91, 128, 90, 139, 91, 90, 97, 91,
124, 157, 91, 90, 90, 129, 144, 91, 91, 147, 130, 91, 90, 90, 91, 90, 140, 90, 92, 90, 90, 109, 91, 93, 90, 156, 91, 91, 102, 91, 91, 90,
90, 106, 91, 166, 90, 125, 91, 90, 126, 91, 109, 91, 94, 91, 91, 179, 91, 90, 94, 91, 102, 91, 151, 90, 90, 101, 90, 91, 125, 184, 90, 90,
90, 93, 91, 151, 90, 139, 180, 103, 115, 167, 91, 120, 139, 135, 91, 91, 170, 109, 90, 93, 91, 91, 128, 90, 139, 91, 90, 97, 91, 124, 157,
91, 90, 90, 129, 144, 91, 91, 147, 130, 91, 90, 90, 91, 90, 140, 90, 92, 90, 90, 109, 91, 93, 91, 145, 215, 114, 91, 121, 91, 150, 91, 91,
153, 91, 141, 90, 91, 91, 90, 123, 91, 109, 90, 101, 169, 213, 133, 195, 90, 132, 143, 91, 139, 138, 158, 151, 99, 91, 108, 99, 91, 90, 91,
91, 90, 91, 131, 91, 109, 91, 93, 90, 156, 91, 91, 96, 132, 91, 91, 106, 91, 90, 119, 185, 91, 96, 90, 132, 90, 91, 90, 142, 109, 91, 94,
91, 91, 124, 90, 170, 90, 90, 91, 90, 99, 91, 90, 91, 110, 121, 161, 117, 115, 137, 90, 91, 90, 109, 91, 93, 90, 155, 106, 91, 121, 90, 90,
91, 137, 90, 232, 90, 91, 91, 94, 90, 171, 90, 91, 103, 102, 109, 91, 93, 90, 129, 148, 91, 102, 91, 91, 145, 178, 91, 115, 90, 90, 91,
104, 90, 90, 92, 249, 90, 90, 91, 109, 91, 94, 91, 90, 152, 91, 90, 91, 117, 90, 91, 111, 91, 91, 118, 90, 145, 90, 100, 116, 90, 90, 99,
90, 109, 91, 94, 91, 91, 128, 126, 90, 161, 151, 90, 109, 91, 90, 90, 94, 144, 106, 90, 94, 90, 90, 90, 109, 91, 94, 91, 91, 124, 91, 105,
90, 169, 91, 90, 116, 91, 142, 90, 90, 91, 109, 91, 93, 90, 140, 150, 108, 91, 90, 111, 91, 91, 194, 98, 90, 169, 90, 109, 91, 94, 91, 91,
141, 90, 171, 90, 155, 90, 111, 91, 90, 130, 90, 91, 90, 97, 90, 90, 109, 91, 94, 91, 91, 121, 90, 90, 90, 90, 90, 90, 99, 90, 165, 119,
90, 106, 90, 90, 91, 109, 91, 94, 91, 91, 93, 90, 95, 90, 113, 90, 99, 90, 156, 90, 90, 90, 138, 170, 109, 91, 94, 91, 91, 92, 90, 169, 90,
90, 90, 107, 90, 90, 91, 90, 95, 91, 91, 109, 91, 93, 90, 171, 90, 90, 91, 90, 91, 90, 91, 129, 144, 90, 90, 120, 90, 91, 91, 169, 90, 91,
109, 90, 95, 245, 90, 131, 135, 90, 90, 154, 90, 91, 91, 91, 111, 90, 90, 91, 91, 128, 91, 96, 91, 109, 91, 94, 91, 91, 124, 91, 105, 91,
119, 91, 132, 99, 90, 90, 90, 150, 160, 116, 91, 91, 91, 90, 96, 90, 91, 93, 91, 116, 91, 151, 90, 109, 111, 127, 91, 113, 91, 169, 186,
90, 90, 158, 91, 90, 90, 90, 117, 91, 160, 90, 91, 96, 90, 90, 91, 109, 91, 94, 91, 90, 95, 91, 90, 147, 167, 90, 160, 90, 160, 104, 90,

Free PDF. See conwaylife.com/book for pattern files and print version.

90, 91, 91, 101, 139, 91, 90, 136, 129, 90, 109, 91, 93, 91, 123, 91, 118, 90, 91, 108, 91, 91, 90, 90, 90, 90, 143, 91, 92, 177, 129, 101,
167, 91, 90, 90, 91, 130, 127, 90, 137, 91, 93, 90, 91, 91, 94, 229, 107, 91, 90, 104, 91, 91, 101, 91, 91, 93, 90, 119, 90, 133, 90, 91,
93, 145, 91, 132, 91, 109, 91, 93, 91, 137, 90, 166, 91, 102, 90, 104, 91, 96, 96, 91, 90, 90, 90, 166, 90, 90, 93, 90, 90, 109, 90, 93,
91, 91, 128, 90, 139, 91, 90, 97, 91, 124, 157, 91, 90, 90, 129, 144, 91, 91, 147, 130, 91, 90, 90, 91, 90, 140, 90, 92, 90, 91, 123, 270,
90, 125, 90, 90, 90, 94, 137, 123, 90, 145, 136, 90, 91, 100, 91, 105, 91, 153, 91, 90, 145, 155, 109, 91, 93, 91, 92, 91, 139, 90, 91, 91,
90, 96, 130, 97, 91, 164, 90, 97, 91, 90, 91, 114, 90, 90, 118, 90, 90, 123, 270, 90, 125, 90, 90, 90, 94, 137, 123, 90, 145, 136, 90, 91,
100, 91, 105, 91, 153, 91, 90, 145, 155, 109, 90, 93, 91, 91, 148, 91, 90, 151, 90, 91, 163, 108, 151, 112, 144, 90, 149, 90, 90, 99, 90,
109, 91, 94, 91, 91, 124, 91, 126, 91, 140, 162, 148, 90, 90, 119, 90, 91, 109, 91, 93, 91, 155, 106, 91, 91, 96, 90, 90, 91, 108, 90, 156,
90, 90, 120, 90, 112, 91, 99, 91, 109, 91, 93, 91, 129, 148, 91, 93, 154, 90, 134, 91, 91, 90, 91, 91, 111, 91, 91, 91, 91, 90, 109, 91,
93, 91, 129, 148, 91, 93, 154, 90, 134, 91, 91, 90, 91, 91, 111, 91, 91, 91, 91, 90, 109, 91, 93, 91, 129, 149, 91, 90, 90, 142, 219, 90,
99, 91, 109, 115, 92, 185, 91, 109, 90, 93, 91, 91, 142, 90, 98, 90, 91, 125, 114, 127, 90, 111, 90, 109, 91, 93, 91, 130, 91, 90, 134, 90,
90, 103, 122, 156, 112, 90, 183, 117, 91, 152, 141, 90, 98, 90, 91, 93, 91, 116, 91, 131, 91, 95, 188, 113, 91, 91, 147, 122, 91, 173, 91,
91, 133, 247, 92, 91, 109, 91, 93, 90, 156, 91, 91, 94, 91, 90, 147, 117, 91, 144, 90, 91, 128, 100, 91, 90, 105, 91, 91, 93, 91, 116, 91,
106, 91, 155, 90, 106, 90, 167, 90, 90, 91, 148, 123, 111, 155, 91, 105, 90, 90, 92, 90, 124, 90, 91, 109, 91, 94, 91, 91, 95, 91, 90, 97,
143, 171, 90, 105, 90, 91, 144, 91, 90, 90, 90, 94, 90, 90, 90, 109, 91, 93, 91, 92, 90, 158, 90, 94, 270, 172, 130, 90, 91, 91, 96, 90,
90, 147, 91, 109, 91, 93, 91, 92, 90, 162, 90, 129, 91, 91, 91, 90, 137, 99, 90, 90, 111, 91, 153, 90, 90, 90, 109, 91, 95, 125, 128, 90,
90, 90, 172, 90, 90, 90, 119, 91, 113, 247, 90, 144, 90, 140, 90, 109, 90, 93, 91, 90, 95, 91, 91, 139, 90, 147, 90, 90, 99, 117, 91, 157,
91, 126, 90, 90, 91, 160, 90, 91, 91, 91, 111, 90, 90, 113, 90, 91, 109, 91, 94, 91, 90, 99, 90, 112, 90, 91, 105, 90, 121, 118, 103, 90,
144, 117, 95, 91, 109, 91, 94, 91, 91, 124, 90, 144, 90, 90, 90, 165, 119, 90, 104, 90, 100, 90, 90, 90, 109, 91, 94, 91, 91, 179, 91, 90,
94, 91, 114, 90, 166, 90, 90, 90, 91, 117, 90, 96, 90, 90, 95, 91, 91, 109, 91, 94, 91, 91, 95, 91, 90, 150, 90, 140, 90, 91, 90, 171, 90,
118, 91, 111, 90, 104, 91, 109, 91, 93, 91, 97, 91, 90, 91, 120, 90, 95, 91, 143, 90, 90, 90, 90, 91, 109, 90, 95, 245, 90, 95, 90, 123,
91, 90, 115, 142, 91, 109, 91, 94, 91, 91, 124, 91, 90, 91, 91, 90, 91, 90, 141, 90, 172, 91, 161, 90, 169, 228, 90, 109, 91, 94, 91, 91,
93, 90, 91, 91, 90, 100, 90, 94, 90, 108, 90, 91, 91, 119, 1000, 109, 91, 95, 113, 90, 134, 90, 1000, 109, 90, 93, 91, 90, 95, 91, 91, 138,
157, 96, 90, 120, 91, 97, 107, 90, 90, 93, 188, 109, 90, 93, 91, 90, 95, 91, 91, 138, 157, 96, 90, 120, 91, 97, 107, 90, 90, 93, 188, 109,
90, 93, 91, 90, 95, 91, 91, 138, 157, 96, 90, 120, 91, 97, 107, 90, 90, 93, 188, 109, 91, 93, 91, 92, 90, 97, 91, 116, 91, 93, 115, 90, 91,
130, (90).

B.3 Scorbie Splitter Maker Timings

In Section 11.4.3, we described how to build a single-channel glider recipe that constructs a Scorbie splitter along its path. This recipe consists of 4 006 gliders (specified by 4 005 glider spacings), and is presented here:[2]

109, 90, 93, 91, 91, 90, 90, 100, 90, 90, 146, 96, 90, 90, 90, 92, 156, 144, 90, 109, 91, 93, 91, 132, 115, 102, 90, 91, 91, 91, 90, 90,
154, 99, 109, 90, 93, 91, 90, 100, 131, 151, 90, 91, 123, 112, 109, 100, 91, 91, 98, 91, 90, 90, 122, 91, 91, 91, 93, 91, 151, 90, 139,
180, 103, 115, 167, 91, 120, 139, 135, 91, 91, 171, 109, 91, 93, 91, 115, 107, 90, 90, 90, 90, 90, 90, 90, 103, 113, 118, 91, 130, 91, 93,
91, 116, 90, 106, 91, 143, 91, 109, 90, 91, 103, 110, 91, 136, 91, 92, 91, 155, 201, 109, 91, 94, 91, 91, 141, 91, 171, 90, 167, 90, 111,
91, 90, 130, 90, 91, 90, 97, 90, 91, 93, 90, 90, 90, 91, 90, 91, 136, 155, 98, 120, 90, 90, 91, 92, 90, 97, 156, 161, 141, 109, 91, 94, 91,
91, 124, 90, 140, 90, 91, 91, 141, 91, 165, 119, 90, 97, 91, 92, 91, 93, 91, 116, 90, 106, 91, 143, 91, 109, 90, 91, 103, 110, 91, 136, 91,
92, 91, 155, 200, 109, 90, 93, 91, 91, 135, 91, 124, 90, 90, 148, 91, 91, 97, 141, 91, 91, 109, 91, 94, 91, 91, 92, 90, 169, 90, 91, 90,
107, 90, 90, 91, 90, 95, 91, 90, 109, 91, 94, 91, 91, 141, 91, 171, 91, 90, 91, 107, 157, 121, 90, 90, 119, 91, 90, 109, 91, 95, 125, 128,
90, 90, 90, 172, 90, 90, 90, 119, 91, 113, 247, 90, 144, 90, 140, 91, 109, 91, 94, 91, 91, 124, 91, 105, 90, 169, 91, 90, 116, 91, 142, 91,
90, 90, 109, 91, 94, 91, 91, 153, 91, 91, 91, 90, 90, 91, 158, 91, 91, 166, 90, 91, 91, 91, 109, 91, 94, 91, 91, 93, 91, 91, 91, 90, 100,
90, 94, 90, 108, 90, 91, 91, 119, 90, 109, 91, 93, 91, 140, 150, 111, 100, 90, 90, 91, 160, 90, 91, 110, 91, 112, 91, 149, 112, 91, 91, 90,
91, 123, 90, 91, 109, 90, 93, 91, 91, 153, 91, 90, 91, 99, 90, 161, 90, 150, 91, 90, 126, 97, 99, 90, 148, 90, 100, 196, 117, 91, 90, 90,
91, 121, 91, 90, 157, 145, 90, 109, 91, 93, 91, 123, 91, 133, 91, 95, 91, 143, 91, 112, 90, 91, 110, 91, 90, 90, 91, 91, 107, 90, 98, 112,
91, 184, 90, 90, 109, 91, 94, 91, 90, 95, 91, 90, 147, 167, 91, 104, 149, 126, 91, 108, 91, 90, 152, 90, 103, 91, 192, 102, 91, 109, 91,
93, 91, 140, 150, 111, 100, 90, 90, 91, 160, 90, 91, 110, 91, 112, 91, 149, 112, 91, 91, 90, 91, 123, 90, 90, 109, 91, 93, 90, 140, 151,
121, 153, 91, 91, 91, 120, 91, 98, 90, 119, 91, 90, 108, 91, 93, 91, 109, 90, 106, 210, 141, 164, 91, 94, 135, 90, 90, 91, 93, 93, 115,
156, 91, 153, 90, 90, 112, 91, 91, 92, 91, 90, 90, 109, 91, 93, 90, 140, 150, 149, 91, 92, 129, 90, 91, 90, 114, 90, 90, 90, 93, 90, 90,
170, 122, 90, 91, 126, 90, 91, 90, 158, 142, 91, 91, 109, 91, 94, 91, 91, 124, 90, 142, 90, 90, 119, 91, 158, 90, 147, 91, 90, 90, 154,
91, 109, 91, 94, 91, 91, 95, 91, 90, 150, 90, 140, 90, 91, 90, 171, 90, 118, 91, 111, 90, 104, 91, 109, 91, 93, 91, 120, 90, 90, 100, 90,
90, 111, 178, 102, 184, 90, 103, 183, 91, 91, 96, 90, 90, 128, 90, 90, 91, 90, 166, 91, 109, 91, 93, 91, 123, 91, 103, 91, 90, 132, 139,
90, 124, 90, 91, 91, 158, 91, 171, 254, 91, 148, 176, 90, 131, 91, 90, 125, 91, 90, 104, 90, 109, 91, 94, 91, 91, 124, 90, 170, 90, 90,
91, 91, 99, 91, 90, 91, 110, 121, 161, 117, 115, 137, 90, 91, 91, 109, 91, 94, 91, 91, 167, 90, 90, 91, 95, 90, 90, 148, 90, 151, 90, 90,
136, 134, 155, 115, 103, 90, 109, 91, 93, 90, 140, 151, 121, 153, 91, 91, 91, 120, 91, 98, 90, 119, 91, 90, 108, 91, 109, 91, 94, 91, 91,
153, 91, 91, 91, 90, 90, 91, 158, 91, 91, 166, 90, 91, 91, 90, 109, 91, 93, 91, 120, 91, 91, 91, 91, 90, 91, 100, 91, 90, 97, 91, 91, 90,
90, 161, 109, 91, 94, 91, 91, 90, 91, 91, 90, 158, 90, 91, 90, 90, 101, 90, 107, 90, 90, 90, 90, 109, 91, 94, 91, 91, 121, 91, 90, 90, 90,
90, 90, 99, 91, 165, 119, 90, 106, 90, 90, 91, 109, 91, 94, 91, 91, 153, 91, 91, 91, 90, 90, 91, 158, 91, 91, 166, 90, 91, 91, 90, 109, 91,
93, 90, 155, 106, 91, 131, 97, 96, 151, 90, 90, 159, 151, 90, 90, 91, 93, 90, 140, 90, 91, 149, 136, 109, 91, 94, 91, 90, 95, 91, 90, 147,
167, 90, 160, 90, 160, 104, 90, 90, 91, 91, 101, 139, 91, 90, 136, 129, 90, 109, 91, 94, 91, 91, 93, 90, 97, 91, 91, 161, 90, 91, 90, 91,
90, 99, 90, 90, 99, 91, 90, 133, 91, 114, 91, 99, 90, 100, 94, 93, 91, 90, 144, 90, 93, 184, 162, 179, 111, 90, 93, 146, 90, 111, 91, 90,
90, 91, 141, 174, 90, 134, 159, 91, 91, 102, 91, 93, 91, 90, 144, 90, 93, 184, 162, 179, 111, 90, 93, 146, 90, 111, 91, 90, 90, 91, 141,
174, 90, 134, 159, 91, 91, 102, 91, 109, 90, 93, 91, 91, 167, 116, 91, 90, 116, 91, 111, 91, 145, 91, 139, 130, 275, 90, 91, 91, 159, 91,
130, 218, 171, 90, 92, 91, 91, 91, 93, 91, 118, 91, 151, 90, 148, 92, 116, 90, 118, 92, 90, 90, 91, 128, 90, 133, 153, 90, 91, 144, 91, 90,
128, 125, 90, 226, 91, 109, 91, 93, 91, 123, 91, 133, 91, 95, 91, 143, 91, 112, 90, 91, 110, 91, 90, 90, 91, 91, 107, 90, 98, 112, 91, 184,
90, 90, 109, 91, 93, 91, 171, 91, 90, 90, 151, 90, 111, 91, 146, 234, 90, 91, 145, 91, 90, 91, 103, 90, 94, 150, 141, 109, 90, 93, 91, 91,
107, 90, 98, 108, 102, 111, 105, 91, 104, 90, 104, 90, 90, 90, 91, 90, 91, 90, 116, 90, 90, 109, 90, 95, 245, 90, 95, 90, 123, 91, 90, 115,
142, 91, 109, 91, 93, 91, 123, 90, 147, 90, 91, 90, 107, 132, 90, 91, 96, 90, 164, 91, 91, 110, 91, 91, 90, 103, 90, 102, 91, 109, 91, 94,
91, 91, 93, 91, 158, 90, 91, 90, 90, 116, 105, 109, 91, 94, 91, 91, 93, 91, 95, 91, 113, 90, 99, 90, 156, 90, 90, 90, 138, 171, 109, 91,
93, 91, 145, 215, 114, 91, 121, 91, 150, 91, 91, 153, 91, 141, 90, 91, 91, 90, 123, 90, 109, 91, 94, 91, 90, 90, 90, 91, 109, 91, 101, 90,
98, 90, 90, 109, 90, 93, 91, 91, 158, 94, 113, 91, 90, 91, 96, 91, 142, 91, 109, 91, 94, 91, 91, 116, 90, 91, 91, 91, 90, 115, 91, 172, 90,
90, 91, 153, 91, 90, 90, 90, 90, 109, 91, 93, 91, 118, 90, 91, 91, 91, 90, 90, 163, 92, 91, 90, 103, 90, 91, 143, 91, 111, 148, 109, 91,
93, 91, 120, 90, 90, 119, 90, 108, 91, 118, 90, 91, 119, 109, 132, 170, 160, 91, 90, 91, 109, 91, 94, 91, 91, 167, 90, 90, 91, 95, 90, 90,
148, 90, 151, 90, 90, 136, 134, 155, 115, 103, 91, 109, 91, 93, 90, 140, 151, 121, 153, 91, 91, 91, 120, 91, 98, 90, 119, 91, 90, 108, 91,
109, 91, 93, 90, 171, 90, 90, 91, 90, 91, 129, 144, 90, 90, 120, 90, 91, 91, 169, 91, 90, 93, 91, 116, 90, 106, 91, 143, 91, 109,
90, 91, 103, 110, 91, 136, 91, 92, 91, 155, 201, 109, 91, 94, 91, 91, 141, 91, 171, 90, 167, 90, 111, 91, 90, 130, 90, 91, 90, 97, 90, 90,
109, 91, 94, 91, 91, 93, 91, 158, 90, 91, 90, 90, 116, 105, 109, 91, 93, 90, 171, 90, 90, 91, 90, 91, 90, 91, 129, 144, 90, 90, 120, 90,
91, 91, 169, 91, 90, 109, 91, 94, 91, 91, 124, 90, 144, 90, 90, 90, 165, 119, 90, 104, 90, 100, 90, 90, 90, 93, 91, 166, 131, 91, 98, 115,
139, 90, 126, 117, 106, 91, 126, 90, 124, 274, 91, 134, 90, 109, 91, 93, 90, 171, 90, 90, 91, 90, 91, 90, 91, 129, 144, 90, 90, 120, 90,
91, 91, 169, 91, 90, 109, 91, 93, 91, 92, 91, 90, 90, 162, 91, 91, 90, 129, 91, 113, 90, 90, 90, 91, 109, 91, 94, 91, 91, 96, 90, 97, 91,
91, 145, 90, 113, 90, 91, 105, 91, 193, 91, 109, 91, 94, 91, 91, 141, 91, 171, 91, 90, 91, 107, 157, 121, 90, 90, 119, 91, 91, 109, 91, 94,

[2]This recipe can be downloaded from conwaylife.com/book/universal_construction. Alternatively, if you are reading this book in Adobe Acrobat Reader, you can click here.

Free PDF. See conwaylife.com/book for pattern files and print version.

91, 91, 124, 91, 126, 91, 140, 162, 148, 90, 90, 119, 90, 91, 109, 91, 95, 125, 112, 91, 121, 120, 90, 90, 90, 90, 138, 90, 100, 90, 143, 91, 90, 129, 90, 123, 119, 91, 109, 91, 94, 91, 91, 96, 90, 112, 91, 143, 91, 90, 91, 145, 97, 195, 127, 136, 116, 91, 90, 90, 96, 187, 90, 91, 109, 91, 93, 90, 156, 91, 91, 102, 91, 91, 90, 90, 106, 91, 166, 90, 125, 91, 90, 126, 90, 109, 91, 93, 90, 137, 90, 110, 249, 91, 91, 103, 93, 91, 90, 90, 148, 91, 96, 237, 163, 91, 112, 91, 150, 91, 91, 100, 121, 109, 91, 93, 90, 171, 90, 91, 90, 151, 90, 111, 91, 146, 234, 91, 90, 90, 172, 90, 159, 118, 91, 301, 90, 90, 91, 91, 90, 100, 90, 155, 127, 101, 90, 109, 91, 93, 90, 140, 150, 132, 212, 103, 90, 98, 90, 148, 90, 90, 91, 91, 91, 119, 101, 108, 90, 91, 91, 119, 91, 109, 91, 94, 91, 91, 124, 91, 126, 91, 140, 162, 148, 90, 90, 119, 90, 91, 109, 91, 93, 90, 118, 91, 90, 90, 91, 152, 167, 99, 91, 151, 91, 91, 102, 91, 91, 91, 109, 91, 94, 91, 91, 124, 90, 144, 90, 90, 90, 165, 119, 90, 104, 90, 100, 90, 90, 90, 109, 91, 94, 91, 91, 179, 91, 90, 94, 91, 102, 91, 151, 90, 90, 101, 90, 91, 125, 184, 90, 90, 91, 109, 91, 93, 91, 123, 91, 103, 91, 90, 132, 139, 90, 124, 90, 91, 91, 158, 91, 171, 254, 91, 148, 176, 90, 131, 91, 90, 125, 91, 90, 104, 90, 109, 91, 94, 91, 90, 136, 90, 91, 90, 155, 90, 134, 91, 112, 91, 91, 113, 100, 116, 90, 116, 91, 90, 90, 141, 187, 91, 93, 151, 113, 90, 97, 90, 109, 91, 93, 90, 140, 150, 143, 91, 104, 91, 90, 91, 106, 109, 91, 99, 90, 90, 118, 160, 90, 99, 104, 214, 122, 91, 91, 181, 91, 90, 90, 167, 90, 96, 90, 109, 91, 94, 91, 91, 93, 91, 91, 91, 90, 100, 90, 94, 90, 108, 90, 91, 91, 119, 90, 124, 126, 91, 90, 91, 90, 90, 90, 94, 190, 104, 91, 90, 103, 94, 91, 90, 93, 90, 171, 91, 90, 103, 91, 91, 109, 91, 94, 91, 90, 96, 91, 91, 95, 90, 90, 91, 91, 115, 90, 139, 90, 90, 167, 91, 116, 90, 105, 109, 91, 93, 91, 92, 91, 90, 90, 162, 91, 91, 90, 129, 91, 113, 90, 90, 90, 91, 109, 91, 94, 91, 91, 124, 91, 90, 91, 91, 90, 91, 90, 141, 90, 172, 91, 161, 90, 169, 228, 91, 109, 91, 93, 90, 140, 150, 108, 91, 90, 111, 91, 91, 194, 98, 90, 169, 90, 109, 91, 93, 91, 92, 91, 90, 90, 162, 91, 91, 90, 129, 91, 113, 90, 90, 90, 90, 109, 91, 94, 91, 91, 93, 91, 95, 91, 113, 90, 99, 90, 156, 90, 90, 90, 138, 171, 93, 91, 116, 90, 106, 91, 143, 91, 109, 90, 91, 103, 110, 91, 136, 91, 92, 91, 155, 200, 109, 91, 94, 91, 91, 93, 91, 158, 90, 91, 90, 90, 116, 105, 109, 90, 93, 91, 91, 135, 91, 124, 90, 90, 148, 91, 91, 97, 141, 91, 90, 109, 90, 95, 245, 90, 95, 90, 123, 91, 90, 115, 142, 91, 109, 91, 94, 91, 91, 116, 90, 91, 91, 91, 90, 115, 91, 172, 90, 90, 91, 153, 91, 90, 90, 90, 90, 109, 91, 93, 91, 97, 91, 90, 91, 120, 90, 95, 91, 143, 90, 90, 90, 91, 158, 94, 113, 91, 90, 91, 90, 90, 90, 109, 91, 93, 91, 117, 90, 91, 90, 163, 91, 117, 91, 136, 90, 114, 300, 91, 91, 130, 90, 142, 90, 91, 94, 91, 109, 90, 93, 91, 91, 158, 94, 113, 91, 90, 91, 96, 91, 142, 90, 109, 91, 93, 90, 171, 90, 90, 91, 90, 91, 90, 91, 129, 144, 90, 90, 120, 90, 91, 91, 169, 91, 90, 109, 91, 94, 91, 91, 124, 90, 144, 90, 90, 90, 165, 119, 90, 104, 90, 100, 90, 90, 91, 109, 91, 94, 91, 91, 136, 90, 90, 91, 114, 91, 91, 90, 99, 90, 97, 90, 91, 90, 126, 90, 137, 90, 90, 109, 91, 94, 91, 91, 95, 91, 90, 150, 90, 140, 90, 91, 90, 171, 90, 118, 91, 111, 90, 104, 91, 109, 91, 93, 91, 97, 91, 90, 91, 120, 90, 95, 91, 143, 90, 90, 90, 90, 91, 109, 90, 95, 245, 90, 95, 90, 123, 91, 90, 115, 142, 90, 109, 91, 93, 91, 92, 90, 162, 90, 129, 91, 91, 91, 90, 137, 99, 90, 90, 111, 91, 153, 90, 90, 90, 109, 91, 94, 91, 91, 179, 91, 90, 94, 91, 114, 91, 166, 90, 90, 90, 91, 117, 90, 96, 90, 90, 95, 91, 90, 93, 91, 118, 91, 151, 90, 99, 153, 119, 94, 112, 90, 101, 90, 90, 111, 91, 159, 90, 100, 90, 93, 91, 109, 90, 106, 210, 141, 164, 91, 94, 135, 90, 90, 91, 93, 93, 115, 156, 91, 153, 90, 90, 112, 91, 91, 92, 91, 90, 91, 109, 91, 93, 91, 92, 91, 90, 90, 124, 91, 142, 90, 90, 91, 91, 112, 90, 102, 102, 103, 90, 90, 90, 117, 112, 90, 189, 90, 90, 109, 91, 93, 90, 155, 106, 91, 131, 97, 96, 151, 90, 90, 159, 151, 90, 90, 91, 93, 90, 140, 90, 91, 149, 136, 109, 91, 93, 90, 171, 90, 91, 90, 151, 90, 111, 91, 146, 234, 91, 90, 90, 172, 90, 159, 118, 91, 301, 90, 90, 91, 91, 90, 100, 90, 155, 127, 101, 90, 109, 91, 93, 91, 118, 90, 91, 91, 91, 104, 219, 91, 135, 105, 154, 90, 91, 164, 91, 132, 90, 90, 140, 94, 93, 90, 96, 90, 90, 91, 149, 90, 90, 161, 100, 109, 91, 94, 91, 90, 124, 91, 148, 90, 92, 168, 90, 91, 101, 90, 118, 90, 91, 91, 91, 91, 91, 102, 90, 90, 98, 90, 93, 91, 165, 146, 177, 91, 165, 283, 158, 91, 90, 90, 107, 91, 90, 91, 90, 91, 91, 161, 252, 109, 90, 91, 91, 131, 90, 106, 272, 170, 98, 135, 90, 90, 109, 91, 93, 91, 132, 115, 107, 90, 157, 278, 91, 94, 90, 142, 194, 103, 90, 138, 220, 108, 91, 101, 90, 91, 90, 90, 149, 90, 91, 159, 96, 90, 109, 91, 101, 169, 213, 120, 90, 150, 244, 97, 90, 163, 265, 90, 153, 174, 130, 167, 133, 263, 106, 90, 166, 120, 131, 91, 90, 120, 122, 147, 123, 90, 106, 112, 118, 126, 90, 91, 135, 109, 91, 99, 91, 90, 90, 90, 118, 96, 100, 124, 144, 91, 90, 90, 120, 91, 90, 91, 140, 91, 91, 91, 94, 90, 90, 126, 90, 161, 90, 92, 91, 145, 91, 109, 90, 93, 91, 91, 167, 116, 91, 90, 116, 91, 111, 91, 145, 91, 139, 130, 275, 90, 91, 91, 159, 91, 130, 218, 171, 90, 92, 91, 91, 91, 109, 90, 93, 91, 91, 151, 93, 107, 91, 91, 90, 90, 148, 91, 90, 144, 91, 99, 91, 129, 123, 170, 90, 109, 151, 112, 152, 91, 149, 90, 90, 125, 91, 100, 100, 109, 91, 94, 91, 90, 96, 90, 91, 146, 240, 109, 90, 93, 91, 91, 107, 90, 98, 108, 102, 111, 105, 91, 104, 90, 104, 90, 90, 90, 91, 90, 91, 90, 116, 90, 90, 109, 90, 95, 245, 90, 118, 91, 91, 91, 152, 91, 120, 91, 91, 101, 91, 112, 111, 90, 90, 93, 111, 90, 107, 90, 90, 90, 122, 142, 165, 91, 90, 91, 109, 91, 93, 90, 123, 91, 130, 99, 153, 107, 104, 133, 90, 119, 90, 90, 101, 172, 90, 125, 90, 91, 90, 90, 90, 93, 91, 116, 90, 154, 90, 90, 91, 124, 255, 160, 91, 91, 149, 90, 108, 90, 127, 96, 91, 122, 90, 109, 90, 93, 91, 91, 98, 90, 90, 100, 90, 90, 96, 90, 97, 163, 90, 98, 229, 117, 90, 104, 91, 91, 90, 90, 109, 91, 90, 109, 91, 93, 90, 140, 150, 136, 91, 91, 91, 90, 104, 91, 90, 176, 91, 91, 122, 127, 90, 92, 90, 91, 95, 90, 99, 97, 91, 142, 96, 104, 136, 93, 91, 166, 142, 90, 130, 209, 133, 183, 151, 90, 161, 103, 162, 91, 123, 91, 155, 90, 101, 91, 90, 120, 102, 90, 90, 91, 91, 91, 109, 91, 93, 90, 171, 90, 91, 90, 151, 90, 111, 91, 146, 234, 91, 90, 90, 172, 90, 159, 118, 91, 301, 90, 90, 91, 91, 90, 100, 90, 155, 127, 101, 90, 102, 121, 107, 90, 109, 145, 90, 134, 90, 181, 95, 285, 91, 90, 90, 157, 271, 90, 90, 115, 94, 116, 183, 112, 90, 170, 90, 91, 109, 91, 94, 91, 90, 96, 90, 91, 146, 240, 104, 191, 106, 91, 100, 91, 101, 122, 90, 90, 90, 90, 91, 156, 107, 91, 90, 151, 90, 91, 91, 143, 125, 97, 91, 117, 91, 98, 152, 91, 123, 271, 90, 125, 90, 91, 90, 155, 90, 120, 90, 104, 304, 90, 108, 91, 270, 127, 91, 99, 117, 90, 90, 196, 91, 91, 90, 124, 126, 90, 142, 91, 99, 135, 108, 90, 99, 90, 135, 90, 164, 175, 154, 91, 107, 144, 90, 129, 91, 163, 90, 90, 114, 247, 90, 147, 130, 91, 90, 118, 90, 109, 90, 93, 91, 91, 126, 90, 105, 90, 110, 90, 122, 91, 136, 91, 91, 90, 91, 90, 91, 104, 94, 90, 91, 109, 91, 95, 124, 120, 110, 91, 91, 91, 108, 182, 123, 91, 160, 90, 164, 90, 153, 107, 91, 97, 145, 91, 91, 119, 112, 104, 91, 114, 247, 90, 90, 138, 90, 90, 90, 103, 90, 109, 90, 93, 91, 91, 126, 90, 105, 90, 110, 90, 122, 91, 136, 91, 91, 90, 91, 90, 91, 104, 94, 90, 90, 93, 91, 163, 224, 90, 105, 90, 111, 91, 91, 113, 110, 114, 144, 90, 91, 209, 90, 171, 90, 93, 200, 90, 157, 91, 123, 90, 103, 90, 116, 90, 93, 91, 163, 224, 90, 105, 90, 111, 91, 91, 113, 110, 114, 144, 90, 91, 209, 90, 171, 90, 93, 200, 90, 157, 91, 123, 90, 103, 90, 116, 91, 109, 91, 93, 91, 123, 90, 99, 90, 90, 117, 90, 90, 91, 90, 104, 91, 121, 236, 119, 91, 108, 91, 120, 91, 90, 118, 91, 111, 90, 91, 147, 91, 90, 105, 219, 109, 91, 94, 91, 90, 96, 90, 91, 146, 240, 93, 91, 151, 91, 131, 153, 113, 91, 91, 141, 100, 90, 119, 295, 90, 131, 129, 99, 90, 117, 132, 90, 91, 142, 90, 92, 223, 94, 90, 144, 91, 136, 91, 116, 90, 93, 91, 151, 90, 139, 179, 94, 90, 94, 160, 117, 90, 90, 166, 91, 101, 90, 91, 100, 309, 90, 324, 90, 90, 109, 114, 91, 118, 225, 91, 90, 109, 234, 90, 91, 90, 95, 90, 134, 170, 93, 91, 165, 151, 90, 148, 190, 90, 152, 122, 122, 91, 158, 347, 132, 317, 114, 90, 127, 91, 133, 91, 95, 224, 366, 90, 111, 90, 150, 90, 91, 132, 90, 90, 91, 130, 91, 93, 91, 166, 96, 90, 90, 90, 161, 91, 124, 214, 132, 91, 90, 90, 91, 135, 337, 116, 107, 163, 90, 90, 116, 91, 102, 91, 143, 412, 103, 91, 91, 113, 91, 90, 91, 109, 91, 101, 169, 213, 120, 90, 150, 244, 97, 90, 163, 265, 90, 153, 174, 130, 167, 133, 263, 106, 90, 166, 120, 131, 91, 90, 120, 122, 147, 123, 90, 106, 112, 118, 126, 90, 91, 134, 109, 91, 93, 90, 97, 91, 91, 92, 159, 101, 285, 160, 90, 99, 127, 94, 90, 154, 90, 95, 91, 90, 101, 251, 135, 90, 166, 90, 142, 97, 166, 91, 91, 132, 90, 91, 140, 90, 154, 109, 90, 93, 91, 90, 100, 95, 90, 129, 91, 90, 113, 169, 90, 90, 91, 91, 91, 132, 133, 96, 152, 98, 90, 149, 129, 90, 102, 90, 91, 91, 90, 134, 91, 161, 174, 109, 91, 93, 90, 156, 91, 91, 94, 91, 90, 147, 117, 91, 144, 90, 91, 128, 100, 91, 90, 105, 91, 91, 93, 90, 90, 90, 91, 90, 91, 136, 155, 98, 120, 90, 90, 91, 92, 90, 97, 156, 161, 141, 109, 91, 93, 90, 156, 91, 91, 94, 91, 90, 147, 117, 91, 144, 90, 91, 128, 100, 91, 90, 105, 91, 91, 109, 91, 93, 91, 117, 90, 91, 90, 163, 91, 117, 91, 136, 90, 114, 300, 91, 91, 130, 90, 142, 90, 91, 94, 91, 109, 91, 94, 91, 90, 96, 90, 91, 146, 240, 109, 90, 93, 91, 91, 107, 90, 98, 108, 102, 111, 105, 91, 104, 90, 104, 90, 90, 90, 91, 90, 91, 90, 116, 90, 90, 109, 91, 94, 91, 91, 141, 91, 171, 91, 90, 91, 107, 157, 121, 90, 90, 119, 91, 91, 109, 90, 93, 91, 91, 158, 94, 113, 91, 90, 91, 96, 91, 142, 91, 109, 91, 94, 91, 91, 136, 90, 90, 91, 114, 91, 91, 90, 99, 90, 97, 90, 91, 90, 126, 90, 137, 90, 91, 109, 91, 94, 91, 91, 179, 91, 90, 94, 91, 102, 91, 151, 90, 90, 101, 90, 91, 125, 184, 90, 90, 90, 109, 91, 94, 91, 91, 124, 90, 140, 90, 91, 91, 141, 91, 165, 119, 90, 97, 91, 92, 91, 109, 91, 93, 91, 123, 90, 129, 90, 90, 111, 142, 91, 90, 120, 91, 142, 98, 93, 91, 166, 130, 90, 90, 113, 91, 90, 97, 169, 122, 91, 109, 112, 109, 91, 90, 119, 91, 105, 225, 90, 90, 136, 90, 151, 91, 90, 90, 134, 90, 124, 126, 91, 90, 91, 90, 90, 90, 94, 190, 104, 91, 90, 103, 94, 91, 90, 93, 90, 171, 91, 90, 103, 91, 90, 93, 91, 109, 90, 160, 158, 91, 112, 103, 95, 92, 147, 157, 141, 137, 91, 90, 90, 90, 90, 109, 91, 90, 135, 91, 91, 101, 166, 91, 97, 128, 90, 109, 91, 94, 91, 91, 141, 91, 136, 90, 90, 90, 140, 91, 131, 90, 91, 107, 90, 90, 91, 141, 90, 91, 109, 91, 93, 90, 169, 90, 91, 103, 91, 133, 90, 90, 91, 91, 90, 110, 91, 93, 90, 112, 172, 91, 109, 91, 95, 113, 90, 134, 90, (90).

B.4 Scorbie Splitter Destroyer Timings

We showed in Figure 11.17(b) how to destroy a Scorbie splitter via six slow xWSSes. The following 461-glider single-channel recipe generates those slow spaceships and thus implements that destruc-

Free PDF. See conwaylife.com/book for pattern files and print version.

tion:[3]

109, 91, 93, 91, 137, 90, 151, 91, 130, 90, 132, 128, 113, 118, 110, 90, 96, 90, 110, 91, 90, 90, 91, 125, 90, 106, 91, 90, 109, 91, 94, 91, 91, 92, 90, 97, 91, 90, 91, 90, 161, 91, 91, 91, 91, 91, 103, 130, 109, 91, 93, 91, 92, 90, 97, 91, 116, 91, 93, 115, 90, 91, 127, 90, 124, 112, 90, 124, 126, 91, 90, 91, 90, 90, 90, 148, 91, 91, 91, 91, 90, 101, 102, 98, 91, 90, 90, 169, 90, 128, 90, 109, 91, 93, 91, 92, 90, 97, 91, 116, 91, 93, 115, 90, 91, 127, 90, 124, 112, 90, 109, 91, 93, 91, 92, 90, 97, 91, 116, 91, 93, 115, 90, 91, 127, 90, 124, 112, 90, 124, 126, 91, 90, 91, 90, 90, 90, 148, 91, 91, 91, 91, 90, 101, 102, 98, 91, 90, 90, 169, 90, 128, 90, 109, 91, 94, 91, 91, 92, 90, 119, 90, 90, 109, 90, 93, 91, 90, 100, 131, 101, 90, 113, 95, 109, 91, 94, 91, 91, 92, 90, 119, 90, 90, 109, 90, 93, 91, 90, 100, 131, 101, 90, 113, 95, 109, 91, 93, 90, 97, 91, 91, 93, 91, 90, 90, 91, 123, 114, 92, 90, 90, 154, 90, 117, 126, 141, 90, 92, 106, 90, 109, 91, 93, 91, 123, 90, 119, 91, 90, 103, 102, 98, 94, 109, 157, 90, 167, 91, 153, 91, 90, 91, 110, 91, 90, 90, 90, 173, 90, 90, 91, 90, 90, 90, 109, 91, 94, 91, 91, 92, 90, 97, 91, 90, 91, 90, 161, 91, 91, 91, 91, 91, 103, 130, 109, 91, 94, 91, 91, 92, 90, 97, 91, 90, 91, 90, 161, 91, 91, 91, 91, 103, 130, 109, 91, 94, 91, 91, 92, 90, 97, 91, 90, 91, 90, 161, 91, 91, 91, 91, 91, 103, 130, 109, 91, 93, 91, 113, 91, 132, 91, 91, 133, 91, 90, 98, 91, 90, 131, 90, 112, 132, 90, 90, 124, 126, 91, 90, 91, 90, 90, 90, 148, 91, 91, 91, 91, 90, 101, 102, 98, 91, 90, 90, 169, 90, 128, 90, 109, 91, 94, 91, 90, 96, 90, 91, 146, 240, 109, 90, 93, 91, 91, 153, 91, 90, 91, 100, 90, 104, 95, 90, 93, 90, 90, 90, 110, 90, 109, 91, 94, 91, 91, 92, 90, 119, 90, 90, 109, 90, 93, 91, 90, 100, 131, 101, 90, 113, 95, 109, 91, 94, 91, 91, 92, 90, 119, 90, 90, 109, 90, 93, 91, 90, 100, 131, 101, 90, 113, 95, 109, 90, 93, 91, 90, 90, 90, (90).

B.5 Cordership Maker Timings

In Section 11.6, we described how to build a single-channel glider recipe that constructs a 2-engine Cordership that travels in the same direction as the glider recipe. This recipe consists of 2 393 gliders (specified by 2 392 glider spacings), and is presented here:[4]

109, 91, 94, 91, 91, 93, 90, 95, 90, 113, 90, 99, 90, 156, 90, 90, 90, 138, 170, 109, 91, 93, 91, 92, 91, 98, 201, 91, 129, 90, 90, 90, 90, 90, 103, 90, 108, 90, 104, 90, 109, 91, 93, 91, 92, 90, 97, 91, 116, 91, 145, 90, 91, 98, 90, 90, 188, 91, 91, 91, 90, 115, 90, 109, 91, 94, 91, 90, 116, 90, 90, 91, 95, 91, 90, 130, 90, 91, 130, 208, 91, 91, 109, 91, 95, 125, 128, 90, 90, 90, 172, 90, 90, 90, 119, 91, 113, 247, 90, 144, 90, 140, 91, 109, 91, 94, 91, 91, 124, 91, 126, 91, 140, 162, 148, 90, 90, 119, 90, 91, 109, 91, 93, 91, 97, 91, 90, 91, 120, 91, 117, 91, 123, 90, 118, 91, 146, 110, 160, 91, 109, 91, 93, 91, 127, 91, 90, 97, 91, 90, 104, 91, 90, 90, 115, 90, 103, 91, 91, 99, 91, 92, 90, 91, 109, 91, 93, 91, 92, 91, 90, 90, 162, 91, 91, 90, 129, 91, 113, 90, 90, 90, 91, 109, 91, 94, 91, 91, 167, 90, 90, 91, 95, 90, 90, 148, 90, 151, 90, 90, 136, 134, 155, 115, 103, 91, 109, 91, 94, 91, 91, 136, 91, 91, 90, 168, 90, 90, 110, 90, 90, 93, 91, 111, 91, 91, 90, 132, 91, 90, 109, 91, 93, 91, 129, 149, 91, 90, 90, 142, 219, 90, 99, 91, 109, 115, 92, 185, 91, 109, 90, 93, 91, 91, 158, 94, 113, 91, 90, 91, 96, 90, 142, 90, 109, 91, 93, 91, 127, 91, 90, 97, 91, 90, 104, 91, 90, 90, 115, 90, 103, 91, 91, 99, 91, 92, 90, 90, 109, 91, 94, 91, 91, 136, 90, 90, 91, 168, 90, 106, 90, 90, 138, 90, 90, 106, 91, 109, 91, 95, 125, 128, 90, 90, 90, 172, 90, 90, 90, 119, 91, 113, 247, 90, 144, 90, 140, 90, 109, 91, 93, 91, 92, 90, 162, 90, 129, 91, 91, 91, 90, 137, 99, 90, 90, 111, 91, 153, 90, 90, 90, 109, 91, 93, 91, 137, 90, 166, 91, 102, 90, 104, 91, 96, 96, 91, 90, 90, 90, 166, 90, 90, 93, 90, 90, 109, 91, 93, 91, 173, 90, 91, 90, 91, 90, 101, 91, 91, 91, 91, 91, 96, 143, 90, 91, 127, 109, 91, 93, 91, 97, 91, 90, 91, 120, 90, 95, 91, 143, 90, 90, 90, 90, 90, 109, 91, 94, 91, 90, 96, 90, 91, 158, 90, 90, 91, 93, 231, 91, 91, 90, 91, 91, 109, 91, 94, 91, 91, 167, 90, 90, 91, 95, 90, 90, 148, 90, 151, 90, 90, 136, 134, 155, 115, 103, 91, 109, 91, 93, 90, 140, 150, 142, 91, 90, 111, 91, 91, 193, 97, 91, 91, 155, 90, 98, 90, 91, 93, 91, 151, 90, 139, 180, 103, 115, 167, 91, 120, 139, 135, 91, 91, 169, 109, 91, 94, 91, 91, 124, 90, 170, 90, 90, 91, 90, 99, 91, 90, 91, 110, 121, 161, 117, 115, 137, 90, 91, 90, 109, 91, 93, 90, 140, 151, 145, 91, 94, 90, 90, 113, 91, 93, 91, 90, 93, 90, 100, 91, 99, 90, 91, 99, 91, 109, 91, 94, 91, 91, 136, 90, 90, 91, 171, 100, 91, 91, 90, 92, 96, 90, 101, 91, 109, 91, 94, 91, 91, 141, 90, 171, 91, 90, 91, 107, 157, 121, 90, 90, 119, 90, 91, 109, 91, 93, 90, 156, 91, 91, 94, 91, 90, 147, 117, 91, 144, 90, 91, 128, 100, 91, 90, 105, 91, 91, 109, 91, 94, 91, 91, 93, 90, 91, 91, 90, 100, 90, 94, 90, 108, 90, 91, 91, 119, 91, 109, 91, 94, 91, 91, 171, 91, 90, 113, 90, 97, 114, 90, 105, 90, 139, 90, 113, 90, 106, 98, 121, 91, 109, 91, 94, 91, 91, 124, 90, 142, 90, 90, 146, 91, 153, 90, 102, 91, 152, 108, 97, 91, 109, 91, 94, 91, 91, 167, 90, 90, 91, 95, 90, 90, 148, 90, 151, 90, 90, 136, 134, 155, 115, 103, 90, 109, 91, 93, 90, 140, 150, 108, 91, 90, 111, 91, 91, 194, 98, 90, 169, 90, 109, 91, 94, 91, 91, 92, 90, 146, 90, 90, 90, 91, 135, 91, 152, 135, 93, 90, 91, 91, 90, 90, 91, 90, 103, 113, 91, 103, 90, 152, 181, 140, 91, 90, 166, 91, 107, 109, 91, 94, 91, 91, 121, 90, 90, 90, 90, 90, 90, 99, 90, 165, 119, 90, 106, 90, 90, 90, 109, 91, 94, 91, 91, 95, 91, 90, 93, 218, 172, 90, 90, 90, 116, 112, 341, 107, 106, 90, 163, 91, 91, 109, 90, 93, 91, 91, 181, 90, 95, 110, 114, 100, 160, 90, 143, 91, 119, 90, 106, 128, 109, 91, 93, 90, 140, 150, 108, 91, 90, 111, 91, 91, 194, 98, 90, 169, 90, 109, 91, 93, 91, 117, 90, 91, 90, 163, 91, 117, 91, 136, 90, 114, 300, 91, 91, 130, 90, 142, 90, 91, 94, 90, 109, 91, 94, 91, 91, 167, 90, 90, 91, 95, 90, 90, 148, 90, 151, 90, 90, 136, 134, 155, 115, 103, 91, 109, 91, 93, 90, 140, 150, 108, 91, 90, 111, 91, 91, 194, 98, 90, 169, 91, 109, 91, 94, 91, 91, 92, 90, 146, 90, 90, 90, 91, 135, 91, 152, 135, 93, 91, 151, 90, 139, 180, 103, 115, 167, 91, 120, 139, 135, 91, 91, 170, 109, 91, 93, 91, 129, 148, 91, 93, 154, 90, 134, 91, 91, 90, 91, 91, 111, 91, 91, 91, 91, 109, 91, 94, 91, 91, 141, 90, 171, 91, 90, 91, 107, 157, 121, 90, 90, 119, 90, 90, 109, 91, 94, 91, 91, 179, 91, 90, 94, 91, 111, 90, 90, 90, 171, 91, 110, 91, 154, 90, 132, 90, 109, 91, 93, 91, 92, 90, 97, 91, 116, 91, 145, 90, 91, 98, 90, 90, 188, 91, 91, 91, 90, 115, 90, 109, 91, 93, 91, 117, 90, 91, 90, 163, 91, 117, 91, 136, 90, 114, 300, 91, 91, 130, 90, 142, 90, 91, 94, 91, 91, 123, 90, 147, 90, 91, 90, 140, 91, 94, 91, 152, 91, 90, 91, 106, 91, 135, 90, 98, 146, 90, 91, 109, 91, 94, 91, 91, 95, 91, 90, 97, 143, 171, 90, 105, 90, 91, 144, 91, 90, 90, 90, 94, 90, 90, 91, 109, 91, 93, 91, 92, 91, 98, 201, 91, 129, 90, 90, 90, 90, 90, 103, 90, 108, 90, 104, 91, 109, 91, 93, 91, 92, 90, 97, 91, 116, 91, 145, 90, 91, 98, 90, 90, 188, 91, 91, 91, 90, 115, 91, 109, 91, 93, 91, 92, 90, 97, 91, 116, 91, 145, 90, 91, 98, 90, 90, 188, 91, 91, 91, 90, 115, 91, 109, 91, 93, 91, 92, 90, 97, 91, 116, 91, 145, 90, 91, 98, 90, 90, 188, 91, 136, 90, 90, 91, 168, 90, 106, 90, 90, 138, 90, 90, 106, 90, 109, 91, 93, 91, 92, 90, 97, 91, 116, 91, 145, 90, 91, 98, 90, 90, 188, 91, 91, 91, 90, 115, 91, 109, 90, 93, 91, 91, 135, 91, 124, 90, 90, 148, 91, 91, 97, 141, 91, 91, 109, 91, 93, 90, 140, 150, 108, 91, 90, 111, 91, 91, 194, 98, 90, 169, 90, 109, 91, 94, 91, 91, 158, 90, 91, 90, 90, 101, 90, 107, 90, 90, 90, 91, 109, 91, 94, 91, 91, 92, 90, 169, 90, 90, 90, 107, 90, 90, 91, 90, 95, 91, 90, 109, 91, 94, 91, 90, 96, 90, 91, 158, 90, 90, 91, 93, 231, 91, 91, 90, 91, 91, 109, 91, 93, 91, 92, 91, 90, 90, 95, 102, 91, 91, 91, 130, 91, 90, 136, 91, 91, 119, 113, 90, 91, 114, 91, 109, 91, 94, 91, 91, 95, 91, 90, 93, 218, 172, 90, 90, 90, 116, 112, 341, 107, 106, 90, 163, 91, 90, 109, 91, 94, 91, 91, 121, 90, 90, 90, 90, 90, 99, 90, 165, 119, 90, 106, 90, 90, 90, 109, 91, 93, 90, 169, 90, 91, 103, 91, 133, 90, 90, 91, 91, 90, 110, 91, 93, 90, 112, 171, 91, 109, 91, 94, 91, 91, 153, 91, 91, 91, 90, 90, 91, 158, 91, 91, 166, 90, 91, 91, 90, 109, 91, 94, 91, 90, 116, 90, 90, 91, 95, 91, 90, 130, 90, 91, 130, 208, 91, 90, 109, 91, 93, 90, 129, 148, 90, 93, 90, 143, 96, 92, 90, 165, 90, 118, 90, 90, 91, 90, 109, 91, 93, 90, 140, 150, 108, 91, 90, 111, 91, 91, 194, 98, 90, 169, 90, 109, 91, 94, 91, 91, 96, 90, 166, 91, 91, 114, 90, 90, 91, 90, 90, 114, 91, 101, 90, 109, 91, 93, 90, 140, 150, 108, 91, 90, 111, 91, 91, 194, 98, 90, 169, 90, 109, 91, 93, 90, 140, 151, 145, 91, 94, 90, 90, 113, 91, 93, 91, 90, 93, 90, 100, 91, 99, 90, 91, 99, 90, 109, 91, 93, 91, 127, 91, 90, 97, 91, 90, 104, 91, 90, 90, 115, 90, 103, 91, 91, 99, 91, 92, 90, 90, 109, 91, 93, 91, 92, 91, 98, 201, 91, 129, 90, 90, 90, 90, 90, 103, 90, 108, 90, 104, 90, 109, 90, 93, 91, 91, 128, 91, 91, 90, 97, 90, 99, 90, 139, 91, 91, 117, 134, 92, 90, 90, 90, 90, 109, 91, 93, 91, 123, 90, 129, 91, 90, 104, 157, 90, 171, 91, 90, 90, 90, 90, 164, 94, 109, 91, 94, 91, 91, 167, 90, 90, 91, 95, 90, 90, 148, 90, 151, 90, 90, 136, 134, 155, 115, 103, 90, 109, 91, 93, 90, 140, 151, 145, 91, 94, 90, 90, 113, 91, 93, 91, 90, 93, 90, 100, 91, 99, 90, 91, 99, 91, 109, 91, 93, 91, 97, 91, 90, 91, 120, 90, 95, 91, 143, 90, 90, 90, 90, 90, 109, 91, 94, 91, 91, 124, 91, 105, 90, 169, 91, 90, 116, 91, 142, 90, 90, 91, 109, 90, 93, 91, 91, 158, 94, 113, 91, 90, 91, 96, 90, 142, 90, 109, 91, 94, 91, 91, 164, 91, 133, 124, 90, 90, 124, 90, 90, 91, 90, 90, 112, 91, 138, 91, 118, 90, 91, 109, 90, 93, 91, 90, 95, 91, 91, 139, 90,

[3]This recipe can be downloaded from conwaylife.com/book/universal_construction. Alternatively, if you are reading this book in Adobe Acrobat Reader, you can click here.

[4]This recipe can be downloaded from conwaylife.com/book/universal_construction. Alternatively, if you are reading this book in Adobe Acrobat Reader, you can click here.

Free PDF. See conwaylife.com/book for pattern files and print version.

147, 90, 90, 99, 117, 91, 157, 91, 126, 90, 90, 91, 160, 90, 91, 91, 91, 111, 90, 90, 113, 90, 91, 109, 91, 94, 91, 91, 95, 91, 90, 104, 90, 90, 97, 91, 91, 94, 191, 97, 90, 126, 90, 93, 91, 118, 91, 151, 90, 159, 91, 92, 90, 136, 90, 90, 154, 90, 101, 104, 165, 129, 90, 109, 90, 93, 91, 91, 181, 90, 95, 110, 114, 100, 160, 90, 143, 91, 119, 90, 106, 129, 109, 91, 94, 91, 91, 179, 91, 90, 94, 91, 111, 90, 90, 90, 171, 91, 110, 91, 154, 90, 132, 90, 109, 90, 93, 91, 91, 135, 91, 124, 90, 90, 148, 91, 91, 97, 141, 91, 90, 109, 90, 93, 91, 91, 158, 94, 113, 91, 90, 91, 96, 90, 142, 91, 109, 91, 93, 90, 123, 91, 103, 90, 91, 119, 90, 90, 172, 166, 90, 90, 138, 90, 95, 91, 90, 90, 96, 91, 109, 91, 93, 91, 137, 90, 166, 91, 102, 90, 104, 91, 96, 96, 91, 90, 90, 90, 166, 90, 90, 93, 90, 90, 109, 91, 94, 91, 91, 92, 90, 169, 90, 90, 90, 107, 90, 90, 91, 90, 95, 91, 90, 109, 91, 94, 91, 91, 141, 90, 171, 90, 155, 90, 111, 91, 90, 130, 90, 91, 90, 97, 90, 90, 109, 91, 94, 91, 91, 95, 91, 90, 93, 218, 172, 90, 90, 90, 116, 112, 341, 107, 106, 90, 163, 91, 91, 109, 91, 94, 91, 91, 167, 90, 90, 91, 95, 90, 90, 148, 90, 151, 90, 90, 136, 134, 155, 115, 103, 91, 109, 91, 93, 91, 97, 90, 91, 111, 91, 116, 91, 94, 330, 91, 90, 95, 91, 90, 90, 91, 123, 90, 91, 152, 90, 90, 109, 90, 93, 91, 91, 181, 90, 95, 110, 114, 100, 160, 90, 143, 91, 119, 90, 106, 128, 109, 91, 94, 91, 91, 93, 90, 158, 90, 91, 90, 90, 116, 104, 109, 91, 94, 91, 91, 92, 90, 169, 91, 90, 116, 90, 161, 91, 104, 90, 109, 91, 93, 90, 171, 90, 90, 91, 90, 91, 90, 91, 129, 144, 90, 90, 120, 90, 91, 91, 169, 90, 90, 109, 91, 93, 91, 123, 91, 118, 90, 91, 108, 91, 91, 90, 90, 90, 90, 143, 91, 92, 177, 129, 101, 167, 91, 90, 90, 91, 130, 127, 90, 137, 90, 109, 91, 93, 91, 92, 90, 158, 90, 94, 270, 172, 130, 90, 91, 91, 96, 90, 90, 147, 90, 109, 91, 93, 91, 123, 90, 147, 90, 91, 90, 140, 91, 94, 91, 152, 91, 90, 91, 106, 91, 135, 90, 98, 146, 90, 90, 93, 91, 118, 90, 137, 91, 173, 93, 158, 90, 90, 90, 118, 90, 91, 90, 151, 154, 167, 91, 101, 149, 115, 90, 95, 260, 213, 112, 90, 109, 91, 93, 90, 156, 91, 91, 94, 91, 90, 147, 117, 91, 144, 90, 91, 128, 100, 91, 90, 105, 91, 90, 109, 90, 93, 91, 91, 148, 91, 90, 151, 90, 91, 163, 108, 151, 112, 144, 90, 149, 90, 90, 99, 91, 109, 91, 94, 91, 91, 179, 91, 90, 94, 91, 114, 90, 166, 90, 90, 90, 91, 117, 90, 96, 90, 90, 95, 91, 91, 109, 91, 94, 91, 91, 96, 90, 112, 91, 143, 91, 90, 91, 145, 97, 195, 127, 136, 116, 91, 90, 90, 96, 187, 90, 91, 109, 91, 94, 91, 91, 96, 90, 112, 91, 143, 91, 90, 91, 145, 97, 195, 127, 136, 116, 91, 90, 90, 96, 187, 90, 90, 109, 91, 101, 169, 213, 133, 196, 91, 133, 151, 90, 109, 90, 98, 121, 138, 93, 91, 109, 91, 93, 90, 153, 143, 90, 112, 176, 90, 143, 91, 109, 91, 93, 91, 123, 90, 129, 90, 90, 111, 142, 91, 90, 120, 91, 142, 99, 109, 91, 94, 91, 90, 90, 90, 90, 109, 91, 101, 90, 98, 90, (90).

B.6 Isotropic Rulestrings

We now describe how to read and construct rulestrings that specify an isotropic 2-state cellular automaton that uses the Moore neighborhood on a 2D square grid. We start with a rulestring of the form `Bx/Sy`, where "`x`" and "`y`" are numeric strings describing how many neighbors lead to the **b**irth and **s**urvival of a cell, respectively. However, we then add modifiers after each integer in the rulestring to either add or remove some of those birth and survival conditions.[5]

The possible modifiers are each specified by a single letter, all of which are described in Table B.3. If a string of modifiers is preceded by a minus sign then the corresponding birth or survival conditions are being removed, rather than added. For example:

- **No modifiers**: If a rulestring contains "`B3`" (with no modifiers immediately after it) then cells are born if they have 3 live neighbors, regardless of their orientation.
- **Positive modifiers**: If a rulestring contains "`B3acj`" then cells are born if they have 3 live neighbors as in the "`a`", "`c`", or "`j`" configurations (up to rotation), but not if they have 3 live neighbors in any other configuration.
- **Negative modifiers**: If a rulestring contains "`B3-kr`" then cells are born if they have 3 live neighbors in any configuration *except* for the "`k`" and "`r`" configurations (up to rotation).

Every rule that can be specified via positive modifiers can also be specified via negative modifiers, and vice-versa, so there is some redundancy in rulestrings. For example, the rule `B3/S2ac3` is exactly the same as the rule `B3/S2-eikn3`: they both mean that a cell will be born and survive if it has 3 live neighbors (regardless of orientation), and a cell will survive if it has 2 live neighbors in one of the "`a`" or "`c`" configurations, but not in one of the "`e`", "`i`", "`k`", or "`n`" configurations. To remove some of this ambiguity, we typically prefer shorter rulestrings over longer ones.

These modifiers can be applied independently after each integer in the rulestring.[6] For example, the rulestring `B3-j6i/S23-c4i` indicates that:

- A dead cell will be born if it has 3 live neighbors, unless they are in the "`j`" configuration.
- A dead cell will be born if it has 6 live neighbors, but only if they are in the "`i`" configuration.
- A live cell will survive if it has 2 live neighbors (in any configuration).
- A live cell will survive if it has 3 live neighbors, unless they are in the "`c`" configuration.
- A live cell will survive if it has 4 live neighbors, but only if they are in the "`i`" configuration.

[5] The rulestring notation that we describe here is sometimes called **Hensel notation**, after its creator Alan Hensel.

[6] However, some modifiers cannot appear after some integers. For example, the modifier "`q`" cannot come after a 2 since there is no corresponding entry in Table B.3.

Free PDF. See conwaylife.com/book for pattern files and print version.

modifier \ neighbors	1	2	3	4	5	6	7
c (**c**orner)							
e (**e**dge)							
a (**a**djacent)							
k (**k**night)							
i							
n							
j							
q							
r							
y							
t							
w							
z							

Table B.3: A list of all possible configurations of the Moore neighborhood of a single cell, up to rotation and reflection, except for the all-dead and all-alive configurations. The letter on the left of each row specifies the modifier abbreviation that is associated with each of the configurations in that row. For example, a rulestring containing `B3k` indicates that a cell with 3 neighbors as in the "`k`" row and "3" column will be born, and a rulestring containing `S4k` indicates survival of a cell with 4 neighbors as in the "`k`" row and "4" column.

Free PDF. See conwaylife.com/book for pattern files and print version.

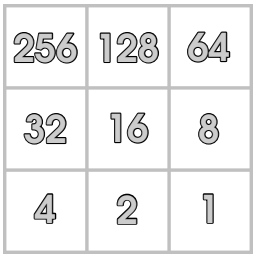
Indeed, the rule `B3-j6i/S23-c4i` is quite similar to Life itself, but with four isotropic modifications, as we saw in Figure 12.8.

Similarly, the rulestring `B3acijn4jktwyz5ijr6-en7c/S2aen3-aceq4acijqty5cikr6ak7c` that we used in Figure 12.10 means that a dead cell will be born if it has 3 live neighbors in one of the "a", "c", "i", "j", or "n" configurations, or if it has 4 live neighbors in one of the "j", "k", "t", "w", "y", or "z" configurations, and so on.

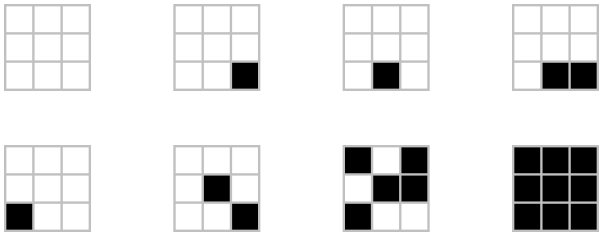
B.7 Non-Isotropic Rulestrings

We now describe how to read and construct rulestrings that specify a not necessarily isotropic 2-state cellular automaton that uses the Moore neighborhood on a 2D square grid. Because there are so many more of these rules than isotropic ones (2^{512} versus 2^{102}), the length of their rulestrings becomes something of a concern, so we use a different format than their outer-totalistic and isotropic counterparts.[7]

The basic idea behind these non-isotropic rulestrings is to simply list what happens to the central cell in each of the $2^9 = 512$ different possible configurations of cells in a 3×3 grid. In order to make that list, though, we need to agree on a fixed ordering of those 512 configurations. To this end, we use the diagram given in Figure B.4, which establishes what the "first" of those 512 configurations is, what the "second" one is, and so on.



(a) The index of a configuration is the sum of the numbers on its live cells.



(b) From left to right, these are configurations (first row) 0, 1, 2, $2+1=3$, (second row) 4, $16+1=17$, $256+64+16+8+4=348$, and $256+128+64+32+16+8+4+2+1=511$.

Figure B.4: To order the $2^9 = 512$ possible configurations of 3×3 live cells, we assign each one an index that is the sum of the numbers from (a) that are on live cells. The "0th" configuration is thus the one with all cells dead, the "1st" configuration is the one with only the bottom-right cell alive, and so on.

When ordering these configurations, we start counting as 0 so that the all-dead configuration is the "0th" one (which is listed first), the configuration with only the bottom-right cell alive is the "1st" one (which is listed second), and so on. To create a rulestring for a non-isotropic CA, we now do two things:

1) We create a 512-bit string of 0s and 1s that specifies which configurations (in the ordering specified by Figure B.4) lead to the central cell being dead or alive, respectively, in the next generation.
   For example, if cells are only born if they have no neighbors or a single neighbor to their southeast, and they only survive if they have neighbors everywhere *except* for to their north and northwest,[8] then there are only three configurations of 3×3 grids of cells that lead to the central cell being alive in the next generation, and they correspond to indices 0, 1, and $64+32+16+8+4+2+1 = 127$. The 512-bit string describing this rule thus has 1s in its

[7] This non-isotropic rulestring format was created by Chris Rowett.

[8] This is the cellular automaton that was used by the knightship of Figure 12.11(b).

Free PDF. See conwaylife.com/book for pattern files and print version.

0th, 1st, and 127th locations, and 0s elsewhere:

```
11000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000.
```

Base 2	Base 64	Base 2	Base 64	Base 2	Base 64	Base 2	Base 64
000000	A	010000	Q	100000	g	110000	w
000001	B	010001	R	100001	h	110001	x
000010	C	010010	S	100010	i	110010	y
000011	D	010011	T	100011	j	110011	z
000100	E	010100	U	100100	k	110100	0
000101	F	010101	V	100101	l	110101	1
000110	G	010110	W	100110	m	110110	2
000111	H	010111	X	100111	n	110111	3
001000	I	011000	Y	101000	o	111000	4
001001	J	011001	Z	101001	p	111001	5
001010	K	011010	a	101010	q	111010	6
001011	L	011011	b	101011	r	111011	7
001100	M	011100	c	101100	s	111100	8
001101	N	011101	d	101101	t	111101	9
001110	O	011110	e	101110	u	111110	+
001111	P	011111	f	101111	v	111111	/

Table B.4: A way of converting a (base-2) bitstring into a base-64 string. Every substring consisting of six bits gets mapped to a single alphanumeric character, or "+" or "/".

2) In order to make the rule's description a bit shorter, we convert this 512-bit string from base 2 to base 64 according to Table B.4.[9] Since each character in the base 64 string encodes $\log_2(64) = 6$ bits of the bitstring, this process gives a $\lceil 512/6 \rceil = 86$-character string that uniquely specifies the rule.[10] Finally, we prepend this base-64 string with the characters "MAP" just to indicate that it is a non-isotropic rulestring.
   For example, if we convert the bitstring that we saw in part (1) to base 64, the first 6 bits ("110000") become "w", several groups of 6 bits ("000000") each become "A", and one of the

[9] This is a fairly standard way of converting a bitstring to a mostly alphanumeric string that shows up in lots of contexts outside of cellular automata as well.

[10] Since $512 \equiv 2 \pmod 6$, the final two bits of the bitstring correspond to the final character in the base-64 rulestring (and that character is determined by just the first two bits from Table B.4). This leads to some redundancy in the base-64 rulestrings: a final character of "A" indicates the same rule as a final character of "P", for example, and any of the final characters "g" through "v" are equivalent to each other.

Free PDF. See conwaylife.com/book for pattern files and print version.

central groups of 6 bits ("010000") becomes "Q". Altogether, the rulestring that describes this cellular automaton is

```
MAPwAAAAAAAAAAAAAAAAAAAQAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA.
```

MAP rules can also be specified for hexagonal and von Neumann neighborhoods. Hexagonal neighborhoods have 7 cells (the center plus 6 neighbors), which gives $2^7 = 128$ possible combinations of live and dead cells that can be encoded by 22 base-64 characters. Von Neumann neighborhoods have 5 cells (the center plus 4 neighbors), which gives $2^5 = 32$ possible combinations of live and dead cells that can be encoded by 6 base-64 characters.

B.8 π Calculator APGsembly Code

In Section 9.6, we developed an algorithm and pseudocode for computing the decimal digits of $\pi = 3.14159\ldots$. We now present the full APGsembly code that implements that algorithm and pseudocode, and was used to compile the π calculator that we saw in Figure 9.15. In particular, APGsembly B.1 and B.2 implements that pseudocode via sliding block (unary) registers `U0`, `U1`, …, `U9` and binary registers `B0`, `B1`, `B2`, `B3` that store the following quantities (the A_n and B_n matrices and quantities q_n and r_n are as defined in Section 9.6):

- `U0`: the top-left corner of the A_n matrix
- `U1`: the top-right and bottom-right corners of the A_n matrix
- `U2`: the current digit being computed
- `U3`: the index of the current digit (`U3` $= 0$ for "3", `U3` $= 1$ for "1", `U3` $= 2$ for "4", and so on)
- `U4`: the number of times that the iteration has run, typically denoted here by n
- `U5`: counter that forces the program to iterate 4 times before printing a digit
- `U6`: the number of bits of memory allocated to the binary registers
- `U7`–`U9`: temporary registers used to help perform arithmetic operations and miscellaneous loops

- `B0`: the top-left corner of the B_n matrix
- `B1`: the top-right corner of the B_n matrix ($= q_n$)
- `B2`: the bottom-right corner of the B_n matrix ($= r_n$)
- `B3`: temporary register used to help perform arithmetic operations

Recall that lines listed with a substate (input value) * just do the same actions and jump operation regardless of whether they receive a `Z` or `NZ` input value.

Free PDF. See conwaylife.com/book for pattern files and print version.

APGsembly B.1 Page 1 of APGsembly code for a π calculator that implements Pseudocode 9.3.

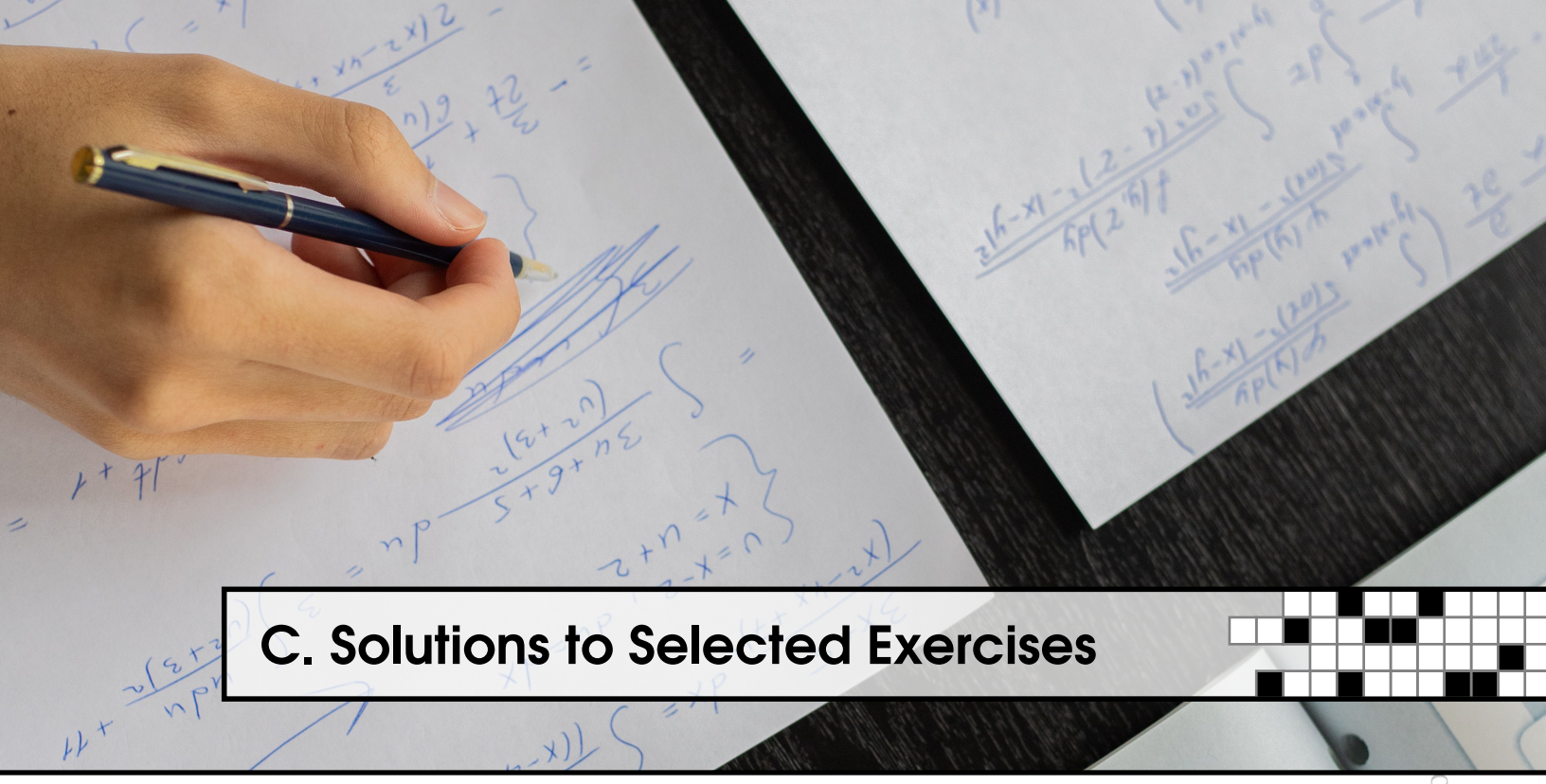
```
#COMPONENTS NOP,OUTPUT,B0-3,U0-9,ADD,SUB,MUL
#REGISTERS {'U1':1, 'U6':6, 'B0':[0,'01'] , 'B2':[0,'1']}
# State    Input    Next state    Actions
# ---------------------------------------
INITIAL;   ZZ;      ITER1;        NOP

# Iterate 4 times per digit.
ITER1;     ZZ;      ITER2;        INC U5, NOP
ITER2;     ZZ;      ITER3;        INC U5, NOP
ITER3;     ZZ;      ITER4;        INC U5, NOP
ITER4;     ZZ;      ITER5;        INC U5, NOP
ITER5;     ZZ;      ITER6;        TDEC U5

# Each iteration, set U0 = U0 + 1, U1 = U1 + 2.
ITER6;     Z;       ITER11;       TDEC U3
ITER6;     NZ;      ITER7;        INC U0, INC U1, NOP
ITER7;     ZZ;      MULA1;        INC U1, NOP

## The MULA states set B3 = B1, B1 = U1 * B1.
# Copy B1 into B3, without erasing B1.
MULA1;     ZZ;      MULA2;        TDEC U6
MULA2;     Z;       MULA3;        TDEC U9
MULA2;     NZ;      MULA2;        TDEC U6, INC U9
MULA3;     Z;       MULA4;        TDEC U7
MULA3;     NZ;      MULA3;        TDEC U9, INC U6, INC U7
MULA4;     Z;       MULA8;        TDEC B1
MULA4;     NZ;      MULA5;        READ B1
MULA5;     Z;       MULA7;        READ B3
MULA5;     NZ;      MULA6;        SET B1, READ B3
MULA6;     *;       MULA7;        SET B3, NOP
MULA7;     *;       MULA4;        INC B1, INC B3, TDEC U7
MULA8;     Z;       MULA9;        TDEC B3
MULA8;     NZ;      MULA8;        TDEC B1
MULA9;     Z;       MULA10;       TDEC U1
MULA9;     NZ;      MULA9;        TDEC B3

# Copy U1 to temporary register U8.
MULA10;    Z;       MULA11;       TDEC U7
MULA10;    NZ;      MULA10;       TDEC U1, INC U7
MULA11;    Z;       MULA12;       TDEC U8
MULA11;    NZ;      MULA11;       TDEC U7, INC U1, INC U8

# Set B1 = U1 * B3 and U8 = 0.
MULA12;    *;       MULA13;       TDEC U8
MULA13;    Z;       MULB1;        TDEC U6
MULA13;    NZ;      MULA14;       TDEC U6
MULA14;    Z;       MULA15;       TDEC U9
MULA14;    NZ;      MULA14;       TDEC U6, INC U9
MULA15;    Z;       MULA16;       TDEC U7
MULA15;    NZ;      MULA15;       TDEC U9, INC U6, INC U7
MULA16;    Z;       MULA20;       TDEC B3
MULA16;    NZ;      MULA17;       READ B3
MULA17;    Z;       MULA18;       READ B1
MULA17;    NZ;      MULA18;       READ B1, SET B3, ADD A1
MULA18;    Z;       MULA19;       ADD B0
MULA18;    NZ;      MULA19;       ADD B1
MULA19;    Z;       MULA16;       TDEC U7, INC B1, INC B3
MULA19;    NZ;      MULA19;       SET B1, NOP
MULA20;    Z;       MULA21;       TDEC B1
MULA20;    NZ;      MULA20;       TDEC B3
MULA21;    Z;       MULA13;       TDEC U8
MULA21;    NZ;      MULA21;       TDEC B1

## The MULB states set B3 = B0, B0 = U0 * B0.
# Copy B0 into B3, without erasing B0.
MULB1;     Z;       MULB2;        TDEC U9
MULB1;     NZ;      MULB1;        TDEC U6, INC U9
MULB2;     Z;       MULB3;        TDEC U7
MULB2;     NZ;      MULB2;        TDEC U9, INC U6, INC U7
MULB3;     Z;       MULB6;        TDEC B0
MULB3;     NZ;      MULB4;        READ B3
MULB4;     *;       MULB5;        READ B0
MULB5;     Z;       MULB3;        INC B0, INC B3, TDEC U7
MULB5;     NZ;      MULB5;        SET B0, SET B3, NOP
MULB6;     Z;       MULB7;        TDEC B3
MULB6;     NZ;      MULB6;        TDEC B0
MULB7;     Z;       MULB8;        TDEC U0
MULB7;     NZ;      MULB7;        TDEC B3

# Copy U0 to temporary register U8.
MULB8;     Z;       MULB9;        TDEC U7
MULB8;     NZ;      MULB8;        TDEC U0, INC U7
MULB9;     Z;       MULB10;       TDEC U8
MULB9;     NZ;      MULB9;        TDEC U7, INC U0, INC U8

# Set B0 = U0 * B3 and U8 = 0.
MULB10;    *;       MULB11;       TDEC U8
MULB11;    Z;       MULC1;        TDEC U1
MULB11;    NZ;      MULB12;       TDEC U6
MULB12;    Z;       MULB13;       TDEC U9
MULB12;    NZ;      MULB12;       TDEC U6, INC U9
MULB13;    Z;       MULB14;       TDEC U7
MULB13;    NZ;      MULB13;       TDEC U9, INC U6, INC U7
MULB14;    Z;       MULB18;       TDEC B3
MULB14;    NZ;      MULB15;       READ B3
MULB15;    Z;       MULB16;       READ B0
MULB15;    NZ;      MULB16;       READ B0, SET B3, ADD A1
MULB16;    Z;       MULB17;       ADD B0
MULB16;    NZ;      MULB17;       ADD B1
MULB17;    Z;       MULB14;       TDEC U7, INC B0, INC B3
MULB17;    NZ;      MULB17;       SET B0, NOP
MULB18;    Z;       MULB19;       TDEC B0
MULB18;    NZ;      MULB18;       TDEC B3
MULB19;    Z;       MULB11;       TDEC U8
MULB19;    NZ;      MULB19;       TDEC B0

## The MULC states set B1 = B1 + (U1 * B0).
# Copy U1 to temporary register U8.
MULC1;     Z;       MULC2;        TDEC U7
MULC1;     NZ;      MULC1;        TDEC U1, INC U7
MULC2;     Z;       MULC3;        TDEC U8
MULC2;     NZ;      MULC2;        TDEC U7, INC U1, INC U8

# Set B1 = B1 + (U1 * B3) and U8 = 0.
MULC3;     Z;       MULD1;        TDEC U6
MULC3;     NZ;      MULC4;        TDEC U6
MULC4;     Z;       MULC5;        TDEC U9
MULC4;     NZ;      MULC4;        TDEC U6, INC U9
MULC5;     Z;       MULC6;        TDEC U7
MULC5;     NZ;      MULC5;        TDEC U9, INC U6, INC U7
MULC6;     Z;       MULC10;       TDEC B3
MULC6;     NZ;      MULC7;        READ B3
MULC7;     Z;       MULC8;        READ B1
MULC7;     NZ;      MULC8;        READ B1, SET B3, ADD A1
MULC8;     Z;       MULC9;        ADD B0
MULC8;     NZ;      MULC9;        ADD B1
MULC9;     Z;       MULC6;        TDEC U7, INC B1, INC B3
MULC9;     NZ;      MULC9;        SET B1, NOP
MULC10;    Z;       MULC11;       TDEC B1
MULC10;    NZ;      MULC10;       TDEC B3
MULC11;    Z;       MULC3;        TDEC U8
MULC11;    NZ;      MULC11;       TDEC B1

## The MULD states set B2 = U1 * B2.
# Copy B2 into B3, without erasing B2.
MULD1;     Z;       MULD2;        TDEC U9
MULD1;     NZ;      MULD1;        TDEC U6, INC U9
MULD2;     Z;       MULD3;        TDEC U7
MULD2;     NZ;      MULD2;        TDEC U9, INC U6, INC U7
MULD3;     Z;       MULD6;        TDEC B2
MULD3;     NZ;      MULD4;        READ B3
MULD4;     *;       MULD5;        READ B2
MULD5;     Z;       MULD3;        INC B2, INC B3, TDEC U7
MULD5;     NZ;      MULD5;        SET B2, SET B3, NOP
MULD6;     Z;       MULD7;        TDEC B3
MULD6;     NZ;      MULD6;        TDEC B2
MULD7;     Z;       MULD8;        TDEC U1
MULD7;     NZ;      MULD7;        TDEC B3

# Copy U1 to temporary register U8.
MULD8;     Z;       MULD9;        TDEC U7
MULD8;     NZ;      MULD8;        TDEC U1, INC U7
MULD9;     Z;       MULD10;       TDEC U8
MULD9;     NZ;      MULD9;        TDEC U7, INC U1, INC U8

# Set B2 = U1 * B3 and U8 = 0.
MULD10;    *;       MULD11;       TDEC U8
MULD11;    Z;       ITER8;        INC U4, NOP
MULD11;    NZ;      MULD12;       TDEC U6
MULD12;    Z;       MULD13;       TDEC U9
MULD12;    NZ;      MULD12;       TDEC U6, INC U9
MULD13;    Z;       MULD14;       TDEC U7
MULD13;    NZ;      MULD13;       TDEC U9, INC U6, INC U7
MULD14;    Z;       MULD18;       TDEC B3
MULD14;    NZ;      MULD15;       READ B3
MULD15;    Z;       MULD16;       READ B2
MULD15;    NZ;      MULD16;       READ B2, SET B3, ADD A1
MULD16;    Z;       MULD17;       ADD B0
MULD16;    NZ;      MULD17;       ADD B1
MULD17;    Z;       MULD14;       INC B2, INC B3, TDEC U7
MULD17;    NZ;      MULD17;       SET B2, NOP
MULD18;    Z;       MULD19;       TDEC B2
MULD18;    NZ;      MULD18;       TDEC B3
MULD19;    Z;       MULD11;       TDEC U8
MULD19;    NZ;      MULD19;       TDEC B2
```

Free PDF. See conwaylife.com/book for pattern files and print version.

APGsembly B.2 Page 2 of APGsembly code for a π calculator that implements Pseudocode 9.3.

```
# Increase the amount of memory that we are allocating to the
# binary registers, by adding U4 to U6.
ITER8;      ZZ;       ITER9;        TDEC U4
ITER9;      Z;        ITER10;       TDEC U7
ITER9;      NZ;       ITER9;        TDEC U4, INC U7
ITER10;     Z;        ITER6;        TDEC U5
ITER10;     NZ;       ITER10;       TDEC U7, INC U4, INC U6

## Extract the units digit from (10^U3) * B1 / B2, as that is
## the digit of pi that we want to print.
# Copy U3 to temporary register U8.
ITER11;     Z;        ITER12;       TDEC U7
ITER11;     NZ;       ITER11;       TDEC U3, INC U7
ITER12;     Z;        ITER13;       TDEC U6
ITER12;     NZ;       ITER12;       TDEC U7, INC U3, INC U8

# Copy B1 into B3, without erasing B1.
ITER13;     Z;        ITER14;       TDEC U7
ITER13;     NZ;       ITER13;       TDEC U6, INC U7
ITER14;     Z;        ITER15;       TDEC U9
ITER14;     NZ;       ITER14;       TDEC U7, INC U6, INC U9
ITER15;     Z;        ITER18;       TDEC B3
ITER15;     NZ;       ITER16;       READ B3
ITER16;     *;        ITER17;       READ B1
ITER17;     Z;        ITER15;       INC B1, INC B3, TDEC U9
ITER17;     NZ;       ITER17;       SET B1, SET B3, NOP
ITER18;     Z;        ITER19;       TDEC B1
ITER18;     NZ;       ITER18;       TDEC B3
ITER19;     Z;        CMP1;         TDEC U6
ITER19;     NZ;       ITER19;       TDEC B1

# Now compare B2 with B3 to see which is bigger. This
# determines which of the two upcoming code blocks to go to.
CMP1;       Z;        CMP2;         TDEC U7
CMP1;       NZ;       CMP1;         TDEC U6, INC U7
CMP2;       Z;        CMP3;         TDEC U9
CMP2;       NZ;       CMP2;         TDEC U7, INC U6, INC U9
CMP3;       Z;        CMP4;         READ B3
CMP3;       NZ;       CMP3;         TDEC U9, INC B2, INC B3
CMP4;       Z;        CMP5;         READ B2
CMP4;       NZ;       CMP8;         READ B2, SET B3
CMP5;       Z;        CMP6;         TDEC B2
CMP5;       NZ;       CMP10;        TDEC B3, SET B2
CMP6;       *;        CMP7;         TDEC B3
CMP7;       Z;        CMP13;        TDEC B2
CMP7;       NZ;       CMP4;         READ B3
CMP8;       Z;        CMP12;        TDEC B3
CMP8;       NZ;       CMP9;         SET B2, NOP
CMP9;       ZZ;       CMP6;         TDEC B2
CMP10;      Z;        CMP11;        TDEC B2
CMP10;      NZ;       CMP10;        TDEC B3
CMP11;      Z;        DIG1;         TDEC U8
CMP11;      NZ;       CMP11;        TDEC B2
CMP12;      Z;        CMP13;        TDEC B2
CMP12;      NZ;       CMP12;        TDEC B3
CMP13;      Z;        SUB1;         TDEC U6
CMP13;      NZ;       CMP13;        TDEC B2

# If B2 <= B3 then subtract B2 from B3.
# That is, start or carry on with the integer division B3 / B2.
SUB1;       Z;        SUB2;         TDEC U7
SUB1;       NZ;       SUB1;         TDEC U6, INC U7
SUB2;       Z;        SUB3;         TDEC U9
SUB2;       NZ;       SUB2;         TDEC U7, INC U6, INC U9
SUB3;       Z;        SUB7;         TDEC B3
SUB3;       NZ;       SUB4;         READ B3
SUB4;       Z;        SUB5;         READ B2
SUB4;       NZ;       SUB5;         READ B2, SUB A1
SUB5;       Z;        SUB6;         SUB B0
SUB5;       NZ;       SUB6;         SUB B1, SET B2
SUB6;       Z;        SUB3;         INC B2, INC B3, TDEC U9
SUB6;       NZ;       SUB6;         SET B3, NOP
SUB7;       Z;        SUB8;         TDEC B2
SUB7;       NZ;       SUB7;         TDEC B3
SUB8;       Z;        CMP1;         TDEC U6, INC U2
SUB8;       NZ;       SUB8;         TDEC B2

# If B2 > B3 we cannot subtract anymore.
# Multiply B3 by 10 and reset U2, or jump ahead and print
# the digit that we have now computed.
DIG1;       Z;        OUT0;         TDEC U2
DIG1;       NZ;       DIG2;         TDEC U2
DIG2;       Z;        DIG3;         TDEC U6
DIG2;       NZ;       DIG2;         TDEC U2
DIG3;       Z;        DIG4;         TDEC U7
DIG3;       NZ;       DIG3;         TDEC U6, INC U7
DIG4;       Z;        DIG5;         TDEC U9
DIG4;       NZ;       DIG4;         TDEC U7, INC U6, INC U9
DIG5;       Z;        DIG8;         TDEC B3
DIG5;       NZ;       DIG6;         READ B3
DIG6;       Z;        DIG7;         MUL 0
DIG6;       NZ;       DIG7;         MUL 1
DIG7;       Z;        DIG5;         INC B3, TDEC U9
DIG7;       NZ;       DIG7;         SET B3, NOP
DIG8;       Z;        CMP1;         TDEC U6
DIG8;       NZ;       DIG8;         TDEC B3

# Print the current digit, which is stored in U2.
OUT0;       Z;        OUTD1;        NOP, OUTPUT 0
OUT0;       NZ;       OUT1;         TDEC U2
OUT1;       Z;        OUTD1;        NOP, OUTPUT 1
OUT1;       NZ;       OUT2;         TDEC U2
OUT2;       Z;        OUTD1;        NOP, OUTPUT 2
OUT2;       NZ;       OUT3;         TDEC U2
OUT3;       Z;        OUTD1;        NOP, OUTPUT 3
OUT3;       NZ;       OUT4;         TDEC U2
OUT4;       Z;        OUTD1;        NOP, OUTPUT 4
OUT4;       NZ;       OUT5;         TDEC U2
OUT5;       Z;        OUTD1;        NOP, OUTPUT 5
OUT5;       NZ;       OUT6;         TDEC U2
OUT6;       Z;        OUTD1;        NOP, OUTPUT 6
OUT6;       NZ;       OUT7;         TDEC U2
OUT7;       Z;        OUTD1;        NOP, OUTPUT 7
OUT7;       NZ;       OUT8;         TDEC U2
OUT8;       Z;        OUTD1;        NOP, OUTPUT 8
OUT8;       NZ;       OUTD1;        NOP, OUTPUT 9

# Check whether or not we just printed the very first digit (3).
# If so, print a decimal point. Either way, increase U3, which
# counts which decimal place we are currently at, and loop back
# to start the next digit calculation.
OUTD1;      ZZ;       OUTD2;        TDEC U3
OUTD2;      Z;        ITER1;        INC U3, NOP, OUTPUT .
OUTD2;      NZ;       OUTD3;        INC U3, NOP
OUTD3;      ZZ;       ITER1;        INC U3, NOP
```

Free PDF. See conwaylife.com/book for pattern files and print version.



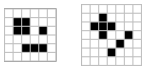
C. Solutions to Selected Exercises

Chapter 1: Early Life

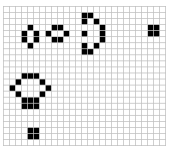
1.1. **(a)** A twin bees shuttle
(b) A block-laying switch engine.
(c) A period 14 oscillator (called the **tumbler**).
(d) A glider-producing switch engine.

1.5. **(a)** Generation 41. **(b)** Generation 42.
(c) Generation 240. **(d)** Generation 776.
(e) Generation 97.

1.7. **(a)** A period 12 spaceship that has two lightweight spaceships at the front. This object is called the **Schick engine**, and we will investigate it in Section 4.4.2.
(b) A period 10 spaceship (called the **copperhead**).
(c) A period 51 oscillator (found in 2009 by Nicolay Beluchenko).
(d) A period 37 oscillator (found in 2009 by Nicolay Beluchenko).

1.8. **(a)**  **(b)** 

1.9. **(a)** They destroy each other.
(b) One possibility is displayed here:



1.10. **(a)** The eater 1 destroys the pre-beehive, leaving only the eater 1 behind.
(b) The pre-beehive evolves into a beehive after 1 generation. Nothing happens to the eater 1 (it is a still life).
(c) The queen bee leaves a pre-beehive behind in the generation right before the beehive itself appears (i.e., 14 generations after the first phase displayed in Figure 1.15).
(d) One possibility is displayed in Figure 3.16(b).

1.16. **(a)** $n = \left\lceil \dfrac{2}{6 - \sqrt{\log_2(2^{36} - 1)}} \right\rceil = 1\,143\,185\,077\,171.$

(b) The new inequality to check is $(2^{36} - 4)^{n^2} < 2^{(6n-2)^2}$, which is true whenever

$$n \geq \left\lceil \frac{2}{6 - \sqrt{\log_2(2^{36} - 4)}} \right\rceil = 285\,796\,269\,287.$$

(c) The new inequality to check is $(2^{25} - 1)^{n^2} < 2^{(5n-2)^2}$, which is true whenever

$$n \geq \left\lceil \frac{2}{5 - \sqrt{\log_2(2^{25} - 1)}} \right\rceil = 465\,163\,192.$$

(d) The problem is that neighboring tiles might interact with each other in different ways even though individually they evolve the same. In particular, if the tile directly to the left of the two tiles displayed in this question has live cells on its rightmost edge, then that tile can interact in different ways with the block and the pre-block. We thus need *two* rows

Free PDF. See conwaylife.com/book for pattern files and print versio

of "buffer" dead cells around each cell that differs between the tiles.

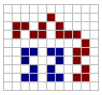
1.17. (a) This pattern creates *two* glider-producing switch engines that fire backward toward the initial debris, rather than just one.

(b) A forward-moving glider hits the switch engines at the front, transforming one of them into a block-laying switch engine and destroying the other, so no more gliders are fired backward toward the initial debris.

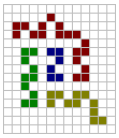
Chapter 2: Still Lifes

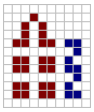
2.1 (a) Strict still life.
(b) Strict still life.
(c) Pseudo still life.
(d) Pseudo still life.
(e) Neither.
(f) Strict still life.

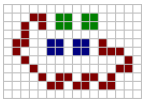
2.3 (a)


(b)


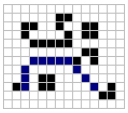
(c)

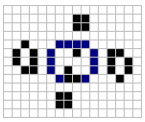

(d)


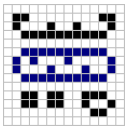
(e)


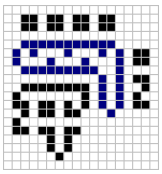
(f)


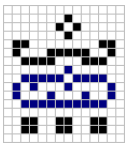
2.4 (a)

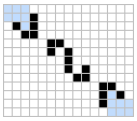

(b)


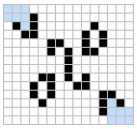
(c)


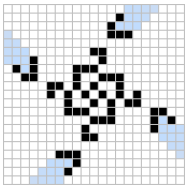
2.7 (a)


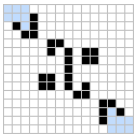
(b)


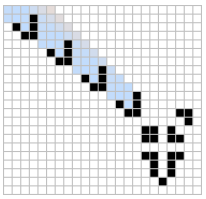
(c)


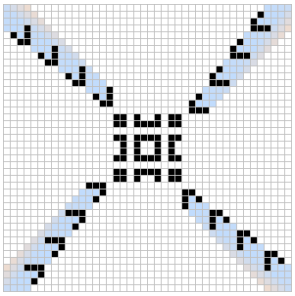
2.9 (a)


(b)


(c)


(d)


(e)


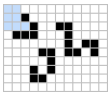
(f)


Free PDF. See conwaylife.com/book for pattern files and print version.

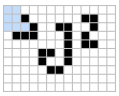
2.13 **(a)**



(b)



2.15 One possibility is as follows:



2.16 If a dead cell has 7 or 8 live neighbors, then at most one of its neighbors is dead, so it is possible to find (up to rotation) a 5-cell "u"-shaped region of neighbors that are all alive. The middle-bottom cell in this "u"-shaped region has at least 4 live neighbors and thus cannot actually be part of a still life, which is a contradiction. The ship is an example of a still life for which a dead cell (the central dead cell) has 6 live neighbors.

2.19. **(a)** 6

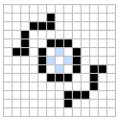
(b) 5

(c) If L is the number of live cells in an $n \times n$ square, we find that $11L \leq 6n^2 + 24n + 24$, so $L \leq (6/11)n^2 + (24/11)n + (24/11)$ and hence the asymptotic density of a still life cannot exceed $6/11$.

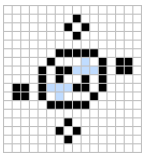
Chapter 3: Oscillators

3.1 These oscillators are (a) **scrubber**, found no later than 1971, (b) **airforce**, found by David Buckingham in 1972, and (c) **pinwheel**, found by Simon Norton in 1970:

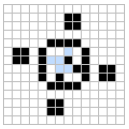
(a)



(b)

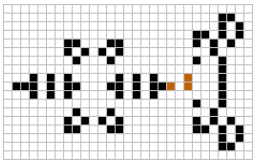


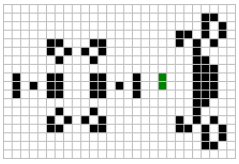
(c)



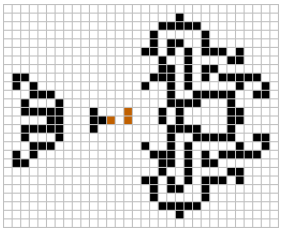
3.5 In these solutions, we use orange cells to denote accessible sparks and green cells to denote cells that oscillate at the full period in our new compound oscillators.

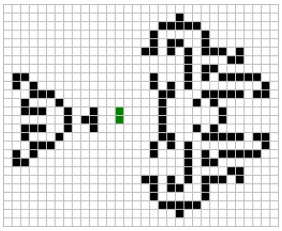
(a) We could either combine the T-nosed p4 with a period 3 sparker or the T-nosed p6 with a period 4 sparker. Here is a T-nosed p6 with a p4 heavyweight emulator:



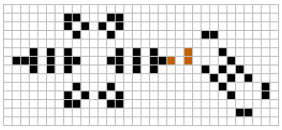


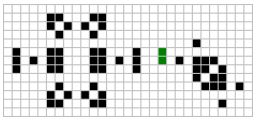
(b) We combine a T-nosed p4 with a p5 heavyweight volcano (a fumarole does not quite work since its spark is a bit too close to the body of the oscillator):



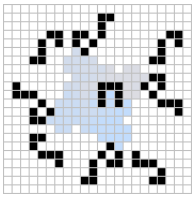


(c) We combine a T-nosed p6 with a p8 figure eight:





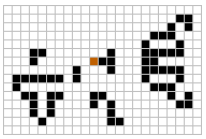
3.7 Each eater 1 can be oriented in one of two different ways:



Free PDF. See conwaylife.com/book for pattern files and print version.

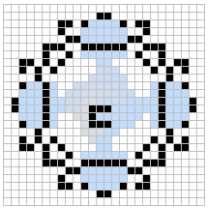
3.9 **(a)** The period is 7.

(b) Using the T-nosed p4 gives the following period 8 oscillator:



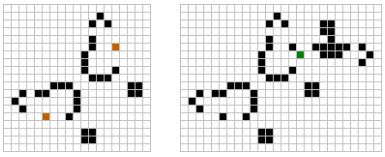
(c) The period can be $7n+1$ for any integer $n \geq 1$.

3.11 One possibility is the following oscillator, called the **pi portraitor**, which was found by Robert Wainwright in 1984 or 1985. It uses four heavyweight emulators that have been welded together to provide the domino sparks.

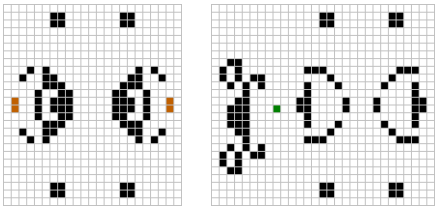


3.12 In these solutions, we use orange cells to denote accessible sparks and green cells to denote cells that oscillate at the full period in our new compound oscillators.

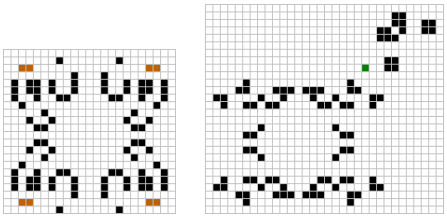
(a) We can combine this period 13 oscillator with a period 3 caterer to create an oscillator with period $\mathrm{lcm}(3,13) = 39$:



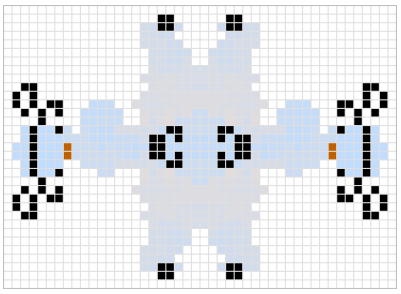
(b) We can combine this period 31 oscillator with a period 4 middleweight emulator to create an oscillator with period $\mathrm{lcm}(4,31) = 124$:



(c) We can combine this period 32 oscillator with a period 6 unix to create an oscillator with period $\mathrm{lcm}(6,32) = 96$:



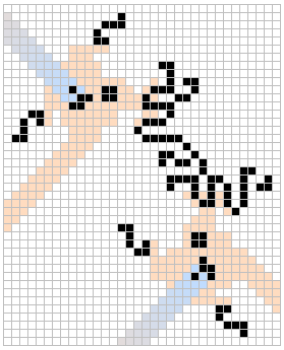
3.15 It has period 44, so we use p4 heavyweight emulators. Originally found by David Buckingham in April 1992, but reduced to the form shown here by Noam Elkies in April 1996:



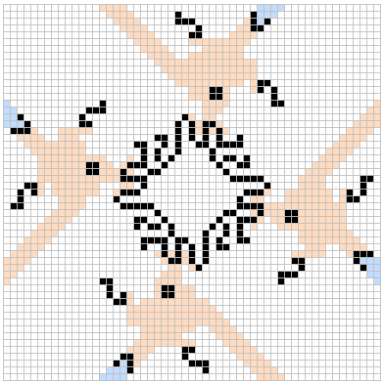
3.18 One possibility is the following period 680 oscillator. Note that by adding extra gliders to this track, we can trivially create oscillators with many other periods as well.



3.19 **(a)**



(b)

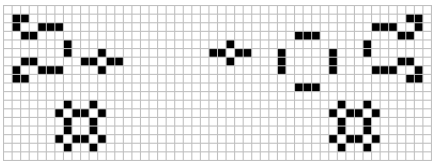


3.20 A pre-honey farm.

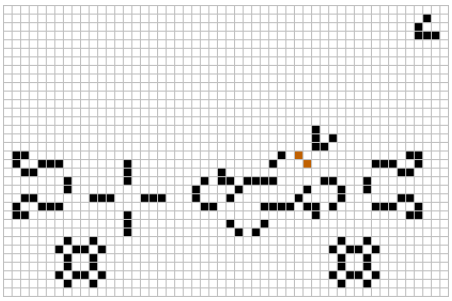
Free PDF. See conwaylife.com/book for pattern files and print version.

3.21 Unfortunately, the block that the glider hits is moved up by 1 cell, so any subsequent gliders will not trigger the same reaction.

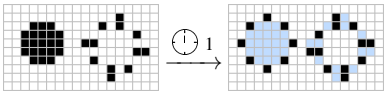
3.25 One possibility is the following oscillator, which was found by Noam Elkies in 1994. It is currently the smallest known period 50 oscillator.



3.26 The traffic jam creates dot and duoplet sparks. One way of using the period 50 oscillator from the solution to Exercise 3.25 to reflect gliders is displayed below:



3.30 One phoenix that works is as follows, where the object on the left dies completely in 2 generations:

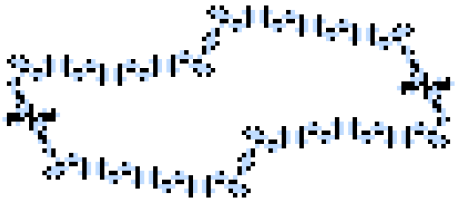


3.33 Suppose that there is a single cell (which we will call X) that oscillates with period 4. There are two possibilities: either X has the same state (alive or dead) in 3 consecutive phases and the other state in the fourth phase, or X has the same state for two consecutive phases and then the opposite state for the other two.

To see that the first possibility cannot happen, suppose without loss of generality that X has the same state in generations 0, 1, 2, and the opposite state in generation 3. Since all other cells are either stable or oscillate with period 2, all of X's neighbors have the same state in generations 0 and 2. Since X itself also has the same state in generations 0 and 2, it must also have the same state in generations 1 and 3, which is a contradiction that shows it cannot oscillate at period 4.

To see that the second possibility cannot happen, suppose without loss of generality that X is alive in generations 0 and 1, and dead in generations 2 and 3. Since all other cells are either stable or oscillate with period 2, all of X's neighbors have the same state in generations 1 and 3. Since X is dead and born after generation 3, it must have exactly 3 live neighbors at generation 3, and thus at generation 1 as well. This contradicts the fact that X is alive at generation 1 and then dies.

3.35 One possibility is as follows:

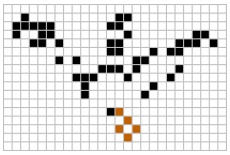


Chapter 4: Spaceships and Moving Objects

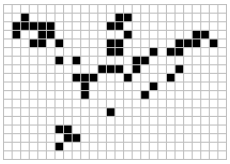
4.1 **(a)** Opposite.
(b) Same.
(c) Same.

4.2 **(a)** Color-preserving.
(b) Color-preserving.
(c) Color-preserving.
(d) Color-preserving.
(e) Color-changing.

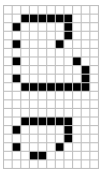
4.6 **(a)**



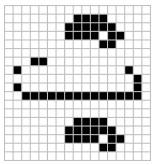
(b)



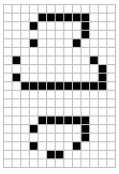
4.9. **(a)**



(b)

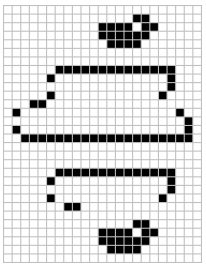


(c)

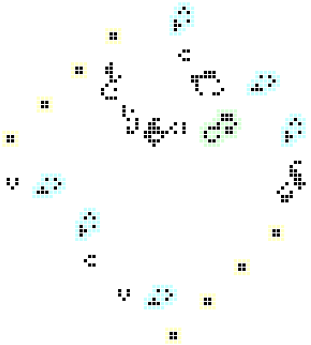


Free PDF. See conwaylife.com/book for pattern files and print version.

4.10

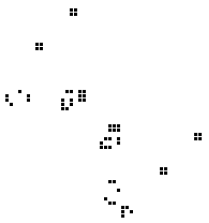


4.15 One possibility is the 7-engine Cordership displayed here, which was originally found by Dean Hickerson in 1993:

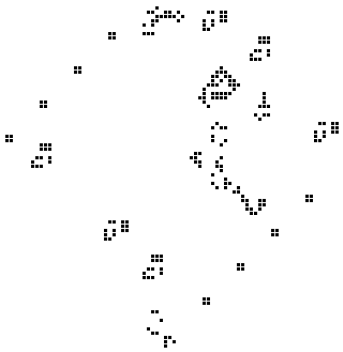


4.16 **(a)** In this reaction, the switch engines are 1 cell diagonally father back relative to the blocks than in the original reaction.

(b) This reaction emits a banana spark, which can be used to reflect a glider as follows:

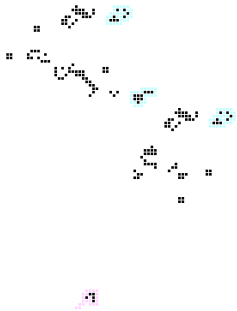


(c) We can just replace the back of the 7-engine Cordership from Exercise 4.15 (or we could have replaced the back to the 10-engine Cordership):

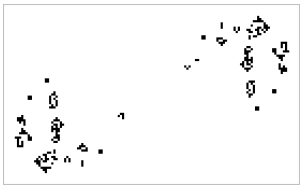


4.18 **(a)** It becomes two non-interacting block-laying switch engines.

(b) One method that works is to notice that this Cordership leaves behind a banana spark, which can be used to rotate the glider as follows:



4.20 **(a)**



(b) The reflection used in part (a) preserves the mod-4 timing of the reflected glider, whereas the other two reflections do not. Thus the reflected glider is not in the correct phase after being reflected, so it cannot bounce more than twice (once off of each Cordership).

4.21 **(a)** 5 switch engines.

(b) One method that works is to use the Cordership reaction from Exercise 4.16(c) to reflect the gliders from this Corderrake backwards:

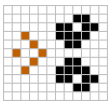


(c) One method that works is to use the glider-to-LWSS reaction displayed in Figure 4.24 to turn each glider from this Corderrake into an LWSS:



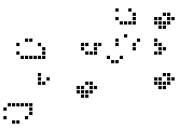
Free PDF. See conwaylife.com/book for pattern files and print version.

4.22 **(a)**



(b) The T-tetromino.

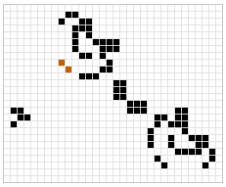
4.23



4.30 **(a)** 3, 18, and 2, respectively.

(b) 3, 8, and 51, respectively.

4.32 This spaceship gives off a duoplet spark at its back end, which can reflect a glider as in Figure 3.17:



Chapter 5: Glider Synthesis

5.2 **(a)**



(b)



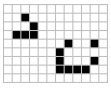
(c)



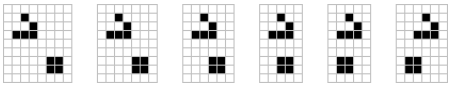
(d)



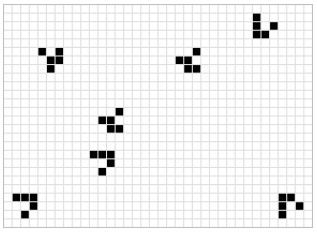
(e)



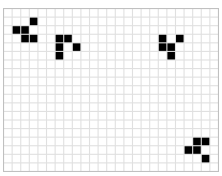
5.4 The six possible ways that a glider can collide with a block are displayed below. From left-to-right, these collisions produce nothing, nothing, nothing, a pi-heptomino, a (pre-)honey farm, and a shifted block. We saw the block-shifting and honey farm-producing collisions in this chapter, and the collision used by the Snark is also the honey farm-producing type. We will see a use of the pi-producing collision in Figure 7.8.



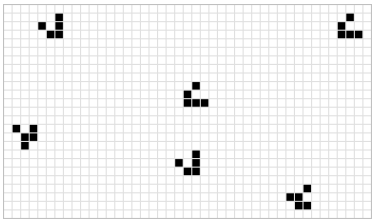
5.5



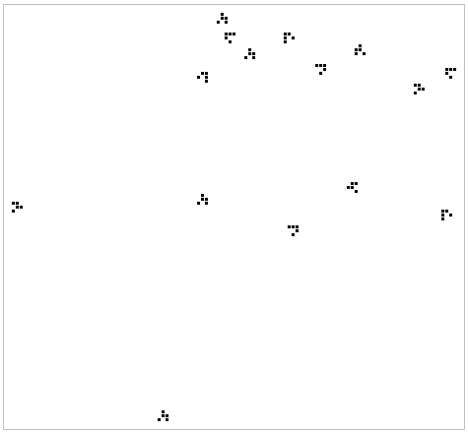
5.13 **(a)**



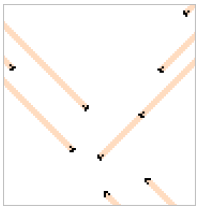
(b)



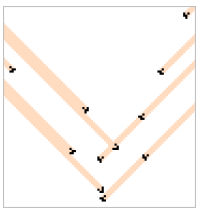
(c)



5.14 **(a)** The following 9-glider synthesis was found by Luka Okanishi almost immediately after the 2-engine Cordership's discovery:

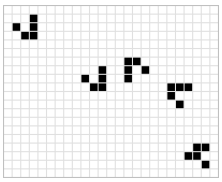


(b)

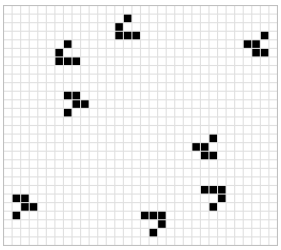


Free PDF. See conwaylife.com/book for pattern files and print version.

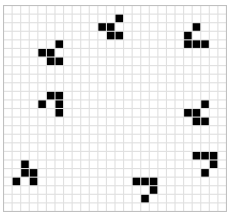
5.15 **(a)**



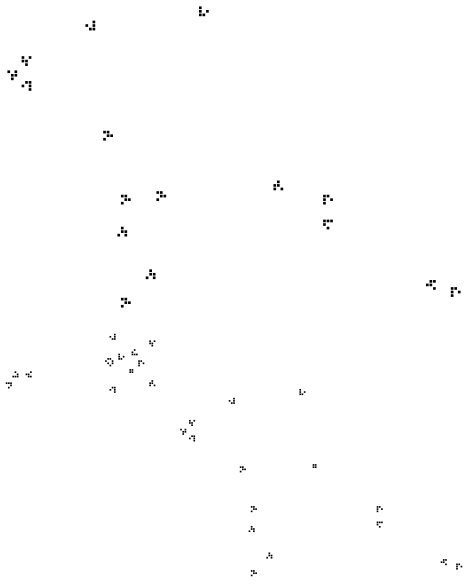
(b)



(c)

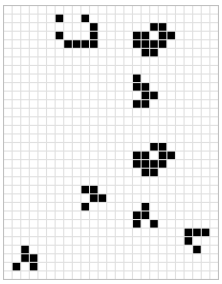


5.17 **(a)**



(b)

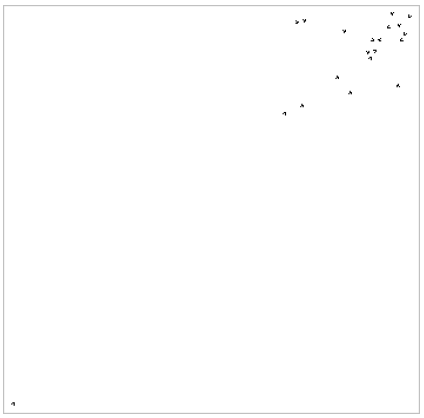
5.19



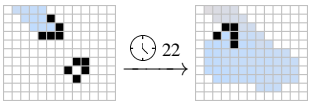
5.29 **(a)** We just remove the 5 gliders that create the boat that trails behind the crab, thus giving us the following 14-glider crab synthesis:



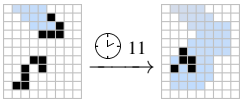
(b) One way is to fire a single glider at the front of the crab (tubstretcher) as shown below. By moving this glider farther away, the resulting still life can be as large as we like (so we can synthesize arbitrarily large strict still lifes via just 20 gliders). For example, the following configuration synthesizes a 209-cell still life.



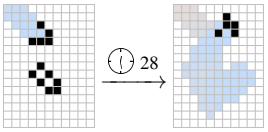
5.31 **(a)**



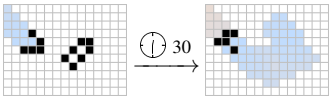
(b)



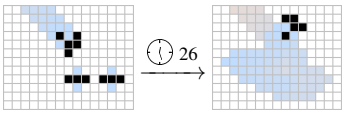
(c) 90°:



180°:

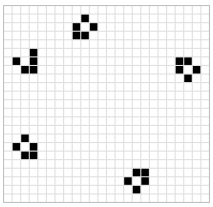


(d)



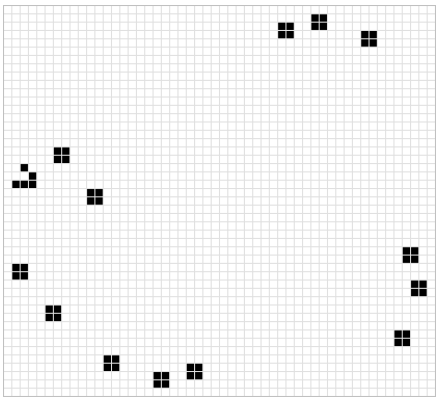
Free PDF. See conwaylife.com/book for pattern files and print version.

5.32 **(a)**

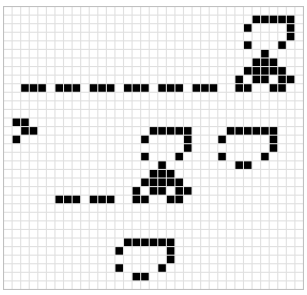


(b) Because the input gliders to the two different turning reactions have different colors. If a single glider hits the same turner four times, it will always end up having the same color that it started with, and thus cannot have the opposite color required to initiate the second turning reaction.

(c)



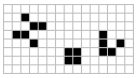
5.33 **(a)**



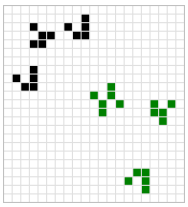
(b) One method that makes use of p60 space rakes is as follows:



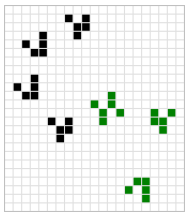
5.34 Our goal is to find a way of firing a glider at a block in such a way that both the glider and the block are destroyed, but a domino spark is produced in the process, thus triggering the clock inserter. One method that works is:



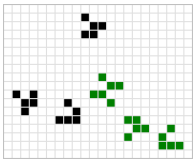
5.35. **(a)**



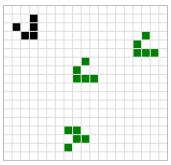
(b)



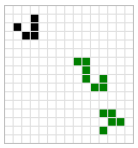
(c)



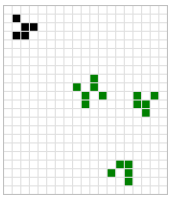
5.36. **(a)**



(b)



(c)



5.41 One solution would be to place a 180-degree reflector in the path of the glider that triggers the clock seed, and then replace it with one of the reflectors from Table 5.5 of opposite parity (e.g., shift the timing by 1 mod 8) if necessary.

5.43 **(a)** The only part of the proof that changes is the bottom row of Table B.2, which becomes (in order, from left to right):

$$3n-10, 23, 20, 17, 14, 11, 1, 38, 43, 48, 53, 5n-2.$$

Since all of these ranks are positive, the proof still works.

(b) $5/3$

(c) The rank of the colliding glider in lane 7 is exactly 0, so we can no longer break ties randomly when inserting gliders into the salvo, and instead we must be careful to break ties by first inserting gliders that are farther to the right along the lines of constant rank before inserting gliders that are to the left.

(d) The minimal value of w is $7/9$, since this choice gives a rank of 0 in lane -7. This time, we must break rank ties by first inserting gliders that are farther to the *left* along lines of constant rank.

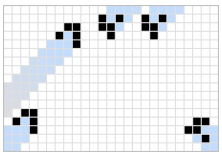
(e) In general, the lines of constant rank have slope $\frac{2w+1}{2w-1}$, so the largest possible slope is $23/5$ (when $w = 7/9$), and the smallest possible slope is $17/11$ (when $w = 7/3$).

5.44 **(a)**

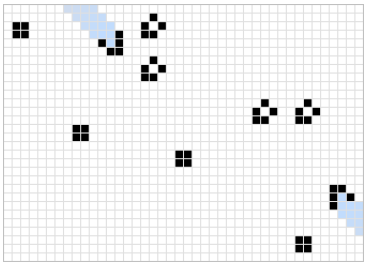


Free PDF. See conwaylife.com/book for pattern files and print version.

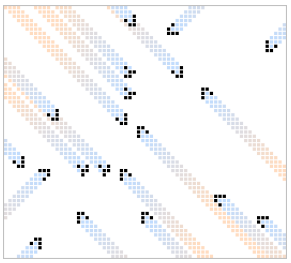
(b)



(c)



(d)



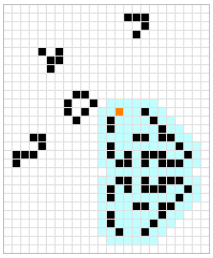
Chapter 6: Periodic Circuitry

6.1 Blocks would get in the way of the central glider track highlighted in green.

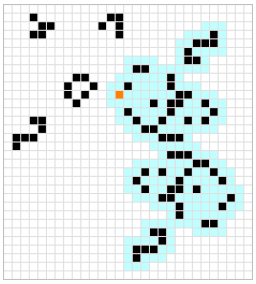
6.23 **(a)** Bumpers are color-preserving, while bouncers are color-changing, so any loop with three bumpers and one bouncer would change the color of the glider exactly once. This is impossible, since the glider's color must return to its starting value by the time the glider completes the loop.

(b) Same as part (a): Snarks are color-preserving, so such a loop would change the glider's color 3 times over the course of the loop. In general, the number of color changes must be even.

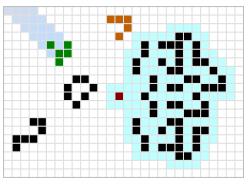
6.24 **(a)**



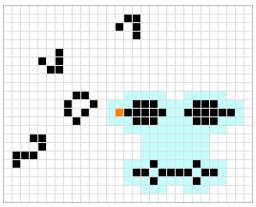
(b)



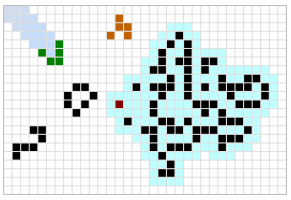
(c)



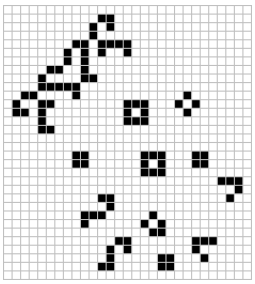
(d)



(e)



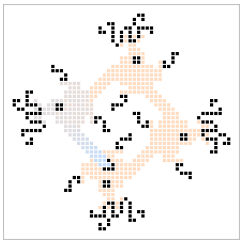
6.25 The p29 shuttle emits a domino (pipsquirter) spark at the bottom, so one method that works is as follows:



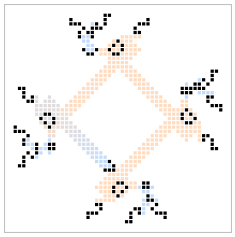
6.27 The repeat time of such a bumper is 44 generations. For a stream with period 44, 55, 66, …, 110, we have smaller bumpers that can perform the same reflection based on p4, p5, p6, …, p5 sparkers.

Free PDF. See conwaylife.com/book for pattern files and print version.

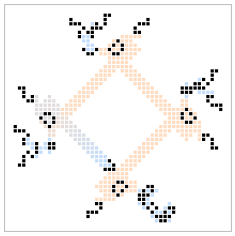
6.30 **(a)** The period of the resulting glider loop is 232, which we could have determined by noting that the original loop had period 216 and this one has the Snarks each 1 cell farther from each other diagonally, so the new period must be $216+4+4+4+4=232$.



(b) The original loop had period 232 and bumpers are 5 generations slower than Snarks, so the new loop has period $232+5+5+5+5=252$:

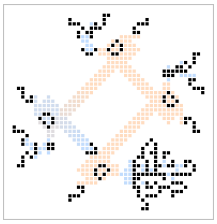


(c) Here we replace the bottom p4 bumper by a p6 bumper:



(d) The loop stops working because its period, 252, is not a multiple of 5.

(e) We shorten the track along two parallel sides by 4 cells to decrease the loop's period to $252-4\times 8=220$, which is a multiple of 5. Replacing the bottom p4 bumper with a p5 bumper then gives the following glider loop:



6.31 The middleweight emulators destroy half of the lightweight spaceship that pass by them, turning those p30 LWSS streams into p60 streams. This usage of the middleweight emulator is called a **filter**—see Section 8.1.1.

6.32 **(a)** 4 more gliders fit in the loop, since we added $2\times 4\times 15=120$ extra generations of travel time around the loop and the gliders are spaces 30 generations apart from each other.

(b) Its period is $30\times 402\,653\,181=12\,079\,595\,430$.

(c) Its period is $30\times 1\,057=31\,710$.

Chapter 7: Stationary Circuitry

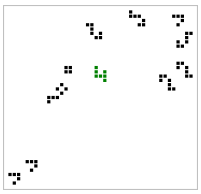
7.1 **(a)** F209, repeat time 60.

(b) R199, repeat time 67.

(c) Fx153, repeat time 60.

(d) L200, repeat time 59.

7.3



7.4 It has more transparent lanes (up to 6 lanes instead of just 2 if we replace the southwest eater 1 with an eater 5 like we did in Exercise 7.3).

7.5 **(a)** The eater 1 at the southwest corner.

(b) The northwestern-most eater 1.

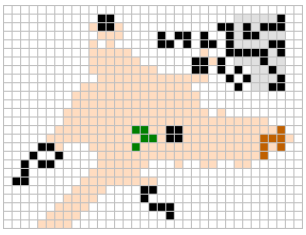
(c) The block.

(d) The northern-central eater 1, and the eater 1 at the southeast.

(e) The eater 1 at the northeast corner.

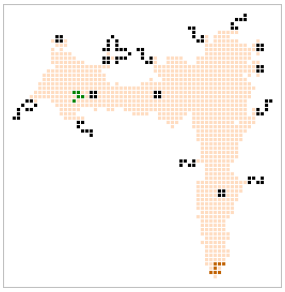
7.6 A pi-heptomino.

7.8



Free PDF. See conwaylife.com/book for pattern files and print version.

7.10 Its repeat time is 90 generations (the same as Lx200):



7.13 For all three of these glider multipliers, we just insert more copies of the Fx77 conduit in the middle of the glider tripler that we saw in Figure 7.11(b):

(a)



(b)



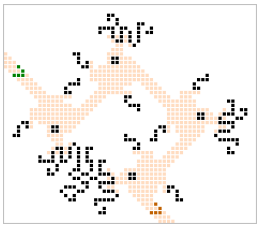
(c)



7.17 **(a)** We can bring them 2 cells southwest, so the glider is delayed by 128 generations instead of 144.

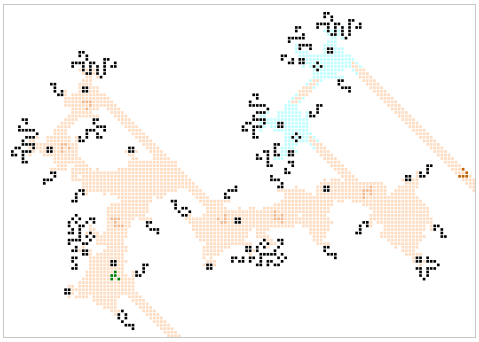
(b) We can bring them 1 cell northwest, but this does not change the glider's delay.

(c)



7.18 The 11th glider is just used to clean up a boat that is left behind. That boat does not interfere with subsequent Corderships being constructed though, so we just didn't bother cleaning it up in the stable conduit.

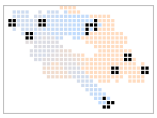
7.20 **(a)**



(b) Just remove the tub in one or both of the semi-Snarks, thus altering which of their input gliders they let through.

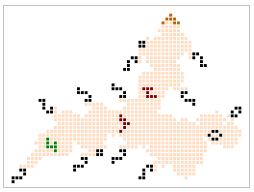
(c) The CP semi-cenark is 3 generations faster than the CP semi-Snark, so replacing a single CP semi-Snark in the circuits from part (b) (which had mod-8 timings of 0, 1, 6, and 7 generations) with a CP semi-cenark results in circuits with timings of $0-3 \equiv 5$, $1-3 \equiv 6$, $3-3 \equiv 0$, and $7-3 \equiv 4 \pmod 8$. The only remaining timing is $2 \pmod 8$, which can be obtained by replacing the *other* CP semi-Snark in the $5 \pmod 8$ circuit with a CP semi-cenark.

7.27 As the conduit name suggests that the Herschel is turned 180 degrees in 60 generations, a closed loop should have a period of 120 generations.

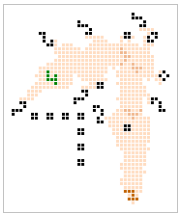


7.30 **(a)** The top-left eater 1 in BR146H overlaps with the bottom-right eater 1 in the original HFx58B.

(b) This conduit is named HL262B:

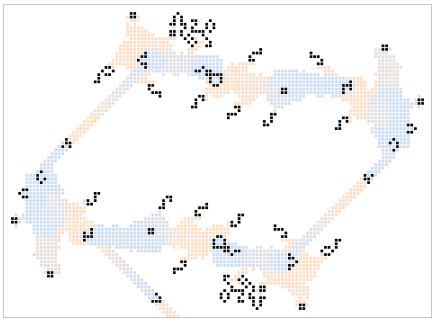


7.32 The variant of HFx58B from Exercise 7.30 is needed to connect to the following conduit, and the final (rightmost) variant of BFx59H needed to dodge the obstacle.



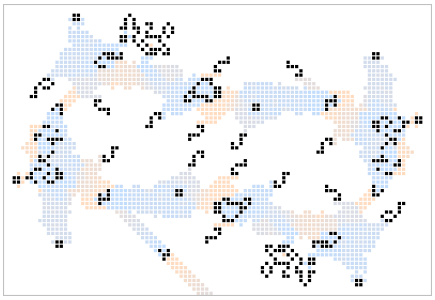
7.33 The Herschel is converted into an R-pentomino (well, generation 4 of its evolution anyway), then a B-heptomino (well, a B-heptaplet), and then back into a Herschel. The conduits that do the conversions are called HLx69R, RF28B, and BFx59H.

7.38 **(a)**



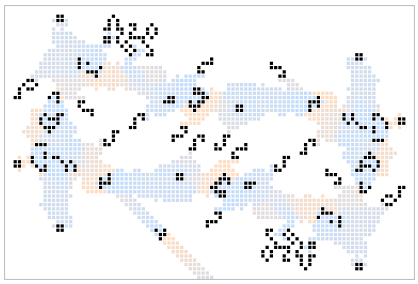
Free PDF. See conwaylife.com/book for pattern files and print version.

(b)



(c) It does not work because syringe overclocking does not work at period 76 or 77—the re-created block hasn't quite settled yet, and a glider following at that distance doesn't make a clean pi heptomino.

(d) Here is a p74 gun:



(e) A p73 following glider hits the re-forming block in a way that doesn't form the correct pi heptomino (similar to part (c)). Also, the two halves of the gun cannot be moved close enough together (the rows of eaters get in each other's way).

7.43 Travelling counter-clockwise from the glider that the ship makes, the stable circuits used are: a Bandersnatch, a Snark, and a syringe attached to an Lx200 conduit (as introduced in Exercise 7.10). That Lx200 conduit duplicates the signal—its output Herschel rebuilds the ship while feeding a glider into the "rectifier" reflector from Exercise 3.23, and its output glider is fed into two Snarks and the glider duplicator from Figure 7.11(a).

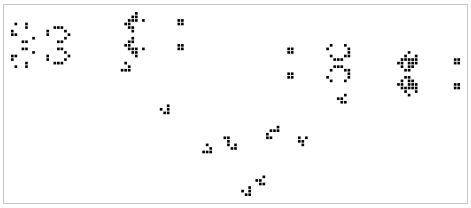
7.46 Just change the rightmost block to an eater 1 to turn the final two-glider output into a three-glider output, as in the difference between Figures 7.5(b) and 7.5(c).

Chapter 8: Guns and Glider Streams

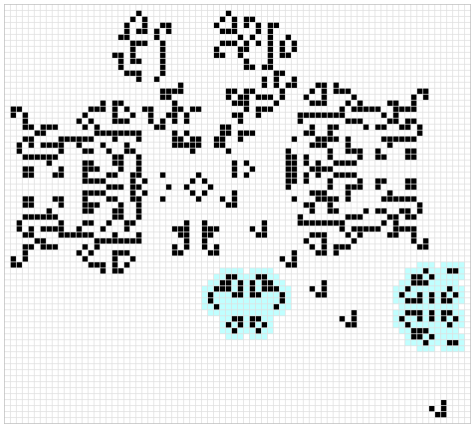
8.1



8.3 Since $322/7 = 46$, we use two p46 twin bees guns:



8.5 One possibility is to use two copies of Rich's p16 as below. Alternatively, the copy of Rich's p16 at the right can be replaced by a blocker.

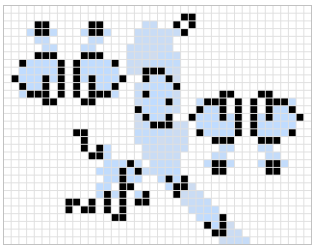


8.6 The filter itself would have to have period 4 (or 2) in order to be able to filter out any of these glider streams. But on the other hand, a glider only moves by 1 cell every 4 generations, so if the spark from the filter affects any gliders then it would have to affect all of them.

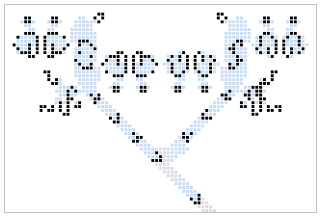
Free PDF. See conwaylife.com/book for pattern files and print version.

8.13 **(a)** A pi-heptomino.

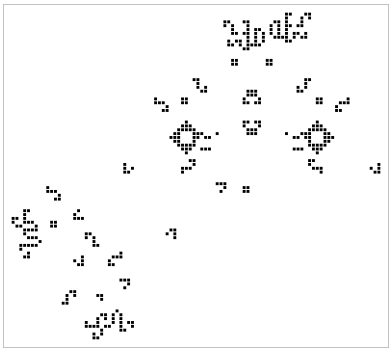
(b)



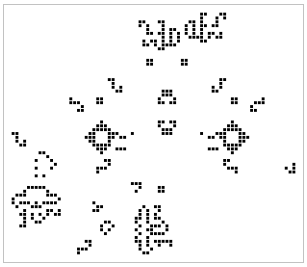
(c) There are some smaller possibilities, but one of the simplest ones to build is as follows:



8.19 **(a)**



(b) Period 5 bumpers are the only ones that we have seen that work (p10 bumpers exist, but p5 are almost always preferable):



Chapter 9: Universal Computation

9.10 Each bit that is stored in the binary register increases its bounding box by 6×6, except for the first few bits that are not right at the edge. Some trial and error and manipulation by hand shows that it takes 182 bits for the bounding box to be 4000×4000 or larger, so it will take 1 000 more bits (i.e., 1 182 in total) for it to be 10000×10000 or larger.

We are thus left with the question of how long it takes before the binary register stores the value 2^{1181}. Each clock tick takes 2^{20} generations, and some investigation of APGsembly 9.5 reveals that it requires approximately $3 \times 2^{n+1}$ clock ticks to store 2^n in the binary register. It thus takes approximately $2^{20} \times 3 \times 2^{1115} = 3 \times 2^{1182}$ generations for the size of the bounding box to exceed 10000×10000 cells.

9.15 This merge circuit corresponds to the `ITER6;Z` line of APGsembly. The reason that the merge circuit is so far west is that the *next* state is `ITER11`, which is dozens of lines later in the APGsembly code.

9.16 **(a)** Just recall that the Taylor series for $\arctan(x)$ centered at $x = 0$ is

$$\arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots,$$

with this series representation being valid when $-1 \le x \le 1$. Since $\arctan(1) = \pi/4$ (by looking at a triangle with angles $\pi/4$, $\pi/4$, and $\pi/2$), we conclude that

$$\frac{\pi}{4} = \arctan(1) = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots.$$

Multiplying both sides of this equation by 4 gives us the desired series.

(b) If we split up the terms in the series from part (a) as suggested by the hint, we get

$$\begin{aligned}
\pi &= (2+2) - \left(\frac{2}{3} + \frac{2}{3}\right) + \left(\frac{2}{5} + \frac{2}{5}\right) - \cdots \\
&= 2 + \left(2 - \frac{2}{3}\right) - \left(\frac{2}{3} - \frac{2}{5}\right) + \left(\frac{2}{5} - \frac{2}{7}\right) - \cdots \\
&= 2 + \left(\frac{4}{1 \cdot 3} - \frac{4}{3 \cdot 5} + \frac{4}{5 \cdot 7} - \cdots\right),
\end{aligned}$$

as desired. Note that regrouping the parentheses like we did does not change the value of the series, despite it only being conditionally convergent, since we did not change the *order* of the terms.

Free PDF. See conwaylife.com/book for pattern files and print version.

(c) Using summation notation now, the series from part (b) can be written in the form

$$\begin{aligned}\pi &= 2+\sum_{k=0}^{\infty}\frac{4(-1)^k}{(2k+1)(2k+3)}\\ &= 2+\sum_{k=0}^{\infty}(-1)^k\left(\frac{2}{(2k+1)(2k+3)}\right.\\ &\qquad\left.+\frac{2}{(2k+1)(2k+3)}\right)\\ &= 2+\frac{2}{3}+\sum_{k=0}^{\infty}(-1)^k\left(\frac{2}{(2k+1)(2k+3)}\right.\\ &\qquad\left.-\frac{2}{(2k+3)(2k+5)}\right)\\ &= 2+\frac{2}{3}+\sum_{k=0}^{\infty}(-1)^k\frac{8}{(2k+1)(2k+3)(2k+5)},\end{aligned}$$

as desired.

(d) Every time we use this method of splitting the terms of the series into two copies of half of those terms, we extract one more term from the parentheses. The next few transformations of the series are as follows:

$$\begin{aligned}\pi &= 2+\frac{2}{3}+\sum_{k=0}^{\infty}\frac{8(-1)^k}{(2k+1)(2k+3)(2k+5)}\\ &= 2+\frac{2}{3}+\frac{4}{15}\\ &\quad+\sum_{k=0}^{\infty}\frac{24(-1)^k}{(2k+1)(2k+3)(2k+5)(2k+7)}\\ &= 2+\frac{2}{3}+\frac{4}{15}+\frac{12}{105}\\ &\quad+\sum_{k=0}^{\infty}\frac{96(-1)^k}{(2k+1)(2k+3)(2k+5)(2k+7)(2k+9)}.\end{aligned}$$

In general, after we perform this procedure m times, we get the series

$$\begin{aligned}\pi &= 2\sum_{k=0}^{m-1}\frac{k!}{1\cdot3\cdots(2k+1)}\\ &\quad+\sum_{k=0}^{\infty}\frac{4m!(-1)^k}{(2k+1)(2k+3)\cdots(2k+2m+1)}.\end{aligned}$$

The infinite sum above is an alternating series with decreasing terms, so it is no larger than its first term: $4m!/(1\cdot3\cdots(2m+1))$, which tends to 0 as $m\to\infty$. It follows that

$$\begin{aligned}\pi &= 2\lim_{m\to\infty}\sum_{k=0}^{m-1}\frac{k!}{1\cdot3\cdots(2k+1)}\\ &\quad+\lim_{m\to\infty}\sum_{k=0}^{\infty}\frac{4m!(-1)^k}{(2k+1)(2k+3)\cdots(2k+2m+1)}\\ &= 2\sum_{k=0}^{\infty}\frac{k!}{1\cdot3\cdots(2k+1)}+0,\end{aligned}$$

as desired.

(e) This comes immediately from the fact that, for every integer $k\geq1$, the k-th term in the series from part (d) is $k/(2k+1)$ times the previous term. A bit more explicitly,

$$\begin{aligned}\frac{\pi}{2} &= 1+\frac{1!}{3}+\frac{2!}{3\cdot5}+\frac{3!}{3\cdot5\cdot7}+\frac{4!}{3\cdot5\cdot7\cdot9}+\cdots\\ &= 1+\frac{1}{3}\left(1+\frac{2}{5}+\frac{2\cdot3}{5\cdot7}+\frac{2\cdot3\cdot4}{5\cdot7\cdot9}+\cdots\right)\\ &= 1+\frac{1}{3}\left(1+\frac{2}{5}\left(1+\frac{3}{7}+\frac{3\cdot4}{7\cdot9}+\cdots\right)\right)\\ &= 1+\frac{1}{3}\left(1+\frac{2}{5}\left(1+\frac{3}{7}\left(1+\frac{4}{9}+\cdots\right)\right)\right).\end{aligned}$$

Multiplying through by 2 then (finally!) gives exactly the desired series (9.1).

9.17 We claim that if p_n, q_n, and r_n denote the top-left, top-right, and bottom-left entries of B_n, respectively, then $p_n=2n!$, $q_n=(2n+1)(2(n-1)!+q_{n-1})$ with $q_1=6$, and $r_n=3\cdot5\cdot7\cdots(2n+1)$. These claims can be proved by induction on n (and we already demonstrated the $n=1,2,3,4$ base cases in Section 9.6).

With these formulas in hand, it suffices to expand the quantity

$$2\left(1+\frac{1}{3}\left(1+\frac{2}{5}\left(\cdots\left(1+\frac{n-1}{2n-1}\right)\right)\right)\right)$$

and see that it equals q_n/r_n. To this end, we expand and get

$$2\left(1+\frac{1!}{3}+\frac{2!}{3\cdot5}+\cdots+\frac{(n-1)!}{3\cdot5\cdot7\cdots(2n-1)}\right).$$

A common denominator of this quantity is $3\cdot5\cdot7\cdots(2n-1)=r_n/(2n+1)$. The corresponding numerator satisfies the recurrence relation $c_n=(2n-1)c_{n-1}+2(n-1)!$ with $c_1=2$, so $c_n=q_n/(2n+1)$. The above quantity thus equals $(q_n/(2n+1))/(r_n/(2n+1))=q_n/r_n$, as desired.

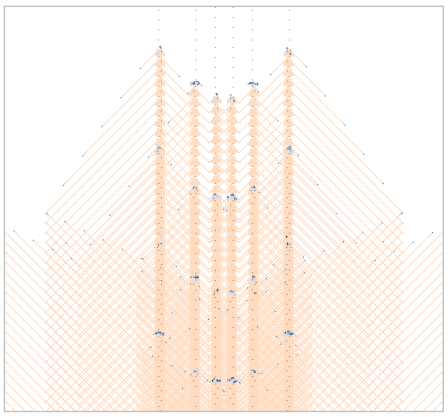
9.18 We change the first 12 lines of that APGsembly to the following:

```
INITIAL;  ZZ;  INIT1;     READ T0
INIT1;    *;   INIT2;     SET T0, READ T2
INIT2;    *;   INIT3;     NOP, SET T2, INC R6
INIT3;    ZZ;  INIT4;     NOP, INC R0, INC R6
INIT4;    ZZ;  INIT5;     NOP, INC R6, INC R6
INIT5;    ZZ;  ITSTART;   NOP, INC R6, INC R6
ITSTART;  ZZ;  ITSTRTB;   NOP, INC R5, INC R5
ITSTRTB;  ZZ;  ITTEST;    NOP, INC R5, INC R5
ITTEST;   ZZ;  IT1;       TDEC R5
IT1;      Z;   IT5;       TDEC R3
IT1;      NZ;  MULA1;     NOP, INC R1
```

Free PDF. See conwaylife.com/book for pattern files and print version.

Chapter 10: Self-Supporting Spaceships

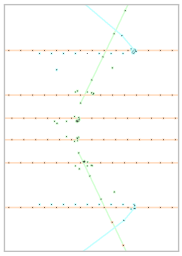
10.2 **(a)**



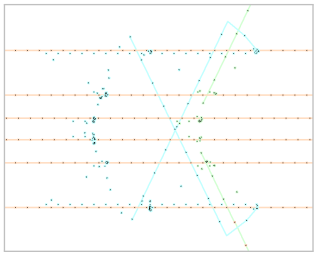
(b) R4L1.

(c) One rephaser followed by R2L23F outputs a glider on the same lane (lane 1) and is shorter.

10.5 **(a)**



(b)



As a side note, this is the same rake as the one from Exercise 10.2, but without the double kickbacks added to it.

(c) R2L16.

(d) Even though this rake is more efficient than the equivalent one given in Table 10.1 (8 rephasers followed by R6L6), we do not ever use a lane 16 rake building silverfish. Furthermore, lane 16 is the *only* one improved by R2L16 – all other lanes can be fired on more efficiently by R2L25.

10.7 (0 and 11 in either order), 8, 7, (29 and 13 in either order), 1, (15, 16, 19, or 26), 8, 0, 14, 22, (1, 2, 3, 4, 5, or 6).

10.10 **(a)** 17.

(b) 23 (plus any non-negative integer multiple of 17, depending on the spacing specifics of the reader's construction, so other possible answers include 40, and 57, and so on).

(c) Repeating the method of part (a) m times lets us synthesize a blinker track that is $17m$ cells farther away than the track from Figure 10.12, and repeating the method of part (b) n times lets us synthesize a blinker track that is $23n$ cells farther away. The number of cells between these two blinker tracks is $17m-23n$. Since $\gcd(17,23)=1$, we know from Bézout's identity (Theorem A.1) that for any spacing c of our choosing, we can find positive integers m and n so that $17m-23n=c$.

(d) Use two copies of configuration from Figure 10.12 on the same pair of blinker tracks. In one of them, move the forward rake back by 13×34 generations, which makes the synthesized blinker trail $13\times 17=221$ cells farther away. In the other one, insert 11×2 extra rephasers on each track, which makes the synthesized blinker trail $11\times 23=253$ cells farther away. The spacing between these tracks is $253-221=32$ cells, as desired.

10.11 $p=3$, $q=7$ works. The smallest positive integers in this case are $m=11$ and $n=19$, which is what Equation (10.1) gives. The trick is to realize that the values of m and n are pushed up when p/q is close to the $1/2$ speed limit—in particular, when $q-2p=1$.

10.13 Backward gliders: $113/45$ (since over the course of 180 generations, the pi-heptominoes and glider separate by 45 cells in the x-direction and $4\times 17+45=113$ cells in the y-direction.
Forward gliders: $23/45$ by the same calculation, but with the y-displacement being $4\times 17-45=23$ cells.

10.18 $56/95$. The slope is the slope between the input and output gliders of the $(23,5)c/79$ reaction, not the slope that the Herschel moves at.

10.19 The reflector from Figure 10.13(b) does not have the correct timing—we would need to combine it with other components like the glider-shifting component from Figure 10.13(a), thus increasing the total xWSS cost. The adjustable reflector from Exercise 10.16 is too long and slow (i.e., its output glider is too far behind its input glider).

10.20 It serves the same purpose as the glider duplicators in the caterpillar. Since the period of the helix is 2×79, not 79 itself, a single copy of the helix only produces half as many gliders as are needed to support the track of Herschels. The loaf fills this gap—it carries the Herschel along in the exact same way that a glider does.

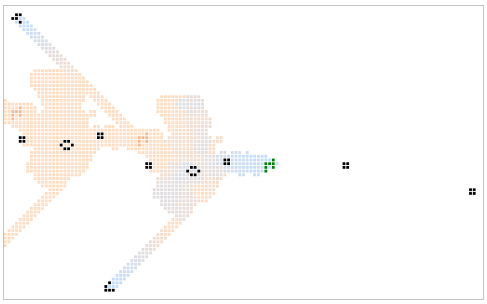
10.23 The backward glider is used at the front end of the caterloopillar to synthesize the unidirectional glider synthesis seed. Since the loaf-pulling and loaf-pushing flotillae themselves only release forward gliders, they need help to form the seed that they collide into.

Free PDF. See conwaylife.com/book for pattern files and print version.

10.24 The 3-xWSS component from Exercise 10.16 reflects near its front and thus can be used to create much faster helices than the 2-xWSS component from Figure 10.22. Helices made from that slower component can be sped up by adding additional copies of the 2-xWSS glider shifter from Figure 10.13(a), but doing so typically makes the helix much larger (and thus more expensive to construct).

10.25 It creates a seed for back half of caterloopillar (which is then used to construct the forward-moving loaf-pushing ships).

10.30 (a)



(b) The block-based Herschel climber might be easier to use since the blocks can be synthesized at essentially any time, whereas the gliders in the glider-based Herschel climber have to show up at exactly the right time. Also, the Herschel travelling on a block trails gives two output gliders that can be used for syntheses rather than just one. However, the glider-based Herschel climber might be easier to use since it just requires one glider per period (i.e., 156 generations) instead of two to synthesize the block.
Overall, it is probably easier to build a spaceship from the block-based track, as the timing constraint probably requires extra kickbacks reactions and/or other circuitry to get around that would be more expensive than the single extra glider needed to synthesize a block.

10.33 The $(13,1)c/31$ and $(27,1)c/72$ crawlers from Figures 10.25(a) and 10.25(b) require helices, since they travel faster than $c/4$ in one of the orthogonal directions. The $(34,7)c/156$ crawler from Figure 10.25(c) does not require a helix since it travels slower than $c/4$ in both directions, so the front-end of a self-supporting spaceship based on it can be supported more easily by forward gliders.

10.34 (a) We want the helix to travel at a speed of at least $13c/31$ in one of the directions. In the notation of the speed (10.2), this means that we want

$$\frac{10k+10m+20}{20k+20m+101} \geq 13/31.$$

Solving for $k+m$ gives $k+m \geq \lceil 693/50 \rceil = 14$, so at least $2\times 14+2+3=33$ xWSSes are needed to build this helix ($2\times 14=28$ xWSSes for the glider-shifting components, and $2+3=5$ for the two reflecting ends).

(b) Suppose that $k+m=14$, as suggested in part (a). We can adjust its speed by adding ℓ cells of free glider travel time, and by delaying the glider duplicator by n generations. In one of the directions, the helix then has a speed of $(10\times 14+\ell+20)c/(20\times 14+n+4\ell+101)=13c/31$. However, this equation has no positive integer solutions (it is equivalent to $21\ell+13n=7$), so no helix of this speed exists.

(c) To construct such a helix with 61 xWSSes, we can set $k=m=n=14$, and have a glider travel $\ell=25$ cells uninterrupted in one of the directions (but 0 cells uninterrupted in the other direction). The resulting helix has speed

$$\frac{(10k+10m+\ell+20, 13k-13m+\ell)c}{20k+20m+n+4\ell+101} = \frac{(13,1)c}{31},$$

as desired.

Chapter 11: Universal Construction

11.4 They are just there so that both ends of the Gemini are identical (and can thus be constructed via the same glider recipe that bounces between them). The unused reflectors at the northwest end are used at the southeast end, and the unused reflectors at the southeast end are used at the northwest end.

11.9 (a) $(1024,5120)c/33699578$.

(b) The ends can be squeezed together 509 full diagonals (508 more than in part (a)), for a speed of $(1024,5120)c/33695514$.

11.10 Mathematically, we are being asked which positive integers p and q are such that $p/q=(m+2048)/(m-2048)$ for some integer $m\geq 3072$. When $m=3072$, we get a slope of 5, and increasing m decreases the slope. Furthermore, solving for m in terms of p and q gives $m=2048(p+q)/(p-q)$. It follows that the attainable slopes p/q are exactly those between 1 and 5 (and by reflection of the Geminoid, between $1/5$ and 1) for which $2048(p+q)/(p-q)$ is an integer (i.e., for which $p-q$ is a divisor of $2048(p+q)$).

11.15 The four-glider slow salvo that this single-lane sequence produces is a p2 slow salvo, so the timing of the perpendicular gliders that we fire matters mod 2.

11.17 -12.

Free PDF. See conwaylife.com/book for pattern files and print version.

11.21 147 gliders.

11.23 If we bring the ends of the Demonoid n cells closer together, its period decreases to $2^{21}-8n$ and its speed increases to $128c/(2^{21}-8n)$. If we squeeze the ends $n=62379$ cells closer together, its bounding box will have size $200\,000\times 199\,950$, its period will be $1\,598\,120$, and is speed will be $128c/1598120 = 16c/199765$.

11.27 **(a)** Since we can insert a few copies of this recipe into the Demonoid without actually increasing its size or period, its new speed is simply $194c/2097152 = 97c/1048576$.

(b) Only $2/3$ of the 44hd block pushes are being used to advance from the current construction site (i.e., the current Scorbie splitter) to the next construction site (i.e., the next Scorbie splitter). The remaining $1/3$ of these block pushes are being used to build the Snark *after* that next Scorbie splitter, but we still can't use the next construction site until these additional $1/3$ of the block-pushing gliders are safely out of the way.

(c) The mysterious $(4/3)88$ term in the denominator comes from the fact that gliders cannot start using the next construction site (i.e., the next Scorbie splitter) until the return glider in the Snarkbreaker makes its way backward past that construction site. If we use an extra n 44hd block pushes in the Demonoid, then it will take $88n$ generations for gliders to go from the current construction site (i.e., the current Scorbie splitter) to the final Snark that is constructed, and another $(1/3)88n$ generations for the return glider to get past the next construction site, allowing it to be used. Altogether, this adds $(4/3)88n$ generations to the period of the Demonoid that consists of just empty space along the glider recipe's trail.

(d) $\dfrac{(2/3)22c}{2009+(4/3)88} = \dfrac{44c}{6379}$.

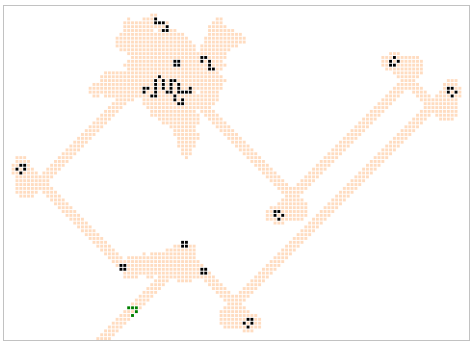
11.29 The Scorbie splitter is about 150–200 lanes off to the side of the single-channel glider recipe (coming from the northwest), so zero-degree recipes would be very expensive (and not already located in pre-existing recipe databases, which only cover up to about 100 lane offsets).

Chapter 12: The 0E0P Metacell

12.5 Even though two copies of this oscillator can be placed in the same loop so as to create a period $272/2 = 136$ oscillator, the two copies interact with each other by suppressing each other's sparks. For example, they suppress each other's dot sparks at generation 20.

12.6
```
MAPARYXfhZofugWaH7oaIDogBZofuhogOiAaIDog
IAAgAAWaH7oaIDogGiA6ICAAIAAaIDogIAAgACAA
IAAAAAAAA
```

12.10 **(a)**



(b) The still lifes displayed in part (a) act as one-time turners and one-time splitters, so they can be separated and repositioned in many different ways, making them easier to position around nearby circuitry and glider paths in the 0E0P metacell.

12.15 That glider destroys a block in a syringe. This blocks the path that is used to copy the parent's single-channel glider stream into the child's nucleus, since we do not want that to happen anymore (an entire copy of that glider stream has already been copied).

12.16 The $1+(7-5)+8(7-3)+64(7-4)+512(7-2) =$ 2787th subsequence.

12.17 Since $1141 = 1+(7-3)+8(7-1)+64(7-6)+512(7-5)$, its northeast, southeast, southwest, and northwest parents were in states 5, 1, 6, and 3, respectively.

Free PDF. See conwaylife.com/book for pattern files and print version.



Bibliography

[BC98] David J. Buckingham and Paul B. Callahan. Tight bounds on periodic cell configurations in Life. *Experimental Mathematics*, 7(3):221–241, 1998.

[BCG82] Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy. *Winning Ways for Your Mathematical Plays, volume 2: Games in Particular*, chapter 25. Academic Press, 1982.

[Bos99] Robert A. Bosch. Integer programming and Conway's game of Life. *SIAM Review*, 41(3):596–604, 1999.

[BT04] Robert A. Bosch and Michael Trick. Constraint programming and hybrid formulations for three life designs. *Annals of Operations Research*, 130:41–56, 2004.

[Coo03] Matthew Cook. *Still life theory*, pages 93–118. Santa Fe Institute Studies in the Sciences of Complexity. Oxford University Press, 2003.

[CS12] Geoffrey Chu and Peter J. Stuckey. A complete solution to the Maximum Density Still Life Problem. *Artificial Intelligence*, 184:1–16, 2012.

[CSdlB09] Geoffrey Chu, Peter J. Stuckey, and Maria Garcia de la Banda. Using relaxations in maximum density still life. In *Proceedings of the Fifteenth International Conference on Principles and Practice of Constraint Programming*, pages 258–273, 2009.

[Elk98] Noam D. Elkies. The still-life density problem and its generalizations. In *Voronoï's Impact on Modern Science, Book I*, volume 21 of *Proceedings of the Institute of Mathematics of the National Academy of Sciences of Ukraine*, pages 228–253. Institute of Mathematics, 1998.

[Epp02] David Eppstein. Searching for spaceships. In *More Games of No Chance*, volume 42 of *MSRI Publications*, pages 433–453. Cambridge University Press, 2002.

Free PDF. See conwaylife.com/book for pattern files and print versio

[Fla94] Achim Flammenkamp. Natural grown oscillators out of random soup. http://wwwhomes.uni-bielefeld.de/achim/oscill.html, 1994. [Online; accessed Dec. 16, 2021].

[Fla04] Achim Flammenkamp. Frequency of Game of Life objects in settled random areas. http://wwwhomes.uni-bielefeld.de/achim/freq_top_life.html, 2004. [Online; accessed Dec. 16, 2021].

[Gar70] Martin Gardner. The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223:120–123, 1970.

[Gar71] Martin Gardner. On cellular automata, self-reproduction, the Garden of Eden and the game of "life". *Scientific American*, 224:112–117, 1971.

[Gar83] Martin Gardner. *Wheels, Life, and Other Mathematical Amusements*. W. H. Freeman and Company, 1983.

[Gib06] Jeremy Gibbons. Unbounded spigot algorithms for the digits of pi. *The American Mathematical Monthly*, 113(4):318–328, 2006.

[Gos84] R. William Gosper. Exploiting regularities in large cellular spaces. *Physica*, 10D:75–80, 1984.

[Gou10] Adam P. Goucher. Universal computation and construction in GoL cellular automata. In *Game of Life Cellular Automata*, chapter 25, pages 505–518. Springer London, 2010.

[HD74] Jean Hardouin-Duparc. Paradis terrestre dans l'automate cellulaire de Conway. *Revue française d'automatique, informatique, recherche opérationnelle*, 8:63–71, 1974.

[Hed69] Gustav A. Hedlund. Endomorphisms and automorphisms of the shift dynamical systems. *Mathematical Systems Theory*, 3:320–375, 1969.

[LMN05] Javier Larrosa, Enric Morancho, and David Niso. On the practical use of variable elimination in constraint optimization problems: 'still-life' as a case study. *Journal of Artificial Intelligence Research*, 23:421–440, 2005.

[Moo62] Edward F. Moore. Machine models of self-reproduction. *Proceedings of Symposia in Applied Mathematics*, 14:17–33, 1962.

[Myh63] John Myhill. The converse of Moore's Garden-of-Eden theorem. *Proceedings of the American Mathematical Society*, 14:685–686, 1963.

[Nie03] Mark D. Niemiec. Synthesis of complex Life objects from gliders. In *New Constructions in Cellular Automata*, pages 55–78. Oxford University Press, London, 2003.

[Nie10] Mark D. Niemiec. Object synthesis in Conway's Game of Life and other cellular automata. In *Game of Life Cellular Automata*, chapter 8, pages 115–134. Springer London, 2010.

[Okr03] Andrzej Okrasinski. Andrzej Okrasinski's website dedicated to the Game of Life. web.archive.org/web/20091027025324/http://geocities.com/conwaylife/, 2003. [Online; archived from the original and accessed Dec. 16, 2021].

[Pou85] William Poundstone. *The Recursive Universe*. William Morrow and Company Inc., 1985.

Free PDF. See conwaylife.com/book for pattern files and print version.

[Ren16] Paul Rendell. *Turing Machine Universality of the Game of Life*. Springer International Publishing, 2016.

[Slo96] Neil J. A. Sloane. Sequence A019473 in *The On-Line Encyclopedia of Integer Sequences*. oeis.org/A019473, 1996. Number of stable n-celled patterns ("still lifes") in Conway's Game of Life, up to rotation and reflection. [Online; accessed Dec. 16, 2021].

[Slo99] Neil J. A. Sloane. Sequence A046932 in *The On-Line Encyclopedia of Integer Sequences*. oeis.org/A046932, 1999. $a(n) =$ period of $x^n + x + 1$ over $GF(2)$, i.e., the smallest integer $m > 0$ such that $x^n + x + 1$ divides $x^m + 1$ over $GF(2)$. [Online; accessed July 28, 2018].

[Slo00] Neil J. A. Sloane. Sequence A056613 in *The On-Line Encyclopedia of Integer Sequences*. oeis.org/A056613, 2000. Number of n-celled pseudo still lifes in Conway's Game of Life, up to rotation and reflection. [Online; accessed Dec. 16, 2021].

[Slo11] Neil J. A. Sloane. Sequence A196447 in *The On-Line Encyclopedia of Integer Sequences*. oeis.org/A196447, 2011. The number of parents of successive approximations used in a greedy approach to creating a Garden of Eden in Conway's Game of Life. [Online; accessed Dec. 16, 2021].

[Wai74] Robert T. Wainwright. Life is universal! *Proceedings of the 7th conference on Winter simulation*, 2:449–459, 1974.

[Wol02] Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 2002.

Free PDF. See conwaylife.com/book for pattern files and print version.

Free PDF. See conwaylife.com/book for pattern files and print version.



Index

Symbols

π calculator 296
(11,0) block pull 138, 356, 362
(11,0) block push 138
(2,1) block pull 41, 137, 245, 272
(2,1) block push 138
(23,5)/79 reaction 327
0E0P metacell 112, 385
2G-to-H 241
31c/240 reaction 311
44hd elbow push 383
90-degree elbow 353

A

Achim's p144 219
acorn 18
ADD component 288
adjustable rake 110, 116
aircraft carrier 33
airforce 453
alive 3
ambidextrous 214
apgsearch 28, 77, 111, 147
APGsembly 276
ark 15, 90
armless construction 250
ash 5

B

B-heptaplet 17, 203
B-heptomino 11, 27, 80, 94, 123, 237
B245 186
B29 87
B60 186
backward rake 313
bakery 29
banana spark 61, 124, 224, 456
Bandersnatch 191
barge 33, 87
barperpole 150
beacon 7, 65
beehive 4, 123
beehive stopper 206
Beluchenko's p7 59
BF22P 205
BFx59H 205, 218
bi-block 34, 109, 123
bi-block fuse 104
big glider 111
billiard table 101, 127

Free PDF. See conwaylife.com/book for pattern files and print versio

binary register 285
binary ruler 287
binary weight 386
bipole 53
blinker 5, 123
blinker fuse 104
block 20, 41, 123, 136
block on table 34
block-laying switch engine 14, 456
blockade 9, 29, 36
blocker 59, 172, 224, 229
blockic seed 143, 151
blockic splitter 140
BLx19R 203, 205
boat 40, 123, 150
  bit 41, 174
  with tail 38
boatstretcher 113, 150, 372
boojum reflector 81, 113
bookend 39
bouncer 113, 171
bounding box 16, 288
BR146H 217
breeder 134, 379
BRx46B 205
BSE22T31 205
buckaroo 61, 113, 154, 246
Bullet's p10 237
bumper 171
bunnies 30
  10b 18
  9 18
burloaferimeter 55
Bx/Sy . . . . . *see* rulestring
Bx106 186
Bx202 186

C

CA . . . . . *see* cellular automaton
caber tosser 254
Callahan G-to-H 211, 346, 352
camelship 379
Canada goose 87, 113
canoe 38
cap 39
catalyst 185
caterer 60, 454
caterloopillar 108, 330
caterpillar 108, 112, 321
cauldron 55
CE 397
cell 3
cellular automaton 5, 387
centipede 337
century 216
character printer 292
chicken wire 44
child 3
cis 38
climber . . . . . *see* crawler
clock 7, 20, 127, 136, 139, 143, 149
  gun 279, 399
  inserter 143, 437
CN 397
Coe ship 97, 214
color-changing 85
color-preserving 85
component stack 279
computational universality 178, 271
computer 279
conduit 68, 183
confused eaters 56
construction arm 250, 346
control clock gun 399
copperhead 106, 111, 451
Corderrake 115
Cordership 90, 108, 131, 149, 369, 402
crab 87, 150, 372
crawler 338

D

David Hilbert 63
dead 3
DEC 272
Demonoid 108, 367, 389
demultiplexer 206, 219
destruction arm 348
diameter 288
dinner table 56, 61
domino spark 58, 87, 143, 224
doppler effect 335
dot spark 59, 87, 224
double kickback 340
double-barrelled gun 165, 180, 218

Free PDF. See conwaylife.com/book for pattern files and print version.

duoplet 61, 455
duoplet spark 163

E

eater 10, 39
1 . . . . . 30, 33, 88, 123, 150
2 . . . . . 51, 88, 127
3 . . . . . 41, 51
5 . . . . . 148, 215
ecologist 95, 96, 129
edge shooter 168
Edna 30
elbow 245, 353
elementary cellular automaton 391
elementary spaceship 106
emu 240
engineered spaceship 107
eureka 65
evolution 8
exponential filter 263
extra long 37

F

F116 186
F117 . . . . . 186, 214
F166 215
factory 205
familiar four 29
fast forward force field 104, 149, 179
fd . . . . . *see* full diagonal
featherweight spaceship . . . . . *see* glider
Fermat prime 259
Fibonacci number 308, 310
figure eight 58, 64, 173
filter 179, 224
finger spark 60
finite-state machine 275
first natural block 207
fishhook . . . . . *see* eater 1
fleet 29
flotilla 89, 110
forward rake 313
fountain 59, 64, 180
full diagonal 85
fumarole 58, 127
fuse 103, 259, 373, 384
Fx119 186
Fx119 inserter 189
Fx77 . . . . 69, 70, 186, 187, 211, 215, 216, 240

G

Garden of Eden 20
gcd . . . . . *see* greatest common divisor
Gemini 112, 346
Geminoid 108, 350
gencols 324
generation 3
glider 6, 123, 125
color 84, 140, 171
duplicator 157
emulator 113
gun 11, 121
lane 85
loop 65
pusher 148, 158
stopper 206
synthesis 121
timing 85, 171
glider-producing switch engine 14, 125
GoL-destroy 374, 384
Gosper glider gun 11, 84, 87, 231
gourmet 62, 79, 204
greatest common divisor 432
grin 20
growing spaceship 150

H

half diagonal 85
half-baked knightship 338
half-bakery 338
hand 355, 360, 406
HashLife 281, 310, 385, 423
hassler 61
hat 29, 128
hd . . . . . *see* half diagonal
heavyweight
emulator 58, 454
spaceship 7, 125, 127, 128
supervolcano 80
volcano 58
hebdarole 58
Heisenburp 169, 209, 315

Free PDF. See conwaylife.com/book for pattern files and print version.

helix 323
Hensel notation 444
heptomino
  B . . . . . *see* B-heptomino
  pi . . . . . *see* pi-heptomino
Herschel 17, 68
  receiver 213
  track 68, 196
  transceiver 212, 242
  transmitter 213
Hertz oscillator 55
HF95P 205
HFx58B 205, 217
HighLife 386, 427
highway robber 208
hivenudger 89, 114
HLx69R 205
honey
  bit 88
  farm 9, 29, 34, 123, 245, 266
  thieves 56, 61
HWSS . . . . . *see* heavyweight spaceship

I

INC 272, 285
induction coil 39, 54
inline inverter 156
INS 397
INT rules 5
interchange 123
irregular stream 156
isotropic 5, 388, 444
iterated logarithm 258

J

jam 53, 235
Jason's p22 . . . . . 56, 224, 233
Jason's p33 235
Jason's p36 235
JavaLifeSearch 23, 31
Jormungant's G-to-H 241

K

Karel's p15 173
keeper 207
kernel 395, 398
kickback reaction 123, 149
killer toads 88
knightship 107, 350, 390
Koch snowflake 302
Kok's galaxy 59

L

L112 . . . . . 82, 184, 186, 211, 241
L156 . . . . . 184, 186, 218, 240
L200 241
lcm . . . . . *see* least common multiple
LE 397
least common multiple 433
Lidka 18, 29
Life-like 4, 387
Lifeline 26, 49
lightweight
  spaceship 7, 121, 125, 128
  supervolcano 80
linear propagator 379, 387
LN 397
loaf 41, 123
loafer 106, 111, 114
lobster 107
long 37
long barge 37, 87
long boat 33, 150
long canoe 38
long long boat 37
long long ship 37
long long snake 38
long ship 37
long snake 33
long^3 snake 38, 147
lookup table 400
looping spaceship . . . *see* reflectorless rotating oscillator
LRBEM 407
LS 397
LSE11T-8 204
lumps of muck 9, 61, 123, 208
LW 397
LWSS . . . . . *see* lightweight spaceship
Lx163 186
Lx200 . . . . . 215, 355
Lx86 186

Free PDF. See conwaylife.com/book for pattern files and print version.

M

machine gun 199, 281
MAP 447
max 434
megacell . . . . . *see* metacell
memory cell 167, 180
Mersenne prime 267
metablinker 424
metacell 385, 422
metageneration 385, 404
metaglider 386
metapixel . . . . . *see* metacell
methuselah 16
middleweight
  emulator 59, 64, 454
  spaceship 7, 125, 128
  supervolcano 59, 80, 236
  volcano 59, 64, 226
mod . . . . . *see* modular congruence
modular congruence 431
mold 53
monochrome 248, 333
Moore neighborhood 5, 302
MSW-1T1 204
MUL component 291
MWSS . . . . . *see* middleweight spaceship
MWSS out of the blue 169

N

NE30T3 . . . . . 189, 231
NE5T-4 . . . . . 189, 214
negentropy 55
neighbor 5
new gun . . . . . *see* twin bees gun
Noah's ark 30
non-isotropic 388, 390, 446
NOP 286
nucleus 395, 402
NW31 . . . . . *see* NW31T120, 216, 267
NW31T120 . . . . . 189, 211
NW31T120_SE7T14 189
NZ 272

O

oblique spaceship 107
octagon 2 53
omniperiodic 73, 101
one-time turner 138, 206, 370
onion rings 44
Orion 87
Orion 2 87
orphan 22
Orthogonoid 379
oscillator 6, 53
OTCA metapixel 422
outer-totalistic 387
overclock 383
overclocking 191
overweight spaceship 90
OWSS . . . . . *see* overweight spaceship

P

p1 megacell 423
p5760 metacell 422
parallel half-baked knightship 338
parallel HBK . . . . . *see* parallel half-baked knightship
parent 3
pentadecathlon 7, 57, 125, 128, 154, 226, 229
pentoad 56
period 7, 53, 83
period multiplier 198
PF81H 205
phase 6
phase shift oscillator 80
phi spark 180
phoenix 75, 392
pi crawler 322
pi portraitor 454
pi-heptomino . . . 9, 62, 80, 112, 123, 237, 461
pianola breeders 379
pinwheel 453
pipsquirter 58, 79, 171, 226
PL8P 204, 205
PNW6T138 205
polyomino 8
polyplet 27
pond 123
PR127R 205
PR8P 204
PR9B 205
pre-beehive 4, 30, 61, 88
pre-block 20, 55

Free PDF. See conwaylife.com/book for pattern files and print version.

pre-honey farm 9, 61, 454
pre-pulsar 64
pre-pulsar shuttle 65, 180
prime number 159, 310
primer 159
pseudo
  spaceship 89
  still life 34
pseudo-period gun 231
puffer 14, 90, 94
pulsar 7, 64, 125
pushalong 88

Q

quadratic filter 262
quadri-Snark 217, 281
quasi still life 50
queen bee 10, 125, 128
  shuttle 11, 61, 148
quetzal 240
quetzalcoatlus . . . . . *see* quetzal
quinti-Snark 217

R

R-pentomino 16, 27, 125
R126 186
R1L0 318
R2L23 318
R2L25 318
R64 . . . . . 69, 186, 241
R6L17 318
R6L6 318
rabbits 30
rake 94, 109, 121
rattlesnake 59, 63
RCT . . . . . *see* reverse caber tosser
READ 286
read head 285, 301
reburnable wick 312
recovery time 39
rectifier 78, 81, 113
recursive filter 255
reflector 65, 66, 84
reflectorless rotating oscillator 381, 389
regulator 174
relatively prime 433
relay 113
Remini 380
repeat time 67
rephaser 114, 244
replicator (pattern) 386
replicator (rule) 388, 425
reverse caber tosser 147
rewind 179
RF28B 203, 205
RF29P 205
RFx36R 205, 217
Rich's p16 . . . . . 77, 132, 149, 224, 236
RNW3T46 205
Rob's p16 . . . . . 77, 224
rock 40, 185
roteightor 56
row printer 294
RR56H 205
RRO . . . . . *see* reflectorless rotating oscillator
Rule 18 391
rulestring 5, 386, 387, 444, 446
Rx140 186
Rx164 186

S

salvo 136
sawtooth 268, 388
SBR . . . . . *see* sliding block register
Schick engine 96, 111, 117, 451
Scientific American 26
Scorbie splitter 366, 441
scrubber 453
SE39T32 189
SE39T59 189
SE7T14 189
seed 136, 143, 151
self-supporting spaceship 311
semi-cenark 216
semi-Snark 198, 216, 221, 399, 404
SET 286
shell 394, 398, 402
shield bug 337
ship 46, 128
shuttle 61
sidecar 89
signal 100
signal elbow 101

Free PDF. See conwaylife.com/book for pattern files and print version.

Silver reflector 67, 211, 381
silverfish 108, 311
Simkin glider gun 217
single-channel synthesis 353, 394, 402
sink 101
Sir Robin 107
slide gun 245
sliding block register 272
slmake . . . . . *see* slsparse
slow glider pair 251
slow glider pairs 249
slow salvo 249
slsparse 143, 151, 190, 367
SMOS . . . . . *see* spaceship made of spaceships
SMOSMOS *see* spaceship made of spaceships made of spaceships
snacker 55–57, 79
snake 33, 40, 128
Snark 67, 81, 85, 138, 211, 457
Snarkbreaker 365, 371
Snarkmaker 363, 440
soba 106, 108
soup 5, 131
source 101
space rake 95, 129
spacefiller 434
spaceship 6, 83
spaceship made of spaceships . . 381, 389, 426
spaceship made of spaceships made of spaceships 390
spaghetti monster 111
spark 57, 171
sparker 58
Spartan 190, 252, 399
speed 83
Speed Demonoid 374
speed of light 83
spider 106
spiral growth 379, 388
splitter 279
sqrtgun 254, 262
stable 38
stable reflector 67
stairstep hexomino 9, 27
state-lookup clock gun 401, 416
still life 4, 33
strange loop 330
StreamLife 385
strict still life 150
SUB component 288
subroutine loop 403
superfountain 59, 236
supervolcano 60, 80
SW1T43 . . . . . 218, 248
switch engine 13, 90, 121, 125, 131, 149
  block-laying . . . . *see* block-laying switch engine
  glider-producing . . . *see* glider-producing switch engine
syringe 190, 214, 355

T

T-nosed p4 . . . . . 60, 79, 229
T-nosed p6 79
T-tetromino 8, 27, 61, 63, 128, 130
table 39
tagalong 86, 88
tail 37, 55
tandem 212, 242
Tanner's p46 . . . . . 62, 167
TDEC 273, 286
teardrop 123
tee 124, 149, 151, 225, 247, 338
TEST 272
thumb spark 60, 87, 172
tick 3
ticker tape gun 165
toad 7, 88
toggle 158
TOLLCASS 28
totalistic 387
tractor beam 256
traffic jam 82
traffic light 8, 29, 123, 128
trans 38
transparent
  catalyst 185
  lane 188, 274
tremi-Snark 216
trigger glider 409
TRIGGER1 375
TRIGGER2 375
trombone slide 194
true-period gun 231
tub 87, 128

Free PDF. See conwaylife.com/book for pattern files and print version.

with tail 38, 128
with tail eater 40
tubstretcher 87, 113
tumbler 451
Turing complete 271
twin bees 11, 127, 148
gun 13, 84, 126, 231
shuttle 13, 61
twin primer 176, 179
TWIT . . . . . *see* tub with tail eater
two blockers hassling R-pentomino . . . . . *see* Wainwright's p72
two eaters 55, 56
two-glider mess 123
two-time turner 150

U

ultrafountain 237
unary 284, 400
universal construction 345
universal constructor 345
universal regulator 174, 268, 281
unix 59, 172, 454

V

vacuum 99
variant 186
very long 37
volcano 60
von Neumann neighborhood 5, 392

W

Wainwright's p72 219
waterbear 108, 327
weekender 106, 111
welding 42
wick 103
wickstretcher 87, 259, 372
wire 100
worker bee 56

X

xWSS 89

Z

Z 272
zebra stripes 44, 99, 434
zero-degree elbow 359
zero-degree reflector 191

Free PDF. See conwaylife.com/book for pattern files and print version.