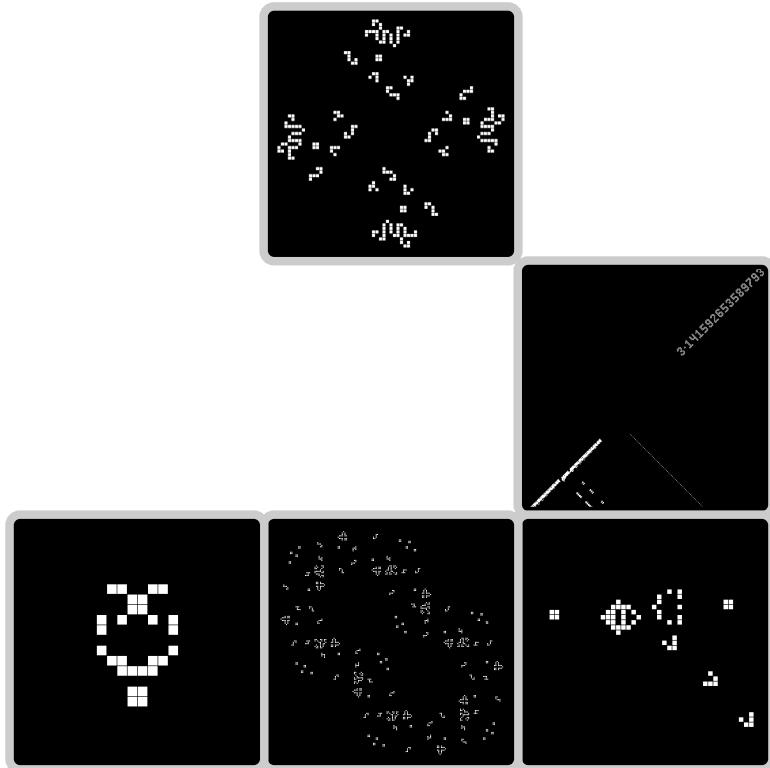


Conway's Game of Life

Mathematics and Construction



Nathaniel Johnston and Dave Greene

Early draft (May 19, 2020).

Not for public dissemination.

Early draft (May 19, 2020). Not for public dissemination.

Copyright © 2020 Nathaniel Johnston and Dave Greene

CONWAYLIFE.COM

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

Early draft (May 19, 2020).

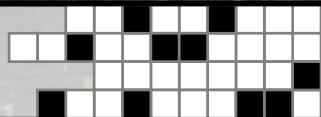
Not for public dissemination.

Early draft (May 19, 2020).

Not for public dissemination.



Contents



Preface	vii
The Goal	vii
Intended Audience	vii
How to Use	viii
Acknowledgments	ix

I

Classical Topics

1 Early Life	3
1.1 Our First Technique: Random Fumbling	5
1.2 Common Evolutionary Sequences	7
1.3 The Queen Bee	9
1.4 The B-Heptomino and Twin Bees	11
1.5 The Switch Engine	12
1.6 Methuselahs and Stability	15
1.7 Gardens of Eden	19
Notes and Historical Remarks	24
Exercises	26
2 Still Lifes	31
2.1 Strict and Pseudo Still Lifes	32
2.2 Still Life Grammar	35
2.3 Eaters	36
2.4 Welded and Constrained Still Lifes	39

2.5	Still Life Density	41
	Notes and Historical Remarks	45
	Exercises	47
3	Oscillators	51
3.1	Billiard Tables	52
3.2	Stabilizing Corners	53
3.3	Composite Periods and Sparks	54
3.4	Hasslers and Shuttles	58
3.5	Glider Loops and Reflectors	62
3.6	Herschel Tracks	64
3.7	Omniperiodicity	69
3.8	Phoenices	71
	Notes and Historical Remarks	73
	Exercises	75
4	Spaceships and Moving Objects	79
4.1	The Glider	80
4.2	The Light, Middle, and Heavyweight Spaceships	83
4.3	Corderships	86
4.4	Puffers and Rakes	88
4.5	Speed Limits	93
4.6	Speed and Period Status	100
	Notes and Historical Remarks	105
	Exercises	107

II

Circuitry and Logic

5	Glider Synthesis	113
5.1	Two-Glider Syntheses	114
5.2	Syntheses Involving Three or More Gliders	116
5.3	Incremental Syntheses	118
5.4	Synthesis of Moving Objects	119
5.5	Developing New Syntheses	123
5.6	A Gosper Glider Gun Breeder	125
5.7	Slow Salvo Synthesis	127
	Notes and Historical Remarks	135
	Exercises	136
6	Periodic Circuitry	143
6.1	Period 30 Circuitry	144
6.2	Primer	148
6.3	Period 46 Circuitry	152

6.4	Bumpers and Bouncers	159
6.5	Glider Timing and Regulators	161
	Notes and Historical Remarks	164
	Exercises	167
7	Stable Circuitry	171
7.1	Herschel Conduits	171
7.2	Gliders to Herschels and Back	175
7.3	Glider-to-Herschel converters	176
7.4	Synthesizing Objects via Conduits	178
7.5	Period Multipliers and Small High-Period Guns	182
7.6	Converters for Other Objects	188
7.7	Factories	191
	Notes and Historical Remarks	194
	Exercises	196
8	Guns and Glider Streams	201
8.1	Glider Deletion	201
8.2	Glider Insertion	204
8.3	Streams of Other Spaceships	206
8.4	Glider Guns of Any Period	208
8.5	True-Period Guns	209
8.6	Slide Guns	221
8.7	Armless Construction	224
8.8	Slow and Irregular Guns	228
	Exercises	236

III

Constructions

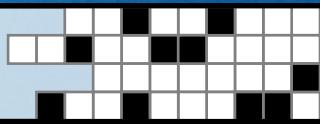
9	Universal Computation	243
9.1	A Computer in Life	243
9.2	A Compiled APGsembly Pattern: Adding Registers	249
9.3	Multiplying and Re-Using Registers	253
9.4	A Binary Register	254
9.5	A Character Printer	262
9.6	A Pi Calculator	263
9.7	A 2D Printer	269
	Exercises	274
10	Self-Supporting Spaceships	279
10.1	Caterpillar	279
	Exercises	279

11	Universal Construction	281
11.1	Demonoids	281
	Exercises	281
12	The OEOP Metacell	283
12.1	Rule Emulation	283
12.2	Structure of the Metacell	284
12.3	Memory Tape	284
12.4	Logic Circuitry	284
12.5	Construction	285
12.6	Encoding Other Cellular Automata in Life	285
12.7	Notes and Historical Remarks	285
	Exercises	287

Appendices and Supplements

A	Mathematical Miscellany	291
A.1	Modular Arithmetic	291
A.2	Greatest Common Divisor and Least Common Multiple	291
A.3	Big-O Notation	292
B	Universality of the Clock Inserter	293
B.1	Timing of Tight Glider Salvos	293
B.2	Timing of the Clock Inserter	294
C	Extra APGsembly Code	297
D	Solutions to Selected Exercises	301
	Bibliography	314
	Index	317

Preface



The Goal

This book provides an introduction to Conway's Game of Life, the interesting mathematics behind it, and the methods used to construct many of its most interesting patterns. This book generally tries to avoid presenting patterns in isolation or as historical notes, but rather tries to guide the reader through the thought processes that went into creating them in the first place.

While we will largely follow the history of the Game of Life as we go through the book, we emphasize that this is *not* the primary goal of the book, but rather it is a by-product of the fact that most recently-discovered patterns build upon patterns and techniques that were developed earlier. Instead, 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.

Intended Audience

While this book does not have any formal mathematical or computer science prerequisites, it is written at a level aimed at students of at least a first-year undergraduate university level. 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 $\sum_{k=1}^n k^2$ for adding numbers, are used frequently and without much explanation. 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.

More specifically, we prove some simple theorems about the Game of Life in Chapters 1 through 5. 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 understand convergence issues. Finally, big-O notation is used to discuss the growth rate of patterns in Sections 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).

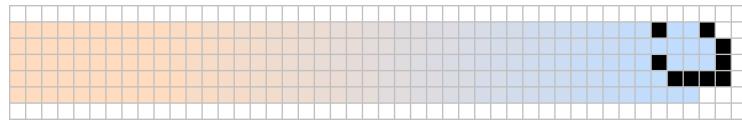


Figure 1: An object moving to the right with a gradient behind it indicating how long ago it was in each location. White cells were never alive, black cells are currently alive, blue cells were alive recently, and orange cells were alive long ago (roughly 75 generations or more).

- 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.

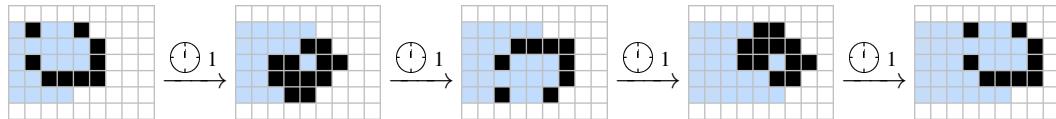


Figure 2: The same object as in Figure 1, but displayed in a more explicit fashion. This image shows how the pattern changes every time 1 generation elapses.

- 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, but highlight 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, dark orange is typically used for the output of the reaction, and dark red is used otherwise (see Figure 4).

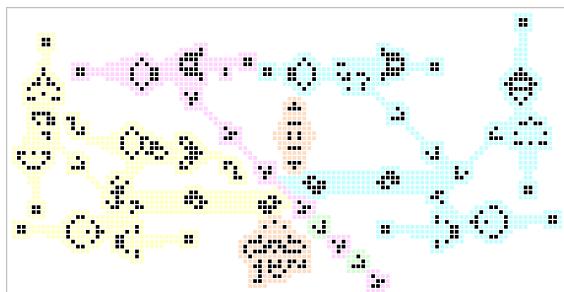


Figure 3: A glider gun made up of several different component reactions that are highlighted in different colors. In particular, gliders are created by a gun highlighted in magenta, lightweight spaceships are created by guns highlighted in aqua and yellow, and those lightweight spaceships are merged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

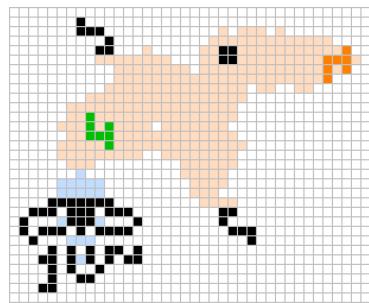


Figure 4: A dark green “Herschel” comes in from the left and creates the dark orange Herschel on the right. Cells highlighted in blue are constantly changing due to being part of an oscillator, whereas cells that are light orange are usually dead, except when the Herschel passes through.

- 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 copy and pasted into Life simulation software like Golly (golly.sourceforge.net) so that it can be explored and manipulated.¹ 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 D.

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 Nicolay Beluchenko, Steven Eker, Mark Niemiec, and Michael Simkin for helpful conversations about content of the book. Thanks to Andrew Trevorrow and Tomas Rokicki for creating the open-source cross-platform CA 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.no-ip.biz/lifeview), which is used on this book’s website (conwaylife.com/book) to display patterns and make them interactive. Thanks

¹For an explanation of how RLE or Macrocell code works, see conwaylife.com/wiki/Run_Length_Encoded or conwaylife.com/wiki/Macrocell.

to Velimir Gayevskiy and Mathias Legrand for the *Legrand Orange Book* LaTeX template from LaTeXTemplates.com.

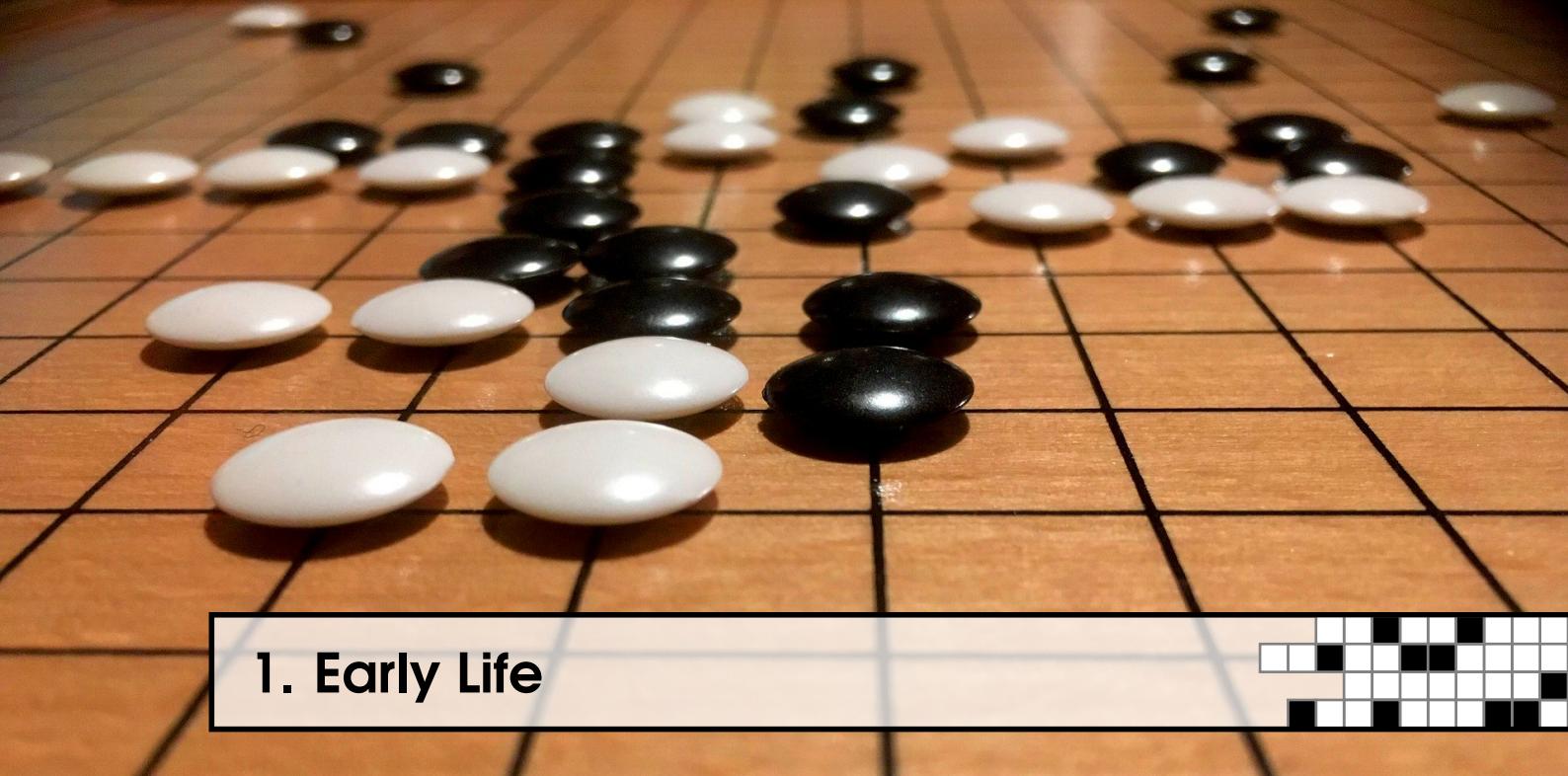
Finally, the authors would like to thank their wives Kathryn and Melanie for tolerating them during their years of mental absence glued to this book, and their parents for encouraging them to care about both learning and teaching. Many thanks also to Mount Allison University for giving the first author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

Classical Topics

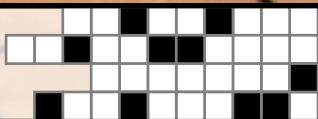
1	Early Life	3
1.1	Our First Technique: Random Fumbling	
1.2	Common Evolutionary Sequences	
1.3	The Queen Bee	
1.4	The B-Heptomino and Twin Bees	
1.5	The Switch Engine	
1.6	Methuselahs and Stability	
1.7	Gardens of Eden	
	Notes and Historical Remarks	
	Exercises	
2	Still Lifes	31
2.1	Strict and Pseudo Still Lifes	
2.2	Still Life Grammar	
2.3	Eaters	
2.4	Welded and Constrained Still Lifes	
2.5	Still Life Density	
	Notes and Historical Remarks	
	Exercises	
3	Oscillators	51
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	
	Notes and Historical Remarks	
	Exercises	
4	Spaceships and Moving Objects	79
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	
	Notes and Historical Remarks	
	Exercises	

Early draft (May 19, 2020).

Not for public dissemination.



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 Life¹ 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 2 live neighbors and thus live to the next generation, and no dead cell has 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*.

¹Named for its inventor, John H. Conway.

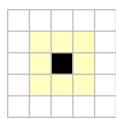


Figure 1.1: A live cell (in black) in the middle and its 8 neighbors (in yellow).

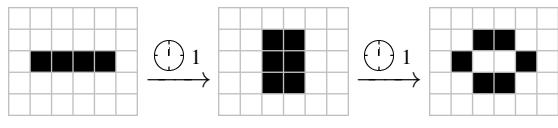


Figure 1.2: A line of four alive cells takes two generations to evolve into a stable object called a *beehive*. The 3×2 object in the middle is sometimes called a *pre-beehive*.

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 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 rules that can be constructed simply by specifying different 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 is *chaotic*. 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, so it is 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. 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 rules are sometimes studied as well, and for brevity they are typically described using a *rulestring* of the form Bx/Sy , where we replace “ x ” by all numbers of live neighbors that lead to the **birth** of a dead cell, and we replace “ y ” by all numbers of live neighbors that lead to the **survival** of a live cell. For example, the Game of Life is described by the rulestring $B3/S23$. We will comment on other rules from time to time (especially in Chapter 10), typically to highlight how they contrast with 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 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*. These are all potentially interesting modifications to make, and they fall under the general umbrella of *cellular automata*, but we will not consider them any further in this book. 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.

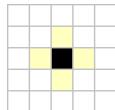


Figure 1.3: A live cell (in black) in the middle and the 4 cells in its von Neumann neighborhood (in yellow).

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

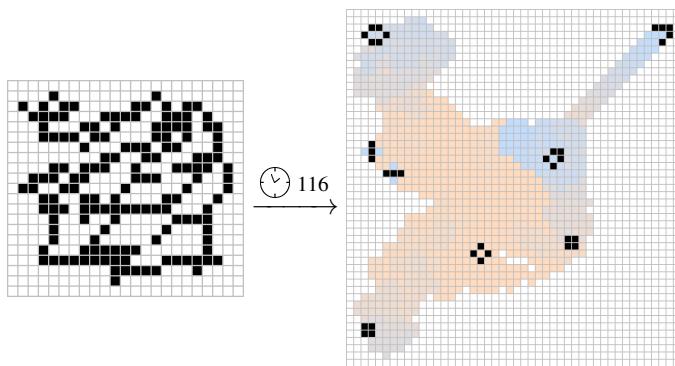


Figure 1.4: Evolving random junk is a decent way to find some small patterns to get us started in Life. The debris on the right introduces us to several still lifes, our first oscillator, and the glider.

In the ash, we see the beehive that we were introduced to 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* that rotates itself by 90 degrees every generation. Patterns like this one, which cycle between

²This 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.

³The term “ash” refers to the fact that it is what is left after a pattern stops “burning”.

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.

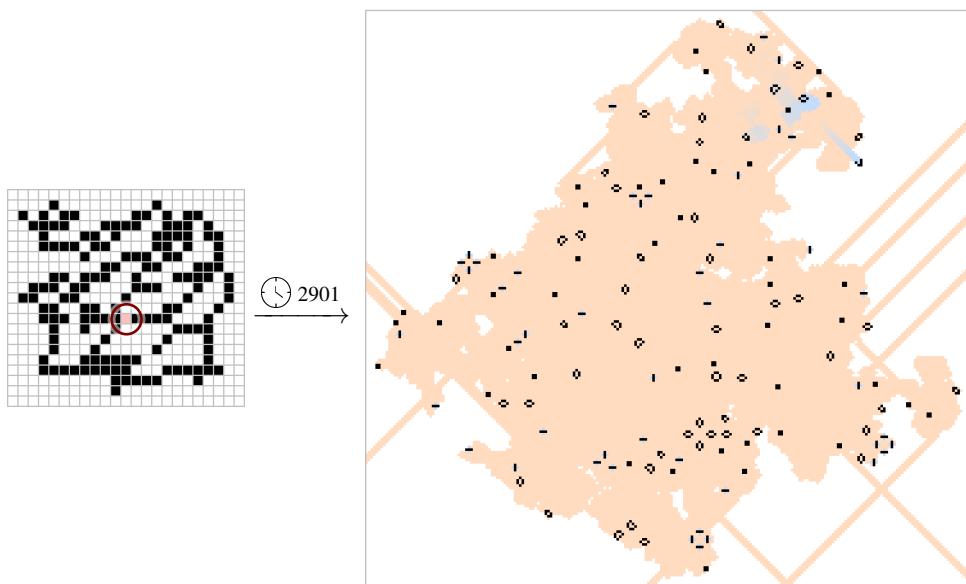


Figure 1.5: Changing just a single cell (circled on the left in red) from alive to dead in our random starting configuration causes its evolution to become completely different. It now takes over 25 times as long to stabilize and its ash contains three new types of still lifes (ponds, ships, and loaves).

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?

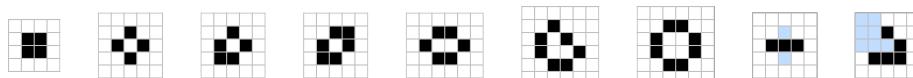


Figure 1.6: From left to right: seven still lifes, called the *block*, *tub*, *boat*, *ship*, *beehive*, *loaf* and *pond*, an oscillator called the *blinker*, and a spaceship called the *glider*. All of these objects frequently appear in the ash left behind by chaotic patterns.

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.

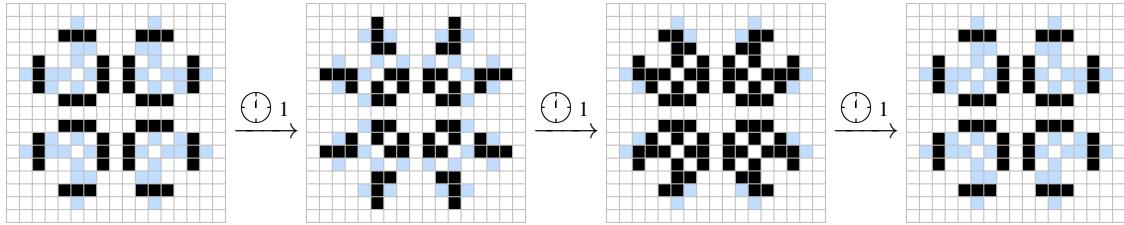


Figure 1.7: A period 3 oscillator called the *pulsar*.

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.

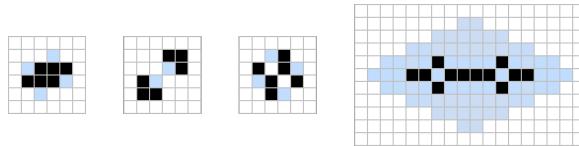


Figure 1.8: From left to right: three oscillators with period 2, called *toad*, *beacon* and *clock*, and a period 15 oscillator called *pentadecathlon*.

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.

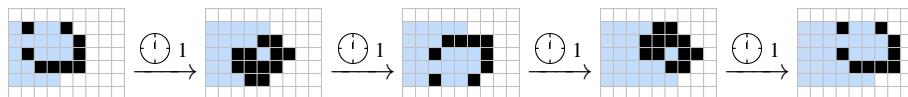


Figure 1.9: The *lightweight spaceship*, which has period 4 and moves orthogonally to the right by 2 cells every 4 generations.

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 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.

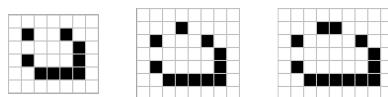


Figure 1.10: From left to right: a lightweight, middleweight, and heavyweight spaceship. They all have period 4 and travel to the right 2 cells every 4 generations.

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

⁴In 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.

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.

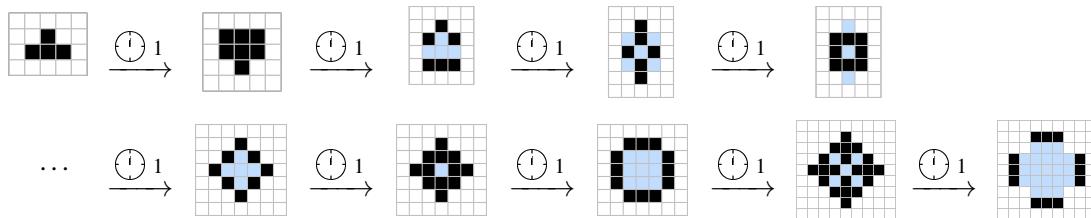


Figure 1.11: A *T-tetromino* (top-left) takes 9 generations to evolve into an arrangement of four blinkers called a *traffic light* (bottom-right).

The T-tetromino also illustrates how symmetry spontaneously forms within Life patterns. While it starts off with only left-right symmetry, after four 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.

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 *honeyfarm*. Appropriately enough, the 7-cell object that starts this evolution is called the *pre-honeyfarm* (as is any other small pattern that follows the same evolution). While only one honeyfarm appears in the ash in Figure 1.5, many other pre-honeyfarms appeared earlier (for example, at generations 749 and 1,159) and were subsequently destroyed.

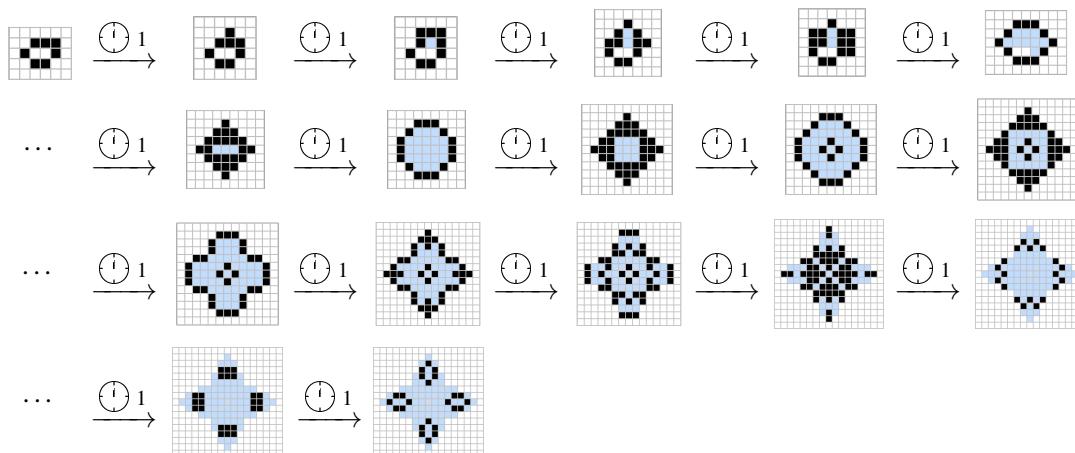


Figure 1.12: A *pre-honeyfarm* (top-left) takes 17 generations to evolve into an arrangement of four beehives called a *honeyfarm* (bottom-right).

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

⁵A *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, ..., 8 live cells are called *dominoes*, *triominoes*, *tetrominoes*, *pentominoes*, *hexominoes*, *heptominoes*, and *octominoes*.

light and the honeyfarm, 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.

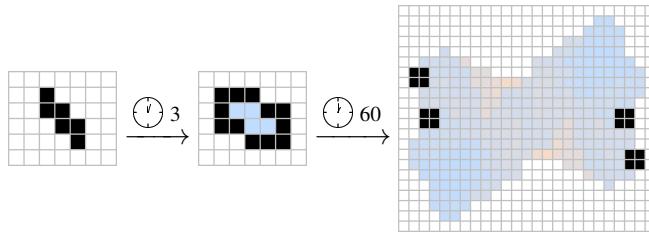


Figure 1.13: The *stairstep hexomino* (left) evolves into an arrangement of four blocks called the *blockade* in 63 generations. This evolution, and any of the intermediate patterns, are often called *lumps of muck*.

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 61 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 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 10.1.

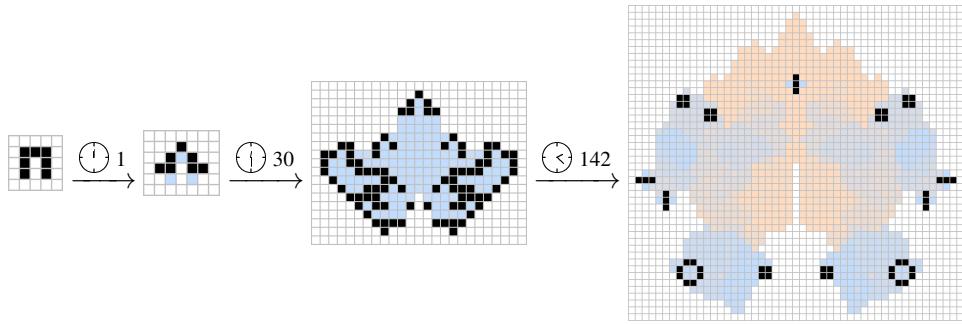


Figure 1.14: The *pi-heptomino* is a commonly-occurring unstable object that moves forward by 9 cells after 30 generations (compare the middle two patterns), but leaves behind debris that prevents it from moving ahead another 9 cells.

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 object⁶ 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.

⁶For example, it appears in generation 1,185 of the soup in Figure 1.5.

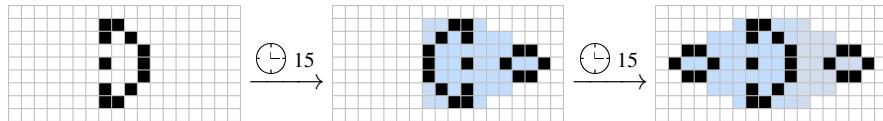


Figure 1.15: A queen bee (left) reflects itself and leaves a beehive behind every 15 generations. After doing this twice, the beehives start interfering with the queen bee, causing it to explode.

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 beehive⁷).

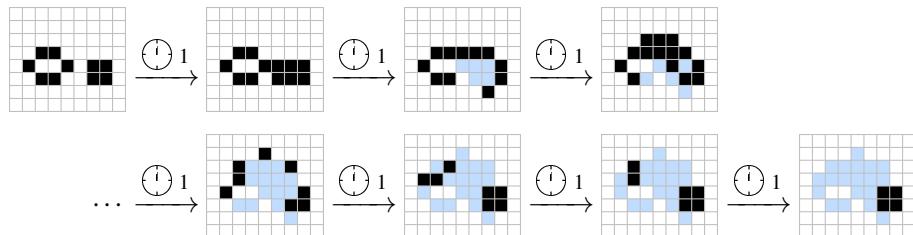


Figure 1.16: Placing a block next to a beehive destroys the beehive in 7 generations, without permanently damaging the block.

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 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.

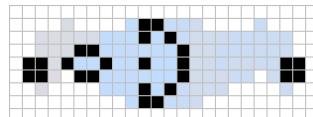


Figure 1.17: The *queen bee shuttle* is a period 30 oscillator constructed by combining the reactions in Figures 1.15 and 1.16.

The queen bee shuttle is our first example on 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.

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

⁷We will look at eaters in more depth in Section 2.3.

⁸The 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).

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

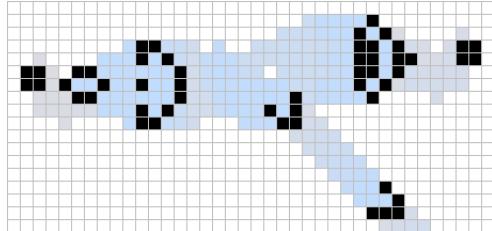


Figure 1.18: The *Gosper glider gun* creates a never-ending stream of gliders by bouncing two queen bees back and forth between two blocks (the object at the top-right is indeed a queen bee, just in a different phase than in the other figures).

1.4 The B-Heptomino and Twin Bees

Another commonly-occurring¹¹ 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 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 often 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.

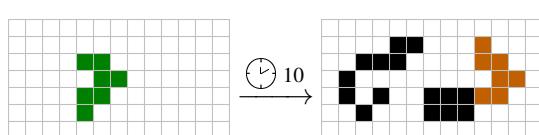


Figure 1.19: A *B-heptomino* (depicted in green on the left) takes 10 generations to move forward by 5 cells (in orange on the right) and create a bunch of junk behind it. The trailing junk subsequently destroys the *B-heptomino*, preventing it from traveling any farther.

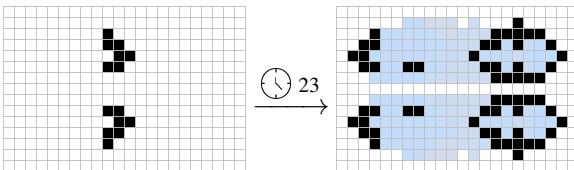


Figure 1.20: Twin bees reflect themselves and leave behind a chaotic mess every 23 generations (the two-cell objects in the middle of the image on the right die off in the next generation and thus have no effect).

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

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

¹⁰Whether 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.

¹¹For example, it occurs at generation 106 of the soup from Figure 1.4 and generation 210 of the soup from Figure 1.5.

¹²The 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”.

¹³This stabilization was found by Bill Gosper in 1971.

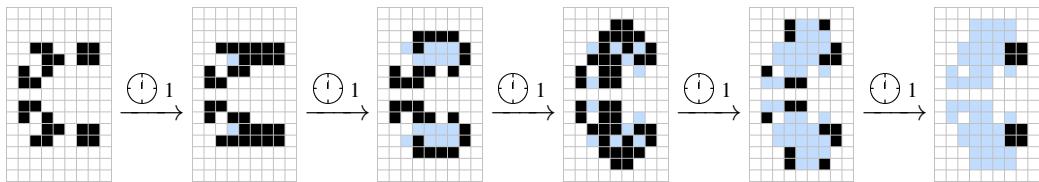


Figure 1.21: Placing two blocks next to the debris left behind by twin bees destroys that debris in 5 generations, while leaving the blocks intact.

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.

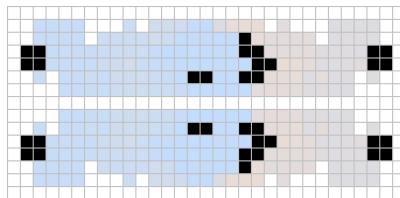


Figure 1.22: The *twin bees shuttle* is a period 46 oscillator constructed by combining the reactions in Figures 1.20 and 1.21.

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 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.

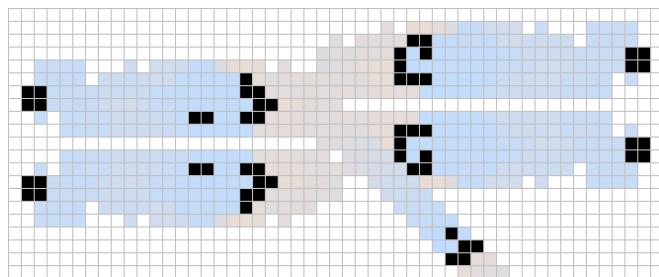


Figure 1.23: The *twin bees gun* is a period 46 glider gun that was constructed by bouncing two twin bees back and forth between each other and two stabilizing pairs of blocks. Note that the pair of objects in the middle-right are indeed twin bees, just in a different phase than the other figures.

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.

¹⁴This gun was also found by Bill Gosper.

¹⁵A 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.

Instead, the switch engine continually moves farther and farther away from where it started, just like a spaceship.

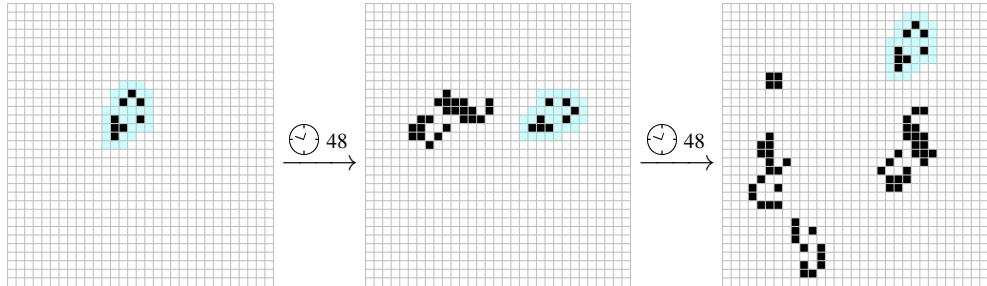


Figure 1.24: A switch engine (outlined in aqua) moves and reflects itself across the diagonal every 48 generations, but leaves behind some chaotic junk in the process. After 96 generations, the switch engine is back in its original orientation, but 8 cells northeast of where it started. The switch engine appears for the last time in generation 1,152, after repeating this entire process 12 times. The chaotic junk then finally collides with the switch engine and destroys it.

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, 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).

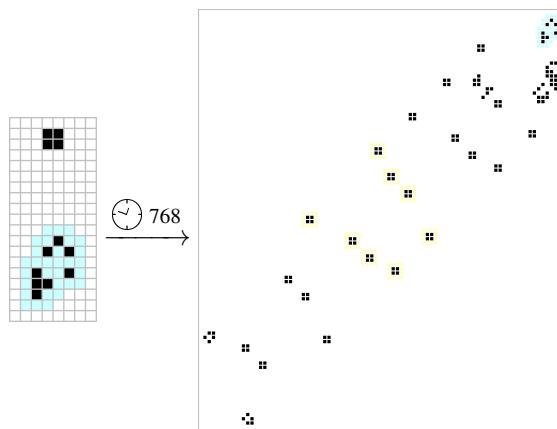


Figure 1.25: A switch engine (outlined in aqua) with a single block strategically placed next to it evolves into a *block-laying switch engine*—a pattern that repeatedly lays a configuration of 8 blocks (outlined in yellow) every 288 generations as it moves northeast.

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).

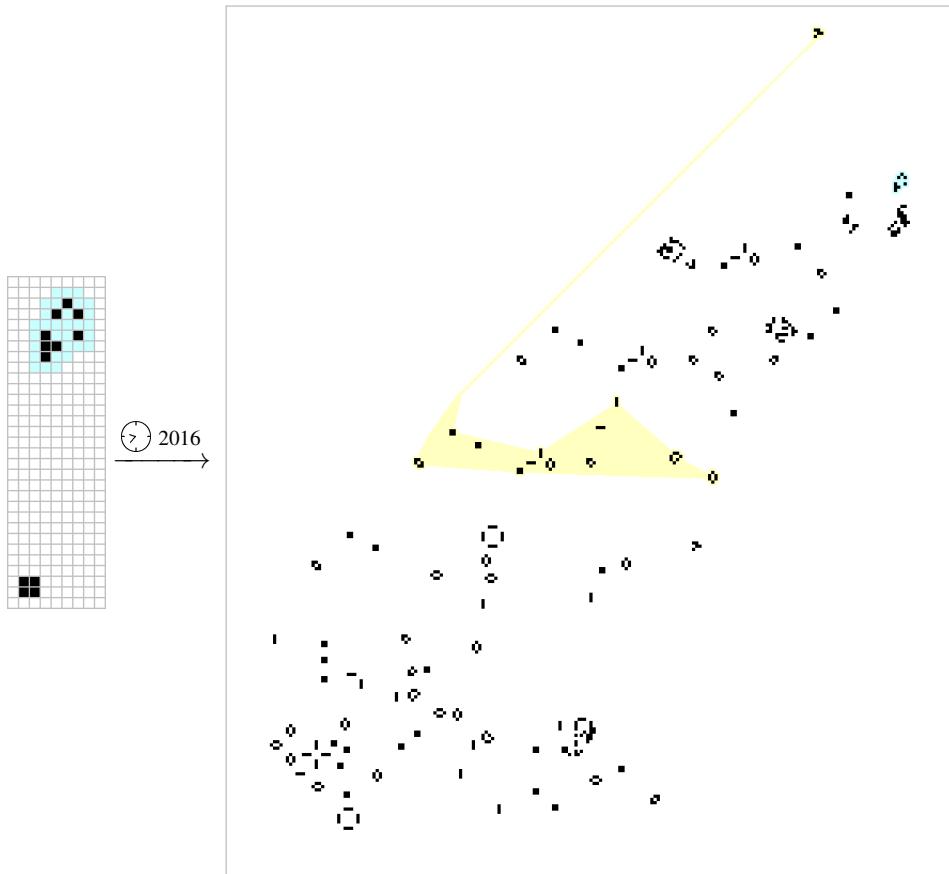


Figure 1.26: A switch engine (outlined in aqua) with a single block strategically placed next to it can also evolve into a *glider-producing switch engine*, which repeatedly lays a messy pattern of 4 blinkers, 8 still lifes, and one glider (outlined in yellow) every 384 generations, as it moves northeast.

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 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.

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

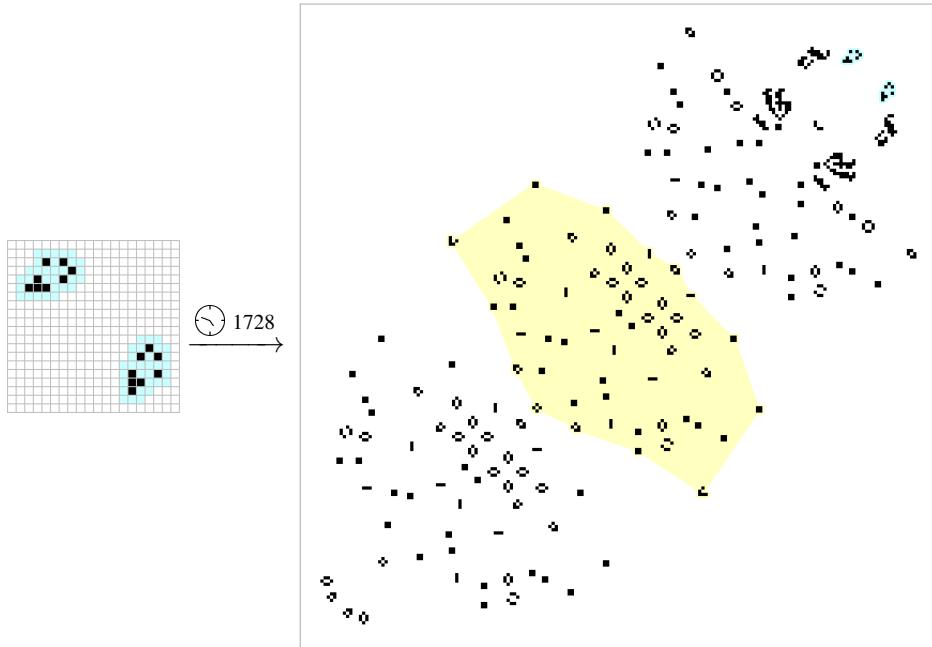


Figure 1.27: Two switch engines (outlined in aqua) can be used to stabilize each other and create an ark. In this particular ark, the switch engines leave behind two gliders, two toads, eight blinkers, and 42 still lifes (outlined in yellow) every 576 generations.

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 2,901 generations to stabilize, respectively. More extreme than either of these examples is the 8-cell switch engine, which takes 3,911 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*, 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 within 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 Life²⁰ in generation 69.²¹

We could similarly ask what the longest-lived pattern with n cells for $n = 6, 7, 8, \dots$ are. The

¹⁷We 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.

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

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

²⁰First noticed by Richard K. Guy in 1970.

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

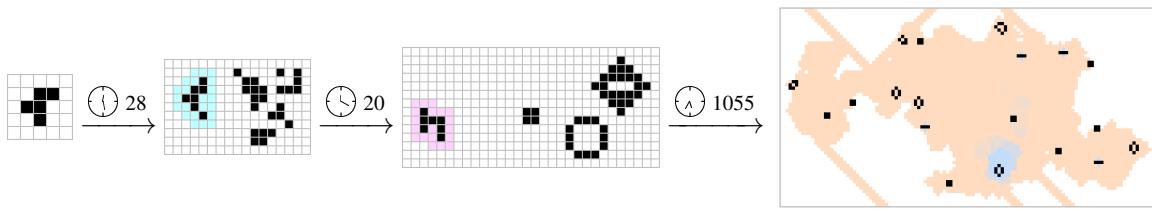


Figure 1.28: An R-pentomino takes 1,103 generations to stabilize—much longer than any other polyomino with 5 or fewer cells. Along the way, it produces numerous other important patterns such as the *B-heptaplet* highlighted in aqua and the *Herschel* highlighted in magenta.

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 $n \leq 9$;²² the longest-lasting known 10-cell methuselah in a small bounding box was not found until 2019.

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.

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

²²Exhaustive 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.

the one displayed in Exercise 1.15). 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 47,575 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 engines²⁴ 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 define methuselahs and what it means for a pattern to stabilize. As yet another example, consider the remarkable 16-cell pattern²⁵ depicted in Figure 1.30, which takes 736,692 generations to stabilize.

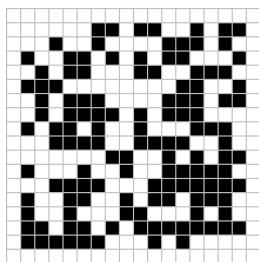


Figure 1.29: A methuselah that fits within a 16×16 bounding box and takes 47,575 generations to stabilize.

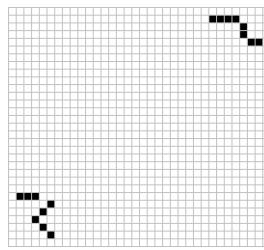


Figure 1.30: A 16-cell pattern that takes a whopping 736,692 generations to stabilize. The 8-cell objects at the top-right and bottom-left individually are just predecessors of the switch engine, so this pattern is an ark.

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 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.

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

²³Found by Adam P. Goucher in February 2019.

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

²⁵Found by Nick Gotts in 2005.

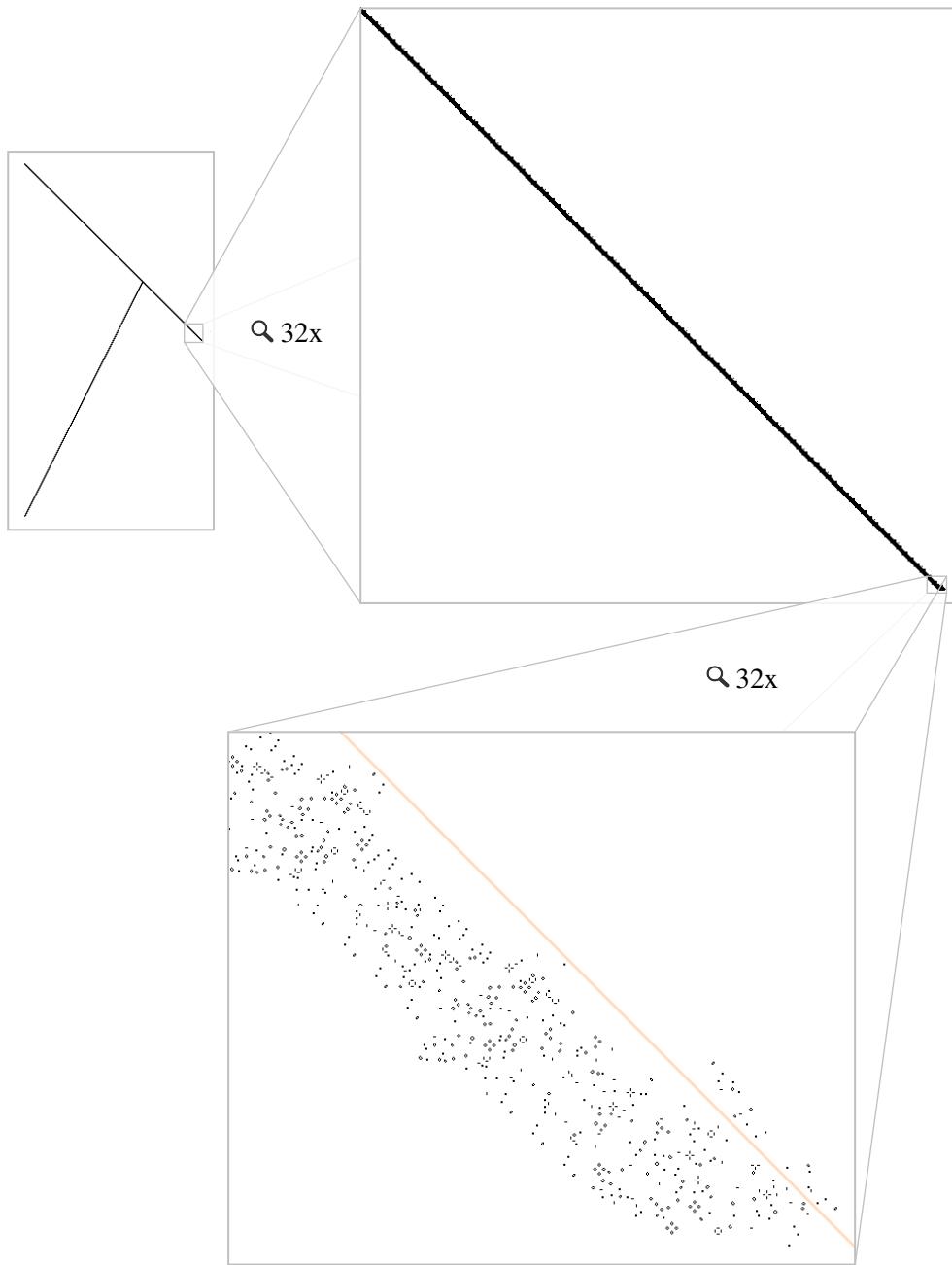


Figure 1.31: The pattern from Figure 1.30 leaves behind this absolutely massive ash after 736,692 generations. We have zoomed in by a factor of $32 \times 32 = 1,024$ on the corner of the ash that the pattern started growing from, which is also the corner that contains the debris that gliders are fired 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.

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 show that there exist patterns for which it is not *possible* to know, even in principle, whether or not they stabilize. In other words, there exist patterns with the unsettling property that, no matter how long we run them, and no matter how clever we are, there is absolutely no mathematical way to know if they will eventually stabilize.

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.

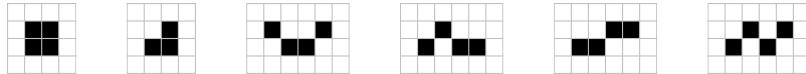


Figure 1.32: Six small parents of the block. From left-to-right, the first three of these objects are the block, pre-block, and grin, respectively, while the other three do not have names.

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 period 2, yet appears 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 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 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.

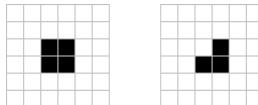


Figure 1.33: Two 6×6 tiles that evolve the same way no matter what is placed outside of the tiles.

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

²⁶Strictly 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.

²⁷Generation 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.

²⁸This 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.

²⁹We 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.

possible patterns that can be present in generation 1, we only need to consider patterns made up of one of $2^{36} - 1$ (not 2^{36}) different tiles in generation 0.

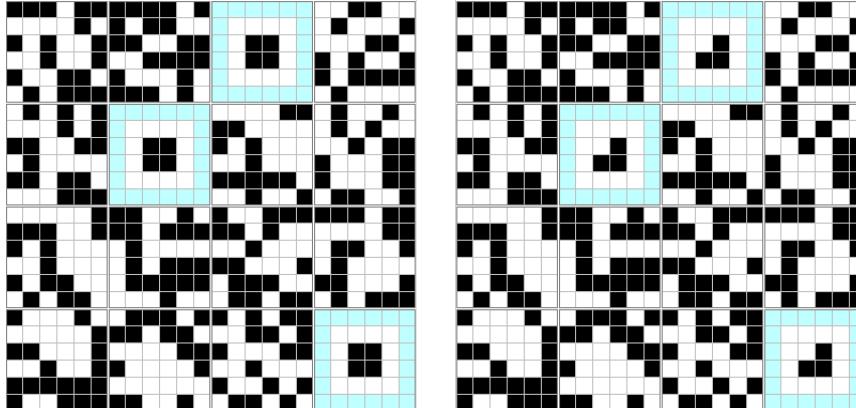


Figure 1.34: A $6n \times 6n$ pattern in the $n = 4$ case, broken down into 6×6 tiles. Each of the tiles containing only a block in the center (outlined in aqua) can be replaced by a tile with a pre-block in the center without affecting the evolution of the overall 24×24 pattern.

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 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 1.14.

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 $2^{n^2}n \times n$ 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.

³⁰In 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 Edens.

³¹This 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.

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

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

- 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 seem not 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.

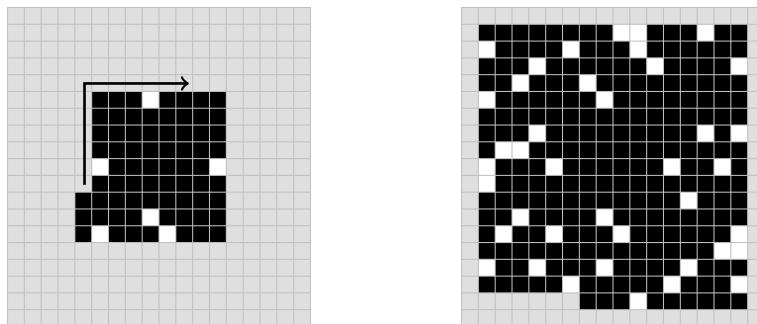


Figure 1.35: A partial Garden of Eden (left) that is being constructed one cell at a time by placing cells in a clockwise spiral, choosing whether they should be alive or dead based on which state results in fewer parents. The completed Garden of Eden (right) has 41 dead cells and 225 alive cells (the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

The pattern that we have constructed is interesting in that it is not only a Garden of Eden, but it remains a Garden of Eden regardless of what 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, but it is currently unknown whether or not there exists a Garden of Eden that does not contain an orphan.

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 1.36.³⁶

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.³⁷ In a similar vein, we now show that there does not exist a Garden of Eden that has a bounding box with height one.³⁸

³⁴In 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.

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

³⁶These patterns can be verified to be orphans via a computer program called *JavaLifeSearch*—see conwaylife.com/wiki/JavaLifeSearch and Exercise 1.19.

³⁷This computation was carried out independently by Steven Eker and Marijn Heule in 2016.

³⁸This theorem was originally proved by Jean Hardouin-Duparc in 1974 [HD74].

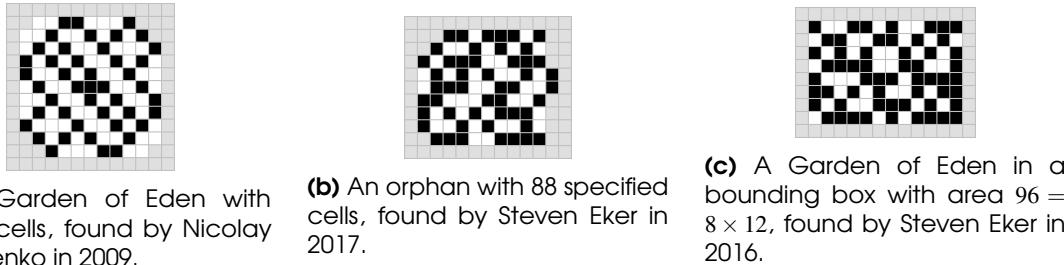


Figure 1.36: The current smallest-known Gardens of Eden by (a) number of live cells, (b) total number of live and dead cells specified in the orphan, and (c) bounding box area.

Theorem 1.2 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 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.

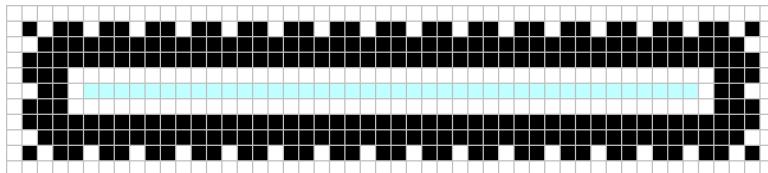


Figure 1.37: A border that can be used to help construct parents of 1-cell-thick patterns. No matter what we place in the central $3 \times n$ region, no cell outside of the $3 \times n$ region will be alive in the next generation. We will fill it in such a way so that the desired $1 \times (n - 2)$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

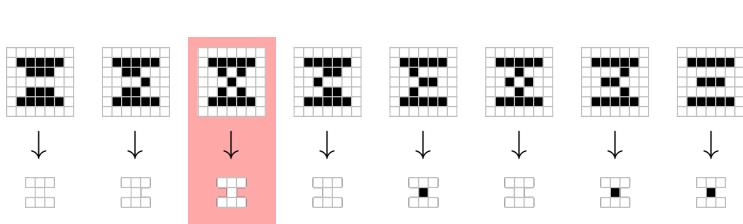


Figure 1.38: All except for one of the 8 different 3×3 squares in which the top and bottom rows are the negation of the middle row evolve so that the top and bottom rows die and the middle cell stays the same.

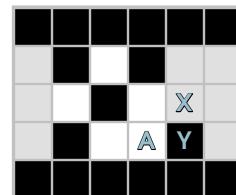


Figure 1.39: In order to correct the problematic 3×3 block from Figure 1.38, we set cell A to be dead and cell Y to be alive, regardless of the state of cell X.

To fix 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 Y

in that figure) to be alive, regardless of whether cell X is alive or dead. It is then straightforward to verify that the resulting pattern will evolve into the $1 \times (n - 2)$ pattern that we desire: all that needs to be checked is that the central cell in this 3×3 block now stays alive (since it now has exactly 3 live neighbors) and that moving the alive cell from A to Y does not affect the evolution of any of the other 3×3 blocks to its right (which can be done via a simple case analysis). ■

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.

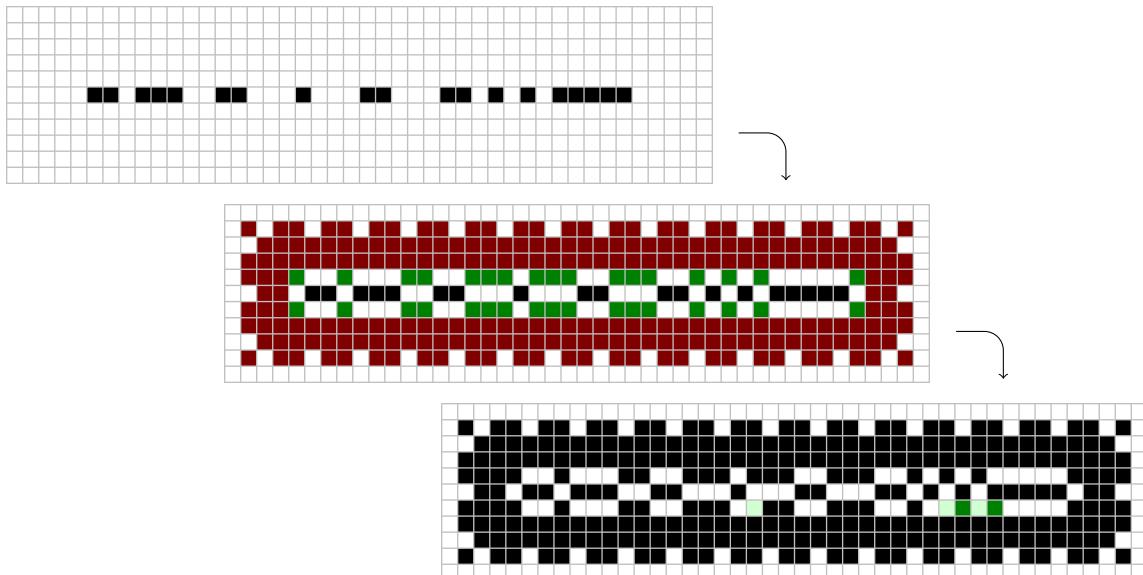


Figure 1.40: How to use the proof of Theorem 1.2 to construct a parent (at the bottom) of a pattern with height 1 (at the top). First, we place the border around the pattern (displayed in dark red in the middle image) and place the pattern's negation in the top and bottom rows of the box (displayed in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

More sophisticated techniques have been used to prove that there similarly do not exist orphans whose bounding boxes are 2 or 3 cells high.³⁹ On the other hand, there does exist an orphan (and thus a Garden of Eden) with height 5, as shown in Figure 1.41.⁴⁰ The question of whether or not there exists an orphan whose bounding box is 4 cells high remains open.

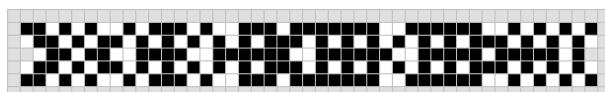


Figure 1.41: A Garden of Eden and orphan that fits within a 5×45 bounding box. No Garden of Eden with height less than 5 is currently known.

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

³⁹This was proved by Steven Eker in 2016, using the same computer-assisted methods introduced by Jean Hardouin-Duparc.

⁴⁰This orphan was also found by Steven Eker in 2016.

has the remarkable property that it has 17,920 distinct parents, every single one of which is a Garden of Eden—one of these parents is displayed in Figure 1.43.⁴¹

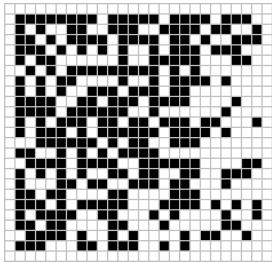


Figure 1.42: A pattern that has a parent but no grandparent.

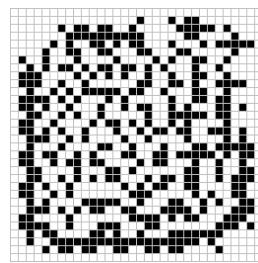


Figure 1.43: A parent of the grandparentless pattern from Figure 1.42.

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.

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 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

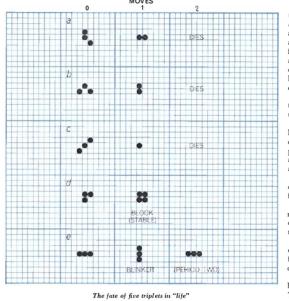
⁴¹Whether or not such a pattern exists was originally asked by Conway in October 1972. It was not resolved until May 2016, when the pattern displayed in Figure 1.42 was found by user “mtve” on the ConwayLife.com forums. Later that month, they even found a pattern that has a grandparent but no great-grandparent and a pattern with a great-grandparent but no great-great-grandparent, though the question remains open of whether or not, for every integer $n \geq 1$, there exists a pattern with a great n -grandparent but no great $^{n+1}$ -grandparent.

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 Cambridge and Caltech College at the University of California, has been in pure mathematics. For instance, in 1967 he discovered new groups—what it "Conway's groups"—that included all but two of the then known sporadic groups. (The two missing ones were found 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 has been mentioned here several other times.



The fate of five triplets in "Life".
© 1970 SCIENTIFIC AMERICAN INC.

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

This month we consider Conway's latest brainchild, a fantastic solitaire pastime he calls "Life." Because of its simplicity and the many possible configurations of a society of living organisms, it belongs to a growing class of what are called "cellular automata," or systems that resemble real-life processes. To play Life you must have a fairly large rectangular board composed of a grid of flat counters of two colors. (Small checkers or pebbles will do.) A rectangular board can be used if you can find flat counters that are small enough to fit together without leaving gaps (but not because they are not flat). It is possible to work with pencil and graph paper but this is not nearly so satisfying as using regular, true counters and a board.

The basic idea is to start with a simple configuration of counters and, through a series of steps, to observe what happens as you apply Conway's "game laws."

Mrs. Perle Quigley, a deceased colleague discussed in "Mathematical Games"

in a previous column, once asked Conway

to explain his rules carefully, after a long period of experimentation. He met three deadlocks:

1. There should be no initial pattern for which the population grows without limit.

2. There should be initial patterns that grow and then die out.

3. There should be simple initial patterns that grow and change for a considerable time without eventually leading to an end in these possible ways: fading away, oscillating, or stabilizing (that is, from becoming too sparse), settling into a stable configuration that remains unchanged, or entering a repeating cycle, oscillating in phase in which they repeat an endless cycle of two or more periods.

These three genetic laws are delightfully simple. First note that each cell of the checkerboard (assumed to be the infinite plane) has eight neighbors, including four adjacent orthogonally, four adjacent diagonally. There are two kinds of neighbors:

1. Sane: Each counter with two or three neighboring counters survives for the next generation.

2. Deaf: Each counter with four or more neighbors dies (is removed) from the system unless it is a corner, edge, or side neighbor or one dies from isolation.

3. Births: Each empty cell adjacent to exactly three live neighbors becomes, 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 generation.

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

0	00000	00000	00000	0	300000	0	000000
0	1	000	0	0	0	00000	00000
0	1	000	0	0	0	00000	00000
0	0	0	0000	0	0	00000	00000
0	0	0	0000	0	0	00000	00000

NUMBER 1 MARCH 1971

Editor and Publisher — Robert T. Wainwright

What you see now is nothing 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 space for news items. But many people are interested in developments still occurring. Many readers (the writer included) have expressed an interest to have some means by which they may continue to exchange news developments. My own prior investment of time and effort in writing this newsletter, however, makes it difficult to 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 newsletter 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 to mail to me, Robert T. Wainwright, 1000 Second Avenue, New York, N.Y. 10020. 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 of the R-pentominoes, all 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 and then a pentomino and so on. This has recently 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.

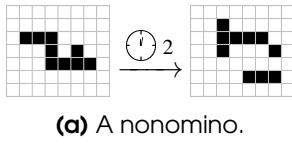
Conway's seven "unknown" heptominoes							
B	C	D	E	F	H	I	
0 00	000	0	000	00	00	00	
000	0	000	000	0	0	000	
0	0	0	00	000	0	000	
				000	000	000	
				0	000	000	0

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—its history is known. This was confirmed by Mr. Hugh H. Neumann of Lorain City, New York.

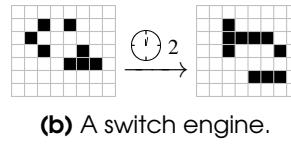
Figure 1.45: The front page of the first issue of *Lifeline*, a newsletter for Life enthusiasts that ran from March 1971 to September 1973.

in the T-tetromino, R-pentomino, B-heptomino, staircase 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.



(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.

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, 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 catalogued 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 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 catalogued 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*,⁴² which is still active today⁴³ 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 six years since it began, it has catalogued more than 1,000 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.

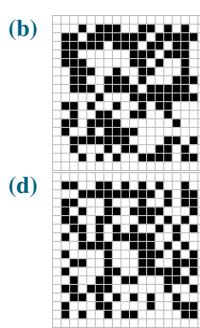
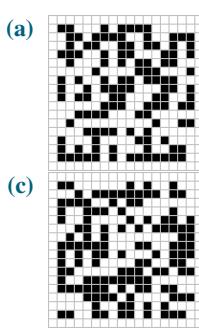
Search Name	Year(s)	Ash Objects Found	Notes
Flammenkamp	1994	5.0×10^9	first automated soup search
Flammenkamp	2004	5.0×10^{10}	—
Okrasinski	2003–2008	4.7×10^{11}	also found methuselahs
TOLLCASS	2009–2011	1.7×10^{11}	online search, multiple CA
apgsearch	2014–present	7.2×10^{14}	online search, multiple CA

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

Exercises

solutions to starred exercises on page 301

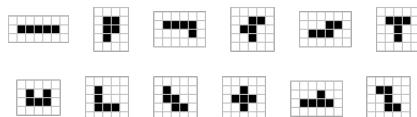
- * **1.1** Evolve each of the following randomly-generated 20×20 soups and describe the most unusual object that forms.



- 1.2** What is the longest lifespan of a pattern with:

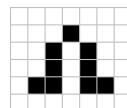
- (a) 1 or 2 live cells?
- (b) 3 live cells?
- (c) 4 live cells? [Hint: You could try writing a computer program to help you.]

- 1.3** 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) 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) The still life below is called a *hat*. Use a hat to stabilize one side of the twin bees shuttle.



⁴²Officially, the “apg” in *apgsearch* are not its author’s initials, but rather stand for **a**sh **p**attern **g**enerator.

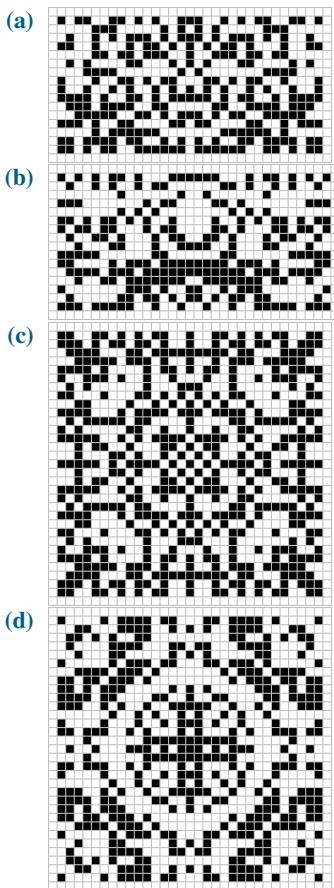
⁴³It is available at catagolue.appspot.com.

***1.5** Find a specific generation at which each of the following unstable objects appears in the evolution of the R-pentomino.

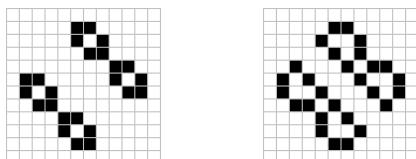
- (a) A T-tetromino.
- (b) A pre-honeyfarm.
- (c) A pi-heptomino.
- (d) A queen bee.
- (e) Lumps of muck (in particular, generation 3 of the stairstep hexomino).

1.6 Use two blocks and three queen bees to create a “double” Gosper glider gun: a gun that emits two streams of gliders.

***1.7** 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.



***1.8** 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*.

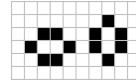


⁴⁴Its 1-generation successor, which is called *rabbits*, was found by Andrew Trevorrow in 1986. This version was subsequently found independently by Robert Wainwright and Andrew Trevorrow.

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

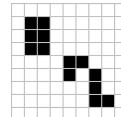
- (b) There are at least two different 8-cell objects that evolve into a bakery. Find one of them. [Hint: Try evolving random soups until you find a bakery, or write a computer program that evolves many different 8-cell objects.]

***1.9** This problem concerns the configuration of two beehives displayed below.



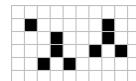
- (a) Evolve the pattern until it stabilizes. What happens to the beehives?
- (b) Create a period 30 oscillator that uses this reaction to stabilize two queen bees that are rotated 90 degrees from each other.

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



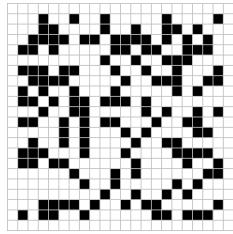
- (a) Evolve the pattern until it stabilizes. What happens to the pre-beehive and eater 1?
- (b) Evolve the pre-beehive and eater 1 individually (i.e., not next to each other). Describe what happens to them.
- (c) Evolve a queen bee until you find a pre-beehive that it leaves behind.
- (d) Use an eater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

1.11 The 9-cell methuselah displayed below is called *bunnies*.⁴⁴



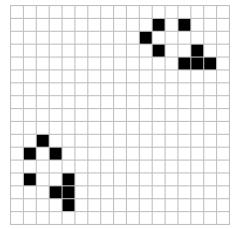
- (a) What is its lifespan?
- (b) It eventually evolves in the same way as each of bunnies 9 and bunnies 10b from Table 1.1. Find the first generation of each object’s evolution where they match.

1.12 The methuselah displayed below is named *Edna*.⁴⁵



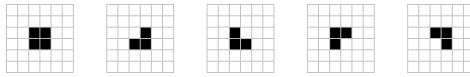
- (a) What is its lifespan?
- (b) What objects can be found in its ash after this pattern stabilizes? Are there any objects that we did not give names to in this chapter?

1.13 An ark called *Noah's ark* is displayed below.⁴⁶ What is the period of this pattern, and what objects does it leave behind as it moves?



***1.14** 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?

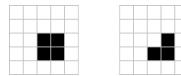
⁴⁵Found by Erik de Neve in January 2010.

⁴⁶This 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.

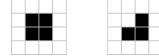
⁴⁷Found by Nick Gotts in 2019.

⁴⁸This pattern was found by Nick Gotts in February 2005.

⁴⁹Created by Karel Suhajda, based on earlier programs by David Bell, Dean Hickerson, and Jason Summers.



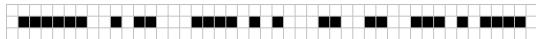
(d) Why can't we replace the 6×6 or 5×5 tiles by the 4×4 tiles displayed below? What part of the proof breaks down?



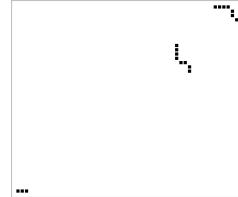
1.15 An 8-cell methuselah with a longer lifespan than the 8-cell methuselah from Table 1.1 (but with a significantly larger bounding box) is displayed below.⁴⁷ What is its lifespan?



1.16 Find a parent of the following 1-cell-thick pattern:



***1.17** The 19-cell pattern displayed below consists of two switch engine predecessors and a blinker,⁴⁸ and takes a staggering 6,526,589 generations to stabilize.



- (a) Evolve the pattern to see how it behaves. What effect does the blinker have that makes 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 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

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

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*⁴⁹ 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) Use JavaLifeSearch to find a predecessor of the following pattern:



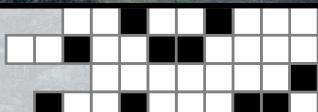
- (b) Use JavaLifeSearch to verify that the patterns from Figure 1.36 really are orphans.
(c) Use JavaLifeSearch to verify that the pattern from Figure 1.42 really does not have any grandparents.

Early draft (May 19, 2020).

Not for public dissemination.



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.

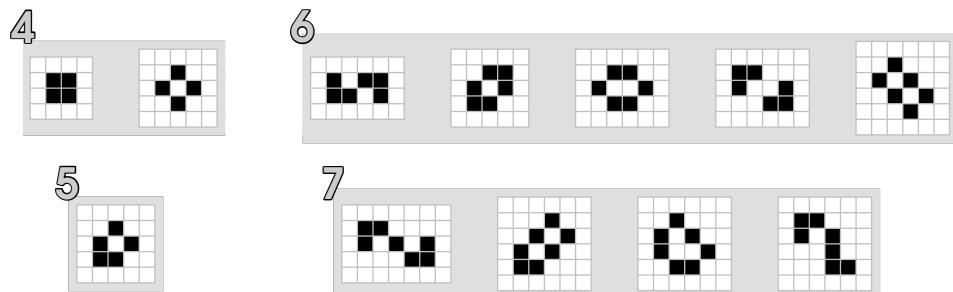


Figure 2.1: All still lifes with 7 or fewer live cells, arranged by their cell count. From left to right: (4 cells) block, tub, (5 cells) boat, (6 cells), snake, ship, beehive, aircraft carrier, barge, (7 cells) long snake, long boat, loaf, and eater 1 (which is sometimes called *fishhook*).

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, 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, ... 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.

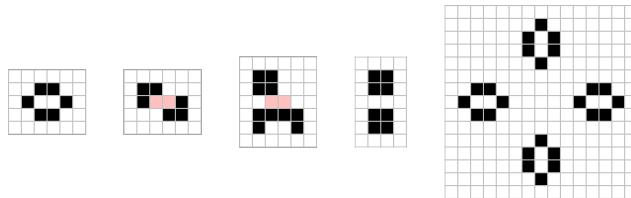


Figure 2.2: From left to right, these still lifes are the beehive, aircraft carrier, *block on table*, *bi-block*, and honey farm. The beehive is a strict still life because it is connected, the aircraft carrier is a strict still life because its two halves are not still lifes by themselves but together 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 honeyfarm 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 honeyfarm 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 subpatterns² (see Figure 2.3 for some examples).

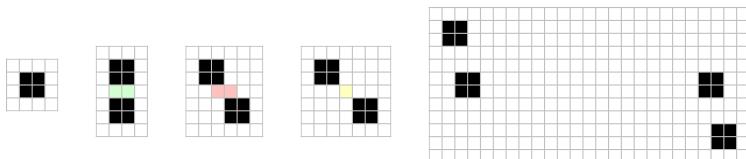


Figure 2.3: From left to right, these are the block, bi-block, two nameless configurations of two blocks, and the blockade. The bi-block is a pseudo still life because of the cells highlighted in green between the blocks that have more than 3 neighbors. The middle configuration is not a still life at 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).

¹In 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].

²This final restriction is made to capture the notion of the individual connected components “almost touching”.

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	1,353	   	620	
16	3,286	  	1,645	
17	7,773	  	4,067	
18	19,044	  	10,843	
19	45,759	  	27,250	
20	112,243	  	70,637	
21	273,188	  	179,011	
22	672,172	  	462,086	
23	1,646,147	  	1,184,882	

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

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].

Given that the definition of a pseudo still lifes 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.



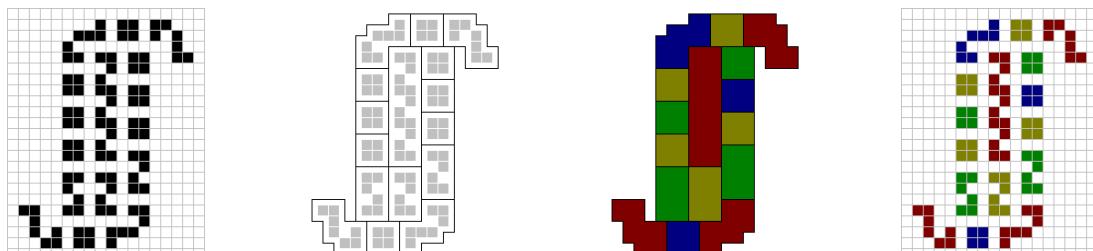
(a) A pseudo still life that can be partitioned into 3 still lifes, but not 2.
 (b) A pseudo still life that can be partitioned into 4 still lifes, but not 2 or 3.

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

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 theorem⁵ 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



(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.

³For 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/life.html.

⁴The 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.

⁵The 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.

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

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 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.

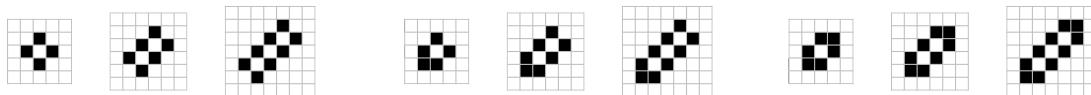


Figure 2.6: From left to right, these still lifes are the tub, barge, *long barge*, boat, long boat, *long long boat*, ship, *long ship*, and *long long ship*. Each of these still lifes can be made as long as we like by continuing the patterns in the obvious way.

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

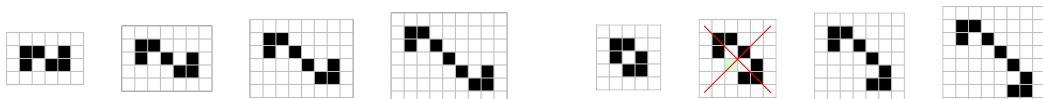


Figure 2.7: From left to right, these objects are the snake, long snake, *long long snake*, *long³ snake*, ship, an object that is not a still life due to the green cell coming to life, *canoe*, and *long canoe*. Again, each of these still lifes can be made as long as we like by continuing the patterns.

In all of these examples, the still lifes are made up from 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 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

⁷These 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 = longⁿ = veryⁿ⁻¹ long = extraⁿ⁻² long. For example, a long³ snake is a long long long snake is a very very long snake is an extra long snake.

⁸Some 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 there 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*.

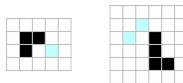


Figure 2.8: A pre-block (left) and a tail (right) are two common ways of stabilizing objects (typically when the cells highlighted in aqua are alive).

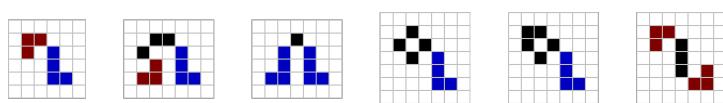


Figure 2.9: Several examples of still lifes that use pre-blocks (highlighted in red) and tails (highlighted in blue) to stabilize their ends. Note that the tails in the 4th and 5th still lifes are not actually required for stability, but just serve to make the still lifes larger.

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

This tells us that still lifes never have “thick” sections—they are made up of single-cell-thick “paths” of live cells that potentially curve and loop around on themselves, possibly plus some isolated blocks. Furthermore, it is often¹⁰ 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).

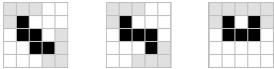


Figure 2.10: Three path segments that cause overcrowding of the central cell and thus must be avoided in still lifes.

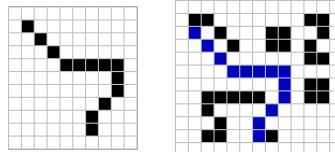


Figure 2.11: A single-cell-thick path of live cells (left) can typically be stabilized by adding other nearby objects and branching paths (right).

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 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).

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*:

⁹The 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.

¹⁰But not always—see Exercise 2.12.

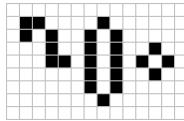


Figure 2.12: A tub, a boat, a loaf, a still life with a tail, or any other “pointy” still life can be used as an induction coil to prevent the birth of 3 orthogonally connected cells.

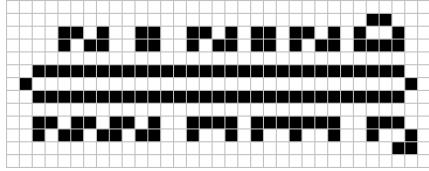


Figure 2.13: A demonstration of several induction coils being used to stabilize an object. The induction coil at the top-right corner is the cap, in the bottom-middle are a table and a way of extending it, and at the bottom-right is a bookend.

still lifes¹¹ 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 (we thus say that it has a *recovery time* of 4 generations), making it the fastest-recovering known glider eater.¹³

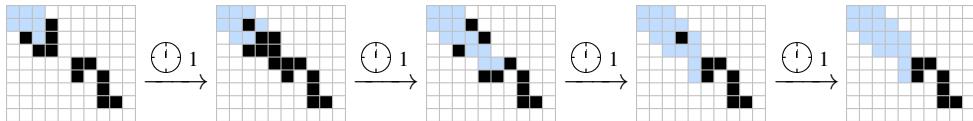


Figure 2.14: Eater 1 is a seven-cell still life that takes 4 generations to eat a glider.

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 unaffected by so many different types of debris hitting its top-left corner make 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.

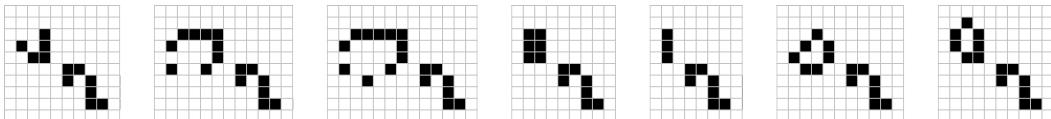


Figure 2.15: Eater 1 can be used to eat a glider, a lightweight spaceship, a middleweight spaceship, a pre-beehive, and many other objects.

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 eater¹⁴ with a recovery time of 6 generations that is made up of two still lifes and is capable of eating gliders

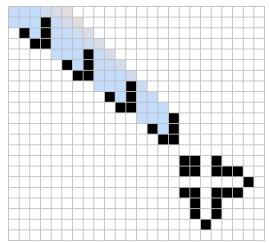
¹¹Strictly speaking, an eater does not need to be a still life, but still life eaters are used so much more frequently than other types of eaters that it is often assumed.

¹²Eater 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.

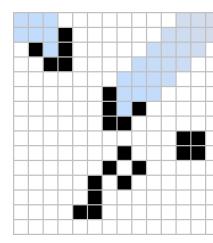
¹³There are also other glider eaters that tie its recovery time of 4 generations—see Exercise 2.13.

¹⁴There 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 just introduce these others as we need them.

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 then return to their initial state after a few generations. 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 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.

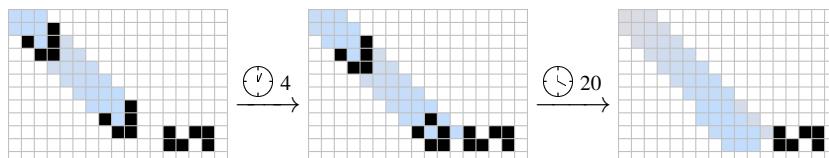


Figure 2.17: A *boat-bit* is a reaction in which a snake (or any other still life containing a pre-block) is used to turn a glider into a boat which then destroys a second glider coming in from the same position. Furthermore, the snake is not even temporarily disturbed at any point throughout this reaction.

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.

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.

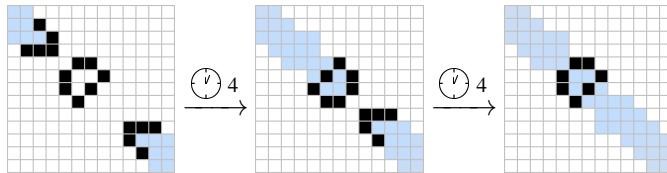


Figure 2.18: A glider can be used to flip a loaf. A loaf can thus be used to eat two gliders coming from opposite directions.

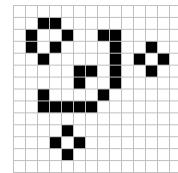


Figure 2.19: Eater 3 is based on the loaf-flipping reaction.

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

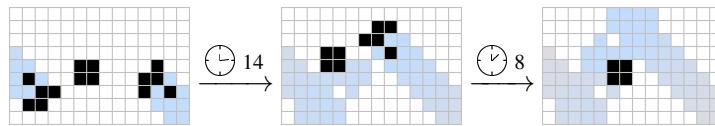
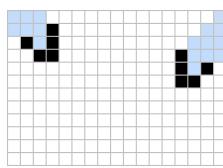


Figure 2.20: A *(2,1) block pull* is a reaction in which a glider pulls a block by 2 cells in one direction and 1 cell in the other. A block can thus be used to eat two gliders coming from opposite directions.

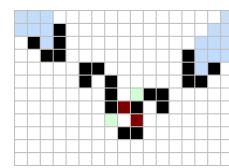
2.4 Welded and Constrained Still Lifes

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 position 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 easter 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.



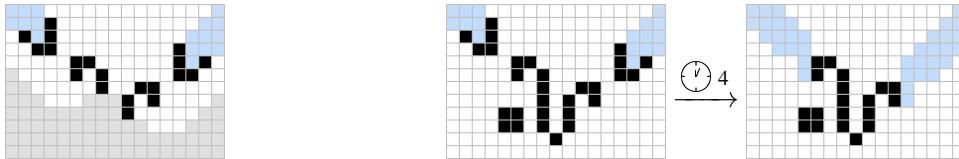
(b) Placing two individual eaters does not work.

Figure 2.21: The two gliders in (a) cannot be eaten by individual easter 1s, since they are too close together and the easter 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 easter 2 or easter 5 as well.

To get around this problem, we combine two easter 1s into a single, larger easter. The key idea is that the only part of easter 1 that is actually involved in the glider eating reaction from Figure 2.14

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

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 and trial-and-error.



(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 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.

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)).



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

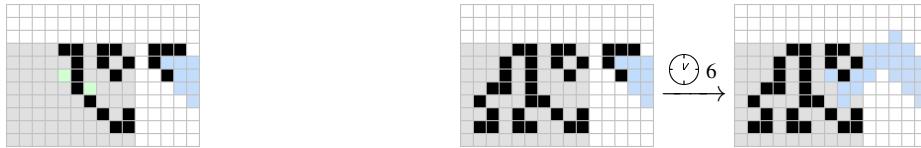
(b) An eater 5 almost works, but sticks just a bit outside of the light gray region.

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 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.

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



(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).

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.

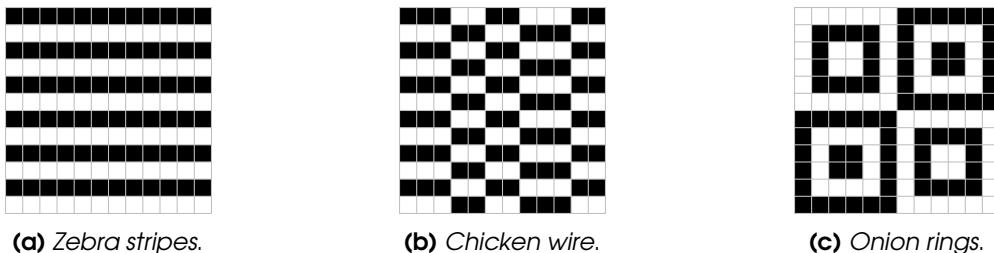


Figure 2.25: When the plane is tiled with these patterns, they create infinite still lifes with density 1/2.

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 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 $n \rightarrow \infty$ 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

¹⁷The 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].

¹⁸The 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 [CSdIB09].

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

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

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

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

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.

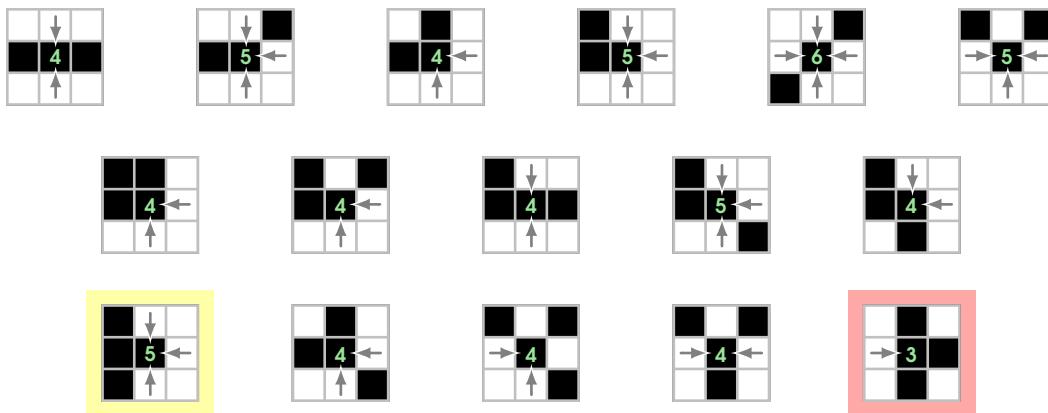


Figure 2.27: All 16 possible orientations of 2 or 3 live neighbors around a live cell, up to rotation and reflection. The dark gray arrows indicate which of its neighboring dead cells give tokens to it. The number of tokens that end up on the central live cell is indicated in green, and is always 2 more 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 cells 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 a birth—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 the 3 cells to its immediate left must be dead so as to avoid it dying—in other words, this cell is the 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. ■

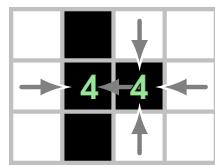


Figure 2.28: To fix the configuration from Figure 2.27 that results in a live cell only having 3 tokens, we transfer an extra token from a bordering live cell that has 5.

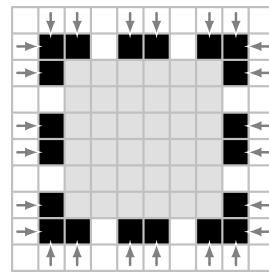


Figure 2.29: This border configuration is the one that causes the largest number of tokens to enter the $n \times n$ square: $4\lceil 2n/3 \rceil$ (the $n = 8$ case is illustrated here, so $4\lceil 16/3 \rceil = 24$ tokens enter the square region).

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:

Proposition 2.3 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, 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 $n = 2, 3, 4, \dots, 10$.

n	Densest Still Life	Prop. 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 Proposition 2.3. The examples displayed here are only unique when $n = 2, 3, 5$, or 7 .

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 used¹⁹ 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 Proposition 2.3 is not too far from optimal: the $n^2/2$ term is right, and the linear term in our bound is $2n/3 \approx 0.6667n$,

¹⁹These 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].

versus the following exact result which has a linear term of $17n/27 \approx 0.6296n$.

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 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

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

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 Elkes 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 carry over in a straightforward way to oscillators.

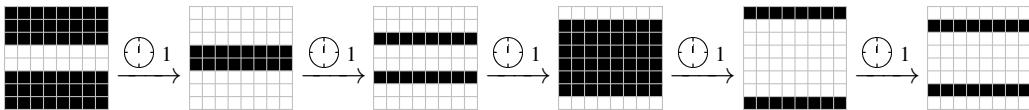


Figure 2.30: A period 6 infinite oscillator that has density $3/4$ in two of its phases. However, its average density over all of its phases is $(3/4 + 1/4 + 1/4 + 3/4 + 1/4 + 1/4)/6 = 5/12 \leq 1/2$.

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.

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 can supply me with the following information: Objects previously mentioned in either Scientific American or LIFELINE, given by their name while the new ones (each of which were sent in by a different referee) are referred to only by size. Regarding cataloguing of this class, see the Still Lifes Catalogue, elaborately recorded in color over 300 such objects including those shown below. The M.I.T. group have discovered some very unusual properties of these objects. One of these, the 13-cell still life which they call the 'eater' will be discussed in several other parts of this issue.

Size	The smallest still life objects showing families of similar structure				Sum	
6 ₃					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) (8.3)	tub w/tail, shuttleagh, (8.4)	9
9			(9.1) (9.2) (9.3)		hat, (9.4), (9.5), (9.6)	7
10+21	4+21	4+21	5+21	6+1	8+1	14 = 0, 1, 2, ...
The remaining unpublished smallest still lifes						
7.1	8.1	8.2	8.3	8.4		
8.3, 8.2	8.2	8.3	8.3	8.3		
9.1	9.2	9.3	9.4	9.5	9.6	
9.5	9.6	9.6	9.6	9.6	9.6	

Class II.A (all period two oscillators) is sufficiently large and common to be further divided into subgroups by symmetry of the active area (period 2 length of 2). For this class, I have chosen to list each group, or explaining what active area symmetry means (i.e., cells in both phases or one phase). I have chosen the more unusual cases, and omitted the more common ones. Since 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.

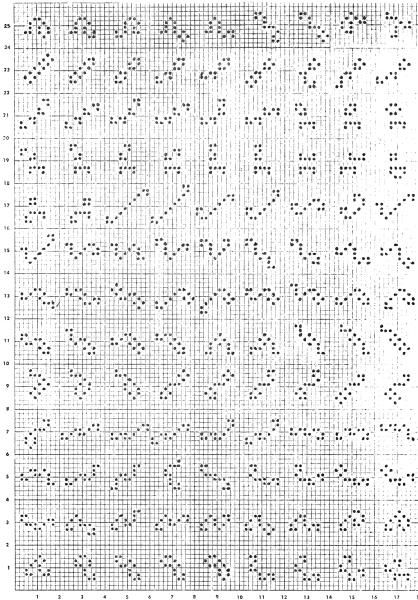


Figure 2.32: A summary of all 121 distinct 12-cell strict still lifes, compiled by hand by Douglas Petrie and Everett Boyer in 1973. Originally published in *Lifeline* vol. 10 in June 1973.

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.

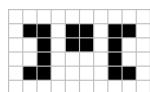


Figure 2.33: A strict still life that would never be found via a greedy still life search, since a block on table would be found before adding the second table.

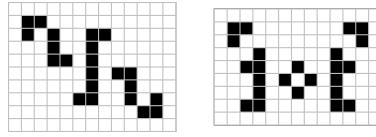


Figure 2.34: Two more strict still lifes that won't be found by a greedy still life search, and had to be added by hand when cataloguing all still lifes with 22–24 cells.

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 some still lifes that use a single cell to stabilize a long line of orthogonally-connected cells, like the one in Figure 2.34. These still lifes had to be added back into the count by hand, and they were the

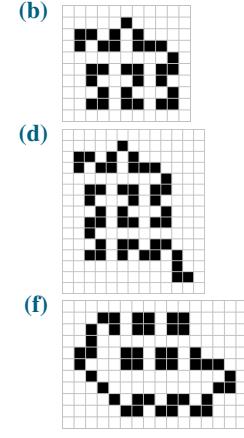
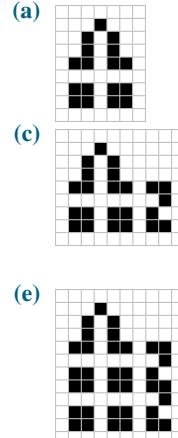
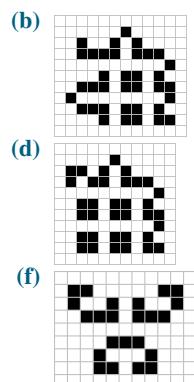
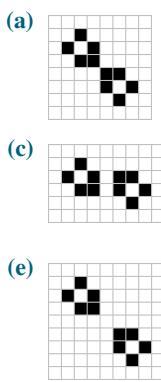
main reason why he did not extend his search to 25-cell still lifes—the number of exceptional still lifes that would need to be added back in by hand is too large, and could potentially lead to errors.²⁰

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.²¹ 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 302

- *2.1 Classify each of the following still lifes as either a strict still life, a pseudo still life, or neither.



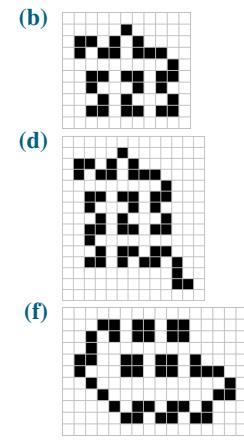
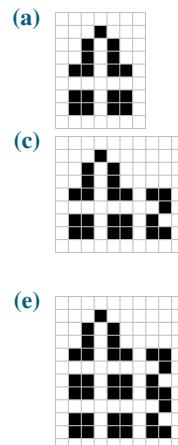
- 2.2 A *quasi still life* is a still life that can be partitioned into two or more disjoint still lifes with overlapping Moore neighborhoods (just like pseudo still lifes), but with all cells that stay dead from underpopulation in the constituent still lifes remaining underpopulated in the overall pattern. 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...

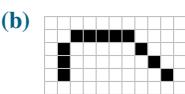
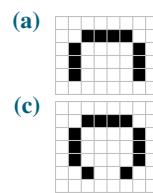


- (i) two pseudo still lifes,
 (ii) two quasi still lifes, and
 (iii) a quasi still life and two strict still lifes.

- *2.3 Partition each of the following pseudo still lifes into the smallest number (either 2, 3, or 4) of still lifes possible.



- *2.4 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 in the path in Figure 2.11 into a still life).



- 2.5 Find all strict still lifes, distinct up to rotation and reflection, with:

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

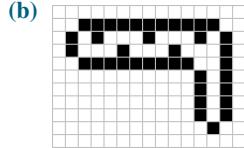
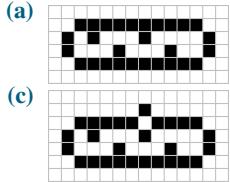
- 2.6 Find all pseudo still lifes, distinct up to rotation and reflection, with:

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

²⁰In 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.

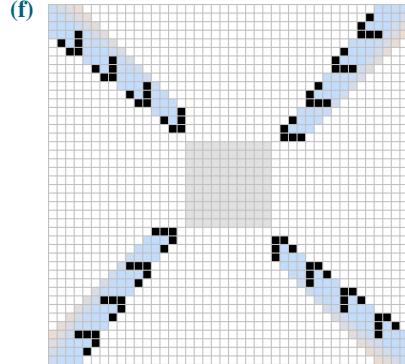
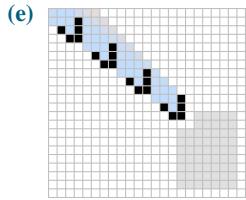
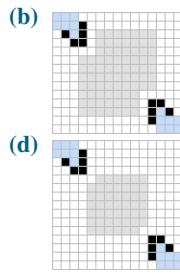
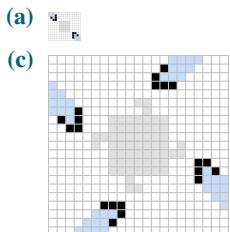
²¹His program is available online at github.com/simeksgol/GoL_still_life_searcher

***2.7** For each of the following patterns, find a way of adding induction coils so as to create a still life.



2.8 Use the Gosper glider gun and an eater of your choice to create a period 30 oscillator.

***2.9** 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.



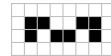
2.10 Show how a single eater 2 can be used to eat...

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

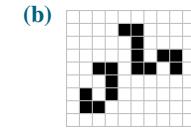
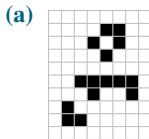
2.11 Recall *eater 3* from Figure 2.19.

- (a) Find a way to use eater 3 to eat a glider.
- (b) Find at least two still lifes that eater 3 can also eat when they are placed near its loaf.
- (c) Use two copies of the loaf-flipping reaction in eater 3 to create a period 8 oscillator.

2.12 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.

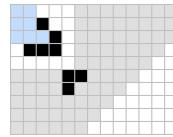


***2.13** Show how each of the following still lifes can be used to eat a glider in only 4 generations, tying them with eater 1 as the fastest-known glider eaters.



2.14 Use a single loaf to eat both of the gliders displayed in Exercise 2.9(a), (b), and (d).

***2.15** 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** 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 Find maximum-density still lifes in $n \times n$ bounding boxes for $n = 4, 6$, and 8 that are different from those displayed in Table 2.2.

2.18 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** 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 In this question, we will 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 Proposition 2.3.]

- (c) Use the formula from part (b) to show that a still life with an $2 \times n$ 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 $n = 2, 3, \dots, 10$. Compare your results with the bound from (b).

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

Early draft (May 19, 2020).

Not for public dissemination.

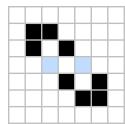
3. Oscillators



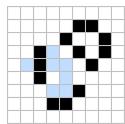
That it will never come again is what makes life so sweet.

Emily Dickinson

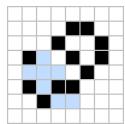
Recall that an oscillator is a pattern that returns to its initial phase after 2 or more generations, and the smallest number of generations required is its *period*.¹ Some oscillators that we have already seen occurred naturally, such as the blinker, pulsar, and pentadecathlon, and we also constructed 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.



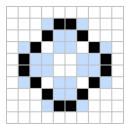
(a) Bipole (p2).



(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) unknown early Lifers 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 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

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.

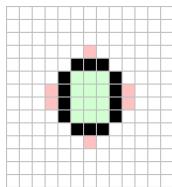
² Jam and mold got their names from the fact that they both consist of something on top of a loaf. To remember which is which, notice that the number of letters in their name matches their period.

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”.

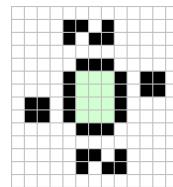
3.1 Billiard Tables

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**.

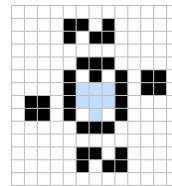


(b) One way of overpopulating the red cells with stable objects. All that remains is to find oscillating debris to place in the inner green section.

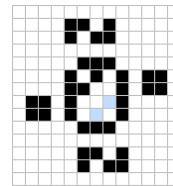
Figure 3.2: A 4×3 box that we would like to turn into an oscillator. The red cells in (a) have 3 live neighbors and thus will be alive in the next generation, unless we crowd them with induction coils. One way of crowding them is shown in (b).

Since there are only $2^{4 \times 3} = 4,096$ 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 4,096 (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.



(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.

Task (2) above is a bit more difficult to take care of, but since there are only 4,096 possible configurations, many of which we clearly do not need to consider (such as the configuration with all

³In 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.

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 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 no-name p10 in Figure 3.4(c)).

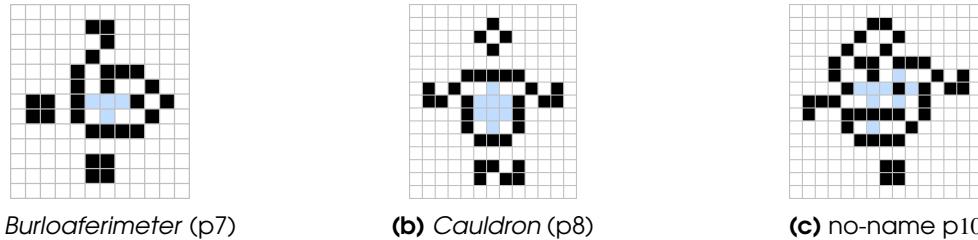


Figure 3.4: Some simple billiard table oscillators that were known right from the early days of Life. They were found by (a) David Buckingham in 1972, (b) Don Woods and Robert Wainwright in 1971, and (c) David Buckingham no later than 1976.

3.2 Stabilizing Corners

Another method that is frequently used to construct oscillators by hand is to simply place different combinations and 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.

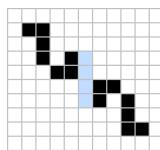


Figure 3.5: *Two eaters* is a period 3 oscillator that consists of two eater 1s eating and rebuilding their corners.

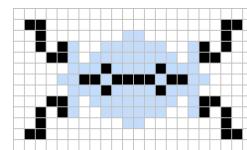


Figure 3.6: *Snacker* is a period 9 oscillator constructed by placing a pentadecathlon in the middle of four eater 1s.

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 re-build 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.

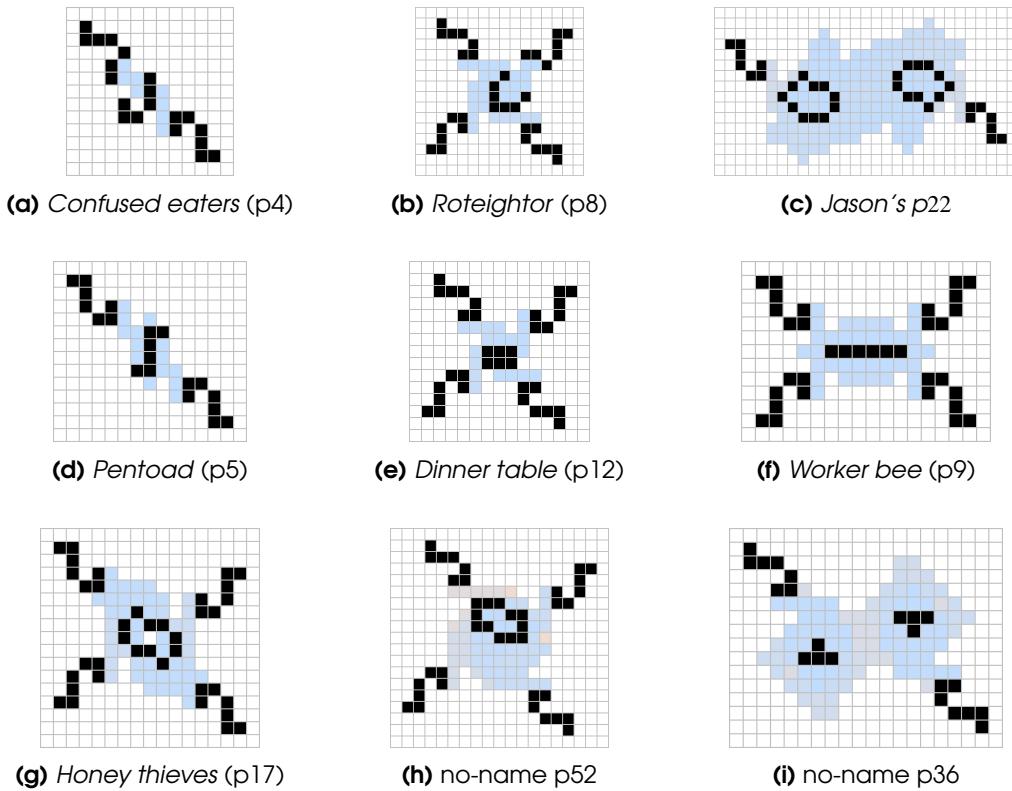


Figure 3.7: Eater 1s can be used to stabilize the corners of several oscillators with a wide variety of periods. These oscillators were found by (a,f) David Buckingham no later than 1972, (b,e) Robert Wainwright in 1972, (c) Jason Summers in 2000, (d) Bill Gosper in 1977, (g) Matthias Merzenich in 2014,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

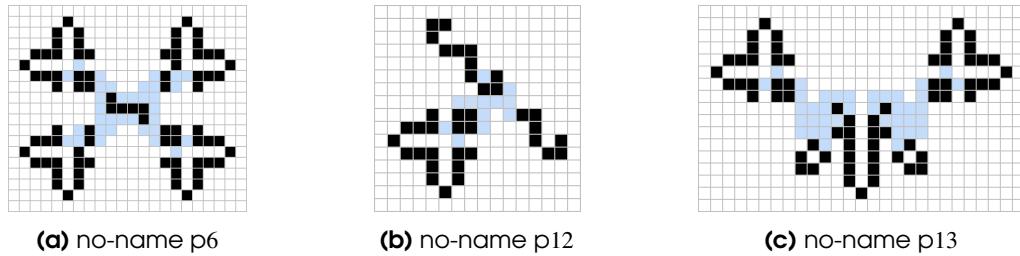


Figure 3.8: Eater 2s can also be used to stabilize the corners of some oscillators. These oscillators were found by David Buckingham in (a) 1977 and (c) 1976, and by (b) Matthias Merzenich in 2015.

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.

3.3 Composite Periods and Sparks

One very simple way to create oscillators with new periods is to simply 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).

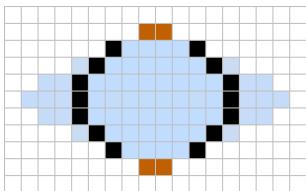


Figure 3.9: One phase of the pentadecathlon has two domino sparks (shown in orange) that immediately die off.

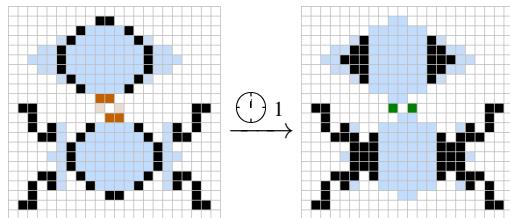


Figure 3.10: A pentadecathlon strategically placed next to a snacker makes a non-trivial period 45 oscillator, since the two cells shown in green on the right are only alive 1 generation out of every 45.

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.

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

⁵While 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.

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

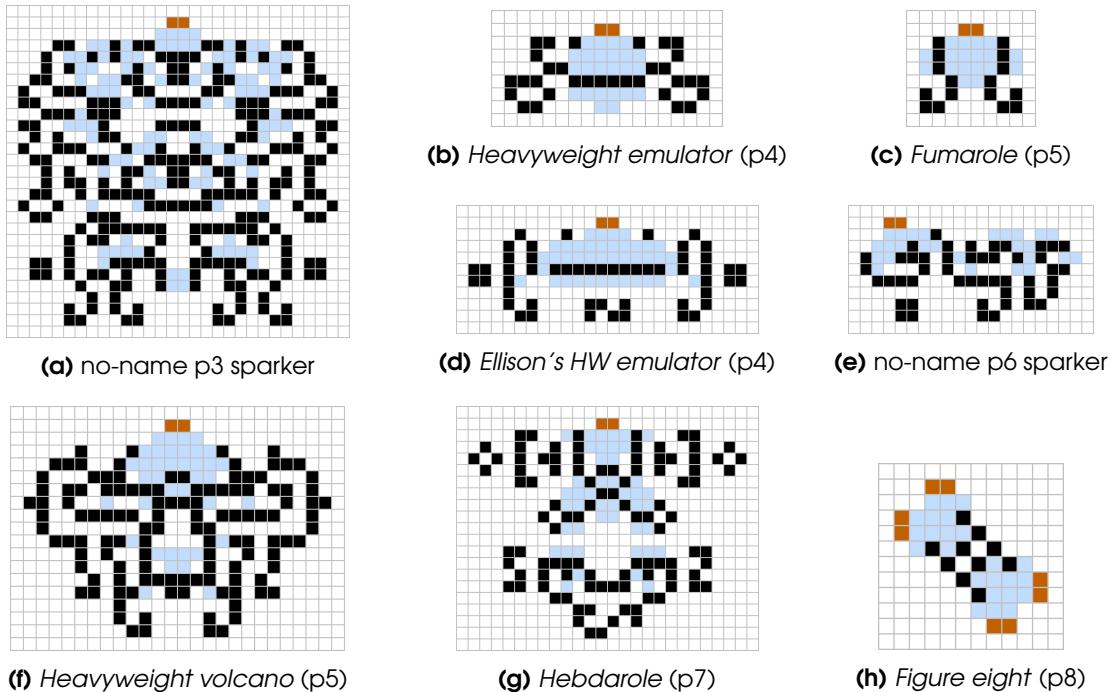


Figure 3.11: A collection of oscillators that emit a domino spark (highlighted in orange). These oscillators were found by (a,e,g) Noam Elkies in 1997, (b) Robert Wainwright in 1980, (c) Dean Hickerson in 1989, (d) Scot Ellison in 2010, (f) Dean Hickerson in 1995 (with improvements to make it smaller 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 pipsquierter, thanks to the multiple orientations of the sparks that it emits.

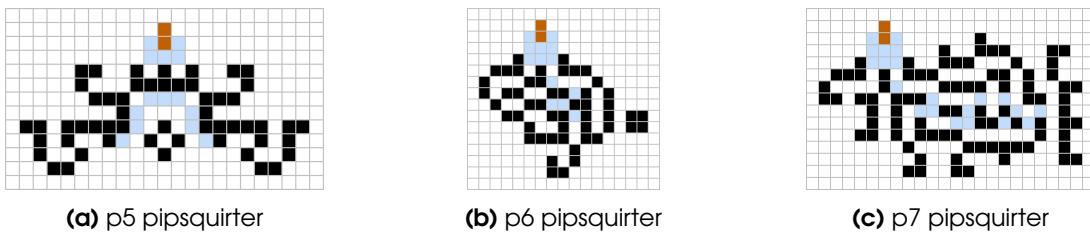


Figure 3.12: Some pipsquirters: oscillators that produce a domino spark (highlighted in orange) perpendicular to their closest edge. The p5 pipsquierter (a) was found by David Eppstein in April 2003, but is somewhat less useful than the other pipsquirters since its spark is closer to the rest of the 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 pipsquierter.

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.

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 middleweight and heavyweight spaceships from Figure 1.10. “Emulators” give their sparks off in the

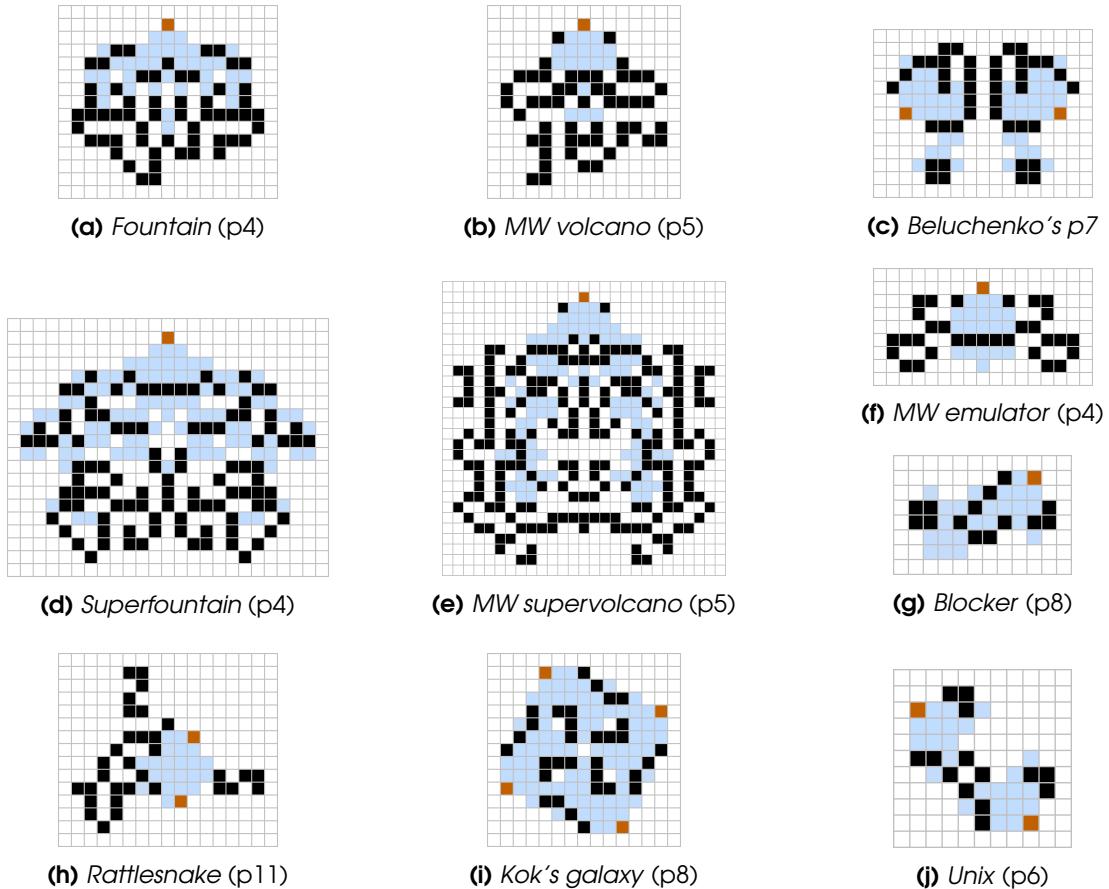


Figure 3.13: A collection of oscillators that emit a dot spark (highlighted in orange). These oscillators were found by Dean Hickerson in (a) 1994, (b) 1992, and (h) 2016, Nicolay Beluchenko in (c) 2009 and (d) 2006, (e) Dongook Lee in 2019, Robert Wainwright in (f) 1980 and (g) no later than 1983 (likely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

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 3.7.

While two dot sparkers cannot be combined with each other to create a non-trivial oscillator (since two dot sparkers 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.

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

⁷Some 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.

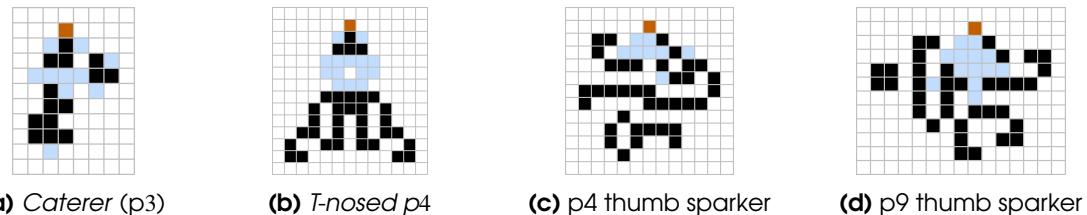


Figure 3.14: Some (a,b) finger sparkers and (c,d) thumb sparkers, with their sparks highlighted in orange. These sparkers were found by Dean Hickerson in (a) 1989 and (d) 1998, (b) Robert Wainwright in 1989, and (c) David Eppstein in 2000. Also, octagon 2 from Figure 3.1(d) is a period 5 finger sparker.

with dot or domino sparks.

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 96.

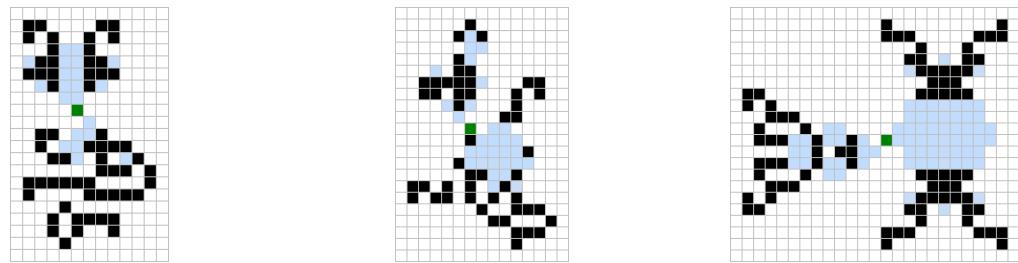


Figure 3.15: Some oscillators that work by combining finger and thumb sparks with other sparks. Finger sparks can be combined with either dot sparks (as in (b)) or domino sparks (as in (c)), but thumb sparks are best paired with domino sparks (as in (a)). In all cases, the cell that oscillates at the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

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 cells⁸), 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.

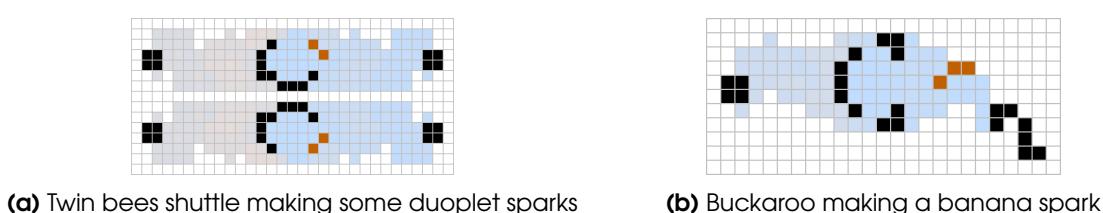


Figure 3.16: Some oscillators that emit slightly less common sparks (highlighted in orange).

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.

⁸The 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.



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-honeyfarm (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*.

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-honeyfarm (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 honeyfarm 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.

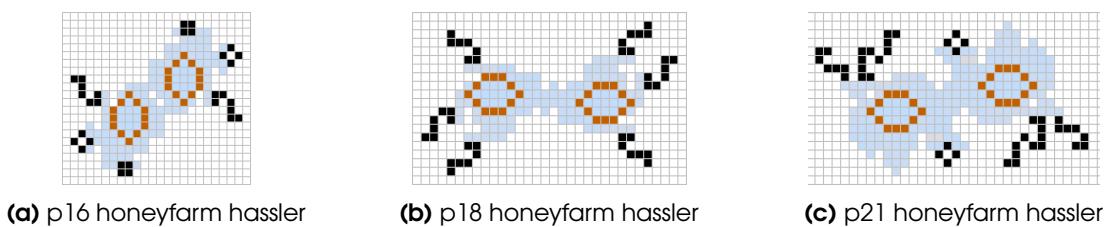


Figure 3.18: Several oscillators that work by hassling pre-honeyfarms (highlighted in orange). These oscillators were found by (a,c) Dongook Lee in 2016 (with improvements to decrease their size by Matthias Merzenich), and (b) Nico Brown in January 2015.

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 this twin bees shuttle. We devote the entirety of Section 6.3.2 to investigating things that can be done with it.

In the examples that we have seen so far, the object being hassled (a pre-honeyfarm 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 20 generations for these blocks and ponds to return to their original formations, so this reaction can be

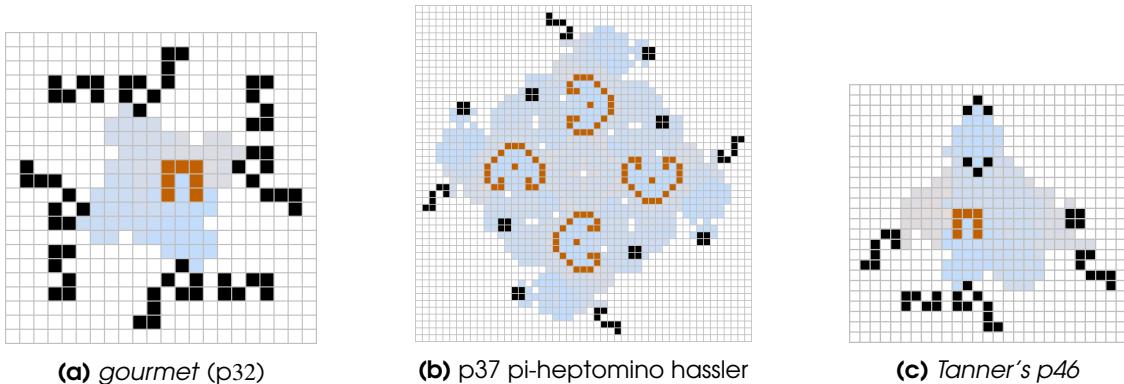


Figure 3.19: Some oscillators that work by hassling pi-heptominoes (highlighted in orange). These oscillators were found by (a) David Buckingham in 1978, (b) Nicolay Beluchenko in 2010, and (c) Tanner Jacobi in 2017. While the oscillator in (b) might not look like it is hassling pi-heptominoes, the chaotic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

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.

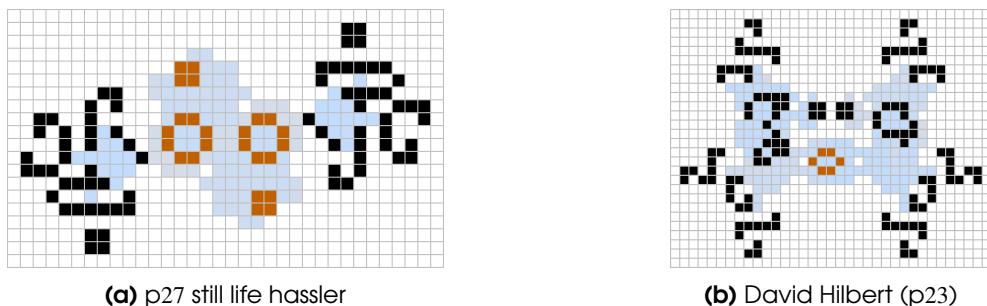


Figure 3.20: Some oscillators that work by hassling still lifes (highlighted in orange). These oscillators were found by (a) Jason Summers in 2005 and (b) Luka Okanishi in 2019.

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-honeyfarms 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.

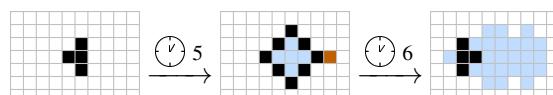


Figure 3.21: A dot spark (highlighted in orange) can be used to hassle a T-tetromino, moving it to the left by 2 cells and mirroring it over the course of 11 generations.

⁹This 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.

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

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 known 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.

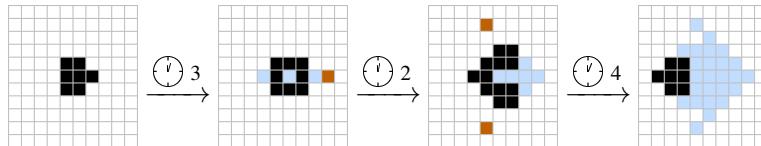


Figure 3.22: Three dot sparks (highlighted in orange) can be used to hassle generation 1 of a T-tetromino (i.e., the 7-cell object on the far left), moving it to the left by 2 cells and mirroring it over the course of 9 generations.

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 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).

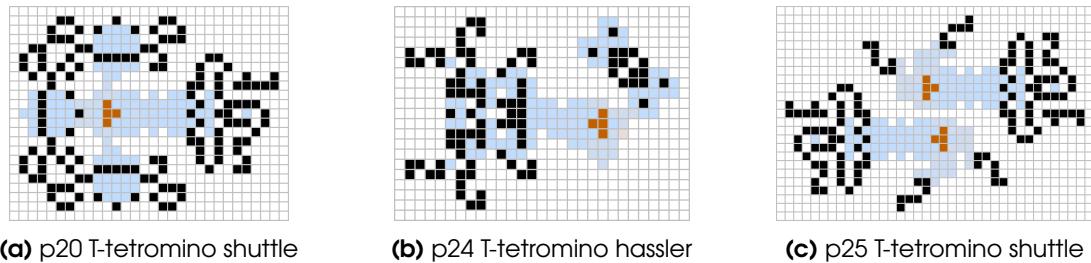


Figure 3.23: Some oscillators that work by hassling and shuttling T-tetrominoes (highlighted in orange). These oscillators were found by (a) Noam Elkies in 1995 and (c) in 1994, and by (b) Bill Gosper in 1994.

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 eights and a fountain 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,¹² 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).

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 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

¹¹The 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.

¹²Much like how placing a B-heptomino next to itself helped us create the twin bees shuttle in Section 1.4.

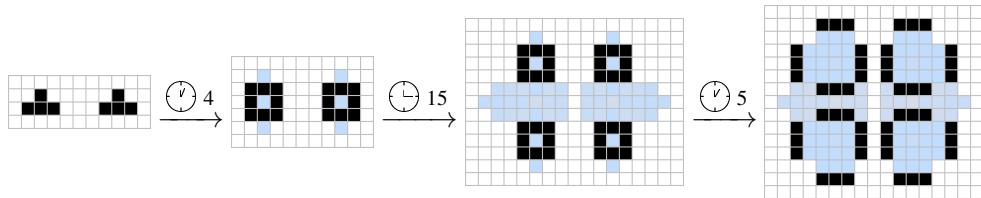


Figure 3.24: A *pre-pulsar* is an arrangement of two T-tetrominoes that duplicates itself and then evolves into a pulsar.

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).

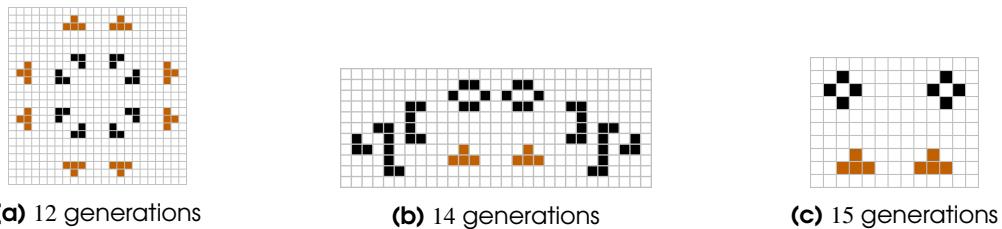


Figure 3.25: Some pre-pulsar pushers that move a pre-pulsar by 3 cells in either (a) 12, (b) 14, or (c) 15 generations. In all cases, the pre-pulsar is highlighted in orange and moves away from the object that does the hassling.

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 $14 + 14 = 28$, 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).

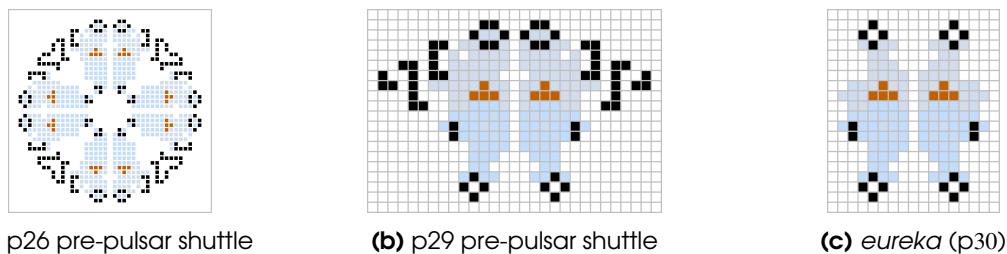


Figure 3.26: Some pre-pulsar shuttle oscillators constructed by pasting together the reactions from Figure 3.25. The period 30 shuttle (c) was the first known pre-pulsar shuttle, and it was found by David Buckingham in 1980.

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 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).¹³

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 use any oscillator that emits one of these sparks to reflect a glider stream, 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.

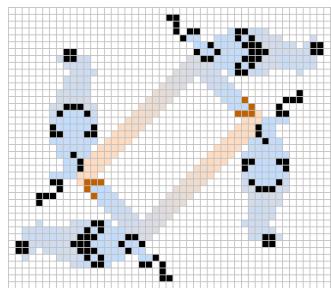


Figure 3.27: A period 90 oscillator consisting of two gliders bouncing around a track made up of four buckaroos.

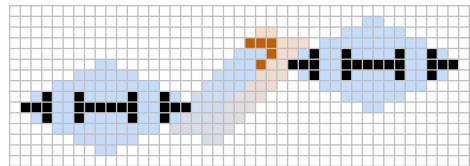


Figure 3.28: A period 60 oscillator consisting of a glider bouncing back and forth between two pentadecathlons.

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 $\text{lcm}(4, 15) = 60$. 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

¹³A detailed analysis of how close gliders can be without crashing into each other is presented in Appendix B.1.

than 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. 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 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*,¹⁴ 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 had a repeat time of 497 generations and is displayed in Figure 3.29(a).

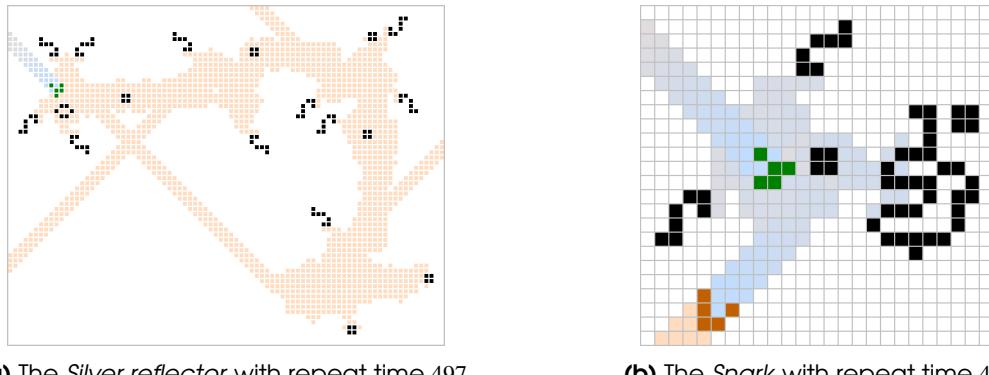


Figure 3.29: Some stable reflectors that reflect an input glider (highlighted in green) to one or more output positions (highlighted in orange). The (a) Silver reflector actually creates 3 output gliders (one to each of the northwest, northeast, and southwest), but they are far away and not displayed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

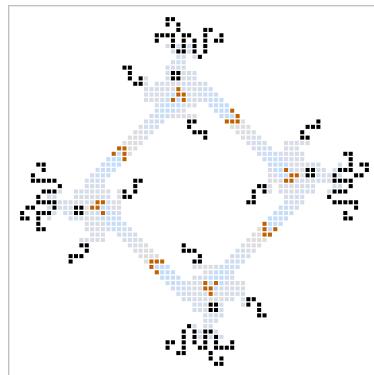
The techniques used to construct the Silver reflector will be investigated in Section 3.6 (and then in much more depth in Chapter 7), but for now we just note that it is perhaps a bit slow for our purposes. We want to create oscillators with as many periods as possible, but this reflector only helps with oscillators of period 497 or greater. Fortunately, faster stable reflectors are now known, with the smallest and fastest one being the *Snark*,^{15,16} 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 oscillators with 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).

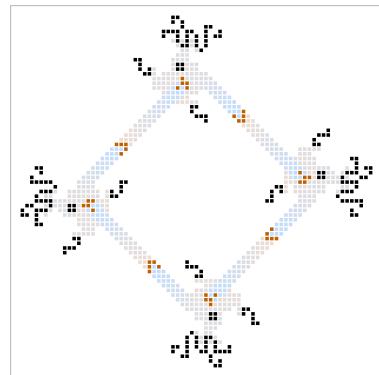
¹⁴Named after its discoverer, Stephen Silver, who constructed it in 1998.

¹⁵The 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.

¹⁶The 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.



(a) A period 43 glider loop.



(b) A period 53 glider loop.

Figure 3.30: The Snark can be used to create oscillators with any period of 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,¹⁷ 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.¹⁸

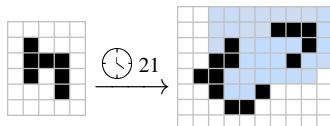


Figure 3.31: A *Herschel* is a chaotic pattern that emits a glider after 21 generations.

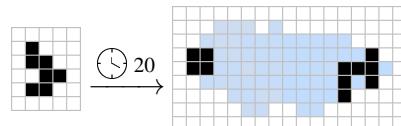


Figure 3.32: The B-heptomino takes 20 generations to evolve into a block and a Herschel.

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 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.¹⁹ This conduit is called *R64*, with the “R” referring to the fact that the Herschel turns to the right 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).

With this conduit, we can start playing a similar game to the one we played with glider reflectors:

¹⁷The 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.

¹⁸It 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.

¹⁹This conduit was found by David Buckingham in September 1995.

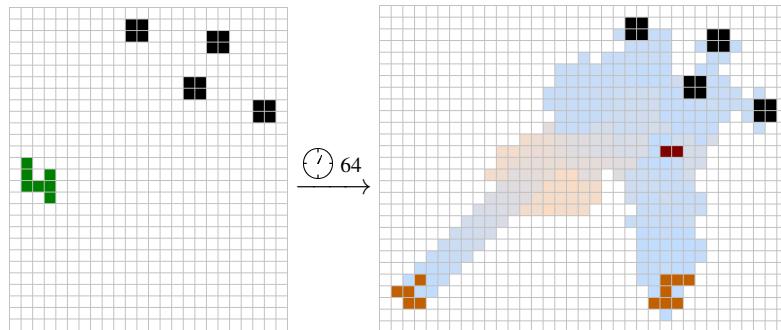
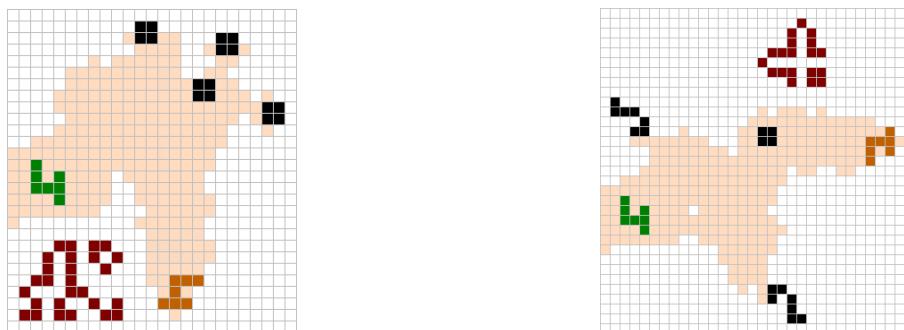


Figure 3.33: *R64*: A conduit consisting of four blocks that displaces and rotates a Herschel by 90 degrees in 64 generations (the domino spark on the right dies after one more generation and thus has no effect).

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 3.21). 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 put any of eater 1, eater 2, or eater 5 in such a way as to avoid colliding with the Herschel and also destroy the glider before it crosses the original Herschel's path. Instead, we can use the eater that we constructed back in Figure 2.24(b),²⁰ which lets this conduit accept Herschels that are spaced 61 or more generation apart (so its *repeat time* is 61 generations).



(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 57 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.

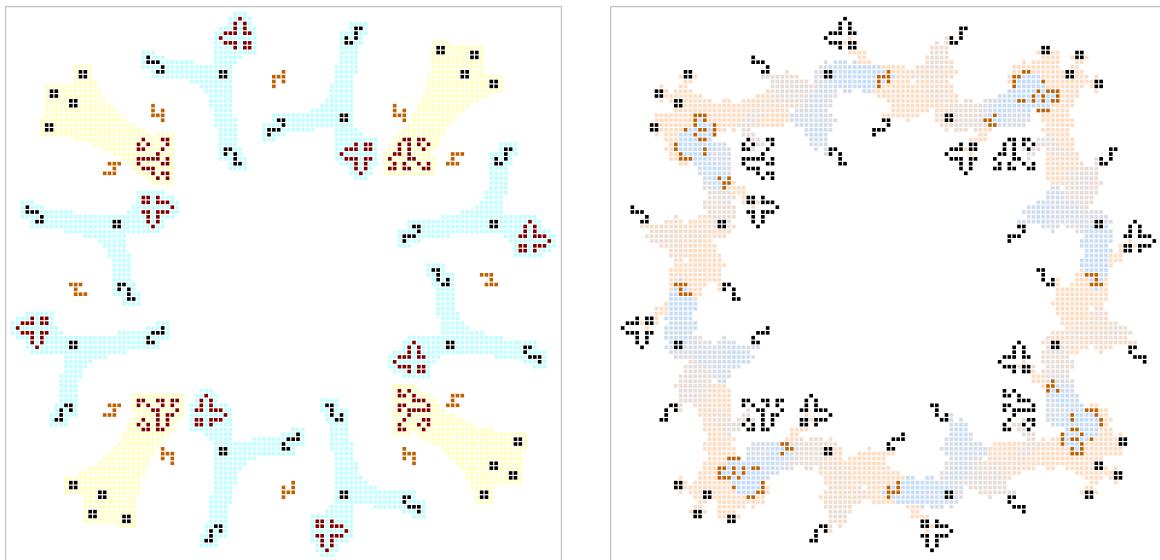
²⁰In fact, this is exactly why we constructed that eater: the constraints that we placed on the size of the eater in Figure 2.24(b) were designed exactly to make sure that it did not interfere with the input or output Herschel from this conduit (compare with Figure 3.34(a)).

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 generations to complete, and the conduit's repeat time is 57 generations. 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 easter that we placed next to the R64 conduit in Figure 3.34(a)).²¹

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).



(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 $872/8 = 109$.

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

²¹We use eater 2 because we can place it slightly closer to the conduit without disrupting it than we can place eater 1.

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 3.24).

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 $(4 \times 64) + (8k \times 77) = 256 + 616k$ generations to traverse the track. To create an oscillator with period p , we require $256 + 616k$ to be evenly divisible by 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. However, they can be used to construct oscillators with all other periods (no smaller than 61):

Proposition 3.1 Let $p \geq 1$ be a positive integer. There exists an integer $k \geq 1$ such that p evenly divides $256 + 616k$ 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 p evenly divides $256 + 616k$ 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, k such that $pa - 616k = 256$. This is a linear Diophantine equation, which is a well-studied mathematical object, and it is known that a and k exist if and only if $\gcd(p, 616)$ evenly divides 256 (see Appendix A.2 for details). 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. ■

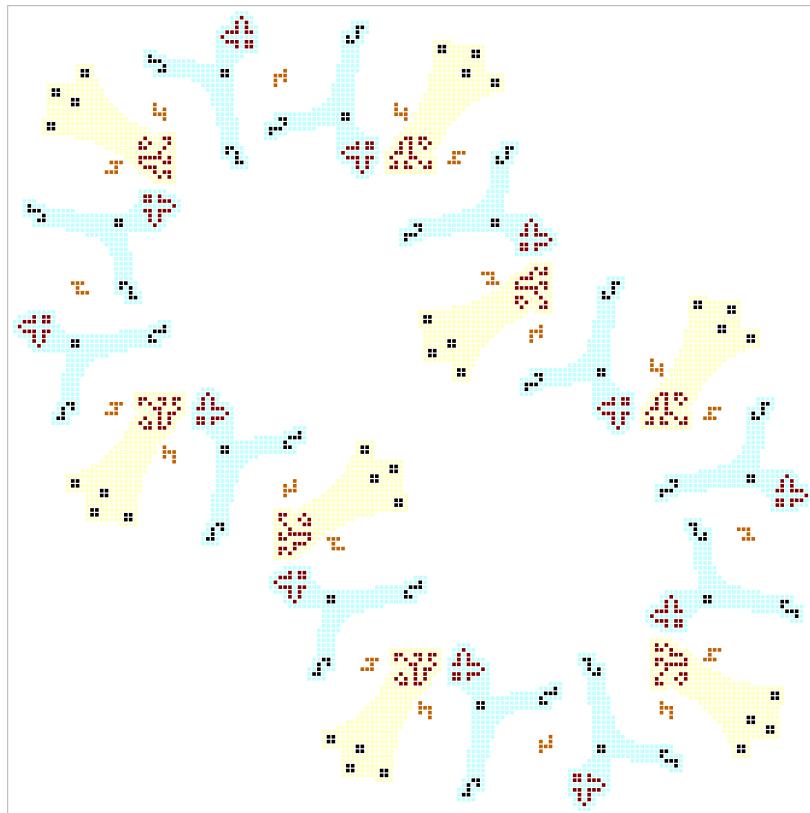


Figure 3.36: A Herschel track that just uses a single Fx77 conduit on some sides so that the next R64 turns the track outward instead of inward. Using this technique, we can construct a wide variety of Herschel tracks of almost any shape.

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 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,²² 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:

Proposition 3.2 Let $p \geq 1$ be a positive integer. Then there exists an integer $k \geq 13$ such that p evenly divides $77(8k - 48)$. 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 Proposition 3.1: we note that p evenly divides $77(8k - 48)$ if and only if we can find positive integers a and k such that $pa - 77 \times 8k = -(77 \times 48)$, if and only if $\gcd(p, 77 \times 8)$ evenly divides 77×48 , which is always the case regardless of the value of p . ■

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 Omniporodicity

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 *omniporodic*, and this omniporodicity 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, omniporodicity in general remains unsolved, as there are still gaps that we do not know how to fill in. In particular, we do not know

²²In 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 $k \geq 13$.

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	burloafometer, hebdarole
8	1970	blocker, cauldron, figure eight, Hertz oscillator, roteightor
9	1972	p9 thumb sparker, snacker, worker bee
10	≤ 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	≤ 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 3.10)
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		none known
39	2000	caterer on p13 sparker (see Exercise 3.10(a))
40	≤ 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 four periods for which it is unknown how to construct an oscillator are 19, 34, 38, and 41. Note that some large-period oscillators were found much earlier than 1996 or 2013, and these are not catalogued in this table. For example, the twin bees shuttle has period 46 and was found in 1971.

how to construct oscillators with period 19, 34, 38, or 41.²³ 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 2000, 2002, 2009, 2010, 2013, and 2019.²⁴

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 be 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 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.²⁵ There’s no such thing in Life as a phoenix spaceship.

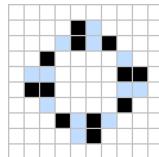


Figure 3.37: A period 2 phoenix oscillator.

Theorem 3.3 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 grey cells in Figure 3.38.

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.

²³Even 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.

²⁴David Buckingham and Noam Elkies found the first period 39 oscillator in 2000, 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 period 43 and 53 were found in 2013, and Luka Okanishi completed the first period 23 oscillator in 2019.

²⁵This theorem was originally proved by Stephen Silver in January 2000.

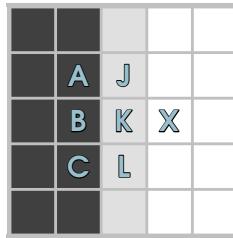


Figure 3.38: A diagram illustrating the fact that no phoenix can extend outside of its original bounding box (depicted by the dark grey cells) by more than one cell (depicted by the light grey cells). In particular, if the phoenix is initially confined to the dark grey cells, then cell X can never be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

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. ■

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 phoenixes 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 phoenixes 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 phoenixes with period 3 or higher. The following theorem²⁶ shows that no phoenix of period 3 exists.

Theorem 3.4 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.

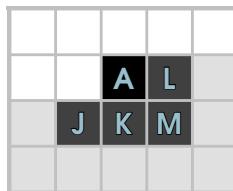


Figure 3.39: A diagram depicting the top-left corner of a hypothetical period 3 phoenix oscillator. Cell A is the leftmost cell in the uppermost row of the oscillator, and exactly three of J, K, L, or M must be alive in the generation before A is alive.

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

²⁶This theorem was also originally proved by Stephen Silver in January 2000.

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.



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.

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,²⁷ but essentially nothing else is known about the (non-)existence of phoenices with period 4 or greater. On 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.

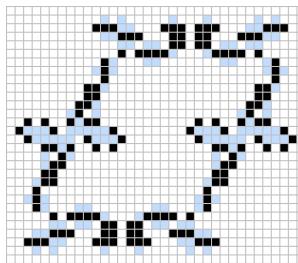


Figure 3.41: A period 3 oscillator with the property that every cell oscillates. It was found by Jason Summers in August 2012.

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 small size and simplicity, 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).

However, oscillators also have great practical use, since 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

²⁷This proof was completed by Alex Greason in September 2019.

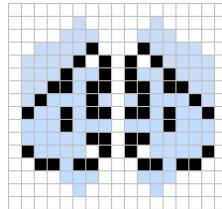


Figure 3.42: Rich’s *p16* is a period 16 oscillator that was found in a random soup by Rich Holmes in July 2016.

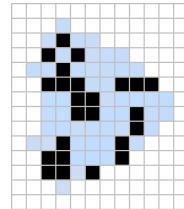


Figure 3.43: Rob’s *p16* is a period 16 oscillator that was found in a random soup by Rob Liston in February 2020.

will construct in later chapters. For example, the entirety of Chapter 6 is devoted to using oscillators to emulate logic circuits that can perform computation.

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 cataloguing 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],²⁸ 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.

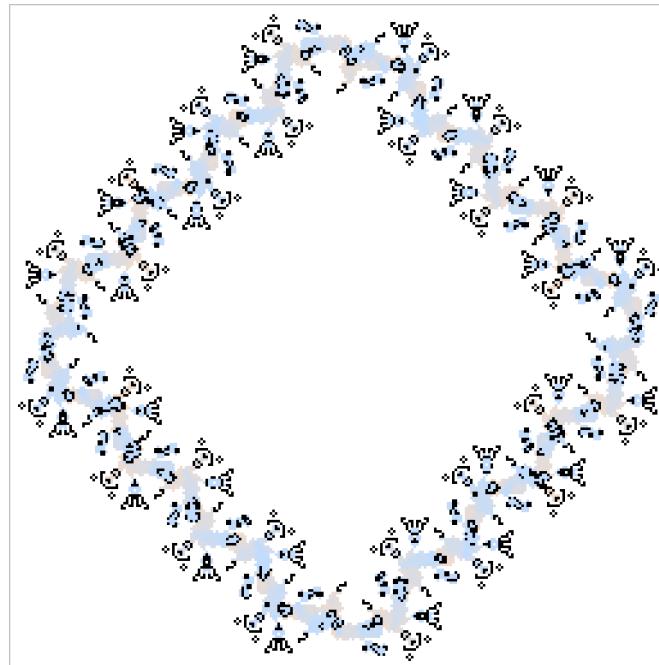


Figure 3.44: A period 56 oscillator that was constructed using periodic Herschel conduits. It was built by Dietrich Leithner in December 1997.

A 180-degree stable reflector with repeat time of 202 generations, called the *boojum reflector*,²⁹

²⁸Although it had been known since at least 1982 that stable reflectors must exist in Life [BCG82], no specific examples had been found prior to Callahan’s construction.

²⁹A “boojum” is a particularly dangerous type of Snark.

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 3.20). 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.

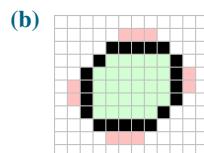
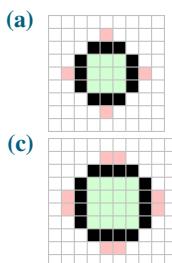
Although Buckingham's original Herschel conduits were used to construct oscillators with period 61 or higher, faster-recovering conduits can be used to create oscillators with period 58, 59, and 60 as well (see Exercise 3.26). In December 1997, Dietrich Leithner found fast oscillating Herschel conduits that even allow for the construction of oscillators with period 56 and 57 (see Figure 3.44 and Exercises 3.27 and 3.28). 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.

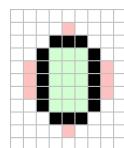
Exercises

solutions to starred exercises on page 303

- *3.1** 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.



- 3.2** Consider the 5×3 box displayed below as a potential starting point for constructing a billiard table oscillator.

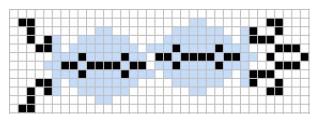


- (a) Find a stable way to crowd the red cells, just like we used two blocks and two snakes to crowd the outside of a 4×3 box in Section 3.1.
 (b) Show that there is no way to fill in the green cells so as to make this pattern an oscillator (try writing a computer program to help you).

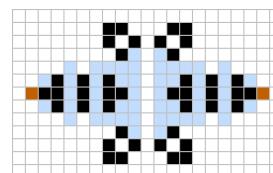
- 3.3** Replace the eater 1s with eater 2s in the oscillators from the following figures, without destroying the oscillator or altering its period:

- (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** 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.

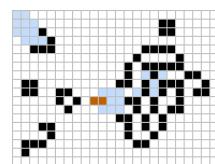


- *3.5** Use the sparks from either the T-nosed p4 or the T-nosed p6 (displayed below) and another oscillator of your choice to create non-trivial oscillators with...



- (a) period 12,
 (b) period 20, and
 (c) period 24.

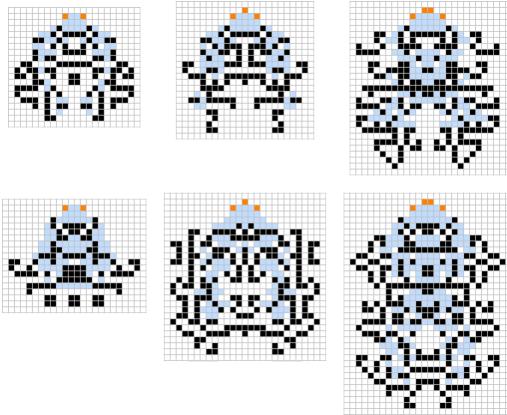
- 3.6** One of the primary uses of pipsquirters is their ability to reflect a glider when a boat, block, and eater 1 are placed nearby as shown below.



The reflector above has period 6. Use this same reaction to create a glider reflector with...

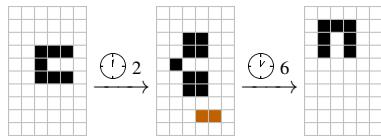
- (a) period 7, and
 (b) period 8.

3.7 Several period 4 and 5 lightweight, middleweight, and heavyweight supervolcanoes are displayed below,³⁰ and they are particularly useful due to how far they push away their sparks.



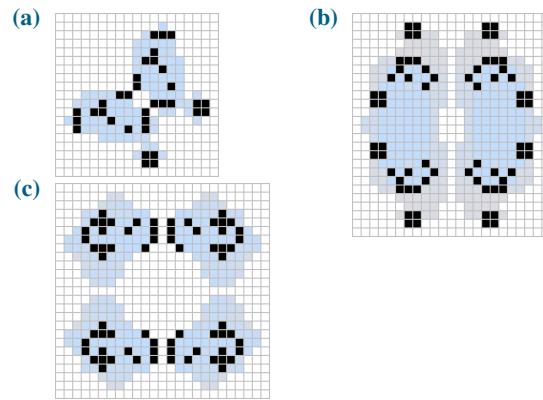
- (a) Use the sparks from two supervolcanoes to construct a period 20 oscillator.
- (b) Use a supervolcano and a thumb sparker to create a non-trivial period 20 oscillator.
- (c) Use a supervolcano to help you construct a T-tetromino shuttle.

***3.8** 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.]

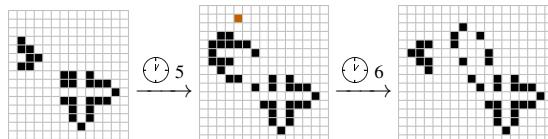


***3.9** Show how the snakes can be replaced by eater 1s in the period 32 “gourmet” oscillator from Figure 3.19(a).

***3.10** 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.

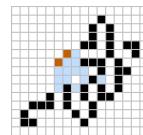


3.11 A dot spark and an eater 2 can be used to reflect a B-heptomino in 11 generations, as shown below. Use this reaction to create a period 22 oscillator.



3.12 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:



3.13 Use two pentadecathlons to construct a glider loop oscillator with period 180.

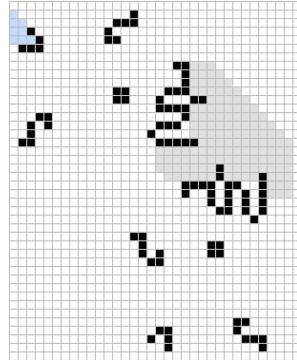
3.14 Use four Snarks to construct an oscillator with period 180.

***3.15** Create a glider loop (of any period) that makes use of exactly six Snarks.

³⁰The p4 LW supervolcano was found by Noam Elkies in 2010, the p5 MW supervolcano was found by Dongook Lee in 2019, the p5 HW supervolcano was found by Karel Suhajda 2004, and the rest were found by Matthias Merzenich in 2010.

*3.16 Sometimes it is useful to weld together reflectors, just like we welded together eaters in the previous chapter.

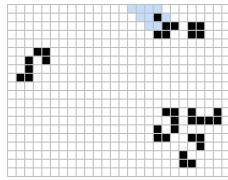
- (a) Weld together the following two partial Snarks (modify only the light gray cells), thus creating a single still life that can reflect gliders from two different sides.



- (b) Modify the reflector from part (a) to create a single reflector that can reflect gliders coming from four different sides.

*3.17 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?

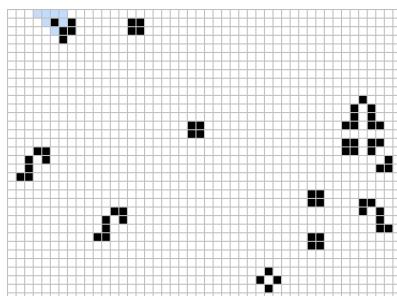
*3.18 The pattern displayed below might be called an “almost Snark”,³¹ since it can *almost* be used as a 90-degree glider reflector. Describe what goes wrong and why it cannot reflect more than one glider.



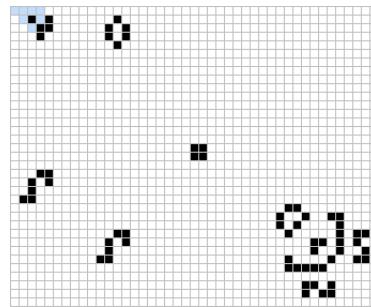
3.19 Create a glider loop that uses at least one buckaroo and at least one twin bees shuttle. What is the smallest possible period of such an oscillator?

3.20 For each of the following 180° 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:



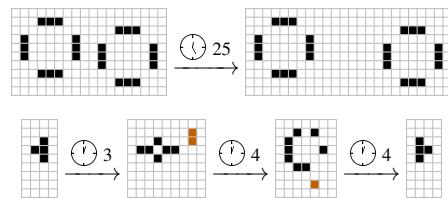
- (b) Rectifier:



3.21 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.

* 3.22 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.

*3.23 What type of sparks does the traffic jam reaction from the top of Exercise 3.22 produce? Use this spark and the oscillator constructed in Exercise 3.22 to create a periodic glider reflector.

3.24 Use the R64 and Fx77 conduits to construct Herschel track oscillators or guns with the following periods.

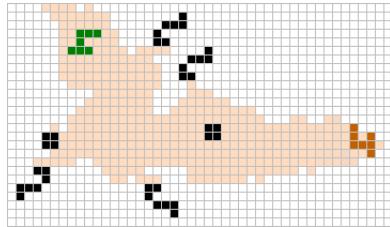
[Hint: Mimic our construction in Figure 3.35.]

- (a) Period 218 oscillator.
 (b) Period 62 oscillator.
 (c) Period 62 glider gun.
 (d) Period 61 oscillator.
 (e) Period 63 glider gun.

3.25 By placing different numbers of Herschels on the track from Figure 3.36, oscillators of which periods can be created?

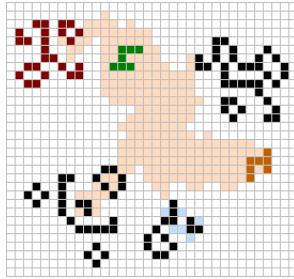
3.26 Consider the Herschel conduit displayed below, which rotates a Herschel counter-clockwise in 112 generations and has a repeat time of 58 generations. This conduit is called *L112*.

³¹Found by Tanner Jacobi in 2015.



- (a)** Construct a period 109 oscillator by using only the L112 and Fx77 conduits (and possibly some eaters to destroy stray gliders).
- (b)** 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.

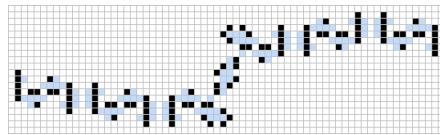
3.27 Consider the (oscillating!) Herschel conduit displayed below, which rotates a Herschel counter-clockwise and flips it in 65 generations, and has a repeat time of 57 generations. Construct a period 57 oscillator by using this new conduit and the Fx77 conduit (and possibly some eaters to destroy stray gliders).



3.28 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.

***3.29** Prove that there is no period 4 oscillator with exactly one cell that oscillates at the full period.

***3.30** 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.



***3.31** 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.

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 2 or more 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).

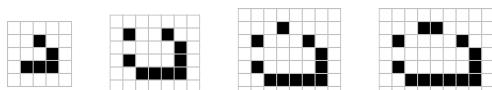


Figure 4.1: The four basic spaceships in Conway’s Game of Life. From left to right, these are the glider (which moves diagonally at a speed of $c/4$) and the light/middle/heavyweight spaceships (which each move orthogonally¹ at a speed of $c/2$).

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 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

¹The term *orthogonal* means straight left–right or up–down, as opposed to diagonal.

²Dean Hickerson found the first known truly new spaceships via computer search in 1989.

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 use 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.

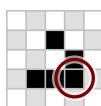


Figure 4.2: This glider's color is white since its leading cell (circled in red) is located at one of the white cells on the checkerboard pattern.

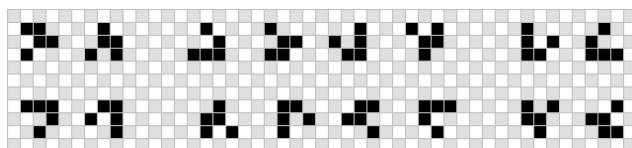


Figure 4.3: A collection of 16 gliders that all have the same color as each other. If any of these gliders were moved to the left, right, up, or down by 1 cell then their color would change.

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.

³And arguably the single most useful *pattern* that exists.

⁴The term *parity* is sometimes used instead of color.

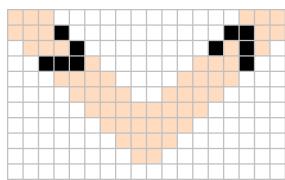


Figure 4.4: A path along which we would like to reflect a glider. Notice that the reflected glider has the opposite color of the original glider.

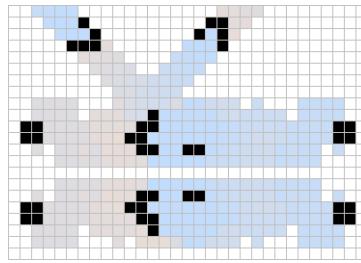


Figure 4.5: Twin bees shuttle is a color-changing reflector that can be used to reflect the glider in the desired way.

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”.

-1	0	1	2	3	4	5	.	.
-2	-1	0	1	2	3	4	5	.
-3	-2	-1	0	■	2	3	4	5
-4	-3	-2	-1	0	■	2	3	4
-5	-4	-3	■	■	■	1	2	3
.	-5	-4	-3	-2	-1	0	1	2
.	.	-5	-4	-3	-2	-1	0	1

Figure 4.6: A glider that is in lane 0, which is highlighted in light grey. 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.

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

⁵In fact, a glider’s color can just be thought of as its lane modulo 2.

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.

.	.	-8	-6	-4	-2	0	2
.	-8	■	-4	-2	0	2	4
-8	-6	-4	■	0	2	4	6
-6	■	■	■	2	4	6	8
-4	-2	0	2	4	6	8	.
-2	0	2	4	6	8	.	.

Figure 4.7: A glider that has timing 0, with the diagonal line of light grey 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.

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.

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, the back end of the glider only serves to overpopulate the front end of the tagalong.

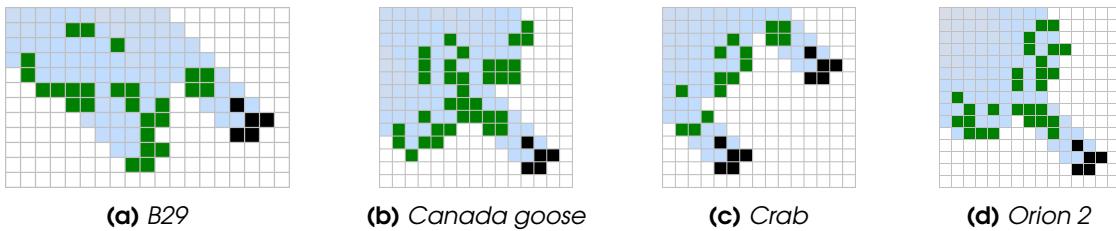


Figure 4.8: Some tagalongs (highlighted in green) that can trail behind a glider. They were found by (a) Hartmut Holzwart in 2004, and Jason Summers in (c) 2000, and (b,d) 1999.⁷

One useful feature of tagalongs is that they often themselves emit sparks as well, so we can sometimes chain them together or use them to produce reactions that are impossible with the original spaceship itself. 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 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

⁶That is, the same diagonal that goes from bottom-left to top-right.

⁷The name “Orion 2” is a reference to a similar, but larger, spaceship called *Orion* that was found by Hartmut Holzwart in 1993.

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.

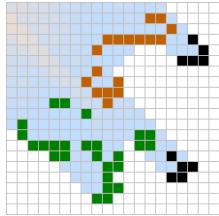


Figure 4.9: The B29 can pull a tagalong (highlighted in orange) that was found by Nicolay Beluchenko in 2005.

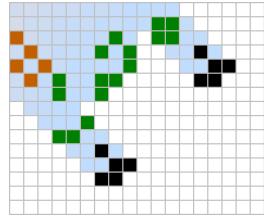


Figure 4.10: A pattern based on the crab called a *tubstretcher* that lengthens a tub (highlighted in orange) by 2 cells every 4 generations.

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 emit such accessible sparks so frequently. 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).

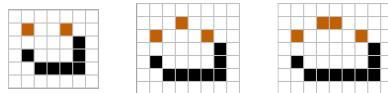


Figure 4.11: The light, middle, and heavyweight spaceship each give off several sparks (depicted in orange) that are extremely useful.

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 puffer, we could eliminate the unwanted debris (typically the blocks and blinkers), leaving only the desired output (typically the gliders).

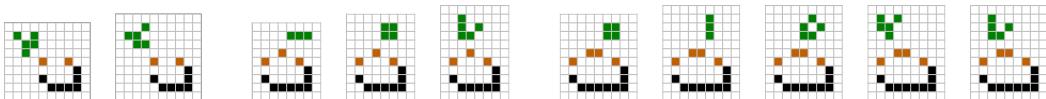


Figure 4.12: Light, middle, and heavyweight spaceships about to destroy some common small objects.

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

⁸This reaction in which a glider turns a beehive into a pond and block is called a *honeybit*.

time, but at the expense of having period 2 and thus only being able to destroy streams of heavyweight spaceships with even period.⁹

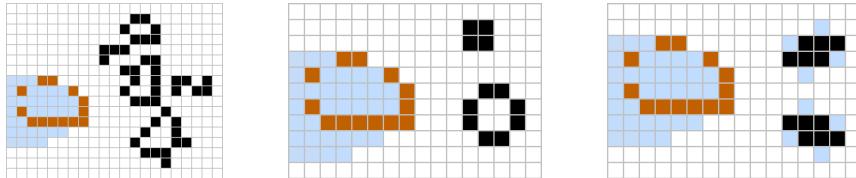


Figure 4.13: Three methods of eating a heavyweight spaceship. The arrangement of two toads on the right is called *killer toads*, and it can also eat many other objects, such as an MWSS in the same position.

4.2.1 Flotillas 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).

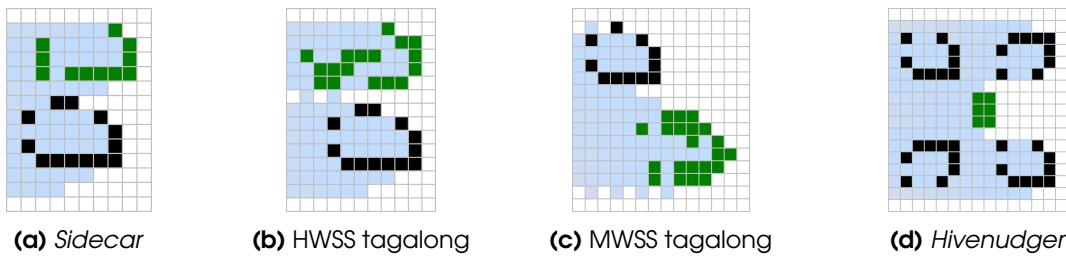


Figure 4.14: Some small tagalongs (highlighted in green) for light, middle, and heavyweight spaceships. These tagalongs were found in 1992 by (a,b,d) Hartmut Holzwart and (c) David Bell.

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 (for example, no pushalongs for the glider are known). There are 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 spaceships¹⁰ and *pseudo spaceships*, which are flotillas in which none of the component spaceships actually change their evolution at all, but at least one dead cell has more than 3 live neighbours 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.

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

⁹The 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.

¹⁰The 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.

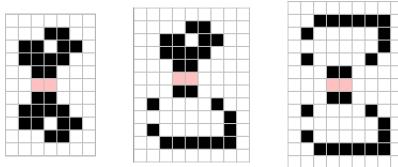


Figure 4.15: There are three ways of placing a LWSS, MWSS, and/or HWSS next to each other so as to create a pseudo spaceship. In all three of these cases, the component spaceships evolve in the exact same way as they would individually, yet in some of their phases the flotilla overpopulates some cells (highlighted in red) that the spaceships individually do not.

flotillas, and some examples involving a lightweight spaceship and a middleweight spaceship are presented in Figure 4.16.

Although there are numerous¹¹ ways of creating flotillas, they are a bit less satisfying than what we got when we combined oscillator sparks in Section 3.3. The main reason these flotillas 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.

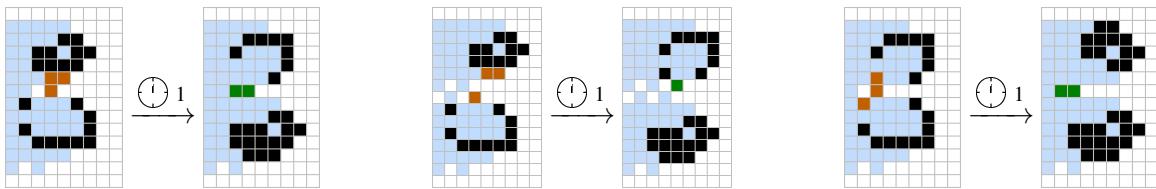


Figure 4.16: There are three ways of placing a LWSS and an MWSS next to each other in order to create a flotilla. The orange sparks interact in such a way as to give birth to the green cells that would not be born in either ship individually.

One way to construct flotillas that are a bit less trivial is to consider what might happen if we 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.

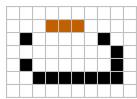


Figure 4.17: An *overweight spaceship*, which is not actually a spaceship since the three orange cells do not form a spark (i.e., they do not die) and they thus interfere with its evolution.

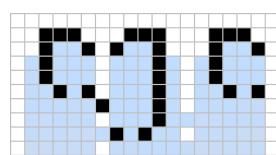


Figure 4.18: A flotilla that uses two lightweight spaceships to suppress the 3-cell “spark” of the OWSS, thus creating a new spaceship.

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 flotillas 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 tagalongs that exists. Overweight spaceships of any length can be stabilized by using an appropriate arrangement of smaller spaceships along their sides. It is even possible to

¹¹To be explicit, the number of flotillas involving two LWSS/MWSS/HWSS 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).

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).

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., the LWSS, MWSS, HWSS, 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 and the main ingredient in the construction of most of them is the reaction between two switch engines displayed in Figure 4.19.

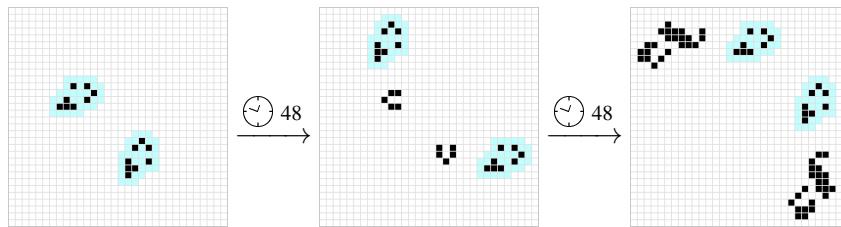


Figure 4.19: Two switch engines (highlighted in aqua) that almost stabilize each other. After 48 generations, they reappear but farther apart and with a couple of inconsequential 5-cell sparks. After the next 48 generations they return to their original relative positions, but along with some troublesome 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 to fix this problem (i.e., to suppress 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. 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.

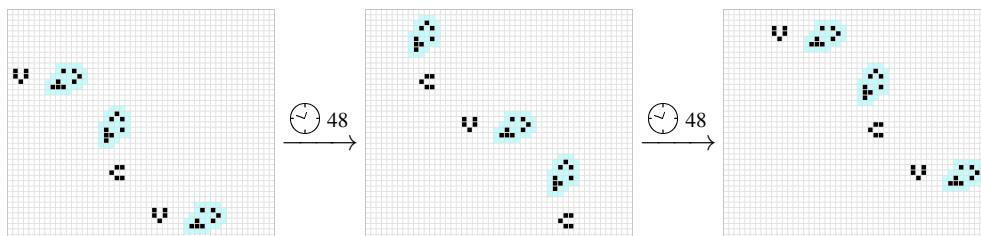


Figure 4.20: An infinitely-long wave of switch engines (highlighted in aqua) forever bounce off of each other and suppress each other’s debris, creating an object that moves at a speed of $c/12$ diagonally.

¹²The first ark was found in 1971, whereas the first spaceship based on switch engines was not found until 1991.

¹³Named after Charles Corderman, who discovered the switch engine and most of the simple puffers based on it in 1971.

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.

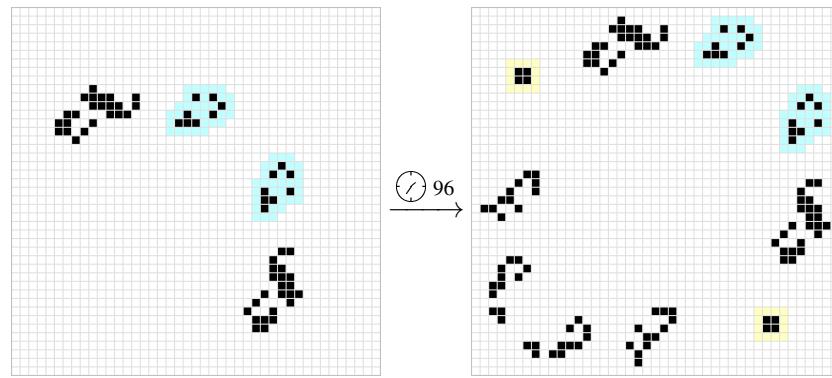


Figure 4.21: The debris left behind by each outermost switch engine temporarily creates a block (highlighted in yellow). This block is subsequently destroyed by the debris at the rear.

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,
- 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.

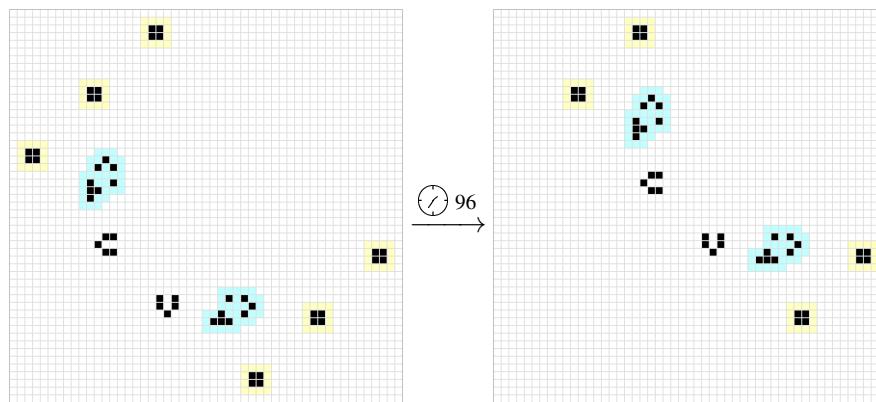


Figure 4.22: A trail of blocks (highlighted in yellow) can destroy and be destroyed by the debris left behind by a switch engine.

There are 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

There are countless 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 have seen all lead to somewhat large Corderships, some particularly clever Corderships are known that use as few as 2 switch engines (see Exercise 4.19).¹⁶

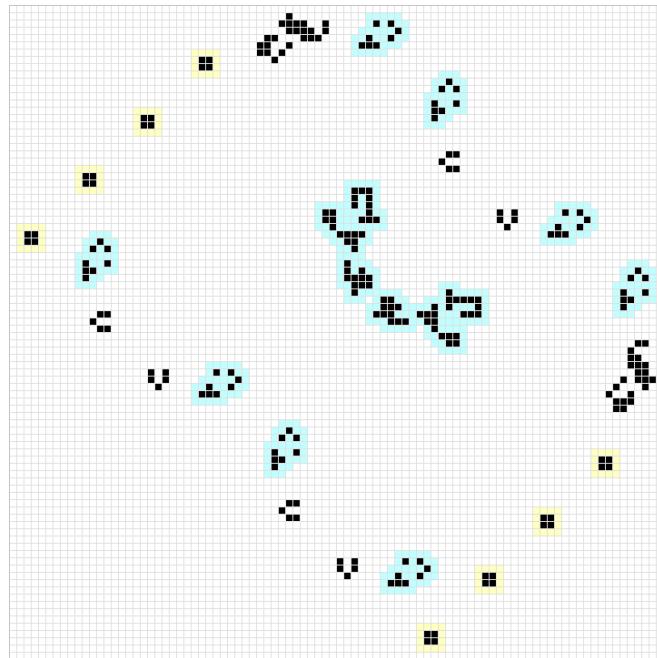


Figure 4.23: A 10-engine Cordership, which is a $c/12$ diagonal spaceship with period 96. In the orientation depicted here, it travels to the top-right, with the 4 switch engines at the front (highlighted in aqua) laying tracks of blocks (highlighted in yellow). The 2 central switch engines (which are 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).

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

¹⁴It 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).

¹⁵The 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.

¹⁶Paul 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 Aidan F. Pierce found a working 2-engine Cordership.

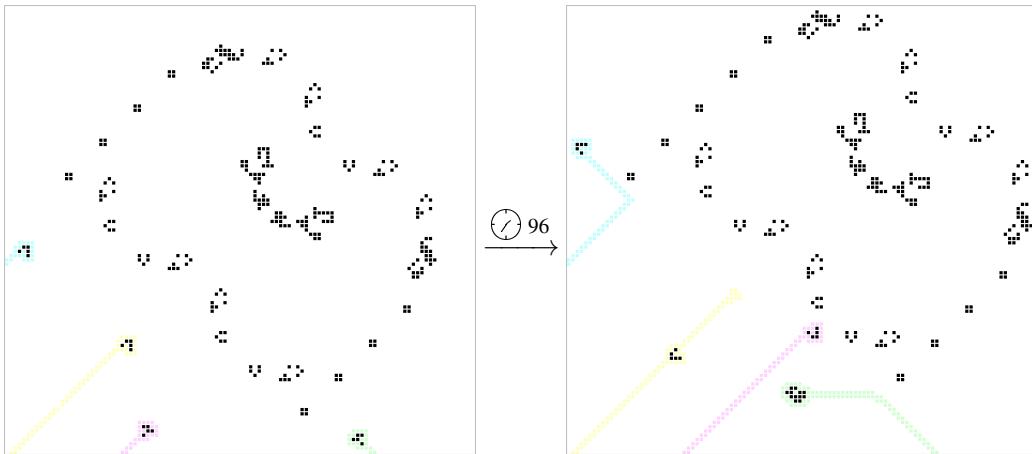


Figure 4.24: The pulsating sparks at the back of the 10-engine Cordership from Figure 4.23 can be used to reflect a glider by 90 degrees (highlighted in aqua and magenta), reflect a glider by 180 degrees (highlighted in yellow), and turn a glider into an LWSS (highlighted in green).

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.

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 5,532 generations to stabilize. However, after that 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

¹⁷It 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.

¹⁸This 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.

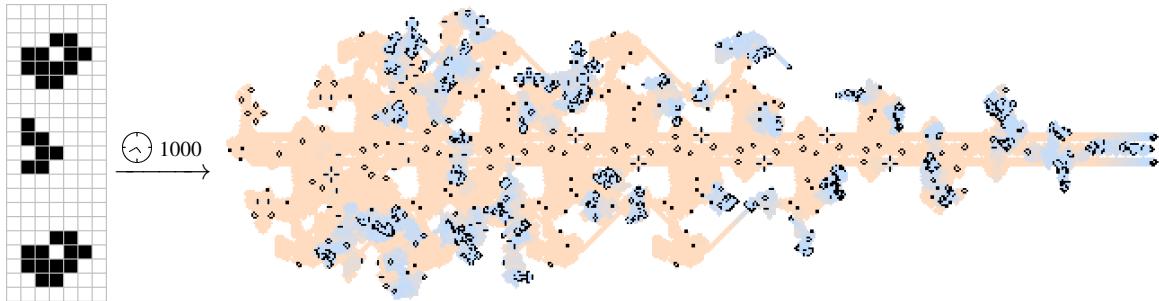


Figure 4.25: A puffer composed of a B-heptomino that has been stabilized by two lightweight spaceships. The debris in the image on the right is extremely chaotic, taking more than 5,000 generations to settle down, but never interferes with the B-heptomino or the lightweight spaceships.

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*.

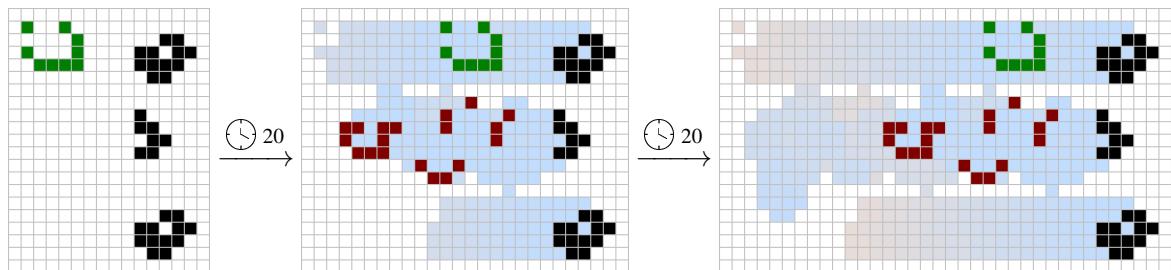


Figure 4.26: If we add an extra lightweight spaceship (displayed in green) to the puffer, its debris is suppressed, resulting in a spaceship called the *ecologist*. Even though the debris (displayed in red) dies off completely, it becomes somewhat large before doing so.

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.

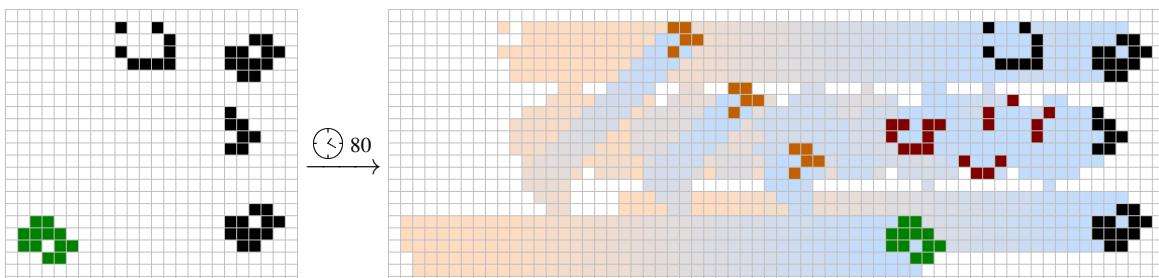


Figure 4.27: The (forward) space rake creates a forward-moving glider every 20 generations. It is constructed by adding yet another lightweight spaceship (displayed in green) to the ecologist in such a way as to transform its large spark into a glider.

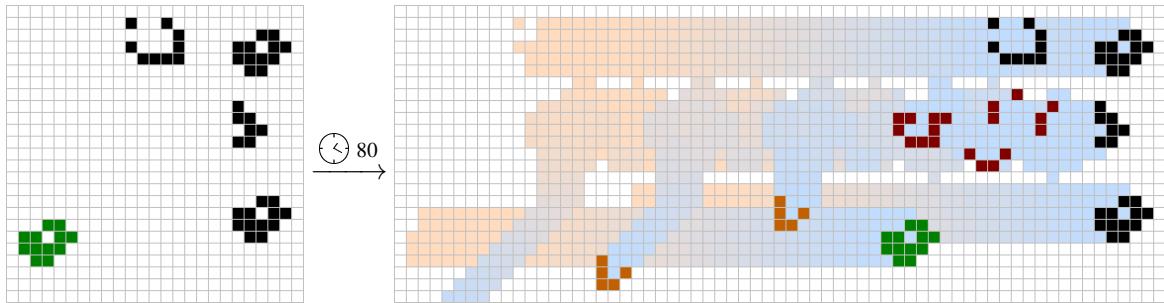


Figure 4.28: The *backward space rake* creates a backward-moving glider every 20 generations, using a slightly differently-positioned lightweight spaceship (displayed in green) than the forward space rake.

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 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

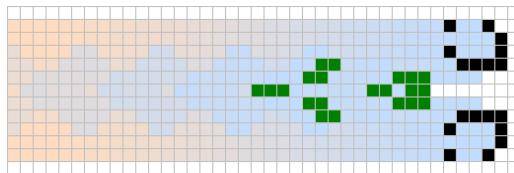


Figure 4.29: The *Schick engine* is a period 12 spaceship that consists of a pulsating tagalong (displayed in green) trailing behind two lightweight spaceships.

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.

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

¹⁹The 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 4.21).

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).

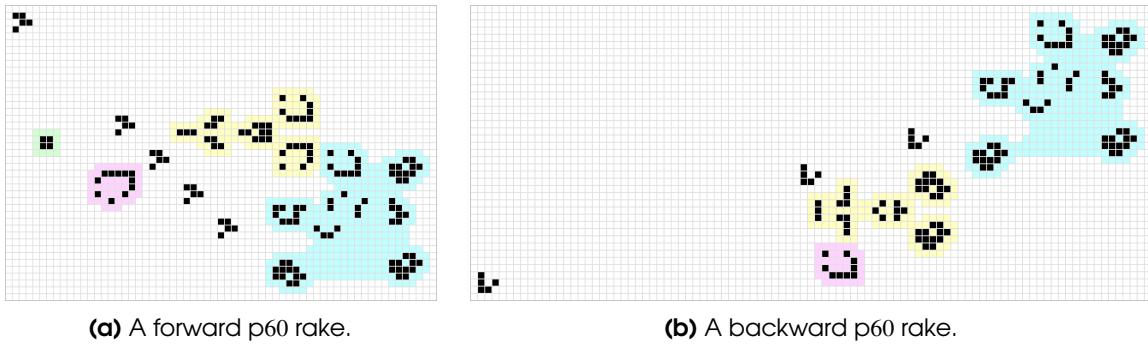


Figure 4.30: Rakes that emit one glider every 60 generations, based on the (a) forward and (b) backward space rakes (outlined in aqua) that emit one glider every 20 generations. A Schick engine (outlined in yellow) is positioned so that it destroys 1/3 of the gliders that pass by, leaves 1/3 of the gliders 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.

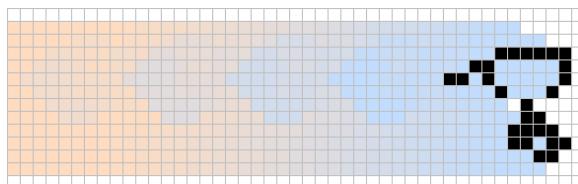


Figure 4.31: The *Coe ship* is a $c/2$ orthogonal period 16 spaceship that consists of some pulsating debris trailing behind a lightweight spaceship and a deformed heavyweight spaceship.

Most notably, we can add some light, middle, and heavyweight spaceships 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 continue travelling backward (for a total of 4 surviving gliders produced every 80 generations).

²⁰Named after Tim Coe, who found it in 1995.

²¹This configuration of two heavyweight spaceships works to turn any $c/2$ backward rake with period 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).

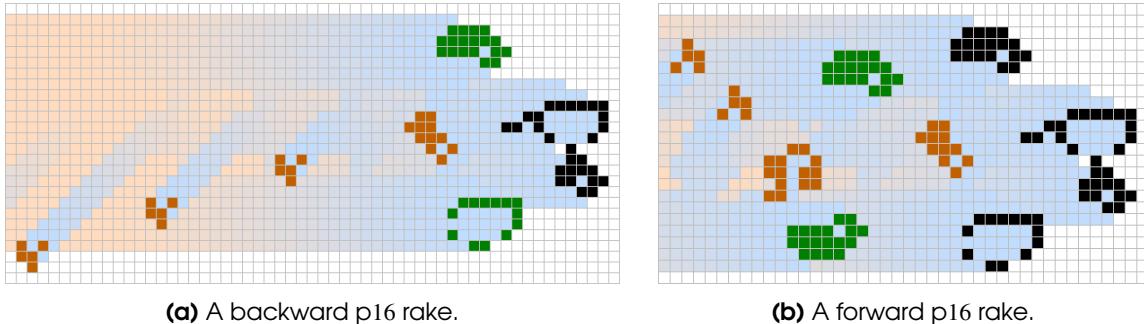


Figure 4.32: Period 16 (a) backward and (b) forward rakes constructed by using heavyweight spaceships (displayed in green) to interact with the spark behind a Coe ship.

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.

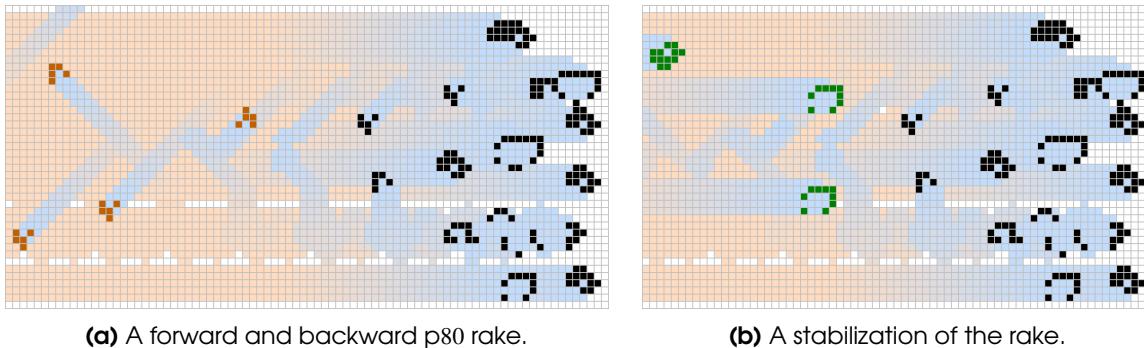


Figure 4.33: When a backward space rake and a backward Coe rake are carefully placed next to each other, their streams cross in such a way that they produce (a) 3 backward gliders and 1 forward glider every 80 generations. The (b) lightweight spaceships displayed in green destroy those 4 output gliders. Removing the top-right green LWSS results in a period 80 forward rake, whereas it 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 the LWSS, MWSS, and HWSS). 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 theorem shows that there is a reason for this: no faster spaceships exist.²²

Theorem 4.1 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.

²²This theorem was originally proved by Conway himself, very shortly after introducing the Game of Life.

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 grey 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 1, 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 grey cells in generation 0, it must be on and below the diagonal line of dark grey cells in generation 2.

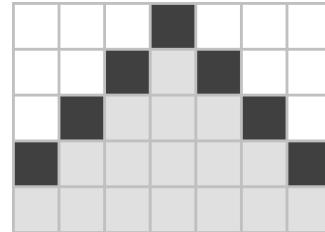


Figure 4.35: A diagram that illustrates Life’s orthogonal $c/2$ speed limit. If a pattern is contained within the light grey cells in generation 0, it must be on and below the diagonal lines of dark grey cells in generation 2.

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 grey cells in generation 0, then it will be on and below the diagonal line of dark grey cells in generation 2. By using this argument again, we see that the spaceship must be one 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 grey cells in generation 0, then we already showed that it must be on and below the diagonal lines defined by the dark grey 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 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.

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 strips must do so at lightspeed.²⁵

²³No object, whether in a vacuum or not, cannot possibly have a speed greater than c , since in one generation it can only affect its 8 neighbors.

²⁴Objects 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.

²⁵This theorem was originally proved by Dean Hickerson in 1993.

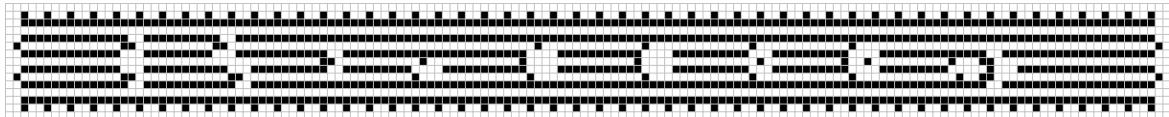


Figure 4.36: A collection of lightspeed signals that travel through zebra stripes. Each of the signals displayed here (i.e., the deformations in the middle of the stripes) travel to the right by 1 cell per generation. These signals were found (in no particular order) by Alan Hensel in 1995, Noam Elkies in 1997, Gabriel Nivasch in 1999, and in some cases, unknown Lifers in the early 1970's.

Theorem 4.2 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.

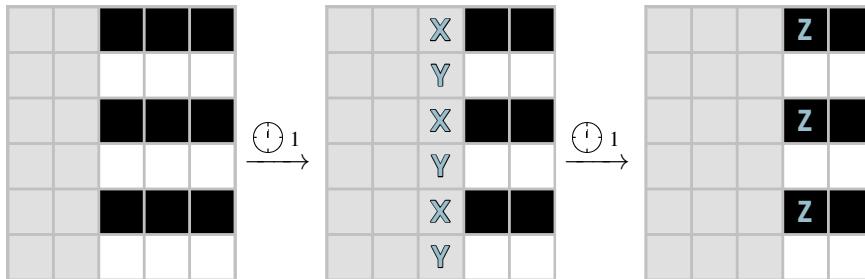


Figure 4.37: A diagram illustrating the fact that every signal that moves parallel through zebra stripes travels at a speed of c . If the signal is contained within the region of light grey cells and its leading edge ever moves to the right (denoted by the Xs and Ys in the middle generation), then it must continue moving by 1 cell every generation.

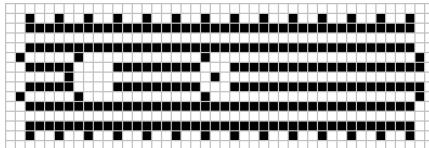
First note that all of the cells marked Y must be dead in generation 1, since they have at least 4 live neighbors in generation 0. But then each of the cells marked X must be alive in generation 1 or else the cells marked Z would be dead in generation 2 due to only having one live neighbor. We have thus 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. ■

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.

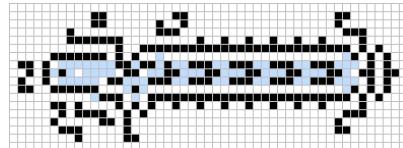
On the other hand, no reasonably small signal elbows (for any signal) are known to date, despite a considerable amount of effort on the part of Life enthusiasts.²⁷ The discovery of a quick-recovering elbow would be a huge discovery, since we could use it to move a signal around in a loop—much like we used reflectors to move a glider around a loop in Section 3.5—creating oscillators with any

²⁶Just like signals are analogous to spaceships, there is also an analogy between sources and glider guns, sinks and eaters, and signal elbows and reflectors.

²⁷Very 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.



(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.

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

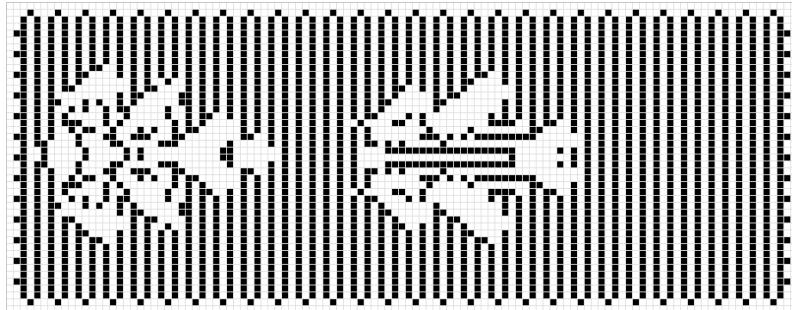


Figure 4.39: Some signals that travel perpendicularly through zebra stripes at a speed of $2c/3$ (to the right). Both of these signals were found by Hartmut Holzwart in 2006.

Theorem 4.3 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 grey 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.

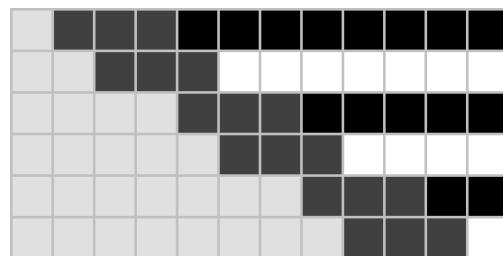
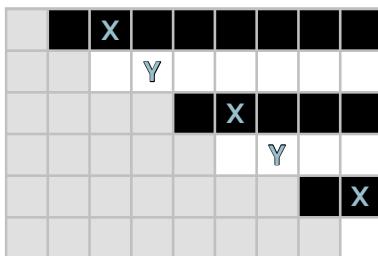


Figure 4.40: A diagram illustrating the $2c/3$ speed limit for a finite signal travelling perpendicularly through zebra stripes. If a signal is contained within the light grey cells in generation 0, then cells X and Y (left) cannot change state in generation 1, so it must be on and to the left of the dark grey cells (right) in generation 3.

²⁸This theorem was originally proved by Hartmut Holzwart in 2006.

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 grey 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 grey 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 grey region in Figure 4.40 in generation 0 will be located on and to the left of the dark grey cells in generation 3. Since the region containing the dark grey cells is just the original light grey 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 possible speed of signal 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 speed (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. It is worth noting that to date the $2c/3$ diagonal signal is the fastest one known—there are currently no known diagonal signals that travel through a stable (p1) wire at the speed of light.

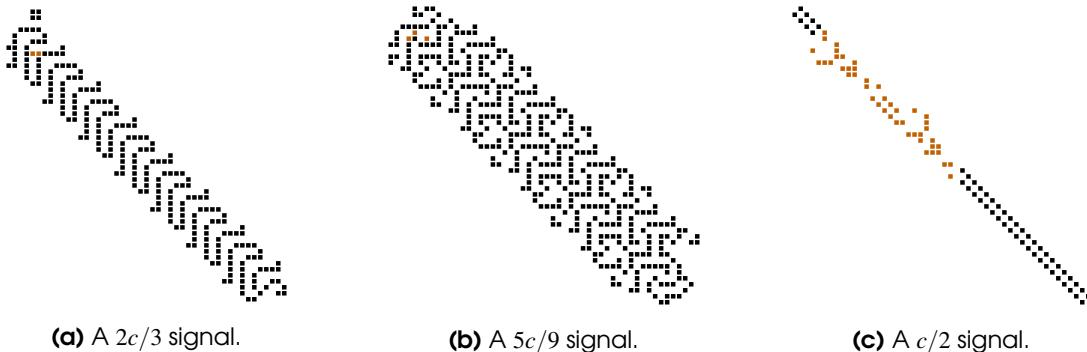


Figure 4.41: Some diagonal signals (travelling to the bottom-right) that were found by (a,b) Dean Hickerson in 1997 and (c) Hartmut Holzwart in 2003. The signals themselves are sometimes difficult to distinguish from the surrounding wire, so they are highlighted in orange. Note that both halves of the $c/2$ signal are indeed needed, since the wire is offset by 1 cell between the two half signals.

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 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.

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

²⁹The 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.

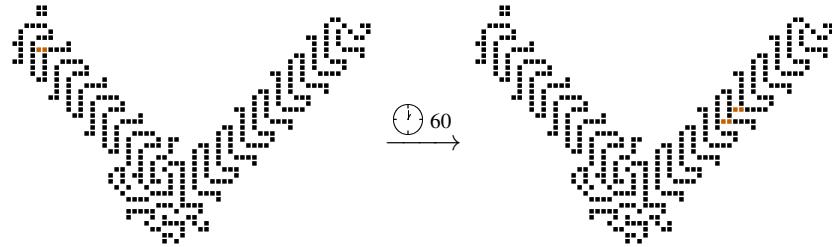


Figure 4.42: A corner that is able to reflect the $2c/3$ diagonal signal (highlighted in orange) by 90-degrees, but which also creates a second copy of that signal in the process and thus cannot be used twice.

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.

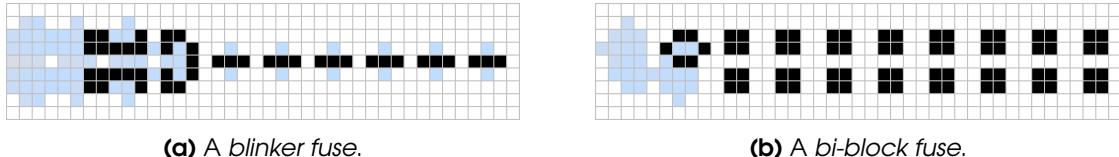


Figure 4.43: Two $2c/3$ orthogonal fuses with (a) period 18 and (b) period 12.

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 gliders are present, then the lightweight spaceship reappears 6 generations later, 11 cells in front of its original position.

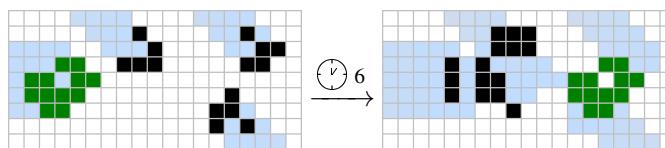


Figure 4.44: The *fast forward force field*: if the lightweight spaceship (displayed in green) is not present, the 3 gliders destroy each other and leave nothing behind, but in the configuration displayed here they “teleport” the LWSS to the right by 11 cells in just 6 generations. The leftover debris on the right simply dies off in a few more generations.

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.

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.

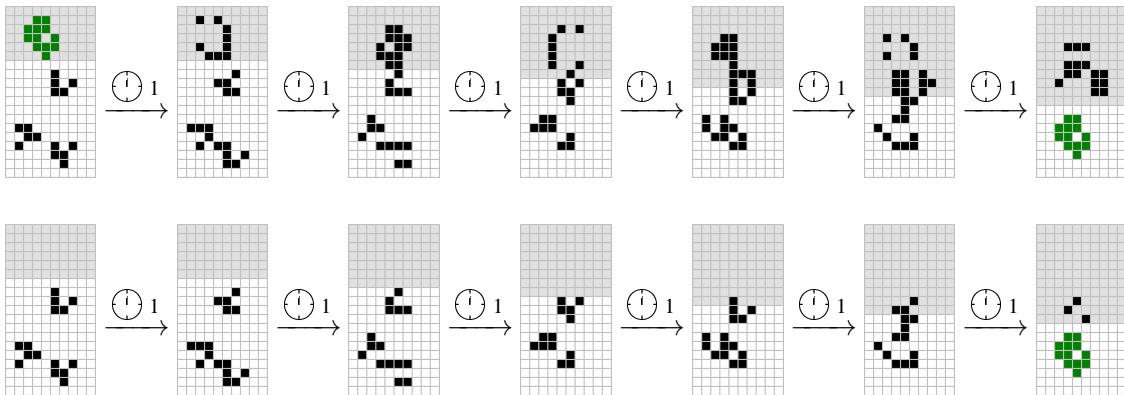


Figure 4.45: The fast forward force field with an incoming LWSS (top row) and without an incoming LWSS (bottom row). The lowest row of cells affected by the LWSS is displayed with a light gray background, and this leading row never progresses more than 1 cell per generation, showing that this reaction 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.

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 $20c/24 = 5c/6$.

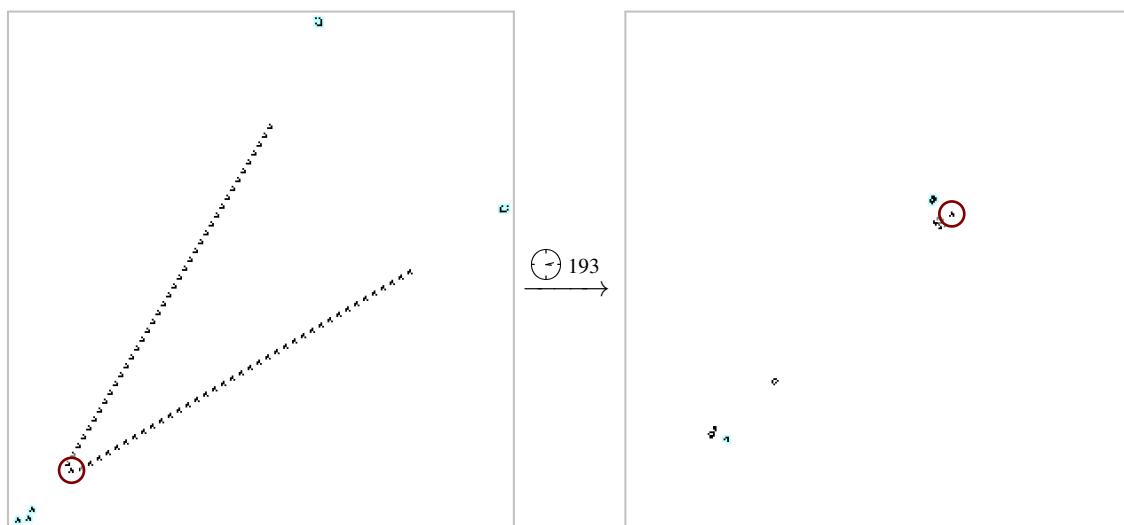


Figure 4.46: A diagonal collision of gliders, found by Jason Summers in 1999, that is able to teleport a glider from the bottom-left (circled in red) to the top-right at the speed of light. If the input glider is not present, the reaction just destroys itself, leaving nothing behind. The gliders and 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.

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.

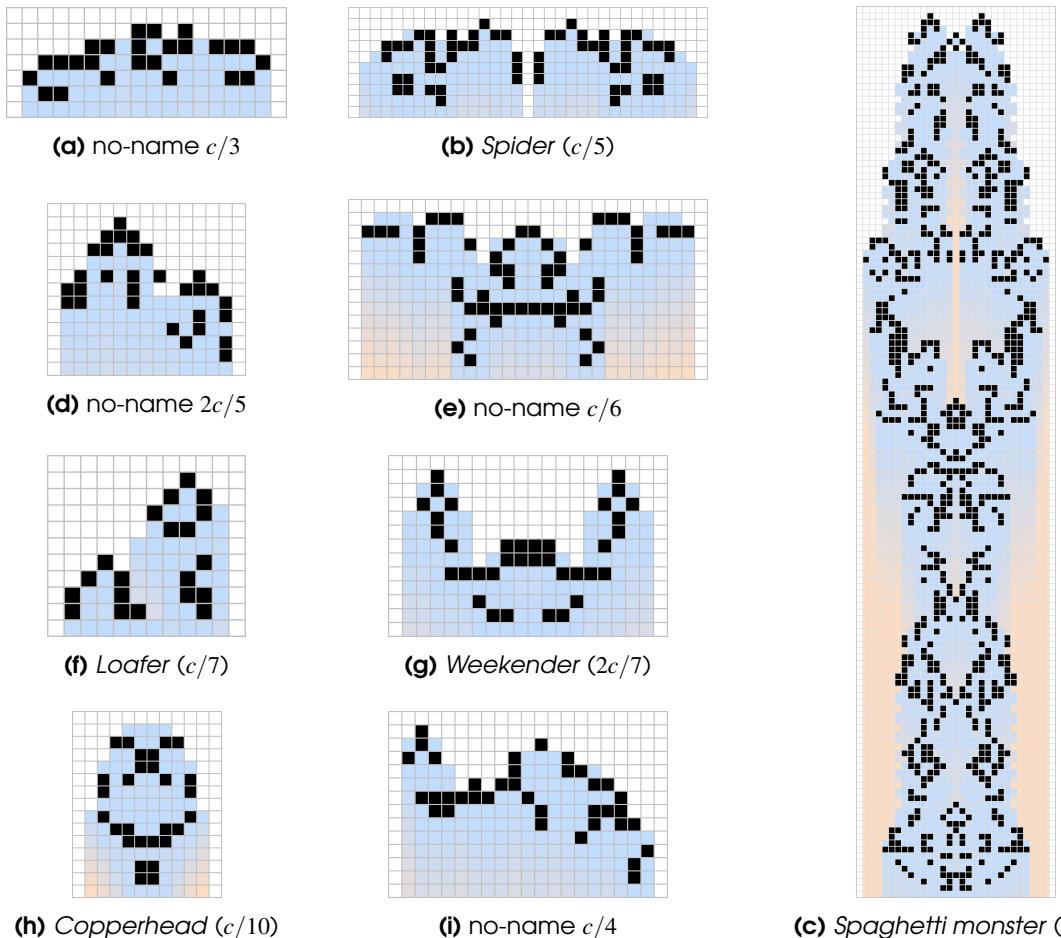


Figure 4.47: A collection of small orthogonal spaceships of various speeds, all oriented so that they travel up. These spaceships were found by (a) Dean Hickerson in 1989, (b) David Bell in 1997, (c) Tim Coe in 2016, (d) Paul Tooke in 2000, (e) Hartmut Holzwart in 2009, Josh Ball (f) in 2013 and (i) in 2012, (g) David Eppstein in 2000, and (h) ConwayLife.com forum user “zdr” in 2016.

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 than it is for low-period objects,³³ so the effectiveness of computer searches drops off quickly.

Nonetheless, 3 new diagonal spaceships, each travelling at a speed that we have not yet seen, are displayed in Figure 4.48. Together with the $c/4$ and $c/12$ spaceships that we are already familiar with, these spaceships represent the only 5 diagonal speeds that have been attained by elementary spaceships.

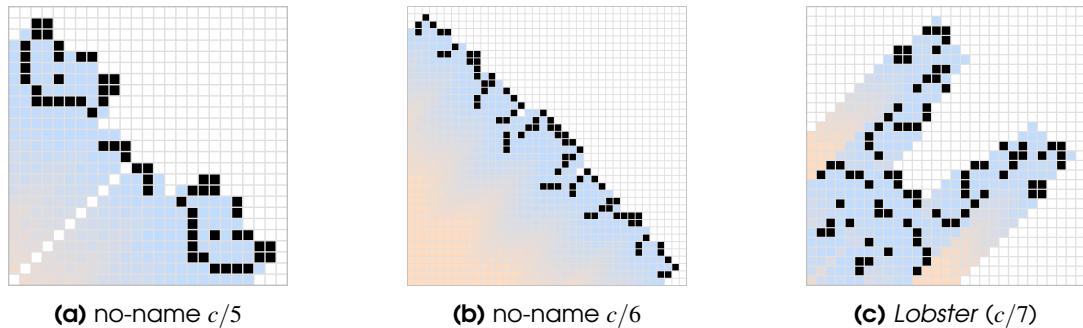


Figure 4.48: The smallest known diagonal spaceships of some unusual speeds, all oriented so that they travel toward the top-right. These spaceships were found by (a) Matthias Merzenich in 2010 and (c) in 2011, and by (b) Josh Ball in 2011.

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, one elementary oblique spaceship is indeed known, and is displayed in Figure 4.49.³⁴

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.

³¹We 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.

³²This definition is admittedly vague, and essentially impossible to make precise.

³³This is the same reason that computer searches have been so effective at finding oscillators with period below 19, but not above.

³⁴Ships 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.

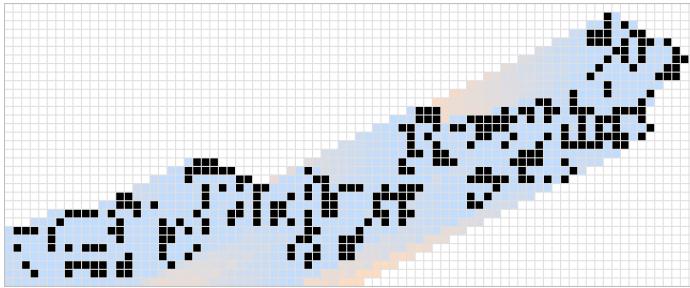


Figure 4.49: *Sir Robin* is an oblique spaceship with speed $(2, 1)c/6$. It was found by Adam P. Goucher in March 2018, based on a partial spaceship that was found by Tomas Rokicki.

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(i)
$c/5$	orthogonal	spider
$2c/5$	orthogonal	Figure 4.47(d)
$c/6$	orthogonal	Figure 4.47(e)
$c/7$	orthogonal	loafer
$2c/7$	orthogonal	weekender
$3c/7$	orthogonal	spaghetti monster
$c/10$	orthogonal	copperhead
$17c/45$	orthogonal	“caterpillar” engineered spaceship
$31c/240$	orthogonal	“centipede” engineered spaceship
all speeds $< c/4$	orthogonal	“caterloopillar” engineered spaceships
$c/4$	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 $< 17c/792$	diagonal	“Demonoid” engineered spaceships ³⁵
$(2, 1)c/6$	slope 2	Sir Robin
$(6, 3)c/2621440$	slope 2	“half-baked knightship”
all speeds $< (1, 1)c/580$	all slopes	“Gemini” engineered spaceships ³⁶

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.

³⁵All speeds slower than $17c/792$ are known to be attainable using a “Demonoid” construction, which we will discuss in Section 11.1. However, this construction has not been explicitly carried out, as the size of the resulting spaceship becomes monstrous as the speed approaches the $17c/792$ limit. The fastest spaceship explicitly built using this method has speed $65c/438852$.

³⁶While essentially any direction and sufficiently slow speed can be attained by a Gemini-like spaceship, most speeds that have been explicitly constructed in this way are in the ballpark of $c/6500$.

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 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 simplest unknown diagonal speed is $c/8$, for which there could potentially exist a period 8 spaceship, whereas all other unknown speeds would necessarily have at least period 9.

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).

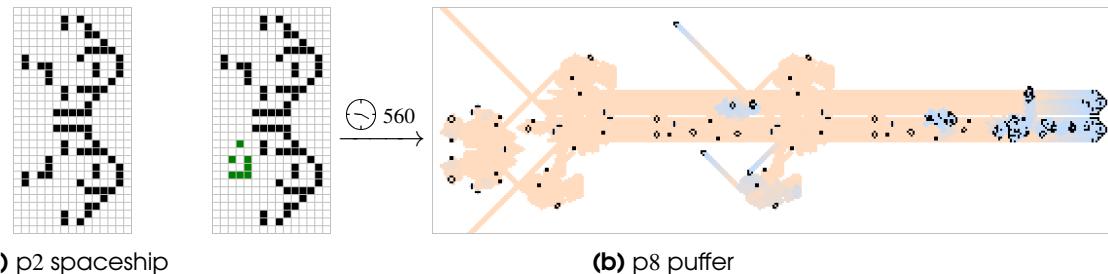


Figure 4.50: (a) A small period 2 spaceship that was found by Dean Hickerson in 1989, and (b) a puffer that can be obtained from this spaceship by changing the shape of one of its rear sparks (displayed in green).

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 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).³⁷

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 $c/2$),

³⁷We will discuss which glider collisions produce which objects thoroughly in Chapter 5.

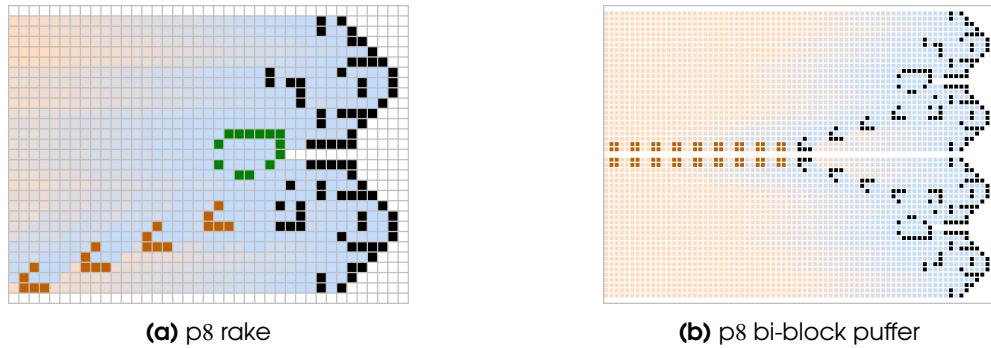


Figure 4.51: (a) A small period 8 rake constructed by using a heavyweight spaceship (displayed in green) to tame the debris left behind by the puffer from Figure 4.50(b), and (b) a way of arranging two of these rakes so as to leave a trail of bi-blocks behind them.

stop burning, and the wick would start being re-constructed 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 bi-block between the front and back halves of the rake increases its period by 32.³⁸ 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 4.12.³⁹

Another method of constructing adjustable-period spaceships,⁴⁰ 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.

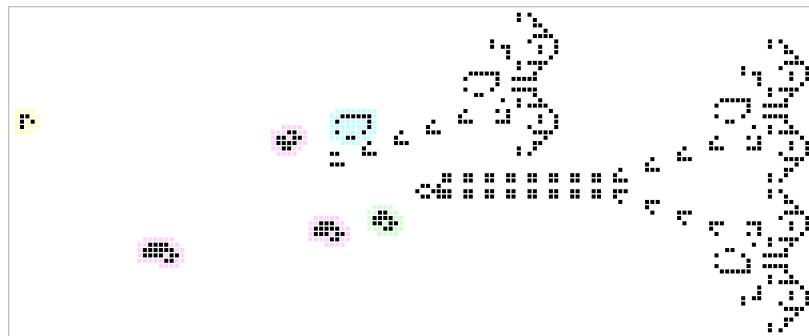


Figure 4.52: An *adjustable rake*, which can be made to have any period of the form $264 + 32n$, where $n \geq 0$ is an integer, by increasing the length of the bi-block wick in the middle. In the form displayed here, it has period 392. The HWSS highlighted in aqua destroys the gliders in the stream 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.

³⁸Each 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.

³⁹A slightly smaller method of turning this rake into a spaceship is described in Exercise 4.34.

⁴⁰Developed by David Bell in 1992, with help from Dean Hickerson.

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 flotillas 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).⁴¹

Hickerson's search program continued to find new types of spaceships throughout the year, including the first $c/3$ spaceship (in Figure 4.47(a)) 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.⁴²

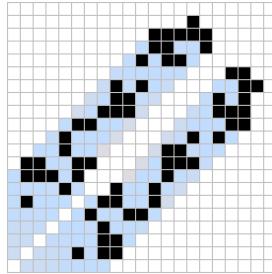


Figure 4.53: The *big glider*: a $c/4$ period 4 diagonal spaceship, and the first diagonal spaceship other than the glider to be discovered. Found by Dean Hickerson in December 1989.

Some other Life enthusiasts continued using search programs to find new spaceships throughout 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; 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)).⁴³

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 one (Sir Robin) not being found until 2018 (by Adam P. Goucher and Tomas Rokicki), despite considerable effort having been put into finding one over the preceding 20 years. Indeed, there was a remarkably close call in March 2004,

⁴¹It is not known exactly which period 2 spaceship was found first—they were all found very close together by the same computer searches.

⁴²His algorithm is now implemented in programs called lifesrc and JavaLifeSearch. See conwaylife.com/wiki/Lifesrc for documentation and download locations.

⁴³However, 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.

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.

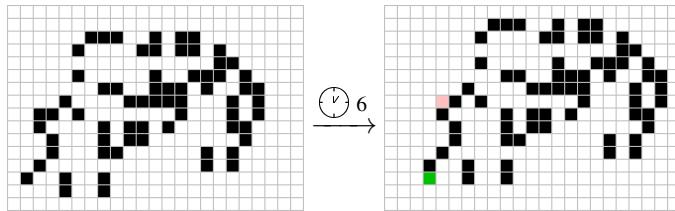


Figure 4.54: An “almost knightship” that travels to the right by 2 cells and up by 1 cell over the course of 6 generations, except with one extra cell born (shown in green) and one extra cell dead (shown in red).

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. 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).

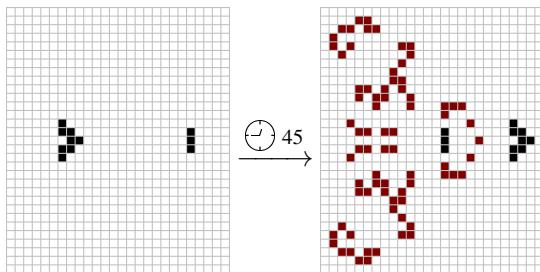


Figure 4.55: A reaction in which a pi-heptomino collides with a blinker in such a way as to move forward by 17 cells in 45 generations while moving the blinker backward by 6 cells. The cells displayed in red on the right die off completely in another 24 generations.

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 $(5120, 1024)c/33699586$ 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.⁴⁴ Several other types of massive 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. Indeed, 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

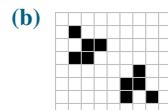
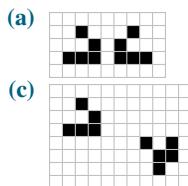
⁴⁴The existence of oblique spaceships and arbitrarily-slow spaceships was already known in 1982 [BCG82], but Gemini was the first explicit construction. Despite having over 800,000 live cells, it was orders of magnitude smaller and faster than such a spaceship was expected to be.

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 305

- *4.1** Determine whether the given pair of gliders have the same or the opposite color as each other.



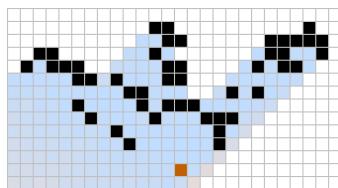
- *4.2** Determine whether the specified glider reflector is color-preserving or color-changing.

- (a) Buckaroo (see Figure 3.16(b)).
- (b) Relay (see Figure 3.28).
- (c) Boojum reflector (see Exercise 3.20(a)).
- (d) Rectifier (see Exercise 3.20(b)).
- (e) Bouncer reflectors (see Figure 6.34).

- 4.3** Recall the tubstretcher that we introduced in Figure 4.10.

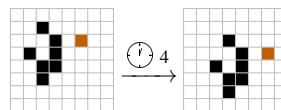
- (a) Modify the tubstretcher so that it stretches a boat instead of a tub.
- (b) Modify the tubstretcher so that it stretches two tubs instead of just one.
- (c) The pattern that you constructed in part (b) grows by 4 cells every 4 generations. Place another period 4 object (either a spaceship or an oscillator) on the Life plane so that the total number of live cells on the plane grows by exactly 1 every generation.

- *4.4** The following $c/4$ diagonal spaceship is called a *swan*,⁴⁵ and it emits a very accessible spark.

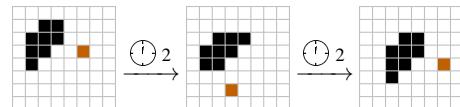


- (a) Use this spark to create a tubstretcher.
- (b) Find a way of colliding a glider with this spark so that two gliders are produced—one traveling southeast and one traveling southwest.

- 4.5** The $c/4$ diagonal tagalong displayed below can be attached to almost any $c/4$ diagonal spaceship that emits a spark. Attach it to three different spaceships.



- 4.6** 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.

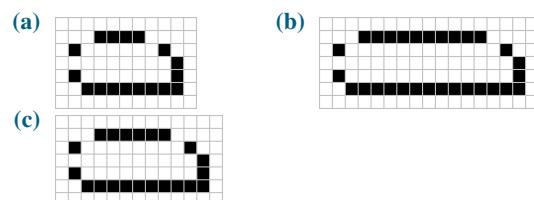


- (a) Find a way to place this tagalong between two spaceships.
- (b) This tagalong is called the *glider emulator*, since it leaves behind a spark that behaves very similarly trailing cell of a glider. Use this spark to attach a copy of the Canada goose tagalong to the spaceship that you constructed in part (a).

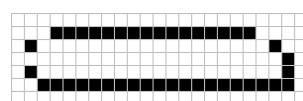
- 4.7** Show how two copies of the tagalong in Figure 4.14(b) can stabilize each other, resulting in a $c/2$ orthogonal spaceship that does not contain a LWSS, MWSS, or HWSS.

- 4.8** There are 5 different flotillas that consist of exactly two middleweight spaceships. Find them all.

- *4.9** Use exactly two heavyweight spaceships to stabilize each of the following overweight spaceships, turning them into flotillas.

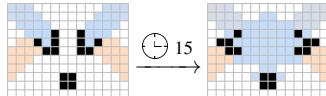


- *4.10** Use (potentially many) light, middle, heavy, and/or overweight spaceships to create a flotilla that includes the following overweight spaceship:



⁴⁵Found by Tim Coe in 1996.

4.11 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).

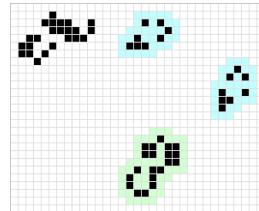
- (a) Create a hivenudger that uses a lightweight spaceship, a middleweight spaceship, and 2 heavyweight spaceships.
- (b) Create a hivenudger that uses a Coe ship as one of its rear corners.
- (c) How many different hivenudgers can be constructed by using light, middle, and heavyweight spaceships at its four corners? [Hint: The answer is not $3^4 = 81$. Try either listing all possibilities or looking up a mathematical result called Burnside's lemma, which can be used to compute the answer.]

4.13 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.

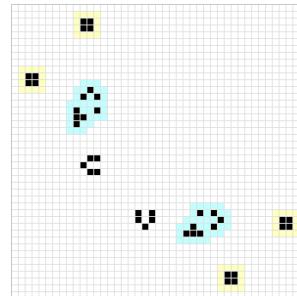
- (a) Explain why it is *not* necessary to consider the color of a loafer that we reflect around a track.
- (b) Explain why it *is* useful to consider the color of a light, middle, or heavyweight spaceship that we reflect around a track. Why are we more restricted when reflecting these spaceships than when reflecting loafers?

4.14 Create a new puffer by replacing one of the lightweight spaceships in Figure 4.25 with a middleweight spaceship.

***4.15** 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 using this reaction.]



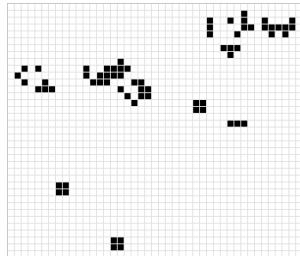
***4.16** A reaction is displayed below in which two switch engines (highlighted in aqua) bounce off of each other and destroy some trails of blocks (highlighted in yellow).



- (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.

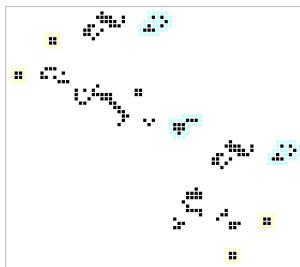
⁴⁶First constructed by Michael Simkin in November 2014.

4.17 Here is an arrangement of two switch engines that evolves into a puffer for a single diagonal row of blocks:



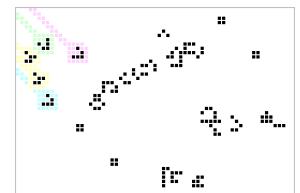
- (a) Find a position for a second copy of this puffer diagonally behind the first one, such that the blocks from the first puffer suppress the creation of blocks in the second puffer, resulting in an adjustable-length 4-engine Cordership.⁴⁶
- (b) Replace the second puffer with a diagonal mirror-image 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** 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.

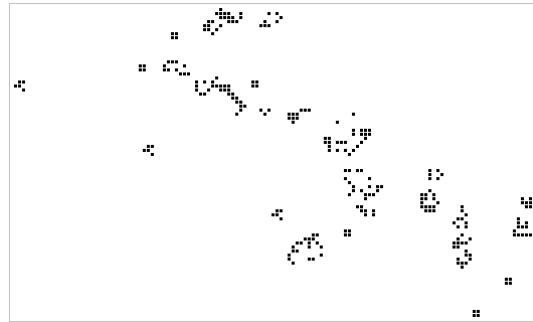
***4.19** A Cordership that makes use of just 2 switch engines⁴⁷ is displayed below, along with several incoming gliders that it can reflect in different ways.



- (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.

***4.20** A *Corderrake* is displayed below, which is a $c/12$ rake based on switch engines that shoots gliders sideways as it moves.⁴⁸



- (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).⁴⁹
- (c) Use this rake, together with other Corderships that we have seen in this chapter, to construct a diagonal rake that shoots lightweight spaceships.

***4.21** 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?

4.22 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.

***4.23** 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.

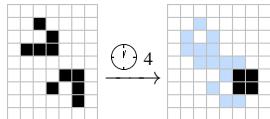
4.24 Create a period 240 forward rake and a period 240 backward rake by replacing the period 20 space rake with the period 60 variant in Figures 4.33(a) and 4.33(b).

⁴⁷Found by Aidan F. Pierce in December 2017.

⁴⁸Both the Corderrake and the 3-engine Cordership were found by Paul Tooke in 2004.

⁴⁹Objects 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.

4.25 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.

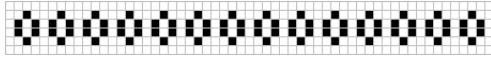
[Hint: If you have trouble finding a two-glider collision that works for you, jump ahead to Table 5.1.]

4.26 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 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) Place a live cell near one end of this beehive wick so as to make a $4c/5$ fuse.
- (b) Find a $4c/5$ fuse that *cleanly* burns through this wick.⁵⁰ [Hint: Try placing some *symmetric* debris near the end of the wick.]

4.29 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** 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 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** Show how the $c/5$ diagonal spaceship in Figure 4.48(a) can be used to reflect a glider by 90 degrees.

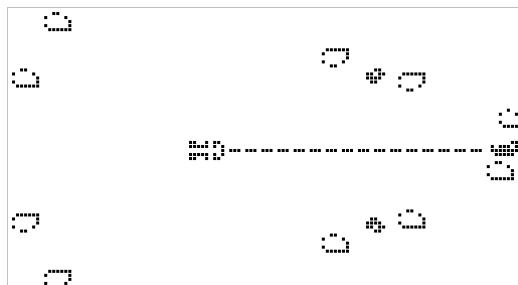
4.33 In Figure 4.51(a) we showed a period 8 rake. What is the smallest period that a rake could conceivably have (i.e., without gliders colliding)?

4.34 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 Create a rake with period equal to exactly 1000.

4.36 The spaceship displayed below has period 1136.⁵¹



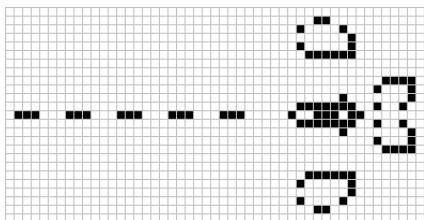
- (a) Watch this spaceship evolve and describe how it works. For example, which part of the pattern lays the blinker wick? What effect do the heavyweight spaceships at the back have?

⁵⁰This fuse was originally found by Dean Hickerson in June 1993.

⁵¹This spaceship was found independently by David Bell and Dean Hickerson in 1992.

- (b) Show how the rear lightweight and heavyweight spaceships can be moved so as to increase this spaceship's period. Use this method to explicitly construct a spaceship with period greater than 1500.

4.37 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) Explain why the reaction from that same figure in which an MWSS destroys a blinker cannot be used to delete the blinker fuse.
- (c) 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.

Early draft (May 19, 2020).

Not for public dissemination.

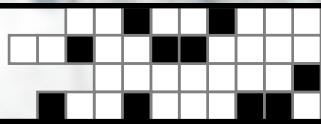
Circuitry and Logic

5	Glider Synthesis	113
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	
	Notes and Historical Remarks	
	Exercises	
6	Periodic Circuitry	143
6.1	Period 30 Circuitry	
6.2	Primer	
6.3	Period 46 Circuitry	
6.4	Bumpers and Bouncers	
6.5	Glider Timing and Regulators	
	Notes and Historical Remarks	
	Exercises	
7	Stable Circuitry	171
7.1	Herschel Conduits	
7.2	Gliders to Herschels and Back	
7.3	Glider-to-Herschel converters	
7.4	Synthesizing Objects via Conduits	
7.5	Period Multipliers and Small High-Period Guns	
7.6	Converters for Other Objects	
7.7	Factories	
	Notes and Historical Remarks	
	Exercises	
8	Guns and Glider Streams	201
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 Construction	
8.8	Slow and Irregular Guns	
	Exercises	

Early draft (May 19, 2020).

Not for public dissemination.

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.

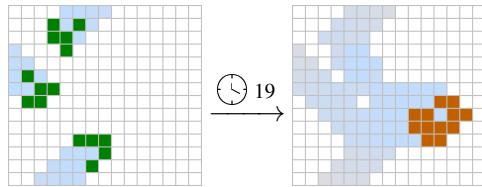


Figure 5.1: Three gliders colliding in such a way as to create a lightweight spaceship.

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 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 catalogue of glider syntheses, the world of Life will open up considerably for

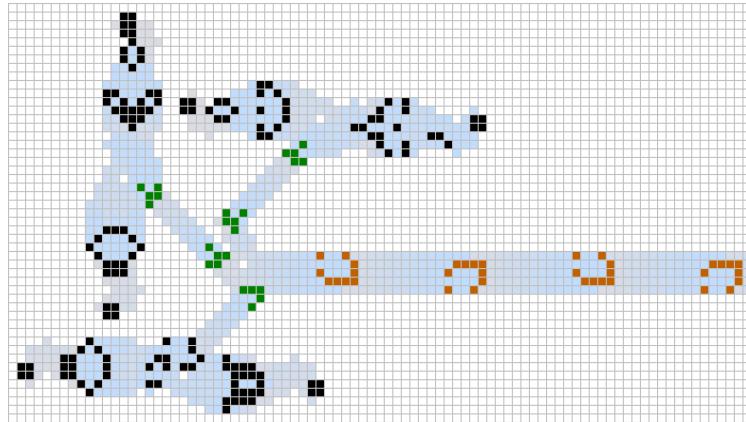


Figure 5.2: A lightweight spaceship gun constructed by using three Gosper glider guns (top-right, top-left, and bottom-left) to shoot three gliders at each other, which collide in the orientation depicted in Figure 5.1 in order to create a lightweight spaceship.

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.

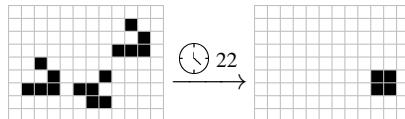


Figure 5.3: A collision of three gliders that produces a block. However, if the two rightmost gliders came from farther away, they would collide with each other before reaching the displayed configuration, so this is not a valid 3-glider synthesis.

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, 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 are essential building blocks of complex patterns, such as eater 1 and the B-heptomino (which

¹There 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.

Result	2-Glider Collisions						Result	Collision
block								
honey farm								
blinker								
traffic light								
pi-hept.								
B-hept.								
loaf + blinker								
glider								
pond								
misc.								
nothing								

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 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 is called the *kickback reaction*, since it produces an output glider traveling in a different direction than either of the input gliders.

we recall evolves into a Herschel). 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 that 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).

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.

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. Some of the more interesting three-glider syntheses that are known are presented in Table 5.2. However, we do stress that this table is not complete: there are known 3-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 synthesis of a single glider in Table 5.2 might seem somewhat silly at first glance, it has the interesting property that the synthesized glider travels perpendicular to each of the input gliders³—a property that is not shared by any of its 2-glider syntheses. For this reason, it is 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).

Glider syntheses of spaceships (such as the three-glider syntheses of the lightweight, middleweight, or heavyweight spaceships) can be used to construct guns for these spaceships, as we already saw at the start of this chapter. The 3-glider collision that creates a glider-producing switch engine is also 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, so 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 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. Similarly, even though we can use 3 gliders to synthesize a glider-producing switch engine (plus lots of debris), we haven't yet seen how to synthesize a switch engine by itself. Again, as far as we know this 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 adjacent Gosper glider guns. Thus 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

²As an example, the pentadecathlon was not known to be synthesizable with only three gliders until Heinrich Koenig found the synthesis given in Table 5.2 in April 1997. Similarly, it was not known how to create *any* infinitely-growing object using just 3 gliders until Michael Simkin found the displayed 3-glider collision in October 2014.

³Even more specifically, two of the gliders collide in such a way as to destroy each other, but they create a banana spark in the process that reflects the third glider as in Figure 3.17.

⁴Synthesizing the Gosper glider gun “piece-by-piece” like this is the quite straightforward, and requires 10 gliders. However, slightly smaller syntheses of the Gosper glider gun are known, requiring as few as 8 gliders.

result	3-glider collisions					
lightweight spaceship						
middleweight spaceship						
heavyweight spaceship						
glider						
pulsar						
pentadecathlon						
R-pentomino						
queen bee						
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.

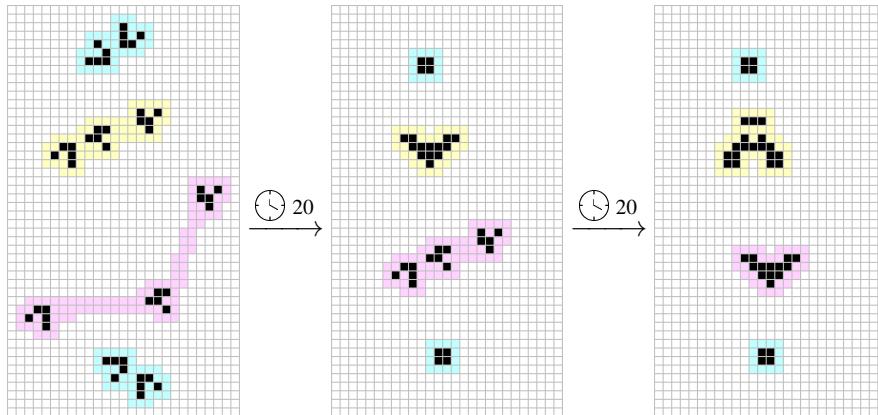


Figure 5.4: A ten-glider synthesis of a Gosper glider gun, which is composed of two syntheses of blocks (highlighted with an aqua background) and two syntheses of queen bees. The queen bee synthesis with the magenta background is the same as the one with the yellow background, but delayed by 20 generations in order to make the queen bees collide in their proper phases.

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.

result	glider syntheses	result	synthesis

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).

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, problems can arise. If the gliders in a synthesis are very close together, it can be very difficult to pack guns or rakes together tightly enough to produce the desired stream of gliders. For example, even the task of creating a heavyweight spaceship gun using the 3-glider synthesis provided in Table 5.2 is quite challenging (see Exercise 5.8). Trying to actually make use of the ten-glider synthesis of a Gosper glider gun displayed in Figure 5.4 would similarly be challenging, as there are just too many streams of gliders travelling along paths that are close together in the same direction.

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.

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.

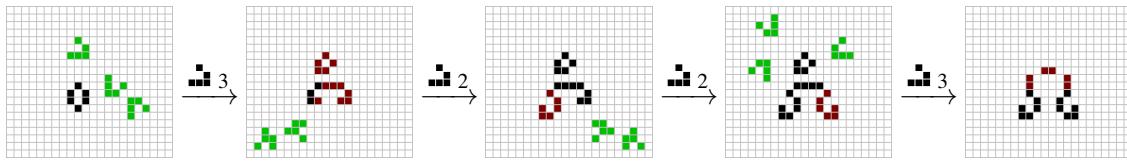


Figure 5.5: A 12-glider incremental synthesis of a fumarole (2 gliders that are not shown are required to synthesize the initial beehive). The cells that were created or modified in the previous step of the synthesis are shown in red, and the gliders that will be involved in the next collision are shown in green.

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.

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 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 32,538 strict still lifes with 18 or fewer live cells. Mark Niemiec maintains a large database of glider syntheses that is available at codercontest.com/mniemiec/lifepage.htm.

5.4 Synthesis of Moving Objects

Incremental synthesis is the 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 more 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.

result	last step of incremental synthesis		
			
ship			
			
tub			
			
tub with tail			
			
hat			
			
queen bee			
			
T-tetromino / traffic light			
			
honey farm			
			
snake			
			
LWSS			
			
MWSS			
			
HWSS			
			
pentadecathlon			

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.

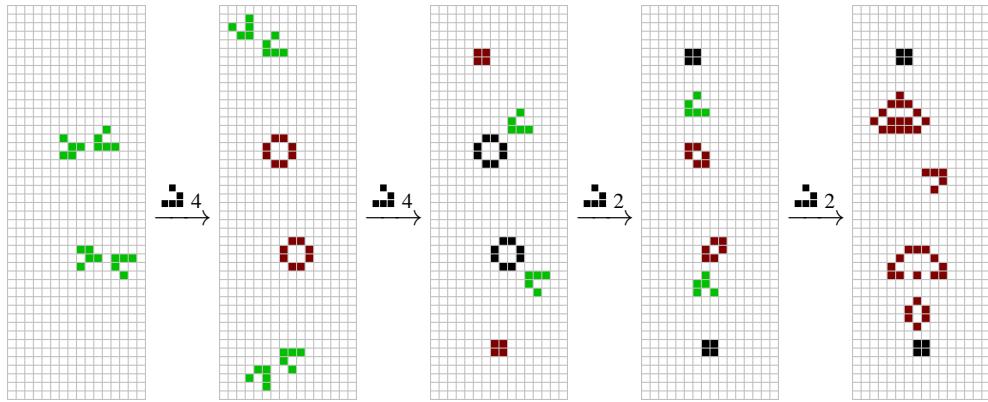


Figure 5.6: An incremental synthesis of the Gosper glider gun. Each stage in the synthesis works by either using two gliders to construct a simple still life or using just one glider to transform one object into another one. The objects that were created or modified in the previous step of the synthesis are shown in red, and the gliders that will be involved in the next collision are shown in green.

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.

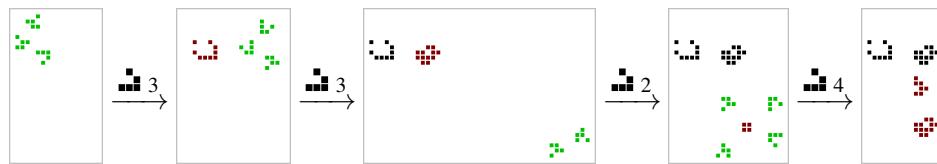


Figure 5.7: An incremental synthesis of the ecologist. The first two steps simply use one of the three-glider syntheses of lightweight spaceships, and the third step uses a two-glider synthesis of a block. The fourth step is slightly more complicated, since we have to construct the B-heptomino at the 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 form one of the incremental syntheses of a lightweight spaceship that we saw in Table 5.4.

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 collision that results in a glider travelling in a different direction). The tricky final stage of this synthesis is made explicit in Figure 5.8.

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).

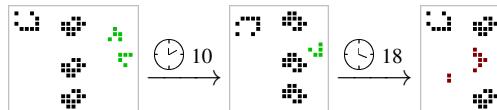


Figure 5.8: Another incremental synthesis of the ecologist. This synthesis has the advantage of never using more than two gliders at a time, and they can all arrive from the lower-left and lower-right (see Exercise 5.17). The top-left glider looks like it might collide with the lightweight spaceships before getting into the indicated position, but because the lightweight spaceships are moving in the same direction as the glider, they never actually cross paths.

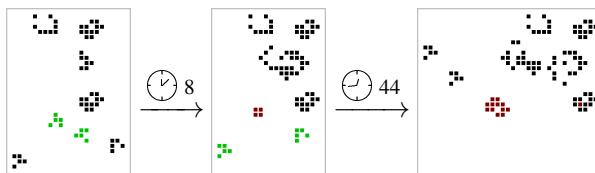


Figure 5.9: An incremental glider synthesis of a (forward) space rake. The same technique can be used to synthesize the backward space rake (see Exercise 5.19).

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 4.21). Thus the various syntheses of lightweight spaceships and T-tetrominoes that we have seen can 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.

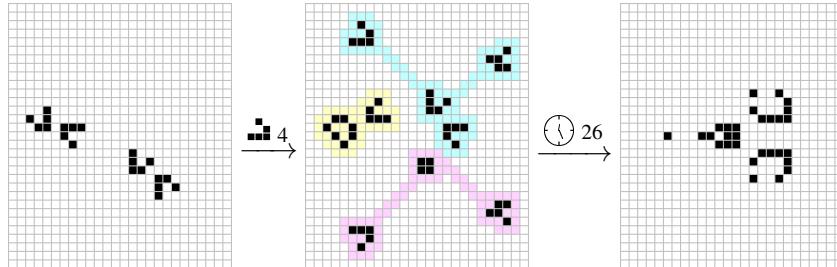


Figure 5.10: A glider synthesis of a Schick engine. Lightweight spaceships are created by the magenta and aqua glider collisions, and a T-tetromino (which becomes the Schick engine's trailing spark) is created by the loaf + glider collision outlined in yellow.

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 synthesize a Schick engine next to it in the appropriate position, or vice-versa (see Exercise 5.18(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 in Figure 5.11.

5.4.3 Corderships

Corderships form 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, and each switch engine requires at least four gliders to synthesize. Thus the 10-engine Cordership from Figure 4.23 would require at least 40 gliders to synthesize, and

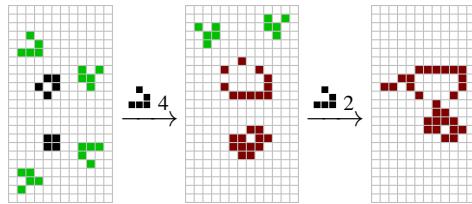


Figure 5.11: An incremental glider synthesis of a Coe ship, which works by synthesizing a lightweight spaceship next to a middleweight spaceship, and then transforming that middleweight spaceship into the necessary deformed heavyweight spaceship.

the 6 front and middle switch engines would have to be constructed at almost the exact same time, as would the 4 rear switch engines.⁵

With these difficulties in mind, it seems desirable to try synthesizing a Cordership that uses as few switch engines as possible—the 3-engine Cordership from Figure ???. Even just with 3 switch engines to synthesize, it is somewhat challenging to find an arrangement of switch engine syntheses that can be placed close enough together without the gliders that are synthesizing different switch engines interfering with each other. Thus we use the 3-direction syntheses of the switch engine rather than the 4-direction synthesis so as to minimize the glider crossings that we have to worry about. In particular, we use the second synthesis listed in Table 5.3 to construct the central switch engine, and we use the third synthesis to construct the other two switch engines.

Unfortunately, one more small problem arises after we use these 12 gliders. While the Cordership is indeed synthesized correctly, the debris from the switch engines take a few generations to sync up with each other, and thus some unwanted junk is left behind. To prevent this junk from forming, we use three extra gliders—one for each switch engine—to suppress the debris until it matches up properly with the debris from the other switch engines. Altogether, this results in the 15-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 cataloguing what objects were created, and (b) syntheses that can be made simply by piecing together the syntheses of type (a). Let’s now start to branch out and 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 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.

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 creation: Rich’s p16 (plus a lot of extra debris) is created when two honeyfarm predecessors and two B-heptominoes collide with two blocks and two beehives (see Figure 5.14).

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

⁵Despite these obstacles, Stephen Silver created a 60-glider synthesis of the 10-engine Cordership in 1998.

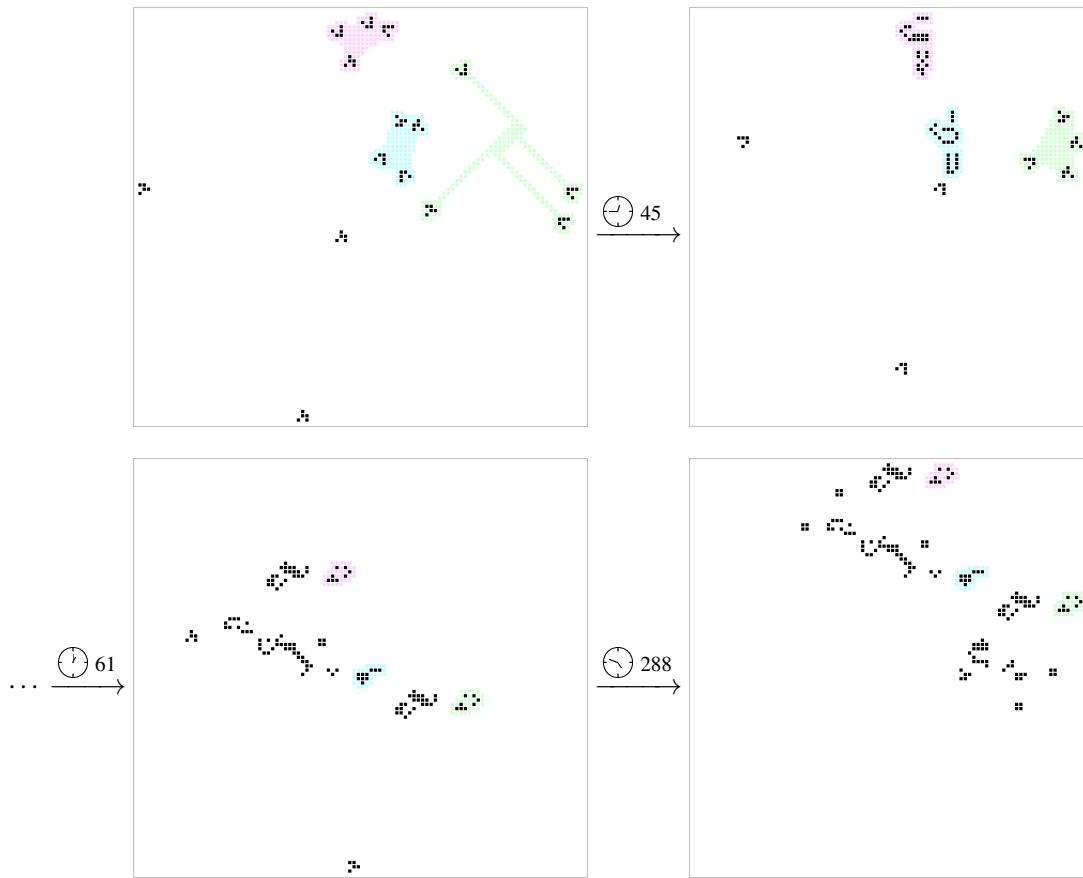


Figure 5.12: A 15-glider synthesis of the 3-engine Cordership, which was found by Jason Summers in 2004. The 3 switch engines are synthesized by the aqua, magenta, and green glider collisions. The remaining 3 gliders clean up some extra debris that is left behind by the switch engines.

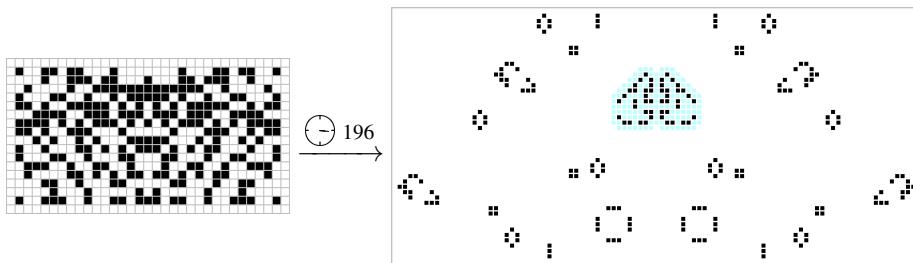


Figure 5.13: The soup that led to the discovery of Rich's p16 (highlighted on the right in aqua).

can 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 we can thus use to 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 the p16 oscillator.

The only somewhat tricky part of synthesizing this collection of objects is creating the two honey farm predecessors, since they are unstable and thus we have to create them after creating the still lifes. Thus 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

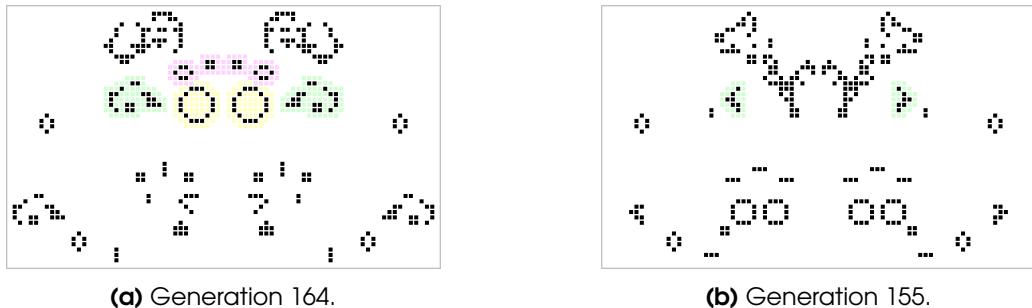


Figure 5.14: Rich's p16 is created when (a) two honey farm predecessors (highlighted in yellow) and two B-heptominoes (highlighted in green) collide with two blocks and two beehives (highlighted in magenta). The honey farm predecessors can be identified by comparing them with Figure 1.12, and the B-heptomino can be identified by going backward to generation 155 of the soup as in (b).

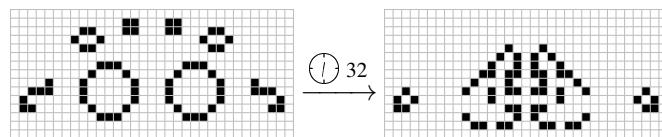


Figure 5.15: A reaction of common objects that results in Rich's p16 (and two unwanted boats).

synthesis of this oscillator is presented in Figure 5.16,⁶ and we stress that there is nothing 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.

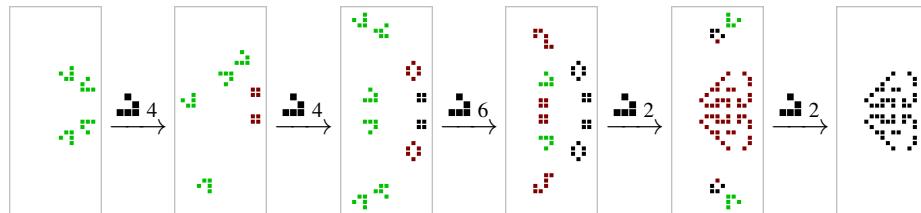


Figure 5.16: An 18-glider incremental synthesis of Rich's p16. The first three stages of synthesis just involve placing still lifes that can each be synthesized by 2 gliders. The next stage synthesizes the honey farm predecessors, and the final stage uses 2 additional gliders to destroy the 2 leftover boats. Objects that were created or modified in the previous step of the synthesis are shown in red, 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 will now construct our first *breeder*: a pattern that grows quadratically⁷ (as opposed to objects like guns or rakes, which grow linearly). In other words, we will construct a pattern 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 our construction is simple enough: we use the rakes we introduced in Section 4.4 to fire gliders in such a way that those gliders synthesize Gosper glider guns (using the incremental synthesis in Figure 5.6). Since the rakes will create Gosper glider guns at a linear rate, and the Gosper

⁶This synthesis was originally found, using the method we outlined in this section, by user "BlinkerSpawn" on the ConwayLife.com forums less than a day after the oscillator's discovery.

⁷There are actually breeders that grow slower than quadratically, and not all patterns with quadratic growth are breeders. We will return to the subject of breeders in Section ?? and give a more precise definition then.

glider guns will then each create gliders at a linear rate, the overall growth of our pattern will be quadratic.

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, and Gosper glider guns are wider than that, so they would collide and destroy each other. Thus we 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 will need four rakes, which we can position as in Figure 5.17.

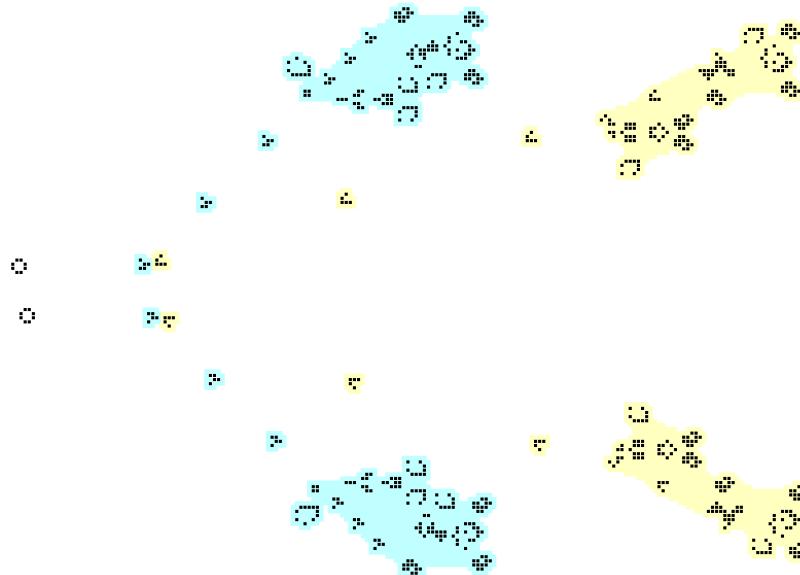


Figure 5.17: The front end of our breeder features two copies of the period 60 backward space rake (outlined in yellow), followed by two copies of the period 60 forward space rake (outlined in aqua), all traveling to the right. The gliders from these rakes collide in such a way as to create ponds, which is the first step of the incremental synthesis of a Gosper glider gun presented in Figure 5.6.

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 need to construct two blocks near each pair of ponds, which we can do with another four rakes. Then we need to hit the ponds with a single glider each in order to turn them into ships, which we do with two more rakes. Finally, we use two more rakes to fire gliders at the pair of ships, turning each of them into queen bees and thus completing the Gosper glider gun synthesis. Our completed breeder is depicted in Figure 5.18.⁸

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 the guns creates two additional 5-cell gliders, for a total population in generation $60n$ of $5n^2 + 44n + 2038$. This quadratic growth is demonstrated 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 all, in order to make the

⁸The first breeder was found by Bill Gosper in the early 1970s, and used the exact same ideas that we used in the construction of ours. However, instead of using space rakes, it used another puffer that emits two blocks and four gliders every 64 generations. The fact that this puffer drops blocks is useful because there is then no need to synthesize them, but the trade-off is that the puffer only shoots gliders backward, not forward, so lining up other collisions is slightly more difficult.

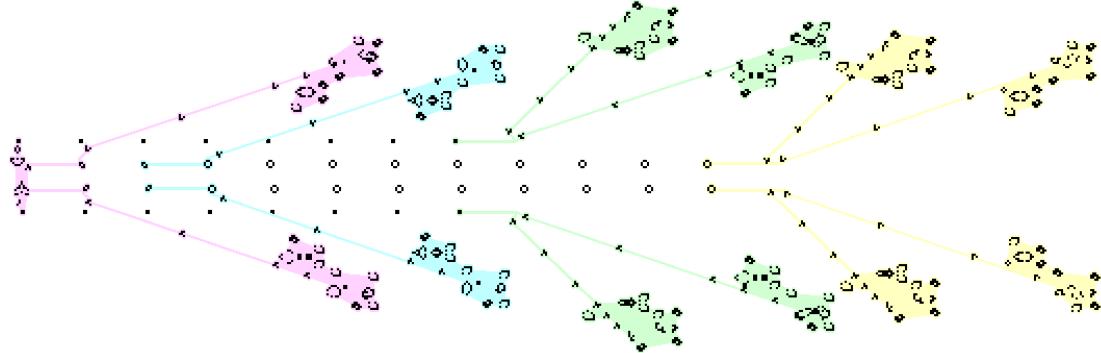


Figure 5.18: A breeder constructed using an incremental glider synthesis of a Gosper glider gun. First, a forward and backward period 60 space rake are used to collide two gliders and create a pond as in Figure 5.17 (outlined on the right in yellow), followed by another forward and backward space rake colliding two gliders to create a block (outlined in green). Next, a backward space rake uses one glider to convert the pond into a ship (outlined in aqua), and finally another backward space rake uses one glider to convert the ship into a queen bee (outlined in magenta), thus completing the synthesis of the Gosper glider gun.

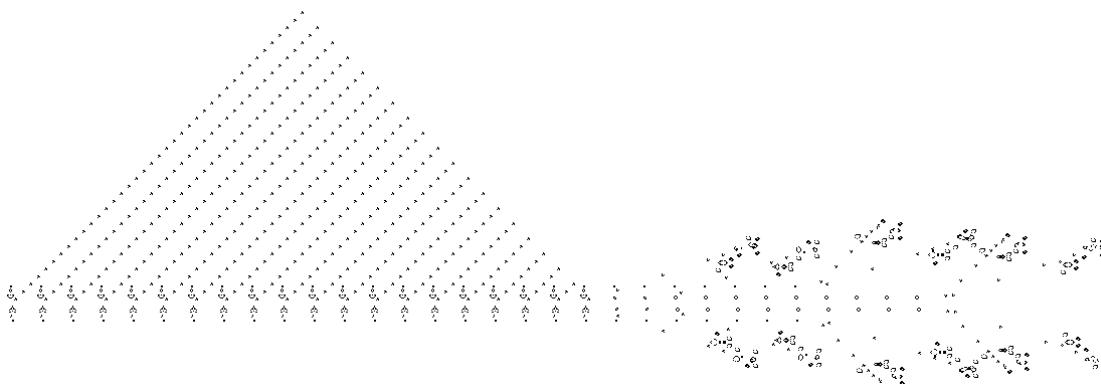


Figure 5.19: After running the breeder for 1200 generations, we see a triangle of gliders form on the left, above the line of Gosper glider guns that has been synthesized. As the breeder moves farther to the right, the triangle continues to expand.

mechanisms that make it work more apparent. The space rakes could be moved much closer to each other without compromising its functionality, 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 will 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.

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 gliders⁹ that (1) are *slow*: only one glider interacts in the synthesis at a time, and (2) 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

⁹Technically, a salvo can be made up of any spaceships, but the term almost always refers to gliders.

of still lifes and period 2 oscillators, respectively.¹⁰ 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 regular glider synthesis. That is, if we can synthesize an object with gliders then we can synthesize it with a p2 slow salvo.¹¹ In order to pin down exactly why this is the case, we will 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 synthesis 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.

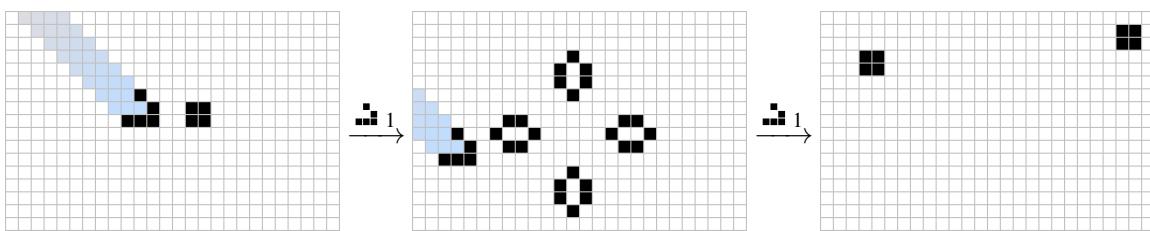


Figure 5.20: A 2-glider slow salvo can turn one block into two blocks.

While the two resulting blocks are in very different positions than the initial block, it turns out that this does not particularly matter, as we are able to use p1 slow salvos to move a block from any position to any other position on the Life plane. To show that such a movement is possible, we will 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.

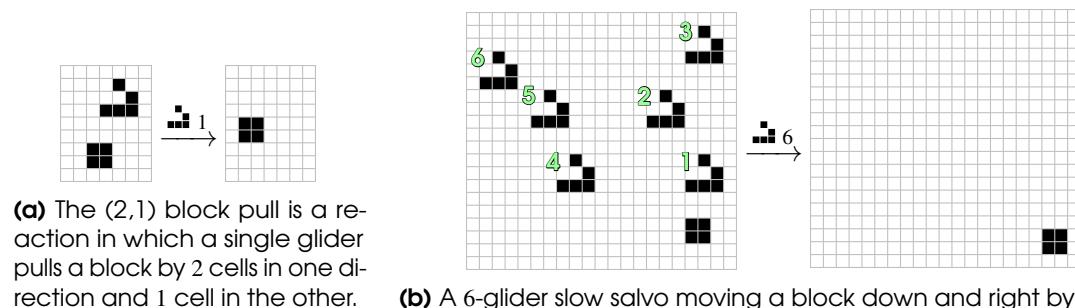


Figure 5.21: Two methods of moving a block around with p1 slow salvos that can be repeated in order to move a block to any position on the Life plane. The order in which the gliders should come in is indicated by the green numbers—their exact timing does not matter, as long as they are far enough apart that each collision settles down before the next glider arrives.

By combining these two reactions, we can move a block a single cell up, down, left, or right. For

¹⁰P2 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 p_n 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 period 1 or period 2.

¹¹However, the slow salvo might contain considerably more gliders than other glider syntheses.

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, perform the movement in Figure 5.21(a), followed by the movement in Figure 5.21(b) twice.¹²
- To move a block left by a single cell, 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, first move it left by a single cell and then perform the movement in Figure 5.21(b).

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 desire.¹³

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. Thus we can 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 5.22, 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 will be invaluable for us when using slow salvos to emulate arbitrary glider syntheses. There are also numerous other 90-degree one-time turners involving various numbers of blocks (and sometimes other small still lifes), but this turner involving just two blocks is perhaps the simplest, and for now it will be enough for our purposes.

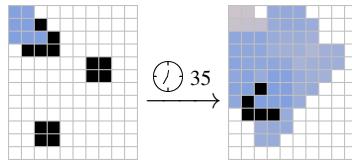


Figure 5.22: A *one-time turner* composed of two blocks that rotates a glider by 90 degrees and destroys the blocks at the same time.

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 5.23. 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 (and due to the clever way we repositioned the second block, this second duplication leaves a pair of blocks in exactly the one-time turner configuration).

Thus we can 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 one glider in a perpendicular direction, while still having a block

¹²This requires a total of 13 gliders, which is far from optimal—another reaction is known that accomplishes the same movement in only 7 total gliders. However, we are just interested in presenting the conceptually simplest method of moving blocks, not necessarily the most efficient method. 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). A slightly more up-to-date table that makes use of p2 salvos rather than just p1 salvos is available at conwaylife.com/forums/viewtopic.php?f=7&t=1072#p8182

¹³In linear algebra terms, what we have shown is that every point in \mathbb{Z}^2 is a positive linear combination of the vectors $(-1, 2)$, $(-2, 1)$, and $(1, -1)$.

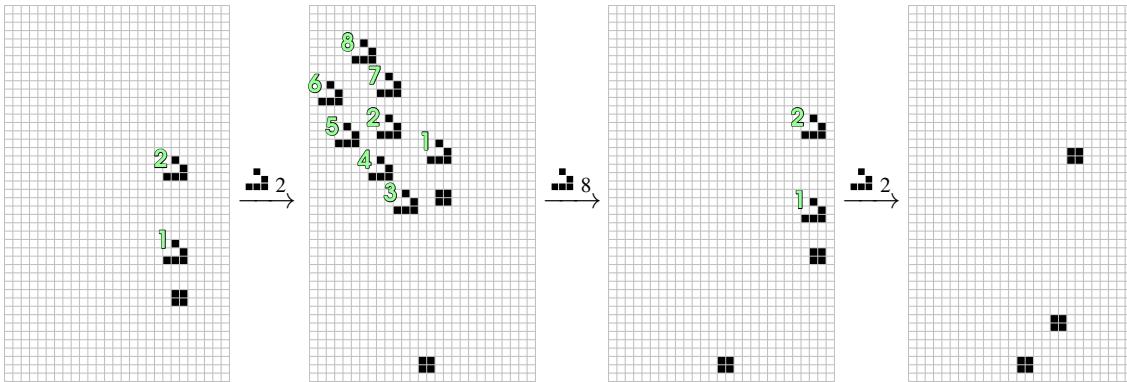


Figure 5.23: A p1 slow salvo of 12 gliders can be used to create the one-time turner from Figure 5.22 and leave us with another block left over (to create additional turners or other objects with). The first two gliders duplicate the initial block, using the reaction from Figure 5.20. The next 8 gliders move one of the blocks down and to the right, and then the final two gliders duplicate that moved block. 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.

left over to perform additional constructions with. Using this technique, we are now able to emulate multi-directional glider syntheses from unidirectional ones. For example, consider the four-glider syntheses of the clock that we saw in Table 5.3, which use one glider coming from each of the 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 5.24.¹⁴

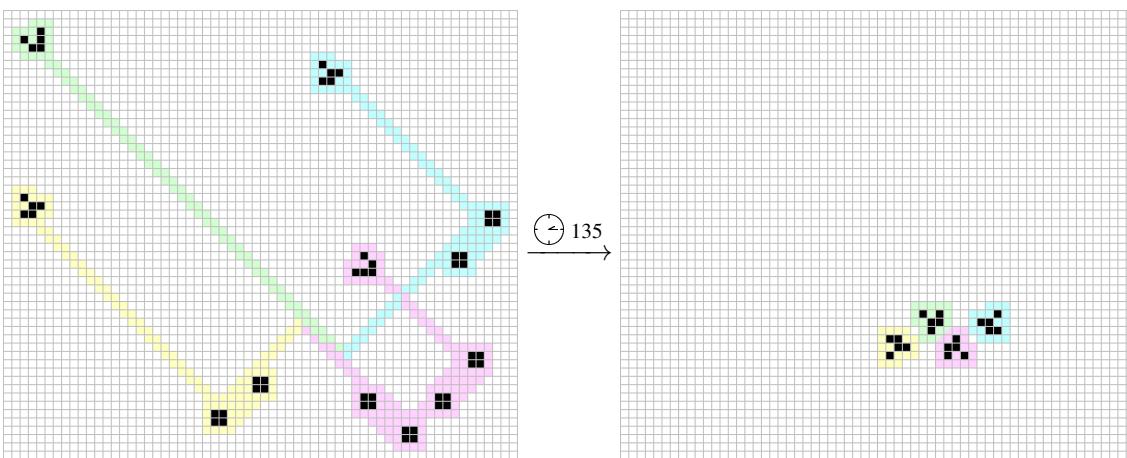


Figure 5.24: One-time turners (which can be synthesized with a p1 slow salvo) can be used to allow unidirectional glider waves (like the one on the left) to emulate multi-directional glider syntheses (like the one on the right). The gliders on the right are in exactly the position of the clock synthesis that we saw in Table 5.3.

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 will need to introduce some methods for using blocks to duplicate gliders and then adjust the timing of those gliders.

¹⁴We do not explicitly demonstrate how to create this arrangement of blocks using a p1 slow salvo, since it would require dozens of gliders.

5.7.3 Splitters and Timing

The first ingredient that we will 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*,¹⁵ and one example is provided in Figure 5.25.

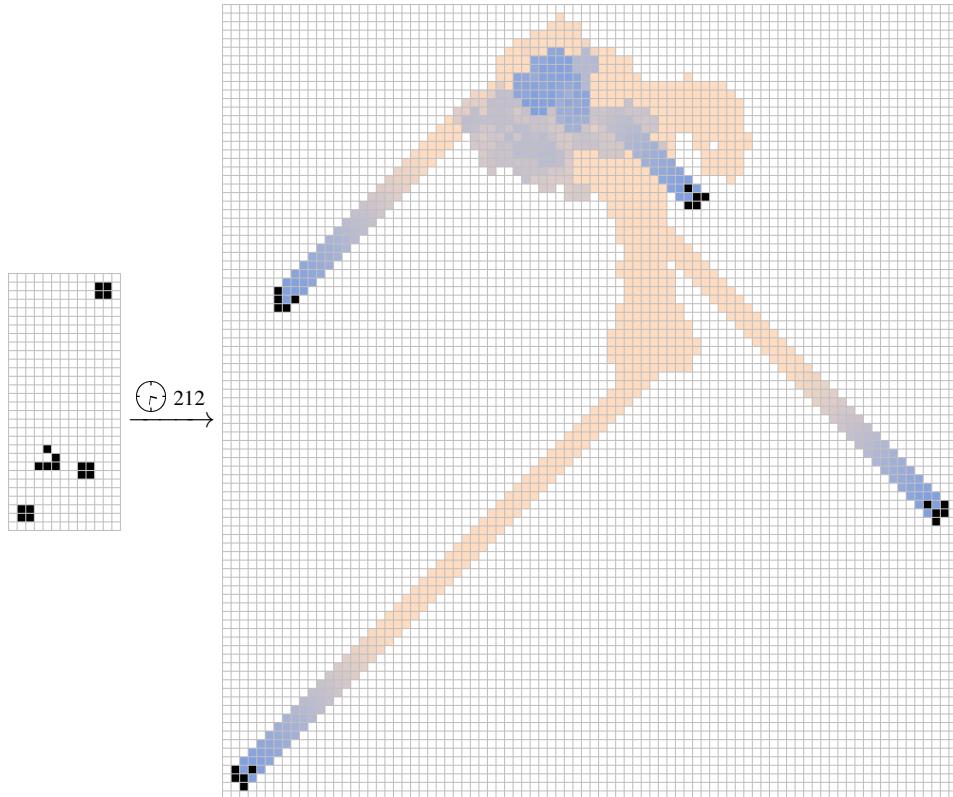


Figure 5.25: A *blockic splitter*, which uses three blocks to turn one glider into four gliders (while destroying the blocks in the process).

This splitter *almost* gets us what we need—it lets us turn one glider into four gliders, and we could chain multiple splitters together to turn one glider into as many gliders as we desire. We can use one-time turners to move these gliders around the Life plane, but unfortunately we are faced with two brand new problems:

- The one-time turner that we introduced in Figure 5.22 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 5.25 (in which the bottom-left glider has the same color as the input glider, whereas 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.
- Even though the one-time turner in Figure 5.22 is pretty good at moving gliders around the plane (ignoring the color issue that we just discussed), it gives us no control over timing, and we need some way of making sure that all of the desired gliders not only arrive at the right place, but they also arrive there at the same time.

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 family of 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

¹⁵The 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.

timing by any amount that we desire.¹⁶

		Offset							
		0	1	2	3	4	5	6	7
Color-Preserving	0								
	1								
Color-Changing	0								
	1								
Color-Preserving	2								
	3								
Color-Changing	2								
	3								

Table 5.5: By using different 90-degree one-time turners together, we can create this collection of 180-degree one-time turners that can be used to 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.

In order to make use of these 180-degree one-time turners to get gliders in (almost) any position and timing that we want, first simply use the 90-degree one-time turner in Figure 5.22 and the 180-degree one-time turners with offset 0 in Table 5.5 to put the gliders into their correct positions. Importantly, be sure to place at least one 180-degree turner in the path of each glider, even if it is not required to get the 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, focus on the glider that gets to its destination last—we will synchronize all

¹⁶This collection of 180-degree one-time turners was compiled by Dave Greene.

of the other gliders with this one. To slow down the other gliders, we note that moving a 180-degree turner 1 cell father away causes its glider to reach its destination 8 generations later (it now has to travel 1 cell, and thus 4 generations, farther in both directions). Thus to delay a glider by n generations, we can move its 180-degree turner father 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 $n \pmod{8}$.

Since we have a complete set of glider-preserving or glider-changing turners capable of delaying a glider by any of $0, 1, 2, \dots, 7$ generations, we thus now 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 5.25 to synthesize a clock is presented in Figure 5.26. 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 have greater control over the glider's output position.

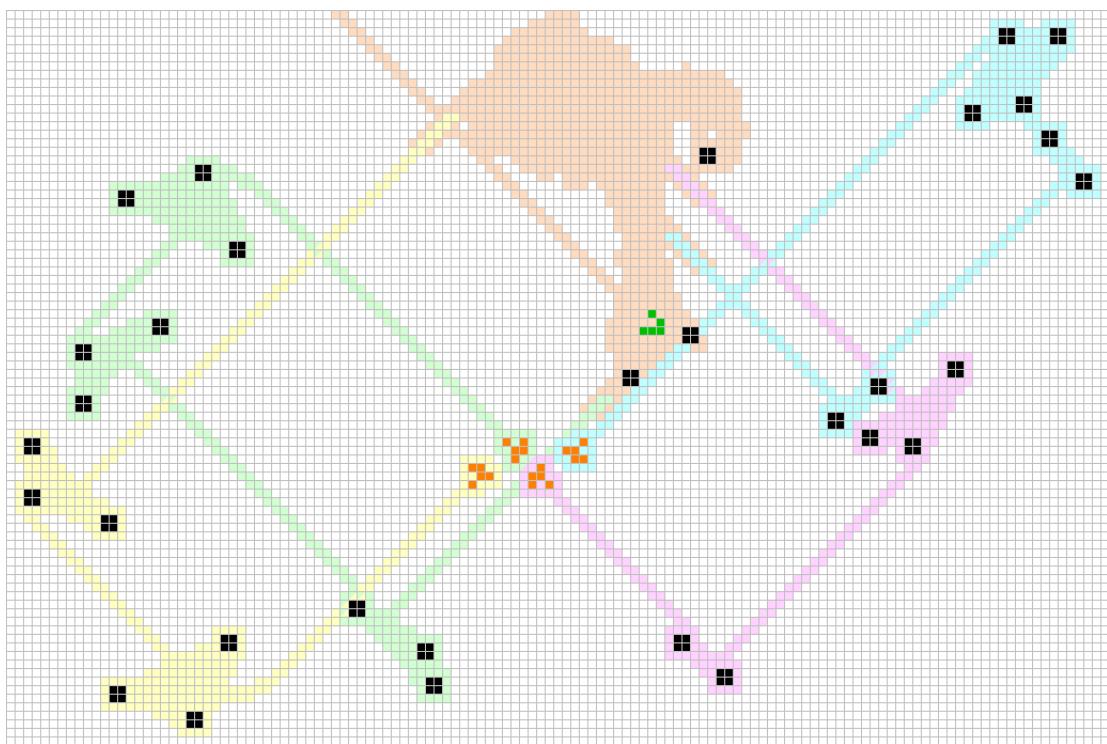


Figure 5.26: A blockic seed for a clock: The blockic splitter at the top-center (highlighted in orange) turns the single incoming glider into four gliders whose paths are outlined in yellow, green, aqua, and magenta. Those gliders are repeatedly reflected via one-time turners in such a way that they end up in the indicated positions required to synthesize a clock after 537 generations. Note that 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.

The arrangement of blocks shown in Figure 5.26 immediately gives a p1 slow salvo for constructing a clock—we “just” have to use a p1 slow salvo to construct the blocks (using the block moving and block duplicating techniques we discussed earlier) and then fire one additional glider to trigger the synthesis. However, a salvo for constructing that arrangement of blocks would contain dozens of gliders—recall that we already needed 12 gliders just to create a single 2-block one-time turner in Figure 5.23. Thus we re-emphasize at this point that even though these techniques show how to turn arbitrary glider syntheses into slow salvo syntheses, they are not efficient. Even very small syntheses can require hundreds or thousands of gliders to emulate in a p1 slow salvo.¹⁷

¹⁷ As an example, Dave Greene constructed an explicit p1 slow salvo consisting of 997 gliders that constructs a configuration of 81 blocks that, when hit by one additional glider, transforms into an 8-glider synthesis of the loafer spaceship. This salvo can be seen at conwaylife.com/forums/viewtopic.php?f=2&t=1006&p=7574#p7574.

5.7.4 Tight Packings of Gliders

There is still one final problem that might arise when trying to emulate an arbitrary glider synthesis with 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 that we have been using since each of the gliders in the final 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 a HWSS displayed in Table 5.2, it is not obvious whether or not we can position the two gliders that are heading southwest close enough to each other (since the various one-time turners that we have access to might interfere with one of the gliders when trying to place the other one). To fix this glider-packing problem, we introduce one final reaction: the *clock inserter*,¹⁸ which uses two opposing gliders to transform a clock into a perpendicular glider, as in Figure 5.27.

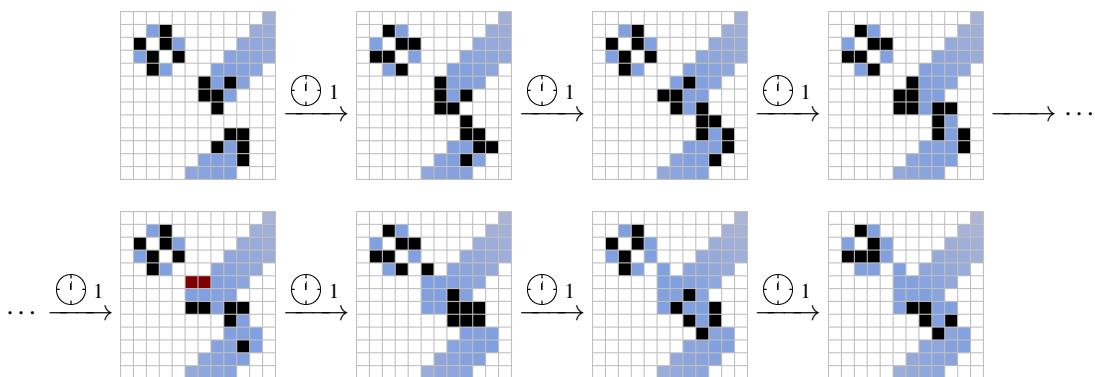


Figure 5.27: The *clock inserter* is a reaction that very cleanly collides two gliders in such a way as to transform a clock into a perpendicular glider. The two-glider collision produces a domino spark in generation 5 (shown in red) and then dies off, while the domino spark initiates the clock-to-glider 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 the back of the reaction, behind the 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. Thus this reaction can be used to place a glider very close in front of, or to the side of, another already-placed glider (see Figure 5.28 for an example 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 back first. Actually *proving* that the clock inserter reaction can be used to build any arrangement of gliders is somewhat technical and messy,¹⁹ since there are many possible ways for gliders to be near each other, so we defer the proof to Appendix B.

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.²⁰ 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 5.26) and then use the clock inserter to insert the close gliders one at a time. We have thus finally proved the following theorem:

¹⁸Found by Martin Grant in December 2014.

¹⁹Chris 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?f=2&t=1512&start=25#p15133.

²⁰Recall that the gliders must be able to reach their positions from arbitrarily far away in order to be considered a valid glider synthesis.

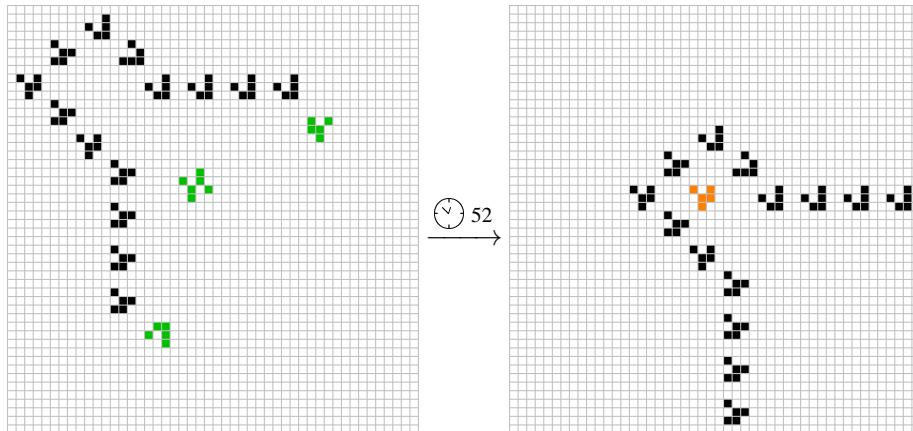


Figure 5.28: A demonstration of how the clock inserter (shown in green on the left) can be used to place a glider (shown in orange on the right) closely beside and in front of other already-placed gliders.

Theorem 5.1 Every pattern that can be constructed via glider synthesis can be constructed by a p2 slow salvo glider synthesis.

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. There are other reactions involving only p1 objects that can similarly be used to place gliders close to each other (see Exercise 5.34), but none of them are quite as clean, simple, and universal as the clock inserter. In particular, there is no known p1 inserter that works for all tight glider packings—some packings require one p1 inserter, while other packings require a different p1 inserter. For this reason, it is widely accepted that we can replace “p2” in Theorem 5.1 with “p1”, but pinning down the details is messy enough that no one has made the proof explicit.

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 45 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 Chapter ??).

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 in patterns 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 5.29(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.

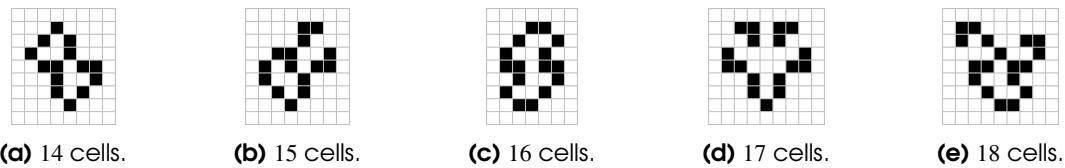


Figure 5.29: The final still lifes with 14–18 cells to be synthesized by gliders.

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 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).

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 5.29(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. Finally, a comparatively quick two-month effort finished the remaining syntheses of 18-cell still lifes in November of that year.

To give an idea of the size of this achievement, recall from Table 2.1 that there are 32,538 different strict still lifes with 18 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 5.30 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.

Rather than trying to push these techniques to synthesize all still lifes with 19 live cells, focus has shifted somewhat to 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, 9, 9, 11, and 12 or fewer gliders, respectively.²¹ 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 5.30 (which was first synthesized by 94 gliders) via just 10 gliders—see Figure 5.31.

It remains an open question whether or not every still life (or every oscillator, or even every pattern that has predecessors) can be constructed by glider synthesis. Closely related is the (also open) question of whether or not there exists a still life whose only parent is itself.²² Finding such a pattern would answer both of these questions, since if a still life’s only parent is itself then it can not be constructed via glider synthesis (or any other means).

Exercises

solutions on page 306

²¹It was not known how to synthesize the 9-cell long³ snake with fewer than 6 gliders until February 2015, when Matthias Merzenich derived a 5-glider synthesis from an apgsearch soup.

²²For the purposes of this problem, a still life together with far away non-interacting dying ash is not considered a different parent than just the still life itself. Conway offered a \$50 prize for a solution to this problem in October 1972, which has gone unclaimed to this day.

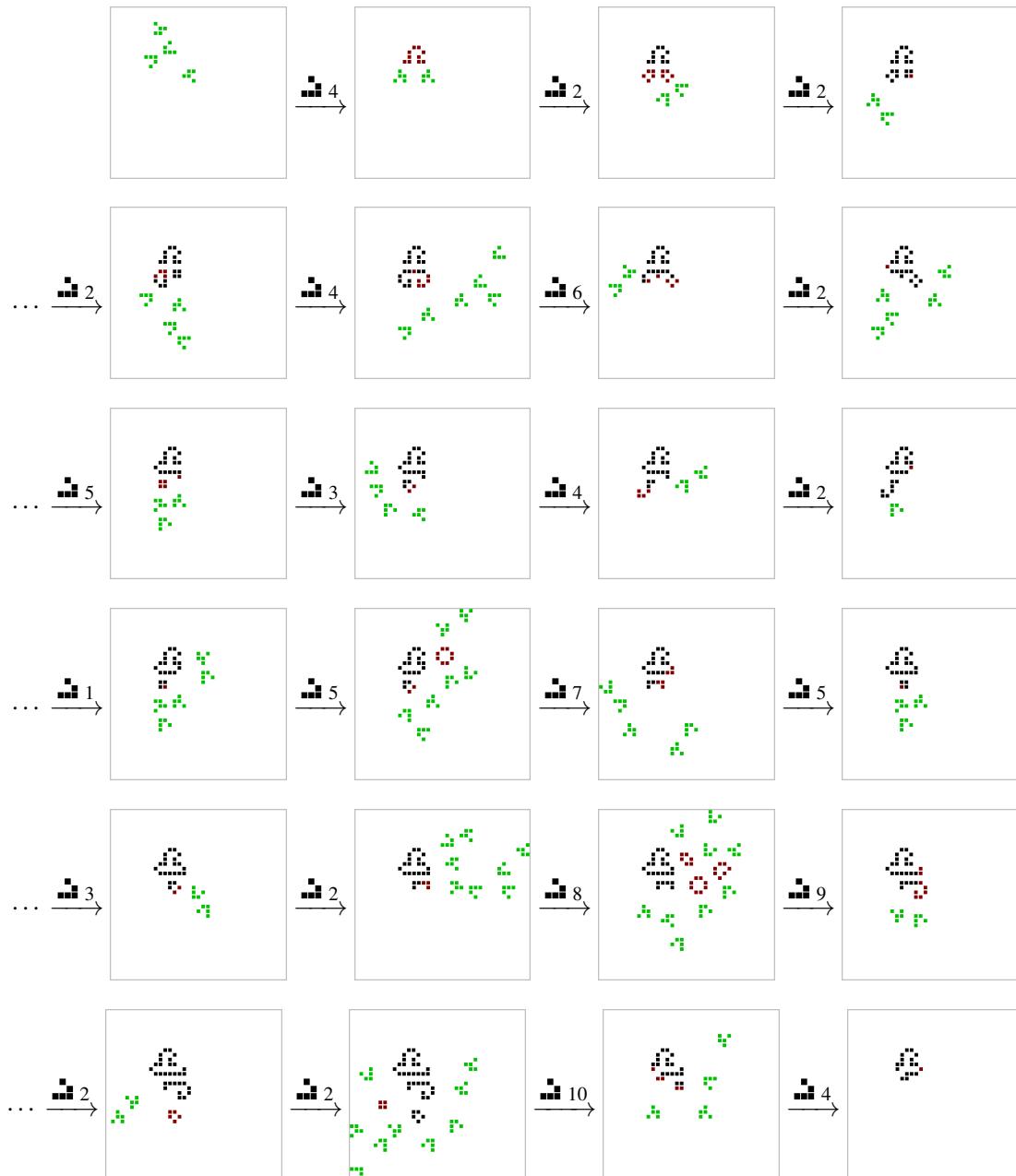


Figure 5.30: A summary of a massive 94-glider incremental synthesis of a hard-to-construct 17-cell still life (shown at the bottom right). Each stage in the synthesis works by using another glider collision to transform one still life into another. The cells that were created or modified in the previous step of the synthesis are shown in red, and the gliders that will be involved in the next collision are shown in green.

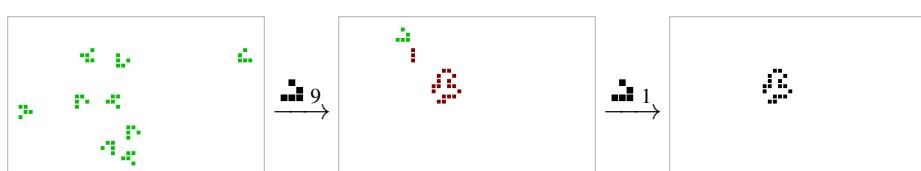
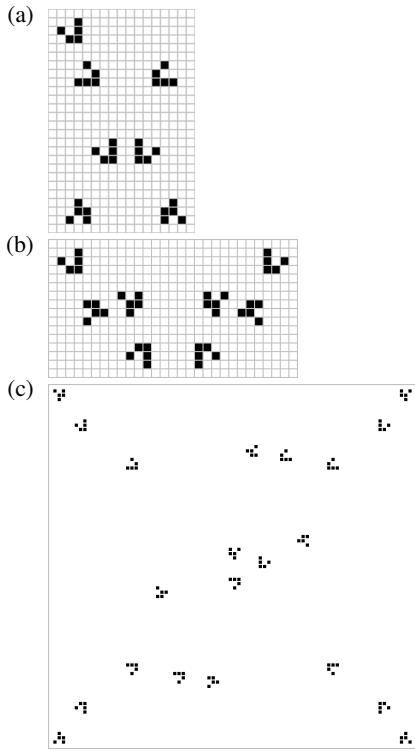


Figure 5.31: A 10-glider synthesis of the 17-cell still life from Figure 5.30 that was constructed with the help of a soup that was found by apgsearch.

5.1 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.



5.2 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 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.

5.4 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 Use the syntheses from this chapter to create a 7-glider synthesis of eater 5.

5.6 Use the syntheses from this chapter to create a backrake that creates a period 120 stream of lightweight spaceships.

5.7 Use three Gosper glider guns to create a middleweight spaceship gun.

5.8 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* (see Figure 5.32) 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)?

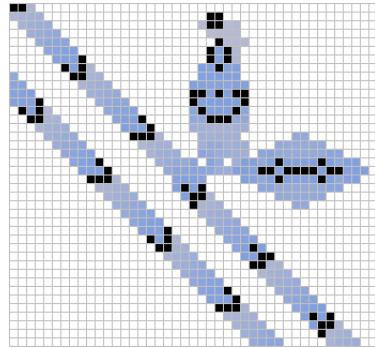


Figure 5.32: A *glider pusher* is a configuration of a queen bee (top center) and a pentadecathlon (middle right) that was found by Dietrich Leithner in December 1993. It pushes a passing glider away by one lane, and thus closes gaps between glider streams with periods that are a multiple of 30.

5.9 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) Use this synthesis to synthesize a twin bees shuttle.
- (b) Use this synthesis to synthesize the twin bees gun from Figure 1.23.

5.11 Create a glider synthesis of the glider pusher from Figure 5.32.

5.12 Construct a glider synthesis of the period 36 oscillator in Figure 3.7(i).

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) A switch engine.
- (b) A clock.
- (c) A 3-engine Cordership.

5.14 Recall the 2-engine Cordership from Exercise 4.19.

- (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 By using the 3-glider collision in Table 5.2 called a “tee”, which produces a glider perpendicular to each of the input gliders, we can modify many glider syntheses so that all input gliders come from two directions that are opposite each other. 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 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 Complete the incremental glider synthesis of the ecologist presented in Figure 5.8 (i.e., construct it in its entirety from an arrangement of gliders). This synthesis should only use gliders coming from the bottom-left and bottom-right. [Hint: You can use two gliders to create a block and then two more gliders to turn the block into a LWSS.]

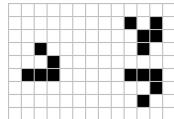
5.18 In Figure 5.9 we showed how to use gliders to turn an ecologist into a forward space rake.

- (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.]

5.19 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 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 Use the following 3-glider collision to reduce the 18-glider synthesis of Rich’s p16 in Figure 5.16 down to 16 gliders.²³



5.22 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.]

²³This 16-glider synthesis was found by Martin Grant.

5.23 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.

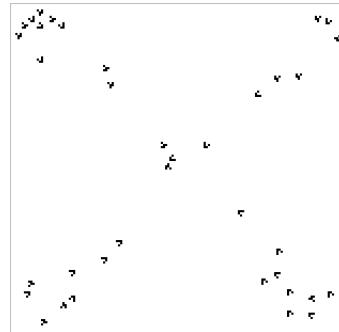
5.24 Construct a breeder that creates Gosper glider guns that are stabilized on at least one side by eater 1s instead of blocks. [Hint: Use the glider synthesis from Exercise 5.16.]

5.25 Create a breeder that uses several rakes to synthesize block-laying switch engines as they move. [Hint: Use a 4-glider synthesis of a switch engine together with the reaction from Figure 1.25.]

5.26 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.

5.27 Use a p1 slow salvo to create the arrangement of 8 blocks in Figure 5.24.

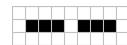
5.28 A 37-glider synthesis of the tubstretcher from Figure 4.10 is displayed below.



- (a) Remove some gliders so as to produce a glider synthesis of the crab.
- (b) Add some extra gliders to destroy the tubstretcher (but not the tub itself) after it has been synthesized. Use this method to show that a fixed number of gliders (say 40 or so) can be used to synthesize arbitrarily-large strict still lifes.

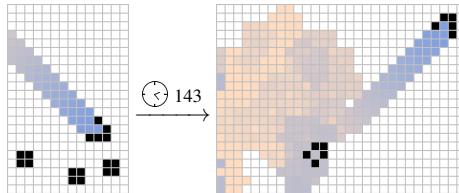
5.29 In this exercise, you will construct some limited-use turners that are different from the blockic one-time turners that we saw in this chapter.

- (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.



5.30 In this exercise, we will practice moving gliders around one-time tracks.

- (a) 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) 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).



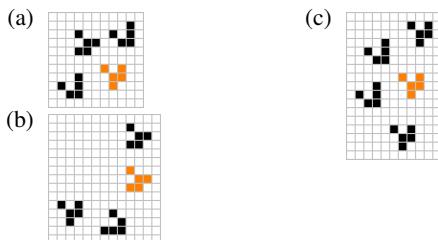
- (c) Use three copies of the two-time turner, together with some additional one-time-turners, to move a glider around a square track twice, and then leave the track in the same direction that it started in (destroying the track in the process).

5.31 One-time turners can be used as tracks for gliders, allowing us to create objects that move 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 Figure ?? to lay two blinker fuses that a glider bounces back and forth between, using the reaction from Exercise 5.29(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 5.29(a)).

5.32 Find a way of placing a block near a clock so that a single glider (rather than a pair of gliders, as in Figure 5.27) can trigger the clock inserter reaction.

5.33 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 5.28. In all cases, the input gliders to the clock inserter must come only from the lower-left and the upper-right.



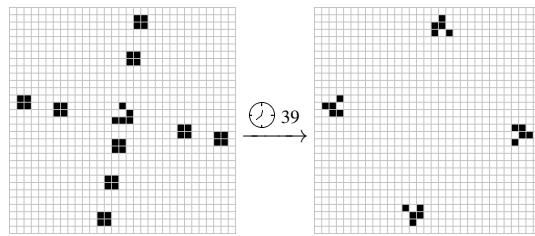
5.34 Many reactions can be used as inserters other than the clock inserter from Figure 5.27 (however, none are quite as good as the clock inserter itself).

- (a) Use the “tee” 3-glider collision 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 to Exercise 5.29(b).]
- (c) Use the clock inserter to insert a glider 14 generations in front of another glider.

5.35 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 5.26.]

- (a) A lightweight spaceship.
- (b) A pulsar.
- (c) A switch engine.

5.36 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 5.25, but the disadvantage of requiring 9 blocks instead of just 3. Use this splitter as part of a blockic seed for a clock.



5.37 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...

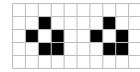
- (a) 7 gliders.
- (b) 10 gliders.
- (c) 9 gliders.

5.38 In the proof of Proposition 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 w times its timing.

- (a) What parts of the proof of Proposition B.1 change if we use $w = 2$? Does the proof still work?
- (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 of Proposition B.1 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 of Proposition B.1 still works? [Hint: There will be a technicality similar to the one from part (b) 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.

5.39 A *seed* is a constellation (i.e., a collection of simple still lifes and maybe small p2 oscillators) that, when hit by one or more gliders, evolves into a more complicated object. Synthesizing a seed via standard glider collisions and then using that seed to synthesize a more complicated object is a particularly useful technique when working with slow salvo synthesis.

- (a) 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) Create a glider synthesis that first creates the pair of boats displayed in part (a) and then creates a switch engine from them.
- (c) Use your solution to part (a) to construct a seed that, when hit by 2 gliders, synthesizes a 2-engine Cordership.
- (d) Create a glider synthesis that first creates your seed from part (c) and then creates a 2-engine Cordership from it.

Early draft (May 19, 2020).

Not for public dissemination.

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: constructing large patterns that do unusual things typically boils down to 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. Thus we 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.

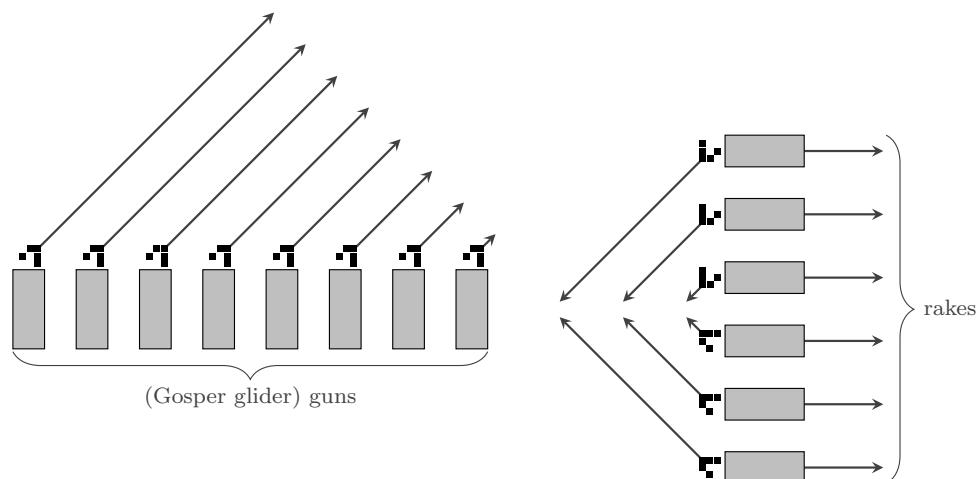


Figure 6.1: A schematic that gives a rough picture of how the breeder that we constructed in Section 5.6 works.

- 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—our goal is to be able 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 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 1.18) and buckaroo (Figures 3.16(b) and 6.2), 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.

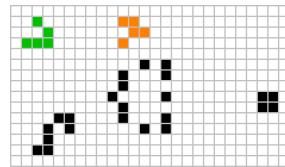
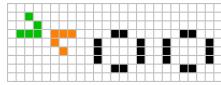


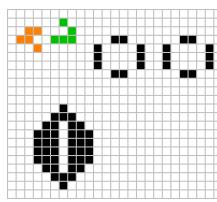
Figure 6.2: A buckaroo can reflect a glider by 90 degrees, from the position marked in green to the one in orange 30 generations later.

6.1.1 Reflectors

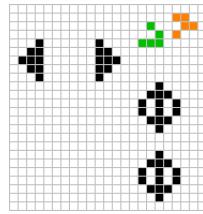
Because the period of the pentadecathlon is 15, which evenly divides 30, it works very well with other period 30 circuitry. In particular, it can be used to reflect gliders in numerous different ways that are shown in Figure 6.3. The reflection shown in Figure 6.3(a) is the same one that we saw back in the period 60 oscillator of Figure 3.28, but the other two are new.



(a) A pentadecathlon reflecting by 180 degrees.



(b) Two pentadecathlons reflecting by 180 degrees.



(c) Two pentadecathlons reflecting by 90 degrees.

Figure 6.3: 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.

In particular, Figure 6.3(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 6.3(a)), and Figure 6.3(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 Figures 6.34(e) and 6.34(f), and two more color-preserving 90-degree options in Exercises 6.23c

and 6.23d.¹

6.1.2 The Inline Inverter

One of the most useful features of the Gosper glider gun is that it can be turned into an *inline inverter*—a reaction that “flips” a period 30 glider stream so that any glider present in the stream is destroyed, while any glider missing from the stream is inserted.² The resulting stream travels in the same direction as the input stream, but is slightly offset (see Figure 6.4(a)).

The inline inverter provides us with a simple way to construct irregular glider streams. For example, if we feed a period 120 glider stream into it, we get as output a period 30 glider stream in which only 3 out of every 4 gliders are present (see Figure 6.4(b)).

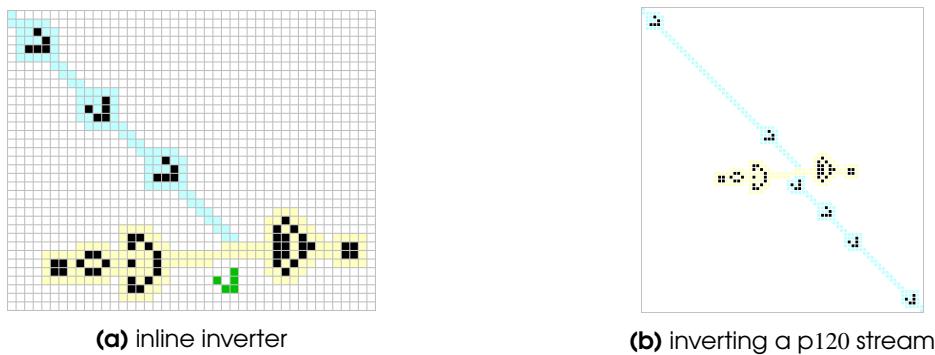


Figure 6.4: The inline inverter is a reaction involving the Gosper glider gun in which the gun fails to produce a glider if it receives a glider as input at the same time. (a) If an input glider is *not* present then an output glider will appear at the location marked in green 30 generations after the phase shown here (i.e., the output stream is 5 cell further south than the input stream). (b) An inline inverter can be used to turn thin glider stream into a thick one (in this case, a period 120 glider stream into a period 30 stream that is missing one out of every four gliders).

Similarly, 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 could of course do the same thing simply by aiming the glider stream from a Gosper glider gun at an eater (as in Exercise 2.8), but this method has the advantage that we can trigger it to release a glider by destroying one of the gliders in the middle 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 6.5.

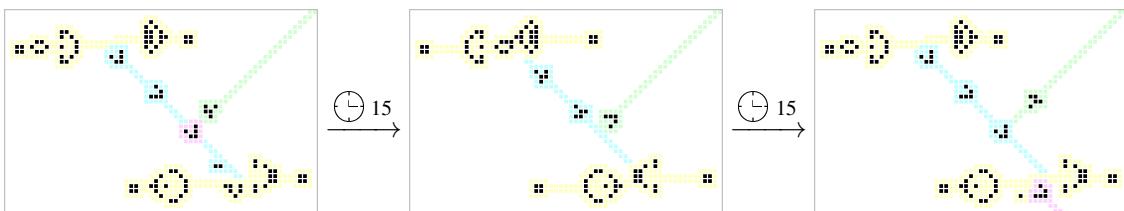


Figure 6.5: Two Gosper glider guns can be placed near each other so as to create a finite stream of gliders (shown here in aqua) between them. Here, we bounce a single glider (in green) off of one of those gliders (in magenta) so that it is destroyed and thus released by the inline inverter to the southeast.

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 15

¹It 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).

²The inline inverter was discovered by David Bell.

cells farther away from each other, we can create glider guns with period equal to $120n$ for any integer $n \geq 1$. Figure 6.6(a) illustrates a gun of this type of period 120.

We should clarify at this point that there are actually many other methods of stream inversion possible as well, but the inverted stream is typically reflected by 90 degrees from the input stream, rather than going in the same direction as it (i.e., the inversion is not “inline”).³ For example, simply firing an irregular stream at a regular stream so that they collide as in one of the two-glider annihilations from Table 5.1 gives an inverter. This inverter can do many of the same things as the inline inverter, such as creating high-period guns as in Figure 6.6(b). If desired, we can turn this inversion into one that is inline simply by pairing it with any compatible 90-degree periodic reflector, or with a Snark as long as the period is 43 or greater.

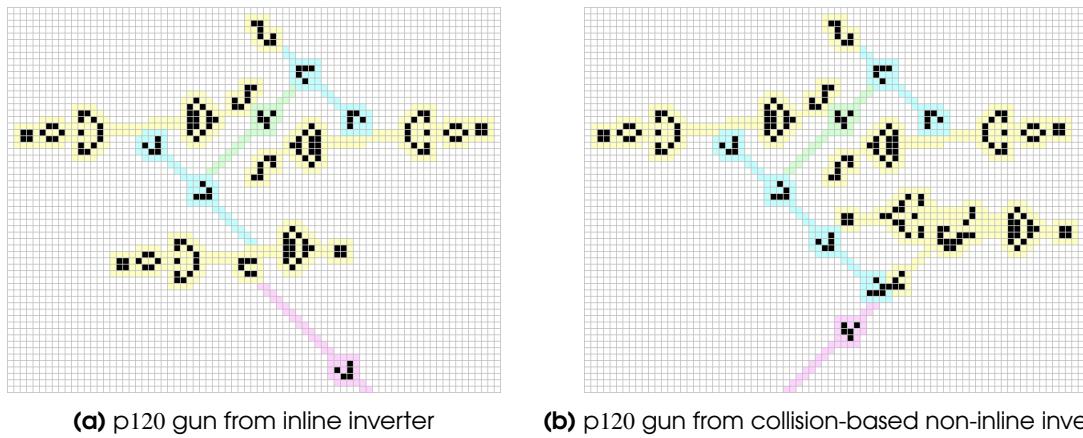


Figure 6.6: Three Gosper glider guns (highlighted in yellow, the bottom of which acts as an inverter) can be used to create a period 120 glider gun. The glider outlined in green bounces back and forth between the two aqua glider streams, destroying one glider in each stream every time it is reflected. The glider that is destroyed in the left stream is then created (in magenta) by the inverter. In (a), the bottom glider gun is an inline inverter whereas it is a non-inline inverter in (b). The period of these guns can be increased by 120 generations by moving the top-right glider gun and eater northeast by 15 cells.

Another slightly more useful (non-inline) inversion reaction is provided in Figure 6.7, in which a glider stream of any period 20 or larger comes in from the northwest, and a stream of the same period (but potentially with some gliders missing) comes in from the northeast. When the stream from the northwest reaches the block, it either passes by unharmed (if there is no corresponding glider in the stream from the northeast), or it collides with the glider from the northeast, producing a single glider to the southwest. Thus two output glider streams are produced: one to the southwest that contains the same gliders and spacing as the northeast input stream (which can be destroyed with an eater if it is not needed), and one to the southeast that is the inverse of the northeast input stream.

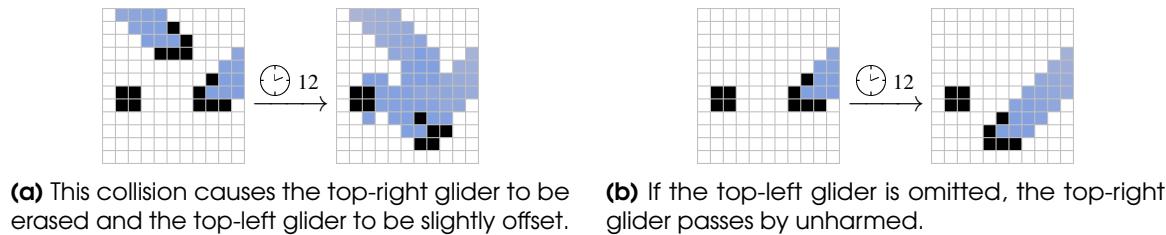


Figure 6.7: A stream inverter that splits the glider stream coming in from the northeast based on which gliders are present in the stream coming in from the northwest. This reaction can be used with any glider stream of period 20 or higher.

By reflecting the inverted stream and using an eater to destroy the other output stream, we can

³We will see a period 46 inline inverter a bit later in this chapter, in Figure 6.27(b).

use this reaction to create an inline inverter of any period at least 20 (as long as the repeat time of the reflector is low enough—see Exercise 6.5). Even more remarkably, we can combine this reaction with another inverter to duplicate arbitrary glider streams (even irregular ones with holes), simply by inverting the inverted output stream, as in Figure 7.10(a). 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.

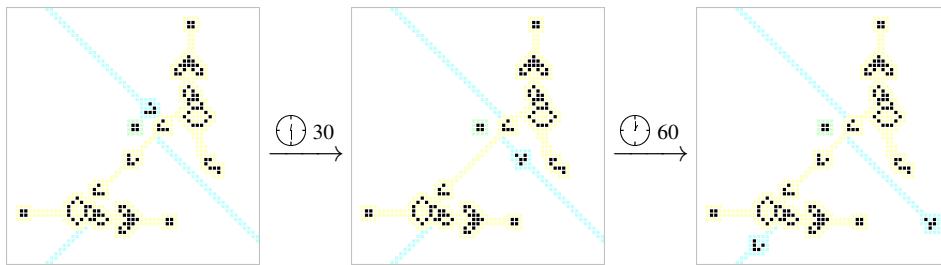


Figure 6.8: A glider duplicator that makes use of the inline inverter and the reaction from Figure 6.7. The input glider from the northwest (highlighted in aqua) is copied to the southeast and destroys the corresponding glider going southwest. The southern inline inverter then creates a glider going southwest, thus duplicating the original glider.

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 closer together (which is extremely useful when trying to implement some tight glider syntheses) is a *glider pusher*,⁴ which is a combination of a (half-stabilized) queen bee and a pentadecathlon that push a glider over by a single lane (see Figure 6.9).⁵

Two other simple reactions that make use of the debris and sparks left behind by queen bees and pentadecathlons are shown in Figures 6.10 and 6.11. 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 6.11 to the pentadecathlon-based reflectors of Figures 6.3(b) and 6.3(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.

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*,⁶ which is a glider gun that can be switched on and off by firing a

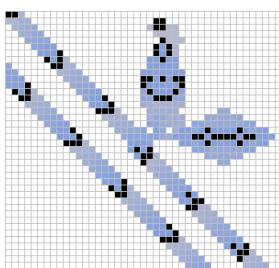


Figure 6.9: A *glider pusher* pushes a glider away by one lane.

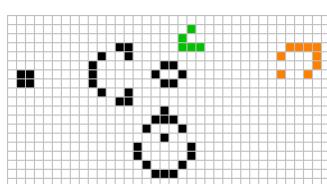


Figure 6.10: Two queen bees can convert a glider into an LWSS.

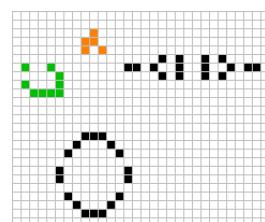


Figure 6.11: Pentadecathlons can convert an LWSS into a glider.

⁴Found by Dietrich Leithner in December 1993.

⁵We first saw this glider pusher in Exercise 5.8b.

⁶Found by Dean Hickerson in April 1996.

single glider into it (contrast this with the inline inverter, which is a glider gun that can be toggled on and 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.

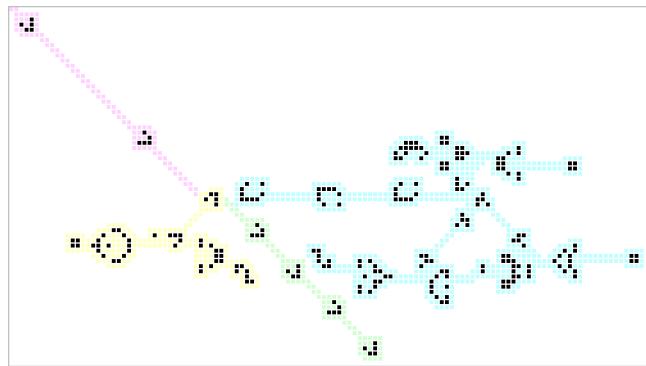


Figure 6.12: A *toggle* that bounces gliders from a Gosper glider gun (highlighted in yellow) off of a stream of lightweight spaceships (highlighted in aqua). The first incoming glider (on the magenta stream) turns off the output stream of gliders (highlighted in green) until another glider comes in 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 pattern that emits a stream of lightweight spaceships with spacing as in Figure 6.13. We call patterns of this type *primers*.⁷



Figure 6.13: A stream of lightweight spaceships in which only the prime-indexed spaceships are present.

6.2.1 A LWSS Gun with an Irregular Stream

Before building the desired primer itself, let’s start by constructing 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 stream of lightweight spaceships and strategically destroy the unwanted spaceships in it (i.e., we destroy all lightweight spaceships corresponding to multiples of 2 or multiples of 3). Figure 6.14 shows how we can fire a glider at a lightweight spaceship so as to destroy them both.⁸ Thanks to this reaction, we can create the desired stream of lightweight spaceships simply by aiming two glider

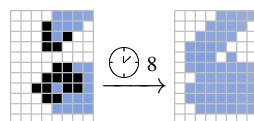


Figure 6.14: A glider and an LWSS destroying each other.

⁷The 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 a stream inversion reaction that we will see in Section ??).

⁸Reactions 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.

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 6.15.

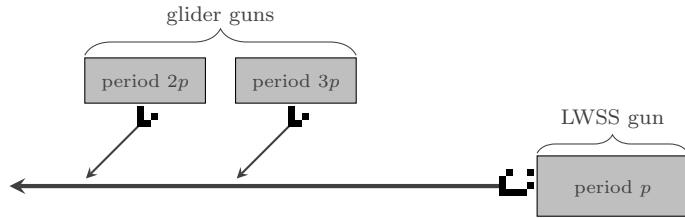


Figure 6.15: A schematic for a LWSS gun that fires an irregular stream of spaceships with the property that the n -th spaceship in the stream is present if and only if n is not a multiple of 2 or 3. Gliders are used to delete every second and every third spaceship in the LWSS stream.

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 6.6(a). In particular, we arrange three period 120 guns so as to synthesize the LWSS 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 6.16.

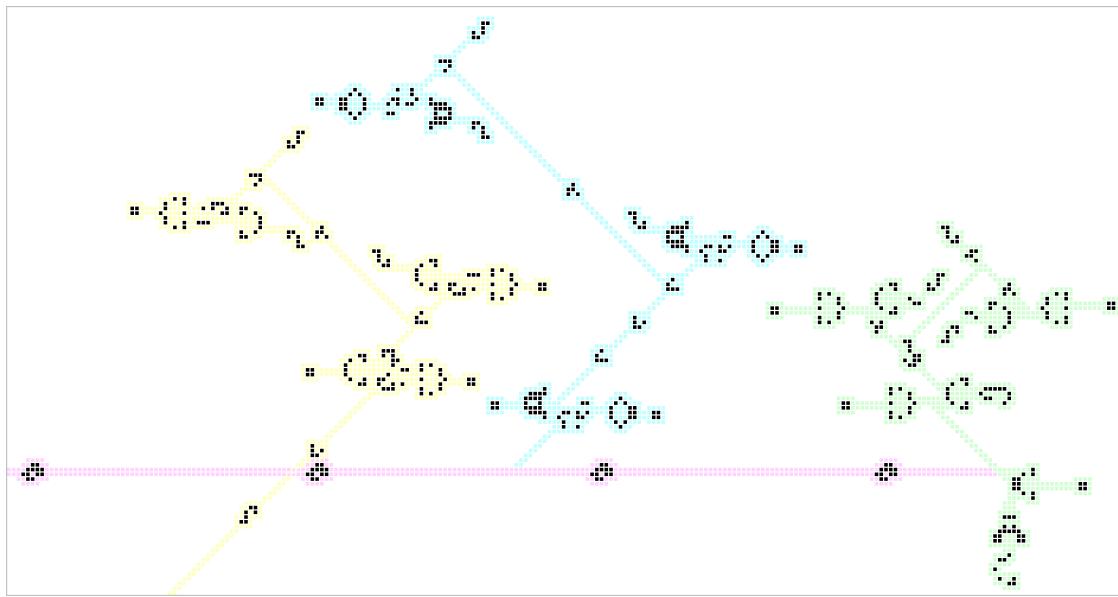


Figure 6.16: A gun that shoots an irregularly-spaced stream of lightweight spaceships. In particular, the period 120 inline inverter gun makes use of the glider-to-LWSS reaction from Figure 6.10 (highlighted in green) to create a period 120 stream of lightweight spaceships (highlighted in magenta). Period 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 ($p = 120$) 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 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 to construct 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 (along which a single glider will travel) between these endpoint Gosper glider guns will increase without bound, giving us glider guns of larger and larger periods. We thus arrive at the schematic presented in Figure 6.17 for synthesizing these high-period glider guns and thus our primer.

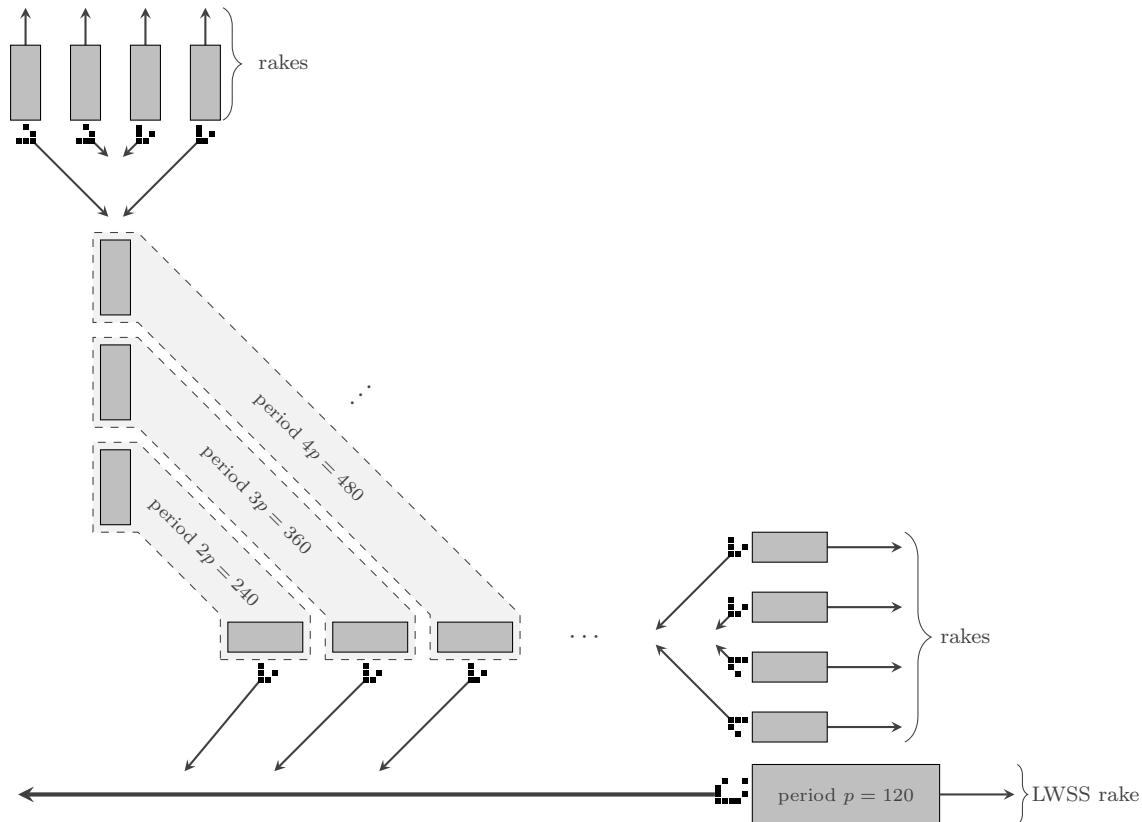


Figure 6.17: A schematic for a primer. For each integer $n \geq 2$, eastward and northward sets of rakes leave behind halves of the period $120n$ glider gun based on the inline inverter, which is used to destroy every LWSS in the bottom period 120 stream whose position is a multiple of n . The only spaceships 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 6.18. 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 up piece-by-piece.

Unfortunately, there are actually quite a few subtle problems with the previous schematic that prevent us from building it as-is:

- 1) The only Gosper glider gun breeder that we know how to build so far has 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 120 stream. However, since we do not yet know how to do this, we instead use the breeders to create glider guns with periods $3p, 5p, 7p, 9p, \dots$

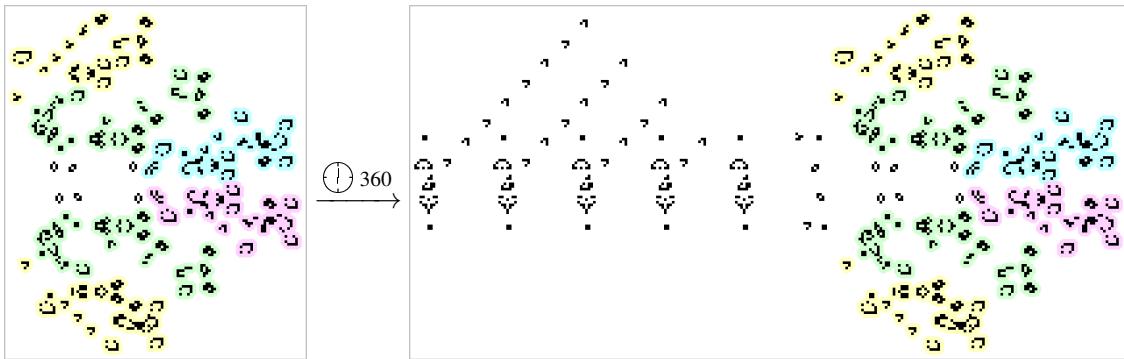


Figure 6.18: A compact breeder that uses 6 period 60 space rakes to carry out an incremental synthesis of a Gosper glider gun. It gets away with fewer space rakes than our original breeder from Figure 5.18 (which used 12) since it cleverly manipulates the debris behind the two frontmost space rakes to create a ship, which is the most difficult part of the Gosper glider gun’s incremental synthesis to create.

to delete any lightweight spaceships whose position in the stream is a multiple of 3, 5, 7, 9, ..., respectively, and 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 6.6(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 6.2). To create a row of these modified Gosper glider guns, we can use the breeder in Figure 6.19, which is just a slight modification of the breeders that we have already seen.⁹

⁹We already constructed a (much larger) breeder for this modification of the Gosper glider gun back in Exercise 5.24.

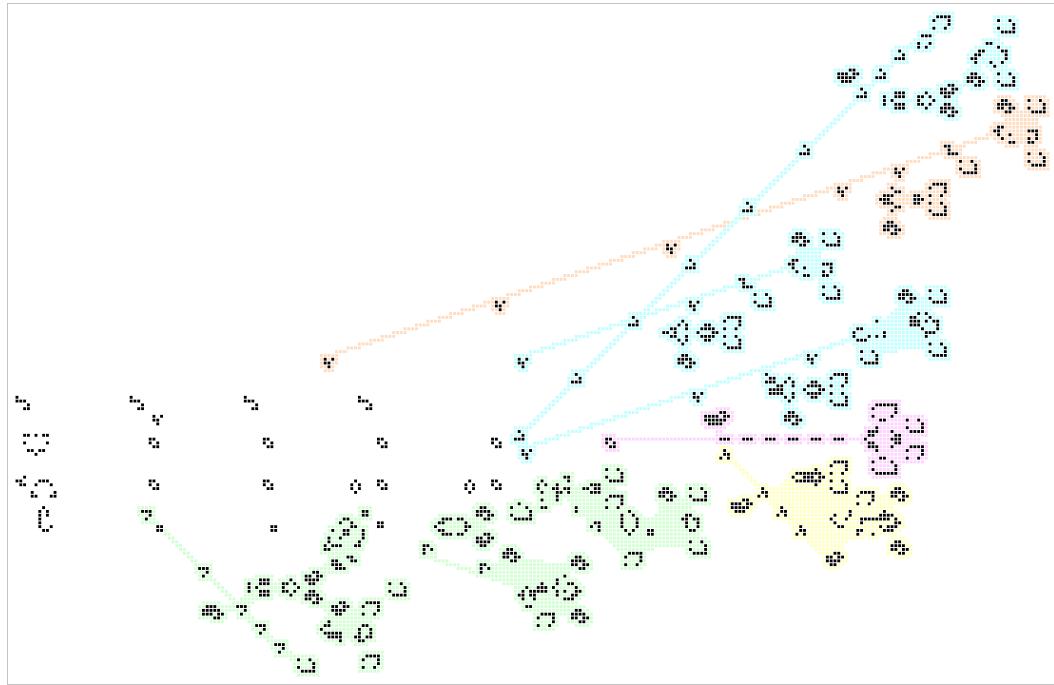


Figure 6.19: A Gosper glider gun breeder that is stabilized on one side by an eater 1 instead of a block. The bottom half (highlighted in green) is the same as the bottom half of the compact breeder from Figure 6.18. The bottom-right space rake (highlighted in yellow) turns one of the blinkers from the 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.

- 3) When constructing the two east–west 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. This row of blocks can simply be synthesized by two space rakes.

After putting all of these pieces together, we arrive at our completed primer in Figure 6.20. 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.

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 (by either 90 degrees or 180 degrees¹⁰) and lightweight spaceships, and even convert gliders *into* lightweight spaceships, and vice-versa, as displayed in Figure 6.21.

We saw one of these glider reflections way back in Section 3.3.1—the twin bees shuttle creates

¹⁰The 180 degree 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.

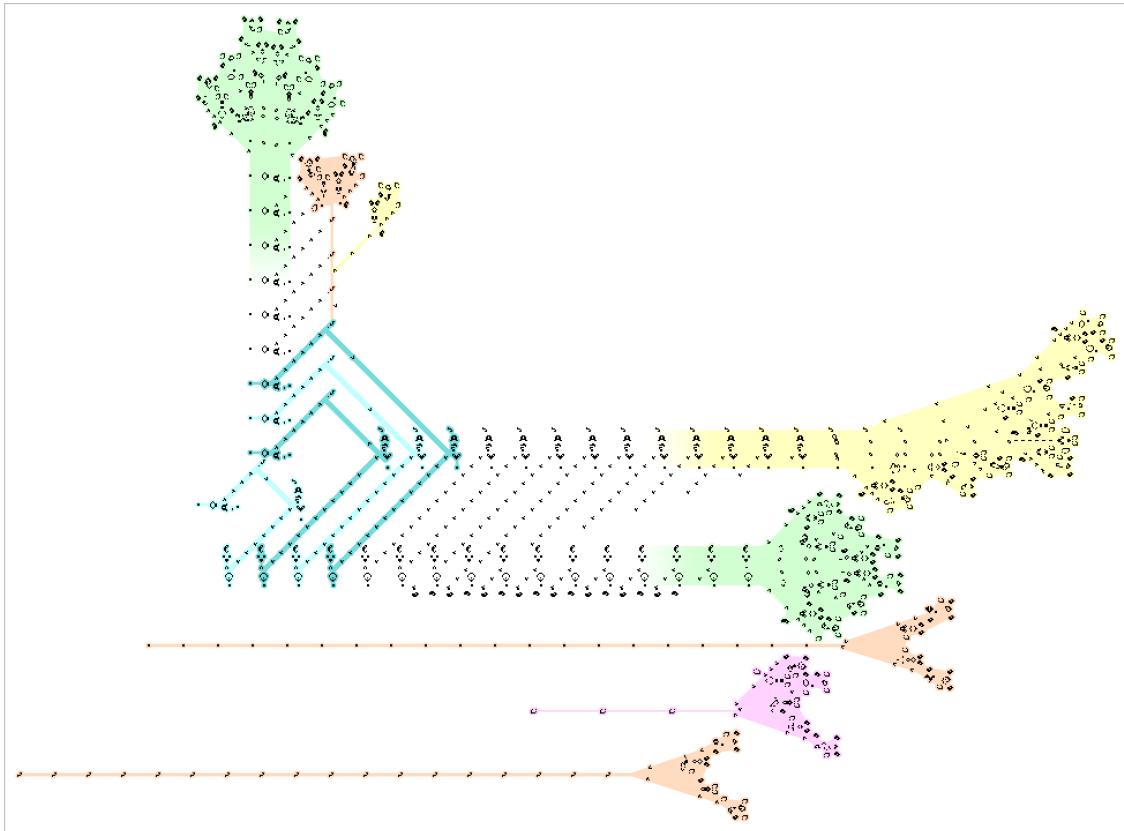


Figure 6.20: The completed primer. The period 120 stream of lightweight spaceships (highlighted in magenta and synthesized by 3 space rakes as in Exercise 5.6) moves to the left and is thinned out by the inline inverter guns (highlighted in aqua). The leftmost inline inverter gun has period 240 and has 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 6.18 (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 6.19 (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 orange) simply synthesize rows of block and eater 1s.

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 produced by the form of the twin bees shuttle that is stabilized on one side by just a single block.

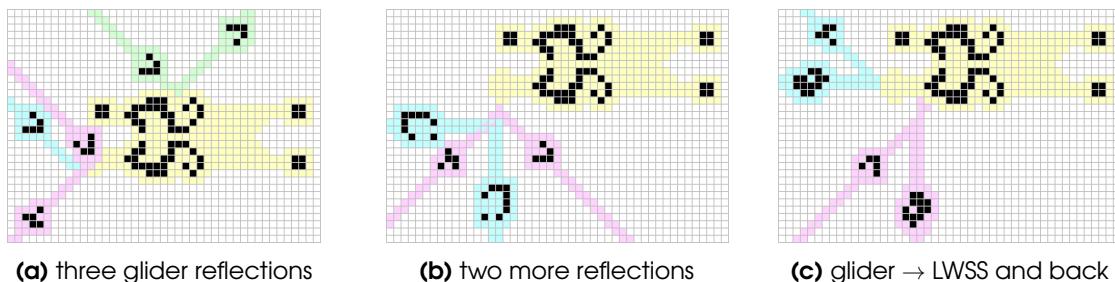


Figure 6.21: The twin bees shuttle (highlighted in yellow) can be used to reflect a glider by 90 degrees in three different ways ((a) green, (a) magenta, and (b) magenta), reflect a glider by 180 degrees ((a) aqua), reflect a lightweight spaceship by 90 degrees ((b) aqua), and turn a glider into a lightweight spaceship and back ((c) magenta and (c) aqua, respectively).

One of the advantages of these new reflections over the old reflection based on the duoplet spark

¹¹This reflection is the one highlighted in green in Figure 6.21(a).

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 6.22(a).¹² Similarly, by making use of the glider-to-LWSS converter we can easily create period 46 glider guns without needing to synthesize them via three glider guns (see Figure 6.22(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 6.22(c) and Exercise 6.14).¹³

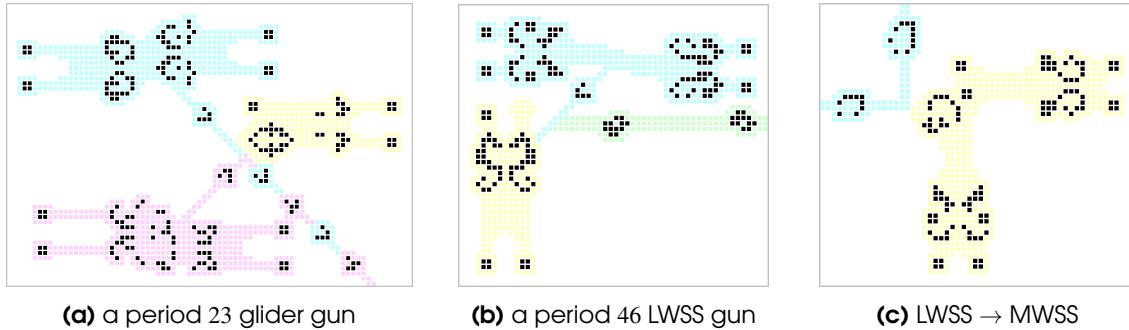


Figure 6.22: The various reflection and conversion reactions involving twin bees can be used to create some useful guns. (a) is a period 23 glider gun constructed by reflecting one period 46 stream into the gaps in another period 46 stream, (b) is a lightweight spaceship gun that uses the glider-to-LWSS conversion from Figure 6.21(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-barreled guns*), and explore some of these reactions in Exercise 6.20.

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 6.23 in which the shuttle reflects a glider by 90 degrees and simultaneously duplicates it. This is somewhat smaller and simpler than the glider duplicator that we saw earlier in Figure 7.10(a), but of course it has the disadvantage of only working at period 46.

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 6.24 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.¹⁴ Thus we can 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 (with different sequences of gliders in their glider loops) near each other, we can even create *ticker tape guns*, which emit any *visual* message of our choosing, rather than just encoding messages in binary. To do this, we think of lightweight spaceships as pixels, with

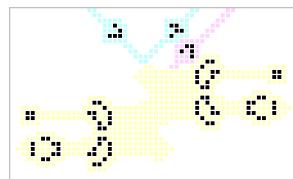


Figure 6.23: A twin bees shuttle reflecting and duplicating a glider.

¹²We say that this is a *pseudo*-period 23 gun since the gun itself operates at period 46. We discuss pseudo-period guns and true-period guns (which oscillate at the same period as the glider streams they produce) in depth in Chapter 8.

¹³By replacing the input LWSS by an MWSS, this LWSS-to-MWSS conversion can also be used to simply reflect a MWSS by 90 degrees.

¹⁴This gun produces lightweight spaceships via the glider-to-LWSS reaction of Figure 6.21(c) instead of just gliders themselves just to make it easier to visualize the output.

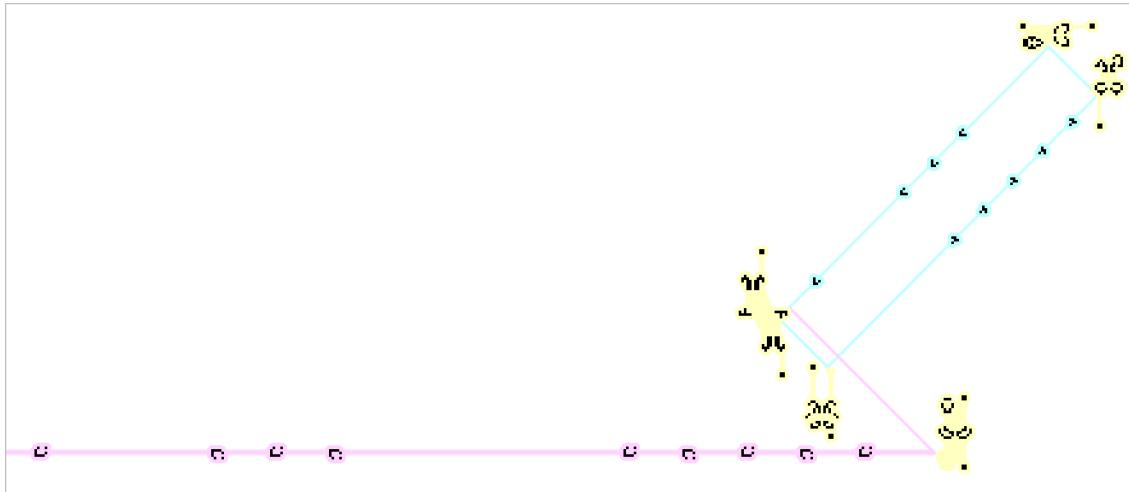


Figure 6.24: A gun that produces an irregular lightweight spaceship stream corresponding to the bitstring 100111000011111000000 (i.e., one LWSS present, then two missing, then three present, and so on). This stream (highlighted in magenta) is produced simply by duplicating the gliders in the loop (highlighted in aqua) that have this same spacing and then performing a few basic reflections and a glider-to-LWSS conversion via twin bees shuttles (highlighted in yellow).

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 gliders each in the loops to create an $n \times m$ image. For example, Figure 6.25 illustrates a ticker tape gun that repeatedly produces an 8×21 array of lightweight spaceships that spells out “HI!”.¹⁵

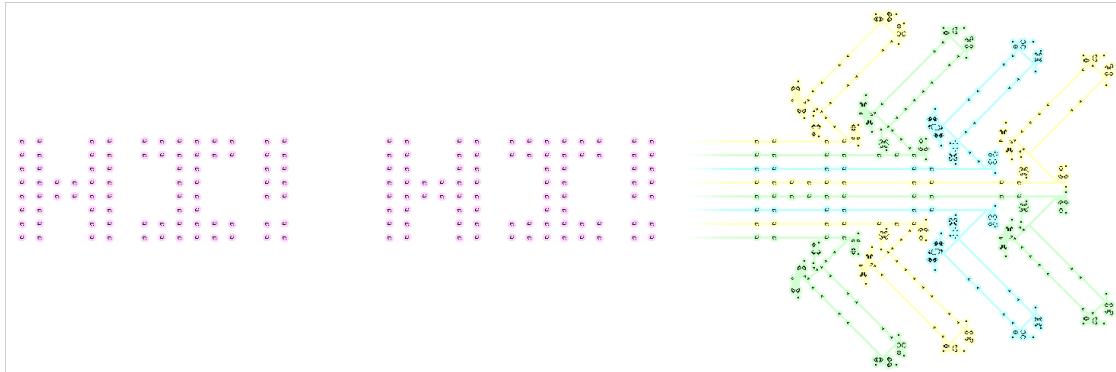


Figure 6.25: An arrangement of 8 loop-based guns that produce an array of lightweight spaceships that spell out “HI!” (highlighted in magenta). Each gun produces one row of the output image (highlighted in yellow, green, and aqua), and the arrangement of gliders in each loop controls which lightweight spaceships are produced in their row.

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 6.24 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).

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. Furthermore, if we replace two of the

¹⁵ 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.

reflectors in the loop with an inverter (such as the one that we used in Figure 6.6(b)), we can then insert gliders into the loop simply by deleting gliders from its inverted portion. This completed pattern is called a *memory cell*, and it is illustrated in Figure 6.26 along with the mechanisms for altering the contents of its memory.

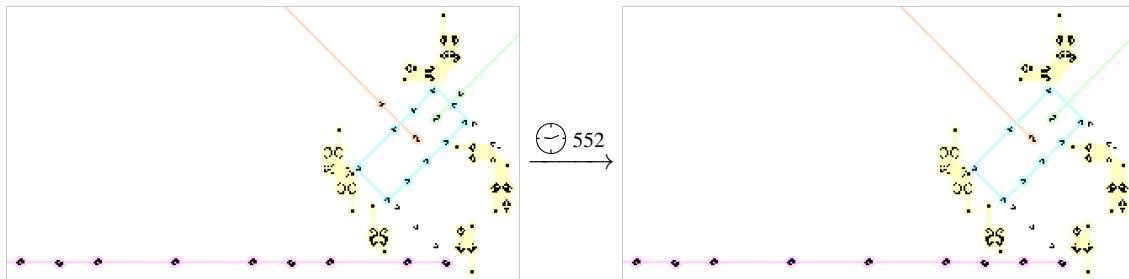


Figure 6.26: A period $12 \times 46 = 552$ memory cell for which the 12-bit string “111010111011” is encoded along its glider loop (highlighted in aqua) and is thus repeatedly fired out as a stream of lightweight spaceships (highlighted in magenta). Bits can be switched from 1 to 0 (i.e., gliders can be deleted) simply by sending in a glider from the northwest (highlighted in orange) 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 “111010101111”.

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 an extremely sparky period 46 oscillator called *Tanner's p46*,¹⁶ which is displayed in Figure 3.19(c). It works by using some still lifes to hassle and stabilize a pi-heptomino on 3 of its 4 sides. In fact, the boat with tail and snake on its southern side are used to rotate the pi-heptomino in the exact same way as in the p32 “gourmet” oscillator that we saw back in Figure 3.19(a).

As is typical of extremely sparky oscillators, Tanner's p46 can be combined with other oscillators or additional copies of itself to create numerous useful objects. For example, it can be combined with a single twin bees to create a new p46 glider gun (see Exercise 6.18), or it can be used to reflect a glider by 180 degrees (see Exercise 6.19), 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. However, by carefully placing two copies of itself next to each other, it can be used to construct an *edge-shooting glider gun*¹⁷ (see Figure 6.27(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 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 new glider stream).

Furthermore, this glider gun can act as a p46 inline inverter (see Figure 6.27(b)) in the exact same way that the Gosper glider gun can act as a p30 inline inverter.¹⁸ Even more remarkably, two of these oscillators can be placed next to each other so as to create the p46 middleweight spaceship gun¹⁹ displayed in Figure 6.27(c), 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 6.27(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 6.28.

¹⁶Found by Tanner Jacobi in October 2017.

¹⁷Found by David Bell just three days after Tanner's p46 itself was found.

¹⁸This inline inverter, just like the original p30 one, was found by David Bell.

¹⁹Found by Matthias Merzenich just one day after Tanner's p46 was found.

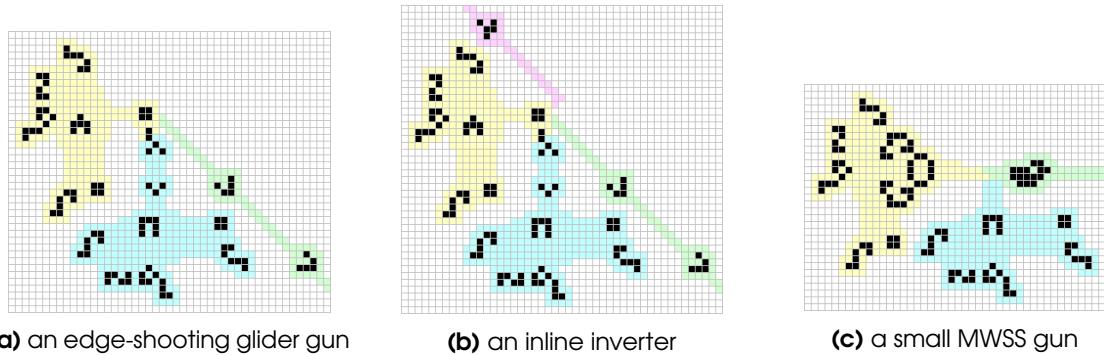


Figure 6.27: Two guns formed by placing two copies of Tanner's p46 next to each other so that their sparks interact. The glider gun in (a) can be used as an inline inverter, as in (b), where the incoming glider (highlighted in magenta) prevents a glider from being fired from the gun at that point in the stream. The gun in (c) is the smallest known MWSS gun.

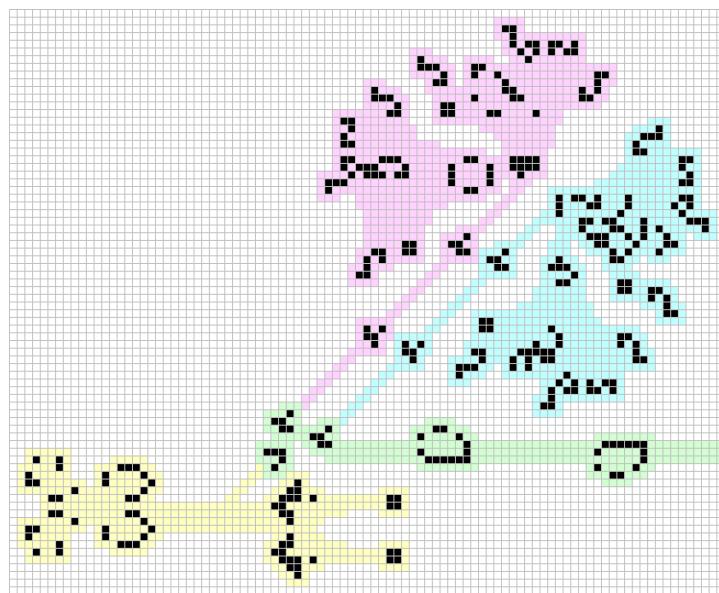


Figure 6.28: A period 46 heavyweight spaceship gun that uses two copies of the edge shooter from Figure 6.27(a) (highlighted in aqua and magenta) to place the required gliders in the HWSS synthesis close together.

6.3.3 Heisenburps

We saw back in Figure 7.10(a) 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.²⁰ 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.

Duplicators of this type are called *Heisenburps*²¹, and by far the simplest one known²² is displayed in Figure 6.29. 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

²⁰In fact, we saw that there are *stationary* circuits that duplicate gliders way back in Figure ??, but these circuits are still a bit more mysterious.

²¹Named for Heisenburg'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.

²²Found by Brice Due in January 2005.

other nearby cells (much like how we used induction coils to overpopulate and stabilize objects in Section 2.2).

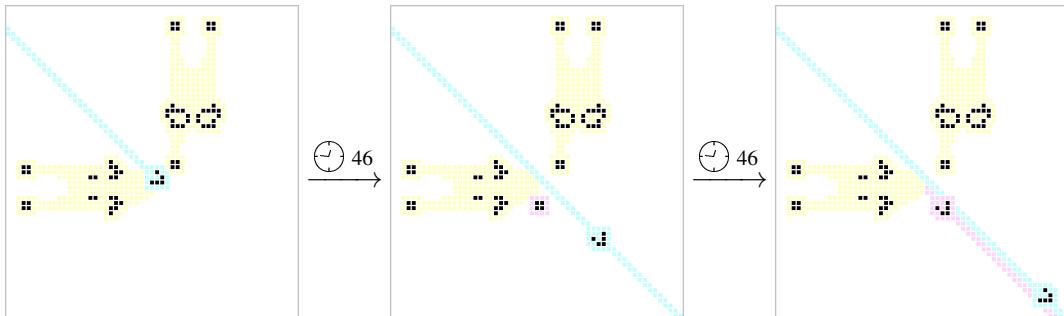


Figure 6.29: A small period 92 Heisenburp that uses two twin bees shuttles (highlighted in yellow) to copy a glider (highlighted in aqua) without affecting it. The first 46 generations are used to create a block (highlighted in magenta) from the glider and the twin bees spark and the next 46 generations are used to turn the block into another glider.

Heisenburps can also 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 6.30, called the *MWSS out of the blue*,²³ detects a passing lightweight spaceship and releasing a middleweight spaceship traveling in the opposite direction in response.

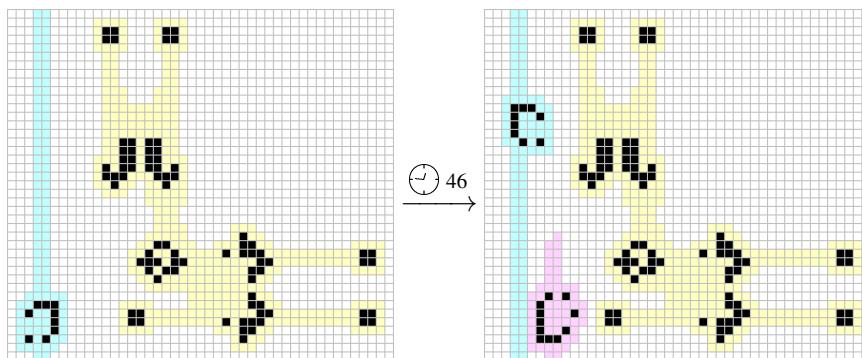


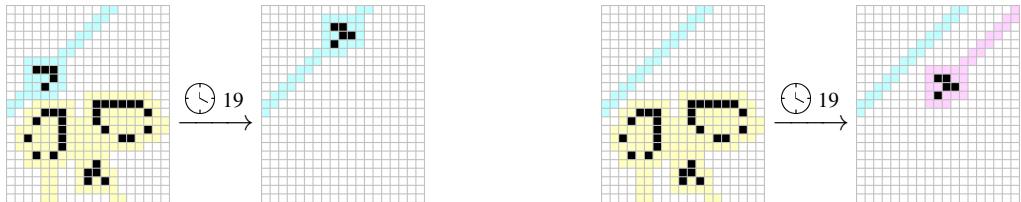
Figure 6.30: The *MWSS out of the blue* is a reaction in which two twin bees shuttles are triggered to create a middleweight spaceship when a lightweight spaceship passes by. The LWSS is not affected at all by the reaction.

There are also some reactions that can be used to *construct* Heisenburps via other periodic circuitry that we have already seen. One of the simplest such reactions is the one displayed in Figure 6.31, 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.²⁴

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 6.32 demonstrates one way to put these pieces together using period 46 circuitry, but any set of sufficiently-high period circuits work.

²³Found by Peter Rott in November 1997.

²⁴Found by Jason Summers in June 1999, as was the first period 46 Heisenburp based on it.



(a) Mutual annihilation if a nearby glider is present.

(b) A single glider is produced if no nearby glider is present.

Figure 6.31: 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 aqua), and (b) a single glider (highlighted in magenta) 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 Heisenurb.

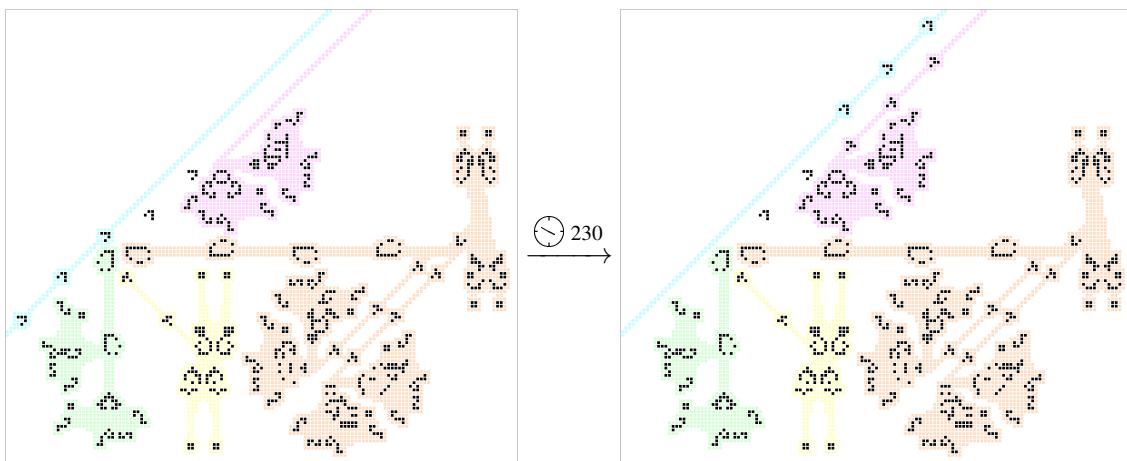


Figure 6.32: A period 46 Heisenurb that uses the glider + MWSS + HWSS reaction from Figure 6.31. These spaceships are all provided by guns that we have already seen—the MWSS gun (highlighted in green) and HWSS gun (highlighted in orange) were introduced in Figures 6.27(c) and 6.28, respectively. The central three-spaceship collision produces a (non-highlighted) glider traveling northeast, which is destroyed by an inline inverter (highlighted in magenta and originally introduced in Figure 6.27(b)). However, if an input glider (highlighted in aqua) is present then no glider is fed into the inline inverter, leading to the input glider's duplication.

In fact, we can use these same ideas to even construct *stationary* Heisenurbs (i.e., Heisenurbs made up entirely of still lifes rather than oscillators and guns) for spaceships that have accessible sparks, like the light/middle/heavyweight spaceships. However, a glider Heisenurb must have an oscillating component, as gliders have no accessible sparks that could be detected by still lifes.

6.4 Bumpers and Bouncers

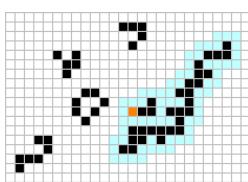
Some periodic circuitry works based only on the existence of certain sparks, and thus can be made to work at any period for which we know of oscillators that emit that spark. For example, the glider-reflecting reactions of Figure 3.17 fall into this category, since any oscillator that produces the correct 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*²⁵ (displayed in Figure 6.33), 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 6.33).

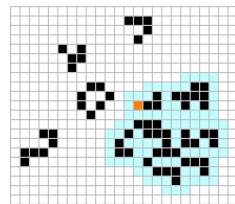
²⁵The p3 bumper was found in April 2018 by Arie Paap, but the reaction itself and most of the other bumpers were found in April 2016 by Tanner Jacobi.



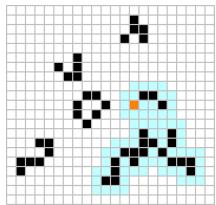
(a) The bumper itself. A spark must be present at the location marked in orange 11 generations after the phase shown here.



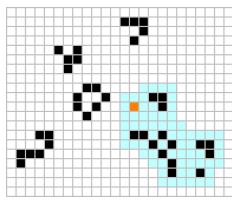
(b) A p3 bumper that uses a custom oscillator.



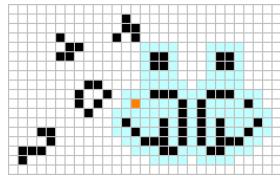
(c) Another p3 bumper that uses a custom oscillator.



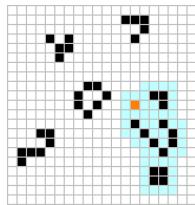
(d) A p4 bumper that uses a custom oscillator.



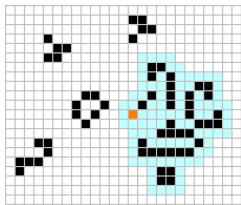
(e) A p6 bumper reflector that uses the unix sparker.



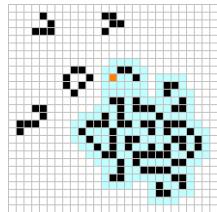
(f) A p7 bumper that uses Beluchenko's p7 Figure 3.13(c).



(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 6.33: 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 above in aqua) and thus can 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 6.23.

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 this reflector is still useful:

- 1) It is considerably 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 (though 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 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).

Another closely-related reaction is called the *bouncer*²⁶ (displayed in Figure 6.34), which consists of an eater 1, a boat, a block, and a domino spark that is typically provided by a pipsquitter. The

²⁶The bouncer as well as most of the low-period pipsquitters that it can use were found by Noam Elkies in September 1998.

bouncer is somewhat less versatile than the bumper since the domino spark is somewhat 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,²⁷ which provides us with some extra flexibility that we did not yet have (recall that both the Snark and the bumper are color-preserving).

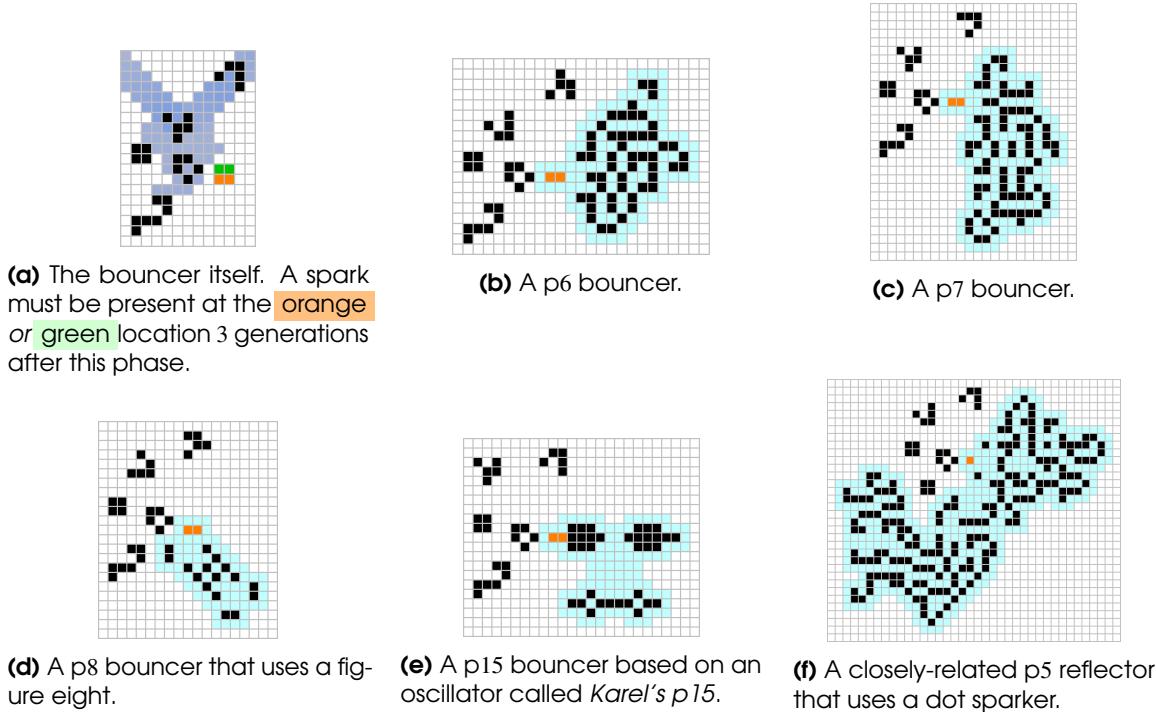


Figure 6.34: Pipsquirters (as well as a handful of other domino sparkers) can be used to with the bouncer to reflect gliders. This allows for the creation of several new reflectors with quite a few periods.

While this family of reflectors only works directly with period of 6 and greater, a very similar reaction can be made to work at period 5, thus giving us our very first period 5 reflector in Figure 6.34(f).²⁸ For a period 5 reflector based on the bumper, see Exercise 6.23.

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. 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.

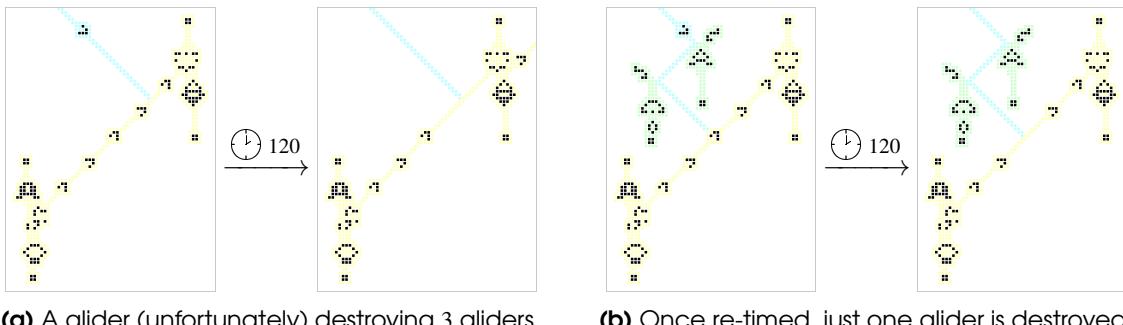
To illustrate how we can overcome this problem, suppose for now that we have a glider at some specific location, and we need to relocate it so that it appears at another pre-specified location with a certain timing. If the input glider is on some kind of schedule—say, it only has a possibility of showing 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 inline inverter of the same period (or another period N gun if necessary), adjust the phase of the period N gun so that it can safely trigger the desired action, and then find a way for the arriving glider to cleanly delete a glider from the stream

²⁷Their names are designed to help us remember which reaction is color-preserving and which one is color-changing—the bumper is color-preserving while the bouncer is color-changing.

²⁸For a more complete and up-to-date collection of bumpers and bouncers, see conwaylife.com/wiki/Bumper_and_bouncer_gallery.

that hits the inverter. The inverter then sends out a glider in response to the resulting hole in the stream, resulting in 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 inline inverter, 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 two reflectors to send the glider some distance in the opposite direction from the perpendicular stream (see Figure 6.35). 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.



(a) A glider (unfortunately) destroying 3 gliders. (b) Once re-timed, just one glider is destroyed.

Figure 6.35: As positioned in (a), the input glider (highlighted in aqua) destroys three of the gliders in the inline inverter (and thus three gliders are released to the northeast). We can “correct” this problem by using two buckaroos (highlighted in green 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.

What if N (the period dictating when input gliders may appear) 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), so instead we could reflect the glider with Snarks and period 8 bouncers and bumpers, which can be used to produce any possible output lane and timing (see Exercise ??).

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? 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 (of any period) as in Figure 6.35. That method will work for some timings, but other timings will result in multiple gliders being deleted or even chaotic explosions.
- We cannot simply reflect the input glider with Snarks, as we can only arrange the Snarks so as to get the first glider on the input stream to have the correct timing, but likely 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 aligned to a particular period, no matter what the precise timing of the input glider is. A mechanism that accomplishes this for a particular 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 discovered universal regulator was built.²⁹ The heart of this regulator is actually the boat-bit from way back in Section 2.3.1. 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 6.36 (a beehive is also created but this can easily be destroyed later).

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 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 6.37, as is the entire glider → boat-bit → synchronized glider conversion process.

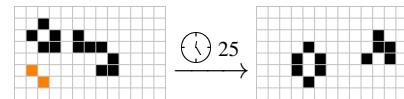


Figure 6.36: A duoplet spark turning a boat-bit into a glider and beehive. If the boat is not present, nothing happens.

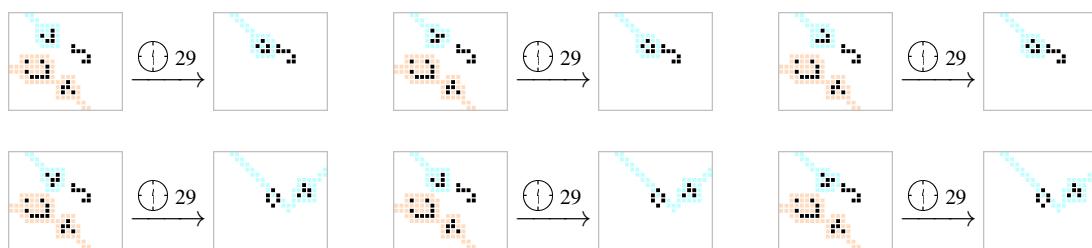


Figure 6.37: An input glider (highlighted in aqua) with six different timings hitting an eater 1 so as to become a boat-bit. The glider-and-LWSS collision (highlighted in orange) creates a duoplet spark that turns that boat-bit into an output glider (as in Figure 6.36). If the timing is such that the boat-bit was already created before the collision occurs (as in the bottom three input timings), the output glider forms 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.

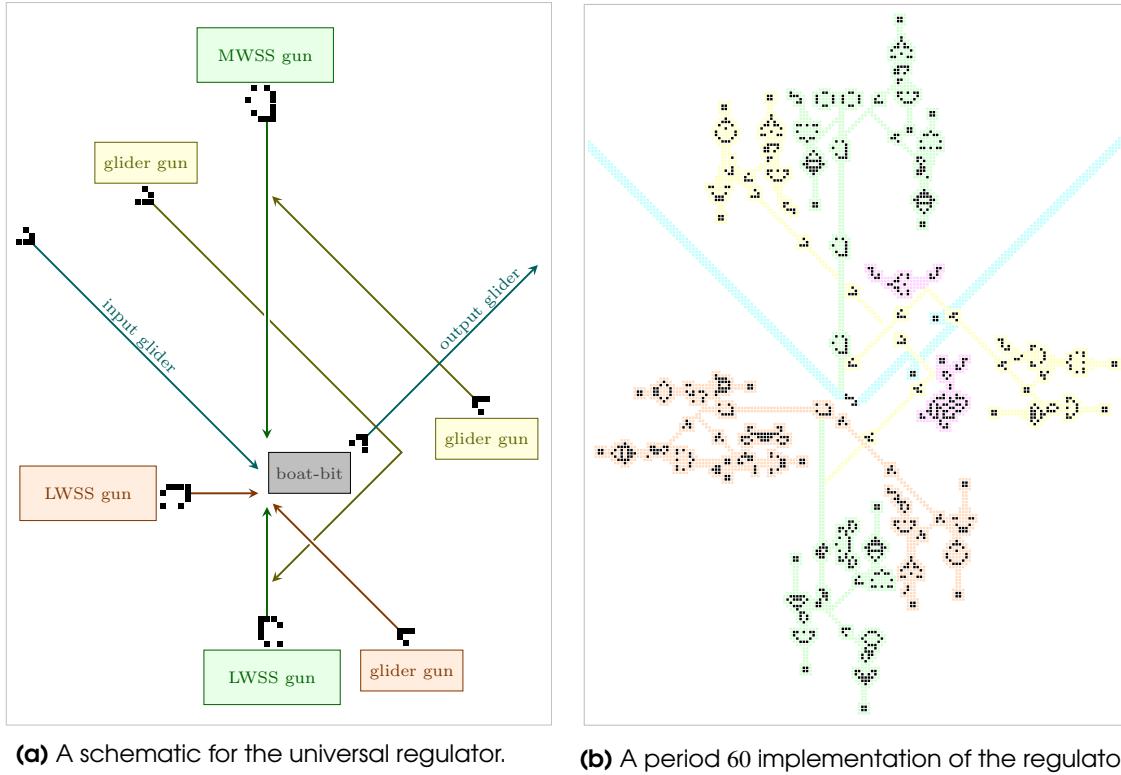
This collision of a glider and an LWSS has the additional advantage of destroying the beehive that was left behind by the duoplet spark in Figure 6.36, so streams of these two spaceships clean up the mess from their own reaction.

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 6.36. 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, we cannot *always* fire these spaceships at each other, since if no input glider is present then they will collide with the eater 1 that has already been synthesized. This problem can be fixed 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 6.38, but we emphasize that a universal regulator with any (sufficiently large) period can be constructed using these same reactions just by swapping out all of the period 60 machinery in this

²⁹Found by Paul Chapman in March 2003.

mechanism 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.



(a) A schematic for the universal regulator.

(b) A period 60 implementation of the regulator.

Figure 6.38: 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 and produces an output glider along the northeast lane with timing that is synchronized to some period (both lanes are highlighted in aqua). The left LWSS gun and bottom-right glider gun (highlighted in orange) are used to create the duoplet spark reaction from Figure 6.37 that is at the heart of the regulator. The top MWSS gun and bottom LWSS gun (highlighted in green) 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). Note that the rightmost suppressing glider path in the final regulator (b) was adjusted for spacing reasons.

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 1990’s, including the first primer (which made use of essentially the same techniques, but slightly different components, 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 6.12). 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 number theory), 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 6.12 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 $b(k) = 0$), then the effect of the toggle is that $b(k) = b(k - 1) \oplus b(k - n)$ for all $k > n$.³⁰ 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 6.39, where the period of the sequence of bits is 10961685 and thus the period of the gun itself is $30 \times 10961685 = 328850550$.

Guns producing wildly different sequences of gliders and with wildly varying periods can be constructed by changing the length of the glider loop (see Exercise 6.29).

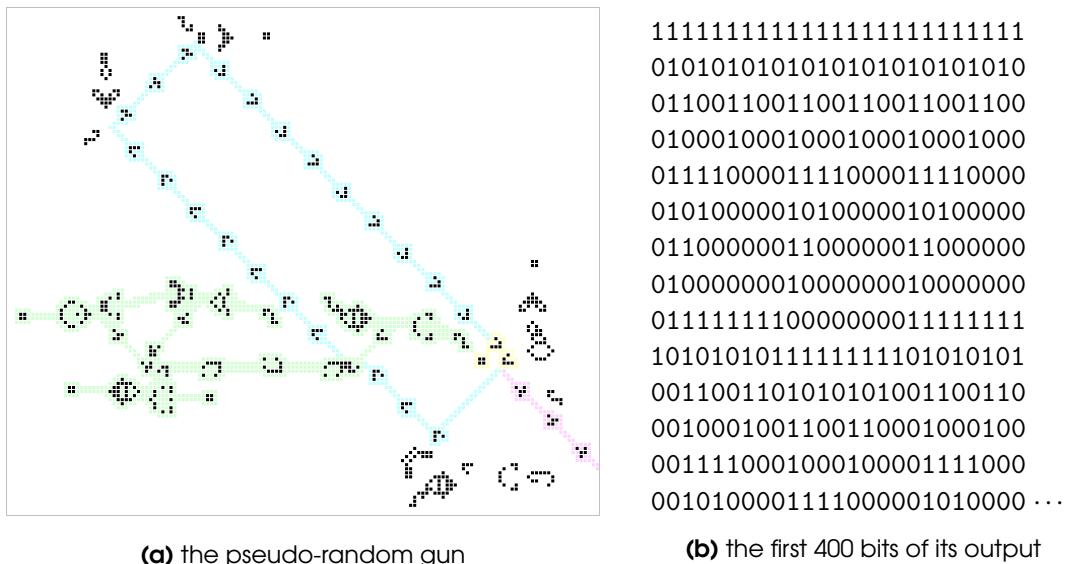


Figure 6.39: A period $30 \times 10961685 = 328850550$ pseudo-random glider gun. It sends gliders around a $25 \times 30 = 750$ -generation track (highlighted in aqua) that contains a toggle (highlighted in green) and the glider duplication/inversion reaction introduced in Figure 6.7 (highlighted on the right in yellow). 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 magenta) 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 $(t/(2(n+2)), 0)$, 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 6.40).

³⁰Here, \oplus refers to the XOR or mod 2 addition of two bits. That is, $0 \oplus 0 = 1 \oplus 1 = 0$ and $0 \oplus 1 = 1 \oplus 0 = 1$.

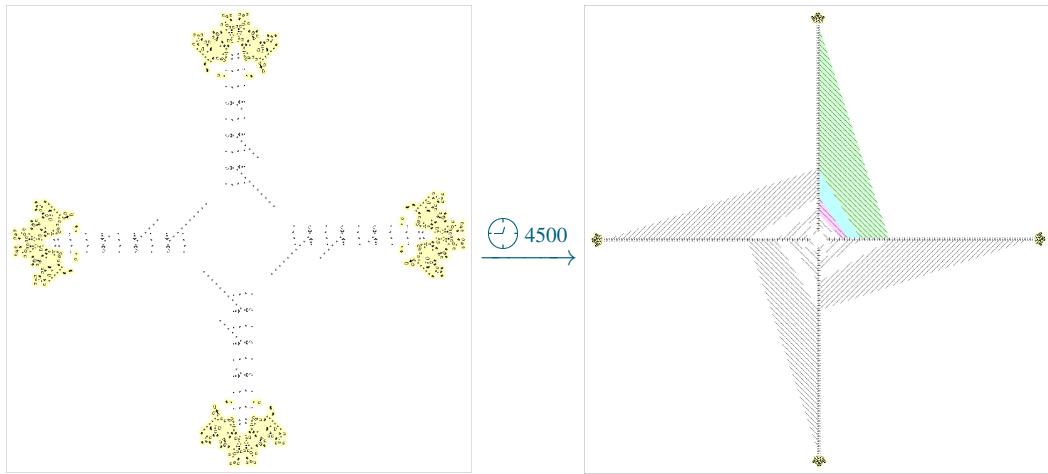


Figure 6.40: An arrangement of four breeders (highlighted in yellow) that produce Gosper glider guns that fire at each other so as to invert each others' streams (shown on the left after the breeders have been moving away from each other for 500 generations). In generation t (shown here on the right with $t = 5000$), the green triangular region $T(1,t)$ is filled with gliders, except the aqua triangular region $T(3,t)$ is empty, except the magenta triangular region $T(5,t)$ is filled with gliders, and so on.

Since the triangle $T(n,t)$ has area

$$\frac{1}{2} \times \frac{t}{2n} \times \frac{t}{2n+4} = \frac{t^2}{16} \left(\frac{1}{n} - \frac{1}{n+2} \right),$$

the occupied area in the first quadrant in generation t (when t is large) is approximately

$$\begin{aligned} T(1,t) - T(3,t) + T(5,t) - \dots &= \frac{t^2}{16} \left(\left(1 - \frac{1}{3}\right) - \left(\frac{1}{3} - \frac{1}{5}\right) + \left(\frac{1}{5} - \frac{1}{7}\right) - \dots \right) \\ &= \frac{t^2}{16} \left(1 - \frac{2}{3} + \frac{2}{5} - \frac{2}{7} + \dots \right) \\ &= \frac{t^2}{16} (2 \arctan(1) - 1) = \frac{(\pi - 2)t^2}{32}, \end{aligned}$$

where the second-to-last equality follows from the fact that $\arctan(x)$ has Taylor series $\arctan(x) = x - x^3/3 + x^5/5 - \dots$, so $\arctan(1) = 1 - 1/3 + 1/5 - \dots$, and the final equality holds because $\arctan(1) = \pi/4$.

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 1982 [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 almost a decade 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 proof (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 the components in Conway and

Gosper's constructions so as to have the correct positioning and timing would typically be infeasible in practice, and the resulting patterns would not be terribly interesting to look at.

Exercises

solutions on page 309

6.1 In the glider gun in Figure 6.6(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 Construct a glider gun based on the inline inverter with...

- (a) period 240.
- (b) period 360.

6.3 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 Construct a period 240 lightweight spaceship gun...

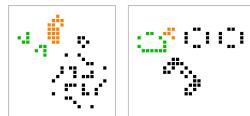
- (a) using three inline inverter guns.
- (b) using one inline inverter gun and the reaction from Figure 6.10.

6.5 Use the reaction from Figure 6.7, together with reflectors and eaters as necessary, to create an inline inverter with for a glider stream with...

- (a) period 43.
- (b) period 36.
- (c) period 25.

6.6 A pair of positive integers $(p, p+2)$ is called a *twin prime pair* if they are both prime. Find a way of aiming a p240 glider gun at the output LWSS stream of the primer from Figure 6.20 so that the resulting LWSS stream now corresponds to twin prime pairs, rather than all prime numbers. [Hint: Position the glider gun so that the glider destroys an LWSS, creating a small still life that then blocks the next glider (and lets the next LWSS, if present, pass). Do not worry about the first two twin prime pairs ((3, 5) and (5, 7) are the only twin prime pairs that overlap), and do not worry about whether the gun outputs the lower or upper number in each twin prime pair.]

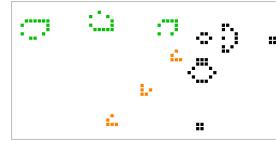
6.7 The periodic circuits displayed below can be used to convert two gliders into an HWSS and an HWSS back into a glider.³¹



- (a) Use the left reaction to create a period 120 HWSS gun.
- (b) Create a glider loop that uses both of these reactions to convert the glider into an HWSS and back.

³¹These reactions were found and simplified by a combination of David Buckingham, Bill Gosper, Dieter Leithner, and Peter Rott.

6.8 The arrangement of queen bees below can be used to convert any of the lightweight, middleweight, or heavyweight spaceships into gliders.



- (a) Use this reaction as well as one of the reactions from Exercise 6.7 to create a glider loop that converts a glider into an HWSS and back.
- (b) Use this reaction to create a gun that emits a glider stream (rather than an LWSS stream as in Figure 6.20) that is spaced according to the prime numbers.

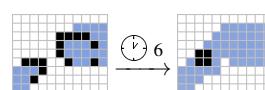
6.9 Construct an oscillator that makes use of the reactions in each of Figures 6.10 and 6.11 to send a glider around a track in such a way that it is converted into an LWSS and back during each loop.

6.10 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.

6.11 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 size of the bounding box of the primer from Figure 6.20 from $951 \times 696 = 661,896$ to 500,000 or less.

6.12 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 the 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.



6.13 Use the patterns from Figure 6.21 to create a loop oscillator in which a glider is converted into a lightweight spaceship and back at some point in the loop.

6.14 Use the patterns from Figure 6.22 to construct a period 46 middleweight spaceship gun.

6.15 Modify the sequence of gliders in the loop of the gun from Figure 6.24 so that it produces lightweight spaceships corresponding to the bitstring “110111010011011101001”.

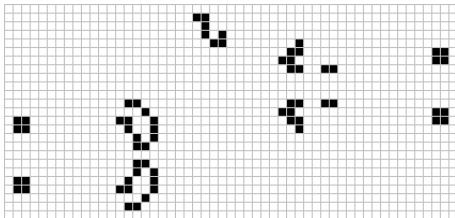
6.16 Using the techniques used in Figure 6.25, create a ticker tape gun that produces smiley faces.

6.17 Fire some gliders at the memory cell from Figure 6.26 so as to change the bitstring that it stores to “101010101010”.

6.18 Create a glider gun that makes use of a single copy of Tanner’s p46 and twin bees that are stabilized only on one side. [Hint: You want the spark from these two objects to hit each other. This gun can be found by hand via trial error, or you could write a computer program to try lots of positionings.]

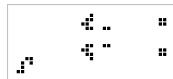
6.19 Show how Tanner’s p46 can be used to reflect a glider by 180 degrees. [Hint: The spark emitted by Tanner’s p46 is quite similar to some of the sparks emitted by the pentadecathlon. Mimic the reflection from Figure 6.3(a).]

6.20 An arrangement of two twin bees shuttles that is particularly useful due to the extremely large spark that it produces is displayed below.



(a) 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) Collide the spark from this object with the following arrangement of twin bees and an eater 1 to create a double-barreled glider gun (i.e., a gun that emits two side-by-side gliders per period).³²



6.21 Use the 9-glider synthesis from Exercise 5.14 to construct a gun that fires 2-engine Corderships.

6.22 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.

³²This double-barreled gun was originally found by Dieter Leithner.

³³This reflector was originally found by Matthias Merzenich in August 2013.

6.23 Construct a bumper reflector with...

- (a) period 16.
- (b) period 22.
- (c) period 15. [Hint: Try using the sparker from Figure 6.34(e).]
- (d) period 5. [Hint: Try modifying the sparker from Figure 6.34(f).]
- (e) period 4, making use of the fountain sparker.

6.24 Explain why the period 11 bumper from Figure 6.33(i) is not useful for glider streams of period below 121. [Hint: You can use smaller bumpers in place of it.]

6.25 The smallest-period single-glider Snark loop has period 216 and is displayed in Figure 6.41. Note that the still life in the center of this loop is the weld of four eater 1s from Exercise 2.9(c).

- (a) 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) 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) Replace one of the p4 bumpers in the glider loop from part (b) with a p6 bumper.
- (d) 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) Move 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.

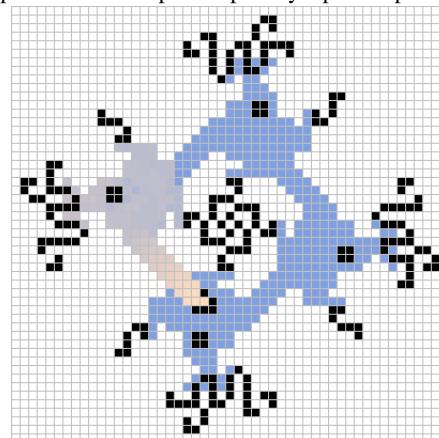
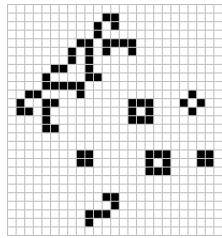


Figure 6.41: A single-glider loop built from Snarks with the smallest possible period—216.

6.26 Use the *skewed p29 pre-pulsar shuttle* displayed below to create a p29 bouncer reflector.³³



6.27 The lightweight spaceship guns in Figure 6.38(b) each contain a middleweight emulator. Describe what role this oscillator plays in the gun.

6.28 Two reflectors are highlighted in magenta in Figure 6.38(b). Identify these reflectors (we have discussed them each earlier).

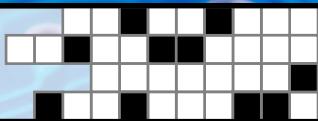
6.29 Changing the number of gliders in the loop portion of the pseudo-random glider-generating gun from Figure 6.39 can drastically change its period.

- (a) 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) Determine the period of the gun constructed in part (a). [Hint: Computer software may be useful here—it is massive.]
- (c) 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.]

Early draft (May 19, 2020).

Not for public dissemination.

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

$$\underbrace{4 \times 64}_{\text{R64 time}} + \underbrace{28 \times 77}_{\text{Fx77 time}} = 2,412 = 36 \times 67 \text{ generations,}$$

(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

¹We saw one of these extra conduits, L112, way back in Exercise 3.26.

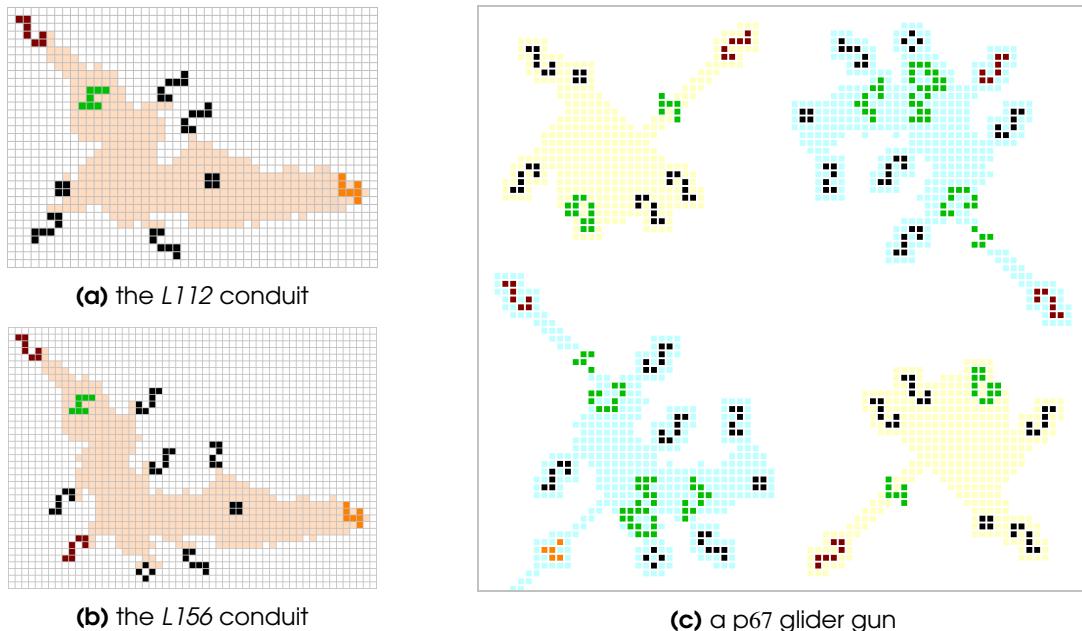


Figure 7.1: A period 67 glider gun can be constructed from just four conduits. The track in (c) uses two copies of the L112 conduit from (a) (highlighted in yellow) and two copies of the L156 conduit from (b) (highlighted in aqua). Both of these conduits rotate an input Herschel (displayed in green) 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, but are not required for the conduits to function.

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 right in **64** generations. More generally, the prefix used when naming conduits is one of:

R: right (clockwise) turn
L: left (counter-clockwise) turn

F: forward (no turn)
B: backward (180-degree turn)

Thus the L112 and L156 conduits are named for the fact that they rotate a Herschel left (counter-clockwise) in 112 and 156 generations, respectively. As one final piece of naming terminology, we insert an “x” between two parts of the conduit’s name if it mirrors the Herschel (as in the Fx77 conduit from Figure 3.34(b)),³ so a conduit can 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.

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

²Be 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.

³To 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).

⁴For a reasonably complete collection of known small Herschel conduits, see conwaylife.com/forums/viewtopic.php?f=2&t=2347

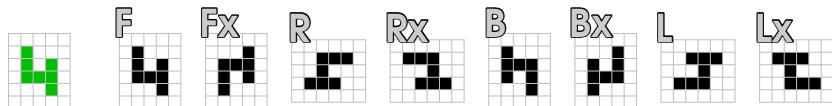


Figure 7.2: Herschels can be oriented in one of 8 ways, which we label here relative to the (arbitrarily-chosen) canonical input orientation displayed on the left in green.

conduits release at least one glider and thus can 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 back in the period 67 gun from Figure 7.1). It is also worth noting where the R64, Fx77, L112, and L156 conduits that we have already seen fit into this table.

	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.

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 this phenomenon, consider the variants of the Fx77 conduit displayed in Figure 7.3.

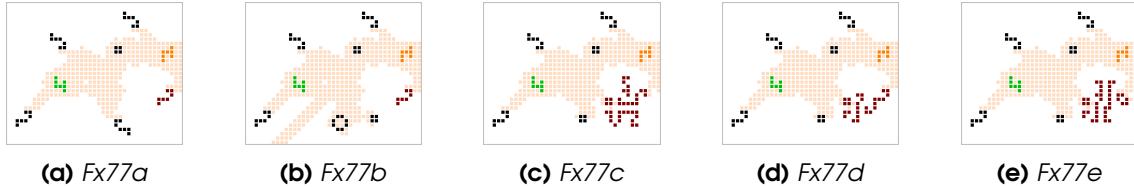


Figure 7.3: Five variants of the Fx77 Herschel conduit, along with eaters or welds (displayed in red) that are capable of destroying the first block left behind by the output Herschel. The “a” variant is the one that we have used up until now (e.g., in Section 3.6 and Table 7.1).

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)⁵—this eater 1 is used 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 the opposite orientation (see Figure 7.4). We note that the “e” variant of Fx77 can also be used in the same situations as the “c” variant, but not the “d” variant.

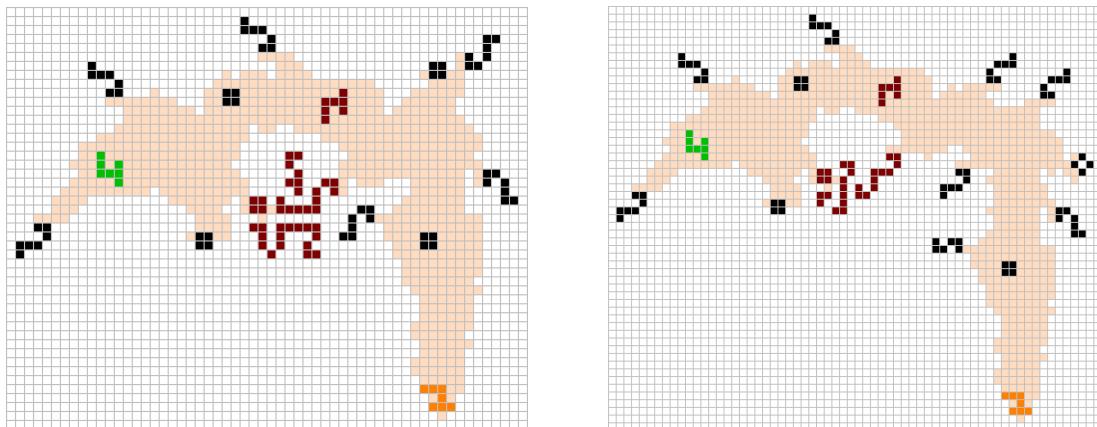


Figure 7.4: We can attach (a) L112 to the output of Fx77c and (b) L156 to the output of Fx77d, but not L112 to Fx77d or L156 to Fx77c.

⁵This block-eating eater 1 can be seen several times in Table 7.1: it is directly above the green input Herschel in each of F117, Fx77, R126, Rx164, B245, Bx106, Bx202, L112, L156, Lx86, and Lx163.

7.2 Gliders to Herschels and Back

In order 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.⁶ Dozens of these types of conduits are also known, some of which even produce multiple gliders traveling 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 gliders traveling in any of the four desired output directions.⁷

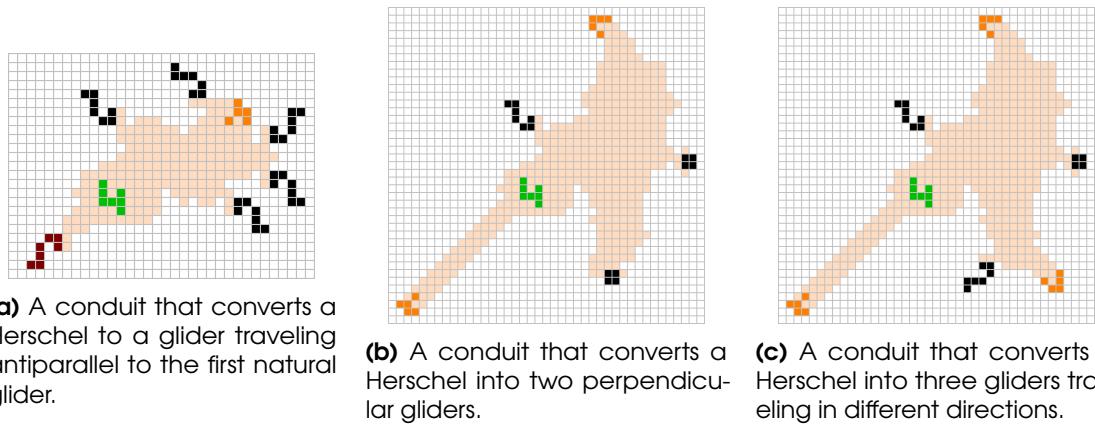


Figure 7.5: A small collection of converters that transform a Herschel (displayed in green) into one or more gliders (displayed in orange). Any unwanted gliders from the last two converters can simply be destroyed by an eater (as we did in (a) via the eater displayed in red).

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 6.27(a)), so too do Herschel-to-glider converters. This makes the conduits in Figures 7.5(b) and 7.5(c) especially useful—the glider that they release to the northwest occupies a diagonal lane so close to the edge of the conduit that it 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). Despite not being an edge-shooter, the conduit from Figure 7.5(a) is useful for a similar reason. In particular, the output glider lane and one other nearby lane are *transparent*—gliders on that lane can pass safely through the circuit without interacting with it. Thus this conduit can be used to insert gliders into a stream just like an edge-shooter, as in Figure 7.6.⁸

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 Herschel conduits contain 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.

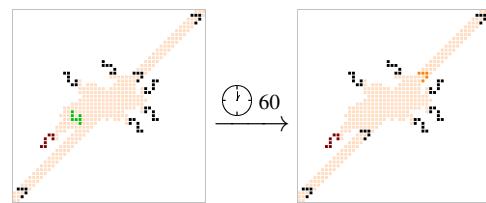


Figure 7.6: A conduit with a transparent lane inserting a glider into a period 60 stream.

⁶We 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.

⁷There are also some simple conduits that turn a Herschel into four gliders, all traveling in different directions—see Exercise 7.21.

⁸A slight variant of this converter that is better in some situations is presented in Exercise 7.4.

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. The `<lane>` quantity is the lane number of the output glider, and similarly `<timing>` is its timing, also relative to the input Herschel.⁹ For example, the conduit in Figure 7.5(a) is called HNE5T-4 (to be clear, the dash in this name is a minus sign, not a separator—the timing of the output glider is -4) and the ones in Figures 7.5(b) and 7.5(c) are both called NW31T120.¹⁰ 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.

With all of this taken care of, some other Herschel-to-glider edge shooters are displayed in Figure 7.7.¹¹

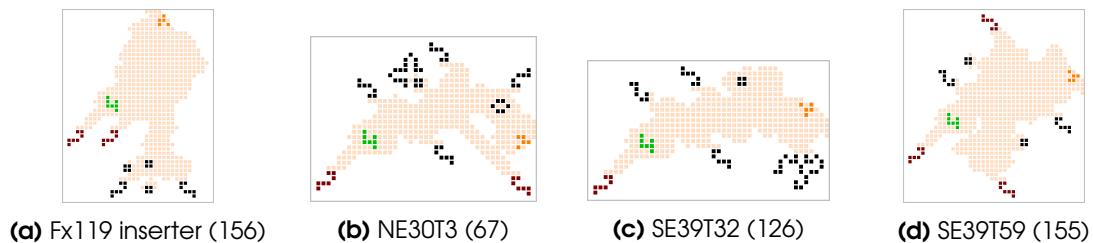


Figure 7.7: Some edge-shooting Herschel-to-glider converters. The number in parentheses is the repeat time of the conduit. Note that (a) is the Fx119 Herschel conduit from Table 7.1 with one more eater and all four of these edge shooters can fire additional gliders by erasing the eaters in red. Also, (b) is not strictly speaking an edge-shooter, but the only part of it southeast of the escaping glider is a single eater than 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.

7.3 Glider-to-Herschel converters

Despite the simplicity of converting a Herschel into a glider, the reverse conversion of a glider into a Herschel is much trickier to implement. For many years the best known G-to-H (glider-to-Herschel) converter was a Herschel-based staged recovery mechanism constructed by Paul Callahan in 1998. A glider striking a beehive can be catalyzed to create a replacement beehive in the same location. However, an extra "junk" beehive also appears that must be removed. The required cleanup glider can be created by appending Fx77 + L112 + Fx77 conduits to the base converter, making a *Callahan G-to-H*. Because of the slow creation of the cleanup glider, the repeat time is a rather high 575 ticks.

Stephen Silver immediately found a variant of the Callahan G-to-H that became the smallest known 90-degree *stable reflector* for well over a decade. The *Silver reflector* took advantage of the fact that a standard NW31 converter produces an output glider on the same lane as an Fx77 conduit, but much sooner, giving a Silver reflector a repeat time of 497 ticks.

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 construct them as to construct 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.

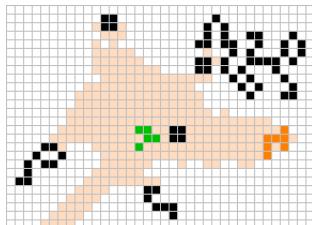
⁹Refer back to Section 4.1.2 if you need a refresher on glider lanes and timing.

¹⁰The first and second natural gliders (i.e., the southwest and northwest gliders in Figure 7.5(c)) are not included in the name of a Herschel-to-glider conduit unless they are its only output.

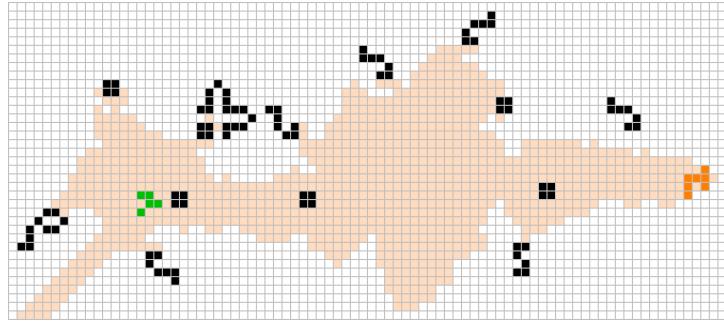
¹¹A more-or-less complete collection of Herschel-to-glider converters is available for download at conwaylife.com/forums/viewtopic.php?f=2&t=1682

- Silver reflectors can be trivially adjusted to produce output gliders in any or all of the four diagonal directions, 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.

Many searches have been run for improved glider-to-Herschel converters in the years since 1998, but so far only one small and fast stable converter has been found, called the *syringe*.¹² Two slightly different versions of it are displayed in Figure 7.8.¹³



(a) The “standard” version of the syringe, and which has repeat time 78 and takes 84 generations to convert the glider into a Herschel.



(b) A larger version that has a repeat time of 115 generations and takes 250 generations to convert the 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.

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 what is currently the smallest and fastest known color-changing stable reflector in Figure 7.9 (recall that the Snark is color-preserving).¹⁴ Explicitly, this reflector uses a syringe to convert the input glider into a Herschel, which is then converted back to a glider (with opposite color and rotated 90 degrees from the input glider). However, since its repeat time is 78 generations (though it also works at a gap of 74 or 75 generations, but *not* 76 or 77 generations, in a phenomenon called *overclocking*), the smaller and faster bouncer reflectors from Section 6.4 are still more useful in many situations.

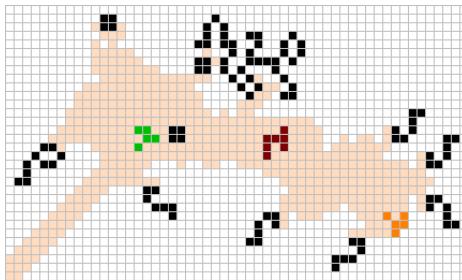


Figure 7.9: A reasonably small and fast color-changing stable reflector with a repeat time of 78 generations, based on the syringe. The input glider (shown in green) comes in from the bottom-left, is converted into a Herschel (shown in red in 84 generations, and then is converted into the output glider (shown in orange) after 53 more generations.

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,

¹²The 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.

¹³A third, slightly more compact, version of the syringe is presented in Exercise 7.6.

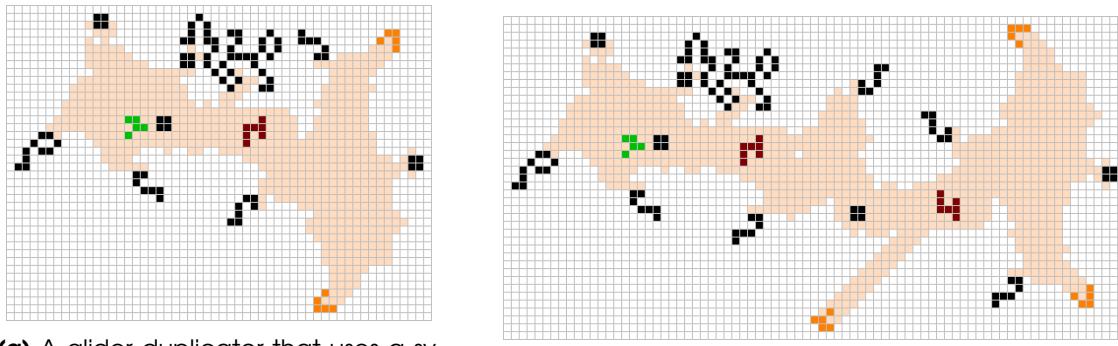
¹⁴Simon Ekström compiled a large collection of stable glider reflectors based on glider-to-Herschel-to-glider conduits, which is available at conwaylife.com/forums/viewtopic.php?f=2&t=1643&start=50#p21708

signal-splitting ability, 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 be larger than a Silver reflector!

We will see examples of uses of Silver reflectors and syringes in Chapter 9.

7.4 Synthesizing Objects via Conduits

To demonstrate some even more extreme examples of how useful the syringe is, we now describe how we can use it to construct conduits that convert a glider into (almost) any other pattern. As a starting point, let's show how to convert a glider into *two* 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 and not destroying the gliders that it naturally emits. 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 7.10 and Exercise 7.8).



(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 7.10: Some conduits that use the syringe to duplicate (or just generally multiply) 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.

Now that we know how to construct conduits that convert one glider into any number of gliders, we can use glider synthesis to create conduits that convert a single glider into any pattern that we know how to synthesize—all we have to do is first convert the 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 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. One such collection of conduits is presented in Table 7.2.

Now that we can use stable conduits to reposition gliders however we like (via the Snark and the color-changing reflector of Figure 7.9) *and* adjust the timing of gliders however we like (via the collection of reflectors 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

		Delay							
		0	1	2	3	4	5	6	7
Color-Preserving (CP)	Snark								
	Color-Changing (CC)								
Color-Preserving (CP)									
	Color-Changing (CC)								

Figure 7.9:

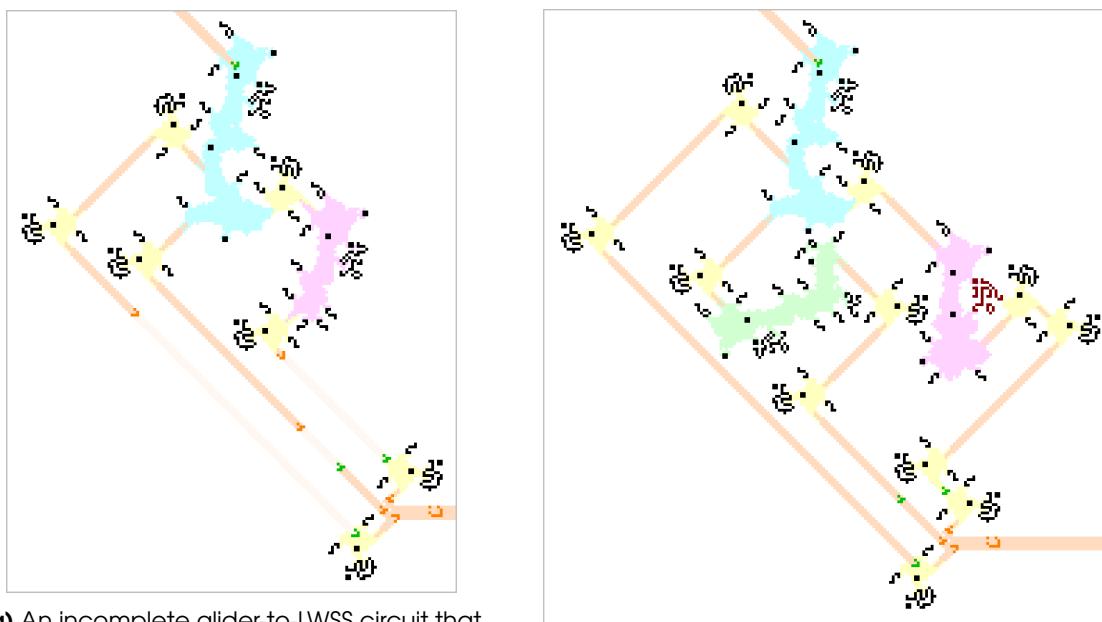
Table 7.2: A collection of glider rephasers that, together with the Snark, can be used to put gliders into any timing and color relative to each other that is desired (compare with Table 5.5, which did the same thing via one-time-turners instead of stable circuits). In all cases, the input glider is highlighted in green and comes in from the top-left, while the location of the output glider is highlighted in orange. To get the output gliders travelling in the desired direction or to adjust their timing by a multiple of 8 generations, just use additional Snarks (e.g., build a trombone slide as in Figure 7.12).

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 7.11(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 7.10(b) to turn the input glider into three gliders. This glider tripler is located at the north end of Figure 7.11(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 did not match that of the lane we needed to put it in. At this point, we have constructed the pattern displayed in Figure 7.11(a)—we have put the three gliders into the correct positions, but we still need to fix their timing.



(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 the rephasers from Table 7.2 and trombone slides to correct the glider timings.

Figure 7.11: 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. Sharks (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).

- 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, 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.
- 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 7.12 (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 7.11(b).¹⁵

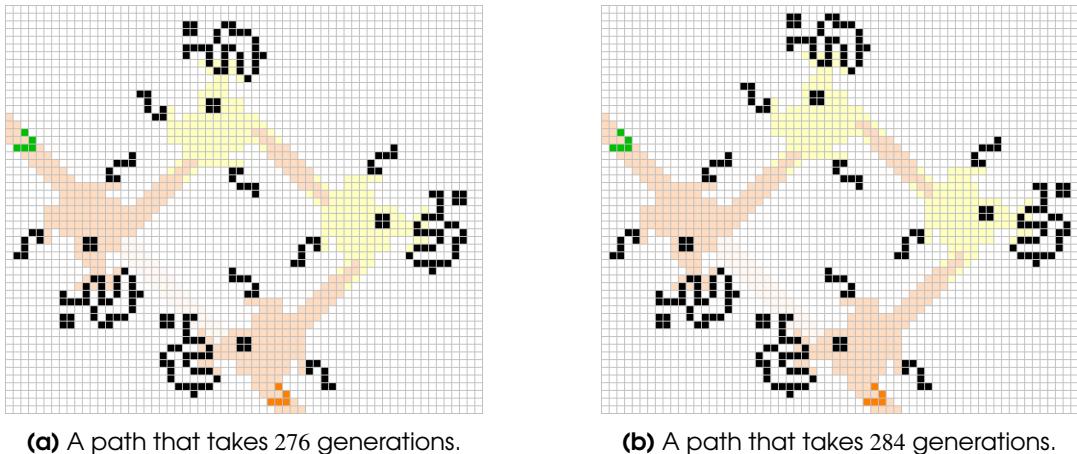


Figure 7.12: A *trombone slide* is a 180-degree reflector can be freely moved closer or farther away along a glider’s path, delaying the glider by 8 generations for each cell that it is moved away. The only difference between (a) and (b) is that we moved the north and east Snarks (highlighted in yellow) northeast by 1 cell in (b), thus delaying the glider by 8 generations. Note that placing the four Snarks along the glider’s path in (a) in the first place delayed it by 144 generations—this delay can be made up for (if necessary) by similarly delaying all other gliders that we are working with.

These ideas 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 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 4.19.

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 7.11(a) when constructing the glider-to-LWSS circuit). This bottom portion of our circuit is displayed in Figure 7.13.

Next, we need to turn one glider into 10 gliders and synchronize them with the positions indicated in Figure 7.13. While it is possible to do this “directly”, it is easier to instead just synchronize just 2 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 7.14. Once we place all 4 of these mechanisms, we just have to synchronize 4 gliders instead of 10—one for each of the mechanisms.

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 7.15.

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 7.10 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

¹⁵Somewhat smaller stable glider-to-LWSS converters are known—see Exercise 8.10.

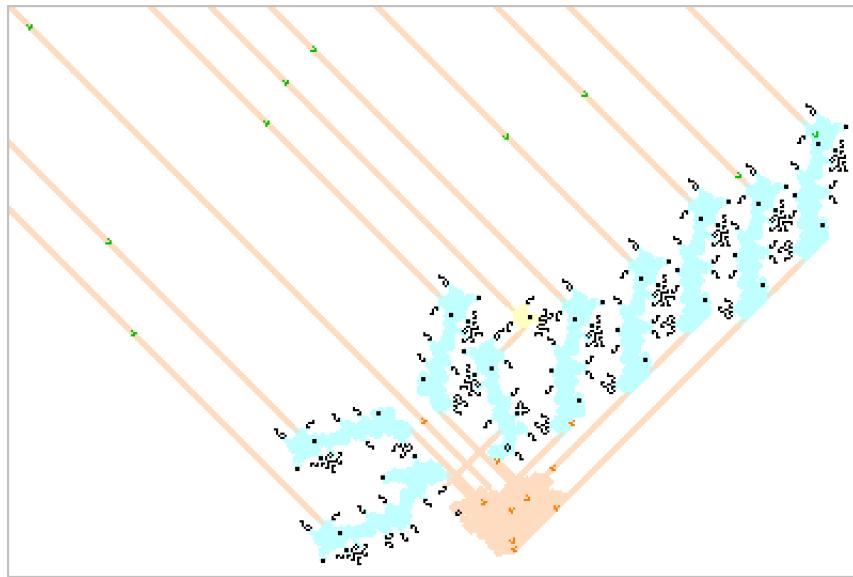


Figure 7.13: An arrangement of Herschel edge shooters (highlighted in aqua) that places 10 gliders coming from the northwest (displayed in green) into the correct positions (displayed in orange) 738 generations later to synthesize a 2-engine Cordership.

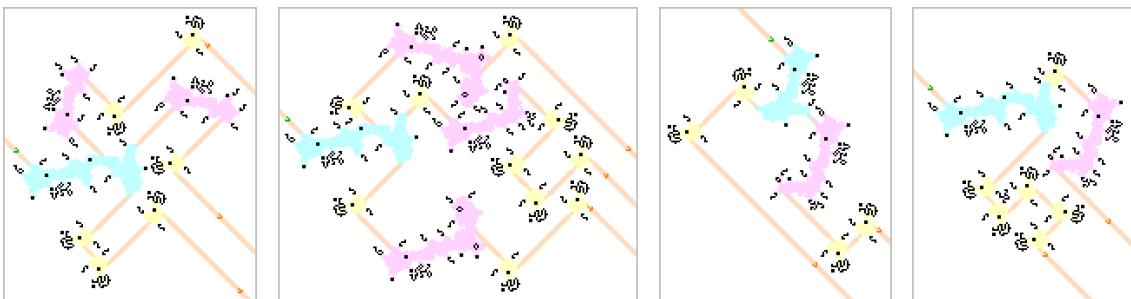


Figure 7.14: Four circuits that, from left to right, synchronize the 3 leftmost input gliders in Figure 7.13, the next three gliders, the next two gliders, and the final two gliders, respectively. All four of these circuits can be constructed using the same methods that we used to build the glider-to-LWSS circuit in Figure 7.11. 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.

timing of our choosing and then only use Snarks for all reflections. We present one such toolkit in Appendix ??.

7.5 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).¹⁶ For example, a period 93 gun that works by using 16 Herschels on the track that we originally constructed in Figure ??, and deleting one of the eater 2s, is displayed in Figure 7.16.¹⁷

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 80.¹⁸ The pattern displayed in Figure 7.17 is made up of

¹⁶Even 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.

¹⁷Unfortunately, this particular track cannot be used to make a period 62 gun, even though placing 24 Herschels on the track makes a period $1488/24 = 62$ oscillator. See Exercise 7.27 for an explanation of the problem and a method of fixing it.

¹⁸Actually, this same method works for periods 74, 75, 78, and 79 too—see Exercise 7.28.

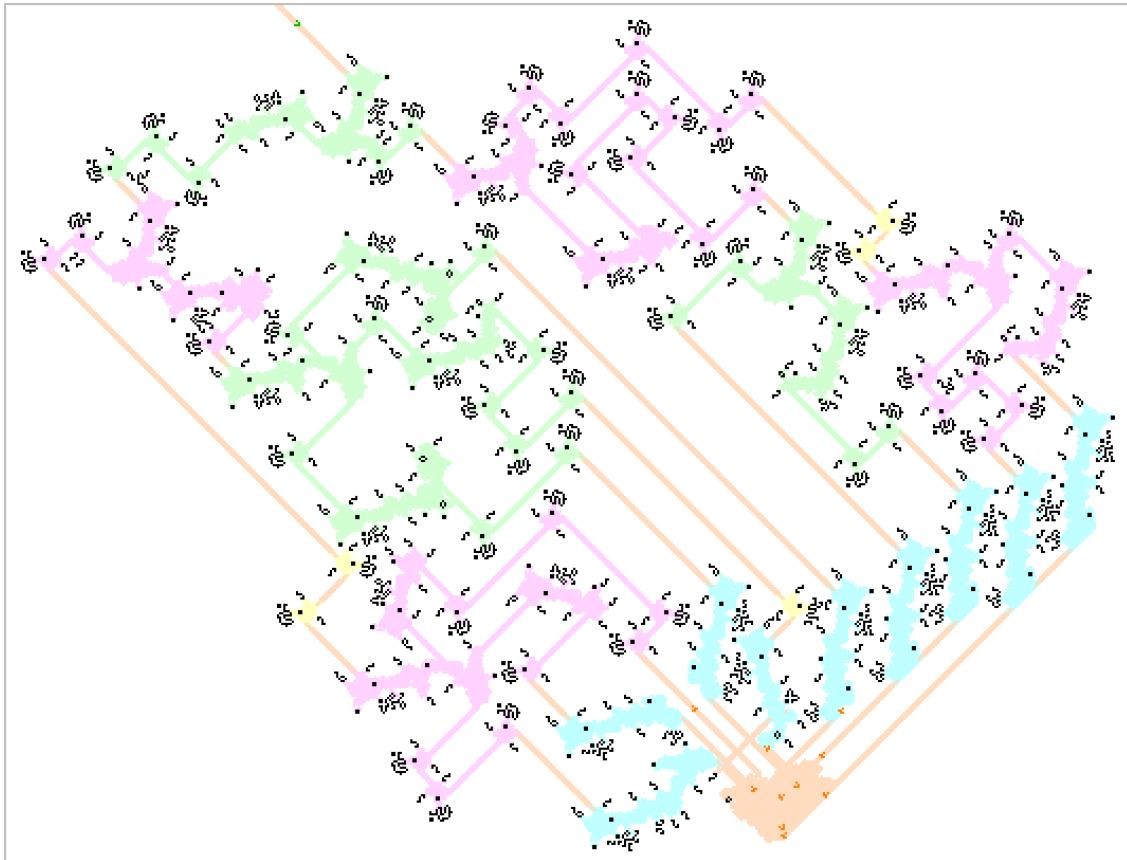


Figure 7.15: A stable circuit that converts a single glider (displayed in green at the northwest) into a 2-engine Cordership (which is synthesized by the 10 gliders displayed in orange at the southeast). Those 10 gliders are synchronized, 2 or 3 gliders at a time, by the various mechanisms highlighted in green and magenta (the four southernmost of which were displayed in Figure 7.14) and fed into the arrangement of Herschel edge shooters (highlighted in aqua) that we originally saw in Figure 7.13.

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.¹⁹ The 8 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 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 as their period does. In particular, a period p gun of this type 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. Thus 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 $(C\sqrt{\log_2(p)}) \times (C\sqrt{\log_2(p)})$ for all large p .²⁰ The key objects used in the construction of such guns (and many of the other interesting stable

¹⁹This adjustable glider gun was constructed by Matthias Merzenich in September 2015.

²⁰If you are familiar with big-O notation, this can be rephrased as saying that we can construct period p glider guns with bounding box of length and width $O(\sqrt{\log(p)})$. We will introduce big-O notation and explore related concepts in Chapter ??.

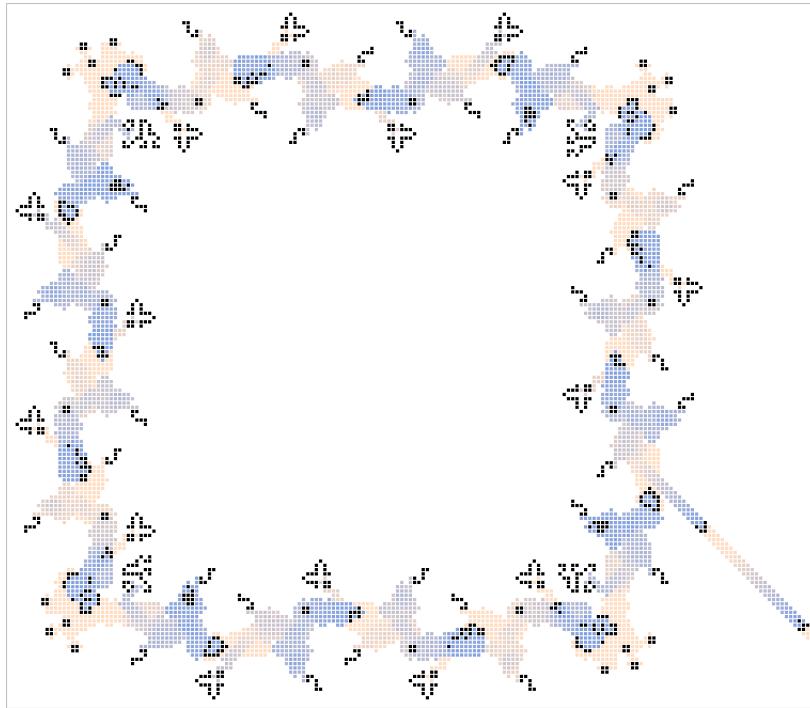


Figure 7.16: A period 93 glider gun consisting of the R64 and Fx77 conduits introduced in Section 3.6. This gun has 16 Herschels on the track instead of 6 as in the original track from Figure ??, making its period $1488/16 = 93$ instead of $1488/6 = 248$. We also deleted one of the eater 2s at the southeast corner to allow a single stream of gliders to escape, thus making it a gun instead of an oscillator.

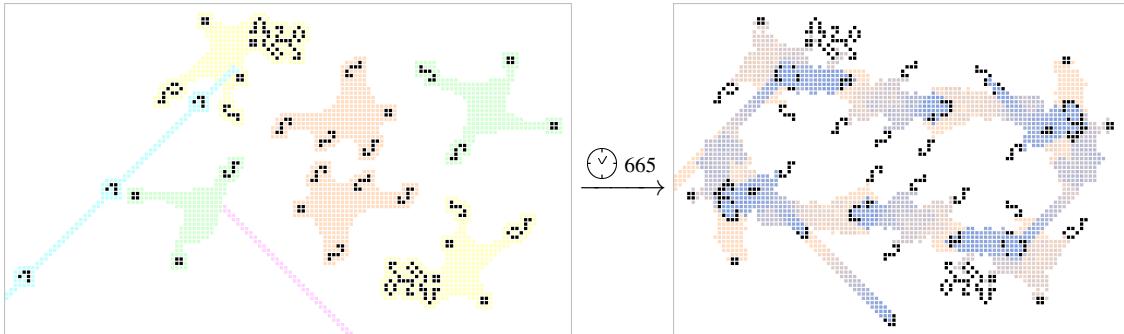


Figure 7.17: A period 80 adjustable glider gun. Eight gliders come in from the southwest (highlighted in aqua) to start the gun. Those gliders are fed into a syringe (highlighted in yellow) and then proceed clockwise through an F117 conduit (highlighted in orange) and the herschel-to-glider converter from Figure 7.5(b) (highlighted in green). Then the glider 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 magenta). To increase the period of this gun by n generations, simply move the top half of the gun northeast by n cells and separate the eight input gliders by n more generations.

constructions we will make from now on) 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.²¹

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 $(2 \times 536) = 1072$ gun. Placing a second semi-Snark in the path of the output gliders then doubles its period again, resulting in the period $(4 \times 536) = 2144$ gun in Figure 7.19(a). We can of course place many more semi-Snarks along the output path of

²¹The color-changing semi-Snark was the first known “small and simple” stable glider period-multiplier, and it was found by Sergey Petrov in July 2013. Its color-preserving variant was found by Tanner Jacobi in October 2017.

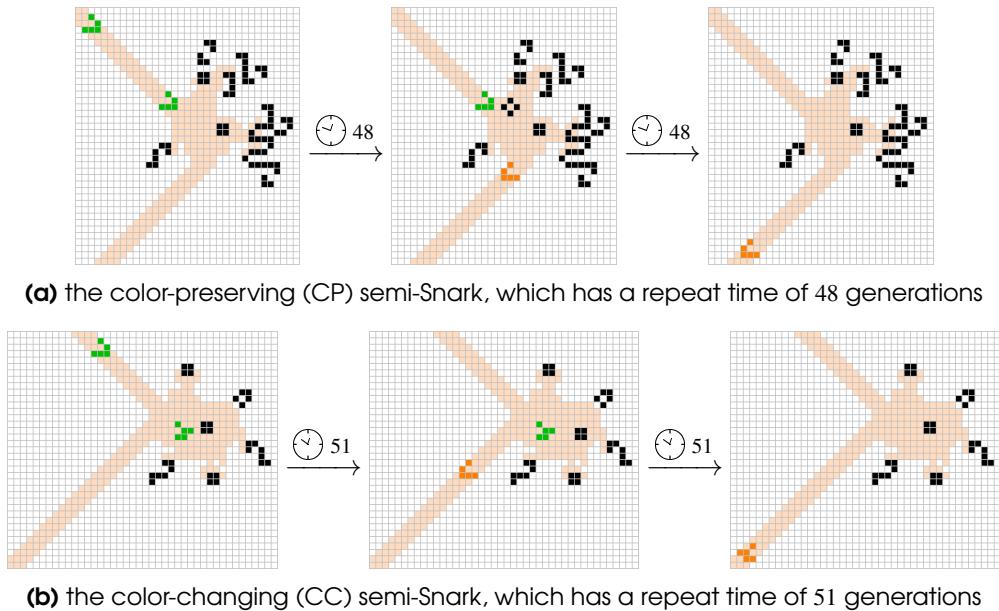


Figure 7.18: The semi-Snarks are stable conduits that produce one reflected glider for every two input gliders. The color-preserving version (a) is the same as a Snark (compare it with Figure 3.29(b)), but with the top eater replaced by a custom conduit that, when hit by a glider, creates a tub that destroys the next glider. In the color-changing variant (b), the first glider is reflected and moves the central block, and the second glider simply moves the block back using the reaction that we saw back in Figure 5.21(a).

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 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 box of size roughly $(30\sqrt{\log_2 p}) \times (30\sqrt{\log_2 p}) = (30\sqrt{n}) \times (30\sqrt{n})$.²² 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 contain a large power of 2 in their prime decomposition. For example, this method cannot help us construct a small gun with period equal to a prime number like 3413277319.

In order 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 2.²³ 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 in Figure 7.20.

However, this method of blocking a specific number of gliders only works the first time, since after the first glider that makes its way all the way through the circuit, all of the semi-Snarks will be set to “1”, rather than to their original bits. For example, after the 19th glider passes in Figure 7.20, the semi-Snarks will encode the bitstring 11111 and thus will 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

²²We 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.

²³There are also small *tremi*-, *quadri*-, and *quiti*-Snarks that multiply the period of a glider stream by 3, 4, and 5, respectively—see Exercises 7.13, 7.14, and 7.15.

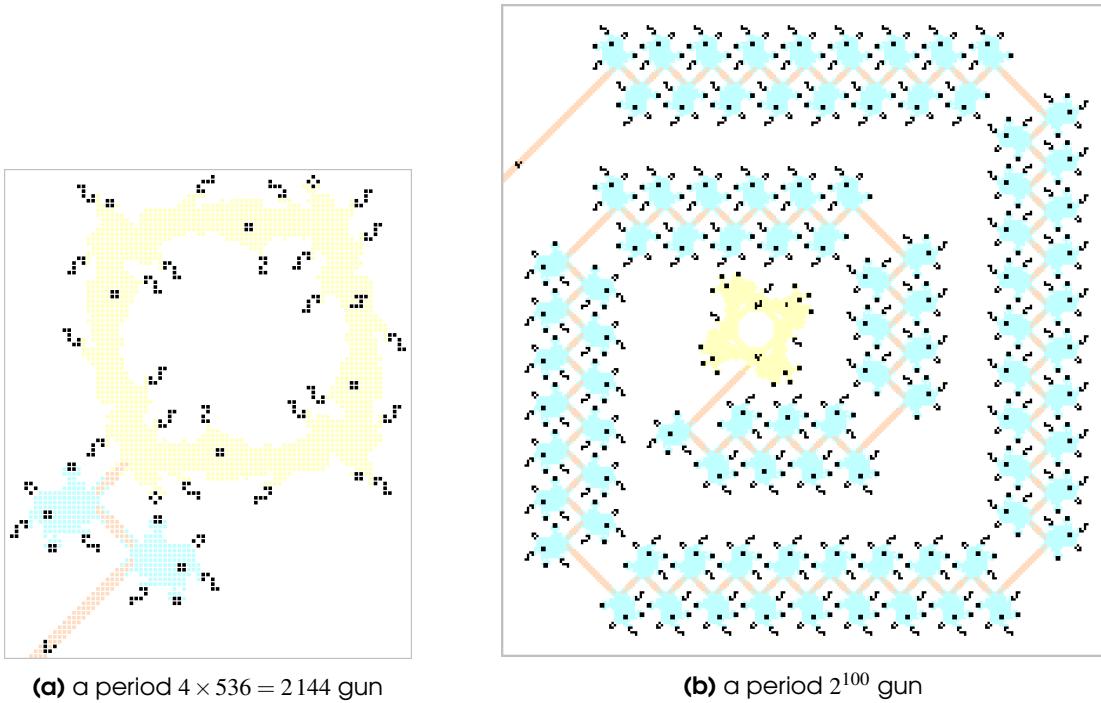


Figure 7.19: Two guns that use semi-Snarks to increase their periods. The gun in (a) uses two semi-Snarks (highlighted in aqua) to multiply the period of a p536 gun (highlighted in yellow) by 4. In the glider gun in (b), the central gun (which consists of four copies of the R64 conduit and is called the *machine gun*) has period 256 and funnels gliders into a spiral of 92 semi-Snarks, resulting in a gun of period $256 \times 2^{92} = 2^8 \times 2^{92} = 2^{100}$.

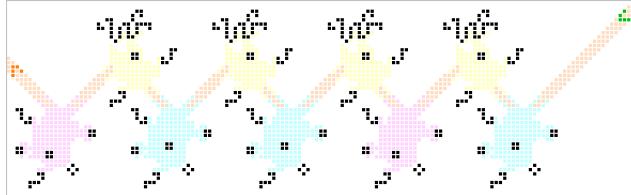


Figure 7.20: If we interpret a semi-Snark that blocks a glider (highlighted in magenta) as a “1” and a semi-Snark that lets a glider pass (highlighted in aqua) as a “0”, then this arrangement of semi-Snarks corresponds to the binary representation 10010 of the number 18. The input glider toggles the 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.

back to “0” before the next input glider hits the circuit.²⁴ One method of implementing this toggle is presented in Figure 7.21.

We thus now know how to multiply the period of any (sufficiently high-period) 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).

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, simply fire a glider at either the east or west side so as to destroy the glider(s) that turned it on.

²⁴Actually, for our purposes it’s even okay if we even toggle those semi-Snarks *after* the next input glider hits the circuit, as long as the reactions do not interfere with each other.

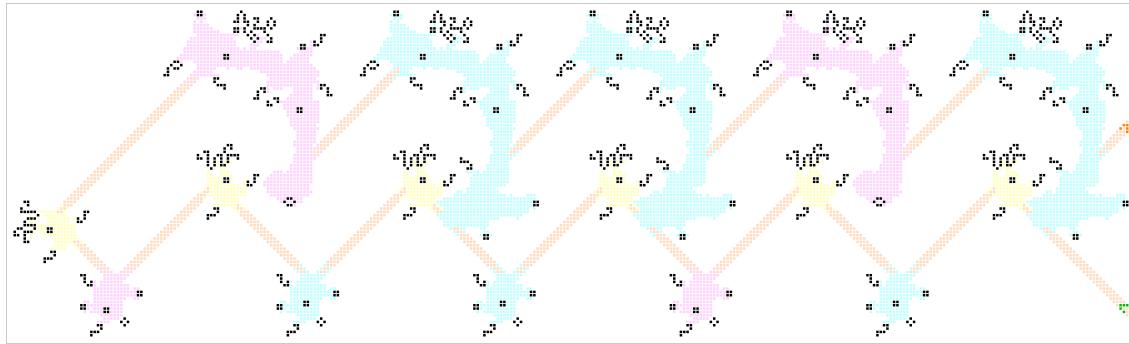


Figure 7.21: A stable circuit that blocks 18 out of every 19 input gliders (displayed in green at the southeast). As in Figure 7.20, the arrangement of semi-Snarke components corresponds to the binary representation 10010 of the number 18, and the components at the top half of the circuit use the output glider to reset the semi-Snarke to their original configuration. In particular, the cyan circuits at the top inject another glider into the bottom half of the circuit that toggles a semi-Snarke back into the “0” position, whereas the magenta circuits at the top do not (and thus leave the semi-Snarke in the “1” position).

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 (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 1087 generations to its period (for a total period of $616 + 1087 = 1703$) by attaching the mechanism displayed in Figure 7.22 to it. By adjusting the spacing and components in the trombone slide of this mechanism, any number of generations at least 1087 can be added to the period of the gun.

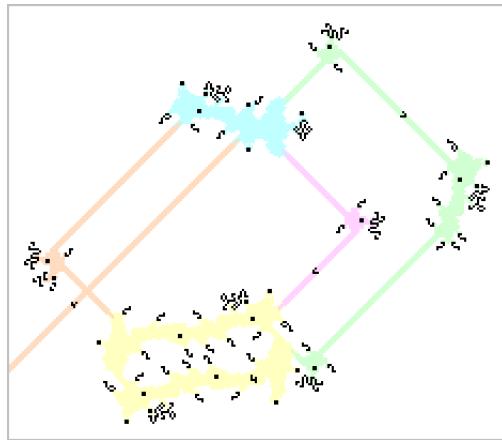


Figure 7.22: A period $616 + 1087 = 1703$ gun that works by feeding the output of a period 616 gun (highlighted in yellow) into a glider tripler (highlighted in aqua). One of the outputs of this tripler escapes to the southwest as the output of the gun, one goes southeast (highlighted in magenta) to 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 reflectors that it uses, we can delay the glider that restarts the p616 gun by any number of generations larger than 471 as well.

By combining all of these techniques, we can now construct small²⁵ guns of *any* (sufficiently large) period. For example, to construct a small gun with large prime period 3413277319, we first compute $\lfloor (3413277319 - 1087)/616 \rfloor = 5541032$, so we want to multiply the p616 gun’s period by this amount. Since 5541032 has binary representation 10101001000110010101000, we arrange

²⁵Just like earlier, by “small” we mean that we can construct guns of period p with bounding box whose length and widths are both $O(\sqrt{\log(p)})$.

semi-Snarks (via the method of Figure 7.21) according to this bitstring, except we wrap them in a spiral (like we did in Figure 7.19(b)) instead of placing them in a straight line so as to save some space. Finally, since $(3413277319 - 1087) \equiv 520 \pmod{616}$, we use the mechanism from Figure 7.22 with an additional 520-generation delay to add $1087 + 520 = 1607$ generations to its period. Then the completed gun has period $(616 \times 5541032) + 1607 = 3413277319$, as desired, and is displayed in Figure 7.23.²⁶

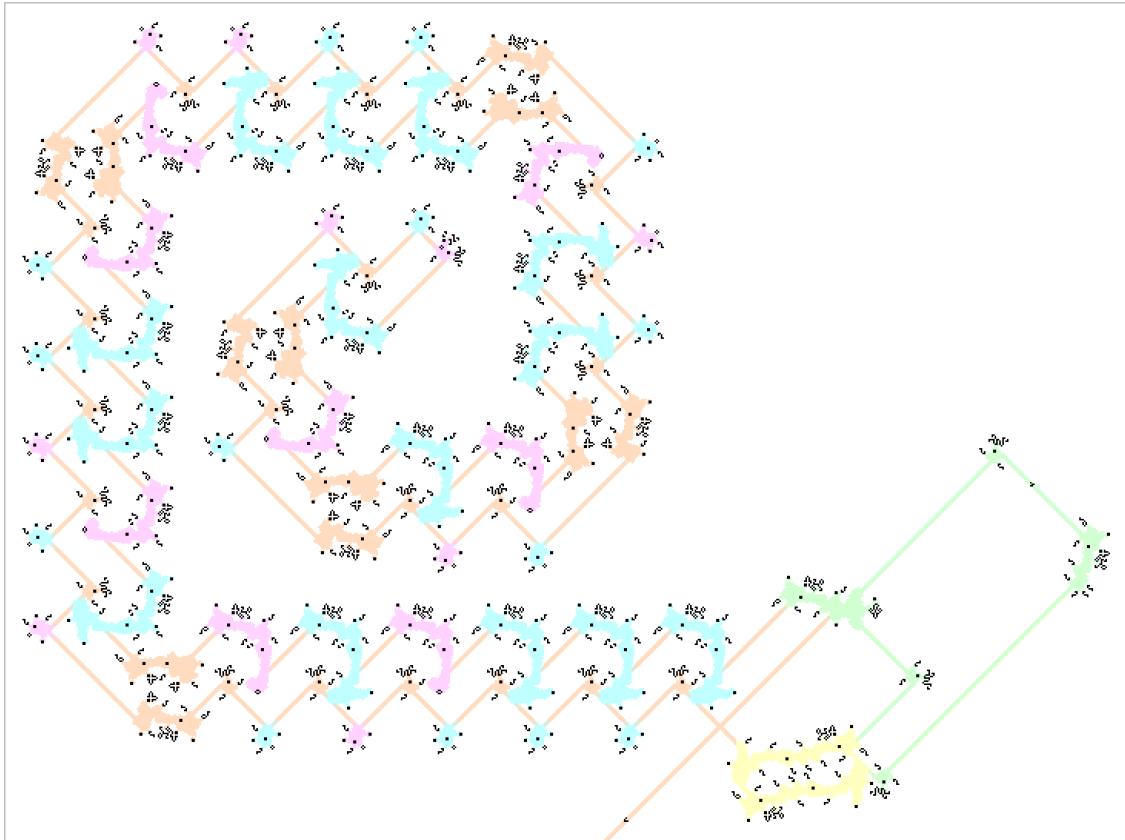


Figure 7.23: A period 3413277319 gun that uses two separate mechanisms to modify the period of a p616 gun (highlighted in yellow). First, its period is multiplied by 5541032 using the method of Figure 7.21, which has binary representation 10101001000110010101000 that is encoded by the semi-Snarks that 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 $(616 \times 5541032) + 1607 = 3413277319$.

7.6 Converters for Other Objects

To finish off this section, we note that it is sometimes also convenient to be able to work with objects other than Herschels and gliders. 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, which is then converted into 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

²⁶Most 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 these guns for any period at least 1703. This script can be found at conwaylife.com/forums/viewtopic.php?p=54265#p54265.

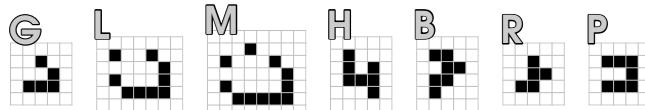


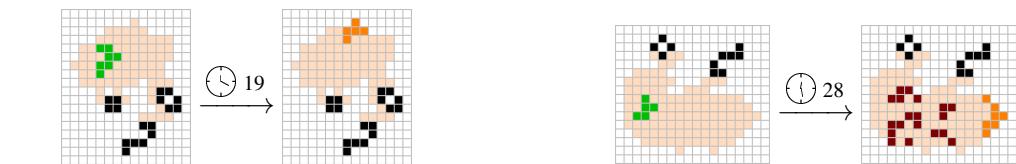
Figure 7.24: For brevity, when manipulating small common objects as signals, we typically use the single-letter abbreviations G (glider), L (lightweight spaceship), M (middleweight spaceship), H (Herschel), B (B-heptomino), R (R-pentomino), and P (pi-heptomino). It is sometimes useful (for example, when naming conduits that convert between these types of objects) to be able to refer to a canonical phase and orientation of these objects, and those are what are displayed here.

the form “X-to-Y”, where X and Y are the abbreviations described by Figure 7.24. For example, a 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.²⁷

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,²⁸ and the output heptomino is in its canonical orientation (i.e., it has `<orientation>` prefix “F”) and is thus named RF28B.²⁹



(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 additional symmetry so that it does not make sense to say whether it is being turned left or right, people sometimes use `<orientation>` code “T” (for turn) instead of “R” or “L”. 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, so it is sometimes called PT8P (rather than

²⁷An 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).

²⁸Actually, this conduit does not produce the B-heptomino itself, but rather the *B-heptaplet* that evolves in the same way.

²⁹The RF28B conduit appears in some of the Herschel conduits that we saw back in Table 7.1: BX202, L156, and Rx164 (see Exercise 7.24).

PR8P or PL8P). However, we typically just pick whatever orientation of the conduit we need and refer to the name that makes sense given that orientation.

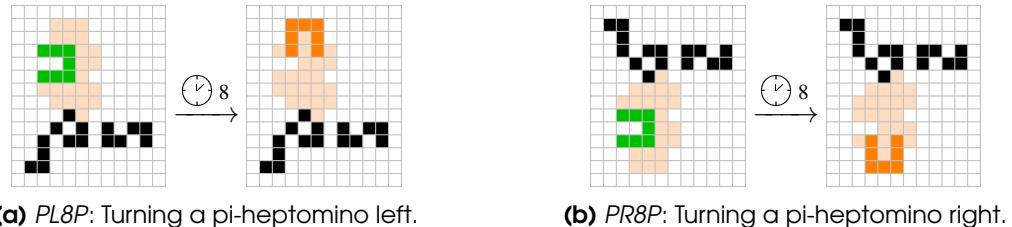


Figure 7.26: PT8P: A conduit that takes 8 generations to turn a pi-heptomino by 90 degrees.

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.³⁰ In a sense, that oscillator (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 7.20.

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 <direction><lane>T<timing>, just like in Section 7.2. Just as was the case back then, we do not dwell on the details of how the <lane> or <timing> are computed, but instead 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 in 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.

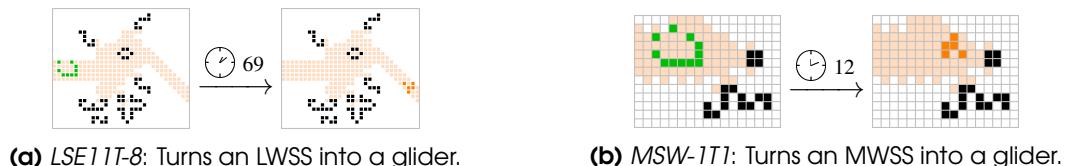


Figure 7.27: Some conduits that turn spaceships into gliders. The (a) LWSS-to-G can also act as an MWSS-to-G (see Exercise 7.18) and was found by Ivan Fomichev in October 2015, and the (b) MWSS-to-G was found by Matthias Merzenich in July 2013.

A selection of conduits that convert between these basics types of signals 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.³¹ We also clarify that a 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 7.25):

$$\text{glider} \xrightarrow{\text{syringe}} \text{Herschel} \xrightarrow{\text{HF95P}} \text{pi-heptomino}.$$

³⁰In fact, the same conduit and pi-heptomino rotation appears in Tanner’s p46.

³¹A much larger and more complete catalog of conduits and converters is available online at conwaylife.com/forums/viewtopic.php?f=2&t=1849&p=23375

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 objects into other types of objects. Most of these conduits are quite old and well-known.

7.7 Factories

Somewhat surprisingly, it is not only useful to convert moving objects like Herschels or gliders into other moving objects, 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, a factory will self-destruct if we do not make use of the still life or oscillator 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).³² 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 self-destruct.

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.³³ The reason that these conduits are useful is that they 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, which blocks the next signal

³²For an almost-oscillator that similarly acts as a block factory, see Exercise 7.29.

³³The 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 glider outputs when making its beehive.

(i.e., glider) that comes in from the northeast.

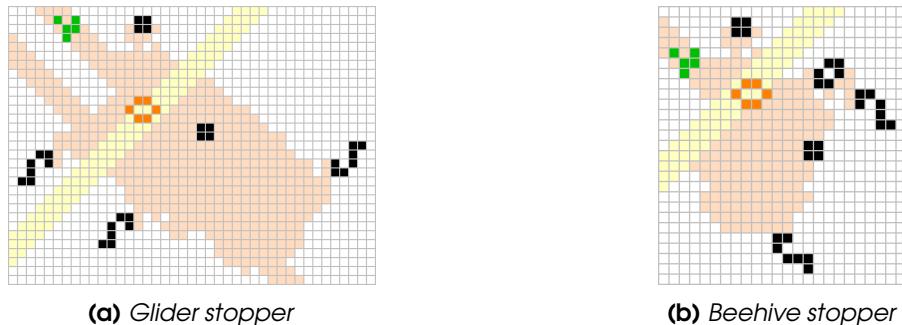


Figure 7.28: Some factories that use a glider (displayed in green) to create a beehive (displayed in orange). If the beehive is present, it destroys (and is destroyed by) a single glider coming from the northeast on the lane highlighted in yellow. Otherwise, that glider passes through the factory unharmed. 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, as is the Herschel-to-loaf conduit displayed in Figure 7.29(b).

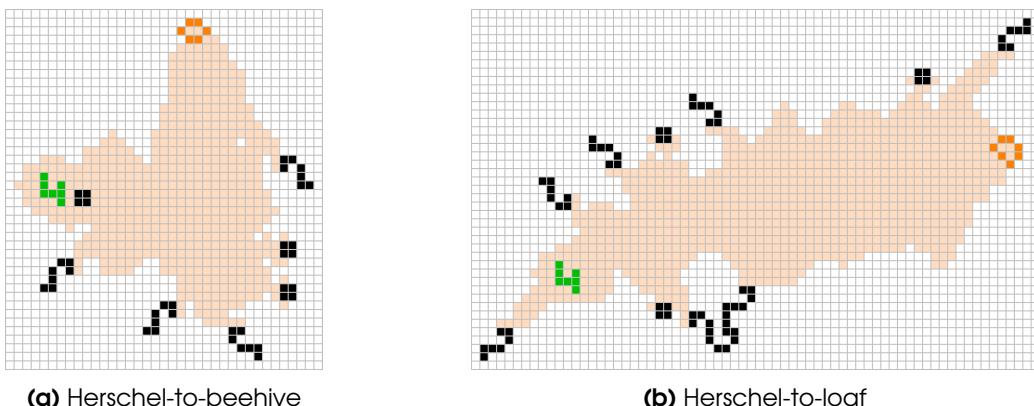


Figure 7.29: Some edgy conduits that can convert a Herschel (displayed in green) into a small still life (displayed in orange). The (b) loaf factory was found by Adam P. Goucher in 2009.

Since a boat can be used as a one-time turner (refer back to Section 5.7.2 and Exercise 5.29), 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. The Herschel-to-boat factory of Figure 7.30(b) is called the *demultiplexer*,³⁴ and it can similarly reflect a glider heading southwest (despite being much less edgy than the other two boat factories). However, it also has the useful property that if the boat is *not* present then that glider simply passes through the factory unharmed (whereas such the glider would destroy either of the factories in Figure 7.30(a) if the boat were not present).

For this reason, the demultiplexer can act as a storage device for a single bit of memory—we can think of the absence or presence of the boat as a “0” or a “1”, respectively. A Herschel going into this factory then corresponds to storing the bit “1” in its memory, and a glider going into this factory has the effect of reading the contents of its memory (and resetting it to “0”).

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*³⁵) in generation 37. For this reason, some Herschel-to-block factories (like the two central ones

³⁴Found by Brice Due in August 2006.

³⁵In analogy with the first natural glider that they make in generation 21.

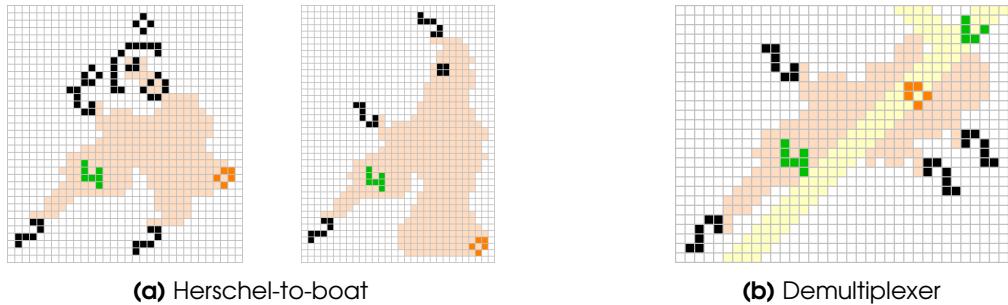


Figure 7.30: Some edgy Herschel-to-boat factories that can be used to reflect gliders. In (b), the input glider follows one of the two paths highlighted in yellow: it is reflected if the boat is present, and it passes straight through the factory otherwise.

in Figure 7.31) are particularly simple and work just by cleaning up the input Herschel’s debris after it makes that block. The leftmost factory in Figure 7.31 is somewhat larger and more complicated than the other block factories, but is particularly useful for how far away it produces its block, as well as the fact that it can act as a period tripler (see Exercise 7.30).

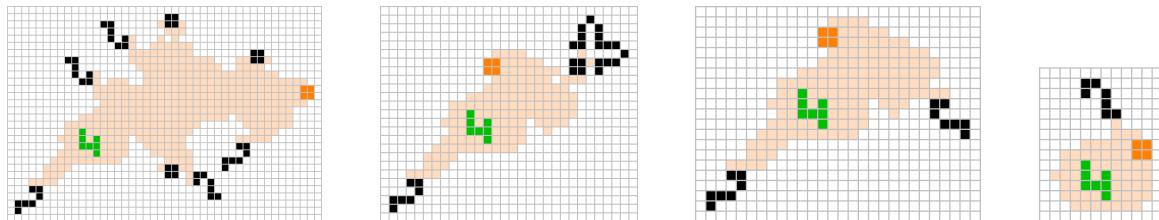


Figure 7.31: Some Herschel-to-block factories.

7.7.1 Highway Robbers

While we have seen numerous glider reflectors so far—the Snark, the Silver reflector, bumpers, bouncers, and Herschel-track-based reflectors from earlier in this chapter—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*.³⁶

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 are destroyed, producing some chaotic debris or a perpendicular glider, if they are hit at their edge by a passing 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.³⁷ 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.

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

³⁶The name refers to the fact that highway robbers “steal” gliders from their lane.

³⁷If hit on its edge by a glider, a (1) block quickly moves as in 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 honeyfarm, or makes debris that is difficult to clean up since it is on the opposite side of the glider.

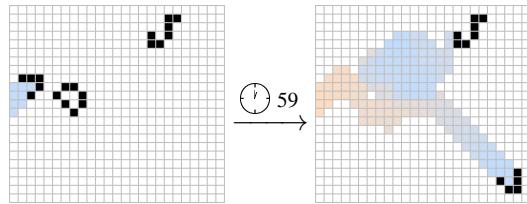


Figure 7.32: When a glider just barely hits a loaf, a perpendicular glider and some debris are created. An eater 1 can be used to absorb the extra debris. Gliders on any further away lanes simply pass by the loaf unharmed.

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 (see Exercise 7.32), it is essentially the fastest one known (some slightly faster highway robbers have been built, but at the expense of being much larger).

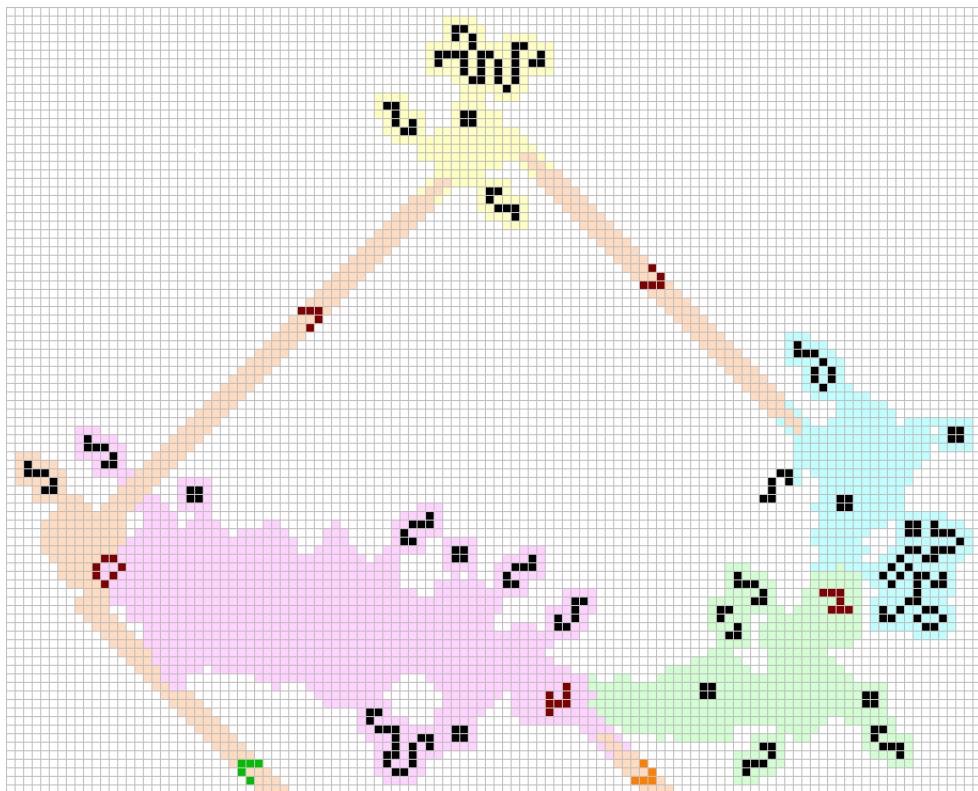


Figure 7.33: A highway robber that was constructed by Chris Cain in March 2015. It uses the glider-loaf collision of Figure 7.32 to create a perpendicular glider that is fed into a Snark (highlighted in yellow) and then a syringe (highlighted in aqua), an L112 conduit (highlighted in green), and finally a Herschel-to-loaf factory (highlighted in magenta) so as to recreate the loaf.

Notes and Historical Remarks

While stable circuitry for carrying out the tasks described in this chapter has existed for a couple 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 standard way to accomplish the reverse task in previous years was to use a *Herschel transceiver*—a device that sends and receives a pair of

gliders traveling the same direction, but potentially on different lanes.³⁸ The first such device to be discovered³⁹ is displayed in Figure 7.34.

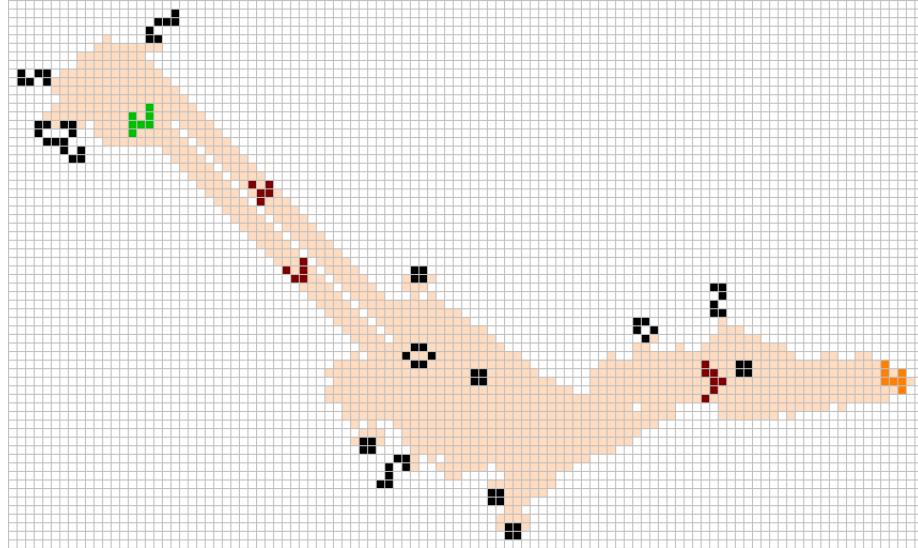


Figure 7.34: A Herschel transceiver that is made up of two components that can be separated by an arbitrary amount—a conduit that converts a Herschel into two gliders at the top-left, and a conduit that converts those two gliders back into a Herschel (via a B-heptomino) at the bottom-right. This device appeared in lots of pre-2015 circuitry, but was made almost completely obsolete by the discovery of the syringe.

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 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,⁴⁰ but they were relatively awkward, needing sparks from oscillators or other extra cleanup, so they were seldom or never used in larger patterns. One other 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 7.26.

The original Herschel transceiver was featured quite often in pre-2015 circuitry, because it provided a relatively inexpensive way of “transmitting” a Herschel signal from one place to another in the form of gliders. It was easy to get a Herschel to generate a *single* glider, but to then get a Herschel back out of that single glider required a large composite circuit, the fastest of which had a repeat time of 497 generations. By contrast, a Herschel transceiver can send the same signal via two gliders with a repeat time of just 117 generations, and its receiver is much smaller as well.

With the appearance of the syringe in 2015 (repeat time 78, or 74 with overclocking), 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 two tandem gliders naturally produce some other reaction of interest, it is still sometimes easier to catch them with a Herschel receiver—especially when their lanes are only 2 cells apart. One such reaction that we will make use of later is the one displayed in Figure 7.35 that uses 3 gliders to push a block and send 2 tandem gliders back.

Another Herschel transceiver that is sometimes useful is the one displayed in Figure 7.36 that works via gliders that are separated by 4 lanes instead of 2, 5, or 6.⁴¹ This transceiver has two

³⁸Such gliders are said to be in *tandem*.

³⁹This Herschel transceiver was found by Paul Callahan—the receiver in October 1996 and the transmitter in May 1997.

⁴⁰By Paul Callahan and by Dieter Leithner

⁴¹This transceiver was found by Sergei Petrov in December 2011.

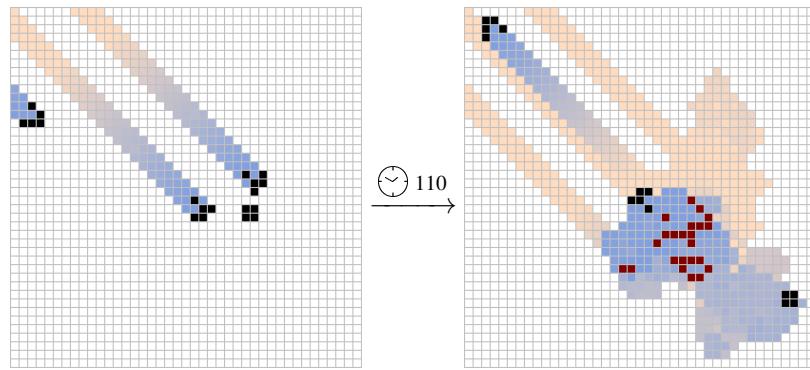


Figure 7.35: A reaction in which 3 gliders push a block southeast by 10 cells and send 2 tandem gliders back, separated by just 2 lanes and thus suitable for input into a Herschel receiver. The debris highlighted in red dies off in 25 more generations.

advantages over the one that we saw earlier: it produces a quick 90-degree glider output in addition to 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, using only Spartan catalysts.

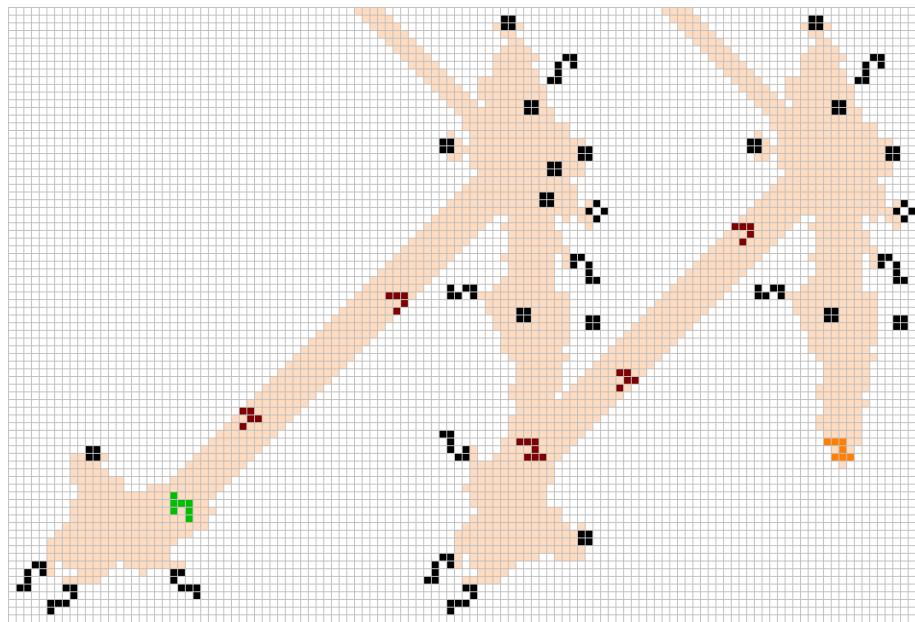


Figure 7.36: A Herschel transceiver that works via tandem glider pairs that are separated by 4 lanes. The receiver that it uses is ambidextrous—it can be configured to work regardless of which of the two input gliders arrives first, as demonstrated here by the middle and right receivers here taking different configurations of input gliders.

Alternatively, prior to the discovery of the syringe, there were some stable conduits known for converting some spaceships *other* than gliders into Herschels. For example, the conduit displayed in Figure 7.37 converts a Coe ship into a Herschel.⁴² However, the reverse transformation of a Herschel into the original spaceship was much less straightforward, and would be performed by converting the Herschel into multiple gliders that we use to synthesize the original spaceship (a Coe ship in this case) via a glider synthesis like the one that we saw in Figure 5.11.

Exercises

solutions on page 310

⁴²This conduit was found by Stephen Silver in September 1997.

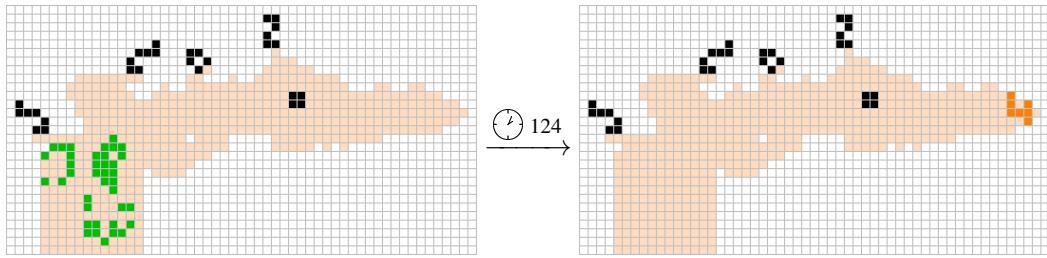
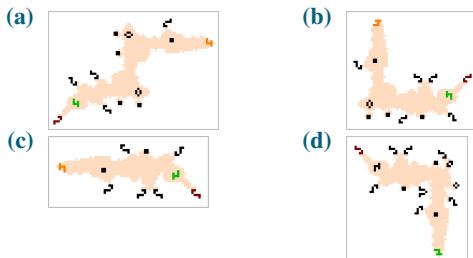


Figure 7.37: A stable conduit that converts a Coe ship into a Herschel.

7.1 Give the name (using the naming scheme of Table 7.1) for each of the following Herschel conduits, as well as their repeat time.



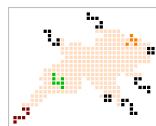
7.2 Construct a stable pattern that eats a Herschel.

[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 Modify the conduit from Figure 7.6 so that it has two side-by-side transparent lanes instead of just one.

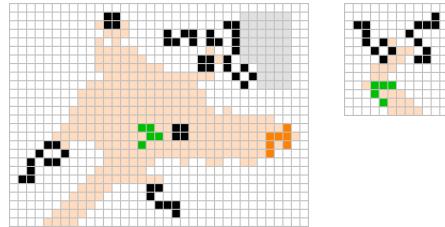
[Hint: There are other ways to eat a glider.]

7.4 A conduit that is very similar to the HNE5T-4 conduit from Figure 7.6 is presented below. Explain why this conduit might be more useful than HNE5T-4 in some situations.



7.5 The syringe works by converting the input glider into another 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?

7.6 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 below on the right.]



7.7 Construct a glider synthesis of the large version of the syringe displayed in Figure 7.8(b).

7.8 Make a stable conduit that converts one glider into exactly...

- (a) 4 gliders.
- (b) 5 gliders.
- (c) 10 gliders.

7.9 One of the still lifes near the eastern edge of the glider-to-LWSS circuit in Figure 7.11(b) is displayed in red. Why is it singled out like this—what makes it different from the other still lifes in that circuit?

7.10 Use the glider syntheses provided below (or any other glider syntheses of your choosing) to construct a stable circuit that converts a glider into the following objects.

- (a) A middleweight spaceship.
- (b) A heavyweight spaceship.
- (c) A Schick engine.

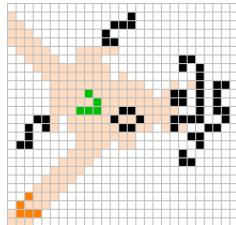
7.11 The 2-direction glider synthesis of the 2-engine Cordership from Exercise 5.14(b) consists of 11 gliders, but when we used it to start our construction of the glider-to-(2-engine Cordership) circuit in Figure 7.13, we only needed 10 input gliders. Explain this discrepancy—what happened to the 11th glider?

7.12 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.

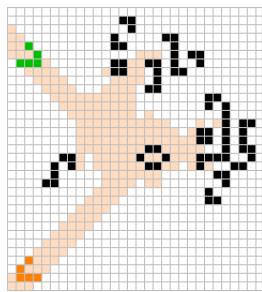
- (b) 6 gliders.
 (c) 47 gliders.

7.13 The following pattern is called a *tremi-Snark*,⁴³ since it only produces a single output glider for every three input gliders that it receives.



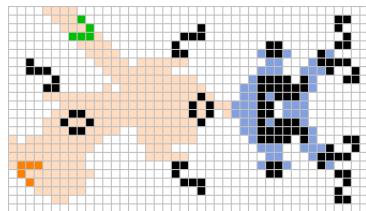
- (a) What is the repeat time of the tremi-Snark?
 (b) Is it color-changing or color-preserving?
 (c) Use tremi-Snarfs to make a glider gun with period 3^{50} .

7.14 The following pattern is called a *quadri-Snark*,⁴⁴ since it only produces a single output glider for every four input gliders that it receives.



- (a) What is its repeat time?
 (b) The quadri-Snark is color-preserving. Construct a similar stable circuit for reflecting one out of every four input gliders, but is color-changing.
 (c) Use quadri-Snarfs to make a glider gun with period 2^{100} that fits in a smaller bounding box than the one from Figure 7.19(b).

7.15 The following pattern is called a *quinti-Snark*,⁴⁵ since it only produces a single output glider for every five input gliders that it receives.



- (a) Is it color-changing or color-preserving?

- (b) What is its repeat time?
 (c) 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).

7.16 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 $616 + 471 = 1087$?

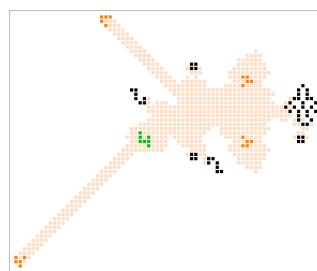
7.17 We claimed that the method of Figure 7.22 can be used to add 1087 or more generations to the period of the *p616 adjustable glider gun*. Explain why the cut-off is 1087 generations.
 [Hint: Try to add 1086 generations to the period of the *p616 gun*—what goes wrong?]

7.18 Show how the LWSS-to-glider converter from Table 7.3 can also function as an MWSS-to-glider converter.

7.19 Use two copies of the B60 Herschel conduit to make a glider gun (the resulting gun is called the *Simkin glider gun*).⁴⁶ What is its period, and how could you determine that period based only on the name of this conduit?

7.20 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?

7.21 The conduit displayed below converts a Herschel into 4 gliders, all traveling in different directions. Break this conduit up 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?]

⁴³Found by Tanner Jacobi in September 2017.

⁴⁴Found by Tanner Jacobi in October 2017.

⁴⁵Found by Tanner Jacobi in October 2018. Some other periodic quinti-Snarfs are also known, but this is the most useful one for the reason explained in part (c) of this exercise.

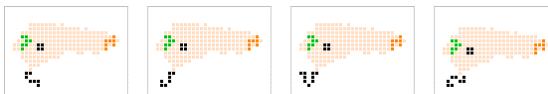
⁴⁶It is named for discoverer, Michael Simkin, who found it in April 2015.

7.22 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.

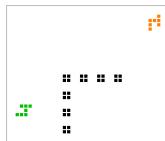


- (a) Explain why the original HFx58B conduit cannot be used as an input to BR146H.
- (b) Attach the variant of HFx58B to both the input and output ends of BR146H. What is the name of this composite conduit?

7.23 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 7.22 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.

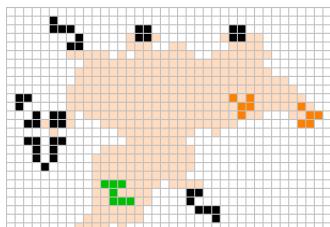


7.24 The L156 conduit from Figure 7.1(b) is made up of 3 conduits that convert the input Herschel into other well-known named objects before converting it back into a Herschel. What are the intermediate objects that the Herschel is converted into, and what are the names of those 3 conduits?

7.25 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.

7.26 An H-to-G6 conduit other than the one at the top-left corner of Figure 7.34 is presented below.⁴⁷



⁴⁷This conduit was found by Matthias Merzenich in December 2011.

⁴⁸Found by Robert Wainwright in 1990 and Achim Flammenkamp in 1994, respectively.

⁴⁹First constructed by Bill Gosper in July 1994.

⁵⁰First constructed by Chris Cain in 2016.

Use two copies of this conduit, together with two copies of the Herschel receiver from the bottom-right corner of Figure 7.34, 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.

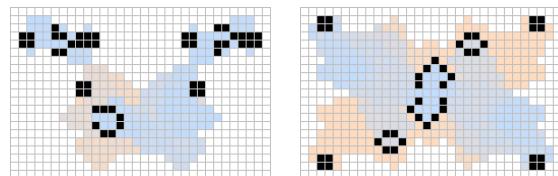
7.27 Recall that the period 93 glider gun in Figure 7.16 has 16 Herschels on a track that each take $16 \times 93 = 1488$ generations to traverse it.

- (a) Adjust the number of Herschels on the track so as to create a period 124 glider gun.
- (b) Explain why this does not become a period 62 glider gun if we place $1488/62 = 24$ Herschels on the track.
- (c) Create a true-period 62 glider gun by making a Herschel loop that uses the L156 circuit for the corners instead of R64 conduits (still use Fx77 conduits for the sides).

7.28 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 the two independent reasons why it does not work.

7.29 The sparky oscillators below are called *Wainwright's p72* (or sometimes *two blockers hassling R-pentomino*) and *Achim's p144*.⁴⁸



- (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).⁴⁹
- (b) Construct a slightly smaller p144 glider gun by using Rich's p16 instead of two blockers hassling R-pentomino.⁵⁰

7.30 This exercise concerns the leftmost Herschel-to-block factory from Figure 7.31.

- (a) What happens if you send a second Herschel into this conduit without cleaning up the block that the first Herschel made?
- (b) What happens if you then send a third Herschel into this conduit?
- (c) Use a syringe and this conduit to turn a Gosper glider gun into a period 90 gun.

7.31 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)?

7.32 Calculate the repeat time of the highway robber from Figure 7.33.



8. Guns and Glider Streams

Assembling large arrays of guns and puffers—other than getting the timing and positioning right—is about 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 use collisions between these gliders to create other objects, and we often have to be careful to have appropriate spacing between the gliders. However, thus 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 61 and higher than we can construct via the (typically very large) 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 any stream of period at least 29, thus doubling its period (the semi-Snark of Section 7.5 can similarly delete every second glider in a glider stream, but it only works at periods 48 and greater).¹ 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. The particular orientation of the glider streams that works is displayed in Figure 8.1, and we see that one of the glider streams is erased completely (half of its gliders are destroyed in the glider-to-block collision with the other stream, and half of its gliders are destroyed by the subsequent block), and 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

¹There is a similar reaction that can be used to double glider streams of period at least 28—see Exercise 8.2.

²In particular, this glider collision is the fifth of the block-producing collisions listed in Table 5.1.

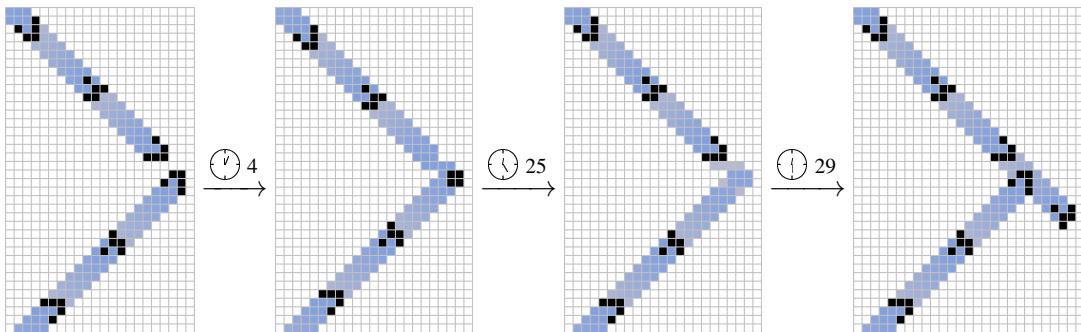
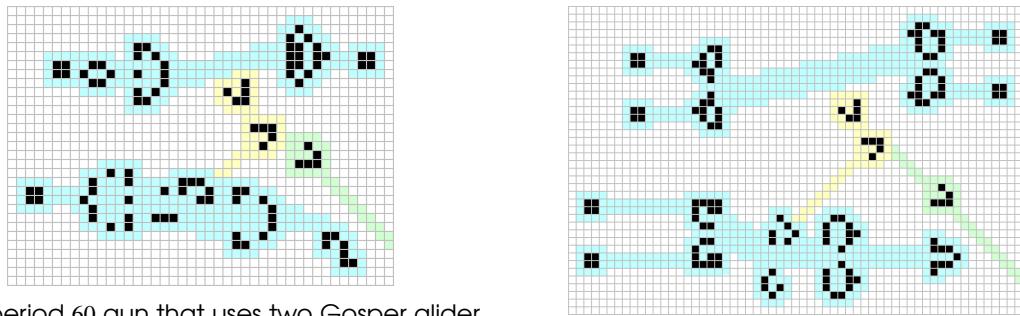


Figure 8.1: If we shoot two glider streams (of period at least 29) at each other in the proper phase, one of the streams is destroyed completely while the other stream is thinned out by a factor of 2. This works because the gliders collide to form a block, and that block then destroys the next glider in the bottom stream, letting the next glider in the top stream pass by unharmed.

twin bees gun.³ We can also iterate this procedure to create (rather large) glider guns with periods as large as we like, by doubling the spacing out the output glider stream as many times as we like.



(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 aqua, gliders that collide to create a block are outlined in yellow, and gliders to survive to create the new, thinner glider stream travelling to the southeast are outlined in green.

We can thus use this technique to multiply 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,⁴ as demonstrated in Figure 8.3.

The idea behind this mechanism is largely the same as it was in the collision in Figure 8.1, only slightly more complicated:

- When the first gliders in the streams collide, they leave behind two blocks.
- When the next gliders in the streams hit the blocks, the blocks are erased and a single blinker is left behind.
- That blinker destroys the third glider in one of the streams, and the third glider in the other stream passes by unharmed.

The effect of firing two glider streams at each other in this way is thus to completely destroy one of the streams and thin out the other stream by a factor of 3. This reaction is demonstrated in Figure 8.4,

³The gun in Figure 8.2(a) is currently the smallest known period 60 gun, but a slightly smaller period 92 gun can be constructed by colliding two twin bees shuttles together directly in a certain way.

⁴The collision we describe here is the third one listed in the “misc” section of Table 5.1.

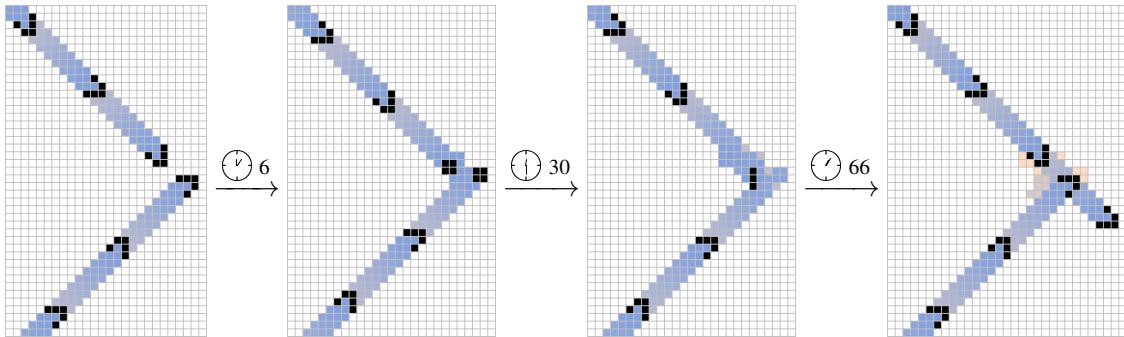


Figure 8.3: If we shoot two glider streams (of period at least 34) at each other in the proper phase, one of the streams is destroyed completely while the other stream is thinned out by a factor of 3.

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).

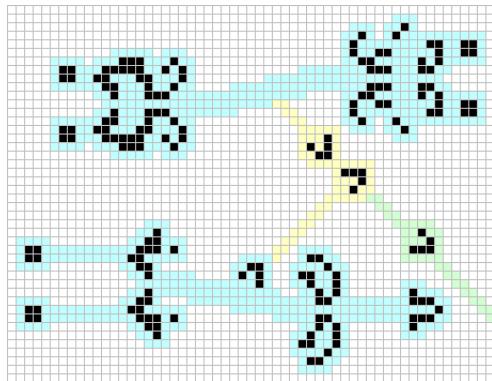
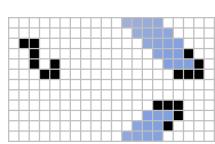
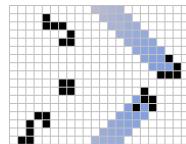


Figure 8.4: We can use two copies of any gun with period at least 34 to create a gun with triple the period. This is demonstrated here with two period 46 twin bees guns creating a period 138 gun. The individual twin bees guns are outlined in aqua, gliders that collide to create two blocks are outlined in yellow, and gliders that survive to create the new, thinner glider stream travelling to the southeast are outlined in green.

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 either 2 or 3, which lets us multiply the gun's period by any number whose prime factorization contains only those numbers. We can thus use this technique to thin out a glider stream by a factor of 4, 6, 8, or 9, for example (but recall from Section 7.5 that we can use semi-Snarks to thin out a glider stream by any factor of our choosing). To fill in some of the gaps in this list, we present in Figure 8.5 some methods for thinning out a stream by a factor of 5 or 7.



(a) A collision that thins out glider streams (with period at least 108) by a factor of 5.

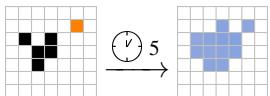


(b) A collision that thins out glider streams (with period at least 46) by a factor of 7.

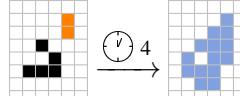
Figure 8.5: Some additional methods of multiplying the period of glider streams by small prime numbers.

8.1.1 Filters

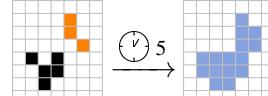
Another way of deleting some gliders within a stream is to use a *filter*, which is an oscillator that pulsates or gives off a spark in such a way as to destroy some of the gliders in the stream, but not all of them (similar to how we used a Schick engine to erase some of the gliders released by a space 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 sparkers (of appropriately large enough periods) to destroy gliders and thin out glider streams.



(a) A dot spark can delete a glider.



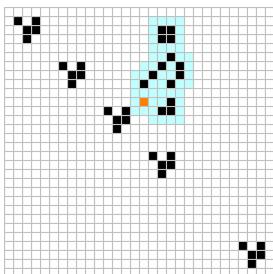
(b) A domino spark deleting a glider.



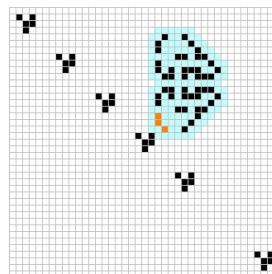
(c) A banana spark deleting a glider.

Figure 8.6: Many different sparks can be used to delete gliders.

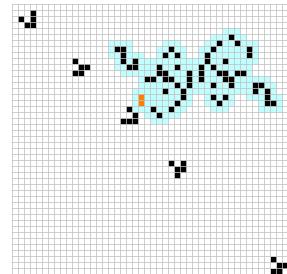
For example, if we place a period 8 blocker next to a glider stream of period $8n+4$ ($n \geq 2$), it destroys every second glider in the stream, thus doubling its period. Similarly, the period 16 oscillator “Rich’s p16” from Figure 3.42 emits a banana spark that can be used to double the period of any glider stream with period $16n+8$ ($n \geq 1$), 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 period 20, 24, and 33 in Figure 8.7.



(a) The dot spark from a blocker (outlined in aqua) 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 (outlined in aqua) can be used to destroy every other glider in a stream of period 24 (or $16n+8$ for any $n \geq 1$).



(c) A p22 sparker (outlined in aqua) 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 that can be used to double the period of certain glider streams.

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 causing a glider to disappear cleanly.

8.2 Glider Insertion

Our next technique is one that is complementary to that of the previous section: it lets us thicken up glider streams (i.e., it lets us decrease their period). The main idea behind this trick is to collide two gliders (or other objects) in such a way as to produce a single glider that is travelling in a different direction. 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 (and thus will crash into other gliders in the

input stream). 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 collision called a *tee* 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)).⁵ Another such method 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 6.22(a)).

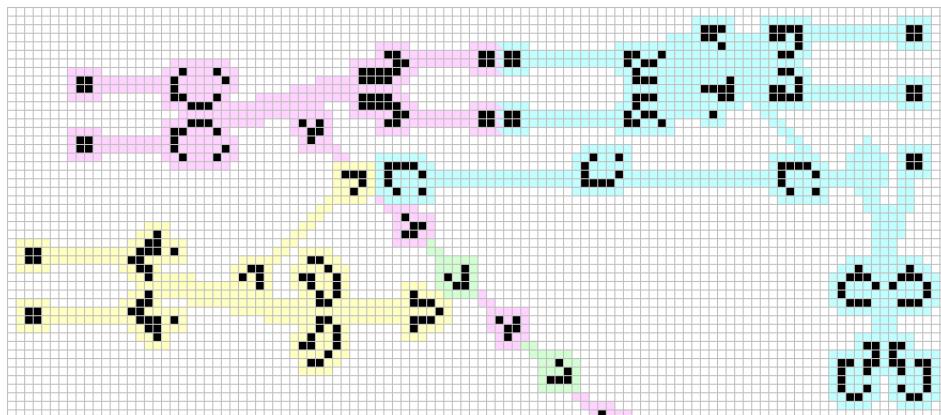


Figure 8.9: A period 23 glider gun constructed from several period 46 glider guns. The top-left glider gun in magenta fires a period 46 stream of gliders. To interlace this stream with another period 46 stream of gliders (outlined in green), we use the LWSS–glider collision of Figure 8.8(b): the glider 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 6.22(b)) outlined in aqua.

Unfortunately, the “tee” from Figure 8.8(a) cannot be used to produce glider streams of period below 25, and the LWSS–glider collision from Figure 8.8(b) can only produce streams of period down to 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 displayed in Figure 8.10(a), which uses

⁵An alternate “tee” reaction is presented in Exercise ??.

two sparks to transform a lightweight spaceship into a glider with essentially no debris occurring outside of that transformation.⁶ 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 typically be provided by a suitably-chosen oscillator. 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 trickier 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).

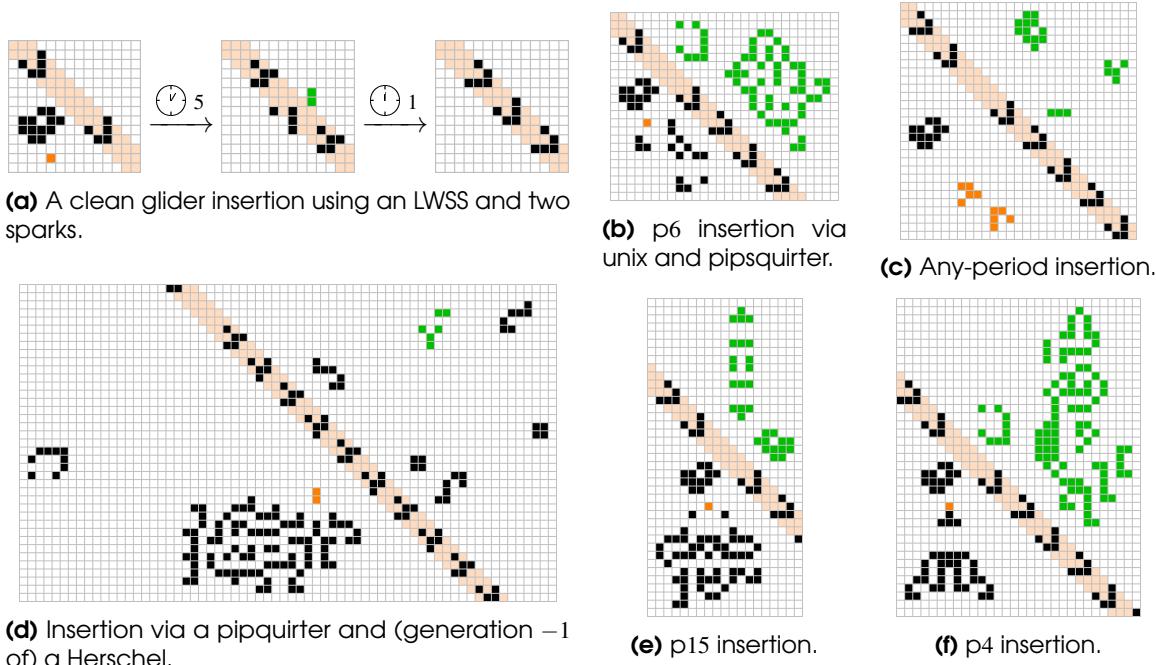


Figure 8.10: (a) A dot spark (in orange), followed by a domino spark 5 generations later (in green) can be used to transform a lightweight spaceship into a glider so cleanly that it can be used to fill in gaps in p14 glider streams—the tightest streams possible. The first spark can be provided by (b,d,e,f) a wide variety of different oscillators or (c) a two-glider collision, and 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.

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 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 these spaceships 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.

On 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

⁶This reaction was found by Dietrich Leithner no later than 1994.

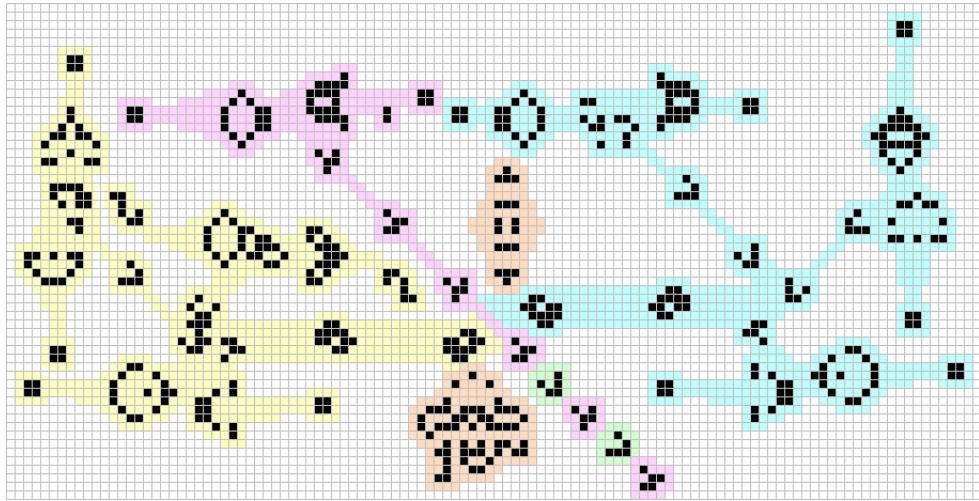
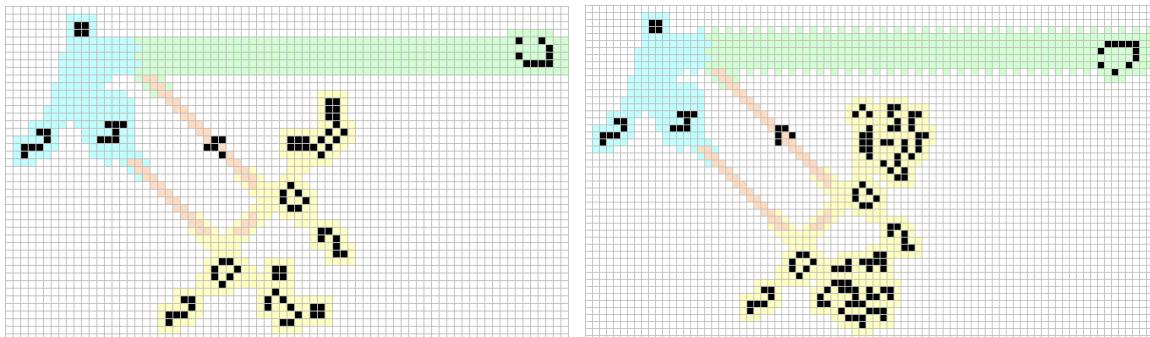


Figure 8.11: A period 15 glider gun constructed from several period 30 Gosper glider guns. The top-left glider gun in magenta fires a period 30 stream of gliders. To interlace this stream with another period 30 stream of gliders (outlined in green), we use the LWSS-LWSS-pentadecathlon-middleweight volcano collision of Figure 8.10(e).

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 turn a Herschel stream into an LWSS stream or an MWSS stream reasonably directly.⁷ However, the tight spacing of the required reflectors is such that this technique works best for streams whose period is a multiple of 3, since p3 and p6 bumpers are some of the only reflectors that fit so close together.



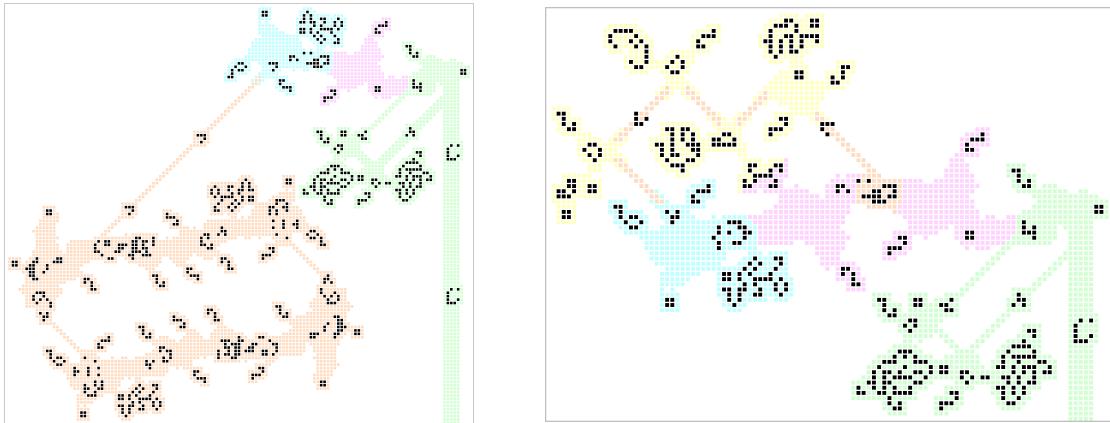
(a) A period 156 Herschel stream being converted into an LWSS stream. (b) A period 171 Herschel stream being converted into an MWSS stream.

Figure 8.12: A conduit (highlighted in aqua) 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 another Herschel conduit or two 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 attaching it directly to a Herschel track rather than to a gun and syringe.

⁷This 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 8.12.

Another (slightly smaller) period 84 LWSS gun of this type is displayed in Figure 8.13(b).



(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.

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 7.5 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.⁸ 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. However, the resulting p14 gun would be quite large—recall that creating a p15 gun required *seven* p30 guns in Figure 8.11, and they just performed 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, as there are some useful devices (like the syringe) that can be made to work at period 84 but not 70. It is straightforward to construct a period 84 glider gun using the adjustable-period gun of Figure 7.17, for example,⁹ 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 to p14. In particular, if we use one of the Herschel-to-multiple-glider edge shooters of Figure 7.7, we can then reflect one of its other output gliders so as to be on the same output lane as the glider that was shot from the edge of the conduit. Figure ??(a) illustrates one way of using this idea with the NE30T3 conduit to construct a p84 gun that emits 2 out of every 3 gliders in a p28 stream.¹⁰

⁸The first period 14 glider gun was built by Dietrich Leithner in November 1994 using these same ideas (and the same glider insertion reaction).

⁹In fact, we already did this in Figure 8.13.

¹⁰By rearranging the reflectors in this gun slightly, its period can be adjusted relatively straightforwardly—see Exercise 8.11.

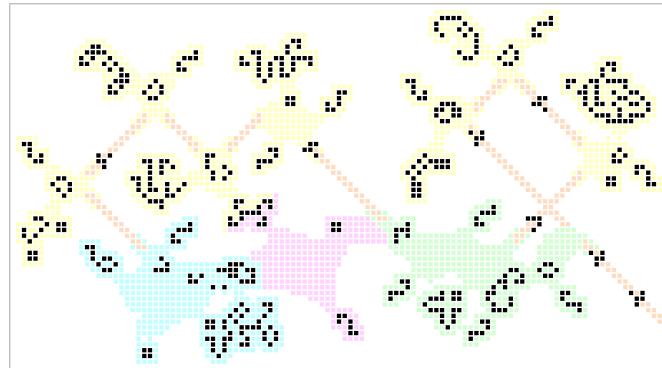


Figure 8.14: A p84 gun that emits 2 out of 3 gliders in a p28 stream. The central Herschel is fed through NE30T3 (highlighted in green) so as to create two output gliders, one of which is reflected (via standard reflectors like bumpers, bouncers, and Snarks, highlighted in yellow) into the same lane 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 8.15. Despite its large size, it is only about 50% larger than the smallest p14 gun that is currently known.¹¹ Furthermore, that smallest known p14 gun makes use of all the same mechanisms, just packed together somewhat more cleverly.

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—many of these guns themselves 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. Such guns are called *pseudo-period guns*, and all of the new low-period guns that we introduced in this chapter are of this type.

While there is nothing inherently wrong with pseudo-period guns, 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.

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 period¹² 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 ad-hoc methods that are known for creating true-period glider guns, and they only work for a few select periods. A brief summary of the known true-period guns of these few periods is presented in Table 8.1, and the next few subsections illustrate and describe the construction of these guns.¹³

¹¹When measured according to bounding box area. When measuring according to population, the gun from Figure 8.15 is roughly tied for smallest known.

¹²Herschel tracks can be used to create true-period guns of large period (period 62 and larger, in particular).

¹³Unfortunately, it probably will not be long before this list of true-period guns becomes out of date (5 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.

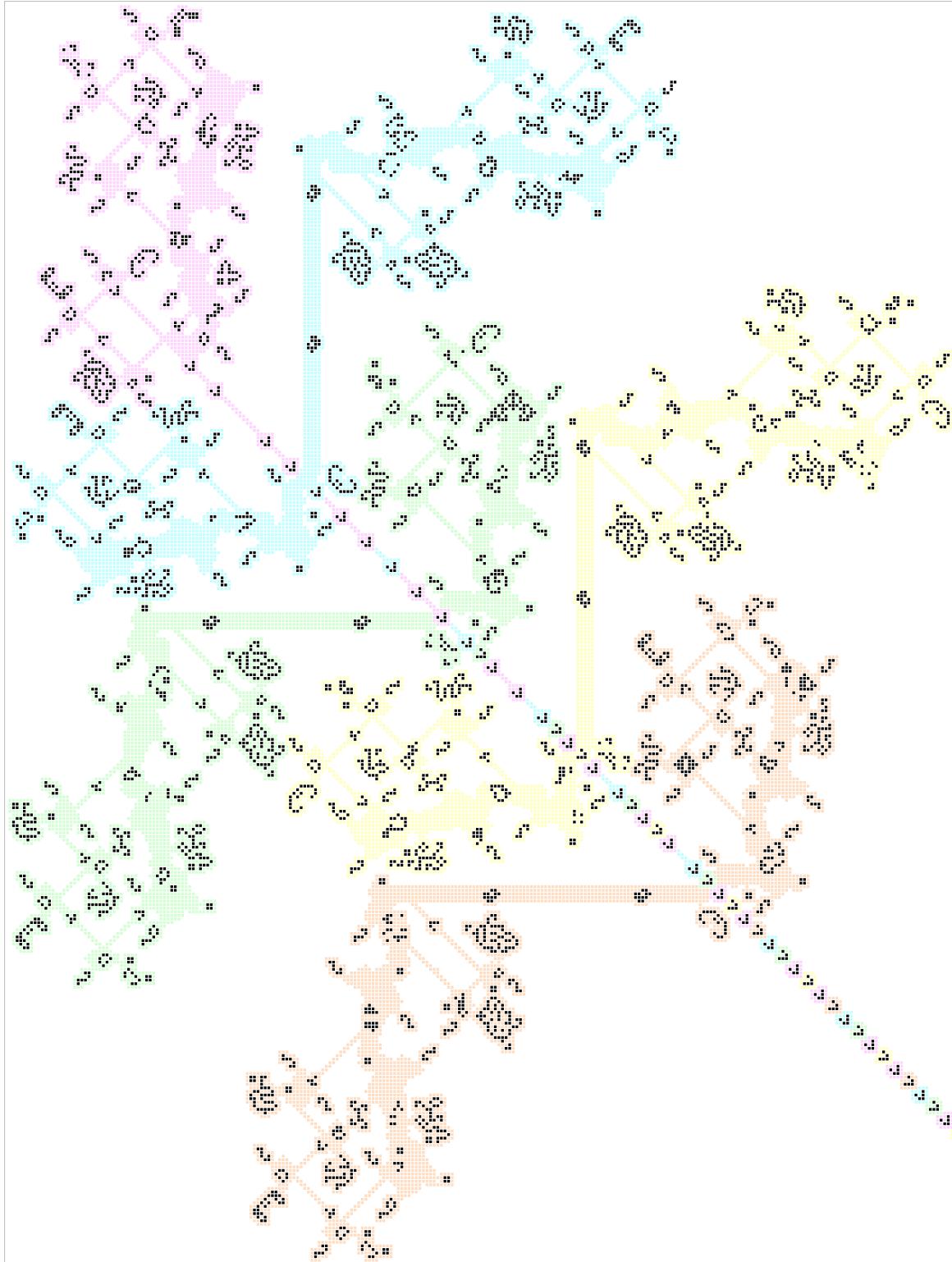


Figure 8.15: A period 14 glider gun that uses the p84 two-glider gun from Figure 8.14 (highlighted in magenta), as well as four copies of a p84 glider insertion mechanism (highlighted in aqua, green, yellow, and orange) to reduce the period of the stream down to $84/6 = 14$. The glider insertion mechanisms work 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.

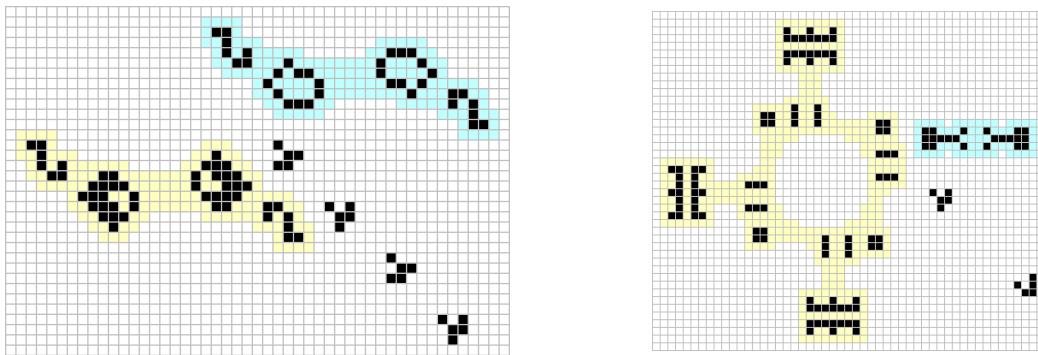
Period	Year	Original Discoverer	Example/Notes
20	2013	Matthias Merzenich Noam Elkies	upcoming Figure 8.19(b)
22	2000	David Eppstein Jason Summers	upcoming Figure 8.16(a)
24	1997	Noam Elkies	upcoming Figure 8.19(a)
30	1970	Bill Gosper	Gosper glider gun (Figure 1.18)
33	2018	Arie Paap Matthias Merzenich	upcoming Figure 8.18(b)
36	2004	Jason Summers	upcoming Figure 8.17(b)
40	2013	Adam P. Goucher Matthias Merzenich Jason Summers	<ul style="list-style-type: none"> smallest known (Figure 8.19(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 8.20)
45	2010	Matthias Merzenich	upcoming Figure 8.16(b)
46	1971	Bill Gosper	twin bees gun (Figure 1.23)
48	1997	Noam Elkies	uses a filter on the p24 gun from Figure 8.19(a)
50	1996	Dean Hickerson Noam Elkies	upcoming Figure 8.22
52	2018	Dave Greene Chris Cain Adam P. Goucher Matthias Merzenich	<ul style="list-style-type: none"> smallest known (Figure 8.28(a)) was constructed in November 2018 with help from ConwayLife.com forums user “Entity Valkyrie” first known (not displayed) was found 2 months earlier
54	1998	Dietrich Leithner	smallest known p54–56 guns (Figure 8.28(c–e)) were
55	1998	Stephen Silver	constructed in October 2018 by Chris Cain
56	1998	Dietrich Leithner	first known p54–56 guns (not displayed) were much larger—roughly 500×500
57	2016	Matthias Merzenich	<ul style="list-style-type: none"> smallest known (Figure 8.28(b)) was constructed in October 2018 by Luka Okanishi first known (not displayed) was large (about 330×350)
58	2016	“thunk” Matthias Merzenich	upcoming Figure 8.32
59	2009	Adam P. Goucher Jason Summers	<ul style="list-style-type: none"> first known (not displayed) was huge ($\sim 4000 \times 3000$) ConwayLife.com forums user “Entity Valkyrie” made a smaller one (Figure 8.24) in October 2018
60	1970	Bill Gosper	two Gosper glider guns (Figure 8.2(a))
61	2016	Luka Okanishi	upcoming Figure 8.30
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. Some of these guns were extremely large, and the guns of the same periods that we introduce in the upcoming sections are sometimes much smaller and make use of more recently-discovered mechanisms.

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 gun¹⁴ 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.

The true-period 45 gun displayed in Figure 8.16(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 8.16(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 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.



(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 8.16: 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.

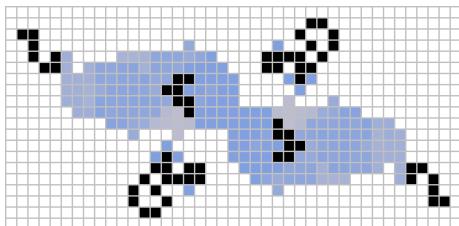
There are also glider guns of periods 33 and 36 that are mostly of this type—they work by colliding together large sparks from two copies of very sparky oscillators of periods 33 and 36, respectively. However, these guns also require a few extra stabilizing components to clean up the glider-producing reaction. For example, the period 36 oscillator in Figure 8.17(a) can be paired up with itself to create the true-period 36 gun in Figure 8.17(b), but it requires an extra heavyweight emulator to clean up a block that is left behind by the sparks otherwise, which would destroy the gun between the first and second glider that it creates.

Similarly, the period 33 oscillator in Figure 8.18(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 in Figure 8.18(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 is not destroyed after it creates its first glider.

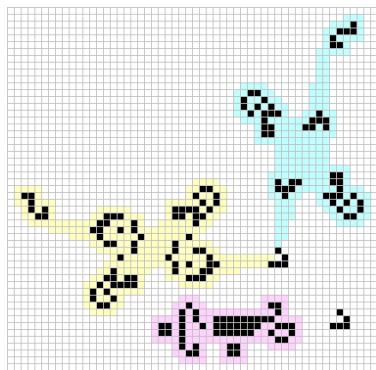
8.5.2 Hasslers: True-Period 20, 24, 40, 44, 48, 50, and 59 Guns

Quite a few true-period guns work by hassling a commonly-occurring unstable object like a T-tetromino or pi-heptomino, 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.

¹⁴Technically, 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.

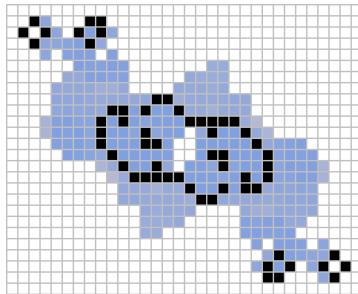


(a) A period 36 oscillator called *Jason's p36* that works by using jams and eater 1s to hassle two B-heptominoes. It was found by Jason Summers in July 2004.

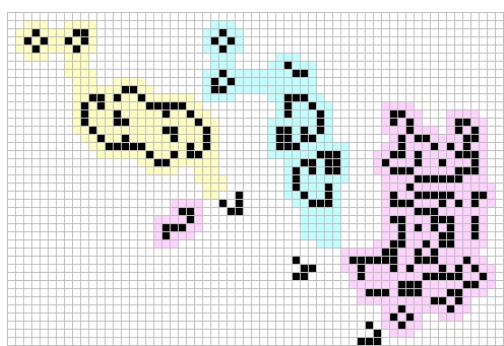


(b) A true-period 36 gun.

Figure 8.17: 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.



(a) A period 33 oscillator called *Jason's p33*. It was found by Jason Summers in August 2000 and shrunk somewhat by Matthias Merzenich in 2013.



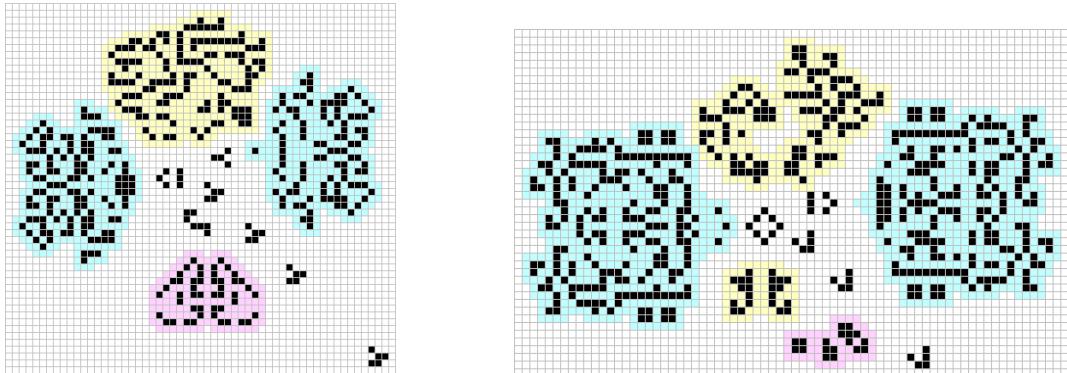
(b) A true-period 33 gun.

Figure 8.18: 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 and 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.

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.¹⁵ It turns out that 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 8.19(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 8.19(b). It is worth noting that this is the smallest-period true-period glider gun that is currently known. Furthermore, we can use a blocker to filter this p20 stream as FSince we can use Rich's p16 to double the period of period 24 glider streams

¹⁵This 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.



(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 8.19: 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).

as in Figure 8.7(a) to quickly and easily turn this into a true-period 40 gun.

The period 44 pi-heptomino hassler from Figure ?? is similarly the basis of all known true-period 44, 50, and 59 glider guns. For example, the true-period 44 glider gun works simply by replacing one of the 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,¹⁶ and the resulting gun is displayed in Figure 8.20.¹⁷

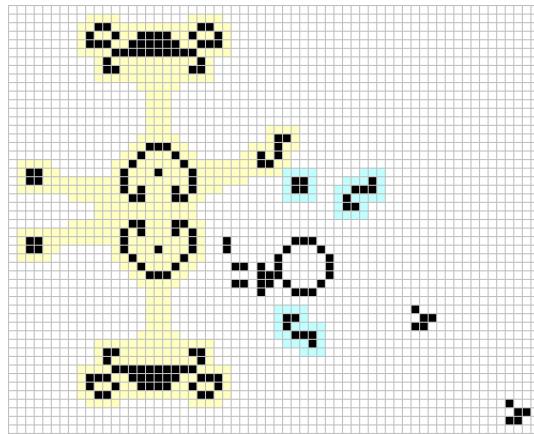


Figure 8.20: A true-period 44 glider gun that works by funnelling some of the debris from the p44 pi-heptomino hassler (highlighted in yellow) through a custom conduit (highlighted in aqua) so as to transform it into a glider.

By swapping out the heavyweight emulators on the sides of this reaction for an arrangement of two blocks and blinker, its period increases to 50 generations, but with the disadvantage that the blinker is pushed away by one cell when the debris hits it, so it needs to be pushed back if it is going to form part of a viable oscillator or gun. There are a number of sparks that can push the blinker back, perhaps

¹⁶The third eater 1 is not part of the conduit, but just replaces one of the other stabilizing blocks for spacing reasons.

¹⁷The first true-period 44 glider gun was found by David Buckingham in 1992. He then found this smaller version in October 1996.

the simplest of which comes from a collision of 3 queen bees. The resulting glider gun is displayed in Figure 8.21. However, there is one problem: this gun is pseudo-period, not true-period, since the queen bees oscillate at period 30 and thus the whole gun oscillates at period 150 (not period 50).

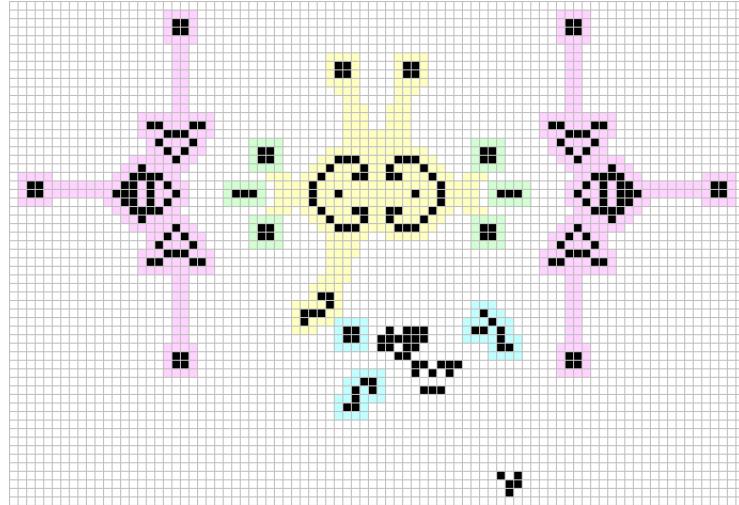


Figure 8.21: A pseudo-period 50 glider gun that works by using an alternate stabilization (highlighted in green), instead of the heavyweight emulators, to change the p44 pi-heptomino reaction (highlighted in yellow) to p50. The reaction pushes the blinker away by 1 cell, so some queen bees (highlighted in magenta) are used to push it back. As before, the conduit that transforms the pi-heptomino debris into a glider is highlighted in aqua.

To turn this gun into a true-period gone, we need a mechanism for pushing back the blinkers that oscillates at period 50 (or a divisor of 50). One such mechanism is the p50 traffic jam oscillator from Exercise 3.22. Unfortunately, this mechanism is rather large and thus, to prevent the glider-generating conduit from colliding with the traffic jam oscillator, we have to use *two* copies of the p50 pi-heptomino hassling reaction as well, so that rather than having the form

$$\text{queen bees} - \text{blinker} - \text{pi-heptominoes} - \text{blinker} - \text{queen bees},$$

as it did in Figure 8.21, the true-period 50 glider gun (displayed in Figure 8.22)¹⁸ has the larger form

$$\text{traffic jam} - \text{blinker} - \text{pi-heptominoes} - \text{blinker} - \text{pi-heptominoes} - \text{blinker} - \text{traffic jam}.$$

Note that this gun can be made somewhat smaller by cleverly using a glider to stabilize one side of the pi-heptomino reaction (see Exercise 8.9).

Finally, this pi-heptomino hassling reaction can also be extended to period 59 by carefully hitting it with gliders. In particular, the reaction displayed in Figure 8.23 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).¹⁹

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 smallest-known true-period 59 glider gun that can be constructed via this method is presented in

¹⁸This gun is a slightly smaller version of the first known true-period 50 glider gun, which was found by Dean Hickerson just one day after David Buckingham found the small p44 gun in 1996.

¹⁹This reaction was found by Jason Summers in December 2009.

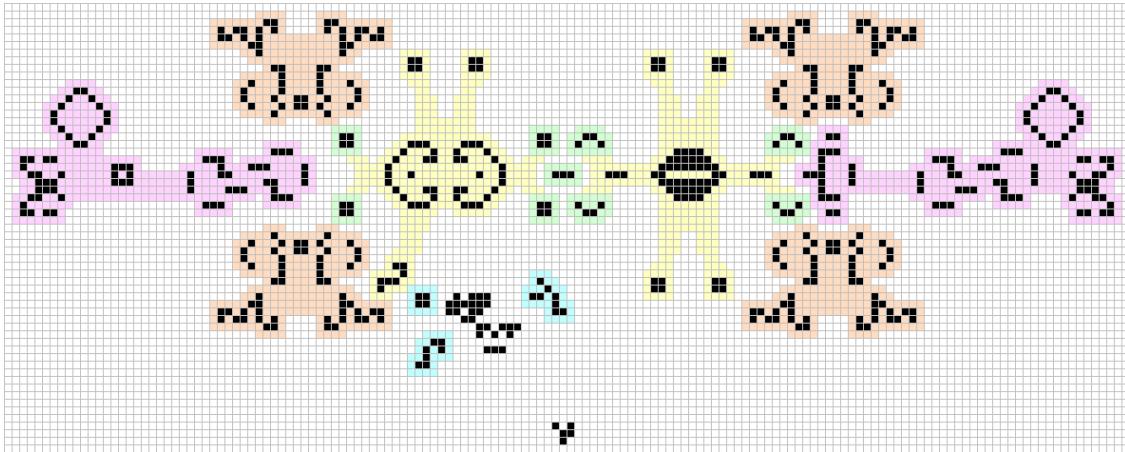


Figure 8.22: A true-period 50 glider gun that works just like the pseudo-period 50 gun, but uses the traffic jam oscillator (highlighted in magenta) and some middleweight volcanoes (highlighted in orange) to stabilize the ends instead of queen bees. To prevent the conduit (highlighted in aqua) from colliding with the middleweight volcanoes, we have to use two copies of the pi-heptomino reaction (highlighted in yellow) rather than just one. As before, a special configuration of blocks and blinkers is used between the different p50 components, and is highlighted in green.

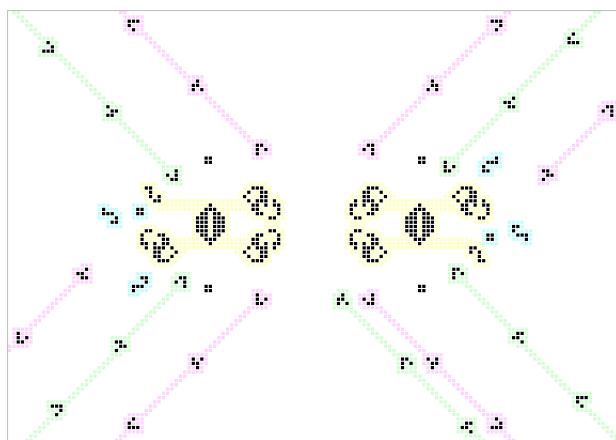


Figure 8.23: A period 59 reaction that takes in 5 gliders (highlighted in green) and spits out 6 gliders (highlighted in magenta). The reaction works by using two copies of the pi-heptomino reaction (highlighted in yellow) to create two gliders via a conduit (highlighted in aqua) and four more gliders when the debris from these reactions collide with each other at the center. Since it emits more gliders than it destroys, it can be used to construct a period 59 glider gun.

Figure 8.24, and it uses Snarks (as well as some of the welded Snarks) to reflect most of the output gliders from this reaction right back into itself, thus “fueling” the creating of the single excess glider.²⁰

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.

8.5.3 Quetzals: True-Period 52, 54–58, and 61 Guns

The remaining known true-period guns are all based on Herschel tracks, but they make use of custom (typically oscillating) components to overcome the fact that a Herschel’s first natural glider will

²⁰The first true-period 59 gun used the same reaction and was also built in December 2009, by Adam P. Goucher. Since the Snark had not yet been discovered, it used (very large) reflectors based on Herschel tracks instead, and had more than 300 000 live cells.

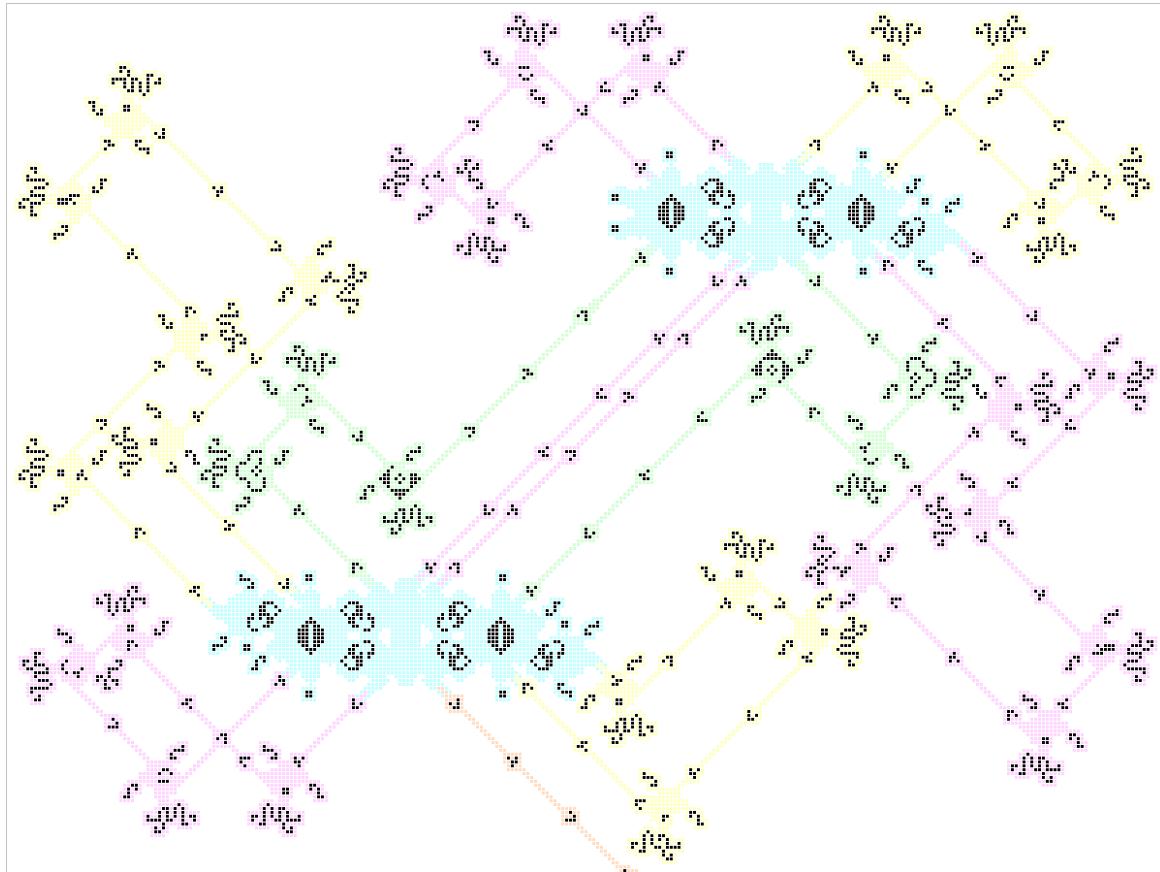
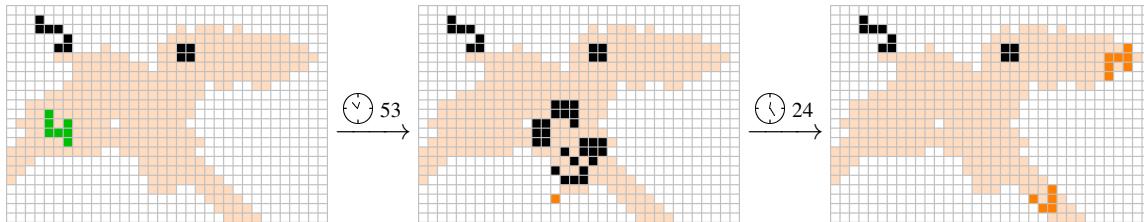


Figure 8.24: A true-period 59 glider gun that was constructed by ConwayLife.com forum user “Entity Valkyrie” in October 2018. It works by using two copies of the 5-to-6 glider reaction from Figure 8.23 (outlined in aqua) and several Snarks to bounce the output gliders of those reactions back into themselves (the tracks that the gliders follow are highlighted in yellow, green, and magenta).

collide with a Herschel following behind it by fewer than 62 generations, and thus cannot be used as the output of the gun. A gun of this type is called a *quetzal* or *quetzalcoatlus* 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” (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 subsequent Herschel. Some custom conduits that implement the first of these ideas (glider extraction) at some low periods are presented in Figure 8.25, and some additional methods of performing this same task are presented in Exercise 8.14. 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.

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 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 51 or 53 are currently known). These various fast



(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.

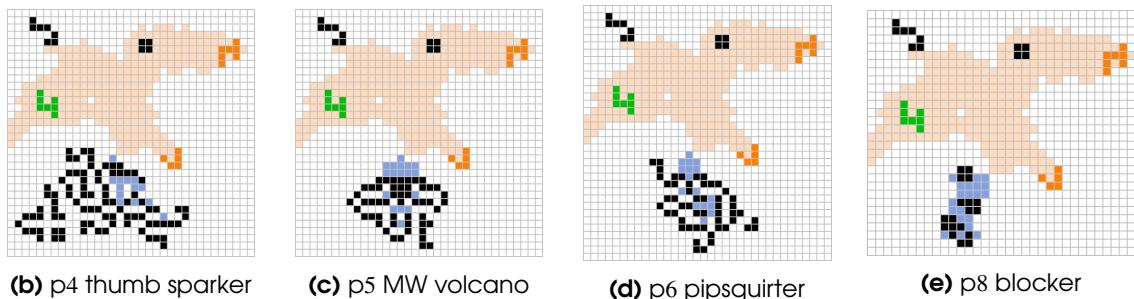


Figure 8.25: A spark can be used to extract a glider from the Fx77 Herschel conduit. This spark can be provided by numerous different oscillators of several different periods.

eaters are displayed on the Fx77 conduit in Figure 8.26.

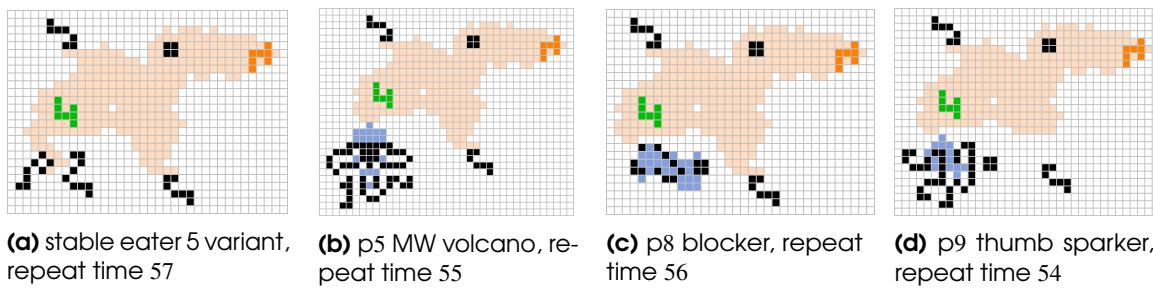
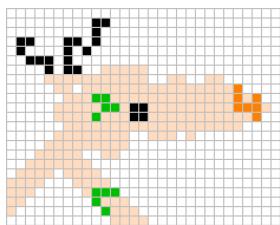


Figure 8.26: The repeat time of the Fx77 Herschel conduit can be reduced by quickly and cleanly eating each Herschel's first natural glider. The repeat time can be made as low as (a) 57 via stable components, or (b-d) 51 via oscillating components (though the oscillating components shown here only go as low as a repeat time of 54).

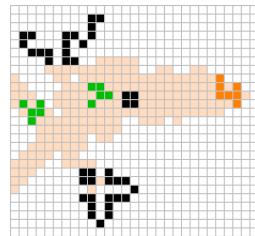
One final component that is extremely useful in these low-period situations is *Jormungant's G-to-H*,²¹ which is the conduit displayed in Figure 8.27 that quickly converts two gliders into a Herschel. While a syringe (refer back to Figure 7.8) has a repeat time of 78 generations, which is too large for 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 (assuming that the first natural glider of the output Herschel can be erased quickly enough to allow for such a low repeat time). 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 8.28. 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

²¹Found by Louis-François Handfield, who goes by the online pseudonym “Jormungant”, in April 2018.



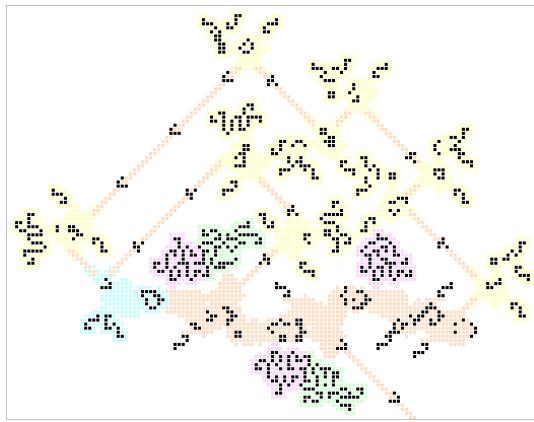
(a) helper glider from the southeast



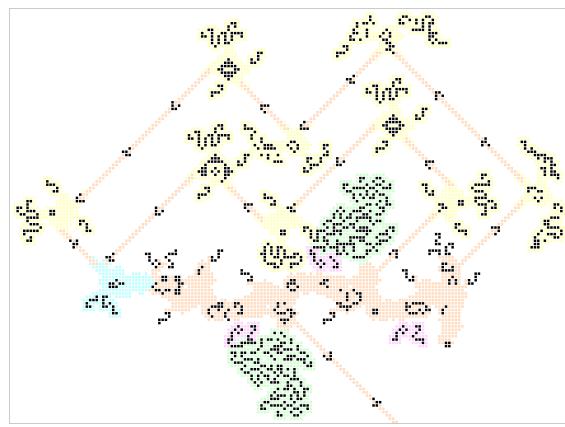
(b) helper glider from the northwest

Figure 8.27: 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.

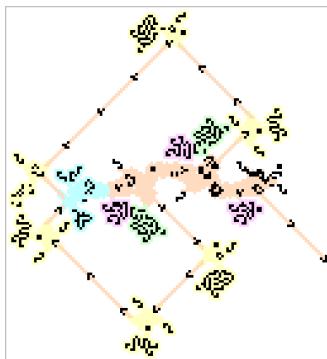
the low-period conduits that we saw earlier (for example, $56 = 2^3 \times 7$, so the period 56 gun can make use of p4, p7, and p8 components).



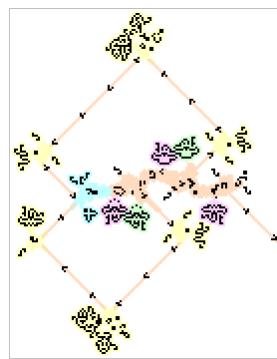
(a) A true-period 52 gun that uses an unnamed p13 oscillator to eat first natural gliders (FNGs) and p4 thumb sparkers to extract other gliders from Herschels.



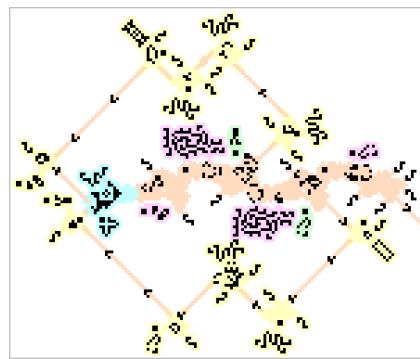
(b) A true-period 57 gun that uses eater 5s to eat first natural gliders and an unnamed p3 oscillator to extract other gliders from Herschels.



(c) A true-period 54 gun that uses p9 thumb sparkers and p6 pipsquirters.



(d) A true-period 55 gun that uses middleweight volcanoes.

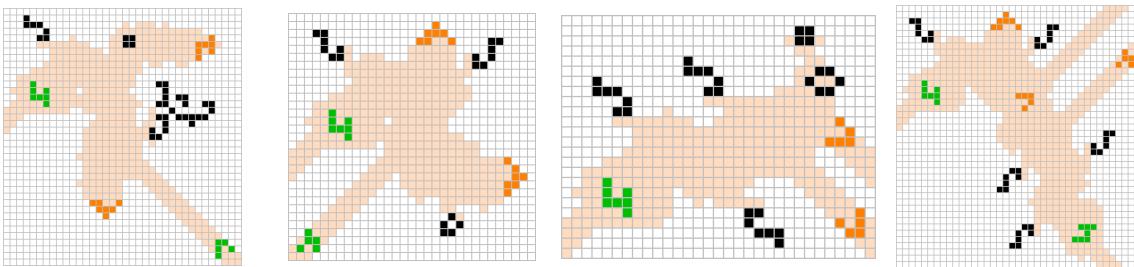


(e) A true-period 56 gun that uses blockers and p8 pipsquirters to eat FNGs and extract other gliders.

Figure 8.28: 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 Fx77's and oscillators to eat its first natural gliders faster than stable components can (as in Figure 8.26, highlighted in magenta). That Herschel is converted into three gliders—two of those gliders are extracted by oscillators (as in Figure 8.25, 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.

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 periods due to not being 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 (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 8.29). 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, and the other can be used as the output of the gun.²²



(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. The conduit on the right is somewhat more complicated, but has the benefit of the output gliders being slightly farther apart.

Figure 8.29: 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 8.16) 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.

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 repeat times of 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.²³ Using this trick lets us construct the true-period 61 glider gun displayed in Figure 8.30.²⁴

This same technique can be used to construct a period 58 true-period 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 8.31 that we can extract a glider from.

The second issue that is somewhat messier to deal with in this period 58 case is that the method we used to reflect a glider (and thus separate the two closely-spaced streams) in Figure 8.30 does not work at period 58. We thus must use a different method of separating those two streams, and one method that works is to use a rephaser as in Exercise 4.11. In particular, using two rephasers puts

²²It is worth comparing these conduits with the Herschel transceivers from the “Notes and Historical Remarks” section of Chapter 7; sometimes it is easier or quicker to send a tandem glider pair rather than individual gliders.

²³This method does not work at period 60 or lower—see Exercise 8.17.

²⁴First true-period 61 gun was found by Luka Okanishi in April 2016. The smaller gun displayed here was constructed the next day with the help of Chris Cain.

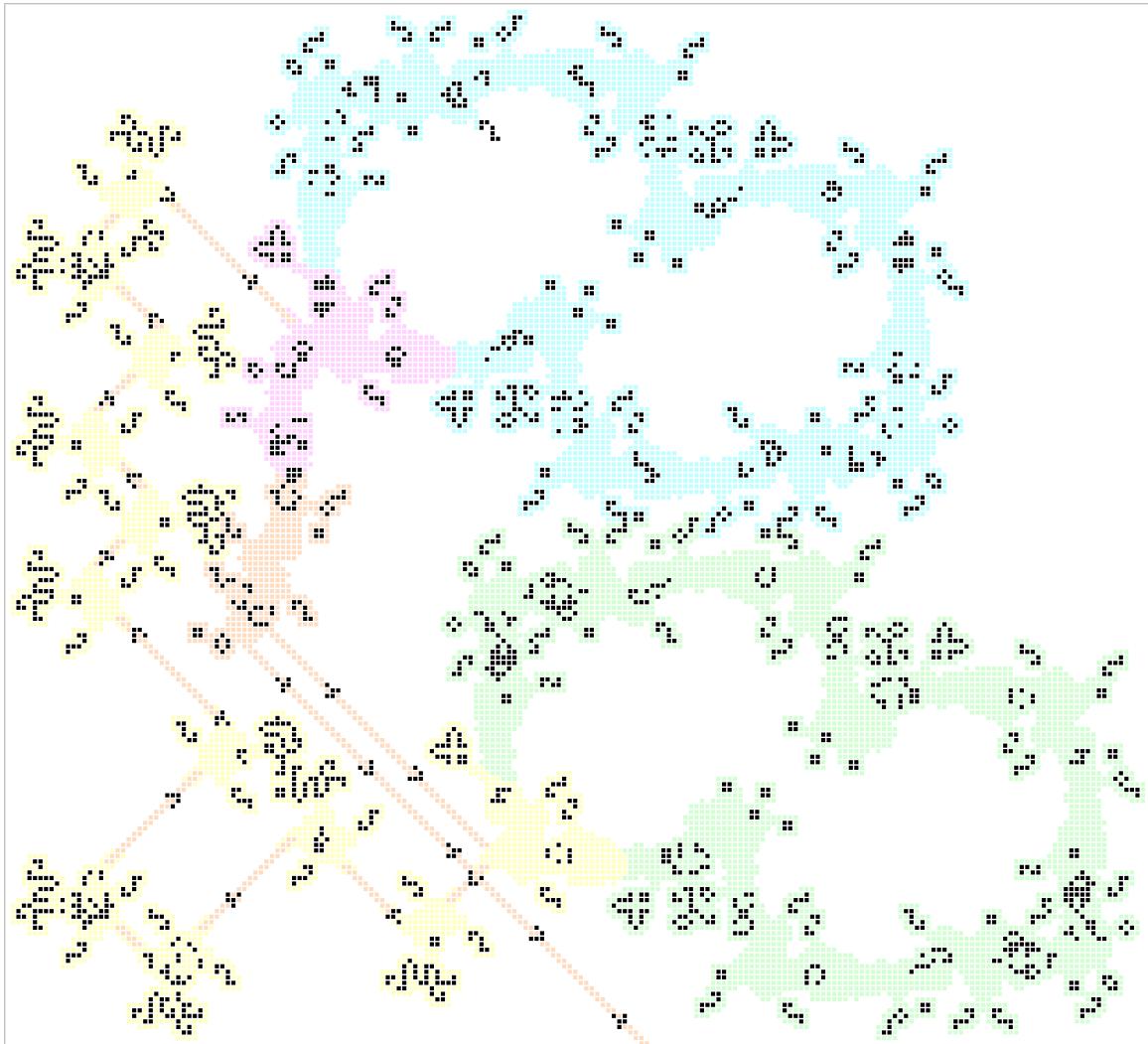


Figure 8.30: A true-period 61 glider gun. It uses one copy of a period 61 Herschel track made up of R64, L112, and L200 conduits (highlighted in aqua) to generate a Herschel via the trick of Figure 8.29(a) (highlighted in magenta). That Herschel is then converted into two gliders via the conduit of Figure 8.29(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.



Figure 8.31: 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).

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 true-period 58 gun is displayed in Figure 8.32,²⁵ and it completes our collection of known periods with true-period guns.

²⁵This gun was found by Matthias Merzenich and ConwayLife.com forum user “thunk”, just one day after the p61 gun.

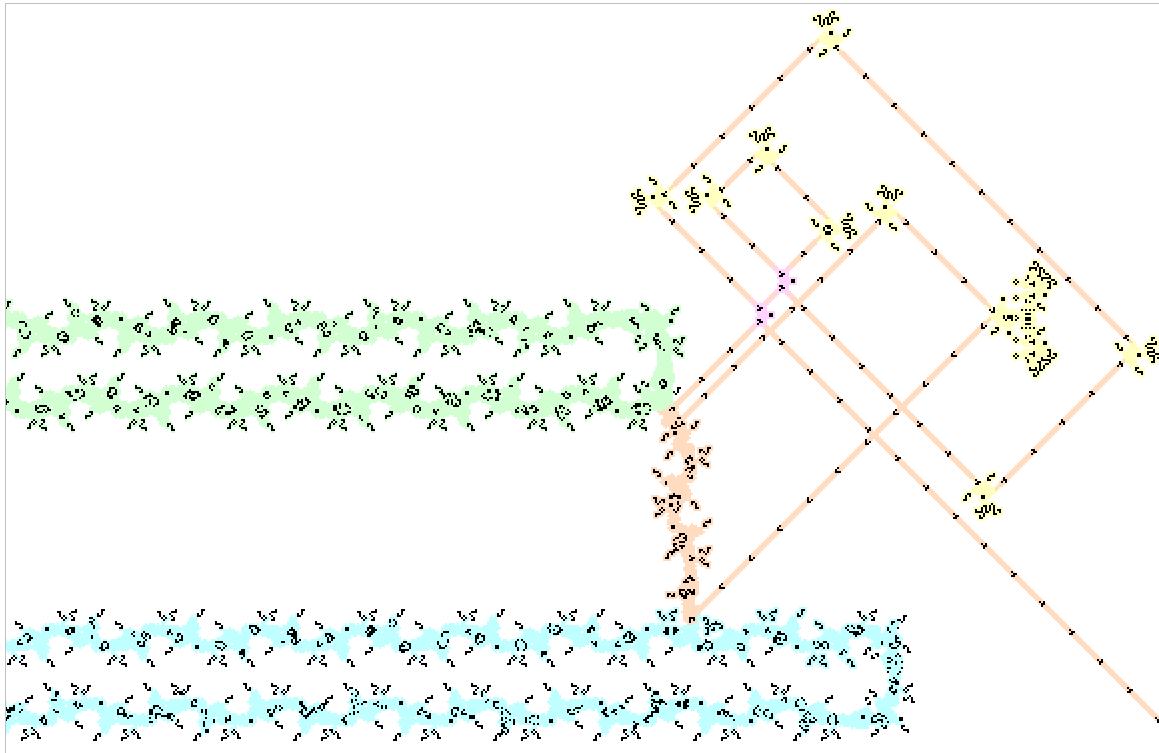


Figure 8.32: A true-period 58 glider gun that works very similarly to the p61 gun of Figure 8.30. A glider is used to extract a Herschel from the period 58 Herschel track oscillator from Figure 8.31 (truncated and highlighted in aqua). That Herschel then collides with another copy of that p58 oscillator (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.

8.6 Slide Guns

Now that we know how to construct glider streams with essentially any spacing of our choosing, 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 8.33.

There are similarly numerous ways of colliding gliders with blocks so as to create a glider traveling perpendicularly 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 8.34. 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, that reactions in parts (e) and (f) of Figure 8.34 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).²⁶

Reflection reactions of this type are particularly useful as they let us use guns along just a finite

²⁶We first saw that a boat can be used as a one-time turner in Exercise 5.29.

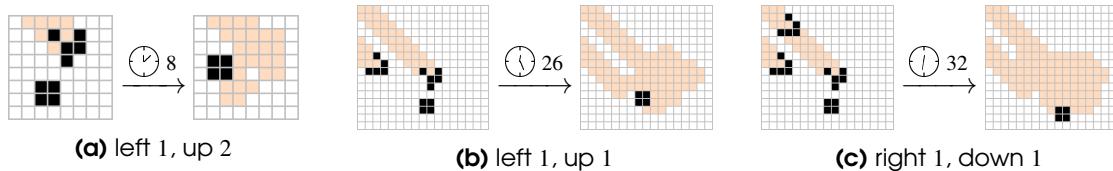


Figure 8.33: Some reactions that use gliders to move a block around the Life plane. By combining these reactions with each other, the block can be moved to any location. The reaction (a) is the same as the one from Figure 5.21(a), while (b) was found by Dean Hickerson no later than March 1990 and (c) 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.

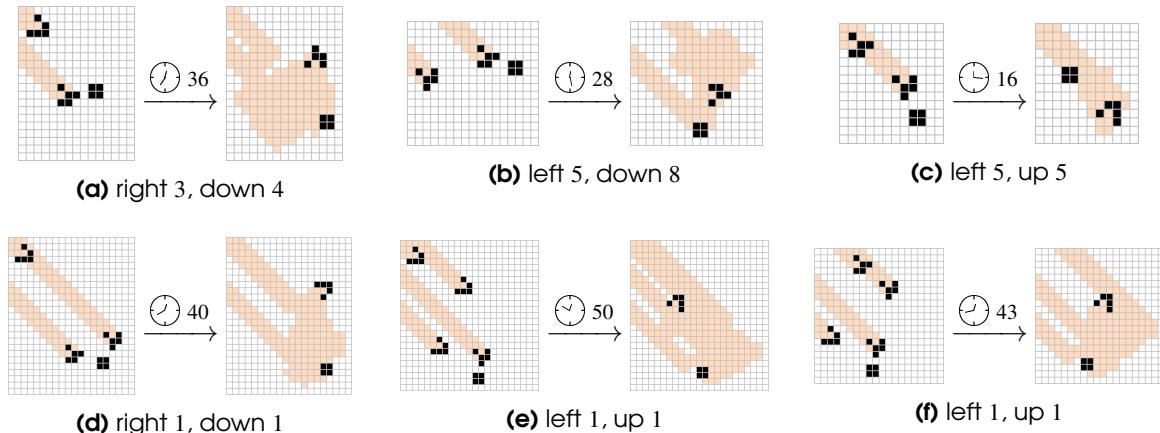


Figure 8.34: Some ways of colliding gliders coming from one direction with a block so as to produce a glider traveling perpendicularly. The block is not destroyed, but it may be moved—it can then be moved back, if desired, by the reactions shown in Figure 8.33. Reactions (e) and (f) were found by Paul Chapman in 2004 and make use of the same first two gliders as the block mover from Figure 8.33(b). Their final glider can be delayed by any amount, and the output gliders of these two reactions have opposite colors.

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 lanes. For example, if we use one glider to shift a block as in Figure 8.33(a) and then two more gliders to shift a block and reflect a glider as in Figure 8.34(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 8.35.²⁷

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 8.34(d) can be used to create a slide gun that has gliders separated by 2 lanes instead of 4 (see Exercise 8.18), and we could even create a “slide gun” that doesn’t slide at all by choosing a block-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 8.34(a) followed by the move of Figure 8.33(a) and then the move of Figure 8.33(b) twice (see Exercise 8.23).

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 down and to the right. The resulting “double” slide gun is displayed in Figure 8.36—it is made up of 10 glider guns (instead of just 3 like the regular

²⁷The first slide gun was constructed by Dietrich Leithner in July 1994. It made use of the reaction from Figure 8.34(d) (see Exercise 8.18).

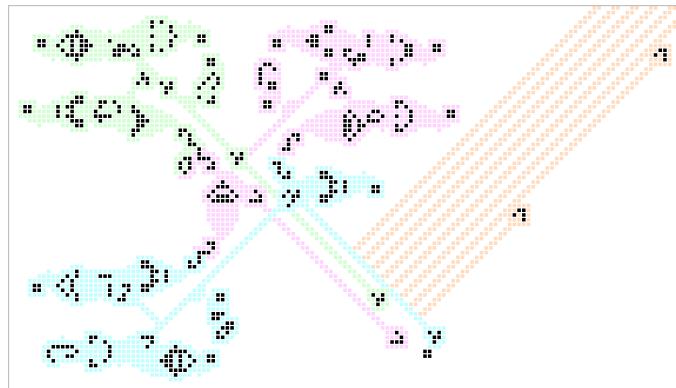


Figure 8.35: The glider highlighted in aqua moves the southeastern block and then the two gliders highlighted in magenta and green move it and create a perpendicular glider. The total resulting movement of the block is 2 cells southeast, so this *slide gun* fires gliders on infinitely many different 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).

slide gun from Figure 8.35), 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).

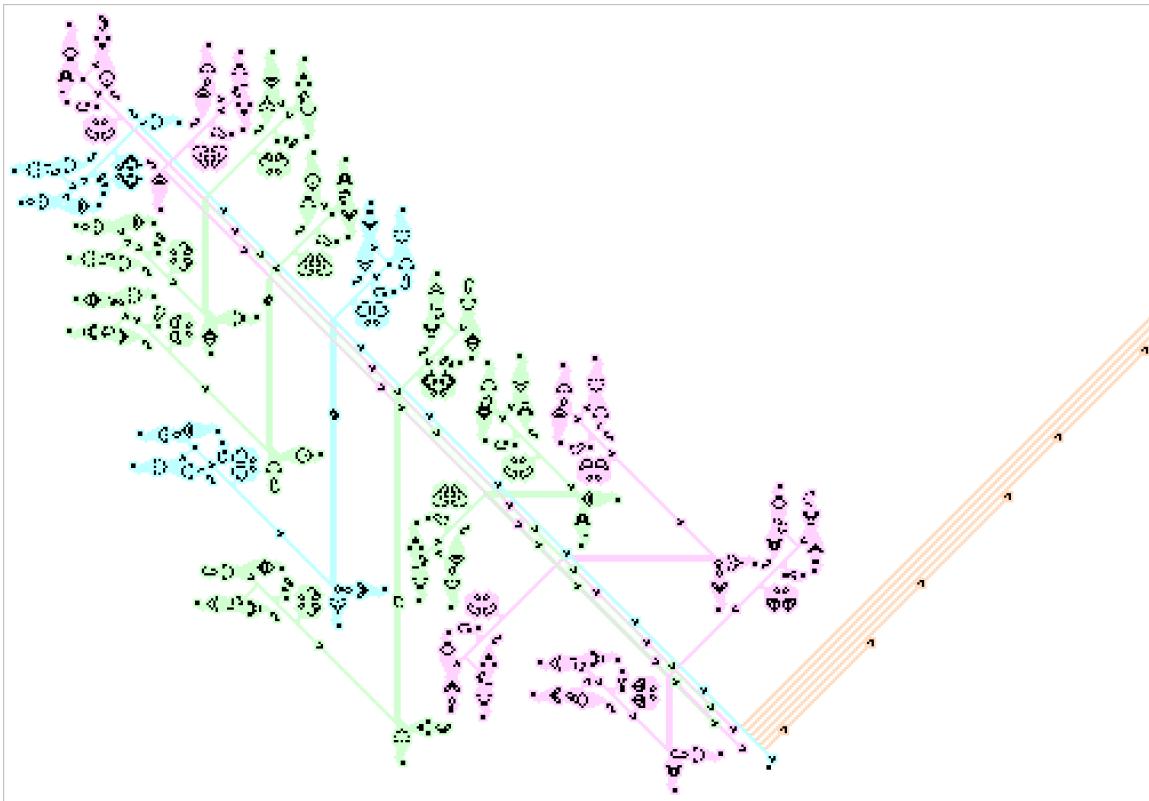


Figure 8.36: A slide gun that fires two gliders on each of its output lanes. The numerous p240 guns that make up this slide gun are each the same as the p120 guns from Figure 8.35, but with Rich's p16 attached to filter the output. The 10 gliders used in the block-pushing recipe consist of two copies of the single-glider block mover from Figure 8.33(a) (highlighted in aqua), two copies of the two-glider block mover from Figure 8.33(b) (highlighted in green), and two copies of the two-glider reflector from Figure 8.33(a) (highlighted in magenta).

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 show how to use these ideas to build a pattern that can create copies of any pattern that can be synthesized by gliders (including itself).

8.7 Armless Construction

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” collision of Table 5.2 and Figure 8.8(a), 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’ll need two separate tee reactions and a total 6 different lanes if we want to generate perpendicular gliders of both colors.

Luckily there is a known component that converts a single Herschel signal directly into a pair of gliders that can be used in an alternate three-glider tee reaction. This means that we won’t need three loop guns to produce three gliders. Instead we can have each glider in a loop gun generate a glider pair using this H-to-2G component. Another gun can produce the corresponding antiparallel gliders.

This method of generating gliders is particularly useful for implementing slow salvo syntheses.²⁸ If we want to create a gun that implements any given slow-salvo synthesis, we could use numerous carefully-placed high period-guns (many of which will have to be edge shooters). But an easier way would be to encode the slow salvo synthesis in a two pairs of (quite large) loop guns, one pair for each glider color. 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.

Unfortunately, slow-salvo syntheses of even moderately-sized objects like Corderships tend to require hundreds of gliders, so huge and unwieldy loop guns would needed to generate those syntheses. However, if we are willing to use 2-direction slow syntheses instead, then we can synthesize a 2-engine Cordership via just 17 pairs of gliders.²⁹

These are *slow glider pairs*, meaning that the action of each pair is completely independent from all pairs before it. 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. This is a very useful feature when the glider production mechanism has a significant speed limitation, as we’ll see in the next section.

Each of the two salvos contains gliders of both colors, so we’ll have to include two pairs of loop guns similar to the gun from Figure 7.17 that duplicates whatever gliders are placed on its track. Each pair of guns will be dedicated to producing all the gliders of one color. The above 17 slow glider pairs can be created by a total of 8 loop guns containing a total of 68 gliders. For each output glider in each salvo, two gliders are added to one of the pairs of guns, depending on the output glider’s color. The result is the 2-engine Cordership gun displayed in Figure ??.

A tee reaction can be a convenient component in many situations where a slow salvo or a synchronized salvo must be generated. 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 of the salvo. A tee reaction can reach across any number of lanes, as long as the three gliders don’t interfere with gliders that have already been placed in the salvo.

8.7.1 Multicolor Armless Construction

It’s a little awkward to have to use a separate pair of loop guns to generate each of the two glider colors. It means there are more glider streams to maintain, and additional constraints come into play

²⁸Slow salvo synthesis was explored in Section 5.7.

²⁹In particular, we make use of the seed for the 2-engine Cordership that we constructed in Exercise 5.39.

because one of the glider colors has to cross the construction area for the other glider color, where the tee collisions are occurring. In future chapters we will come across situations where it's much better to keep the number of individual signal storage loops to a minimum. So the question arises: is there a way to encode both colors of output glider into a single pair of loop guns?

There is in fact a known way to do this. 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 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 74 ticks.

The simplest known reactions of this type are displayed in Figure 8.37, and they make use of 3 antiparallel 2-glider collisions to generate an output perpendicular glider of either color. Slowing down the input gliders in one half of these reactions by $8n$ generations (i.e., moving them backward by two full diagonal steps) will move the output glider in that direction by $2n$ lanes, or one full diagonal. So we can use these two 6-glider recipes to create an output glider on any perpendicular lane of our choosing.

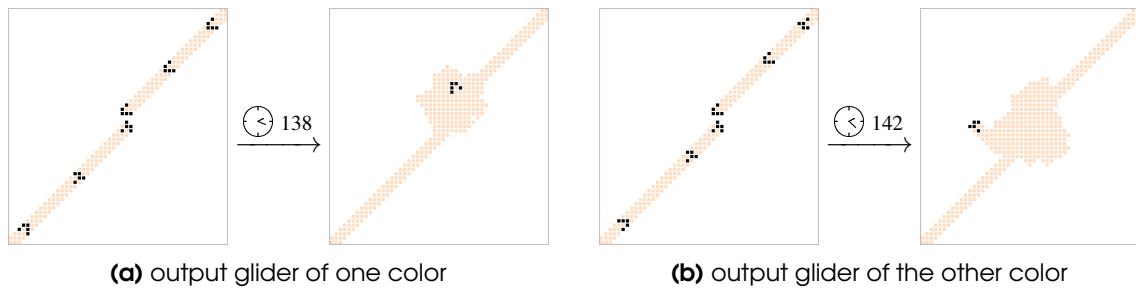


Figure 8.37: A way of colliding 3 single-lane antiparallel pairs of gliders so as to create perpendicular gliders of either color. The collisions are *slow*, so the glider pairs can be delayed relative to each other by any number of generations without affecting the overall reaction (well, the delay must be even since some of the intermediate collisions create p2 traffic lights).

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 any slow-salvo recipe we might want. For example, if we want to create a gun that cycles between firing gliders on four consecutive lanes, we first figure out which $4 \times 3 = 12$ 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.³⁰ In particular, for each output glider that we wish to generate, we first choose which collision from Figure 8.37 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 (2 full diagonals) at a time until the output glider is produced in the correct lane. The resulting gun is displayed in Figure 8.38.

As a more advanced example, the same 17 glider pairs in the Cordership recipe from the previous section can be created by a total of just 4 loop guns instead of eight. Each gun has a track containing $3 \times 17 = 51$ gliders each, giving us the 2-engine Cordership gun displayed in Figure 8.39.

8.7.2 Comparison of Methods

A tee-reaction armless design has a lower total population (only two stored loop gliders per output glider) and consequently a faster repeat rate. To change the output lane it's only necessary to move a single glider by $2N$ cells diagonally; to change the output glider's timing we simply delay or advance the two paired gliders in two connected loop guns.

Multicolor armless designs have other compensating advantages besides the smaller number of loops. Interleaving different colored gliders becomes relatively trivial, with no need to worry about gliders from the back pair of guns interfering with the forward guns' insertion reaction (or vice

³⁰This is the same method that we used in Section 6.3.1 to create ticker tapes.

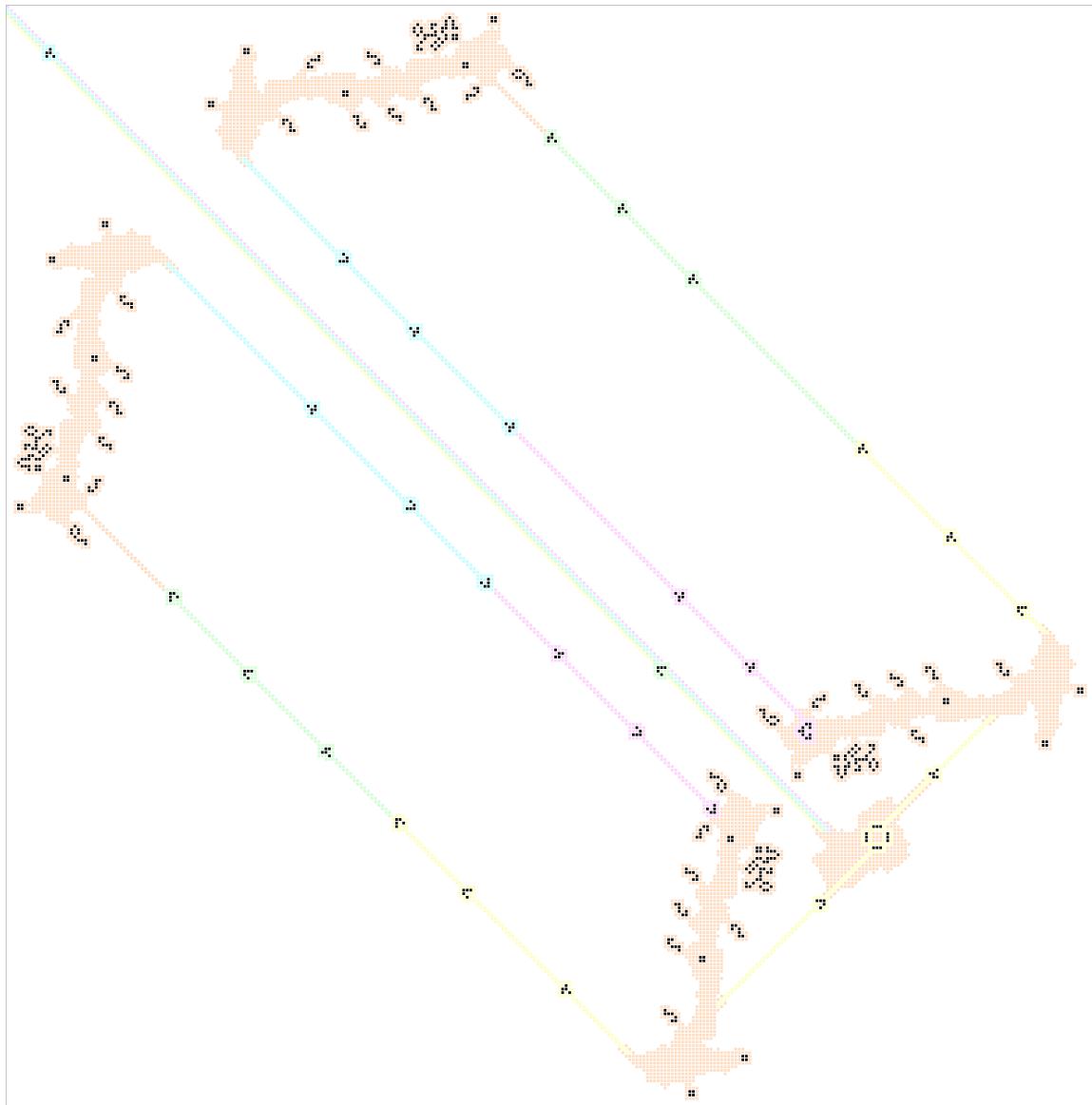


Figure 8.38: A gun that fires gliders to the northwest on four consecutive lanes (highlighted in order from right-to-left in magenta, aqua, green, and yellow). The first and third gliders are created by the reaction from Figure 8.37(a), while the second and fourth gliders of the other color are created by the reaction from Figure 8.37(b).

versa). The paired streams can be reflected into place with just plain Snarks if needed, and they're ambidextrous, if that's ever useful: swap the glider triplets to the opposite guns, and the glider will fire in the opposite direction. A tee-reaction setup can't do that.

Thus, choosing one of these mechanisms over the other has some definite implications about priorities. If overall simplicity is more important for a given project, then it might make sense to use three glider pairs to insert each glider. The conduit that creates two gliders for the tee reaction is not something we know how to build easily, especially with slow salvos, so it wouldn't be a reasonable part of any circuitry that must itself be constructed by gliders. But it's very easy to assemble Spartan circuitry that puts simple pairs of gliders on two opposing lanes.

The biggest advantage of armless construction is that the general form of the gun would not significantly change if we instead wanted the gun 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

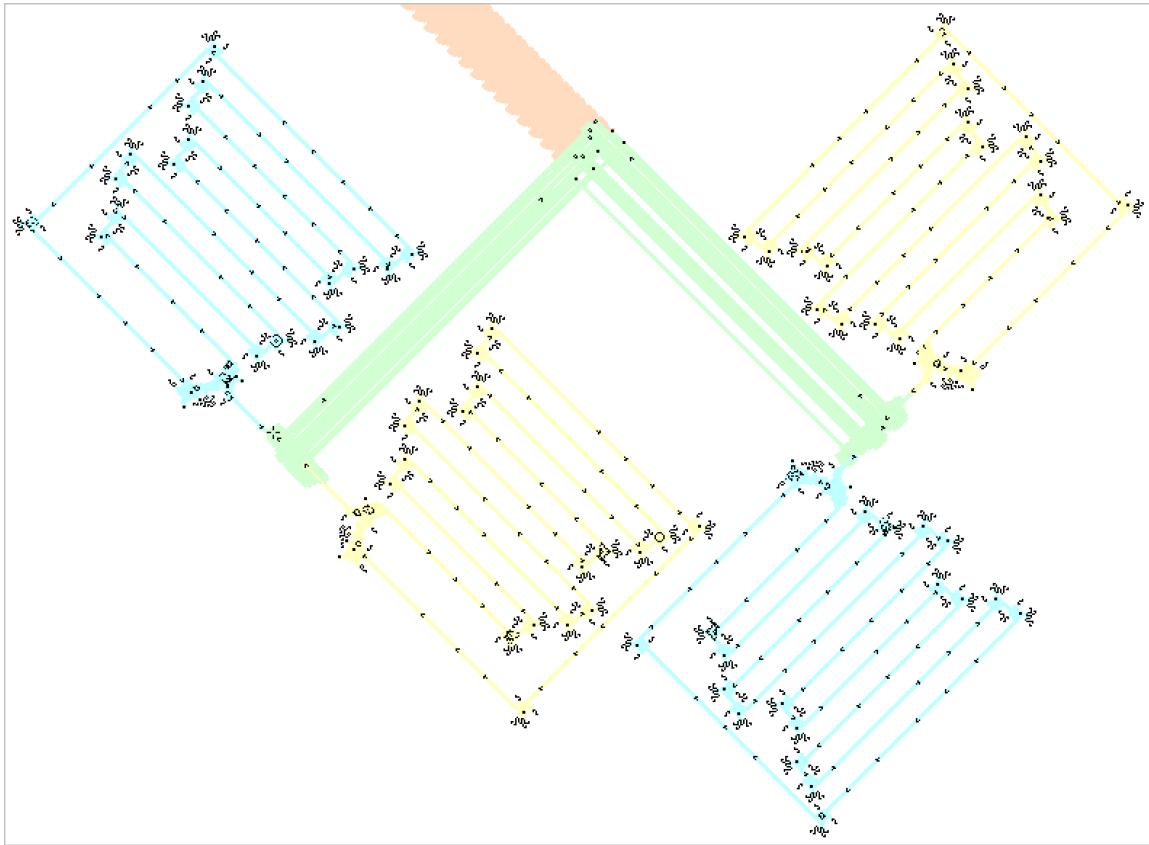


Figure 8.39: A 2-engine Cordership gun that works by using four loop guns (highlighted in aqua and yellow) that each contain 51 gliders along their track (the tracks are “folded” via Snarks just to reduce their size). Those loop guns create 17 slow glider pairs along various lanes (highlighted in green), which synthesize the Cordership.

Chapter 11.

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 6.21, 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 7.15, and two different ways of using armless construction to fire 2-engine Corderships using loop guns are shown above.

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 gets longer and longer. This can be achieved by making use of the block-pushing reaction that we saw back in Figure 7.35—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 7.34. 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 asymptotically proportional to \sqrt{t}), and one example is displayed in Figure 8.40.³¹

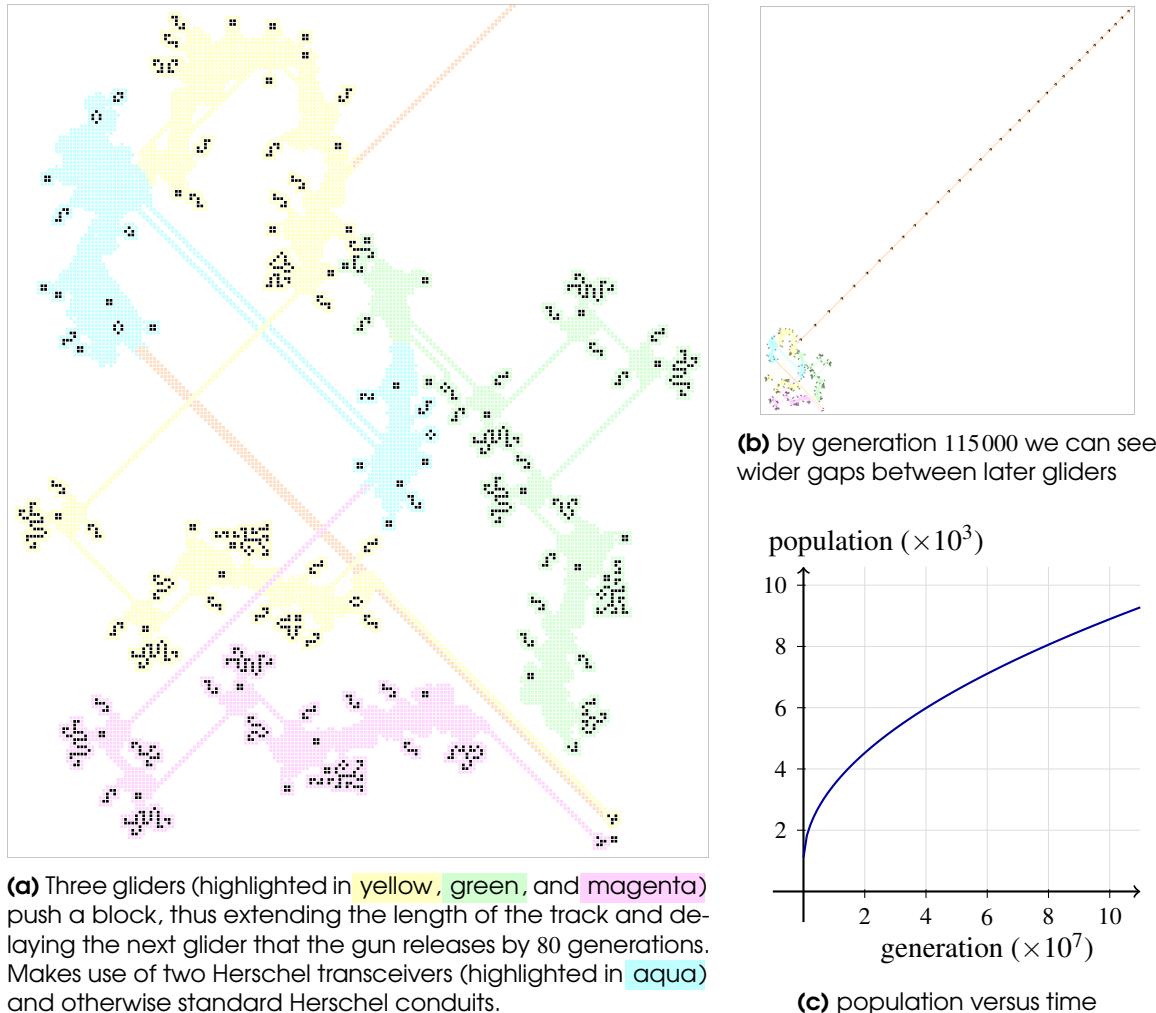


Figure 8.40: A *sqrtgun* that fires gliders slower and slower. Its population in generation t is asymptotically proportional to \sqrt{t} . Note that in (b), the spacings between objects are to scale, but the objects themselves have been magnified (the bounding box at that point is roughly $30,000 \times 30,000$, so the objects are too small to see).

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.

Such guns are called *caber tossers*, and the simplest ones to construct make use of a single glider bouncing off of the back of a Cordership (after all, we already saw how to bounce gliders

³¹The first *sqrtgun* was constructed by Dean Hickerson sometime around 1991.

off of the back of Corderships in Figure 4.24 and Exercise 4.19, 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,³² thus letting a single glider from each stream escape and repeat the process. One configuration that works is displayed in Figure 8.41.³³

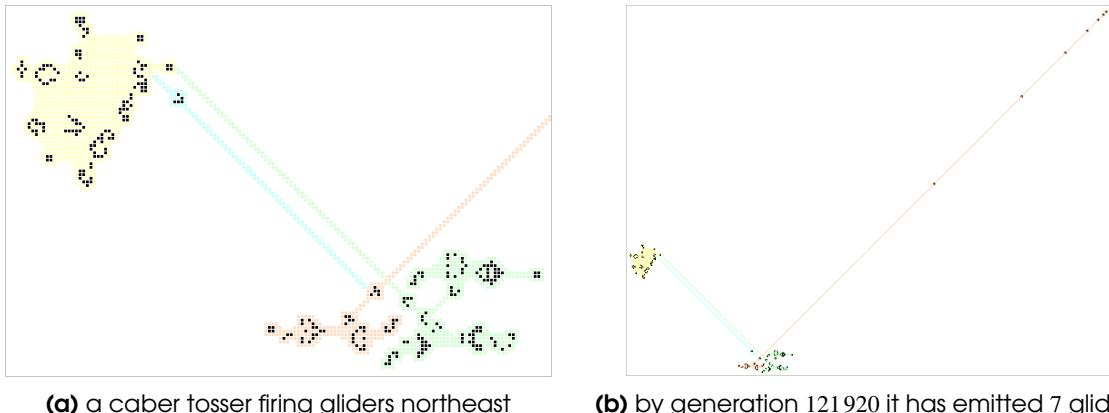


Figure 8.41: A *cabер tosser* with logarithmic growth rate. A period 60 glider gun (highlighted in green) fires gliders at a 2-engine Cordership (highlighted in yellow) to be reflected back, but is blocked by a period 30 glider gun (highlighted in orange). When the triggering glider (highlighted in aqua) blocks this stream, it allows a single glider through from both guns and thus 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.

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 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 asymptotically proportional to $\log(t)$).³⁴

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.

One such device is displayed in Figure 8.42.³⁵ This object is called a *recursive filter*, and when it receives a glider from whatever gun is attached to its input³⁶ (typically a slow gun like a caber tosser,

³²This configuration can easily be found by hand—recall that roughly one third of all two-glider collisions result in the gliders destroying each other, and those are what we want here.

³³The first caber tosser was built by Dean Hickerson in May 1991 using this same arrangement of guns, but it used a slightly different glider reflection involving the larger 13-engine Cordership instead of the 2-engine Cordership.

³⁴All logarithms that we consider are base-2 (however, it does not matter what base we use in this particular case).

³⁵Constructed by Alexey Nigin in July 2015.

³⁶The input glider must be aligned to period 120. The caber tosser from Figure 8.41 thus works fine for generating input to this recursive filter, but if we wanted to use the sqrtgun from Figure 8.40, for example, then we would have to insert a universal regulator in between (see Exercise ??).

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.³⁷
- 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.

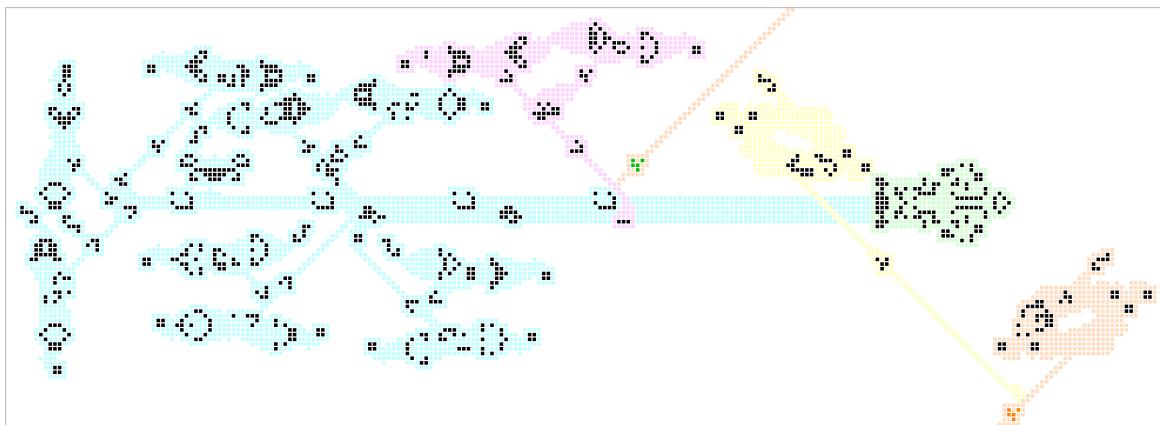


Figure 8.42: A recursive filter that receives input gliders from a gun that is aligned to p120 and emits output gliders much more slowly. The guns highlighted in aqua on the left create a stream of LWSS pairs that are blocked by the gun central gun highlighted in magenta, and similarly the p120 gun highlighted in yellow blocks the output gun at the southeast. When an input glider is received from the northeast, it blocks the magenta gun, thus letting a 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.

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 $480 \times 2^{n+1} - 727$ 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, it takes about 500 generations for the first glider from the caber tosser of Figure 8.41 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

$$480 \times 2^{104+1} - 727 \approx 1.947 \times 10^{34}.$$

Indeed, we see in Figure 8.43 that after 1920 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

³⁷This loaf-pulling reaction is called a *tractor beam*, and it can be used to create many other patterns with strange growth rates—see Exercise ??.

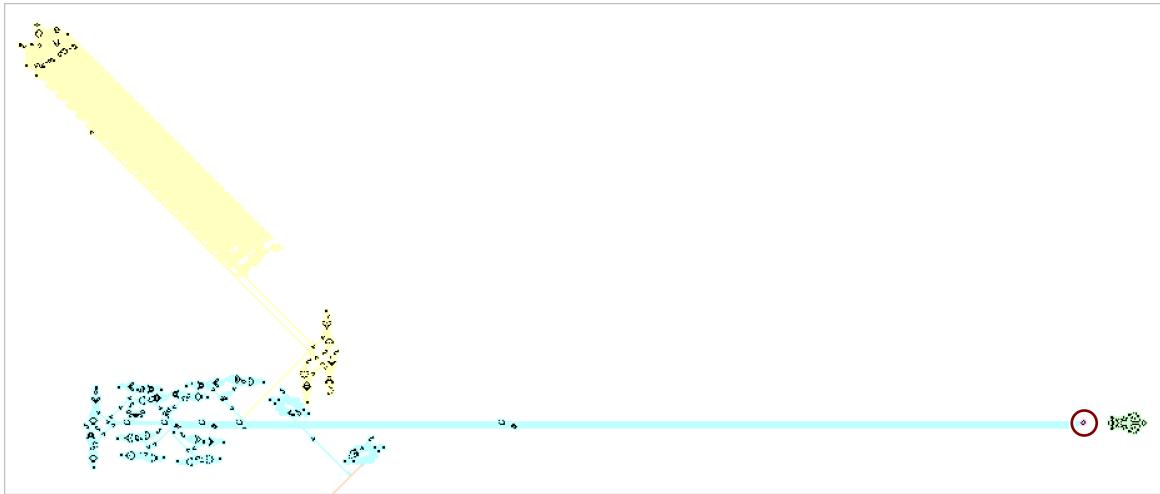


Figure 8.43: If we use a caber tosser (highlighted in yellow) as input to a recursive filter (highlighted in aqua), the result is a gun that grows extraordinarily slowly. The first glider from the caber tosser creates a loaf (circled in red) 624 cells away from the recursive filter, thus requiring $624/6 = 104$ 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.

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 $\Theta(2^{2n})$ generations. Indeed, the pattern in Figure 8.43 doesn't release its second output glider until approximately generation

$$480 \times 2^{19471113219505603606989361234575175/6+1},$$

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, we see that the third output glider takes $\Theta(2^{2^{2n}})$ generations to produce, and in general the k -th output glider takes

$$\Theta\left(\underbrace{2^{2^{2^n}}}_{k \text{ levels}}\right)$$

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 \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 1.³⁸

$$\log^*(t) = \begin{cases} 0 & \text{if } t \leq 1, \\ 1 + \log^*(\log(t)) & \text{if } t > 1. \end{cases}$$

As if that weren't slowly-growing enough, we could even use recursive filters in sequence 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 8.43 (see Exercise ??), we get a pattern whose population in

³⁸Equivalently, $\log^*(t)$ 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 .

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 1:³⁹

$$\log^{**}(t) = \begin{cases} 0 & \text{if } t \leq 1, \\ 1 + \log^{**}(\log^*(t)) & \text{if } t > 1. \end{cases}$$

This gun won't emit even a single glider until somewhere around generation $2^{2^{33}}$, 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 distinguish between these two possibilities.

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 6.20 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 65537, corresponding to $n = 1, 2, 4, 8$, and 16,⁴⁰ 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 8.33).

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.⁴¹ 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 further.

One mild hiccup that arises when constructing such a pattern is that 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 Section 4.5 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 $c/2$. The resulting pattern, displayed in Figure 8.44, makes use of the $4c/5$ fuse from Exercise 4.28 and the beehive puffer/wickstretcher from Exercise 4.29. It 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 65537 each destroy a single tub near the southwest corner of the pattern. If a sixth Fermat prime is ever computed, 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

$$120 \times 2^{2^{33}} \approx 1.16 \times 10^{2,585,827,975},$$

³⁹To give a rough idea of how slowly this function grows, we note that if t is an integer with a trillion digits then $\log^{**}(t) = 4$.

⁴⁰It is known that if $2^n + 1$ is prime then n must be a power of 2.

⁴¹We could similarly use the twin prime calculator (instead of a Fermat prime calculator) that we constructed in Exercise 6.6. However, there are currently hundreds of trillions of twin prime pairs known, so it would not be easy to make it self-destruct only when a new one is found.

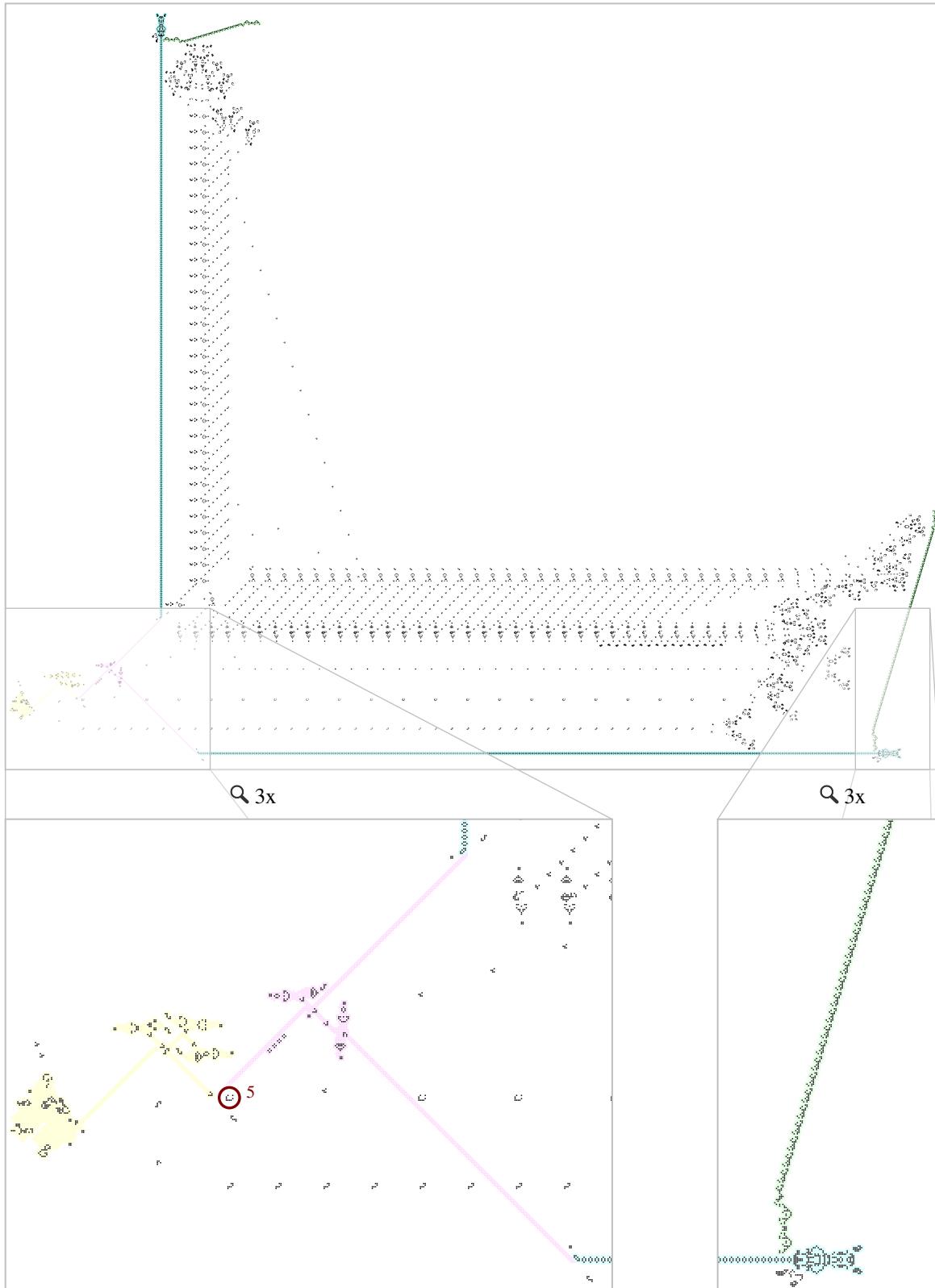


Figure 8.44: A Fermat prime calculator that self-destructs and has bounded population if a sixth Fermat prime exists, but grows without bound otherwise. A caber tosser (highlighted in yellow) fires gliders at the lightweight spaceships that the primer creates corresponding to integers of the form $2^n + 1$ (the LWSS corresponding to the Fermat prime 5 is circled in red). 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 65537). 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.

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 65537) and the primer finally stabilizes after roughly 21 million generations.

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 1970’s and were used to create the same period 60 gun that we constructed in Figure 8.2(a)—see Figure 8.45. 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 that were known of by 1971. Bill Gosper made a p1100 gun sometime around 1984, Dean Hickerson constructed a p94 gun in 1990, Bill Gosper made a p144 gun in 1994 (see Exercise 7.29), and by the early 1990’s David Buckingham had constructed guns of period 44 and $136 + 8n$ ($n \geq 1$), mostly based on his early work with Herschel tracks. 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.

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 explicitly-constructed glider guns of all periods from 14 to 1 000. 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).

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 8.15. 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, the current smallest known p16 gun (measured according to bounding box, which is 158×158) is displayed in Figure 8.46. This gun uses a small p24 glider gun (one that is slightly smaller and newer than the one we saw in Figure 8.19(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).

This collection of small glider guns is now being maintained by Chris Cain and Dave Greene. Links to the up-to-date version of the collection can be found at conwaylife.com/wiki/Dieter_and_Peter's_glider_gun_collection.

Exercises

solutions on page 311

- 8.1** 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.

- 8.2** 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.

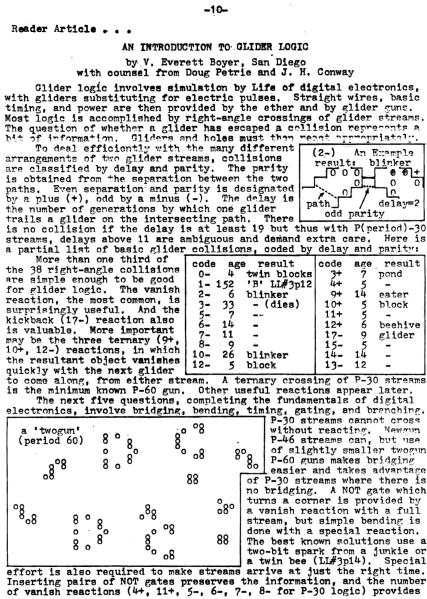


Figure 8.45: A page from the September 1973 issue (Volume 11) of *Lifeline* that discussed glider logic and how it can be used to make a p60 glider gun (shown at the bottom left).

8.3 Use the reaction from Figure 8.5(b) to create a period 322 glider gun.

8.4 The stream inverter/duplicator from Section 6.7 is quite useful if we want to duplicate a glider stream that took a lot of effort to create.

- (a) Use three (not four!) Gosper glider guns and a single extra block to create a gun that fires two p60 streams of gliders.
- (b) Use four (not six!) Gosper glider guns and two extra blocks to create a gun that fires three p60 streams of gliders.

8.5 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 We saw several filters that can double the period of every stream of lightweight spaceships with period $4n + 2$. Explain why there cannot exist a filter capable of doubling the period of every stream of *gliders* with period $4n + 2$.

8.7 Create a double-barrelled true-period 44 glider gun (i.e., a gun that produces two streams of gliders).

8.8 The true-period 50 glider gun in Figure 8.22 uses two copies of the central pi-heptomino reaction. Create a true-period 50 glider gun that uses three copies of that reaction.

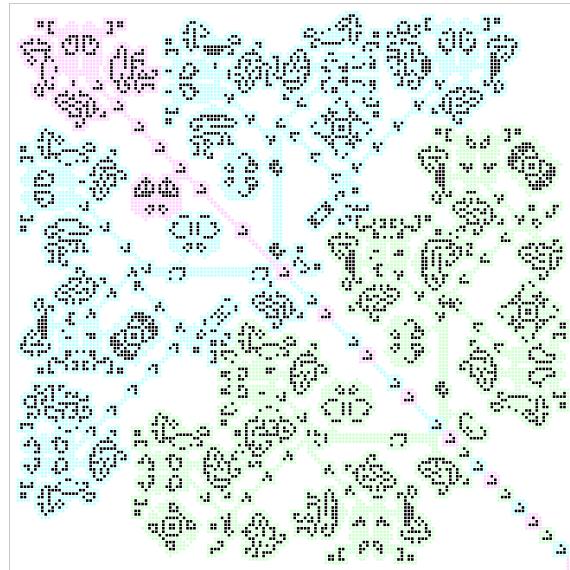
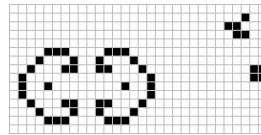


Figure 8.46: A tightly-packed p16 glider gun that uses a p48 gun (highlighted in magenta) and two p48 glider insertions (highlighted in aqua and green).

8.9 Another method of stabilizing one side of the p50 pi-heptomino reaction used in the true-period 50 glider gun is displayed below.



Use this reaction to create a true-period 50 glider gun with a lower population than the one displayed in Figure 8.22. 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 known bumper period that works with period 50 mechanisms?).

8.10 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 7.11(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.

8.11 Adjust the reflectors in the gun from Figure ??(a) to create a gun that emits 2 out of 3 gliders in a p29 (instead of p28) stream.

8.12 Adjust the reflectors reaction from Figure ??(b) to make it work with Herschel streams of period 180 (instead of 156).

8.13 Show how to stabilize Jason's p36 (see Figure 8.17(a)) with caterers instead of jams.

8.14 Show how to extract a glider from an Fx77 conduit like we did in Figure 8.25 via the following oscillators:

- (a) a unix (p6),
- (b) a pentadecathlon (p15), and
- (c) Beluchenko's p7 (Figure 3.13(c)).

8.15 The p56 glider gun displayed in Figure 8.28(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.

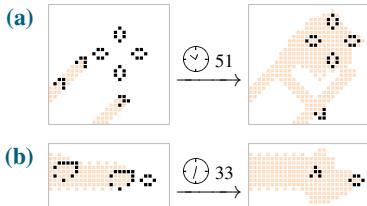
- (a) Modify the p54 glider gun from Figure 8.28(c) so that its bounding box has at least 300 fewer cells in it.
- (b) Modify the p55 glider gun from Figure 8.28(d) so that its bounding box has at least 700 fewer cells in it.

8.16 Determine the repeat times of the four conduits from Figure 8.29.

8.17 Explain why the method of separating the two close glider streams in the p61 gun from Figure 8.30 (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.

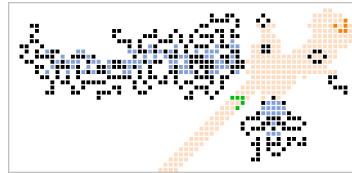
8.18 Use the reaction from Figure 8.34(d) to create a slide gun that fires gliders separated by 2 lanes (instead of 4 lanes, like the slide gun from Figure 8.35).

8.19 Slide guns can be made to work by pushing any object, not just a block. Use the following reactions⁴² to create slide guns:



8.20 Use the small p120 glider gun from Exercise 7.19 to rebuild the slide gun from Figure 8.35 within an 85×70 bounding box.

8.21 The device below⁴³ can be attached to the output of any gun with period $5n$ ($n \geq 2$) to create an edge-shooting gun of the same period:



- (a) Use this reaction to create an edge-shooting p240 gun.
- (b) Use 10 copies of the gun from part (a) (and perhaps the p120 glider gun from Exercise 7.19) to rebuild the slide gun from Figure 8.36 in a much smaller way. In particular, this edge shooter lets you avoid using the glider insertion mechanism from Figure 8.8(b).

8.22 Create a slide gun that fires gliders separated by just 1 lane.

[Hint: This is tricky! The reactions like Figure 8.34(d) that move a block 1 cell diagonally result in slide guns with a separation of 2 lanes.]

8.23 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.

8.24 Create a slide gun that fires 3 gliders on each of its output lanes (instead of 2, like in Figure 8.36).

[Hint: Using an edge-shooting gun like the one from Exercise 8.21 will help keep the size of your slide gun down.]

8.25 "Armless" glider production is a useful and highly adjustable mechanism, but some adjustments have consequences that may not be intuitively obvious.

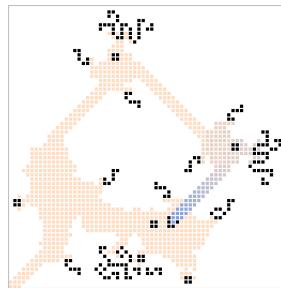
- (a) In a pair of loop guns that are connected to create an armless salvo, if two glider streams are identical in length and timing, what limitation does that place on the output salvo?
- (b) What behavior appears if you try to make output gliders with two loop guns with periods that are *different* by a multiple of 8?
- (c) Find a design for a "drifting armless-tee" gun that would allow multiple cycles of the above behavior but avoid the resulting problem.

⁴²Both found by Jason Summers in 1999.

⁴³Found by Arie Paap and Tanner Jacobi in late 2018 in a period $30n$ ($n \geq 2$) form, and adjusted to the form shown here in late 2019.

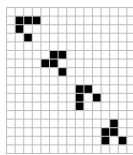
8.26 Tee reactions can be used to build simple unidirectional slow salvos, or even synchronized salvos.

- (a) Use the idea from Section 8.7 (separate loops for different glider colors) with the following period-387 loop gun ...



to build an armless 4-consecutive-lane shotgun like the one from Figure 8.38, but with period 387.

- (b) How can the shotgun from (a) be modified to allow the gliders to be placed at the minimum possible distance shown below? Create a new version of the shotgun that produces these four gliders at period 387.



[Hint: Even though the first and third gliders are the same color, they can't be stored in the same loop because they're too close together.]

- (c) Adjust the shotgun from (b) to build an armless 4-consecutive-lane gun where the entire cycle repeats with period 129. [Hint: $129 \times 3 = 387$, so add two more gliders to each of the p387 loops in the gun from (b).]
 (d) Adjust the shotgun from (b) to build an armless 4-consecutive-lane gun with the minimum repeat time of 87 ticks. [Hint: $387 + 6 \times 8 = 435 = 87 \times 5$, so first adjust each loop to be 6fd bigger.]
 (e) In a shotgun with one pair of loop guns per glider color, why can recipes consisting of gliders with alternate colors be stored more efficiently than consecutive gliders on the same colors?
 (f) The "armless-tee" Cordership gun (Figure ??) has already been adjusted to close to its recipe's minimum repeat time of 2061 ticks. What could be added to this gun to allow a repeat time of less than 2000 ticks?
 (g) How could the repeat time be reduced *without* increasing the number of loop guns?

8.27 In an "armless-tee" gun that produces pairs of colliding gliders, 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) Find a case where a glider follows another glider by 74 ticks in the "armless-tee" Cordership gun. Then find the other three gliders that are synchronized with that glider, and delay all four gliders by one tick. What happens?
 (b) Return the four gliders to their original positions, and instead of delaying all four gliders, advance them all by one tick. What happens?
 (c) 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?

8.28 With fewer separate loops in a multicolor armless design, changing the period of the gun becomes somewhat simpler. (Review)

- (a) Adjust one of the loop guns in the multicolor-armless Cordership gun, to make it an even period. How does the period change when a second syringe-based splitter (syringe+NW31) is added to the loop? [Hint: Start by moving the outermost two Snarks a long distance away to make space.]
 (b) How many of these syringe+NW31s would have to be added to change the loop gun to a "HashLife-friendly" period – e.g., period $2^{12} = 4,096$?
 (c) Find an alternate reflector in the Stable Circuitry chapter that changes the period (mod 8) by 5, with the same repeat time. How many of these must be added to allow for a p4096 loop?
 (d) Optionally, make the exact same addition to the other three loop guns to produce a working p4096 Cordership gun.

8.29 Explain why we cannot rewind the caber tosser from Figure 8.41(a) so that the triggering glider in aqua is traveling northwest before it is reflected by the caber tosser.

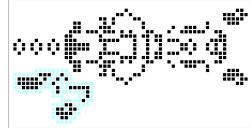
8.30 Construct a caber tosser that uses the same Cordership and glider bouncing off of it as in Figure 8.41(a), but uses stationary circuitry (e.g., syringes and Herschel conduits) in place of the glider guns.

8.31 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 8.41(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).

8.32 Rebuild the sqrtgun from Figure 8.40 without any Herschel tranceivers—use syringes (and other “modern” Herschel conduits) instead.

8.33 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 8.44.

8.34 The beehive puffer that is used at the bottom-right and top-left of the Fermat prime calculator (see Figure 8.44) is displayed below.



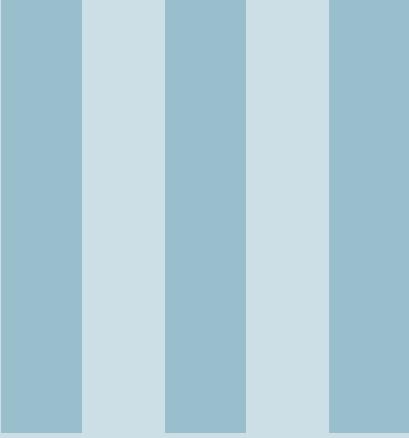
This puffer still works even if the cells highlighted in aqua are deleted. What is their purpose in the Fermat prime calculator?

8.35 Recall the small p16 oscillator from Figure 3.43 called Rob's p16.

- (a) Show how Rob's p16 can be used to filter a p24 LWSS stream, creating a p48 stream.
- (b) What LWSS stream periods can be filtered by Rob's p16?
- (c) Use the filtering reaction from part (a) to reduce the bounding box of the p16 glider gun that we saw in Figure 8.46 by at least 2 rows or columns.

Early draft (May 19, 2020).

Not for public dissemination.



Constructions

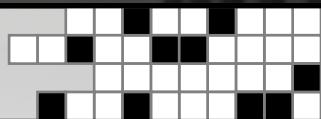
9	Universal Computation	243
9.1	A Computer in Life	
9.2	A Compiled APGsembly Pattern: Adding Registers	
9.3	Multiplying and Re-Using Registers	
9.4	A Binary Register	
9.5	A Character Printer	
9.6	A Pi Calculator	
9.7	A 2D Printer	
	Exercises	
10	Self-Supporting Spaceships	279
10.1	Caterpillar	
	Exercises	
11	Universal Construction	281
11.1	Demonoids	
	Exercises	
12	The OEOP Metacell	283
12.1	Rule Emulation	
12.2	Structure of the Metacell	
12.3	Memory Tape	
12.4	Logic Circuitry	
12.5	Construction	
12.6	Encoding Other Cellular Automata in Life	
12.7	Notes and Historical Remarks	
	Exercises	

Early draft (May 19, 2020).

Not for public dissemination.



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”) [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 toolkit¹ that can be used to construct patterns to perform arbitrary computations, and even print the output of those computations in a font made up of blocks. For example, the culmination 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 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 simple steps, or “instructions”. Each step has a specific effect on data stored in memory. In turn, values stored in memory can affect the flow of

¹Developed by Adam P. Goucher in 2009 and 2010.

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

To construct our first usable computational logic component, we observe that we can store a non-negative integer in the position of a block that is moved forward and backward via the block-moving operations that we used to make slide guns (see Section 8.6). 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 8.33(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 8.33(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 the “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 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.

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 8.33(b), so as to pull the block one cell closer, and
- an INC input that produces the three gliders from Figure 8.33(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.

²We 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.

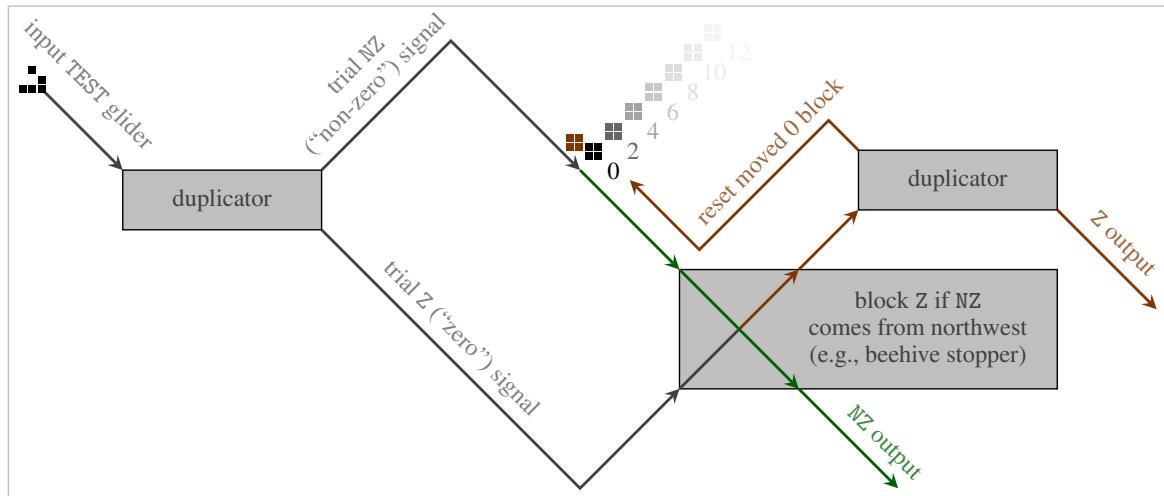


Figure 9.1: A schematic of a TEST circuit that tests whether or not a sliding block is in the “zero” (Z) position. If the block is in a non-zero (NZ) position then the trial NZ glider passes by without affecting it at all. Otherwise, the zero block is shifted via the $(2, 1)$ block pull reaction, the 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.

A sliding block register with these three inputs could certainly be used in a Life computer pattern, 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 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.

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). Our next goal is to figure out how to construct Life “program” patterns that make use of these circuits (sliding block registers) to perform actual calculations. But before delving into the details of how to actually 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. Pseudocode that implements this algorithm is provided by Pseudocode 9.1.

When we write pseudocode like this, we use labels like U_0 , U_1 , U_2 , ... to denote the various

³By 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.

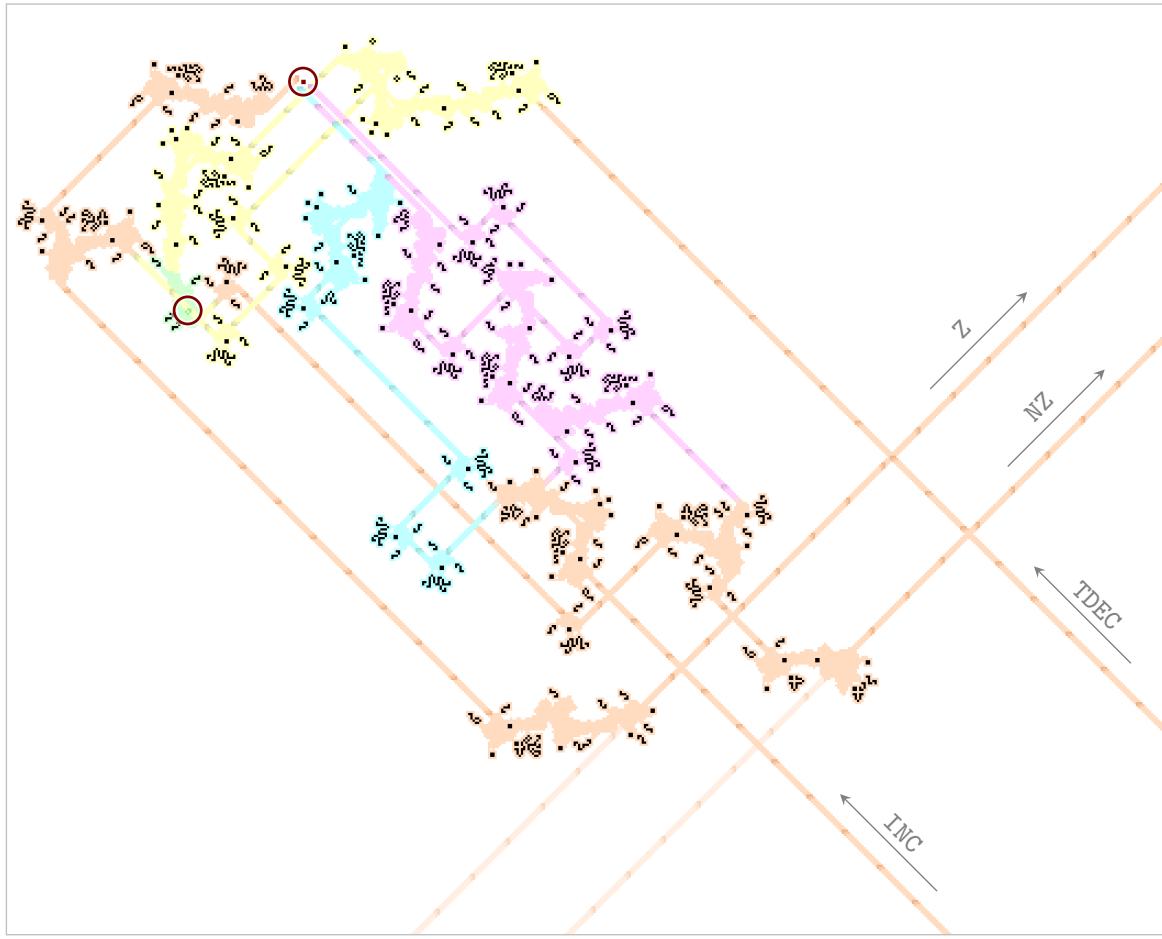


Figure 9.2: A sliding block register. If a glider enters on the INC lane, the aqua and magenta conduits split it into three gliders that push the sliding block (circled in red near the top) northwest by 1 cell. If a glider enters on the TDEC lane, it enters the TEST circuit (the first half of which is 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.

registers used by the computer program.⁴ Also, for this 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 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 $U_0 = 0$, jump to 6
 - 2: if $U_1 = 0$, jump to 8
 - 3: decrement U_0
 - 4: decrement U_1
 - 5: jump to 1
 - 6: OUTPUT "U0 <= U1"
 - 7: HALT
 - 8: OUTPUT "U1 < U0"
 - 9: HALT
-

⁴The ‘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.

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. For this reason, the first line of our pseudocode from now on will always perform some task (like a TDEC) that initializes a Z or NZ value, so that there’s always a “return” (i.e., most recent output) value available in the main program loop.

By making the changes outlined in points (1) and (2) above, our original Pseudocode 9.1 is transformed into the 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 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 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

⁵Our 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

⁶The 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.

⁷In 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.

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 action 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 small 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 indicated that $U0 \leq U1$, 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 we 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 and using IDs like this instead of line numbers makes it much easier to manage our code. Indeed, 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 this action can only print the digits 0–9 and periods (.),

⁸Also, 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.

⁹Similarly, 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 C.

not more complicated expressions like "U0<=U1". The HALT_OUT action halts the computer¹⁰ (presumably because we are done our desired computation) and sends out a glider (which could 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 one action that produces a return value. Indeed, if multiple actions produce a return value then it is not clear which substate should be jumped to next, and if no action produces a return value then no substate will be jumped to and the computer will stop running altogether. This is the reason that the ID1;Z and ID2;Z lines of APGsembly 9.1 contain the HALT_OUT lines: the OUTPUT action does not produce a return value, so we need to add another action to that line that *does*.¹¹

The only actions that produce return values that we have introduced so far are TDEC operations (which may return either a Z or an NZ) and HALT_OUT operations (whose return value is irrelevant). 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.¹² 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,¹³ and what the initial values of the registers should be, respectively.¹⁴

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

¹⁰For 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.

¹¹As 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.

¹²Links to the APGsembly compiler (and an emulator that can be used to help debug APGsembly programs) can be found at conwaylife.com/wiki/APGsembly.

¹³“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).

¹⁴If 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
```

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.

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 bumped back up to 0 automatically instead of staying at -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*. These are 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-traveling 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, which is simply 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

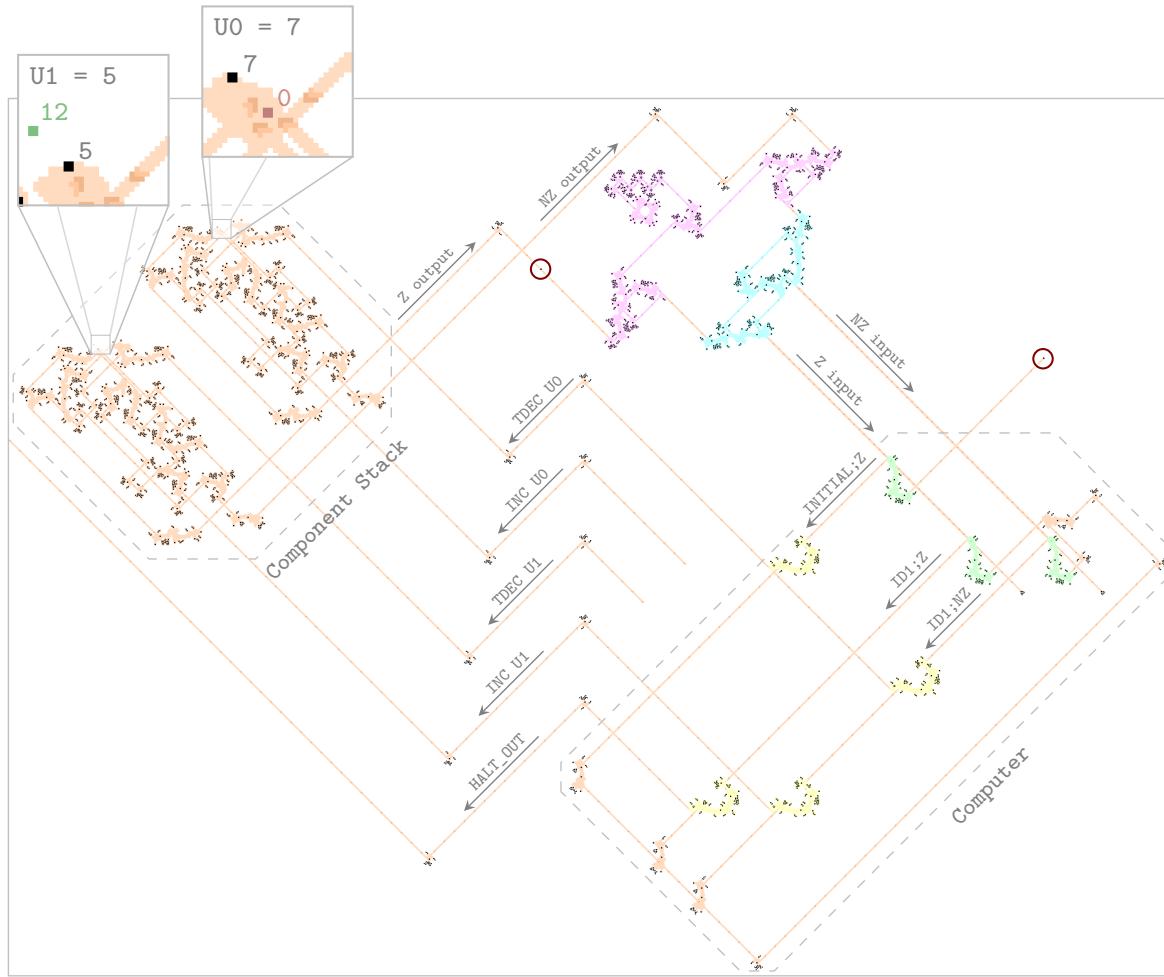


Figure 9.3: A compiled Life pattern that adds the values of the registers U_0 and U_1 (which have been pre-loaded with the values 7 and 5, respectively), as in APGsembly 9.2. The registers at the west feed their output into the period 2^{20} clock gun (highlighted in magenta with two separate universal 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 state 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 $INC\ U_0$ and $TDEC\ U_1$ lanes are disconnected from the computer because those actions are not called by any line of APGsembly). 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.

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 one 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.

¹⁵The computer in Figure 9.3 only has one path around its southeast corner along which gliders can be reflected back into the demultiplexers. This is just because every single line of APGsembly 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 at the southeast.

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 have a single Z/NZ output bit. The first example 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.

Other components may have more or fewer registers. 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 “SQ” component.

No matter how many components are added to the stack, they’re all called in the same way, by a single glider traveling 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. It is perfectly okay for a single line of code in APGsembly to trigger multiple different actions, even if they make use of the same component. It is the responsibility of the programmer to know when this can be done safely, and when this would cause the component’s circuitry to fail. Two incompatible actions have to be triggered from separate states (i.e., in separate computer clock ticks).

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 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 use¹⁸ 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 2²⁰, 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

¹⁶This 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).

¹⁷HashLife 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 re-used whenever that same square is encountered in the future.

¹⁸This universal regulator was constructed by ConwayLife.com forums user “Jormungant” in April 2020.

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 7.5; we attach period multipliers to another gun. This particular gun (displayed in Figure 9.5) uses quadri-Snarks (see Exercise 7.14) attached to a p256 machine gun, but semi-Snarks could have been used instead at the expense of the gun being slightly larger.

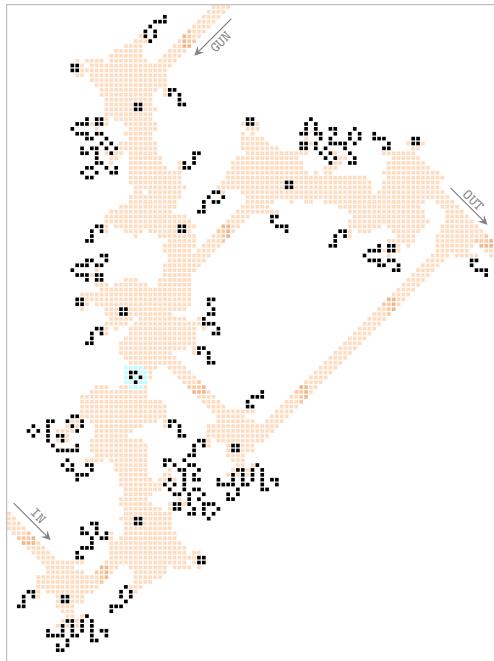


Figure 9.4: A stable universal regulator that works by having the input glider create a boat (highlighted in aqua), which suppresses the glider on the southeast path and allows the northeast glider to escape.

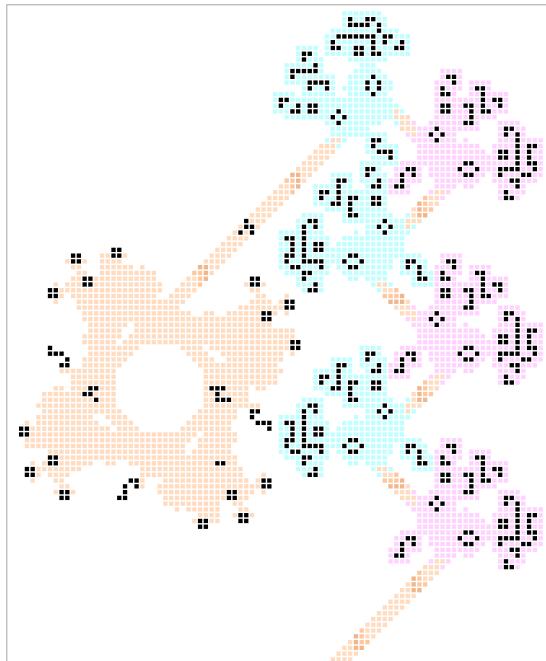


Figure 9.5: A period 2^{20} gun that works by using 6 quadri-Snarks (highlighted in aqua and magenta) to repeatedly quadruple the period of a p256 machine gun.

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 of the calculator and feeds into the component stack.
- The component stack keeps track of the calculator's memory and performs actions on that memory. It then feeds into 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 Re-Using 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. If we wish to multiply U_0 by U_1 and store the result in U_2 then we would like to simply repeatedly add the register U_1 to U_2 a total of U_0 times. However, there is a

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 halts the computer for one or more clock ticks until an output signal is received from the component stack.

slightly problem with this idea—our method of adding two registers from APGsembly 9.2 zeroes out one of the registers while adding it to the other. Indeed, the only way 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 $U_2 = U_0 * U_1$, every time we add U_1 to U_2 , we do so by first copying U_1 to a new temporary register U_3 (zeroing out U_1 in the process), and then looping over U_3 so as to add it to each of U_1 and U_2 . 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 $U_2 = U_0 * U_1$ and zero out U_0 . The register U_3 is used just temporarily (it starts and ends at the value of 0) to store the value of U_1 . After this computation completes, the registers will have the values $U_0 = 0$, $U_1 = 5$, $U_2 = 35$, and $U_3 = 0$.

#COMPONENTS U0-3,HALT_OUT			
#REGISTERS {"U0":7, "U1":5}			
# State	Input	Next state	Actions
INITIAL;	ZZ;	ID1;	TDEC U0
# Loop over U_0 , TDECing it until it hits 0, and then halt.			
ID1;	Z;	ID1;	HALT_OUT
ID1;	NZ;	ID2;	TDEC U1
# Copy U_1 into U_3 while setting $U_1 = 0$.			
ID2;	Z;	ID3;	TDEC U3
ID2;	NZ;	ID2;	TDEC U1, INC U3
# Loop over U_3 , adding its value to U_1 (restoring it) and U_2 .			
ID3;	Z;	ID1;	TDEC U0
ID3;	NZ;	ID3;	TDEC U3, INC U1, INC U2

A Life pattern that is 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 southeast end 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 U_0 / U_1 we repeatedly subtract U_1 from U_0 until $U_0 = 0$. The number of times that we were able to completely subtract U_1 is the integer part of U_0 / U_1 , and the value that was contained in U_0 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.

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 U_0+1 clock ticks to add the value of U_0 to U_1 , and the multiplication code of APGsembly 9.3 takes roughly $2 * U_0 * U_1$ clock ticks to multiply U_0 by U_1 .²¹

²⁰After this code is executed, the value of U_1 is preserved, but the value of U_0 is not. If we want to preserve the value of U_0 , we can use *another* temporary register (see Exercise 9.3).

²¹In fact, even the addition and subtraction by 1 operations take longer to complete the larger the value of the register is, since the sliding block takes longer to interact with when it is farther away.

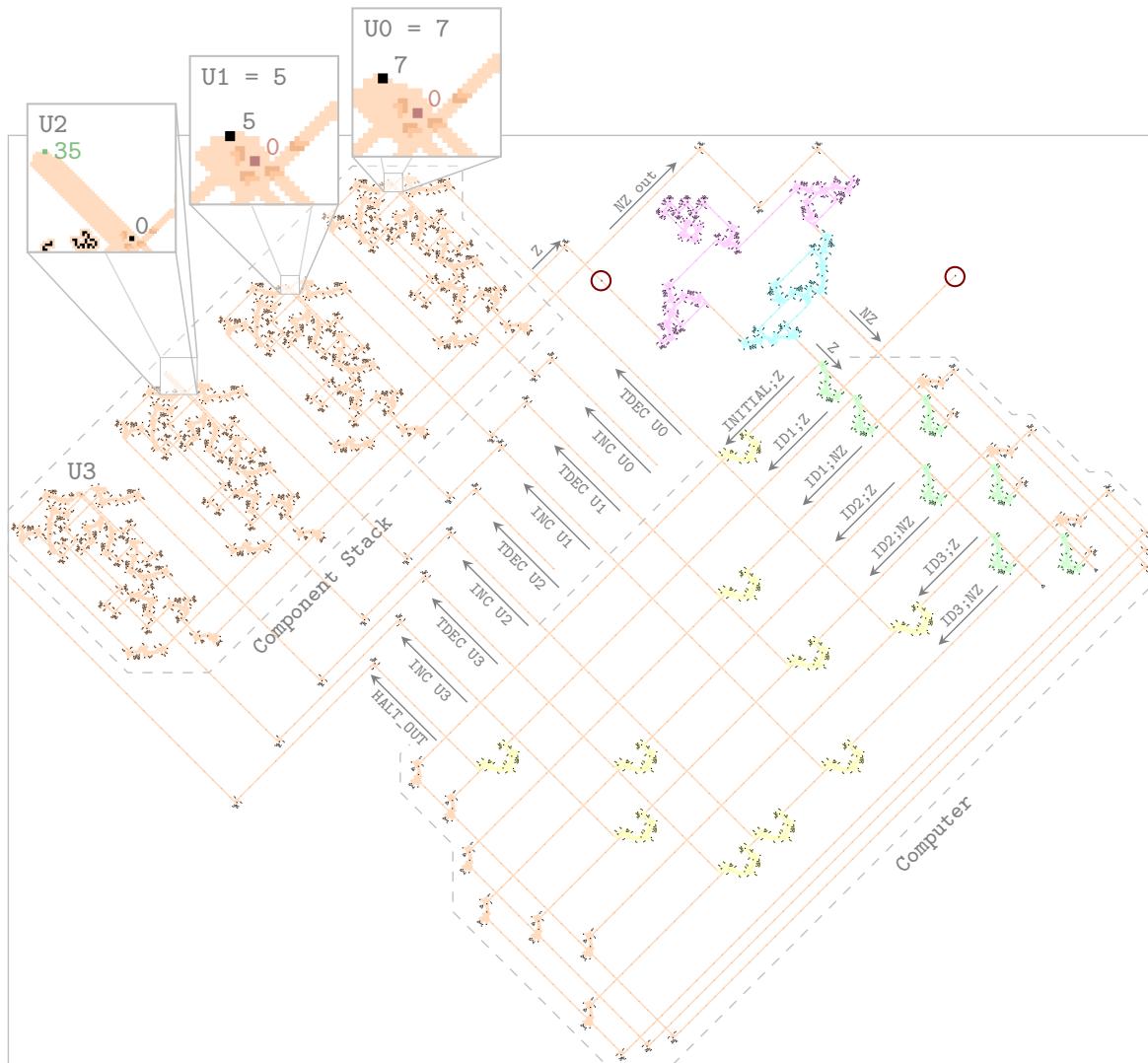


Figure 9.6: A compiled Life pattern that multiplies the values of the registers U_0 and U_1 (which have been pre-loaded with the values 7 and 5, respectively) and stores the result in U_2 , as in APGsembly 9.3. The color scheme and general structure of this pattern is the same as in Figure 9.3.

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 integers in *binary* (i.e., the base-2 numeral system).²² 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 $O(\log(n))$ space instead of $O(n)$ 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. Indeed, 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.

The four actions available to this binary register are a bit more abstract than the actions of the

²²Other bases larger than 2 would also be quicker to work with than unary, but they would be even more complicated to implement.

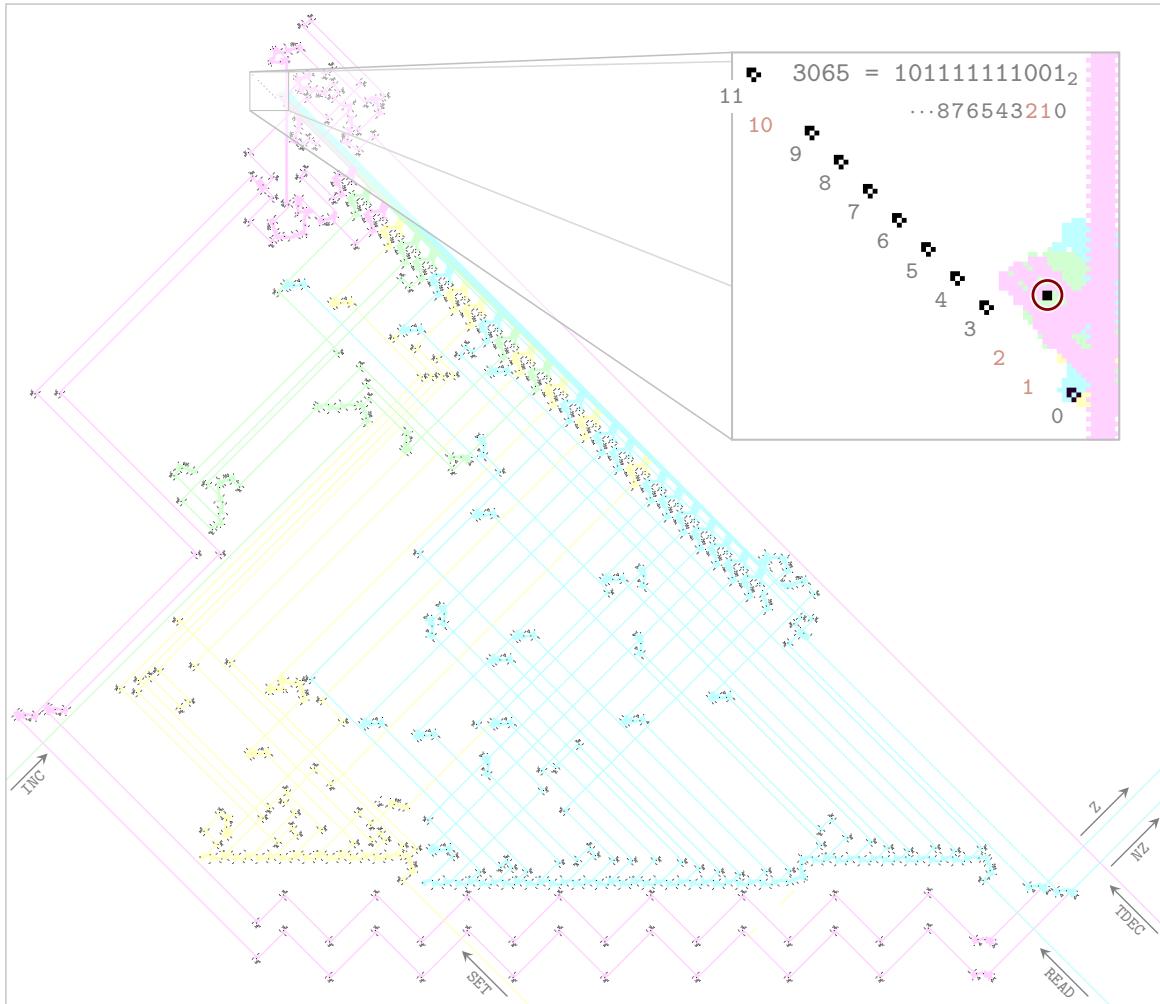


Figure 9.7: A binary register that uses a sliding block to keep track of the location of the read head (circled in red at the 0 location) and boats to represent bits (shown here storing the value $3065 = 10111111001_2$). The sliding block and boat bits are manipulated by four different slow salvos corresponding 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.

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 location, one step farther away.
- A TDEC action moves the read head to the previous bit location (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.²³
- 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

²³This 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 binary register from Figure 9.7 has a sliding block register baked into it for exactly this reason.

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 B_n to denote the n -th binary register (where we start counting at $n = 0$), just like we use U_n 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 B_0 .²⁴ 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 B_0 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 B_0 , 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 standard programming abbreviation, short for “NO OPeration”; 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.²⁵

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 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

²⁴The subscript “2” indicates that we are writing the number in binary.

²⁵It may be tempting to merge multiple lines of APGsembly ?? 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 (B_0) 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).

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 B_0 .

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 ??) is that it exhibits a new type of slow growth that we have not yet seen. We explored 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., longest bounding box side length) in generation t is $O(\log(t))$, since this is the rate at which new most significant bits are added to the end of the binary register.

This diametric growth rate is much slower than any other unbounded pattern that we have seen so far (all of which have been $O(t)$). 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 one bit to the next, possibly many times in a row.

In order to make arithmetic operations easier to perform, we introduce additional components that are custom-made for storing these carry bits. For example, the ADD and SUB components displayed in Figures 9.8 and 9.9, respectively, perform bitwise operations to assist in the addition and subtraction of binary registers.²⁶ 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

²⁶These 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.

component similarly performs the subtraction of two bits while keeping track of the borrow bit.²⁷

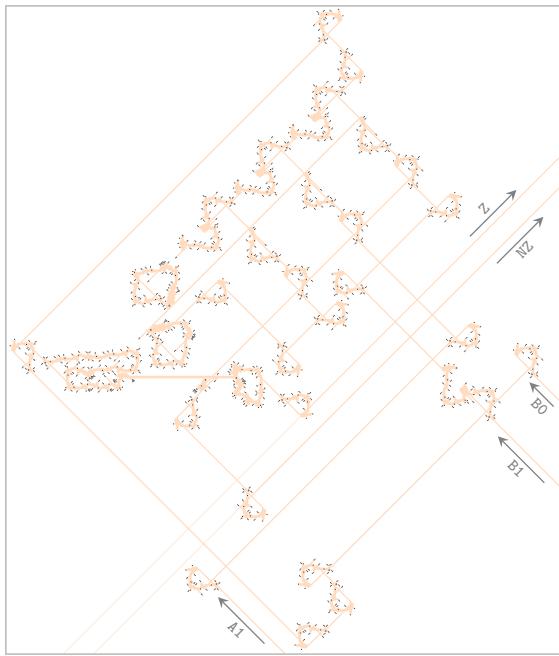


Figure 9.8: The ADD component. To add up two bits x and y while respecting carry bits, input $A1$ if $x = 1$ (provide no input if $x = 0$) and then $B1$.

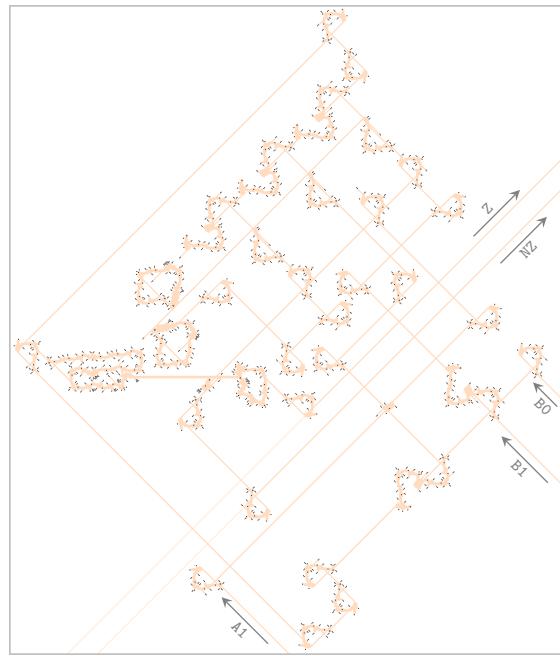


Figure 9.9: The SUB component. To subtract a bit y from x while respecting carry bits, input $A1$ if $x = 1$ (provide no input if $x = 0$) and then $B1$.

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 $(x + y + z) \bmod 2$ 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 and 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 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).²⁸ 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.²⁹

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

²⁷Unlike 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 one per Life pattern—the two types of registers are the only ones we can use multiple copies of.

²⁸Be careful: this means that B0 and B1 can be both components (as in INC B0 or READ B1) and actions (as in ADD B0).

²⁹This 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.

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,	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.³⁰ These actions would return the outputs

NZ,	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 BO-1,U0-2,ADD,NOP,HALT_OUT				#COMPONENTS BO-1,U0-2,SUB,NOP,HALT_OUT			
#REGISTERS {"BO": [0, '0001011'], "B1": [0, '100111'], "U0": 8}				#REGISTERS {"BO": [0, '0001011'], "B1": [0, '100111'], "U0": 8}			
# State	Input	Next state	Actions	# State	Input	Next state	Actions
# -----				# -----			
INITIAL;	ZZ;	ADD1;	TDEC U0	INITIAL;	ZZ;	SUB1;	TDEC U0
# Copy U0 into U2, with the help of U1				# Copy U0 into U2, with the help of U1			
ADD1;	Z;	ADD2;	TDEC U1	SUB1;	Z;	SUB2;	TDEC U1
ADD1;	NZ;	ADD1;	TDEC U0, INC U1	SUB1;	NZ;	SUB1;	TDEC U0, INC U1
ADD2;	Z;	ADD3;	TDEC U2	SUB2;	Z;	SUB3;	TDEC U2
ADD2;	NZ;	ADD2;	TDEC U1, INC U0, INC U2	SUB2;	NZ;	SUB2;	TDEC U1, INC U0, INC U2
# Loop over U2 to add B1 to B0, one bit at a time.				# Loop over U2 to subtract B1 from B0, one bit at a time.			
ADD3;	Z;	ADD7;	TDEC B0	SUB3;	Z;	SUB7;	TDEC B0
ADD3;	NZ;	ADD4;	READ B0	SUB3;	NZ;	SUB4;	READ B0
ADD4;	Z;	ADD5;	READ B1	SUB4;	Z;	SUB5;	READ B1
ADD4;	NZ;	ADD5;	READ B1, ADD A1	SUB4;	NZ;	SUB5;	READ B1, SUB A1
ADD5;	Z;	ADD6;	ADD B0	SUB5;	Z;	SUB6;	SUB B0
ADD5;	NZ;	ADD6;	ADD B1, SET B1	SUB5;	NZ;	SUB6;	SUB B1, SET B1
ADD6;	Z;	ADD3;	TDEC U2, INC B0, INC B1	SUB6;	Z;	SUB3;	TDEC U2, INC B0, INC B1
ADD6;	NZ;	ADD6;	SET B0, NOP	SUB6;	NZ;	SUB6;	SET B0, NOP
# Move the B0 and B1 read heads back to least significant bit.				# Move the B0 and B1 read heads back to least significant bit.			
ADD7;	Z;	ADD8;	TDEC B1	SUB7;	Z;	SUB8;	TDEC B1
ADD7;	NZ;	ADD7;	TDEC B0	SUB7;	NZ;	SUB7;	TDEC B0
ADD8;	Z;	ADD8;	HALT_OUT	SUB8;	Z;	SUB8;	HALT_OUT
ADD8;	NZ;	ADD8;	TDEC B1	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 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 $B1 \leq B0$; if you are unsure of which binary register is the smaller one then you should first compare them (see Exercise 9.8).

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 9.6). 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 10.³¹

³⁰It is also a good idea to call ADD B0 after adding two registers, since this clears the internal memory of the ADD component.

³¹The 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.

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.³² The MUL component is displayed in Figure 9.10, though it is not very enlightening to look at.³³

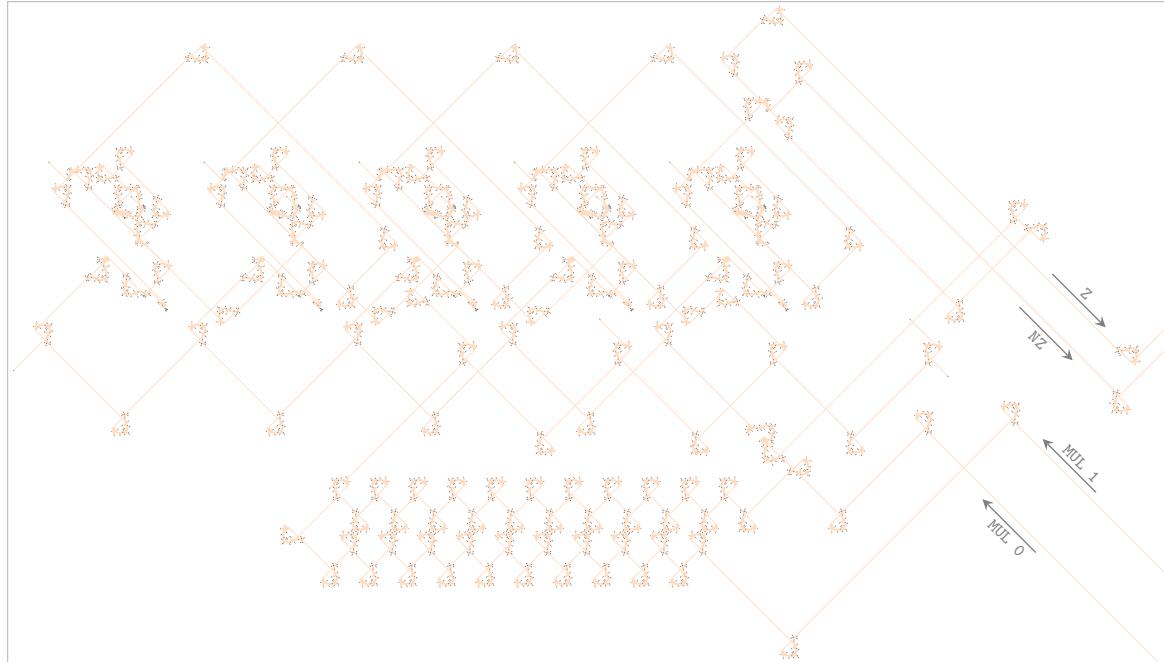


Figure 9.10: The MUL component, which keeps track of multiple carry bits so as to help us easily multiply a binary register by 10.

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 just as quickly and easily as we could add and subtract the values of binary registers. Complete APGsembly code for carrying out this multiplication is provided in APGsembly 9.7.

³²The 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.

³³Even 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.

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 demonstrate how to construct a component that is capable of printing any characters (e.g., digits or letters) of our choosing, by placing blocks in empty space at its current cursor position. Fortunately, we have already seen most of the reactions that we need to assemble a printer of this type. Very much along the lines of the unary sliding-block register and the binary tape register in previous sections, a printer needs to use a variant of slide gun technology to move some kind of placeholder object (a block) that marks the current cursor position. Furthermore, it needs to be able to optionally place other objects (more blocks) nearby to serve as the pixels in printed characters.

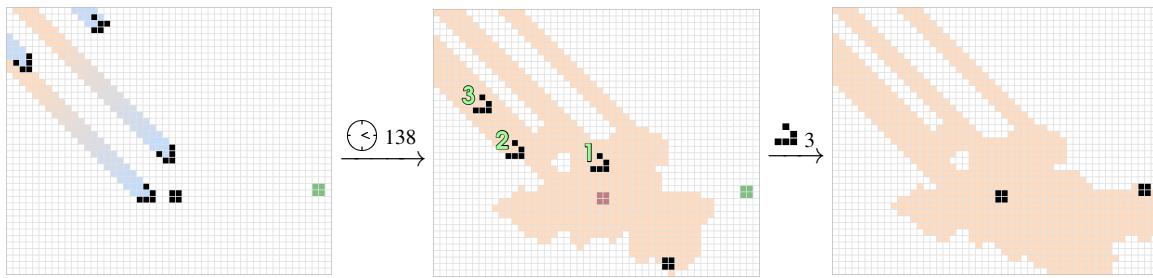
In order to make our printer somewhat simpler to construct, we will have it move and position blocks via slow salvos like the ones that we introduced back in Section 5.7. Indeed, doing so lets us not have to worry about the precise timing of the gliders that we use to move and duplicate blocks—we just have to position them correctly (which is trivial via an array of edge-shooting Herschel conduits³⁴). One particular salvo for placing a pixel block is displayed in Figure 9.11(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.

Similarly, Figure 9.11(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.

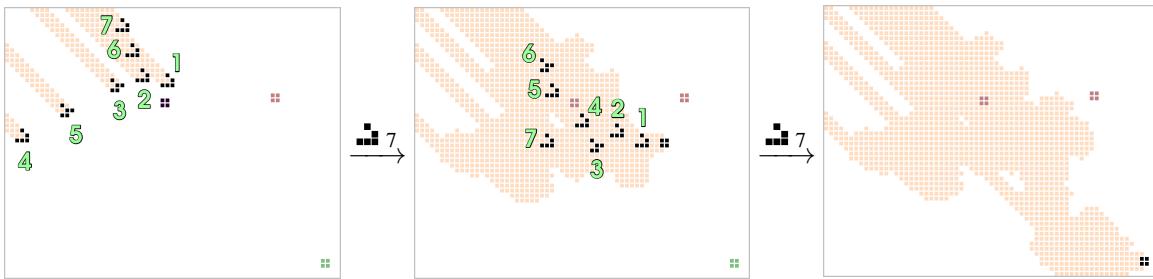
To construct our printer, we simply use one cursor block for each pixel high that the printed characters will be. For example, if we wish to print characters that are 15 pixels tall, we use 15 cursor blocks, separated from each other by 32 full diagonals, and we aim the pixel-placing and cursor-pushing glider salvos from Figure 9.11 at each of them.

To create those salvos in a particular row, we just use the stable circuitry like Snarks, syringes, and

³⁴This 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.



(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 9.11: Glider salvos that can be used to print block pixels at a separation of 32 full diagonals.

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 9.11(b). If we want to print the next pixel in the current row, we instead create the salvo from Figure 9.11(a) and then trigger the cursor-pushing mechanism. A device called a *row printer* that implements both of these operations is displayed in Figure 9.12.

The end result of using (many copies of) this row printer 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. Signals for all the rows propagate through the rectangular area that makes up the "lookup table" for the printer. When the circuit stabilizes again, the effect is that all the pixels encoded by the positions of merge circuits in the lookup table have been written out to the Life universe in the form of appropriately-spaced blocks.

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 TDECing 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 9.15.

9.6 A Pi 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 $\pi = 3.14159\dots$.

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:

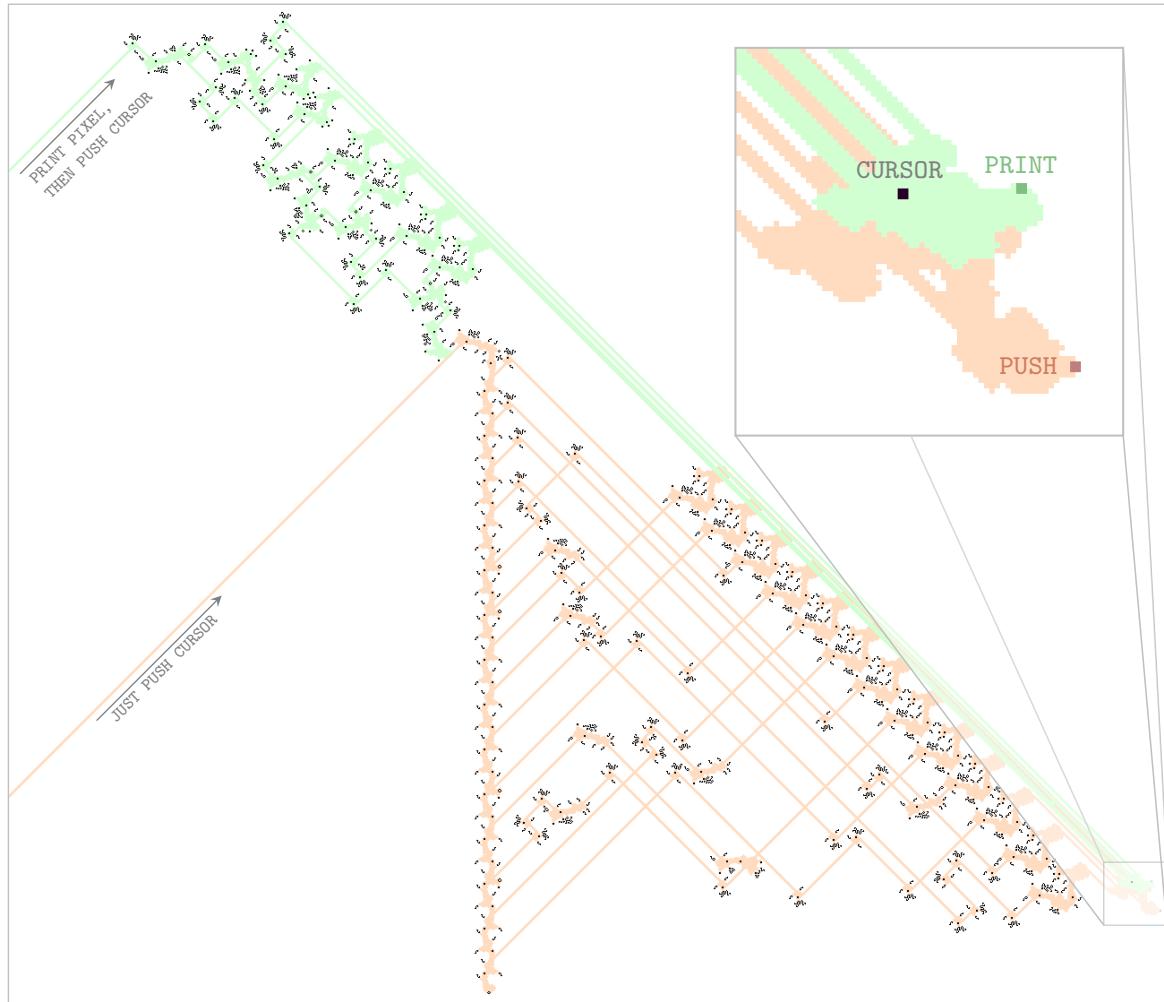


Figure 9.12: A *row printer* that can be used to either push a cursor block forward by 32 full diagonals (highlighted in orange), or print a pixel block (highlighted in green) and *then* push the cursor block.

- It makes use entirely of integer arithmetic, rather than floating-point arithmetic (i.e., arithmetic involving numbers that have no non-zero digits after the decimal point).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 series representation for π (see Exercise 9.10 for an explanation of where this series comes from):

$$\pi = 2 \left(1 + \frac{1}{3} \left(1 + \frac{2}{5} \left(1 + \frac{3}{7} \left(\dots \left(1 + \frac{k}{2k+1} (\dots) \right) \right) \right) \right) \right). \quad (9.1)$$

Indeed, if you truncate the above series after a finite number of additions, you get better and better

³⁵We 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.

APGsembly 9.8 APGsembly code for printing the contents of the sliding block register U0 (which is pre-loaded with a value of 7 here) 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

approximations of π . For example,

$$2 = 2.0000, \quad 2\left(1 + \frac{1}{3}\right) \approx 2.6667, \quad (9.2)$$

$$2\left(1 + \frac{1}{3}\left(1 + \frac{2}{5}\right)\right) \approx 2.9333, \quad 2\left(1 + \frac{1}{3}\left(1 + \frac{2}{5}\left(1 + \frac{3}{7}\right)\right)\right) \approx 3.0476,$$

and so on, with these terms getting closer and closer to $\pi \approx 3.14159\dots$.

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 it is reasonable to implement in APGsembly, we now introduce a slightly different way of computing these exact same approximations of π .

To this end, define the following 2×2 matrix that depends on a positive integer k :

$$A_0 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad A_k = \begin{bmatrix} k & 2k+1 \\ 0 & 2k+1 \end{bmatrix} \quad \text{for all } k \geq 1. \quad (9.3)$$

For example,

$$A_1 = \begin{bmatrix} 1 & 6 \\ 0 & 3 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 2 & 10 \\ 0 & 5 \end{bmatrix}, \quad A_3 = \begin{bmatrix} 3 & 14 \\ 0 & 7 \end{bmatrix}, \quad \text{and} \quad A_4 = \begin{bmatrix} 4 & 18 \\ 0 & 9 \end{bmatrix},$$

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_k s: $B_n =$

$A_0 A_1 A_2 \cdots A_n$.³⁶ For example,

$$B_1 = A_0 A_1 = \begin{bmatrix} 2 & 6 \\ 0 & 3 \end{bmatrix}, \quad B_2 = A_0 A_1 A_2 = \begin{bmatrix} 4 & 40 \\ 0 & 15 \end{bmatrix},$$

$$B_3 = A_0 A_1 A_2 A_3 = \begin{bmatrix} 12 & 308 \\ 0 & 105 \end{bmatrix}, \quad \text{and} \quad B_4 = A_0 A_1 A_2 A_3 A_4 = \begin{bmatrix} 48 & 2880 \\ 0 & 945 \end{bmatrix}.$$

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). For example,

$$\frac{q_1}{r_1} = \frac{6}{3} = 2.0000, \quad \frac{q_2}{r_2} = \frac{40}{15} \approx 2.6667,$$

$$\frac{q_3}{r_3} = \frac{308}{105} \approx 2.9333, \quad \text{and} \quad \frac{q_4}{r_4} = \frac{2880}{945} \approx 3.0476.$$

This gives us a way of computing better and better approximations of π using just integer arithmetic until 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}).

Since each term in the series (9.1) is approximately $1/2$ as large as the term that came before it, 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. For simplicity, we simply round this quantity up to 4 and thus note that if we want n decimal places of π , it suffices to use the approximation based on B_{4n} .³⁷ For example,

$$B_4 = \begin{bmatrix} 48 & 2880 \\ 0 & 945 \end{bmatrix}, \quad \frac{q_4}{r_4} = \frac{2880}{945} \approx 3.0476,$$

$$B_8 = \begin{bmatrix} 80640 & 108103680 \\ 0 & 34459425 \end{bmatrix}, \quad \frac{q_8}{r_8} = \frac{108103680}{34459425} \approx 3.1371,$$

$$B_{12} = \begin{bmatrix} 958003200 & 24835120128000 \\ 0 & 7905853580625 \end{bmatrix}, \quad \frac{q_{12}}{r_{12}} = \frac{24835120128000}{7905853580625} \approx 3.1414,$$

are approximations of π that are accurate to at least 1, 2, and 3 decimal places.

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 A_n matrix
- U1: top-right and bottom-right corners of 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 B_n matrix
- B1: top-right corner of B_n matrix ($= q_n$)
- B2: bottom-right corner of B_n matrix ($= r_n$)

³⁶Here we are using matrix multiplication, which works via the formula $\begin{bmatrix} a & b \\ 0 & c \end{bmatrix} = \begin{bmatrix} d & e \\ 0 & f \end{bmatrix} = \begin{bmatrix} ad & ae + bf \\ 0 & cf \end{bmatrix}$.

³⁷There is actually 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 -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 -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 31,000,000,000,000 decimal places of π are known, and this algorithm computes them all correctly.

Pseudocode 9.3 Pseudocode for computing and printing the decimal digits of π . If any variable is used before it is set, it is assumed to start at a value of 0.

```

1: # Initialize the A and B matrices.
2: set U1 = 1, B0 = 2, B2 = 1

3: # Never stop computing and printing digits.
4: loop forever:
5:   # Iterate 4 times before printing a digit.
6:   loop 4 times:
7:     # Update the A matrix.
8:     set U0 = U0 + 1, U1 = U1 + 2

9:     # Update the B matrix.
10:    set B0 = U0 * B0
11:    set B1 = U1 * (B0 + B1)
12:    set B2 = U1 * B2
13:  end loop

14:  # Compute and print the desired digit of B1 / B2.
15:  set U2 = the U3-th digit after the decimal point of B1 / B2
16:  OUTPUT U2

17:  # Print a decimal point after the 0-th digit (3), and loop.
18:  if U3 = 0 then OUTPUT .
19:  set U3 = U3 + 1
20: end loop

```

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. On the other hand, 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 4 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 9.6, respectively.

However, one piece of Pseudocode 9.3 that is tricky to implement is the digit extraction at line 15. In order to carry out this step using just integer arithmetic, we do the following variant of integer division:

- 1) 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, where s is the number of times that we have already performed step (1) ($s = 0$ the first time we perform this step). If this is the digit that we are currently interested in, we can stop. Otherwise, we proceed to step (2) below.
- 2) Multiply B1 by 10 and then return to step (1) above.

In order to implement step (1) 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 9.8. Similarly, we saw how to do the multiplication by 10 that step (2) 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 C.

The resulting calculator that is compiled from that APGsembly code is presented in Figure 9.13.

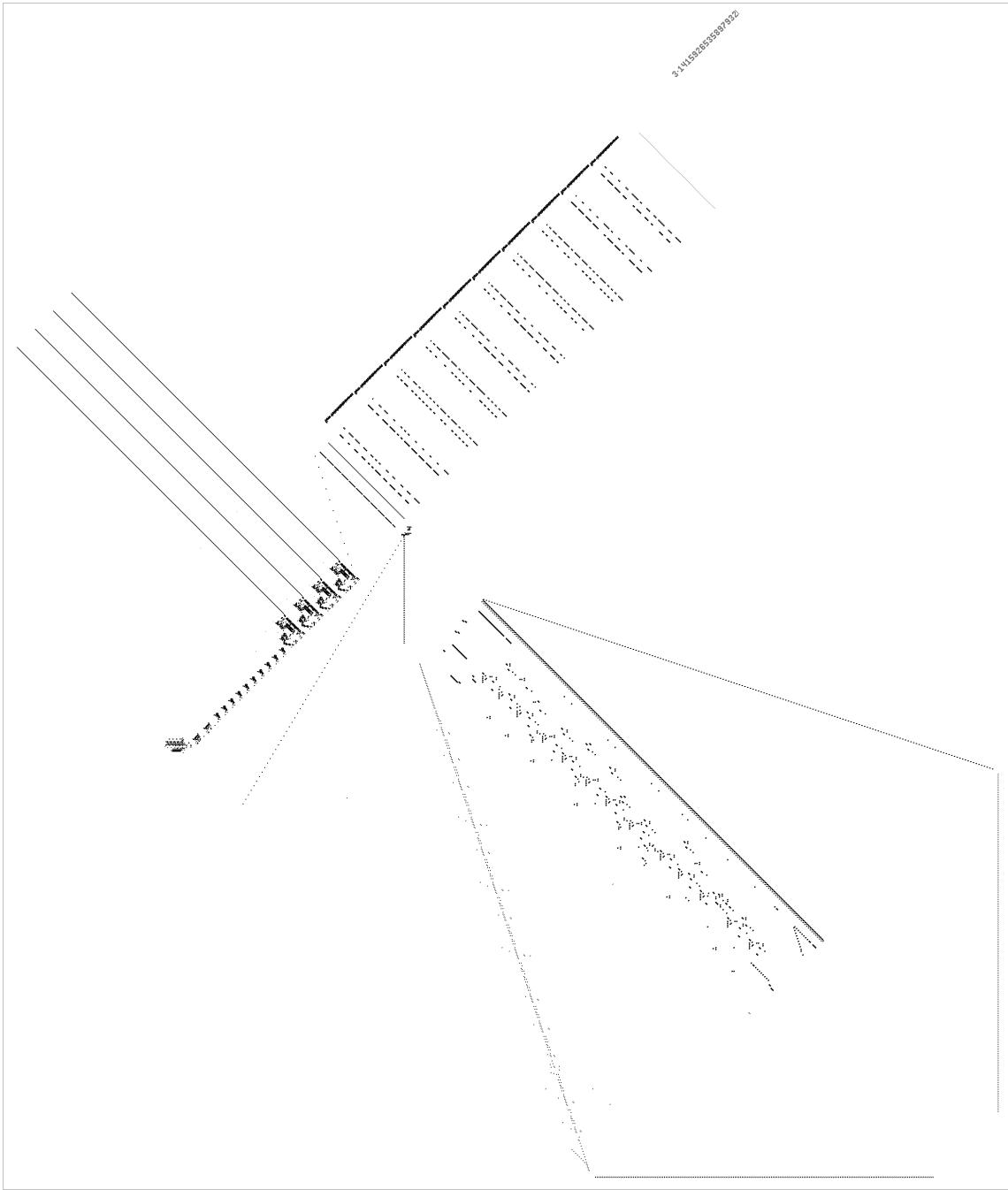


Figure 9.13: After about 168 trillion generations, the π calculator has computed and displayed the first 16 digits of π : 3.1415926535897932. Placeholder for now. Will add annotations and highlighting later.

9.6.1 Computing Other Constants

This same algorithm that we used in our π calculator can also be used to compute many other mathematical constants of interest. 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

$$\begin{aligned} & \frac{a}{b} \sum_{k=0}^{\infty} \frac{(p+q)(2p+q)\cdots(kp+q)}{(r+s)(2r+s)\cdots(kr+s)} \\ &= \frac{a}{b} \left(1 + \frac{p+q}{r+s} \left(1 + \frac{2p+q}{2r+s} \left(1 + \frac{3p+q}{3r+s} \left(\cdots \left(1 + \frac{kp+q}{kr+s} (\cdots) \right) \right) \right) \right) \right). \end{aligned} \quad (9.4)$$

For example, the π series (9.1) arises in the case when $a = 2$, $b = 1$, $p = 1$, $q = 0$, $r = 2$, and $s = 1$.

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

$$A_0 = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \quad \text{and} \quad A_k = \begin{bmatrix} kp+q & kr+s \\ 0 & kr+s \end{bmatrix} \quad \text{for all } k \geq 1.$$

The computation of the B_n matrices from the A_k s, 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 $e = 2.71828\dots$, which makes frequent appearances in calculus and statistics. It has the well-known series representation

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} = 1 + \left(1 + \frac{1}{2} \left(1 + \frac{1}{3} \left(1 + \frac{1}{4} (\cdots) \right) \right) \right), \quad (9.5)$$

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 $U0 = U0 + p$, $U1 = U1 + r$ ”.

Indeed, modifying the APGsembly code for the π calculator from Appendix C is similarly straightforward. It can be turned into APGsembly for an e calculator by just change 3 lines of code: the #REGISTERS line, ITER6;NZ line, and ITER7;ZZ line (see Exercise 9.11). Other constants like $\sqrt{2}$ can similarly be computed by making analogous minimal changes (see Exercise 9.12).

9.7 A 2D Printer

While the decimal printer 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

³⁸If $k = 0$ 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.

³⁹This interpretation explains the abbreviation B2D for this component; it stands for “binary 2-dimensional”.

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.

- 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. However, we will see it in each of the upcoming Figures ?? and 9.15. 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).

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 9.14, and is determined as follows:

- 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).

⁴⁰Just 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 pointer location.

Component	Actions	Functionality and return value
Un: sliding block register stores a non-negative integer in unary	INC Un TDEC Un	increases the value of the register by 1, no return value returns Z if Un = 0 and NZ otherwise, and <i>then</i> 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 <i>then</i> 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 <i>then</i> decreases X position by 1 if NZ
	TDEC B2DY	returns Z if Y read head is at least significant bit and NZ otherwise, and <i>then</i> 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	helps us add one binary register to another one, as in APGsembly 9.6
	ADD B0	
	ADD B1	
SUB: binary subtractor	SUB A1	helps us subtract one binary register from another one, as in APGsembly 9.6
	SUB B0	
	SUB B1	
MUL: binary multiplier	MUL 0	helps us quickly multiply a binary register by 10, as in APGsembly 9.7
	MUL 1	
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.

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

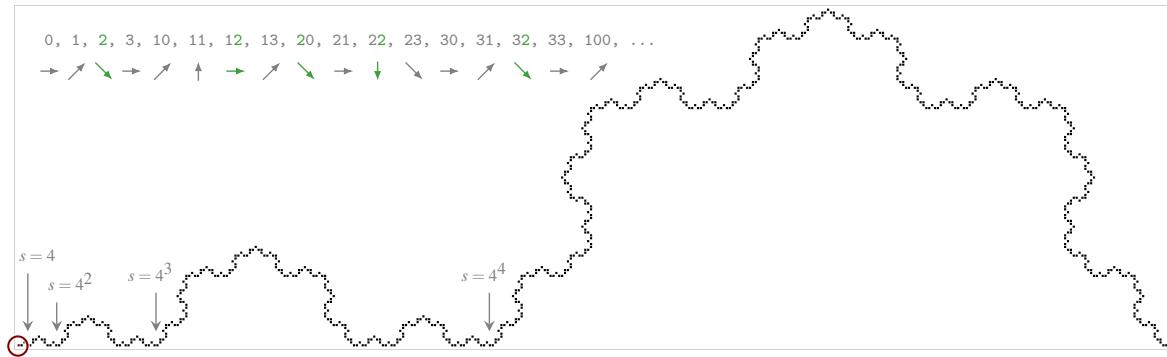


Figure 9.14: The (leftmost portion of the) image that results from starting at the bottom-left corner (circled in red) and printing pixels according to a path-following procedure based on counting in base 4 (in the variable s). These bulbous shapes are better and better approximations of one half of an 8-pointed version of the Koch snowflake fractal.

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 . Indeed, this fractal is one 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:

APGsembly 9.9 APGsembly code for the 8-pointed Koch snowflake printer.

<pre>#COMPONENTS U0-5,B0,B2D,NOP #REGISTERS {"U4":6, "U5":8} # State Input Next state Actions # ----- INITIAL; ZZ; DIRO; 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</pre>	<pre># Move B0 read head to least significant bit. RESETB0; Z; INCBOA; 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. INCBOA; Z; INCBOB; SET B0, NOP INCBOA; NZ; INCBOE; INC B0, NOP INCBOB; ZZ; INCBOC; INC B0, NOP INCBOC; ZZ; INCBOD; READ B0 INCBOB; Z; MOD8A; TDEC U5, INC U2 INCBOB; NZ; MOD8A; TDEC U5, INC U2, SET B0 INCBOE; ZZ; INCBOF; READ B0 INCBOF; Z; INCU2A; TDEC U4, SET B0 INCBOF; 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; DIRO; TDEC U2 RESET4; NZ; RESET4; TDEC U0</pre>
--	---

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 ($= s$)

The Life pattern⁴¹ that results from compiling this APGsembly is displayed in Figure ???. Since the 2D printer component prints on a diagonal grid, the image that is printed by this pattern is rotated counter-clockwise by 45 degrees from the one in Figure 9.14.

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 introduced 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 bounding box side lengths in generation t are both $\Theta(\sqrt{\log(t)})$.

The basic idea behind this slowly-growing pattern is the same as it was for the $O(\log(t))$ binary ruler from Section 9.4.1. We count in binary, except instead of placing the bits of the resulting number in a single 1D row via a (B) binary register, we arrange them in a 2D triangular array via a (B2D) 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 9.15. 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 this growth down even more by decreasing the speed of this pattern’s clock gun, but this will not change the growth rate’s asymptotics.

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 Turing machine could accomplish [BCG82]. But 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’s Turing machines in 2000–2011, Paul Chapman’s Minsky Register Machines in 2002,

⁴¹Originally constructed by Michael Simkin in 2019.

⁴²This contrasts with our results of Section 8.8.3: even though there is no limit on how slowly the population of a pattern can grow, there is a limit on how slowly its bounding box can grow.

⁴³The boats that the 2d printer prints are offset from each other by 16 full diagonals.

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

```

Adam P. Goucher’s Spartan universal computer-constructor in 2009 [Gou10] (which was the basis for almost all of the components that we investigated in this chapter), and Nicolas Loizeau’s 8-bit programmable computer in 2016.

Each of these computational devices used different Life mechanisms to implement logic gates, memory, program storage, and so on.

Exercises

solutions to starred exercises on page 312

9.1 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 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?

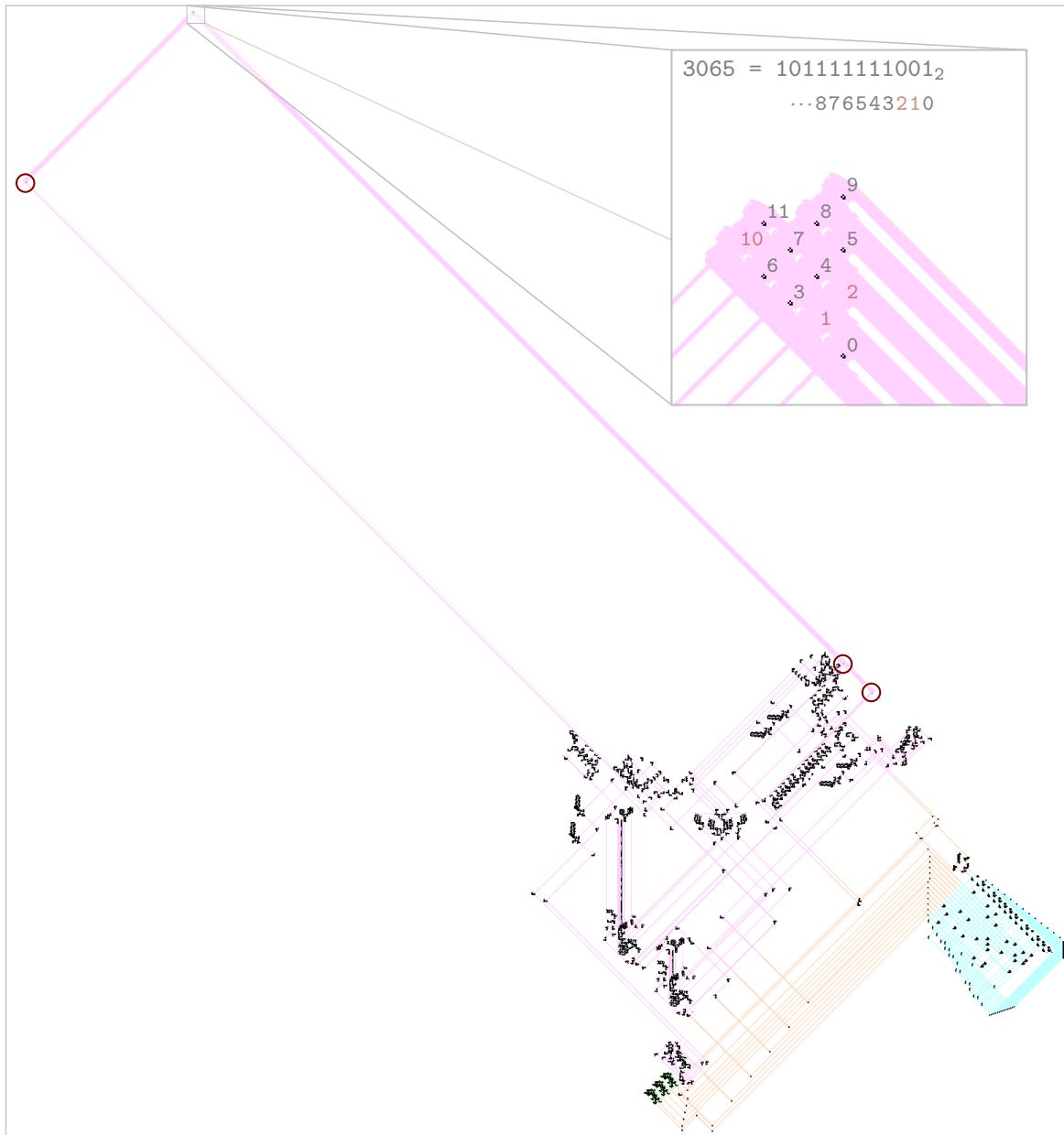


Figure 9.15: A pattern with minimal asymptotic diametric growth rate $\Theta(\sqrt{\log(t)})$, compiled from APGsembly 9.10. The computer at the bottom-right, highlighted in aqua, feeds into three sliding block registers (highlighted in green) and a 2D printer (highlighted in magenta). The 2D printer 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 $3065 = 10111111001_2$.

9.3 Modify APGsembly 9.3 for multiplying two sliding block registers so as to complete the same computation without erasing the value of U_0 .

[Hint: Add another temporary register.]

9.4 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 U_0 / U_1 in U_2 and the remainder of that division in U_3 . 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 $U_1 = 0$ 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” U_4 to 1).
- (d) Use the APGsembly compiler that is linked at conwaylife.com/wiki/APGsembly to compile a Life pattern that implements your APGsembly code from part (b).

9.5 Write APGsembly code that stores the value 241 in a binary register B_0 (using INC and SET actions as in APGsembly ??, not via the #REGISTERS header).

9.6 Suppose that U_0 and B_0 are a sliding block and binary register, respectively, storing some values. Write APGsembly code that computes $U_0 * B_0$ and stores it in B_0 .

9.7 In this exercise, suppose that U_0 is a sliding block register whose value has already been set.

- (a) Write APGsembly code that stores the value 2^{U_0} in the binary register B_0 .
- (b) Write APGsembly code that stores the value 10^{U_0} in the binary register B_0 .
- (c) Write APGsembly code that stores the value 3^{U_0} in the binary register B_0 .

9.8 Write APGsembly code that determines which of two binary registers B_0 and B_1 is storing a larger value (i.e., code that outputs different results depending on whether $B_0 \leq B_1$ or $B_1 < B_0$, analogous to APGsembly 9.1 for unary registers).

9.9 Explain why the 2d printer (displayed in Figure 9.15) prints its bits so far away from the computer and other logical circuitry.

[Hint: You can move the sliding block at the top-left corner a bit closer to the logical circuitry without breaking anything, but something goes wrong if you move it, for example, 5,000 cells closer.]

***9.10** In this exercise, we explore where the series (9.1) for π comes from.

- (a) Explain why $\pi = 4 - 4/3 + 4/5 - 4/7 + 4/9 - \dots$. [Hint: We did this already in the “Notes and Historical Remarks” section of Chapter 6.]
- (b) Manipulate the series from part (a) to show that

$$\pi = 2 + \left(\frac{4}{1 \cdot 3} - \frac{4}{3 \cdot 5} + \frac{4}{5 \cdot 7} - \frac{4}{7 \cdot 9} + \frac{4}{9 \cdot 11} - \dots \right).$$

[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

$$\pi = 2 + \frac{2}{3} + \left(\frac{8}{1 \cdot 3 \cdot 5} - \frac{8}{3 \cdot 5 \cdot 7} + \frac{8}{5 \cdot 7 \cdot 9} - \dots \right).$$

- (d) Repeat this method over and over again to rewrite this series as

$$\pi = 2 \left(1 + \frac{1!}{3} + \frac{2!}{3 \cdot 5} + \frac{3!}{3 \cdot 5 \cdot 7} + \frac{4!}{3 \cdot 5 \cdot 7 \cdot 9} + \dots \right).$$

- (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*.]

***9.11** Modify just 3 lines of the APGsembly code from Appendix C to turn the π calculator into an e calculator. [Hint: We described the changes that need to be made in Section 9.6.1.]

9.12 In this exercise, we use the following series to tweak the π calculator to instead compute

$$\sqrt{2} = \sum_{k=0}^{\infty} \frac{(2k+1)!}{2^{3k+1}(k!)^2} = \frac{1}{2} + \frac{3}{8} + \frac{15}{64} + \frac{35}{256} + \frac{315}{4096} + \dots.$$

- (a) 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) Change just a few lines of the APGsembly for the π calculator (from Appendix C) to account for these different values of a , b , p , q , r , and s .
- (c) If you compile the APGsembly code that you made in part (b), the resulting pattern will not compute the digits of $\sqrt{2}$ correctly. Why not? What additional change must you make to the introductory lines of that APGsembly code to make it function properly?

9.13 The e calculator from Exercise 9.11 can be sped up 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 speed 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.⁴⁴
- (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 9.11).

9.14 Write APGsembly code that determines how many decimal digits the integer stored in a sliding block register has.

9.15 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: This is much more difficult. Make use of the code from Exercise 9.14]

9.16 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 9.15.]

9.17 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 9.16 so that an integer n is not printed if the integer division n/k has a remainder of 0 for some $2 \leq k \leq n - 1$.]

9.18 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 integers.

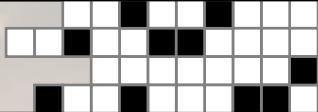
⁴⁴Even though decreasing the clock speed decreases the number of Life generations needed to perform a calculation, it typically does not decrease the real-world time that it takes Life software like Golly to evolve the pattern to the point of having performed that calculation. The reason for this is that the HashLife algorithm used by Golly is able to extremely quickly skip over “wasted” generations where the clock is just waiting, since so little of the Life plane changes during those generations.

Early draft (May 19, 2020).

Not for public dissemination.



10. Self-Supporting Spaceships



Life is too important a matter to be taken seriously.

Oscar Wilde

Nothing yet.

10.1 Caterpillar

Stuff.

Notes and Historical Remarks

Stuff.

Exercises

solutions on page 313

10.1 An exercise could be placed here.

10.2 Another exercise could be placed here.

Early draft (May 19, 2020).

Not for public dissemination.



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

Just testing some stuff out for now.

11.1 Demonoids

Stuff.

Notes and Historical Remarks

Stuff.

Exercises

solutions on page 313

11.1 An exercise could be placed here.

11.2 Another exercise could be placed here.

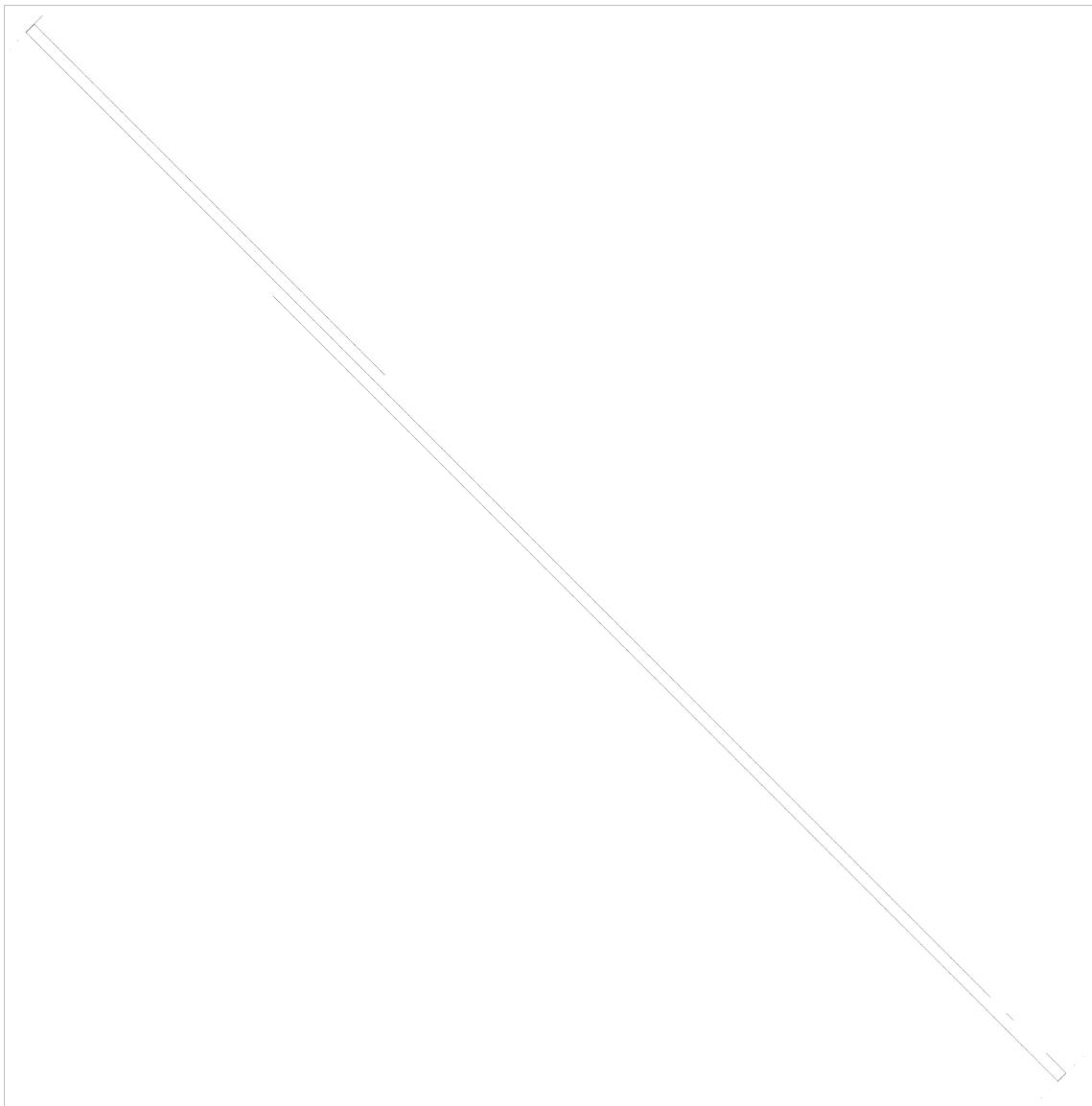
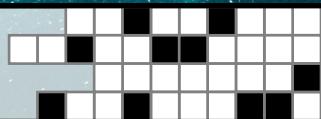


Figure 11.1: A demonoid. Just testing some stuff for now.



12. The OEOP Metacell



If you couldn't predict what [Life] did then probably that's because it's capable of doing anything.

John H. Conway

The previous chapter introduced universal construction and presented several adjustable spaceships using these techniques. This chapter explores a more complex application of universal construction: a self-reproducing cell which interacts with its neighbours to emulate Conway's Game of Life or a different cellular automaton on a much larger and slower scale.

12.1 Rule Emulation

Conway's Game of Life is a 2-state 9-neighbour rule: the next state of a cell depends on the current state of the cell together with the current states of the eight surrounding cells. Building a metacell that directly emulates Life would be highly nontrivial for several reasons:

- **Quantity of neighbours:** each metacell would need to be able to construct a copy of itself in up to eight different positions. If both the north and east neighbours are present, it may be difficult to manoeuvre a construction arm to build the north-east neighbour.
- **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 single-use and reusable stable reflector, respectively.

One of the main ideas behind the OEOP metacell is that we can emulate any 2-state 9-neighbour rule with an 8-state 4-neighbour rule, where a cell (x, y) can only be alive at time t if $x + y + t$ is even. This means that whenever a cell is alive, all four neighbours are dead, so there is plenty of empty space; moreover, every live cell always dies.

We use the cell states $\{0, 1, 2, 3, 4, 5, 6, 7\}$, where the background is state 0. In even generations in the emulating rule, the pattern consists entirely of cells in states $\{0, 7\}$; a live cell at position (x, y) in generation t of the original rule is encoded by a state-7 cell at position $(x + y, y - x)$ in generation $2t$ of the emulating rule.

In odd generations, the pattern consists entirely of cells in states $\{0, 1, 2, 3, 4, 5, 6\}$. The transition rule for computing generation $2t + 1$ from generation $2t$ is a fixed function:

$$h : \{0, 7\}^4 \rightarrow \{0, 1, 2, 3, 4, 5, 6\}$$

specially chosen such that the following function is injective:

$$H : \{0, 7\}^9 \rightarrow \{0, 1, 2, 3, 4, 5, 6\}^4$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \mapsto \begin{pmatrix} h(a_{11}, a_{12}, a_{21}, a_{22}) & h(a_{12}, a_{13}, a_{22}, a_{23}) \\ h(a_{21}, a_{22}, a_{31}, a_{32}) & h(a_{22}, a_{23}, a_{32}, a_{33}) \end{pmatrix}$$

Moreover, without loss of generality we can let $h(0, 0, 0, 0) = 0$.

The transition rule for computing generation $2t + 2$ from $2t + 1$ is a function which depends on the rule being emulated:

$$g : \{0, 1, 2, 3, 4, 5, 6\}^4 \rightarrow \{0, 7\}$$

The injectivity of H means that we can specify the function g such that the composition $g \circ H$ is exactly the transition rule we want to emulate (up to having renamed the cell states to ‘0’ and ‘7’). Making the necessary assumption that a dead cell surrounded entirely by dead cells remains dead, we have $g(0, 0, 0, 0) = 0$. Consequently, we can define a function:

$$f : \{0, 1, 2, 3, 4, 5, 6, 7\}^4 \rightarrow \{0, 1, 2, 3, 4, 5, 6, 7\}$$

where f coincides with g on the domain of g and coincides with h on the domain of h .

The OEOP metacell can be programmed to emulate any of the 8^{8^4-1} zero-preserving 8-state 4-neighbour rules. It takes 2^{35} generations for the OEOP metacell to run one generation of the 8-state rule (emulated at a 45-degree angle), and therefore 2^{36} generations to emulate one generation of the 2-state 9-neighbour rule (in the usual orientation).

12.2 Structure of the Metacell

The large-scale structure of the metacell is tripartite, and each of the three parts is constructed separately. The outer **shell** has exact order-4 rotational symmetry, consisting of four spiral arms which propagate gliders inwards. Only one of these arms is actually used; the other four exist only to enforce the strict symmetry.

Inside the shell is the **kernel**, which does not have any symmetry constraints. The south corner of the kernel contains logic circuitry for regulating the metacell’s lifecycle and computing the state of the metacell based on the inputs from the four neighbouring cells. The kernel also contains an output path, shown in crimson in Figure 12.1, which can connect to one of the four construction arms (violet, dashed) or to the input spiral arm of one of the four neighbours. During the operation of the metacell, the glider stream is redirected between the different possible outputs by deleting Snarks.

The largest region of the metacell is the **nucleus**, a huge boustrophedonic glider loop with a period of exactly 2^{29} . The nucleus is populated by over three million gliders, which together encode all of the construction recipes along with the lookup table for the metacell’s rule.

12.3 Memory Tape

Stuff.

12.4 Logic Circuitry

Stuff.

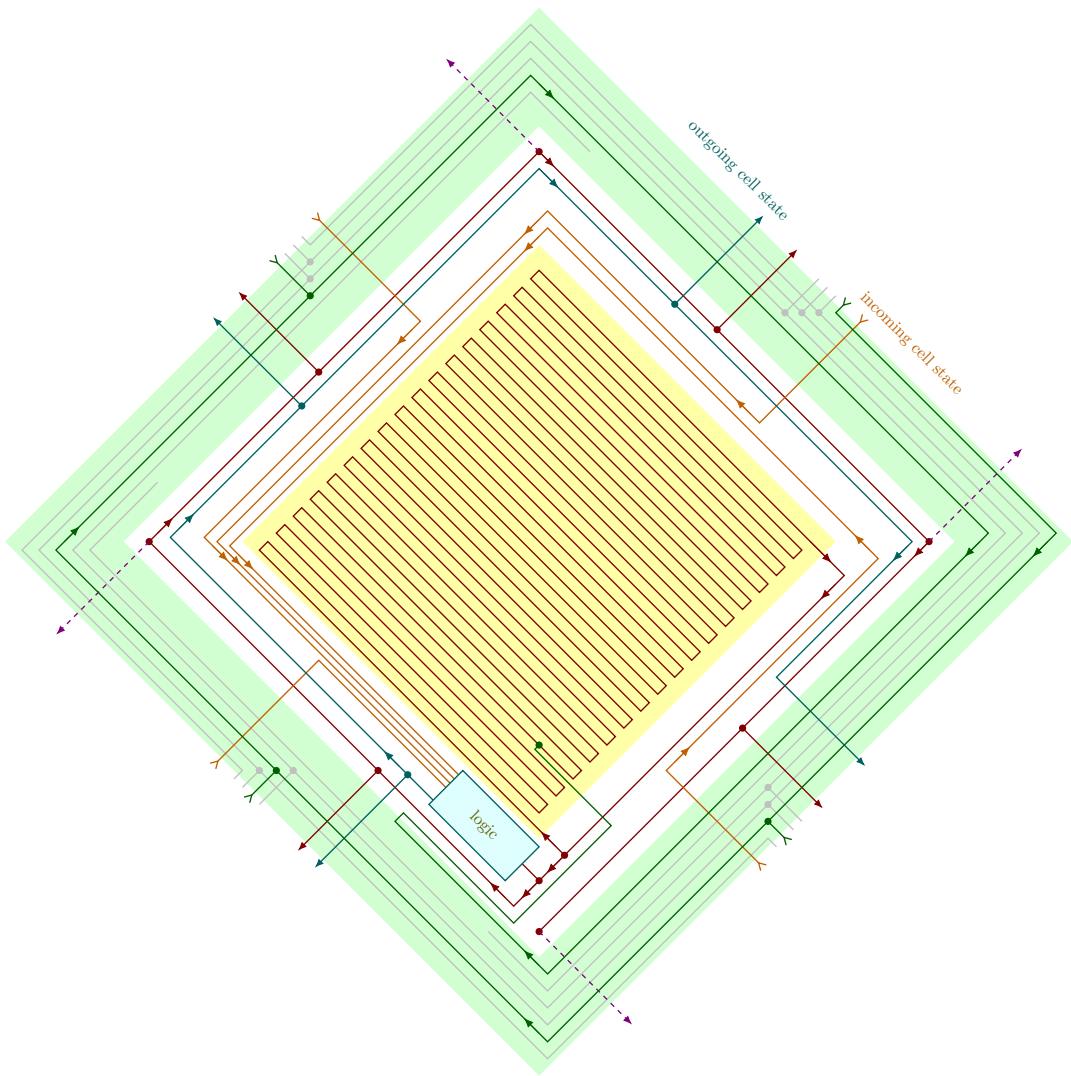


Figure 12.1: A schematic for the OEOP metacell. The symmetrical outer **shell** is highlighted in pastel green, and the **nucleus** is highlighted in yellow. The unhighlighted region between them is the **kernel**.

12.5 Construction

Stuff.

12.6 Encoding Other Cellular Automata in Life

Stuff.

12.7 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 OEOP. The first one was the *p5760 metacell*, which was constructed by David Bell in January 1996. This metacell is much smaller and faster than OEOP, with a period of just 5760 generations and a bounding box size of just 500×500 (see Figure 12.2). However, there are numerous trade-offs that make this metacell less useful:

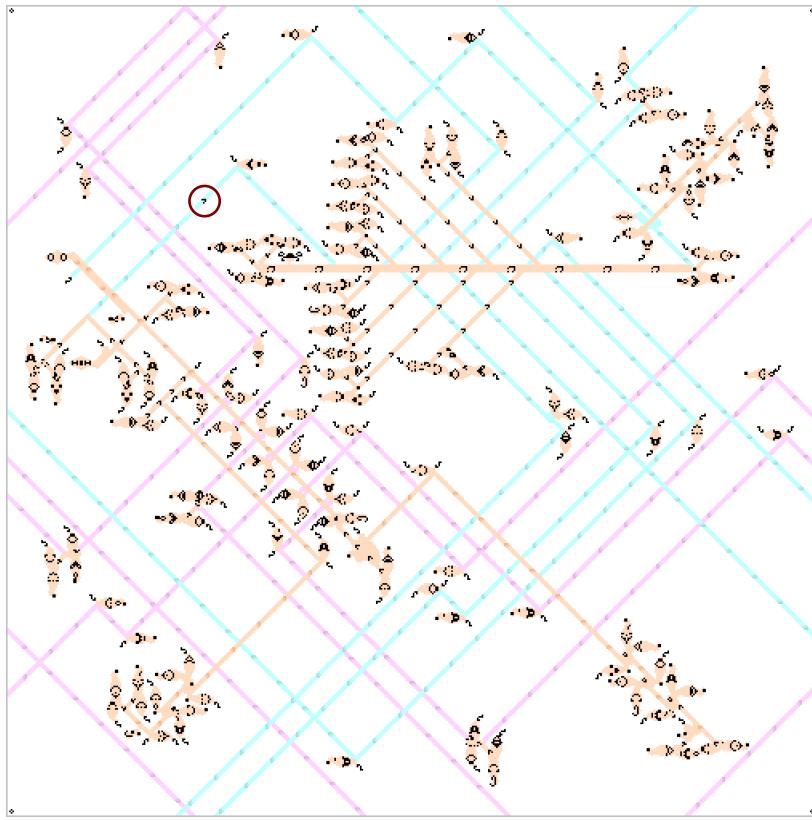


Figure 12.2: The *p5760 metacell*. Whether the cell is considered “alive” or “dead” is determined by the presence or absence of a glider at the location circled in red. That glider, if present, is duplicated 8 times and sent to its neighbors along the 8 output paths highlighted in cyan, signaling to them that it is alive. Similarly, neighboring alive cells send their signals to this one along the input paths highlighted in magenta.

- 1) It is hard-wired to emulate Life, and cannot easily be modified to emulate most of the 2^{512} different non-isotropic Life-like cellular automata.
- 2) 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 thus is not interesting to look at from a far-out zoom level.
- 3) “Dead” cells must be placed on the Life plane, which means that, for example, spaceships cannot be emulated by this metacell unless the pattern is infinitely large.

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 Life-like cellular automata. Indeed, built 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 metapixel (see Figure ??).

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 2,048 times as long and wide and 35,328 times as slow (see Figure 12.3).

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

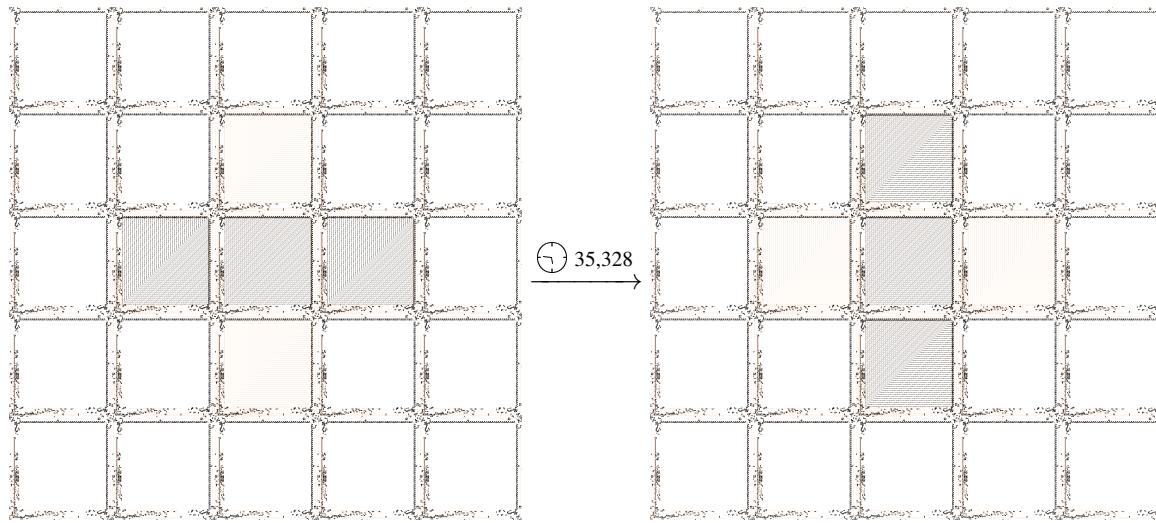


Figure 12.3: A *meta-blinker*: an arrangement of OTCA metapixels that emulates a blinker, but with period $2 \times 35,328$.

box of $2^{15} \times 2^{15}$ and a period of 2^{24} . The new features this time that warranted the extra size and delay were twofold:

- 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.¹
- All 2^{512} non-totalistic Life-like cellular automata can be emulated by this metacell, versus the 2^{18} outer-totalistic Life-like cellular automata that can be emulated by the OTCA metapixel.

Finally, Adam P. Goucher spent 2014–2018 constructing the 0EOP metapixel, which solved problem (3) described earlier—it does not require a background grid of “dead” cells to be placed on the Life plane.

Exercises

solutions to starred exercises on page 313

12.1 An exercise could be placed here.

12.2 Another exercise could be placed here.

¹This 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.

Early draft (May 19, 2020).

Not for public dissemination.



Appendices and Supplements

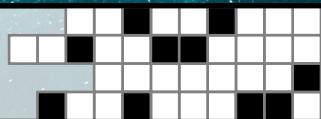
A	Mathematical Miscellany	291
A.1	Modular Arithmetic	
A.2	Greatest Common Divisor and Least Common Multiple	
A.3	Big-O Notation	
B	Universality of the Clock Inserter	293
B.1	Timing of Tight Glider Salvos	
B.2	Timing of the Clock Inserter	
C	Extra APGsembly Code	297
D	Solutions to Selected Exercises	301
	Bibliography	314
	Index	317

Early draft (May 19, 2020).

Not for public dissemination.



A. Mathematical Miscellany



In this appendix, we present some of the miscellaneous bits of mathematical knowledge that we needed to discuss certain concepts or to prove certain theorems. 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 Arithmetic

To be filled in later. Maybe not actually needed?

A.2 Greatest Common Divisor and Least Common Multiple

The *greatest common divisor* (or *GCD* for short) of two non-zero integers a and b is the largest positive integer that evenly divides both a and b . We often denote the GCD of a and b by $\gcd(a, b)$. For example, $\gcd(6, 15) = 3$. Our primary interest in the GCD comes from the fact that it tells us exactly which equations of the form $ax + by = c$ (where a, b, c are given integers, and x, y are variables we are trying to solve for) have integer solutions.¹ In particular, the following theorem answers this question:

Theorem A.1 — Bézout's identity. Let a, b , and c be non-zero integers. Then the equation $ax + by = c$ has a solution where both x and y are integers if and only if c is a multiple of $\gcd(a, b)$. Furthermore, if a and b have opposite signs then x and y can be chosen to both be positive.

Proof. To prove the “only if” direction of the theorem, we note that $\gcd(a, b)$ evenly divides each of a and b , so for all integers x, y it evenly divides $ax + by$ as well, so c must be 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 x, y such that $ax + by = c$. To this end, let x' and y' be integers that make $ax' + by'$ as small (but positive) as possible. For convenience, define $s := ax' + by'$. When dividing a by s , the remainder r is also of the form $r = ax + by$ since it is obtained by subtracting a multiple of $s = ax' + by'$ from a . Since the $0 \leq r < s$, 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).

¹An equation of this form is called a *linear Diophantine equation*.

It follows that c/s is an integer (since c is a multiple of $\gcd(a, b)$, which is a multiple of s), so $x = x'(c/s)$, $y = y'(c/s)$ is a pair of integers satisfying $ax + by = (ax' + by')(c/s) = s(c/s) = c$, as desired.

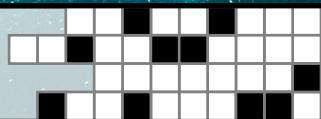
To prove the final claim that x and y can be chosen to be positive when a and b have opposite signs, note that if x and y are integers such that $ax + by = c$ then for any integer k it is also the case that $a(x + k(b/\gcd(a, b))) + b(y - k(a/\gcd(a, b))) = c$. By taking k large enough and positive (if $a < 0$ and $b > 0$) or large enough and negative (if $a > 0$ and $b < 0$), this gives us a positive solution to the equation. ■

The *least common multiple* (or *LCM* for short) of two positive integers a and b is the smallest integer that is a multiple of both a and b . We often denote the LCM of a and b by $\text{lcm}(a, b)$. For example, $\text{lcm}(6, 15) = 30$. It is straightforward to verify that $\text{lcm}(a, b) = ab/\gcd(a, b)$ for all a, b .

A.3 Big-O Notation

Stuff.

B. 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 5.27 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 p2 slow salvo.

Proposition B.1 Multiple copies of the clock inserter reaction can be used to construct every possible glider salvo.

B.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 neighbourhoods of those lanes do not overlap enough to cause an unexpected birth between the gliders.

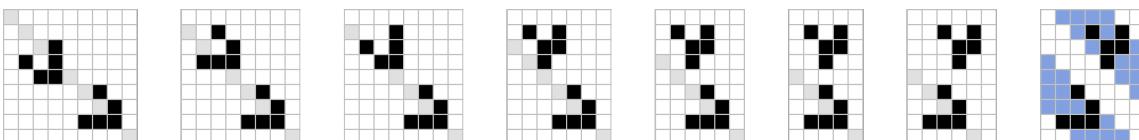


Figure B.1: The closest timings that are possible between two gliders in nearby lanes without colliding with each other. From left to right, these gliders differ by 0, 1, 2, 3, 4, 5, and 6 lanes, and their timings differ by 14, 14, 14, 13, 11, 9, and 7 generations. The rightmost image shows that gliders that differ by 7 lanes can never collide with each other as their paths are not close enough to each other.

B.2 Timing of the Clock Inserter

The basic idea behind proving Proposition B.1 is to show that we can use the clock inserter 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 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

.	.	.	-9	-6	-3	0
.	.	.	-8	-2	1	
.	.	.	-7	-4	2	
.	.	-9			3	
.	.	-8	-5	-2	1	4
.	.	-7	-4	-1	2	5
.	-9	-6	0	3	6	
.	-5		1	4	7	
-7			2	5	8	
-9	-6	-3	0	3	6	9

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 $-6, -5, \dots, 5, 6$.

If the timing of the glider in a particular lane is small enough (i.e., far enough below 0), it will be 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 $-6, -5, \dots, 5, 6$ that *does* collide with the clock inserter, and these values are presented in Table B.1.

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.

¹While 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.

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 .

Case 2: The colliding glider is in one of the lanes $\dots, -9, -8, -7$ or $7, 8, 9, \dots$

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 $\dots, -9, -8, -7$ or $7, 8, 9, \dots$ 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 Proposition B.1, it is worth focusing on some of the values in Table B.2 and clarifying some points about the rank of a glider. Firstly, the obvious pattern of the minimal timing increasing by 2 generations every time you move 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. Secondly, 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.

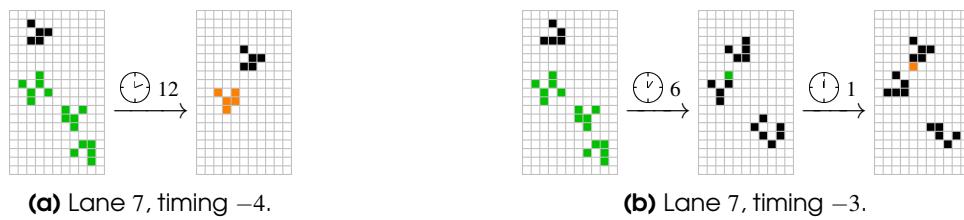
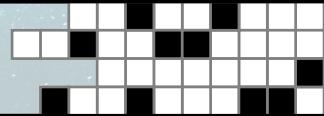


Figure B.3: A clock inserter attempting to insert a glider in lane 0 at timing 0 when there is already a glider in lane 7 with (a) timing -4 and (b) timing -3 . In (a), the new glider is successfully inserted, but in (b), a cell that is outside of the path of the glider that is being inserted (shown 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).

In fact, this oddity 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 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 5.38).

C. Extra APGsembly Code



In Section 9.6, we developed an algorithm and pseudocode for computing the decimal digits of $\pi = 3.14159\dots$. 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 9.13. In particular, APGsembly C.1 and C.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: top-left corner of A_n matrix
 - U1: top-right and bottom-right corners of 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: number of bits of memory allocated to the binary registers
 - U7–U9: temporary registers used to help perform arithmetic operations and miscellaneous loops
- B0: top-left corner of B_n matrix
 - B1: top-right corner of B_n matrix ($= q_n$)
 - B2: bottom-right corner of 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 listed actions and jump operation regardless of whether they receive a Z or NZ input value.

APGsembly C.1 Page 1 of APGsembly code for a π calculator that implements Pseudocode 9.3.

```

#COMPONENTS NOP,DIGITPRINTER_SE,B0~3,U0~9,ADD,SUB,MUL
#REGISTERS {"U1":1, "U6":6, "B0":2, "B2":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;     MULA7;   TDEC B1
MULA4;  NZ;    MULA5;   READ B1
MULAS;  Z;     MULA6;   READ B3
MULAS;  NZ;    MULA6;   SET B1, SET B3, NOP
MULA6;  *;    MULA4;   INC B1, INC B3, TDEC U7
MULA7;  Z;     MULA8;   TDEC B3
MULA7;  NZ;    MULA7;   TDEC B1
MULA8;  Z;     MULA9;   TDEC U1
MULA8;  NZ;    MULA8;   TDEC B3
# Copy U1 to temporary register U8.
MULA9;  Z;     MULA10;  TDEC U7
MULA9;  NZ;    MULA9;   TDEC U1, INC U7
MULA10; Z;    MULA11;  TDEC U8
MULA10; NZ;    MULA10;  TDEC U7, INC U1, INC U8
# Set B1 = U1 * B3 and U8 = 0.
MULA11; *;    MULA12;  TDEC U8
MULA12; Z;    MULB1;   TDEC U6
MULA12; NZ;    MULA13;  TDEC U6
MULA13; Z;    MULA14;  TDEC U9
MULA13; NZ;    MULA13;  TDEC U6, INC U9
MULA14; Z;    MULA15;  TDEC U7
MULA14; NZ;    MULA14;  TDEC U9, INC U6, INC U7
MULA15; Z;    MULA19;  TDEC B3
MULA15; NZ;    MULA16;  READ B3
MULA16; Z;    MULA17;  READ B1
MULA16; NZ;    MULA17;  READ B1, SET B3, ADD A1
MULA17; Z;    MULA18;  ADD B0
MULA17; NZ;    MULA18;  ADD B1
MULA18; Z;    MULA15;  TDEC U7, INC B1, INC B3
MULA18; NZ;    MULA18;  SET B1, NOP
MULA19; Z;    MULA20;  TDEC B1
MULA19; NZ;    MULA19;  TDEC B3
MULA20; Z;    MULA12;  TDEC U8
MULA20; NZ;    MULA20;  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;   MULB14;  TDEC U8
MULB14; Z;    MULB13;  TDEC U9, INC U6, INC U7
MULB14; NZ;   MULB14;  TDEC U8
MULB15; Z;    MULB18;  TDEC B3
MULB15; NZ;   MULB15;  READ B3
MULB16; Z;    MULB16;  READ B0
MULB16; NZ;   MULB16;  READ B0, SET B3, ADD A1
MULB17; Z;    MULB17;  ADD B0
MULB17; NZ;   MULB17;  ADD B1
MULB18; Z;    MULB18;  TDEC B3
MULB18; NZ;   MULB18;  SET B0, NOP
MULB19; Z;    MULB19;  TDEC B0
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

```

APGsembly C.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

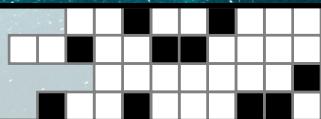
# Check whether or not we just printed the very first digit (3).
# If so, print a decimal point. Either way, increase U3, which
# counts which decimal place we are currently at, and loop back
# to start the next digit calculation.
OUTD1; ZZ; OUTD2; TDEC U3
OUTD2; Z; ITER1; INC U3, NOP, OUTPUT .
OUTD2; NZ; OUTD3; INC U3, NOP
OUTD3; ZZ; ITER1; INC U3, NOP

```

Early draft (May 19, 2020).

Not for public dissemination.

D. Solutions to Selected Exercises



Chapter 1: Early Life

- 1.1.** (a) A twin bees shuttle
 (b) A period 14 oscillator (called the *tumbler*).
 (c) A block-laying switch engine.
 (d) A glider-producing switch engine.
- 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).
- 1.9.** (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).
- 1.5.** (a) Generation 41.
 (b) Generation 42.
 (c) Generation 240.
 (d) Generation 774.
 (e) Generation 97.
- 1.8** (a)
 (b)
1.14. (a) $n = \left\lceil \frac{2}{6 - \sqrt{\log_2(2^{36} - 1)}} \right\rceil = 1143185077171.$
 (b) The new inequality to check is $(2^{36} - 4)^n < 2^{(6n-2)^2}$, which is true whenever

$$n \geq \left\lceil \frac{2}{6 - \sqrt{\log_2(2^{36} - 4)}} \right\rceil = 285796269287.$$

 (c) The new inequality to check is $(2^{25} - 1)^n < 2^{(5n-2)^2}$, which is true whenever

$$n \geq \left\lceil \frac{2}{5 - \sqrt{\log_2(2^{25} - 1)}} \right\rceil = 465163192.$$

 (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 of “buffer” dead cells around each cell that differs between the tiles.
- 1.17.** (a) The blinker causes a second backward-firing switch

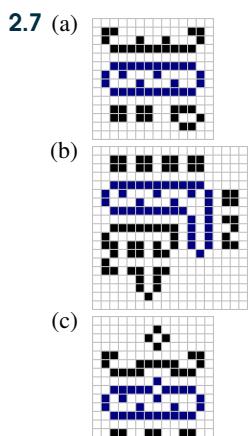
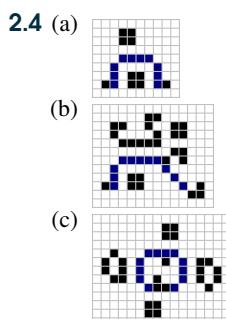
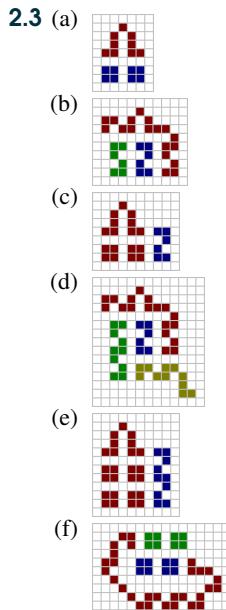
engine to form, so now there are two separate glider streams being fired back at the initial debris.

- (b) A forward-moving glider hits the switch engines at

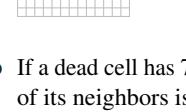
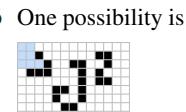
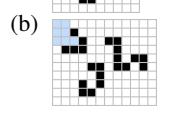
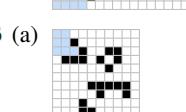
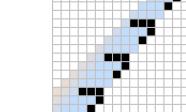
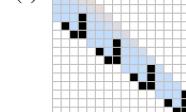
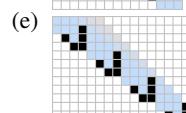
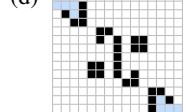
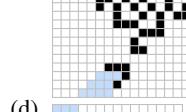
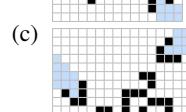
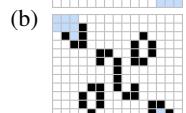
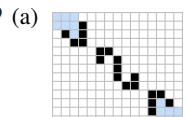
the front, transforming them into block-laying ones rather than ones that shoot gliders backward.

Chapter 2: Still Lifes

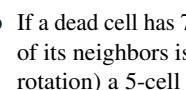
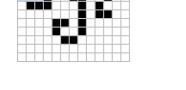
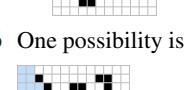
- 2.1** (a) Strict still life.
 (b) Strict still life.
 (c) Pseudo still life.
 (d) Pseudo still life.
 (e) Neither.
 (f) Strict still life.



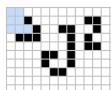
- 2.9** (a)



- 2.13** (a)



- 2.15** One possibility is as follows:



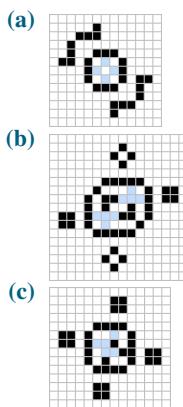
- 2.16** If a dead cell has 7 or 8 live neighbors, then at most one of its neighbors is dead, so it is possible to find (up to rotation) a 5-cell “u”-shaped region of neighbors that are all alive. The middle-bottom cell in this “u”-shaped

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

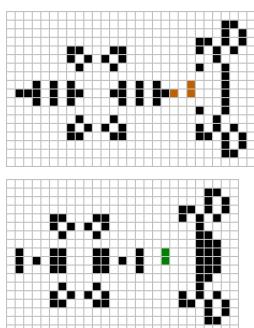
Chapter 3: Oscillators

- 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:

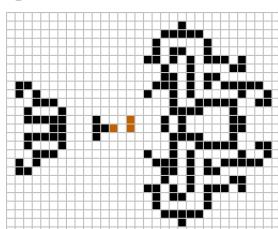


- 3.5** In these solutions, we use orange cells to denote accessible sparks and green cells to denote cells that oscillate at the full period in our new compound oscillators.

- (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:

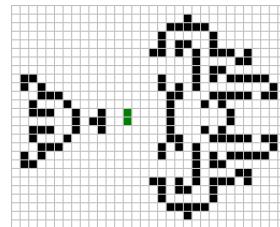


- (b) We combine a T-nosed p4 with a p5 heavyweight volcano (a fumarole does not quite work since its spark is a bit too close to the body of the oscillator):

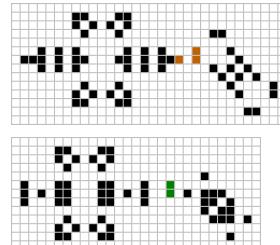


- (b) 5

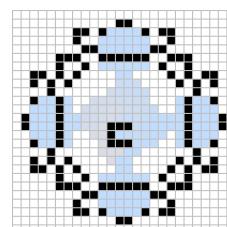
- (c) If L is the number of live cells in an $n \times n$ square, we find that $11L \leq 6n^2 + 12n$, so $L \leq (6/11)n^2 + (12/11)n$ and hence the asymptotic density of a still life cannot exceed $6/11$.



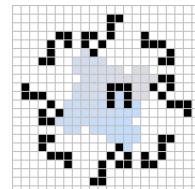
- (c) We combine a T-nosed p6 with a p8 figure eight:



- 3.8** 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.

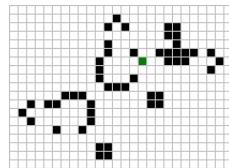
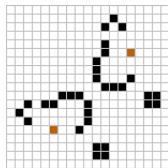


- 3.9**

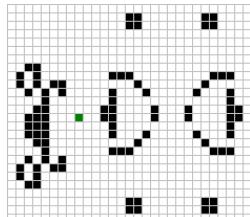
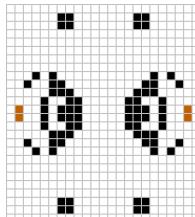


- 3.10** In these solutions, we use orange cells to denote accessible sparks and green cells to denote cells that oscillate at the full period in our new compound oscillators.

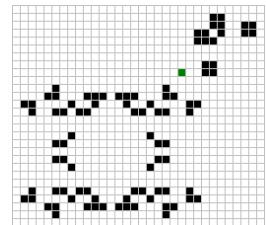
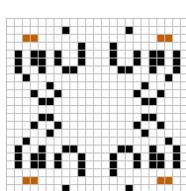
- (a) We can combine this period 13 oscillator with a period 3 caterer to create an oscillator with period $\text{lcm}(3, 13) = 39$:



- (b)** We can combine this period 31 oscillator with a period 4 middleweight emulator to create an oscillator with period $\text{lcm}(4, 31) = 124$:



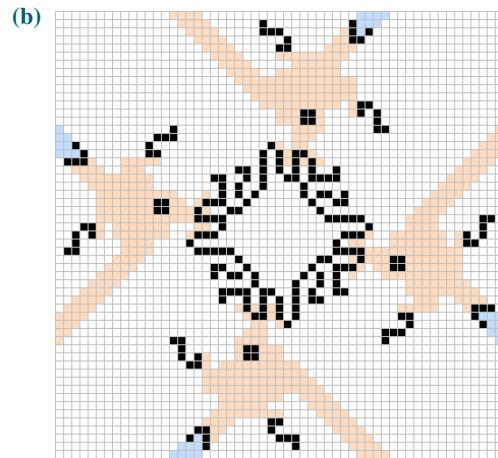
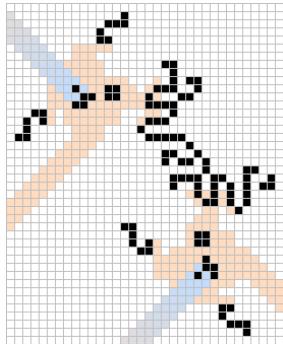
- (c)** We can combine this period 32 oscillator with a period 6 unix to create an oscillator with period $\text{lcm}(6, 32) = 96$:



- 3.15** 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.



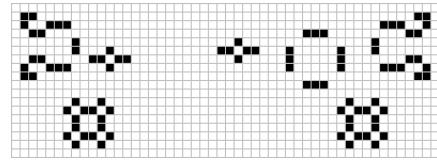
- 3.16 (a)**



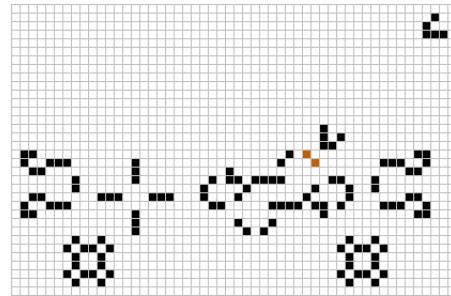
- 3.17** A pre-honeyfarm.

- 3.18** Unfortunately, the block that the glider hits is moved up by 1 cell, so any subsequent gliders will not trigger the same reaction.

- 3.22** One possibility is the following oscillator, which was found by Noam Elkies in 1994. It is currently the smallest known period 50 oscillator.



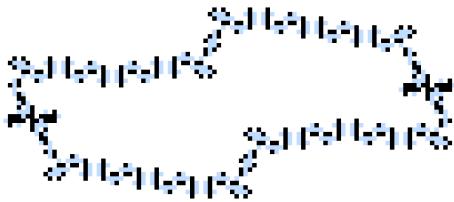
- 3.23** The traffic jam creates dot and duoplet sparks. One way of using the period 50 oscillator from the solutions to Exercise 3.22 to reflect gliders is displayed below:



- 3.29** Suppose that there is a single cell (which we will call X) that oscillates with period 4. Thus cell X must either be alive in 3 of its phases or dead in 3 of its phases. Suppose without loss of generality that cell X has the same state (alive or dead) in generations 0, 1, 2, and the opposite state in generation 3.

Since all other cells 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.

- 3.30** One possibility is as follows:



Chapter 4: Spaceships and Moving Objects

4.1 (a) Opposite.

(b) Same.

(c) Same.

4.2 (a) Color-preserving.

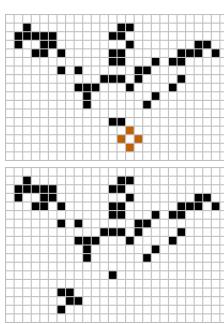
(b) Color-preserving.

(c) Color-preserving.

(d) Color-preserving.

(e) Color-changing.

4.4 (a)

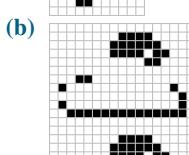


(b)

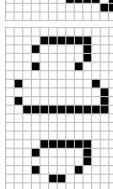
4.9 (a)



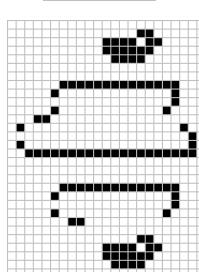
(b)



(c)

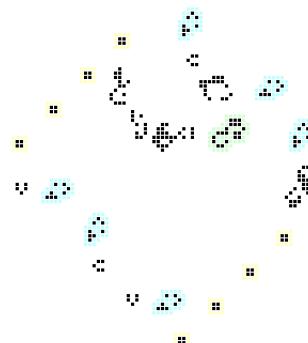
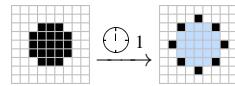


4.10



4.15 One possibility is the 7-engine Cordership displayed here, which was originally found by Dean Hickerson in 1993:

3.31 One phoenix that works is the following one that dies completely in 2 generations:

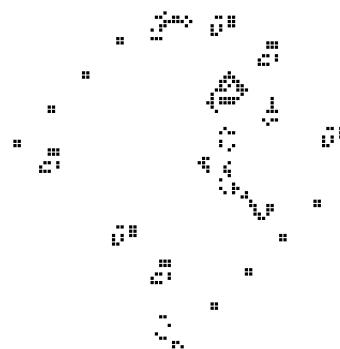


4.16 (a) In this reaction, the switch engines are 1 cell diagonally farther back relative to the blocks than in the original reaction.

(b) This reaction emits a banana spark, which can be used to reflect a glider as follows:

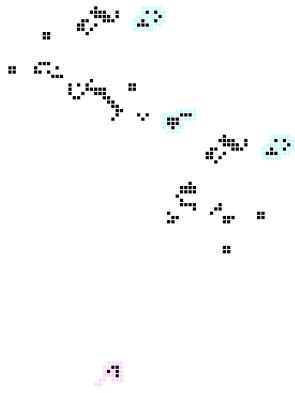


(c) We can just replace the back of the 7-engine Cordership from Exercise 4.15 (or we could have replaced the back to the 10-engine Cordership):



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) 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:



Chapter 5: Glider Synthesis

5.2 (a)

(b)



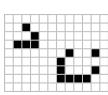
(c)



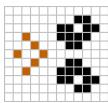
(d)



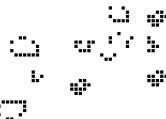
(e)



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-)honeyfarm, and a shifted block. We

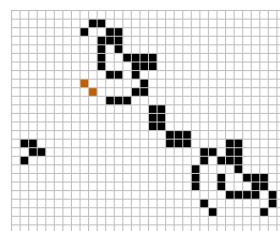
4.21 (a)

(b) The T-tetromino.

4.23**4.30** (a) 3, 18, and 2, respectively.

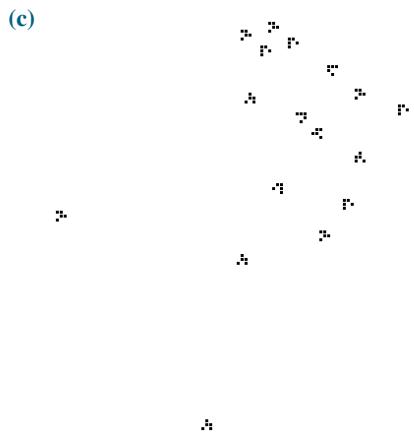
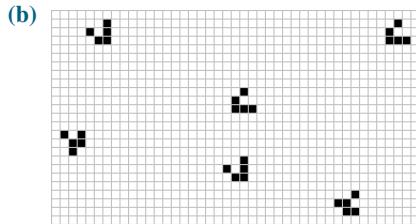
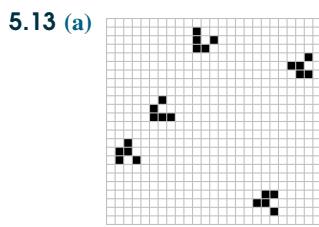
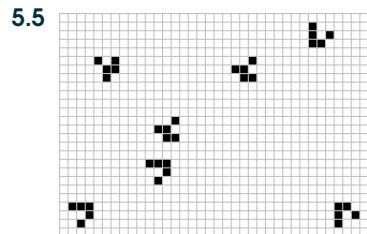
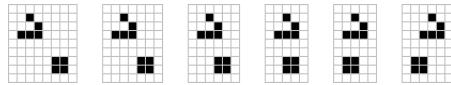
(b) 3, 8, and 51, respectively.

4.32 This spaceship gives off a duoplet spark at its back end, which can reflect a glider as in Figure 3.17:

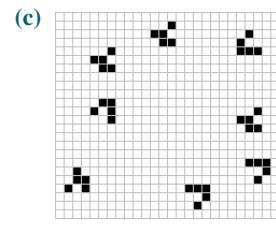
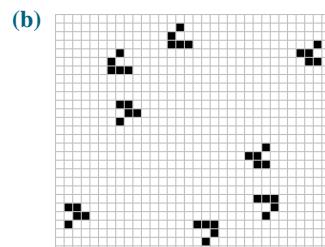
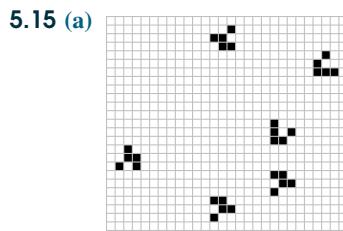
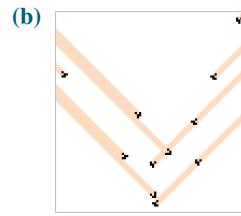
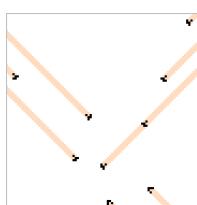
**4.19** (a)

(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).

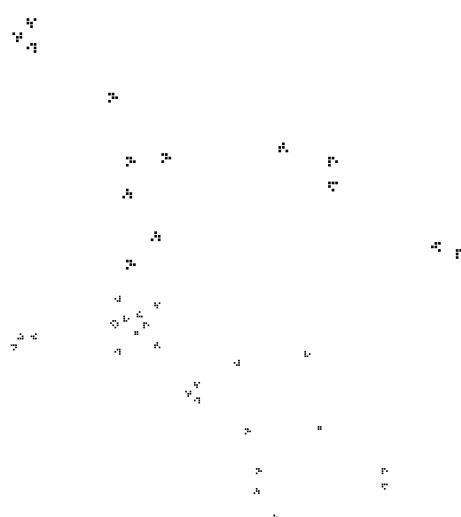
saw the block-shifting and honeyfarm-producing collisions in this chapter, and the collision used by the Snark is also the honeyfarm-producing type. We will see a use of the pi-producing collision in Figure 7.8.



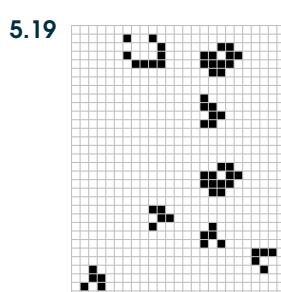
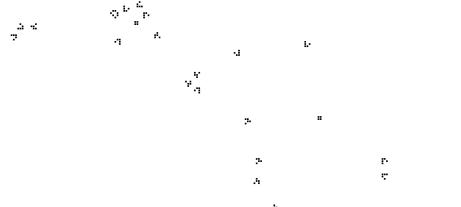
5.14 (a) The following 9-glider synthesis was found by Luka Okanishi almost immediately after the 2-engine Cordership's discovery:



5.16 (a)

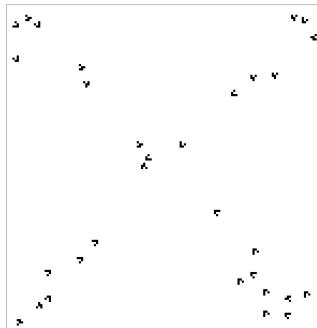


(b)

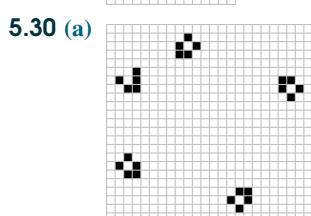
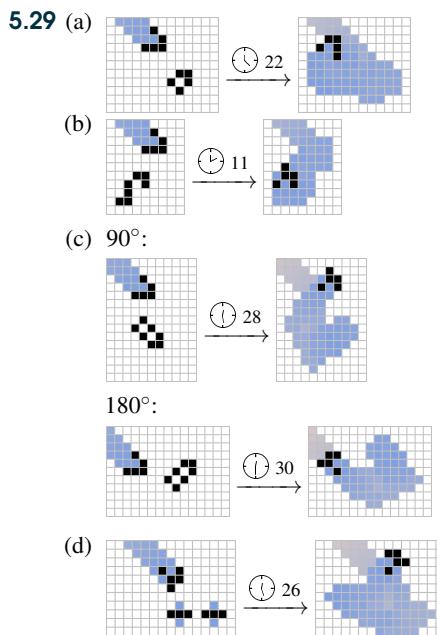


5.28 (a) If we remove four gliders from the top-left and two

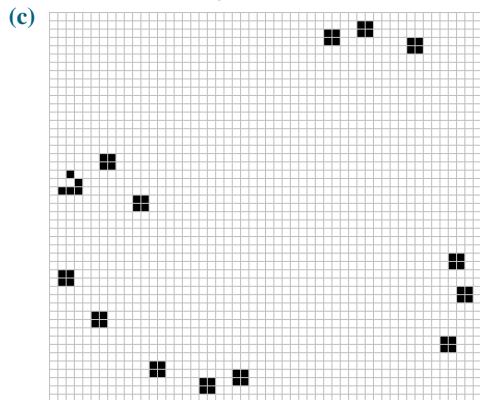
gliders from the bottom-left, we arrive at the following crab synthesis:



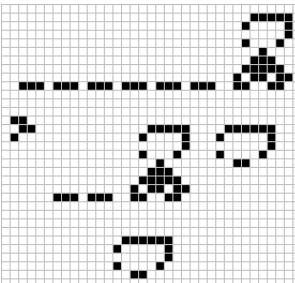
- (b) One way is to fire two gliders at the front of the crab (tubstretcher) as follows. By moving these gliders farther away, the resulting still life can be as large as we like. For example, the following configuration synthesizes a 209-cell still life.



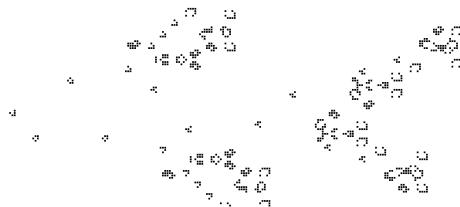
- (b) Because the input gliders to the two different turning reactions have different colors. If a single glider hits the same turner four times, it will always end up having the same color that it started with, and thus cannot have the opposite color required to initiate the second turning reaction.



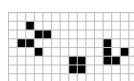
5.31 (a)



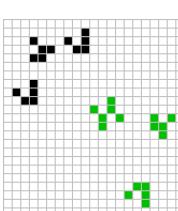
- (b) One method that makes use of p60 space rakes is as follows:



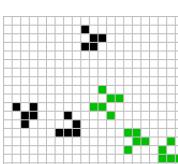
- 5.32** Our goal is to find a way of firing a glider at a block in such a way that both the glider and the block are destroyed, but a domino spark is produced in the process, thus triggering the clock inserter. One method that works is:

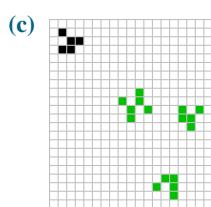
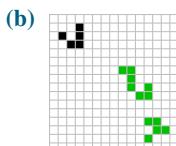
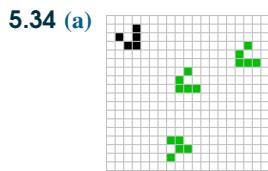
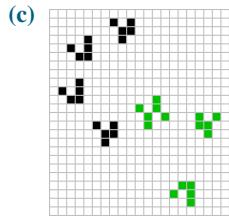


5.33 (a)



(b)





5.38 (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.

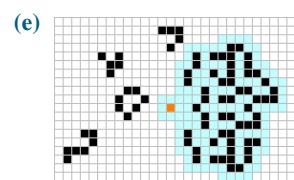
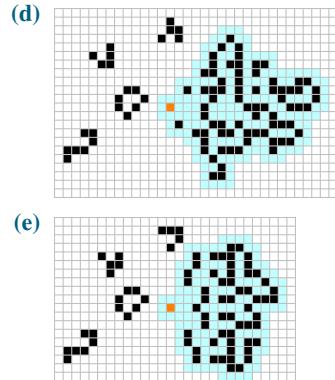
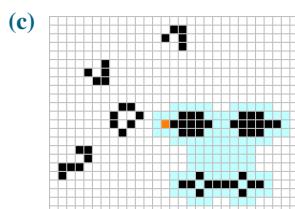
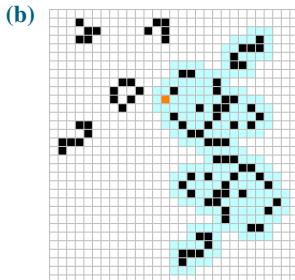
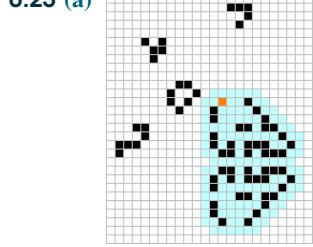
(b) 5/3

(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.

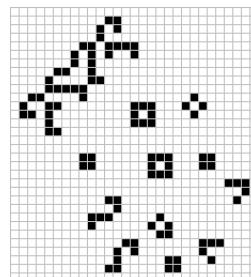
(d) The minimal value of w is 7/9, since this choice gives a rank of 0 in lane -7. This time, we must break rank ties by first inserting gliders that are farther to the *left* along lines of constant rank.

(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 $w = 7/3$).

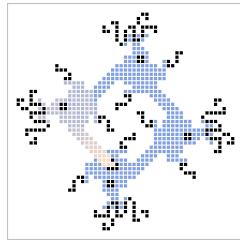
Chapter 6: Periodic Circuitry



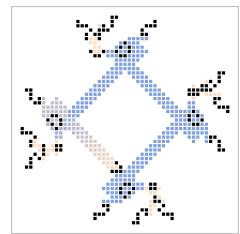
6.26 The p29 shuttle emits a domino (pipsquirter) spark at the bottom, so one method that works is as follows:



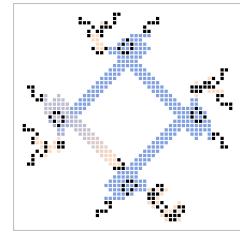
6.25 (a) The period of the resulting glider loop is 232, which we could have determined simply by noting that the original loop had period 216 and this one has the Snarks each 1 cell farther from each other diagonally, so the new period must be $216 + 4 + 4 + 4 + 4 = 232$.



- (b) The original loop had period 232 and bumpers are 5 generations slower than Snarks, so the new loop has period $232 + 5 + 5 + 5 + 5 = 252$:

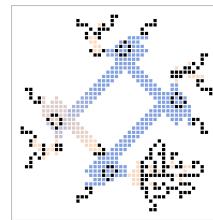


- (c) Here we replace the bottom p4 bumper by a p6 bumper:



- (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:



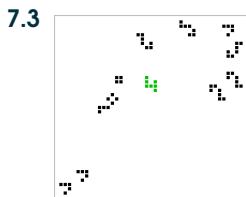
- 6.29** (a) 4 more gliders fit in the loop, since we added $2 \times 4 \times 15 = 120$ extra generations of travel time around the loop and the gliders are spaced 30 generations apart from each other.

- (b) Its period is $30 \times 402,653,181 = 12,079,595,430$.

- (c) Its period is $30 \times 1,057 = 31,710$.

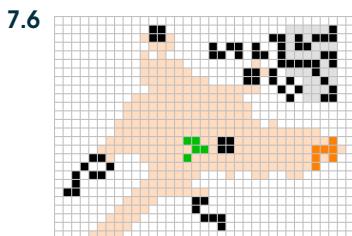
Chapter 7: Stationary Circuitry

- 7.1** (a) F209, repeat time 60.
 (b) R199, repeat time 67.
 (c) Fx153, repeat time 60.
 (d) L200, repeat time 59.



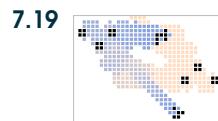
- 7.4** It has more transparent lanes (up to 6 lanes instead of just 2 if we replace the southwest eater 1 with an eater 5 like we did in Exercise 7.3).

- 7.5** A pi-heptomino.



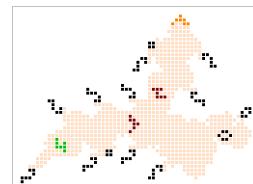
- 7.8** For all three of these glider multipliers, we just insert more copies of the Fx77 conduit in the middle of the

glider tripler that we saw in Figure 7.10(b):

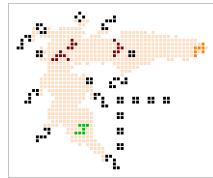


- 7.22** (a) The top-left eater 1 in BR146H overlaps with the bottom-right eater 1 in the original Hfx58B.

- (b) This conduit is named HL262B:



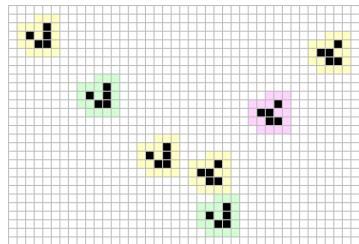
- 7.23** The variant of Hfx58B from Exercise 7.22 is needed to connect to the following conduit, and the final (right-most) variant of BFx59H needed to dodge the obstacle.



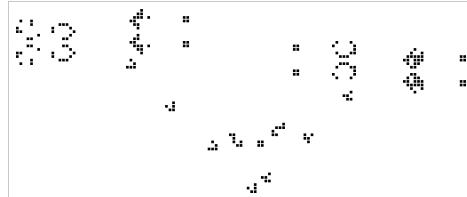
- 7.24** The Herschel is converted into an R-pentomino (well, generation 4 of its evolution anyway), then a B-heptomino (well, a B-heptaplet), and then back into a Herschel. The conduits that do the conversions are called HLx69R, RF28B, and BFx59H.

Chapter 8: Guns and Glider Streams

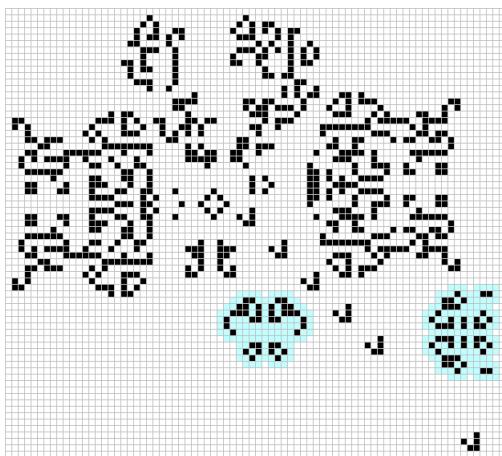
8.2



- 8.3** Since $322/7 = 46$, we use two p46 twin bees guns:

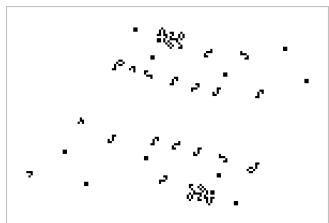


- 8.5** One possibility is to use two copies of Rich's p16 as below. Alternatively, the copy of Rich's p16 at the right can be replaced by a blocker.



- 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 the spark from the filter would have to affect every single glider that passes by it if it affects any of them.

7.28 (a)



(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).

8.9 (a)



- (b)** Period 5 bumpers are the only ones that work:



Chapter 9: Infinite Growth

- 8.29** This does not work because the phase of the Cordership in the caber tosser is not actually such that the glider was reflected a few generations ago (rather, it is offset 48 generations from the phase in which it can reflect a glider). The reason for this is that the previous trip that the glider took from the guns to the Cordership (if

it actually took place) would have taken 240 generations, which is not divisible by 96 (the period of the Cordership). However, the *next* trip from the guns to the Cordership takes 480 generations, which *is* divisible by 96, so the reaction works from this point on.

Chapter 9: Universal Computation

- 9.10 (a)** Just recall that the Taylor series for $\arctan(x)$ centered at $x = 0$ is

$$\arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots$$

This series converges when $x = 1$ (by the alternating series test from calculus), and $\arctan(1) = \pi/4$ (by looking at a triangle with angles $\pi/4$, $\pi/4$, and $\pi/2$), so

$$\frac{\pi}{4} = \arctan(1) = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$$

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

$$\begin{aligned} \pi &= (2+2) - \left(\frac{2}{3} + \frac{2}{3}\right) + \left(\frac{2}{5} + \frac{2}{5}\right) - \dots \\ &= 2 + \left(2 - \frac{2}{3}\right) - \left(\frac{2}{3} - \frac{2}{5}\right) + \left(\frac{2}{5} - \frac{2}{7}\right) - \dots \\ &= 2 + \left(\frac{4}{1 \cdot 3} - \frac{4}{3 \cdot 5} + \frac{4}{5 \cdot 7} - \dots\right), \end{aligned}$$

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.

- (c)** Using summation notation now, the series from part (b) can be written in the form

$$\begin{aligned} \pi &= 2 + \sum_{k=0}^{\infty} \frac{4(-1)^k}{(2k+1)(2k+3)} \\ &= 2 + \sum_{k=0}^{\infty} (-1)^k \left(\frac{2}{(2k+1)(2k+3)} + \frac{2}{(2k+1)(2k+3)} \right) \\ &= 2 + \frac{2}{3} + \sum_{k=0}^{\infty} (-1)^k \left(\frac{2}{(2k+1)(2k+3)} - \frac{2}{(2k+3)(2k+5)} \right) \\ &= 2 + \frac{2}{3} + \sum_{k=0}^{\infty} (-1)^k \frac{8}{(2k+1)(2k+3)(2k+5)}, \end{aligned}$$

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

$$\begin{aligned} \pi &= 2 \sum_{k=0}^{m-1} \frac{k!}{1 \cdot 3 \cdots (2k+1)} \\ &\quad + \sum_{k=0}^{\infty} \frac{4m!(-1)^k}{(2k+1)(2k+3) \cdots (2k+2m+1)}. \end{aligned}$$

The infinite sum above is an alternating series with decreasing terms, so it is no larger than its first term: $4m!/(1 \cdot 3 \cdots (2m+1))$, which tends to 0 as $m \rightarrow \infty$. It follows that

$$\begin{aligned} \pi &= 2 \lim_{m \rightarrow \infty} \sum_{k=0}^{m-1} \frac{k!}{1 \cdot 3 \cdots (2k+1)} \\ &\quad + \lim_{m \rightarrow \infty} \sum_{k=0}^{\infty} \frac{4m!(-1)^k}{(2k+1)(2k+3) \cdots (2k+2m+1)} \\ &= 2 \sum_{k=0}^{\infty} \frac{k!}{1 \cdot 3 \cdots (2k+1)} + 0, \end{aligned}$$

as desired.

- (e)** This comes immediately from the fact that, for every integer $k \geq 1$, the k -th term in the series from part (d) is $k/(2k+1)$ times the previous term. A bit more explicitly,

$$\begin{aligned} \frac{\pi}{2} &= 1 + \frac{1!}{3} + \frac{2!}{3 \cdot 5} + \frac{3!}{3 \cdot 5 \cdot 7} + \frac{4!}{3 \cdot 5 \cdot 7 \cdot 9} + \dots \\ &= 1 + \frac{1}{3} \left(1 + \frac{2}{5} + \frac{2 \cdot 3}{5 \cdot 7} + \frac{2 \cdot 3 \cdot 4}{5 \cdot 7 \cdot 9} + \dots \right) \\ &= 1 + \frac{1}{3} \left(1 + \frac{2}{5} \left(1 + \frac{3}{7} + \frac{3 \cdot 4}{7 \cdot 9} + \dots \right) \right) \\ &= 1 + \frac{1}{3} \left(1 + \frac{2}{5} \left(1 + \frac{3}{7} \left(1 + \frac{4}{9} + \dots \right) \right) \right). \end{aligned}$$

Multiplying through by 2 then (finally!) gives exactly the desired series (9.1).

-
- 9.11** We change the first 12 lines of that APGsembly to the following:

```

INITIAL; ZZ; INIT1; READ TO
INIT1; *; INIT2; SET TO, 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

```

Chapter 10: Self-Supporting Spaceships

- 10.1** Stuff.

Chapter 11: Universal Construction

- 11.1** Stuff.

Chapter 12: The OEOP Metacell

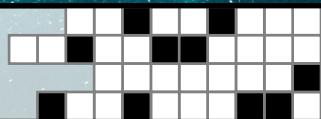
- 12.1** Stuff.

Early draft (May 19, 2020).

Not for public dissemination.



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 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 *Voronoi’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.
- [Fla94] Achim Flammenkamp. Natural grown oscillators out of random soup. <http://wwwhomes.uni-bielefeld.de/achim/oscill.html>, 1994. [Online; accessed May 6, 2016].

- [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 May 6, 2016].
- [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.
- [Gib06] J. Gibbons. Unbounded spigot algorithms for the digits of pi. *The American Mathematical Monthly*, 113(4):318–328, 2006.
- [Gos84] R. Wm. 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.
- [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. geocities.ws/conwaylife/, 2003. [Online; accessed May 6, 2016].
- [Slo96] N. 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 Jan. 30, 2020].
- [Slo99] N. 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] N. 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 Jan. 30, 2020].
- [Slo11] N. 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 May 14, 2016].

Index

Symbols

- π calculator 265
- (2,1) block pull 39, 130, 246
- 0E0P 106
- 2G-to-H 220

A

- Achim's p144 201
- acorn 16
- ADD component 260
- adjustable rake 104, 110
- aircraft carrier 31
- airforce 305
- alive 3
- ambidextrous 198
- apgsearch 26, 73, 105, 138
- APGsembly 249
- ark 14, 86
- armless construction 227
- ash 5

B

- B-heptaplet 15, 191
- B-heptomino 11, 25, 76, 89, 117
- B245 175
- B29 82
- B60 175

- bakery 27
- banana spark 58, 118, 206, 307
- barge 31, 82
- beacon 7, 62
- beehive 4, 117
- beehive stopper 194
- Beluchenko's p7 57
- BF22P 193
- BFx59H 193
- bi-block 32, 103, 117
- bi-block fuse 98
- big glider 105
- billiard table 95, 121
- binary 257
- binary register 257
- bipole 51
- blinker 5, 117
- blinker fuse 98
- block 19, 39, 117, 129
- block on table 32
- block-laying switch engine 307
- blockade 8, 27, 32
- blocker 57, 162, 206
- blockic 133
- blockic seed 135, 142
- BLx19R 191, 193
- boat 38, 117, 141
- boat with tail 35
- boat-bit 38, 165
- boojum reflector 77, 107

- bookend 37
 bouncer 107, 162
 bounding box 15, 260
 BR146H 201
 breeder 127
 BRx46B 193
 BSE22T31 193
 buckaroo 58, 107, 226
 bumper 161
 bunnies 27
 10b 16
 9 16
 burloafometer 53
 Bx106 175
 Bx202 175

C

- caber tosser 231
 Canada goose 82, 107
 canoe 35
 cap 37
 caterer 58, 305
 caterloopillar 102
 caterpillar 102, 106
 cauldron 53
 cell 3
 cellular automata 5
 centipede 102
 chicken wire 41
 child 3
 cis 35
 clock 7, 19, 120, 130, 132, 136, 140
 inserter 136, 295
 clock gun 252
 Coe ship 92, 198
 color-changing 80
 color-preserving 80
 component stack 252
 computational universality 168, 245
 computer 252
 conduit 65, 173
 confused eaters 54
 construction arm 227
 copperhead 100, 105, 303
 Corderrake 109
 Cordership 86, 102, 124, 140
 crab 82, 141

D

- David Hilbert 60

- dead 3
 DEC 246
 demultiplexer 194
 diameter 260
 dinner table 54, 59
 domino spark 55, 83, 136, 206
 dot spark 56, 83, 206
 double-barreled gun 156, 170
 duoplet 58, 306
 duoplet spark 155

E

- eater 10, 36
 eater 3 48
 eater 1 27, 31, 83, 117, 141, 142
 eater 2 48, 83, 120
 eater 3 38
 eater 5 140, 199
 ecologist 90, 91, 123
 edge shooter
 edge-shooting gun 158
 edge-shooting gun 158
 Edna 28
 elbow 224
 elementary spaceship 101
 emu 219
 engineered spaceship 101
 eureka 62
 evolution 7
 extra long 35

F

- F116 175
 F117 175
 factory 193
 familiar four 27
 fast forward force field 98, 141, 169
 fd *see* full diagonal
 featherweight spaceship *see* glider
 Fermat prime 235
 Fibonacci number 279
 figure eight 56, 61, 163
 filter 169, 206, 238
 finger spark 57
 finite-state machine 249
 first natural block 194
 fishhook 31
 fleet 27
 flotilla 85, 105
 fountain 57, 61, 170

- full diagonal 81
- fumarole 56, 121
- fuse 97, 235
- Fx119 175
- Fx119 inserter 178
- Fx77 66, 67, 175, 176

G

- Garden of Eden 19
- Gemini 102, 106
- generation 3
- glider 6, 117, 119
 - color 80, 133, 162
 - gun 11
 - lane 81
 - pusher 140
 - synthesis 115
 - timing 81, 162
- glider duplicator 149
- glider emulator 107
- glider gun 115
- glider loop 62
- glider pusher 149
- glider stopper 194
- Gosper glider gun 11, 80, 83, 211
- gourmet 60, 76, 158, 192
- grin 19
- growing spaceship 142

H

- half diagonal 81
- half-baked knightship 102
- HashLife 254, 279, 289
- hassle 59
- hassler 59
- hat 26, 122
- hd 56
 - see* half diagonal
- heavyweight
 - emulator 56
 - volcano 56
- heavyweight emulator 305
- heavyweight spaceship 7, 119, 120, 122
- hebdarole 56
- Heisenburg 159
- heptomino
 - B 89, 117
 - pi 117
- Herschel 15, 65
 - receiver 197
 - track 65, 184

- transceiver 196
- transmitter 197

Hertz oscillator 52

HF95P 193

HFx58B 193, 201

highway robber 195

hivenudger 84, 108

HLx69R 193

honey farm 27, 32, 117, 122, 224, 239

honey thieves 54, 59

honeybit 83

honeyfarm 8

HWSS *see* heavyweight spaceship

I

INC 246, 258

induction coil 36, 52

inline inverter 147

interchange 117

isotropic rules 5

iterated logarithm 234

J

jam 51, 215

Jason's p22 54

Jason's p33 215

Jason's p36 215

JavaLifeSearch 21, 28

Jormungant's G-to-H 220

K

Karel's p15 163

kickback reaction 118, 140

killer toads 84

knightship 101

Koch snowflake 272

Kok's galaxy 57

L

L112 77, 174, 175

L156 174, 175, 201

Lidka 16, 27

Lifeline 24, 46

lightweight spaceship 7, 115, 119, 122

loaf 38, 117

loafer 100, 105, 108, 135

lobster 101

long 35

long barge 35, 82

- long boat 31, 141
- long canoe 35
- long long boat 35
- long long ship 35
- long long snake 35
- long ship 35
- long snake 31
- long³ snake 35, 138
- LSE11T-8 192
- lumps of muck 9, 59, 117, 195
- LWSS *see* lightweight spaceship
- Lx163 175
- Lx86 175

M

- machine gun 188, 255
- memory cell 158, 170
- meta-blinker 289
- metacell 287
- methuselah 15
- Middleweight
 - emulator 57
- middleweight
 - supervolcano 57
 - volcano 57
- middleweight emulator 61, 306
- middleweight spaceship 7, 119, 122
- middleweight volcano 61
- mold 51
- Moore neighborhood 4, 272
- MSW-1T1 192
- MUL component 262
- MWSS *see* middleweight spaceship
- MWSS out of the blue 160

N

- negentropy 52
- neighbor 4
- new gun *see* twin bees gun
- Noah's ark 28
- NOP 259
- NZ 246

O

- oblique spaceship 101
- octagon 2 51
- omniperiodic 69, 96
- one-time turner 131, 194
- onion rings 41
- Orion 82

- Orion 2 82
- orphan 21
- oscillator 6, 51
- OTCA metapixel 288
- overclocking 179
- overweight spaceship 85
- OWSS *see* overweight spaceship

P

- p1 megacell 288
- parent 3
- pentadecathlon 7, 55, 119, 122, 146
- pentoad 54
- period 6, 51, 79
- period multiplier 186
- PF81H 193
- phase 6
- phoenix 71
- pi portraitor 305
- pi-heptomino 9, 59, 76, 106, 117, 216, 312
- pinwheel 305
- pipsquirt 56, 75, 162, 208
- PL8P 192, 193
- PNW6T138 193
- polyomino 8
- polyplet 25
- pond 117
- PR9B 193
- pre-beehive 4, 27, 59, 84
- pre-block 19, 53
- pre-honeyfarm 8, 59, 306
- pre-pulsar 61
- pre-pulsar shuttle 62, 171
- prime number 150, 279
- primer 150
- pseudo
 - spaceship 84
 - still life 32
- pseudo-period gun 211
- puffer 13, 86, 89
- pulsar 6, 61, 119
- pushalong 84

Q

- quadri-Snark 200, 255
- quasi still life 47
- queen bee 9, 119, 122
 - shuttle 10
- queen bee shuttle 58, 140
- quetzal 219

quetzalcoatlus *see* quetzal
 quinti-Snark 200

R

R-pentomino 15, 25, 119
 R126 175
 R64 65, 66, 175
 rabbits 27
 rake 89, 103, 115
 rattlesnake 57, 61
 READ 258
 read head 258, 272
 recovery time 37
 rectifier 75, 77, 107
 recursive filter 232
 reflector 62, 63, 80
 stable reflector 64
 regulator 164
 relay 107
 repeat time 64
 rephaser 108, 222
 rewind 169
 RF28B 191, 193
 RF29P 193
 RFx36R 193
 Rich's p16 74, 125, 141, 216
 RNW3T46 193
 Rob's p16 74
 roteighter 54
 row printer 265
 RR56H 193
 rulestring 4
 Rx140 175
 Rx164 175

S

salvo 129
 SBR *see* sliding block register
 Schick engine 91, 105, 111, 303
 Scientific American 24
 scrubber 305
 seed 129, 143
 semi-Snark 186, 203
 SET 258
 ship 43, 122
 shuttle 59
 sidecar 84
 signal 94
 elbow 95
 Silver reflector 64

Simkin glider gun 200
 sink 95
 Sir Robin 102
 slide gun 225
 sliding block register 246
 slow glider pairs 227, 228
 slow salvo 227
 snacker 53, 55, 75
 snake 31, 38, 122
 Snark 64, 77, 80, 131, 196, 309
 soup 5
 source 95
 space rake 90
 spaceship 6, 79
 spaghetti monster 100
 spark 55, 161
 sparker 55
 speed 79
 speed of light 79
 spider 100
 splitter 252
 sqrtgun 231
 stable 36
 stairstep hexomino 8, 25
 still life 3, 31
 strict still life 141
 SUB component 260
 superfountain 57, 216
 supervolano 216
 supervolcano 57, 76
 switch engine 12, 15, 83, 86, 115, 120, 124, 140
 block-laying 13
 glider-producing 13, 119
 syringe 179, 184, 199

T

T-nosed p4 58
 T-tetromino 8, 25, 59, 60, 122, 124
 table 37
 tagalong 82, 84
 tail 35, 53
 tandem 197, 222
 Tanner's p46 60, 158
 TDEC 247, 258
 teardrop 117
 tee 141, 142, 207, 227
 TEST 246
 thumb spark 57, 83, 162
 tick 3
 ticker tape gun 156

toad 7, 84
 toggle 149
 TOLLCASS 26
 tractor beam 233
 traffic jam 77, 217
 traffic light 8, 27, 117, 122
 trans 35
 transparent 177, 247
 tremi-Snark 200
 trombone slide 183
 true-period gun 211
 tub 82, 122
 tub with tail 35, 122
 tub with tail eater 37
 tubstretcher 83, 107, 141
 tumbler 303
 Turing complete 245
 twin bees 11, 120, 140, 141
 shuttle 12
 twin bees gun 12, 80, 118, 211
 twin bees shuttle 58
 twin prime 169, 235
 twin primer 166, 169
 TWIT *see* tub with tail eater
 two blockers hassling R-pentomino *see*
 Wainwright's p72
 two eaters 53
 two-glider mess 117

two-time turner 142

U

unary 257
 universal regulator 164, 254
 unix 57, 162, 306

V

vacuum 94
 variant 176
 very long 35
 volcano 57
 von Neumann neighborhood 4

W

Wainwright's p72 201
 weekender 100, 105
 welding 39
 wick 97
 wickstretcher 83, 235
 wire 94
 worker bee 54

Z

Z 246
 zebra stripes 41, 94