16 Nov, 2022

Building arbitrary Life patterns in 15 gliders

The Conway's Game of Life community celebrated a landmark achievement on November 9th, 2022. An idea years in the making, the "Reverse Caber Tosser" design finally had all of the pieces it needed to achieve its stated goal.

That goal is simple. Select any pattern that can be built in Life – for example, the Waterbear. Begin with a small number of gliders (now 15) in an otherwise empty Game of Life universe. After enough time goes by, those gliders need to build that pattern. No extra leftover debris, no stray

This post will talk about how it works, how we got there, and why it's so cool.

How is this even possible?

From an information theory perspective, an arbitrary selected pattern can have unbounded amounts of complexity. Every one of these patterns needs a different initial set of gliders.

With 15 gliders (or any fixed number), the only difference between one initial set and another is where they start. Since we have infinitely many patterns to create, we must list infinitely many 15 glider arrangements. This is possible, but only by taking advantage of a correspondingly unbounded amount of distance between at least one pair of gliders in the recipe.

With that constraint comes some hope. The fact that we need to use a lot of distance, guides us to our first principle. Distance itself encodes information.

The prompt to get people thinking about this came in 2015, in a truly bizarre fashion.

Questionable beginnings

It began with Gustavo, a Conway's Game of Life enthusiast from Brazil who had a tendency to go far off topic in the forums. In a new thread, Gustavo rambled about a secret leaked Morse code signal from MIT he was decoding, which seemed to be discussing "Game of Life – Spaceship Synthesis Research". The community interaction with this thread was honestly hilarious. Some people trying to humor Gustavo, asking for

Everything about this story was imaginative nonsense, with no substance. User “gameoflifeboy” described it as “the weirdest thread on the forum so far in 2015”. But one part stuck out. Gustavo said that through his eavesdropping, he learned that a previously unknown spaceship, larger than the Caterpillar, had a synthesis in 386 gliders.

Adam Goucher, a mathematician of some notoriety and author of the cp4space blog, treated this as plausible. Not plausible in origin, as the story was just the brainchild of an internet troll. But rather, plausible that a super-complex pattern can be synthesized in a relatively small number of gliders. In his words:

> **As long as we can synthesise a universal constructor in 385, the entire blueprint could be encoded in the *distance* between the final glider and the constructor (by Godel coding or otherwise).**
>
> — Adam Goucher

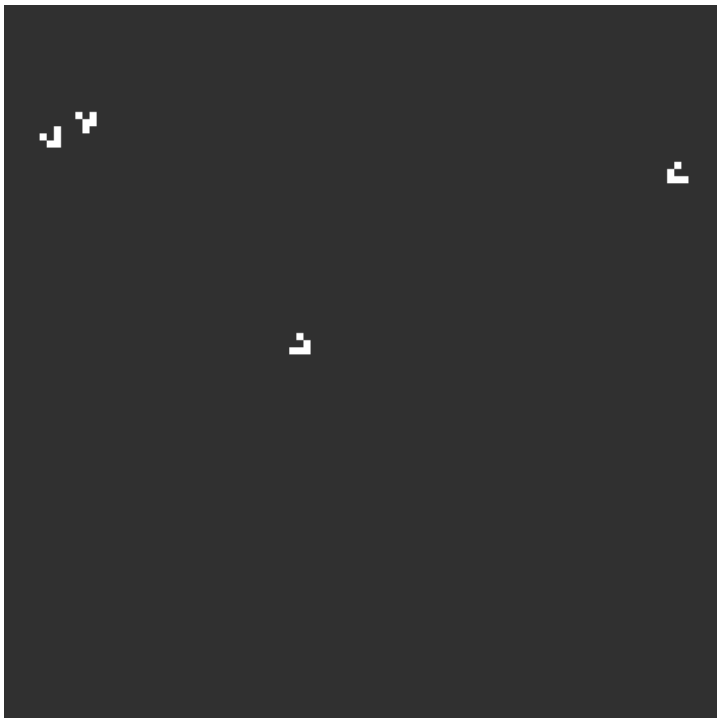
Universal constructor

Universal constructors came up in my Waterbear post as well. I’ll try to treat the topic a little better this time around, by analogy to Turing Machines.

One often hears that a Turing Machine is just as powerful as a modern computer, in that it can run all of the same algorithms (but maybe taking more time). And similarly, one often hears that much weaker systems, like Rule 110, are just as powerful as a Turing Machine. Despite being

equivalent computational ability, is deemed “Turing Complete”.

In shifting back to Conway’s Game of Life, replace computation by construction. Systems are equivalent if they can build all of the same things, regardless of efficiency. For a canonical baseline, we can use arbitrary collections of gliders converging from infinity, like in the pattern below synthesizing a “weekender” spaceship.



This easy-to-watch synthesis takes 33 gliders. There is now a footnote on the corresponding wiki page saying that it can be constructed in fewer using RCT.

While gliders colliding aren’t as powerful as just scribbling a pattern of your choosing (notable examples), they’re pretty darn close. And so a universal constructor can be regarded as any system (glider-based or otherwise) allowing a recipe for every possible glider construction.

Example

single shared lane. Watch through to the end to see how it could build items far away from that lane.

0:00

This serves to show that gliders along a single lane are, in a sense, “just as powerful” as synchronized gliders coming from all directions. The two systems can both build the same things. It definitely takes more gliders when they are all on a single lane. But, it has the same eventual capability.

In order to prove this, the community needed a process to convert a recipe from one system into another. The reduction looks like this:

> Build synchronized gliders from multiple directions by creating a “seed constellation”, made of a bunch of objects that turn into gliders when the reaction reaches them.
>
> Build a seed constellation by sending gliders one at a time towards a target (when gliders are not synchronized, this is called a “slow salvo”).
>
> Find a way to use an “elbow” to convert incoming partially-synchronized gliders from the single lane, into a desired slow salvo.

This is just a taste of the jargon to follow. But in principle, it makes sense. Seed constellations are as powerful as glider collisions. Slow salvos can make seed constellations, so they too are just as powerful as glider collisions. And construction

down.

Shortcuts

A lot of effort in the past years has gone into directly creating slow salvo recipes for important objects. Notably, in the previous video, there wasn't much of a seed constellation at all. Instead, the big important object in the reflector emerged from a single glider reaching the existing carefully-constructed mess.

However, if you want to get it done quick, a lot of the times making any old recipe will go through the whole reduction. See below for an example seed constellation building Alan Hensel's famous 1994 "decimal counter" pattern.

At a further zoom, once built:

The sequence of digits is encoded in the loops of gliders at the bottom. This pattern from 1994 is often regarded as the progenitor of all “display” patterns, which have come a long way since.

Back on topic

Now we have seen a couple examples of universal constructor systems, and a couple examples of interesting things they can construct. Everything you have already seen, can now be made in 15 gliders.

The weekender. The specific collection of 33 gliders that I showed making a weekender. The collection of thousands of single lane gliders with

Ultimately, the achievement of the Reverse Cabel Tosser (RCT) project is that we can now build all of these things cleanly, using 15 initial gliders.

Synthesizing a universal constructor

It's slightly paradoxical to consider a "synthesis" of any of the current examples of universal constructors. These systems embed the recipe for the Actual Thing Being Constructed, in the spacing or position of an unbounded number of objects, frequently gliders. Naively, each instance of a construction therefore takes a different number of gliders, failing the goal.

Or does it?

It's not unheard of to use some small number of gliders to create a larger number of gliders. As a rather trivial example, the Gosper Glider Gun takes 8 gliders to build, and 2 to destroy. Letting it run for longer before destruction will create more gliders. But, the gliders out of a gun like this are highly regular, and won't encode any more novel system of construction no matter how long you let it run.

What if..

What about a design that somehow built and destroyed every possible thing, in order, until an interrupt arrived that told it to stop on the current thing? It's a nice thought experiment, but it quickly runs into problems. The task of destroying a thing without knowing what it is, is hard. What's more, some things include spaceships that escape at high speed. Nothing you could build would be able to go catch and destroy it.

No, what was needed was a way to read information from the "far away" stuff multiple times, and have each read communicate new information. In no way would a single read suffice.

The Caber Tosser

Way back in 1991, Dean Hickerson built a pattern which bounced a glider back and forth between a stationary object and a distant spaceship. Each time, the length of the trip would double. Something similar was used in the recent tetration machine, but Hickerson's old design was built for niceness, rather than anti-optimized for slowness.

“cordership”, moves diagonally 8 tiles every 96 ticks, or 1 per 12 on average. If the original distance between them is *D*, then the amount of time it takes for the glider to catch the cordership is *D* / (1/4 – 1/12), or 6*D* generations. Then the glider turns around, and covers the same distance to get back to where it started, another 6*D* generations. In total, 12*D* generations elapse, moving the cordership *D* tiles further away, a clean doubling.

With the stationary object configured to do something each time it reflects a glider, the result looks like the below.

Here we start to see the problem with watching things at these scales..

Each glider sent out takes twice as long as the previous. This gives logarithmic population growth, which is cool on its own.

In reverse

Now consider what happens if the cordership, instead of moving away from the rest of the machine, moves towards it. It still can bounce a signal back and forth a number of times.

A two-glider signal reflected from an approaching cordership into two different gliders in the other direction

But now there's a limit. Eventually the cordership arrives, and the whole thing goes splat.

What you get is a series of signals, each one taking half as long to arrive as the previous, until the distance can no longer be halved.

Bit-reading

What's more, the new glider doesn't need to go back to the cordership right away. It could do so with a slight delay, using some synchronizing circuitry. Perhaps it could do so in a way that "rounded down" the previous halving.

If we call a glider we need to round down a 1, and one that we don't a 0, then the original position of the cordership gives us a list of 1s and 0s. A finite list of 1s and 0s is a potential recipe. Just as we hoped, there's a way that distance encodes information. And not just information, but discrete bits that are read at separate times.

This is the Reverse Caber Tosser, or RCT. The name refers to any system that bounces a signal

The messy details

Now all that we needed to do was

> Find a way for these 0s and 1s to function as a universal constructor
>
> Find the simplest glider recipe for a system with Reverse Caber Tosser properties
>
> Build a recipe with the 0s and 1s that not only made the Actual Thing Being Constructed, but also cleaned up the entirety of the RCT's mess

As you could probably guess, not one of these tasks was simple. In most cases, the community focused only on the first two bullet points, leaving the latter as a theoretical certainty. As a result, most of the original designs were incomplete, never building an Actual Thing. But the community tracked the glider count anyway.

Incomplete design #1: 329 gliders

Dave Greene and Adam Goucher were the first to check off bullet point #1. Having ample experience with construction arms from his previous ambitious projects, Greene stripped the concept down to a barebones system of PULL and DFIRE. A PULL would grab the next block in a long line of blocks, and move it closer to the machinery. A DFIRE would destroy that block, and send a glider in a perpendicular direction towards the construction zone. This was messy, but all of the mess was stationary objects. Importantly, there were no rogue spaceships at any point. And the gliders being sent towards the construction

With a simple proof of concept pattern that sent a couple of each of PULL and DFIRE, they announced success. A short while later, “Goldtiger997” came up with a glider synthesis of this pattern, in 329 gliders. That’s bullet point #2. Every combination of PULLs and DFIREs could be generated by moving the handful of gliders that built the cordership, back to the proper distance.

In Dave Greene’s blog post about it, he remarks that in order to actually succeed at its goals, the PULLs and DFIREs needed to build a lot of self-destruction circuitry right before the cordership arrived with a splat. That’s bullet point #3. Nobody made this, but they were content to believe it possible.

Getting more minimal with Switch Engines

For a little bit more game of life jargon, the Switch Engine is the core reaction of the cordership. Discovered back in 1971 by Charles Corderman, when Conway’s Game of Life was just a year old, it travels 8 cells diagonally every 96 generations. Unfortunately, a single switch engine leaves enough extra junk behind that it eventually succumbs to the void.

The corderships combine multiple switch engines in a way that cleans up the junk. However, there are two ways to stabilize a single switch engine.

The first is the “block-laying switch engine”. By adding a single block to a switch engine, it becomes stable. The extra junk becomes a simple diagonally repeating pattern of blocks. This is the mechanism used in the 329 glider RCT to provide arbitrarily many blocks with a finite recipe.

The second is the “glider-producing switch engine”. This has a much bulkier ash pattern, which includes a consistent barrage of gliders in the direction of travel.

The block-laying and glider-producing switch engines are the two most common infinite growth patterns in the game of life, in terms of frequency emerging from random initial conditions. The glider-producing switch engine in particular, became the star of the show in RCT reductions.

Cheaper, Messier, and Just as Capable

A glider-producing switch engine (GPSE) moves at the same speed as a cordership, and creates its own predictable stream of gliders. The first reductions to the RCT's glider cost came from replacing most of the stationary circuitry by glider streams shooting from GPSEs. Twelve GPSEs in total could replace the entire decoder mechanism, leading to a much more resounding splat at the moment of convergence. Since a GPSE takes only 4 gliders to synthesize, and the decoder mechanism was the most expensive part of the synthesis, this slashed the cost of the RCT to 59 gliders. This reduction offloaded *even more* work to the embedded recipe, as now it had 12 ash trails to clean up.

Other insights during 2018 reduced this to 35 gliders, with some changes in the decoding

afterthought, not completed.

Over the next two years, insights from Chris Cain and Adam Goucher saved 3 more gliders, bringing the cost to 32. Then in September 2020, a new contributor made a shocking improvement.

MathAndCode

Daniel Vargas, also known as MathAndCode, joined the forums in late August 2020. His first post was in the thread discussing the RCT project. He asked why the approaching object encoding the distance needed to be a cordership variant. Why couldn't it be a single GPSE? A GPSE would be cheaper and more versatile.

Adam Goucher supplied a good answer, but not a perfect one. There were technical challenges because of the higher period of the GPSE. It seemed like it would be more difficult to get working. MathAndCode wasn't dissuaded from trying.

Less than two weeks later, he posted "Potential universal construction in 16±2 gliders".

The key reactions

The GPSE indeed has a higher period than a cordership – it takes 384 ticks to complete an orbit rather than 96. Due to the doppler effect, using a GPSE to handle incoming gliders from an antiparallel GPSE requires that two different interactions, 192 ticks apart, both return a signal.

Now, if you send a glider towards an oncoming GPSE as you would towards a cordership in the original RCT, a few things might happen:

It emits a stray glider into the void which could never be cleaned up (bad)

It collides with some of the ash left behind, returning a new glider back the way it came (good)

It removes/prevents creation of one glider from the GPSE's stream (also good!)

The last bullet point is good because a **hole** where you expect a glider, is just as useful of a signal as a glider.

MathAndCode found a key pair of interactions. One of them returned a glider, and the other 192 ticks apart returned a hole. Not only did this allow a GPSE to replace the oncoming cordership, it solved the "rounding down" problem outright. The circuitry, if it could even be called that anymore, became even simpler.

Four GPSEs and a target

With this redesign, the number of GPSEs needed to create a functioning RCT dropped to four.

One that encodes the recipe.

One that supplies the signal gliders to interact and read distance parity.

Two more, at a 90 degree angle, to coordinate the circuitry, clean up all of the other gliders, and supply construction commands for 0 and 1s.

When MathAndCode made his post, it wasn't entirely clear that the available 0 and 1 commands were a true universal set. But Adam Goucher did some automated searching through the binary strings, and found hope. Without making any unrecoverable messes, there were all of the following (text copied from Adam's post here):

allow arbitrary-integer fd moves;

a nondestructive glider emission reaction;

a destructive block creation reaction which throws the block beyond any of the lanes used for either of the above;

a 14fd pull which can follow an elbow-duplication operation.

This was a universal set. Since the four GPSEs took 4 gliders apiece, and the initial target object required one more glider, MathAndCode had dropped the magic number from 32 to 17. The community celebrated. People pondered whether there would be any additional snags getting the universal constructor to actually clean up the whole mess. But it all seemed possible in theory. Right?

A nagging question about cleanup

How is it possible that the messy GPSEs, leaving a smear of debris across space, could ever be cleaned up? Every bit read by the constructor required **doubling** that amount of debris, so we needed a solution that didn't care how much debris there was.

The solution for that problem was corderships. The constructor could build corderships that naturally deleted objects in the debris stream when they passed by. A finite amount of work to handle an unbounded amount of debris. The constructor could also build "corderabsorbers" at the furthest point of the debris stream, which made that cleanup crew self destruct at the end. It didn't matter how far that furthest point was, because there were already some unique objects at the original location where the GPSE was created. The constructor could leverage those as glider targets to start building.

read a heavily-delayed signal. Also, the spots in the encoding GPSE's stream where the bit reads happened had very different debris, so the cleanup had to be able to handle that. Still, the community understood a theoretical solution to all of the problems.

Theory and premature optimization

The rest of the task was connecting "we have a universal constructor" to "here is the gargantuan task for this universal constructor to accomplish". Similar to finishing the Waterbear, this is a strange mix of busywork and pioneering development. It's like rote programming in a language nobody has ever used before.

The community enjoyed optimizing something that didn't even exist yet. Even more acronyms sprung forth, the most important one being the Decoder and Better Construction Arm, or DBCA. In effect, the DBCA was the only thing that the universal constructor actually built using its universal set. Once built, the DBCA directly consumed those same 1s and 0s. It collected them into 9-bit codons to do far more powerful operations using a far more powerful construction arm. By spending a lot of bits to build something it didn't need to build, the RCT now could get a lot of mileage out of every additional bit. The cost more than paid for itself, even just during the process of building the cleanup corderships. The community was confident that this was the right choice for minimizing the total number of bits needed in the recipe.

Technological limitations

of 2 . By the time the DDCA was constructed, the population of the life universe (mostly GPSE debris) would be of a similar magnitude. The best Game of Life emulator could maybe handle populations and generation counts of order 2^{10000}, but this was well past its limit.

In the interest of actually being able to demo this accomplishment, the community came up with a technique called the “semilator”. Added to the initial recipe were two streams of orthogonal spaceships. Every time a pair arrived, it would add a new output bit to the constructor, without changing the bouncing RCT glider signal at all.

The demo patterns became 28 bits of “proper” RCT, with a million or more injected bits in between the second and third RCT signal. Everything was adequately spaced to function the same as if the bits were truly read from the RCT, but now it could fit in Golly.

Over the finish line

Dave Greene, with admirable patience organizing the myriad trains of thought present in this community, made an official checklist of remaining problems for the project. Pavel Grankovskiy, (Pavgran, who was also the main designer in the Conway’s Game of Life portion of my tetration post), methodically checked items off the list, one after another.

On November 9th, Pavgran shared a semilator pattern for the 16-glider construction of the Decimal Counter. All of the careful cleanup was now a reality.

Counting is hard

Well, the technology was effectively the same throughout. But, during 2022, a user known as dani or Danielle began seriously and systematically investigating switch engines. She found a way to synthesize the pair of GPSEs controlling all the circuitry and construction, together in only 7 gliders instead of separately using 8. This led to 16 being the magic number during the bulk of the checklist. After posting the complete demo for RCT 16, Pavgran worked with dani to find yet another reduction, now reducing the price of making the initial target. One of the alternate 4-glider GPSE recipes ejected a stray glider, which another GPSE stream could intercept similarly to the extra glider in the older design. The first handful of bits needed to be updated, along with some of the cleanup logic, but the rest of the design would function just as well.

4 in the top left, 4 in the bottom right, and 7 in the bottom left, for 15 gliders in total. In between them, some over-500,000-digit number of tiles of empty space.

Breakdown of Bits

For the updated 15-glider construction of the Decimal Counter, there are 1665791 bits. Most are supplied by the semilator, but the first 2 and final 26 come from the RCT mechanism. With a different initial position, all bits would come from the RCT. (The pattern would also be so large as to need a new specialized emulator to be able to view it).

The high level breakdown of how those bits are used is as follows:

> 1274729 – build a DBCA and pass control to it
>
> 192584 – build a new constructor that reads stored data instead of live data
>
> The final 200093 bits get stored in the Binary Storage and Retrieval device, these same 200093 bits are counted below:
>
> The Big Splat happens, everything afterward is read from storage
>
> 133900 – lots of cleanup
>
> 58173 – build the Actual Thing Being Constructed (will be different number of bits for different ATBC)
>
> 8020 – constructor finishes cleanup (including itself!) in a way that activates the seed constellation, leaving only the ATBC behind

Notably, building the DBCA takes up roughly 75% of the total bits. However, the remaining 25% of the bits accomplish 80% of the total work, as measured by number of slow-salvo gliders. This reflects the drastic efficiency improvement of using a better construction arm.

Something to watch

considered here is effectively unfathomable. Fortunately, some of the community members involved in this task have done their best to make a demo-viewing demo script.

I've recorded it below. Enjoy!

0:00

Achievement Complexity Computation Conway's Game Of Life

4 thoughts on "Building arbitrary Life patterns in 15 gliders"

Building arbitrary Life patterns in 15 gliders - My Blog

Nemo says:

2022-11-22 at 1:09 am

Hello. Quite an impressive achievement. But isn't the thought that the distance can only encode "a single 0 or 1 at a time" quite an inefficient relic from programmer/chip designed thought?

Conway's game isn't limited to purely binary circuitry. One could build a sort of "analog" clock construction, which counted time spent since sending a signal. As distance can be an arbitrarely large number, then the number generated from "counting the distance" could be used (along with a decoder) to encode the instructions for any constructible structure.

This allows potentially to use much less generations and resources than the "rounding down into a single 0 or 1 each time a glider returns" method. You would only need one glider and a "clock" to represent any number with an initial distance.

Reply

Nemo says:

2022-11-22 at 1:27 am

Also and furthermore, going with my "clock" idea, the distance stuff doesn't even need to be gliders. It can be a static 4-block , since all we need is for it to be far away, and the clock will send the glider and count how far away it is. Alternatively, if the

automatically after being created, and have the glider arriving stop the count. The number can be arbitrary in regards to how far away the initial glider (or block) is.

Reply

Biggiemac42 says:
2022-11-22 at 3:34 pm

Thanks for writing!

Something like this is covered in the "What if?" but the issue complicating things is that you need to read and decode multiple chunks of information at different times. One read, even if it contains a lot of information, won't spell an unambiguous recipe without a massive decoding scheme, comparable to the bouncing already happening here. 15 gliders (assuming the goal is just to minimize that number) would be hard to beat. If the goal is instead to find a larger fixed number with a different-and-possibly-simpler approach, sure I could believe it possible.

In any rate, I don't think the concept of "single 0 or 1" is a relic

Leave a Reply

Your email address will not be published.
Required fields are marked *

Comment *

Name *

Email *

Website

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT