A Turing Machine In Conway's Game Life.

Paul Rendell

I have constructed a Turing Machine in Conway's Game Life (figure 1). In this paper I describes the machine's parts, how it works and the principle choices made during the construction.

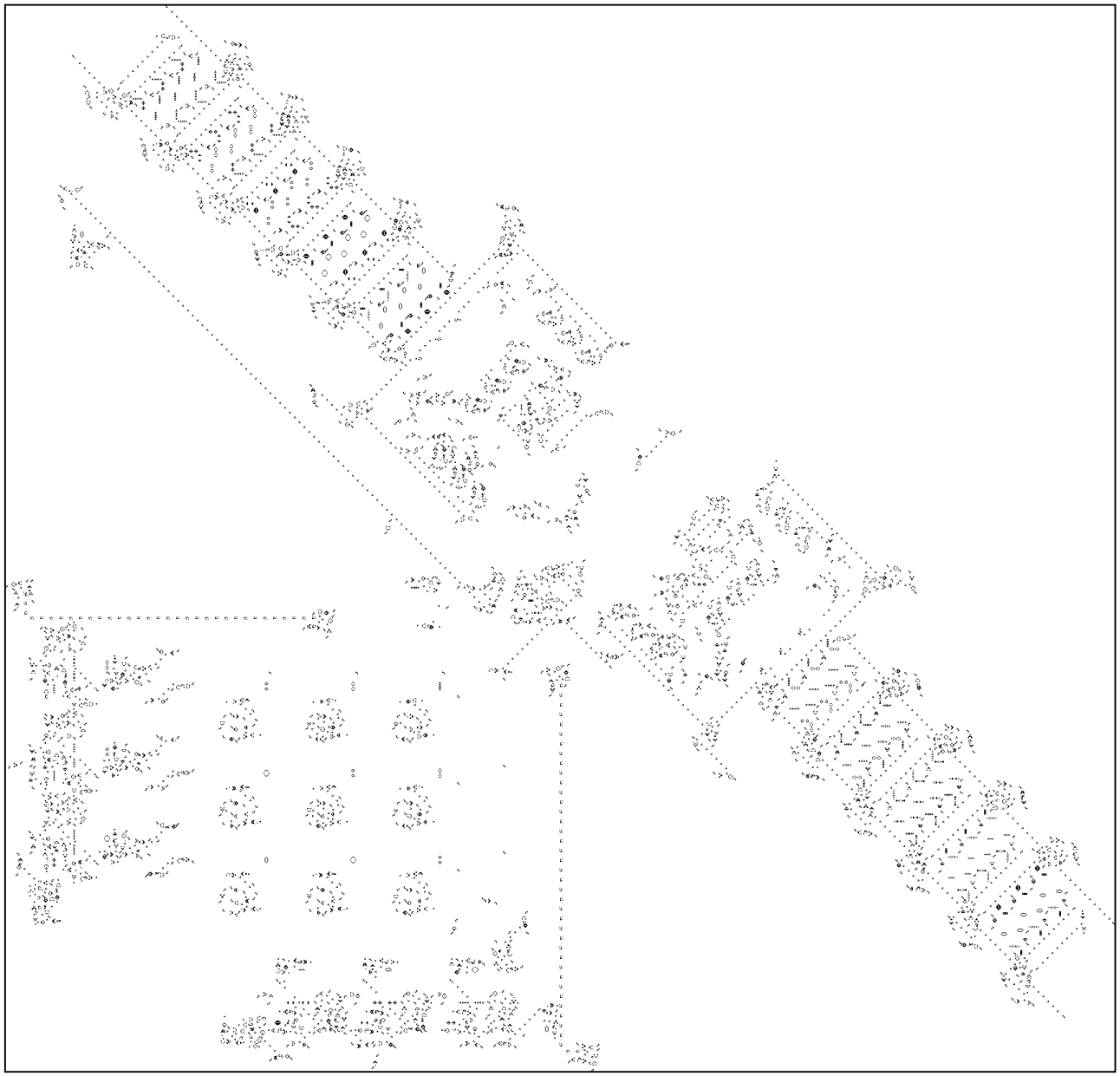

Figure 1The Complete Turing Machine

The Game of life was invented by John H Conway. It is a cellular Automa, a class of simulation, where an infinite universe is divided into cells. Each cell has a state and the states change according to strict rules. All the changes occur simultaneously to create time measured in generations. For Conway's Game of Life the cells have 2 states (live and dead) and the rules are based on the number of touching cells which are live. If a cell is live and has 2 or 3 live neighbours then it will stay alive otherwise it will die. If a cell is dead and has exactly 3 live neighbours it will change to live (be born).

I have looked at variations on the rules to see what happens and found that some sets cause patterns to disappear quickly and others cause patterns to expand and fill the universe. The Game of Life rules seem to be close to the boundary between. Most simple patterns are stable or will shrink to stable patterns quickly but a few expand.

A pattern called a glider was quickly found. This pattern repeats itself every 4 generation but offset diagonally by one cell. A family of 3 patterns which move orthogonally by two cells in 4 generations was found by Conway. These are called the Light Weight Space Ship (LWSS) Medium Weight Space Ship (MWSS) and Heavy Weight Space Ship (HWSS). The term space ship is now the generic name for moving patterns.

The pattern which really opened things up was is the glider gun, found by Bill Gosper. It generates gliders every 30 generations. Now guns have been found which produce gliders at most periods above 14 [2]. There are now more than 700 basic patterns, which have now been named and are described in Stephen Silver's Lexicon [4].

A Turing Machine is a mathematical device invented by Alan Turing. He wanted the simplest possible theoretical computer to use in mathematical statements. He used his Turing Machine to prove a number of important things about computing, most famously, the halting problem. It is often possible to determine if a particular computer program with a particular set of input data will run forever or halt. Turing proved that there is no mathematical procedure for doing this in a finite time for every possible program and its data.

A Turing Machine has a potentially infinite tape to hold the input data and to store the results. The machine has a read/write head which moves along the tape and reads one symbol and replaces it by another. The actions of a particular Turing Machine are determined by a program in the form of a Finite State Machine. For all combinations of state and symbol that could be read from the tape the Finite State Machine description gives:

- the new state
- the symbol to write to the tape
- the direction to move (left right or Stop)

Turing found that it was possible to create a Universal Turing Machine. This machine requires for its input a description of a particular Turing Machine and the data that that machine would use. The universal machine then simulates the particular machine. I have designed my Turing Machine so that it can be extended to allow a Universal Turing Machine to be implemented. I have however built a smaller and quicker version.

It was proved a long time ago that a Turing Machine could be constructed in Life. This proof is based on the fact that simple logic can be performed and therefore constructs using this logic can be built. The universe is theoretically infinite so these constructs can be infinite in size to support the infinite tape of a true Turing Machine. I have now constructed such a machine.

My interest in Conway's Game Life took off when I wrote a Game of Life program for my Amstrad CPC 64 a Z80 machine with 64k of memory. I used the program to develop my understanding of Z80 machine code. The resulting program had a closed universe of 64*124 cells and the ability to merge patterns for very rapid manual searches. I used this to find most of the logic patterns required for the Turing Machine. My starting point for patterns was Martin Gardener's articles in Scientific American [6]. The most significant pattern I made was the memory cell which took quite a lot of effort to fit into this small universe.

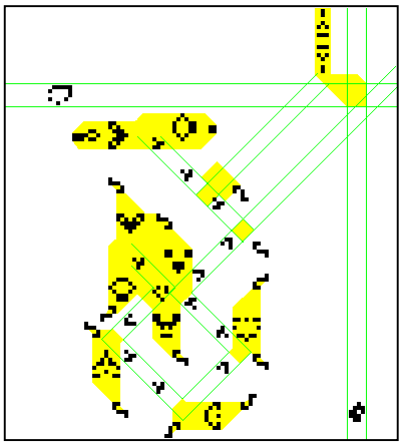

Figure 2 Memory Cell

Figure 2 is the version of the cell adapted for the Turing Machine. It is addressed by the MWSS going across the top and the LWSS coming up the side. The Pentadecathlon in the top right tidies up the collision to leave a glider which opens an 8 glider hole in the gate to the cell. The gate is the glider gun across the top firing down to the right and blocking the output of the cell. The heart of the cell is the fanout pattern in the centre. This pattern uses two queen bees to reflect glider streams. They are arranged back to back so that the queen bee reflecting the output of the glider gun on the left stabilises the one reflecting the input signal. In the process the input signal is duplicated. For the memory cell 3 standard queen bee reflectors are used to loop one output back to the input so that the pattern in the loop repeats forever.

In 1999 I got a PC and looked on the internet to see what other people had done. I was astounded by the fantastic patterns which had been found and the size of universe which could be explored by the freeware programs like Johan Bontes Life32 [3]. Amongst the patterns were some simple patterns that I had missed like a Period 30 LWSS and MWSS gun. With these I knew that I had all the parts needed to build an array of my memory cells which could be

addressed like a computer memory and return the contents of the addressed cell. Using the value to do something and obtaining a new address to fetch a new values etc. is the next step to building some computing device, a potentially continuous loop of activity driven entirely by the data in the cells. The simplest such device is a Turing Machine. The idea became more compelling the more I realised how simple it could be. It would not need lots of registers or complex operation codes just a tape.

Having decided that I would attempt to build a Turing Machine I started with the must uncertain element, the Tape. I chose to build the tape from two stacks so that to move the Tape past the read/write head would require one stack to perform a pop and the other to perform a push. With this arrangement there is no representation for the piece of tape with the current symbol on it. The machine replaces this symbol by pushing a symbol onto one of the stacks at the start of the cycle.

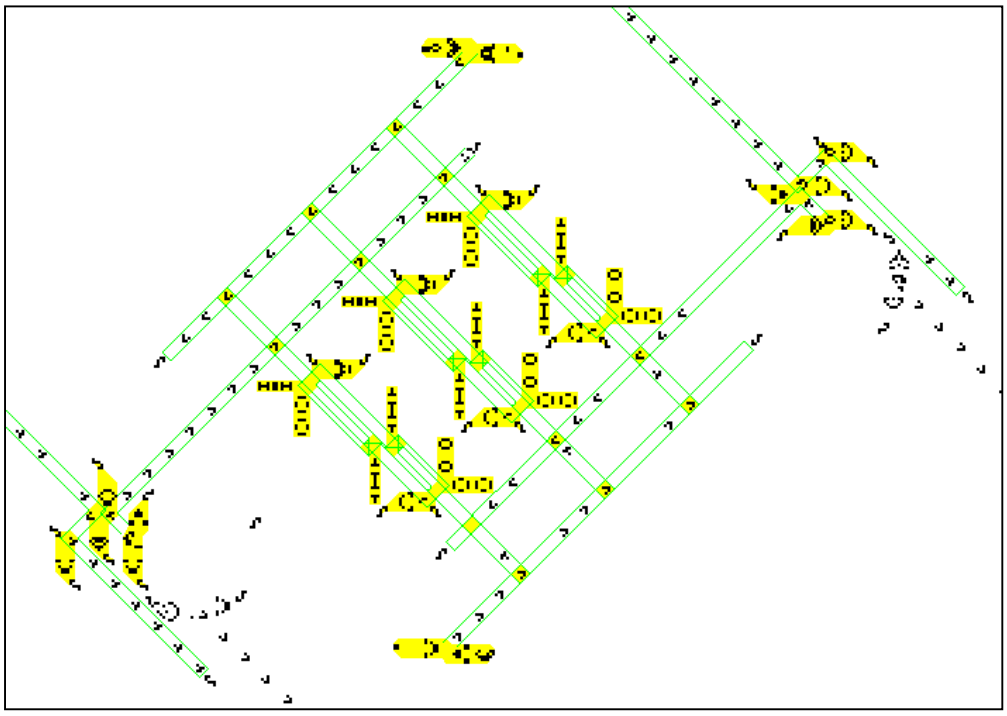

Figure 3 Stack Cell

I chose a design which traps the symbol gliders between two opposing glider streams. The method I found of delaying the symbol gliders during the push operation was to use pentadecathlons to create a convoluted path to the next cell. The same arrangement is required for the pop path to maintain the alignment. Figure 3 shows two cells and the delay mechanism. This shows the pattern during development. The logic to duplicate the streams of gliders keeping the symbols in one cell is not complete and has not been applied to the outer walls. In figure 3 the lower cell is about to let its symbol out through a gap 4 gliders wide and the hole to let the first glider into the top cell can be seen on control signal on the left.

In order to ensure that the design can be extended to include a Universal Turing Machine I needed an example of one. I found 2 in Marvin Minskys Computation Finite and Infinite Machines [5]. The smallest requires 4 symbols and 7 states and the larger, more straight forward design, uses 8 symbols and 23 states. The small machine requires a lot of tape and therefore it is quite probable that an example pattern using this machine would be much larger and much slower to run than the same example using the large machine. I examined the large machine and found a few tricks to reduce it to 8 symbols and 16 states. This allowed me to fix the meaning of the contents of the memory cells as follows:

- 4 gliders for the next state
- 3 gliders for the symbol to write
- 1 glider for the direction (left or Right)

When all 8 gliders are missing the Turing Machine will stop.

The design for the Finite State Machine was to use two period 30 MWSS guns. One modulated by the next state to select the row and one modulated by the symbol read from the tape to select the column. The pattern at the foot of each column and the end of each row had to recognise an address and generate an MWSS or LWSS which then go through the matrix and hit each other by the selected cell.

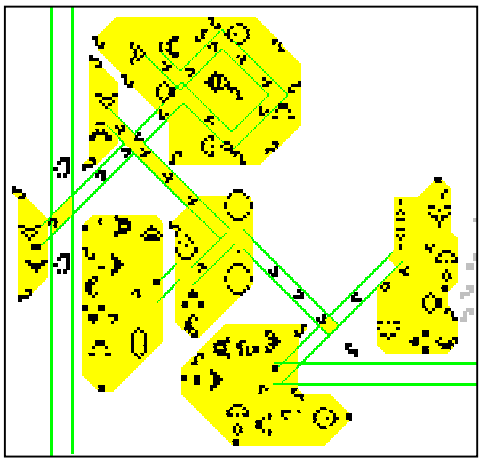

Figure 4 Row Selection

Figure 4 shows the version for selecting a row. The gliders from the glider gun to the left are destroyed by the MWSS of the address stream but survive if an MWSS is missing. The resulting pattern is compared with the contents of the memory cell at the top. The comparison is made by a head on collision, the results of which are sensed by the output of another glider gun. Its gliders are destroyed by a mismatch. When not destroyed these gliders form the reset leg of a set reset latch in the centre of the pattern. The set leg is the inverted output of a P240 gun. The latch output is sensed at the end of the address cycle by another P240 gun on the right. If the glider from this gun is not destroyed by the output of the latch it triggers the pattern at the bottom to generate an MWSS.

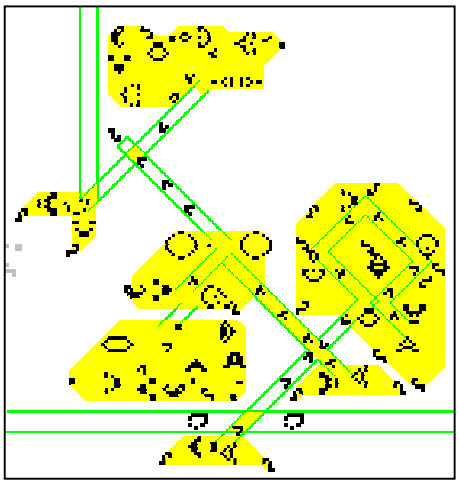

Figure 5 Column Selection

The only difference required for the column address is to replace the final MWSS generator by a LWSS generator as shown in figure 5.

The latch design exploits the two collision modes of two P30 glider streams meeting at 90 ° and out of phase. Gaps in one stream switch the mode so that the head of its gliders interact with the tails of the other streams gliders. Gaps in the other glider streams switch the mode back. In this version both modes are stabilised with a pentadecathlon. One mode produces the output gliders.

I wanted a design for the finite state machine which allowed additional rows and columns to be added easily. I chose to use a 240 generation frame to match the cycle time of the memory cells. I added an address present mark to the row and column addresses so that the comparitor for row and column 0 could distinguish this address from a frame with no address in.

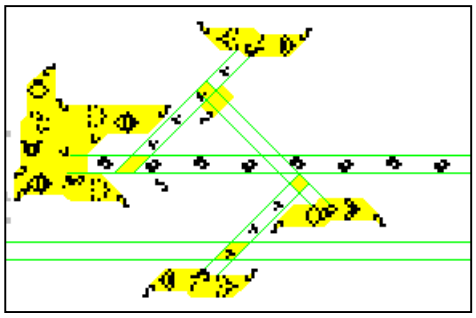

Figure 6 Row Data collectors

Figure 6 shows the method chosen to collect the output from the selected memory cell. This detects the MWSS generated by the row address comparitor and uses this to make an 8 glider hole in the glider stream blocking a P30 LWSS gun. This releases 8 LWSSes which collect the data from the selected memory cell somewhere down the row.

Figure 7 shows the variation of this design used to pick up the remaining LWSSes at the end of the selected row and transfer the data to the stack. This is triggered directly from the MWSS of the column address and incorporates a P240 gun to detect the address present mark.

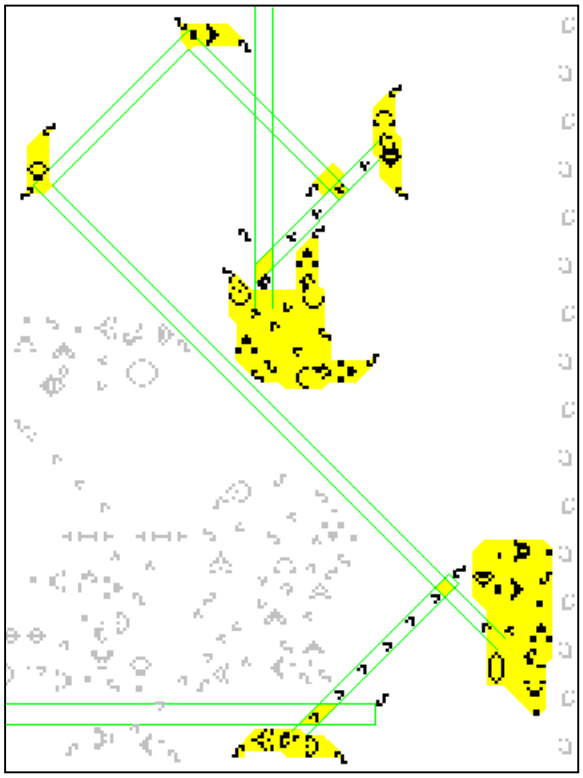

Figure 7 Column Data Collection

The junction between the Finite State Machine and the stacks represented quite a challenge. From the Finite State Machine comes the data which must be split into information to feed the stacks and the next state. The next state must be returned to the Finite State Machine at the time that a symbol is popped from one of the stacks.

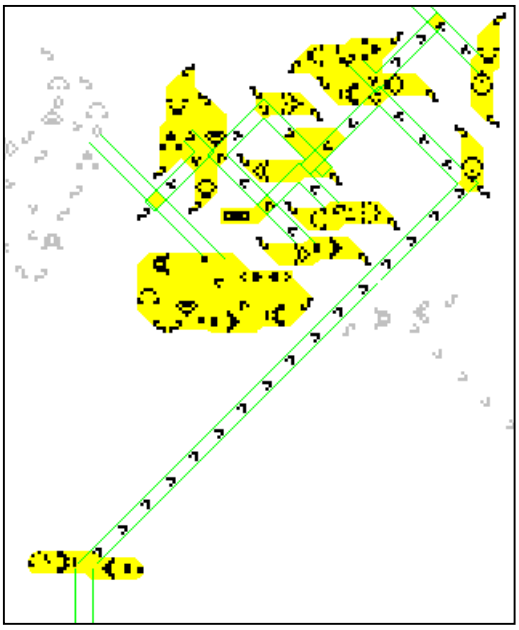

Figure 8 Signal Detector

I chose to maintain the 240 generation frame and use another latch to detect the presence of some data. This had the important consequence that it provided a method of stopping the machine with no extra components. The result is the Signal Detector shown in figure 8. The heart of the detector is a set reset latch. This is a variant of the one used in figure 4, it uses a queen bee reflector to provide the output. One mode of the latch prevents the queen bee reflecting a glider. A Negative feedback loop, containing a fanout, forms the reset leg of the latch. The inverting reaction had to be stabilised with a pentadecathlon to get the loop to exactly 240 generation long. This left the tuneable leg of the fanout for blocking the output of the P240 gun which checks the state of the latch at the end of each frame.

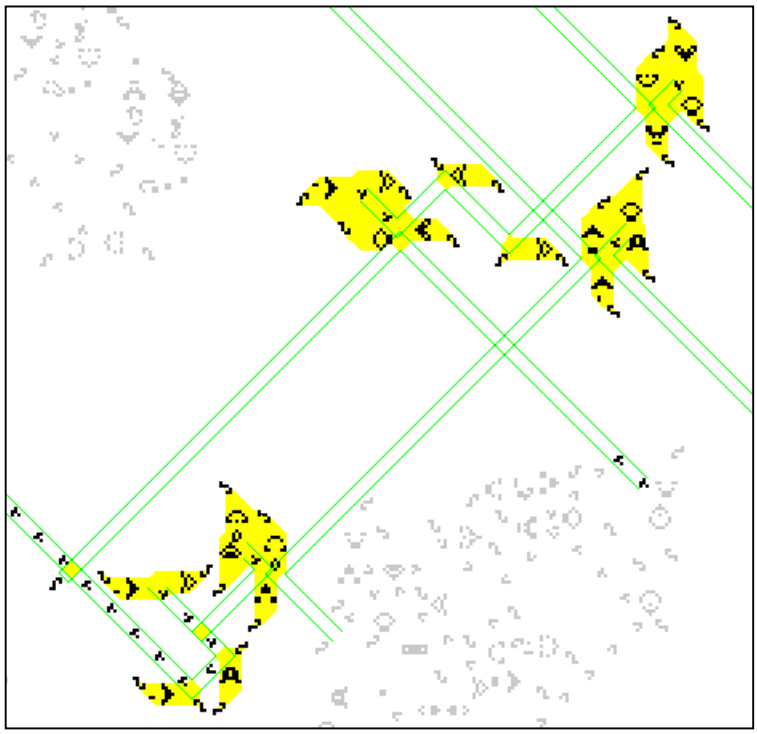

Figure 9 Signal Distributor

Figure 9 shows the next stage. The original data from the Finite State Machine and the output of the Signal Detector are passed to each stack with another copy of the data starting a long loop back to the Finite State Machine. This last is modified at the bottom of the pattern by using the signal detector output to create the address present mark for the Finite State Machine row address. Part of the way through the loop, the pattern in figure 10 tidies up the next state address by deleting 3 gliders. This is done using a P240 gun to create a hole 3 gliders wide, inverting the result and deleting the 3 leading gliders in each frame. This leave the address present mark followed by the next state.

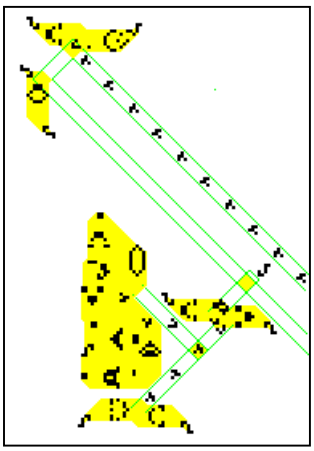

Figure 10 Next State Delay

The other outputs from figure 9 go to each stack. Both stacks get a signal detected glider and the data from the memory cell. The next step is to change this into 3 signals. A push control, a pop control and the data to push. The bottom stack version of this is shown in figure 11. The data comes down from the top and is inverted. A P240 gun samples the direction mark in the frame. If a glider was present in this position in the data, the inverted signal has a hole which the sampling glider passes through. This

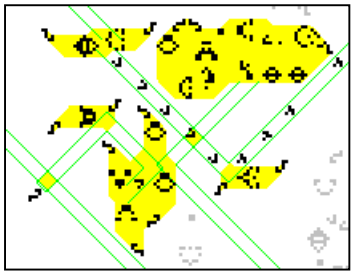

Figure 11 Stack Control Conversion

then goes through a fanout with one output becoming the pop control and the other deleting the signal detected glider. If the operation is a push then the signal detected glider is not deleted and performs the push operation.

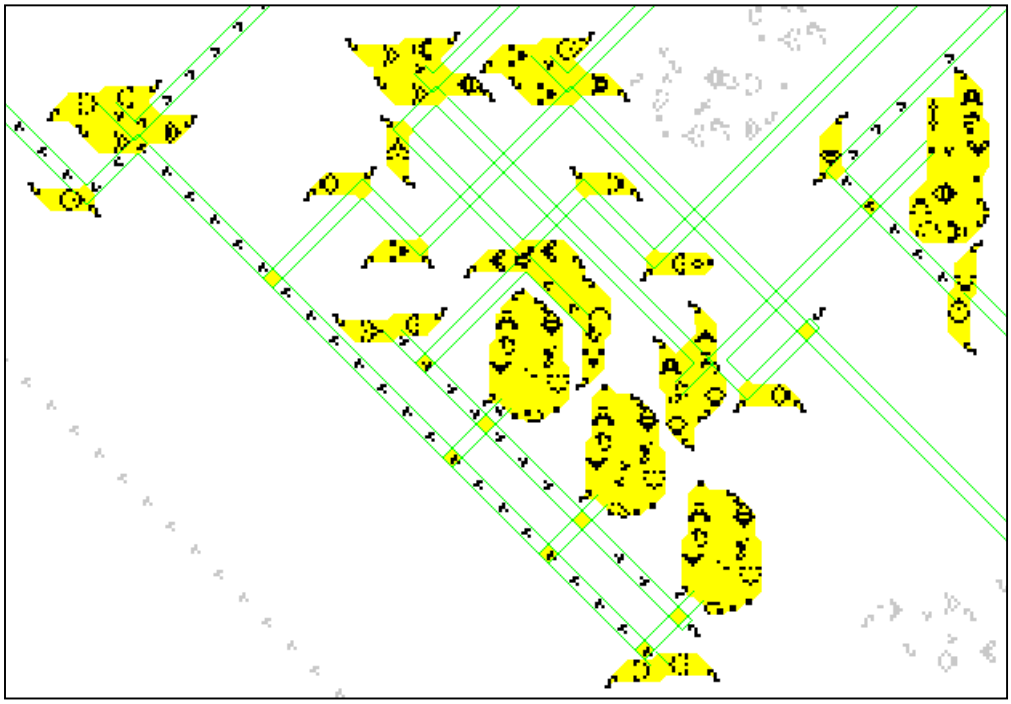

Figure 12 Left Stack Control

Figure 12 shows the version for the top stack. This layout is a little different so that the signal detected glider becomes the pop control this time. From this point on the two stacks are symmetrical except for the slight difference in the layout of the path the data takes to reach the gate allowing it onto the stack.

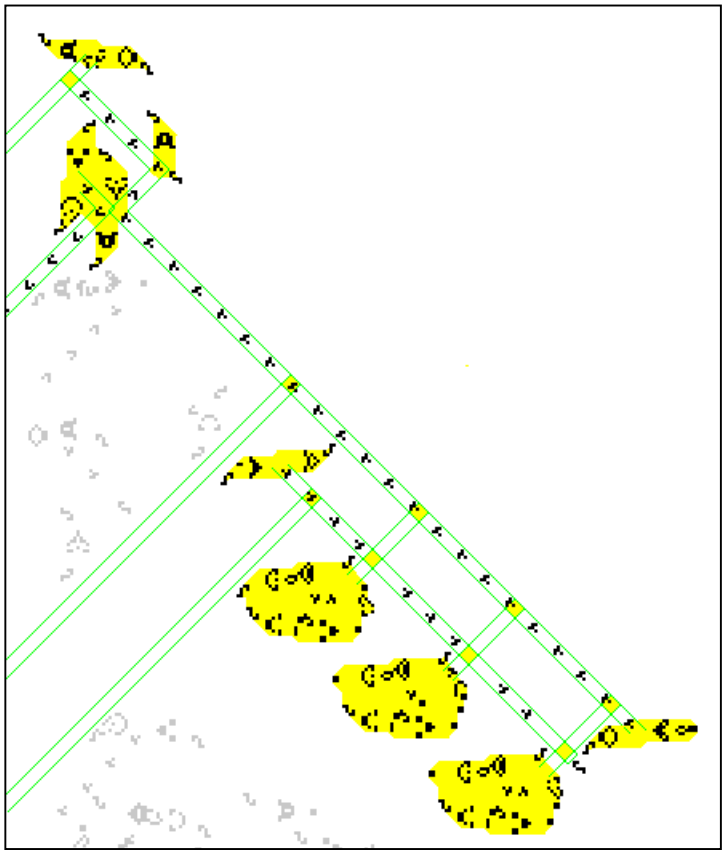

Figure 13 Right Stack Control

Figure 12 shows the creation of the control signals for the left hand side of the stack. Holes in the control signals open stack cells for the symbol to leave during a pop and enter during a push. The push control just needs to go through a fanout so that one copy goes to the right hand stack control and the other copy makes a 4 glider hole in this stack control. The pop operation needs 3 copies. One goes to the other stack control (figure 13), one goes to the gate which allows data onto the stack (figure 14), and the other goes to the pattern in the centre bottom which makes the 3 holes required for the symbol gliders to enter the stack cell. This pattern is actually a bit bigger than could be made with two fanouts but I like the look of it. Three P120 guns are synchronised so that each puts one hole in the stack control but the output of all are blocked by another glider stream. The pop control makes a 3 glider hole in this to let them though.

Figure 13 shows the right stack control. The push control glider makes a 4 glider hole in the control signal to let the symbol gliders out and the pop control glider activates the pattern for making the entry holes. This pattern differs from the one for the left stack control as the blocking gliders go the other way.

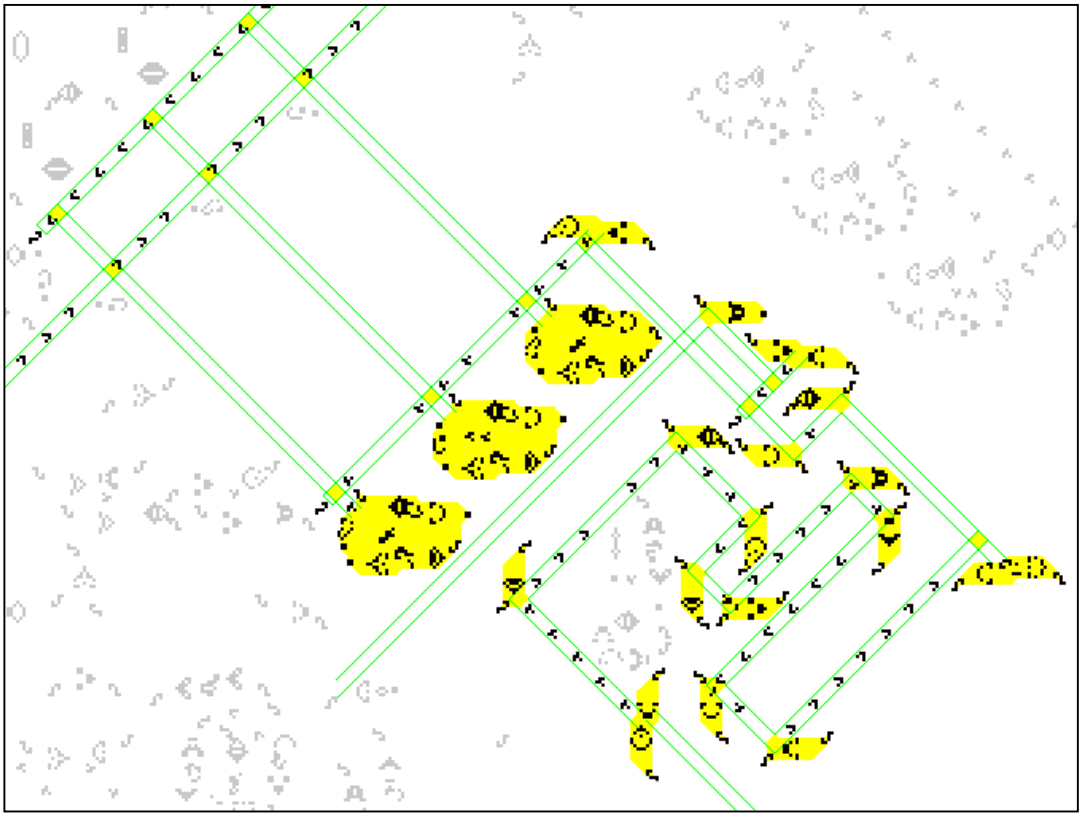

Figure 14 Stack Symbol Input Gate

The gate which allows the symbol onto the stack is feed the symbol gliders in every cycle. This comes through a bit of a delay loop sown in the bottom right of figure 14. A glider from the stack control logic arrives during a push operation and makes a 3 glider hole in a blocking glider stream to allow the symbol gliders through only in the pop cycle. These gliders make a hole in another blocking glider stream. This time the stream is blocking the output of three P120 guns which are aligned and synchronised to inject the symbol gliders into the stack. The normal stack controls will have ensured that the stack wall has holes to allow the symbol gliders in.

A bit of a trick is used to get the symbol gliders out of the stack during a push operation. Figure 15 shows the pattern. A P120 gun at the bottom right is normally blocked by the stack wall. This has two functions. Firstly the hole it makes together with the holes made by any symbol gliders make a 4 glider pattern which is ideal for the addressing the Finite State Machine. This extra hole has becomes the address present mark. Secondly during a pop operation the 4 holes which are required to let the 3 gliders out also let the P120 gun output though. It then passes in front of the stack where is makes a hole 4 gliders wide in a blocking glider stream. The pattern of gliders let through is the stack output. The gliders in the stack cell are destroyed by 3 copies of a pattern known as a blocker.

The output of both stacks are combined though an inverting reaction and feed back to the Finite State Machine.

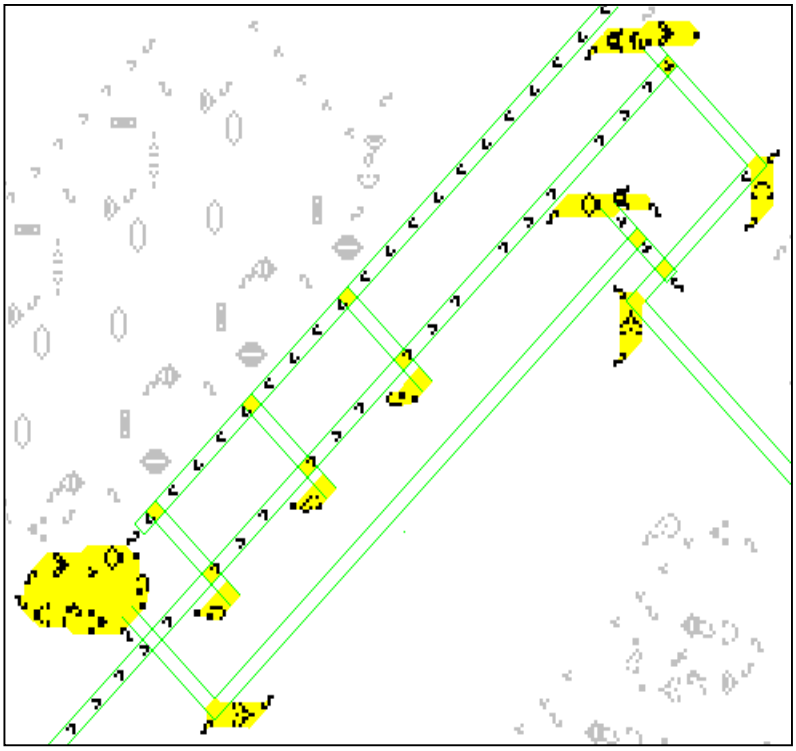

Figure 15 Stack Output

The program I chose for the Turing Machine is one that duplicates a pattern of 1's. With 2 1's on the tape to the right of the reading position it takes 16 cycles to stop with 4 1's on the tape. This takes over one hour on my computer. The Finite State Machine for this program is shown in figure 16. The symbol which causes a state transition is shown at the base of each arrow and the next state and direction half way along it. For example, if the machine reads a 1 in state zero it will change to state 1, write a 2 and move the reading head to the right.

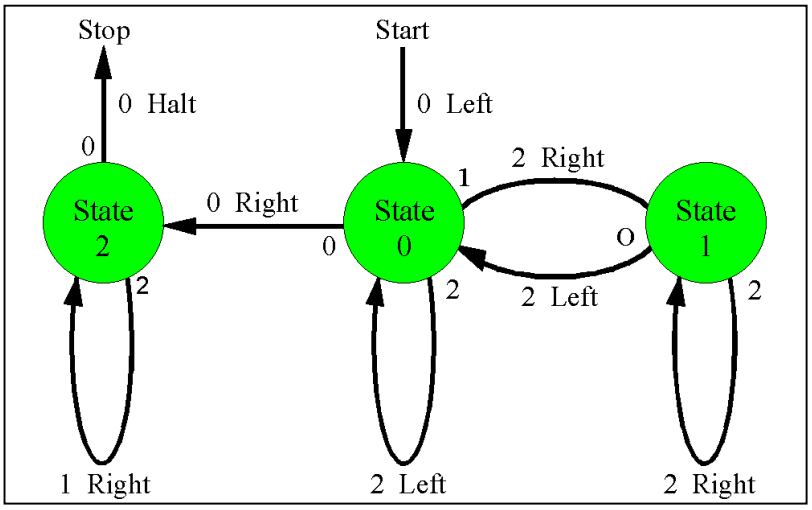

Figure 16 The Turing Machine Program

The start transition has been implemented by a P240 gun placed behind a blocking glider stream. This is synchronized so that when the blocking glider is deleted it inserts the instruction in the path taken by gliders from the stack.

The Turing Machine takes 11040 generations for one cycle. About 6300 generations are spent in the stack part and 4740 in the Finite State Machine part. Adding a row or column adds 528 generations to each cycle which unfortunately needs adjusting to be a multiple of 240 generations. The Finite State Machine big enough for the large Universal Turing Machine will require 16 rows and 8 columns. That is an extra 18 frames of 240 generations making 15360 generations in total.

This Universal Turing Machine will only simulate a Turing Machine with two symbols and a tape with one only one potentially infinite end. The other part of the Universal Turing Machine tape holds the description of the Turing Machine it is simulating. A few simple transformations convert the machine in figure 16 into an acceptable equivalent machine. The description of this machine would take up more than 1000 squares of the Universal Turing Machines tape. The size of the stacks would be the most important contribution to the size of any example Universal Turing Machine pattern.

The patterns presented here can by found on my web site [1] many of them are animated by a Java Applet written by Paul Callahan [7]. The patterns can be downloaded and run on many freeware programs. I use life32 written by Johan Bontes [3]. The pictures themselves where generated by a life program called MCELL [8].

I am now working on building a Turing Tape generator. This will generate Stack cells faster than the machine can use them. This project presents a very different set of problems. The technique used for the Turing Machine was to build each part and then gradually add parts together. For the Turing Tape generator I need more help from automated tools to place the components. The generator will need a large number of similar parts but finding an order in which they are can be assembled without unwanted collisions in the process will be difficult.

Bibliography

1. Paul Rendell	Conway's Game Life Turing Machine www.rendell.uk.co/gol
2. Dieter Leithner and Peter Rott	Dieter and Peter's Gun Collection "http://www.mindspring.com/%7Ealanh/guns.zip" and "http://www.mindspring.com/%7Ealanh/guns2.zip"
3. Johan Bontes	Life32 PC Program for Conway's Game Life "http://psoup.math.wisc.edu/Life32.html"
4. Stephen Silver	Stephen Silver's Life Lexicon "http://www.argentum.freeserve.co.uk/lex_home.htm"
5. Marvin L. Minsky	Computation: Finite and Infinite Machines. Prentice-Hall, Inc., 1967.
6. Martin Gardner	Mathematical Games articles in Scientific American: ♦ On Cellular automata, self-reproduction, and the game "life". February 1971 ♦ The fantastic combinations of John Conway's new solitaire game "life". October 1970
7. Paul Callahan	Java Applet was written by Paul Callahan
8. Mirek Wojtowicz	Mirek's Cellebration (MCELL)"http://www.mirwoj.opus.chelm.pl"

End of Paper