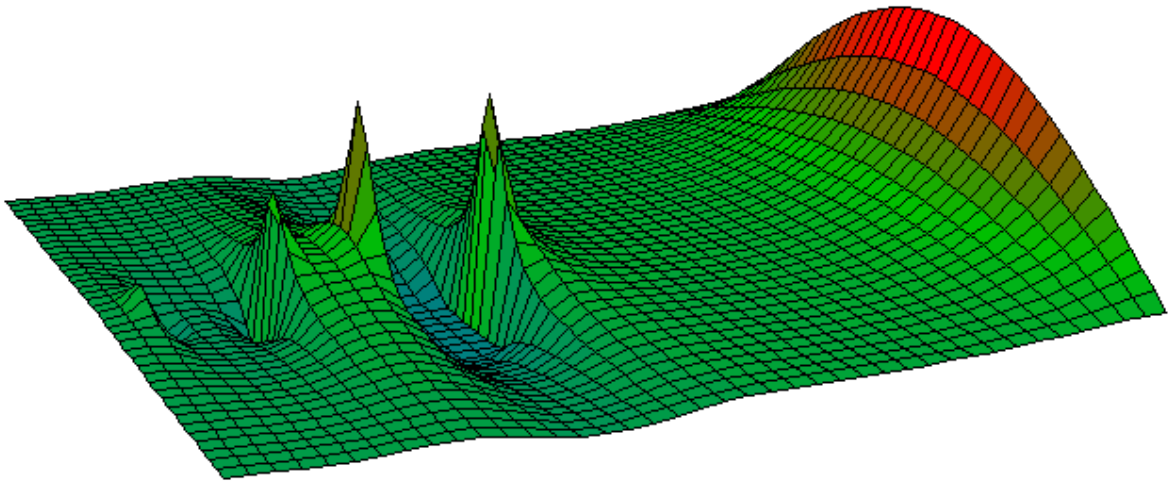
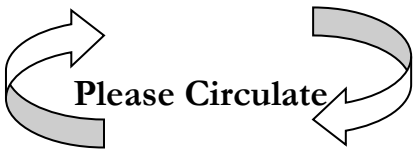
Journal of J

An interdisciplinary journal on J programming language and applications in science





MPM press

A open access Journal

ISSN: 2174-9280

Vol.6, No.1 June 2018

Finding Automata Exhibiting Complexity via Input Entropy

Cliff Reiter

Lafayette College, Department of Mathematics, Easton PA, 18042 U.S.A.

reiterc@lafayette.edu

Cellular automata consist of an arrangement of cells where each cell has a finite neighborhood such that at any given time each cell is in some state, typically chosen from a finite collection. The state of each cell is updated in time steps using a rule that depends on the states of the neighborhood of the cell at the previous time step. For this note we will consider 1-dimensional and 2-dimensional arrays with periodic boundary conditions for our arrangements of cells. Cellular automata are attractive because they allow one to model behavior using local models; that is, the next state of a cell depends only on its state and the state of its neighbors. Cellular automata are remarkable for their diverse behaviors which are often classified as ordered, chaotic or complex [13, 18, 19]. While each of those classes may have applications of interest, the complex ones are often the most intriguing.

During the spring of 2016 I was teaching my special topics course using Fractals, Visualization and J, 4th edition, [14] and I investigated Larger than Life automata using J as demonstrations in the course. The problem of finding examples with complex behavior arose and I used some ad hoc techniques to find visually interesting examples with apparent complex behavior. I eventually wrote a note for Vector based on those demonstrations [15, 16]. Subsequent to that, I was reviewing well known formulas for entropy as related to information content:

$$E = -\sum p_i \ln(p_i).$$

Wuensche [20] uses entropy to distinguish automata as ordered, chaotic or complex. In earlier reading about entropy I didn't understand what the p_i were in the context of automata but he made it clear that these were the proportional frequencies that the states in neighborhoods appear. Wuencsche [20] called this ***input entropy***. Input entropy evolves (and becomes more meaningful) with time. The key to computing these proportions in J is key: `#/.~` since that gives frequency counts when applied to a list of the states of all the neighborhoods. This note explores using input entropy on searches for complex general 1-dimensional and 2-dimensional automata and for complex 1-dimensional and 2-dimensional Larger than Life automata. Finding complexity remains a bit of an art as much as a science, but there is plenty of delight in how easy J makes the computations and in the images of complexity that we find.

1. One Dimensional Automata and Input Entropy

Below we load some scripts. The first is designed for use with the chapter on automata in [14] and we will use several utilities from it in our illustrations. The others load plotting and statistics functions.

```
   load '~addons/graphics/fvj4/automata.ijs'
   load 'plot'
   load '~addons/stats/base/base.ijs'
```

We will begin with some examples of general Boolean 1-dimensional automata on neighborhoods of radius 2 and hence, width 5. These are general in the sense that for each neighborhood the result is chosen in a manner that is independent of similar neighborhoods. To specify such an automaton we need to specify what happens on the 32 neighborhoods of width 5. Below we specify such a list, `r`, and show the corresponding neighborhoods in the columns beneath its entries. Then we apply the rule to two test neighborhoods.

```
   $r=:#:2345678901
32

   r
1 0 0 0 1 0 1 1 1 1 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 0 1 0 1

   |:#:i.32
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

   lauto
{~ #.

   r&lauto 0 0 0 0 0
1
   r&lauto 0 0 0 0 1
0
```

As we want to automate the process on an array of cells, we define a dyadic adverb that takes the window radius as its adverb argument and the rule list as its left argument while the right argument is an array of the states of the cells. Note we use moving windows of width one plus twice the radius on periodic extension by the radius on both ends.

```
   Auto=:1 : 0
:
((1+2*m) x&lauto\ m nperext) y
)

   2 nperext 0 0 0 0 0 1 1 1 1 1
1 1 0 0 0 0 0 1 1 1 1 1 0 0

   r 2 Auto 0 0 0 0 0 1 1 1 1 1
0 0 1 0 0 1 0 1 0 0
```

It is fairly natural to apply the automaton iteratively and to assemble the states of the iterates in a square array.

```
   Autoevo=:1 : ' m Auto^:(i.@#@])'

   r 2 Autoevo ?.16$2
0 1 0 1 1 0 0 1 1 1 0 0 0 0 1 0
1 0 1 0 0 0 1 1 0 0 0 0 0 0 1 1
1 1 1 1 0 0 1 0 0 0 1 1 0 0 1 0
0 0 0 0 0 1 1 1 0 0 1 0 0 1 0 1
1 0 1 0 0 1 0 0 0 1 1 1 1 0 0 1
0 1 1 1 1 1 1 0 0 1 0 0 0 0 1 1
1 0 0 1 1 0 0 0 1 1 1 0 0 0 1 0
1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 1 1 0 1 1 1 0 0
1 1 0 1 1 1 0 0 1 0 1 0 0 0 0 1
0 1 1 0 0 0 0 1 0 0 1 1 0 0 0 1
1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0
1 1 0 1 1 0 0 1 0 0 0 0 0 1 0 0
1 0 1 0 0 0 1 1 1 0 1 0 0 1 1 1
1 1 1 1 0 0 1 0 1 1 1 1 1 1 0 0
1 0 0 0 0 1 0 1 0 0 1 1 0 0 0 1
```

Next we do this for an image size illustration and we replicate pixels in 2 by 2 blocks to facilitate seeing the image. The result is shown in Figure 1. We will need to explain a bit more, but the example illustrates chaotic behavior.

```
   b=:r 2 Autoevo ?.256$2

   2 spix i.2 3
0 0 1 1 2 2
0 0 1 1 2 2
3 3 4 4 5 5
3 3 4 4 5 5

   view_image 2 spix b{0,:3#255
```

We define an adverb `Ie` below so that we can take a look at how the input entropy evolves. The expression `(1+2*m) ]\]` computes all the neighborhoods of appropriate width. And `#/.~` gives the frequencies each neighborhood appears and then we compute proportions and take a sum of products with logarithms. We show the result on a random array of cells and on a periodic arrangement of cells.

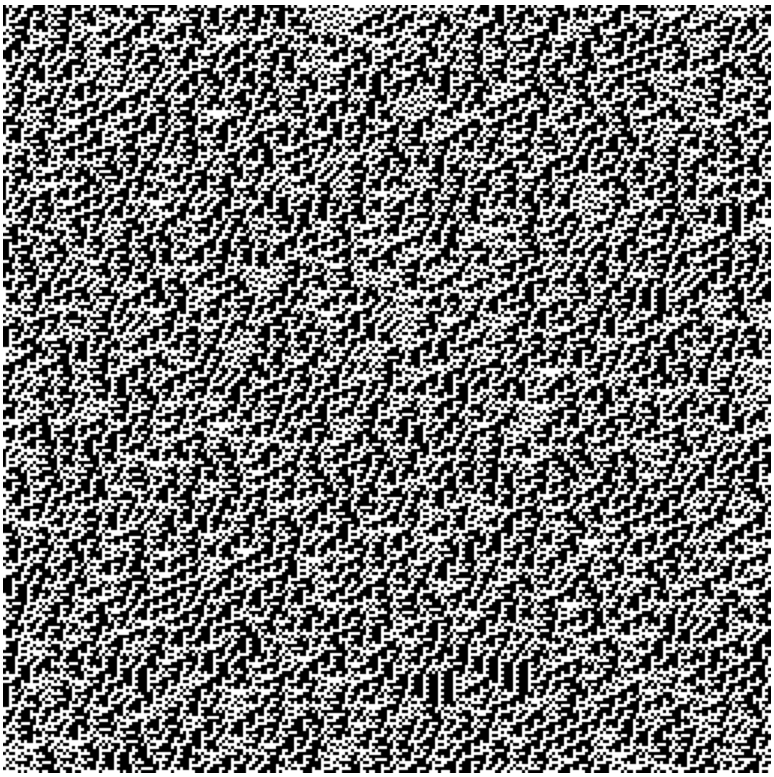


Figure 1. A rule giving rise to chaotic behavior.

```
   Ie=:1 : 0("1)
-+/(*^.)(# %~ [: #/.~(1+2*m) ]\])m nperext y
)

   2 Ie ?.128$2
3.29569

   2 Ie 128$0 0 1
1.23866
```

Then we plot the result for each row (iterate) of `b`. The result is shown in Figure 2.

```
   plot 2 Ie b

   stddev 2 Ie b
0.0490799
```

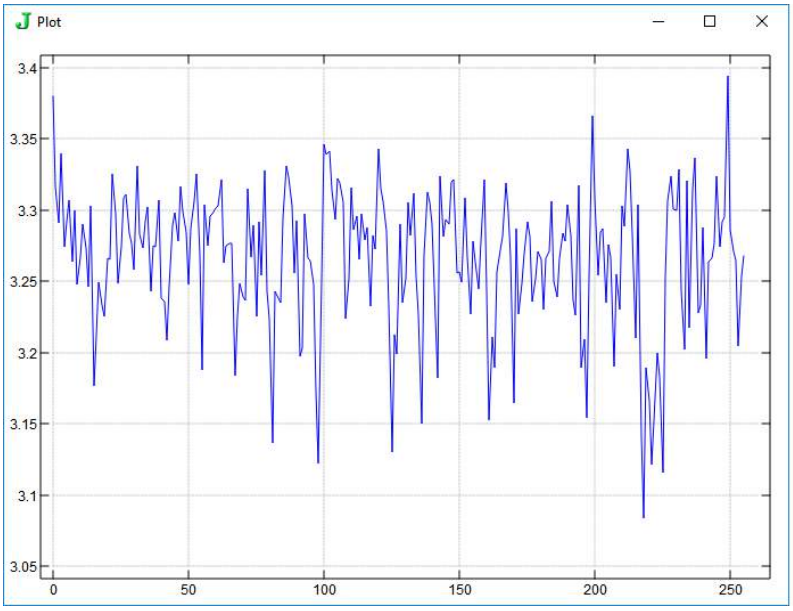


Figure 2. Input entropy for the chaotic automaton.

The guidance in [20] is that automata that show order, complexity and chaos can be distinguished by the following.

	order	complexity	chaos
avg Ie	low	medium	high
stddev Ie	low	high	low

In particular, in this illustration, we see the input entropy is high while the standard deviation is low. Those statements are, of course, not precise, but they do give useful guidance.

We consider another example.

```
   $r=:(32#2)#:2001432084
32

   b=:r 2 Autoevo ?.256$2

   view_image 2 spix b{0,:3#255

   plot 2 Ie b

   stddev 2 Ie b
0.192845
```

The standard deviation of the input entropy is high while its average is moderate. The evolution is shown in Figure 3 and the input entropy in Figure 4.

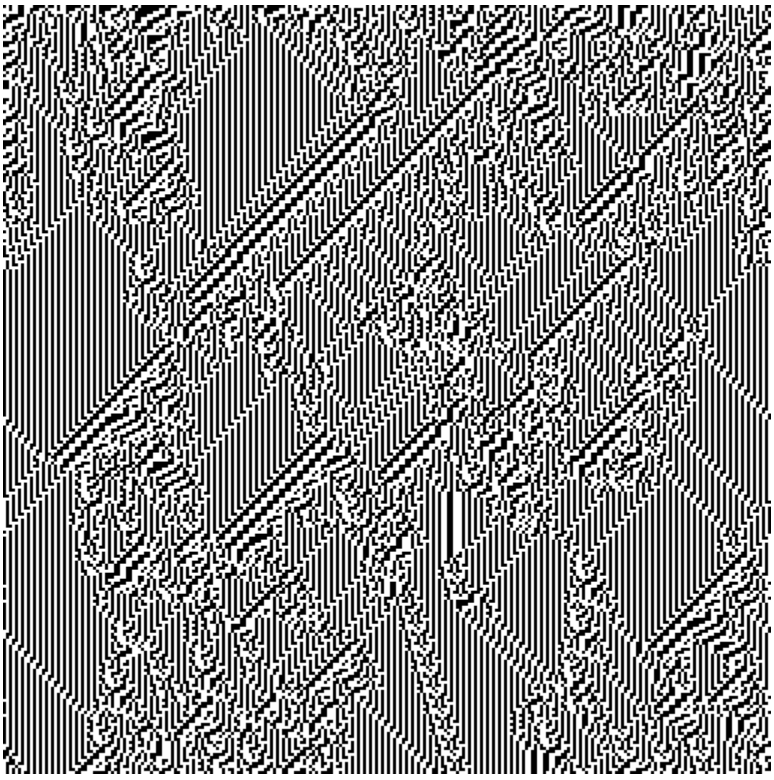


Figure 3. An automaton with complex behavior.

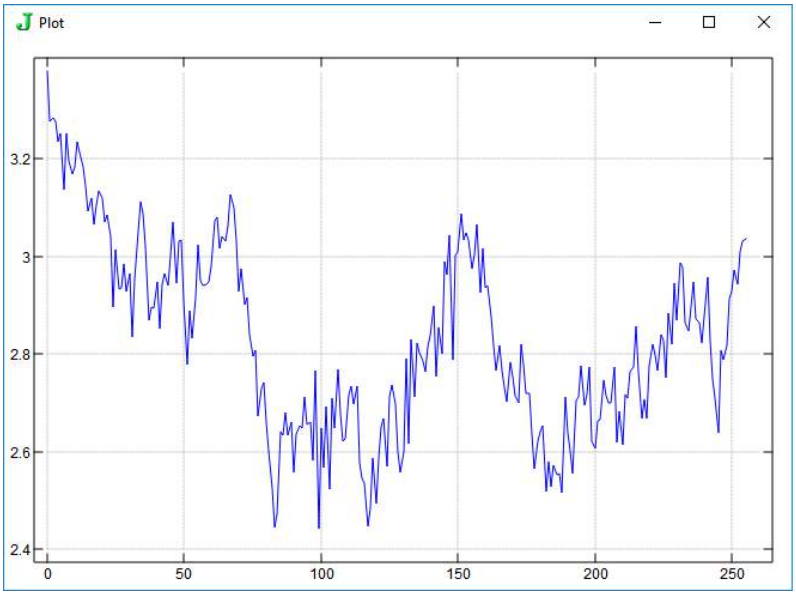


Figure 4. Complex behavior has moderate input entropy.

We consider an example with ordered behavior.

```
   #.r=:?.32#2
1505908773

   b=:r 2 Autoevo ?.256$2

   view_image 2 spix b{0,:3#255

   plot 2 Ie b

   stddev 2 Ie b
0.109284

   stddev 128}.2 Ie b
0.0809936
```

We note it has low input entropy and reasonably low standard deviation. Figure 5 shows the evolution while Figure 6 shows the input entropy. This example makes it clear that we should avoid transient behavior.

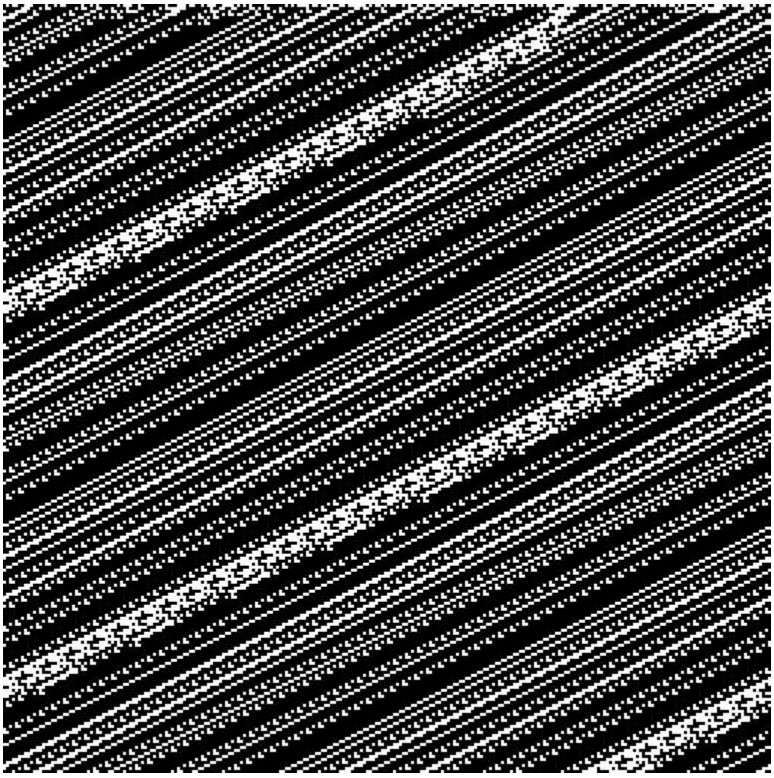


Figure 5. An automaton with ordered behavior.

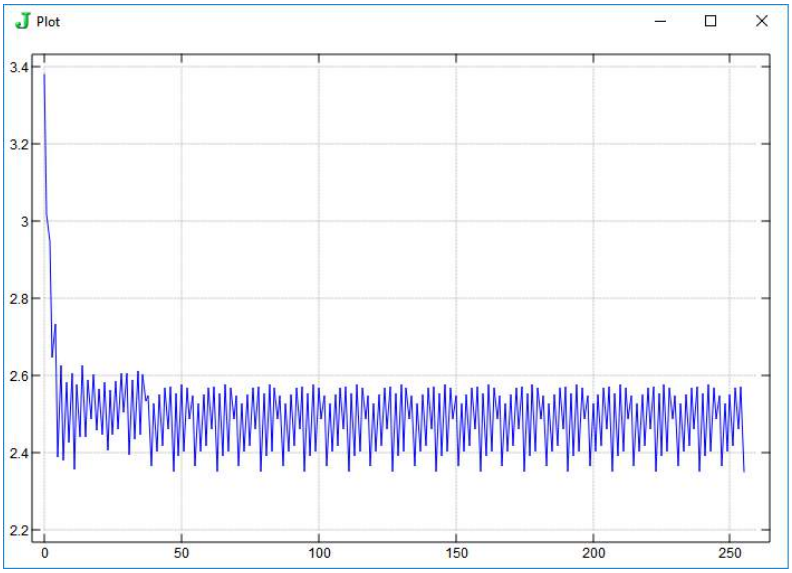


Figure 6. Input entropy of an automaton with ordered behavior.

2. A Visualization Scheme Based on Three Iterates

We next turn to describing a scheme that we often find aesthetically pleasing and which adds some information to the visual display of these types of automata. We return to our example with complex behavior from the previous section. We begin with the Boolean array of successive iterates and taking windows of size three, moving the axis of length 3 to the end so that it can represent an RGB value. Each triple represents a cell at a step and its two previous values. We multiply by `2}.b` so that if a cell is dead, the RGB will be black. We rearrange the RGB values so that an alive cell will be green if it was dead on the previous two iterates and will be cyan or yellow if it was alive at one of the previous iterates and it will be white if it was also alive on the previous two iterates. The result is shown in Figure 7.

```
   r=:(32#2)#:2001432084

   b=:r 2 Autoevo ?.256$2

   $3 ]\ b
254 3 256

   $(2}.b)* 1|:3]\ b
254 256 3

   t=:0 2 1{"1 ]255*(2}.b)* 1|:3]\ b

   view_image 2 spix t
```

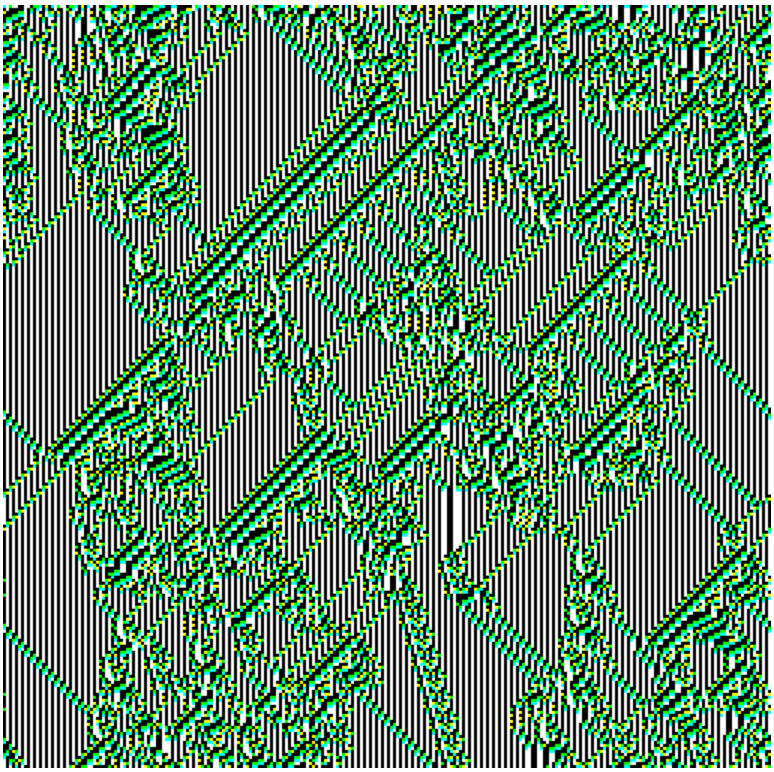


Figure 7. A complex automaton with color determined by 3 iterates

Below we see the frequency that each of those RGB values occur. Nearly half the pixels are black and almost as many are white. Then we look at an enlarged version of a zoom into the lower left of the image which is shown in Figure 8. The individual colors used for each pixel are easier to see in that image.

```
({.;#)/.~,/t
```

0 255 255	5566
255 255 255	21518
0 255 0	5162
0 0 0	29908
255 255 0	2870

```
view_image 8 spix _64 64{.t
```

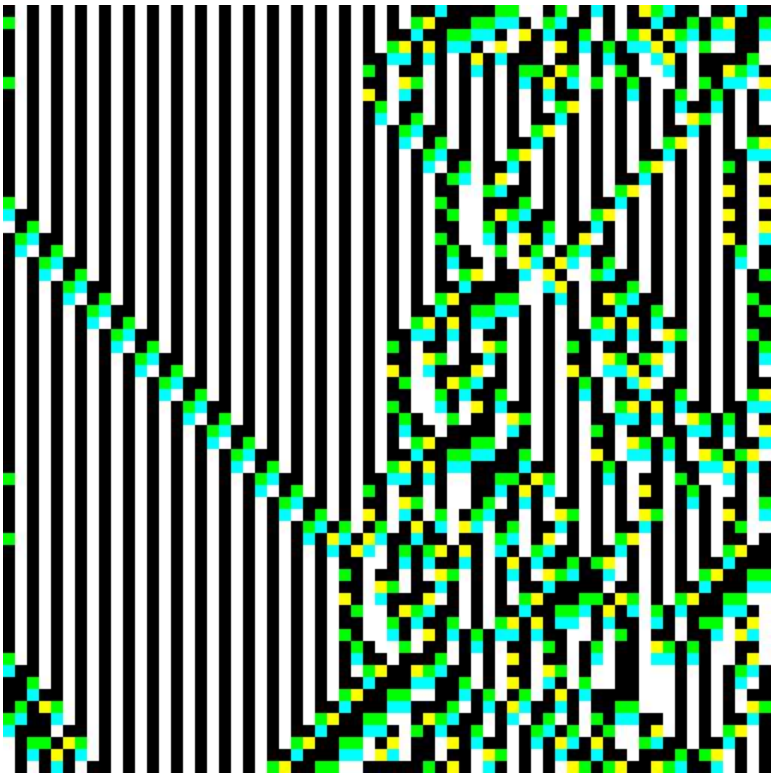


Figure 8. A zoomed version of lower left of Figure 7.

3. Finding General 1-dimensional Automata with Complexity

We eventually will share a conjunction we used to find complex 1-dimensional automata using input entropy; however, first we will discuss intermediate approaches and see some results. We began by looking for general 1-dimensional automata on neighborhoods with radius 2 that had standard deviation of input entropy greater than 0.1 and that were not periodic with a period less than 25. We produced 100 random such examples. Typical results often appeared complex as in Figure 9-11 although Figure 10 appears that it might soon become ordered. Figure 12 also arose that way but would probably be classified as chaotic. Each of the illustrations in this section require the definition of a Boolean rule list. The corresponding rules are given as scripts at [17] for readers wanting to duplicate or explore specific examples given here.

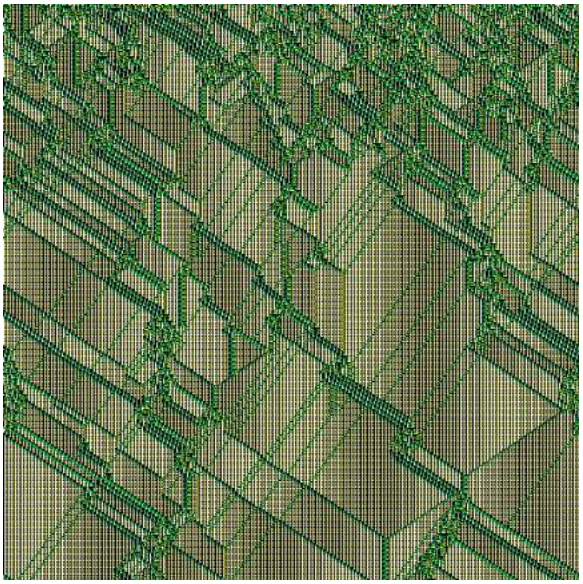


Figure 9. Found 1-dimensional Automaton

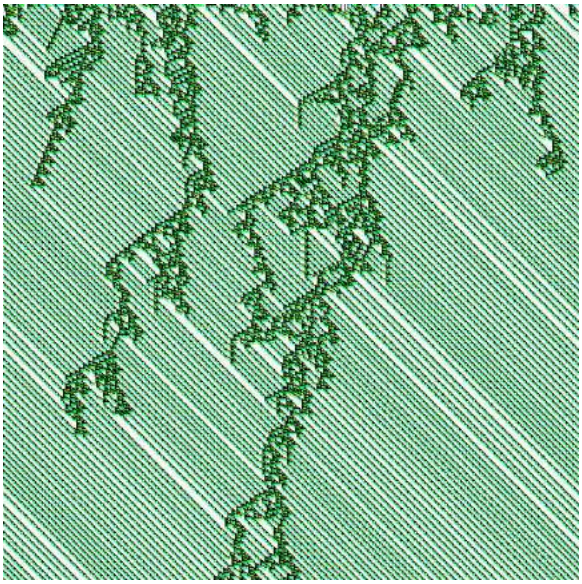


Figure 10. Found 1-D Automaton

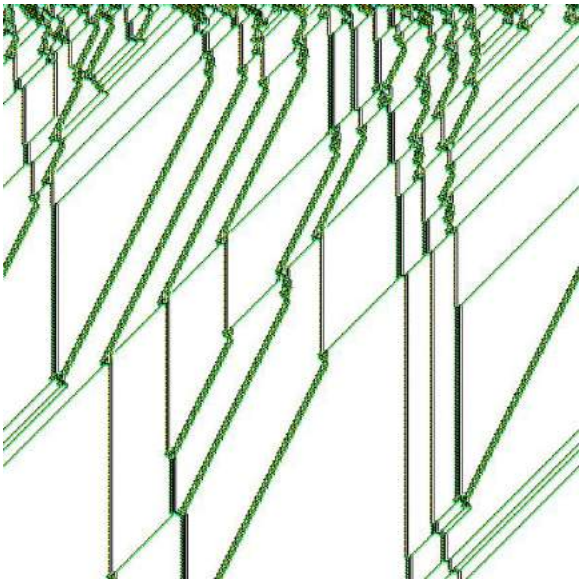


Figure 11. Found 1-D Automaton

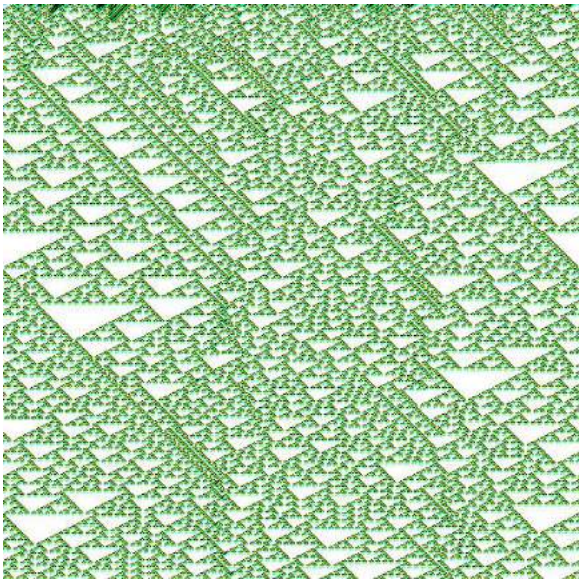


Figure 12. Found 1-D Automaton

Other illustrations include Figure 13 which appears to be near order and Figure 14 which is ordered, but not periodic with a short period. Note that while we might surmise that the automaton shown in Figure 13 becomes ordered, it may well be true that careful choice of input configurations allow computations to be done which is often associated with complexity. Other favorites found using this method are shown in Figures 15-18. We then repeated the experiment gathering some data on input entropy. Some favorites are shown in Figures 19-22.

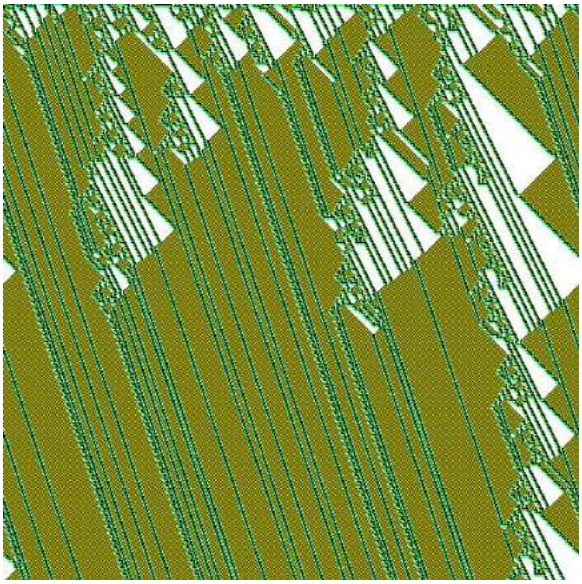


Figure 13. Found 1-D Automaton

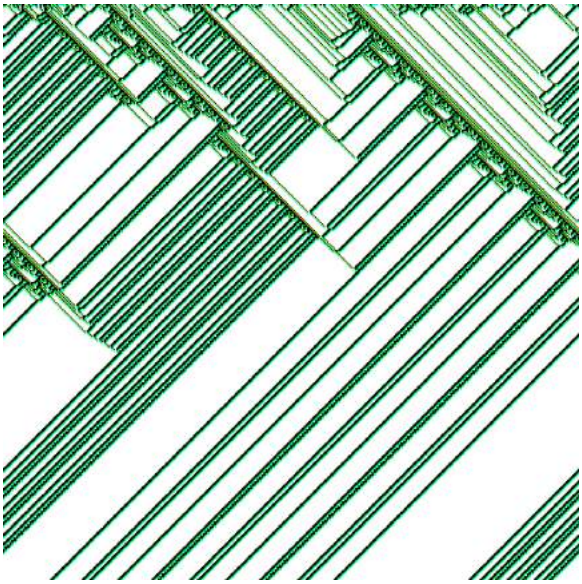


Figure 14. Found 1-D Automaton

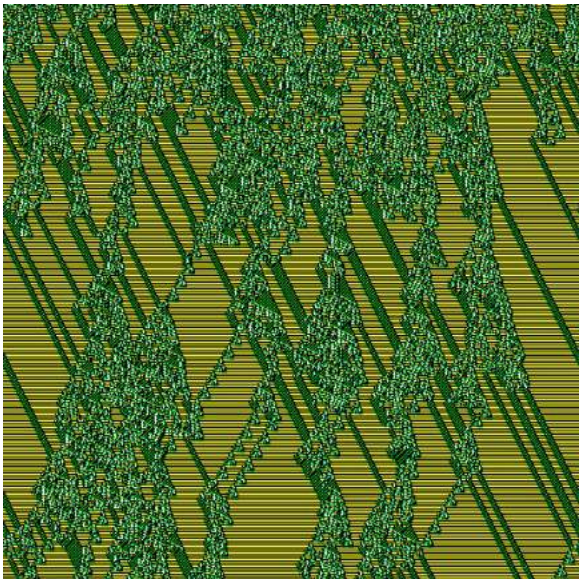


Figure 15. Found 1-D Automaton

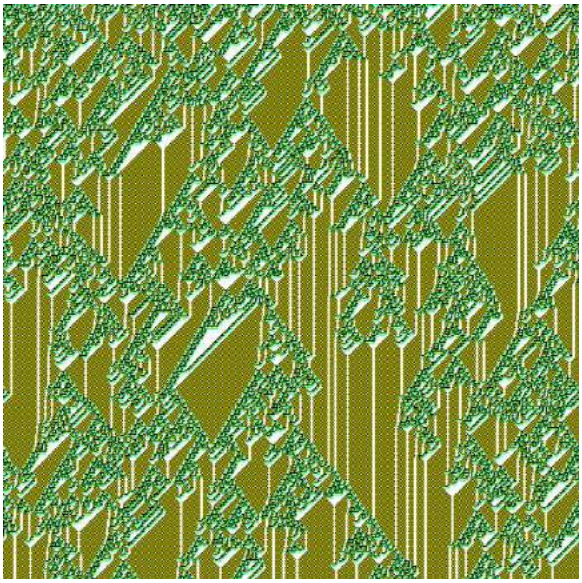


Figure 16. Found 1-D Automaton

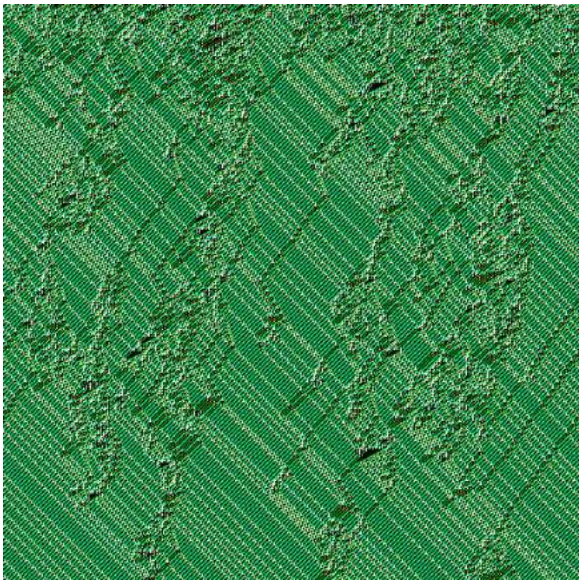


Figure 17. Found 1-D Automaton

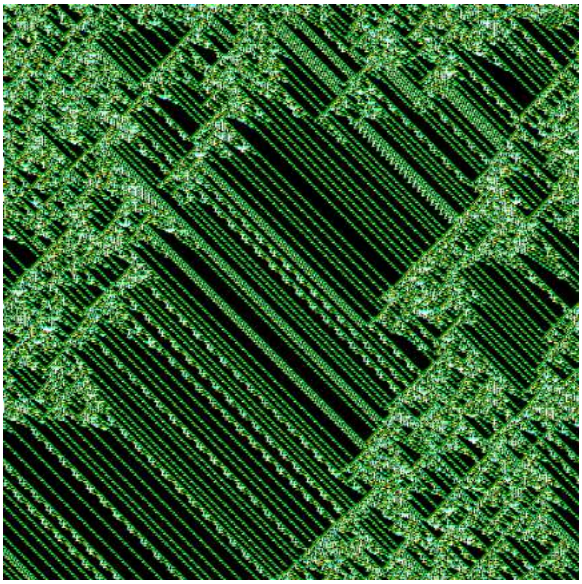


Figure 18. Found 1-D Automaton

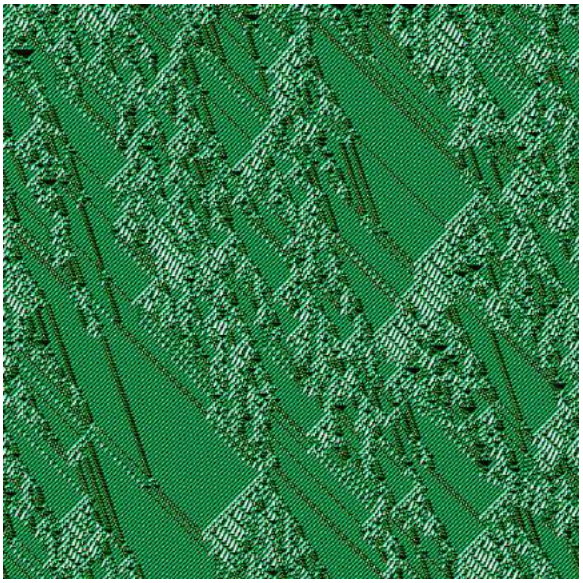


Figure 19. Found 1-D Automaton

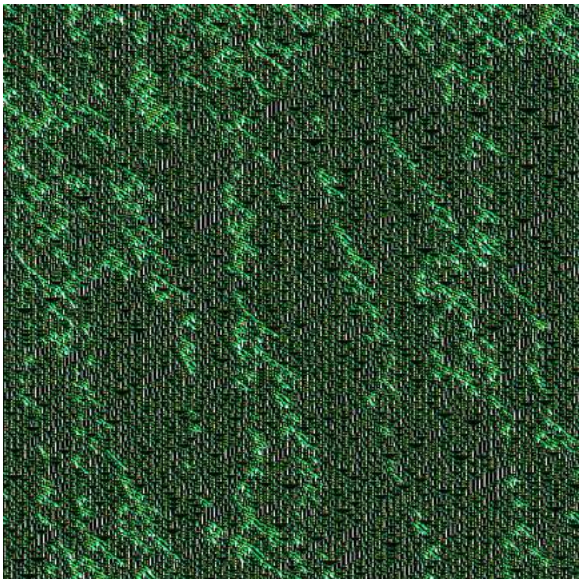


Figure 20. Found 1-D Automaton

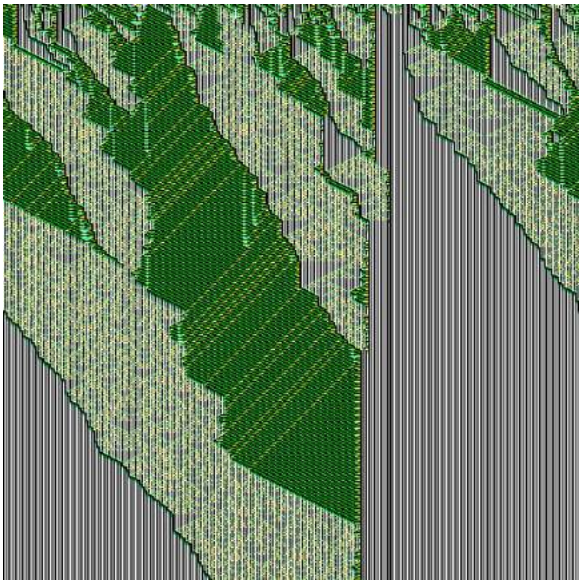


Figure 21. Found 1-D Automaton

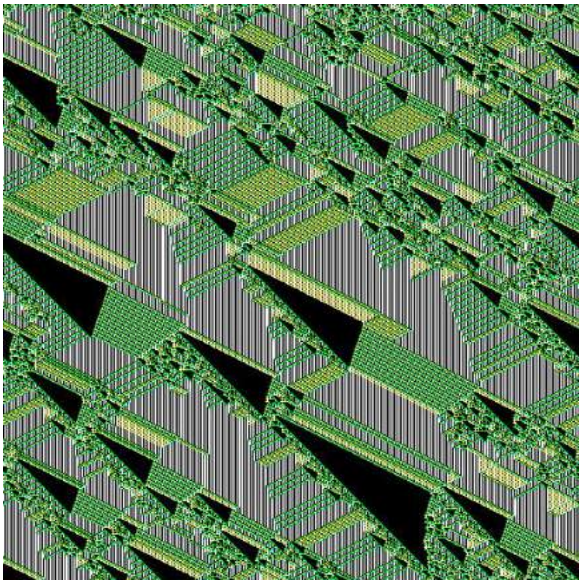


Figure 22. Found 1-D Automaton

We then added the condition that the average input entropy be greater than 2 and the standard deviation should be greater than 0.12. The periodicity requirement was dropped. This resulted in undesirable images with eventual translational order as in Figures 23-24. Other favorites found using this method are shown in Figures 25-28.

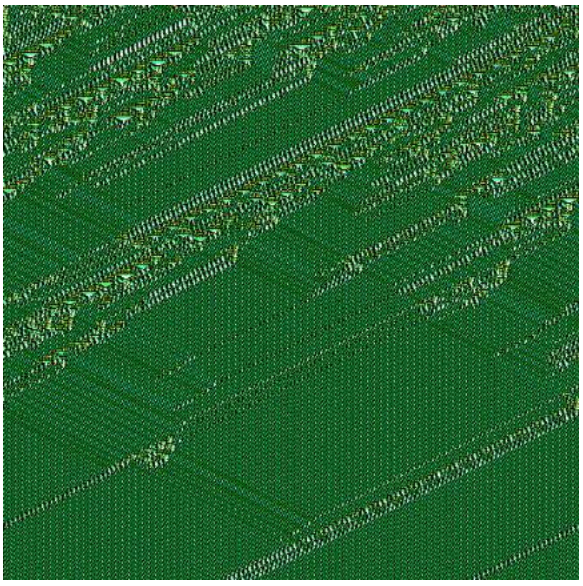


Figure 23. Found 1-D Automaton

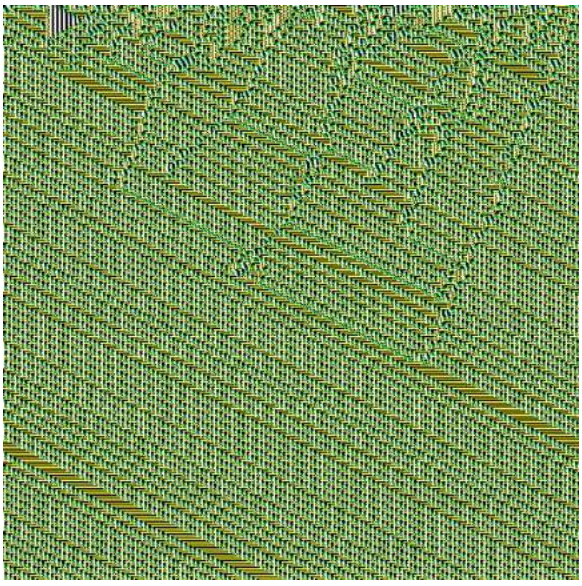


Figure 24. Found 1-D Automaton

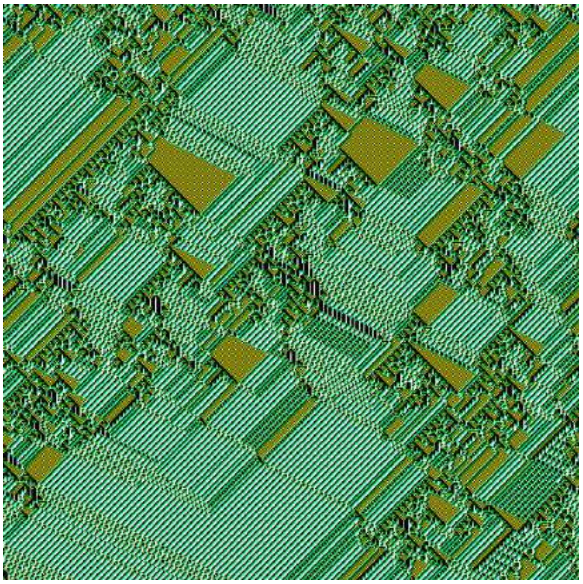


Figure 25. Found 1-D Automaton

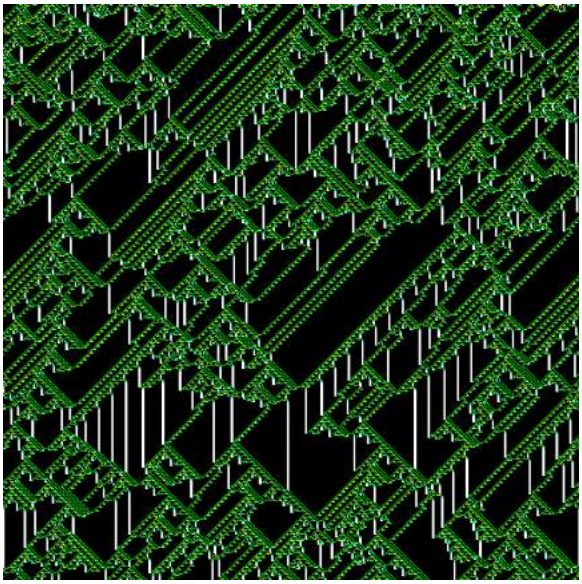


Figure 26. Found 1-D Automaton

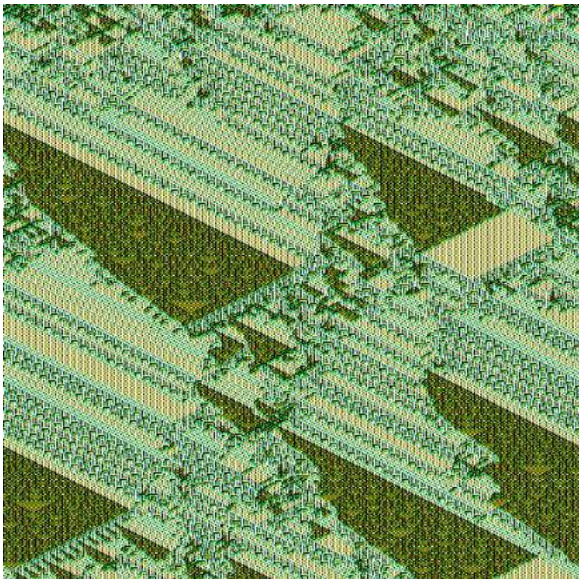


Figure 27. Found 1-D Automaton

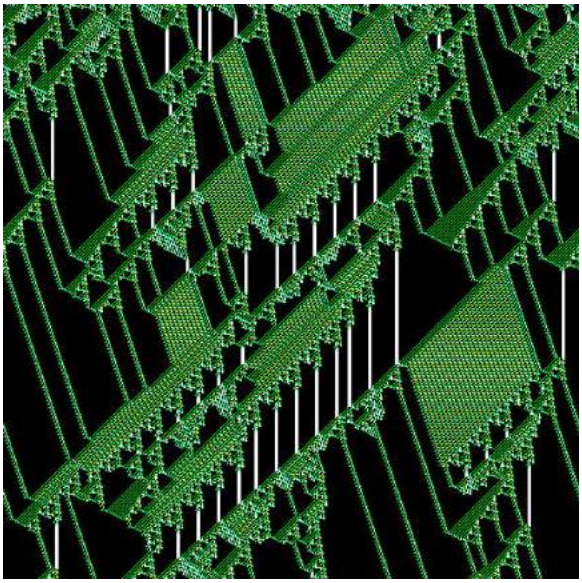


Figure 28. Found 1-D Automaton

Lastly, we demanded the same conditions on input entropy but also required the last iterate not be an easy translate of a recent row. Figures 29-37 show favorites produced this way. Figure 37 nevertheless probably belongs in the chaotic regime.

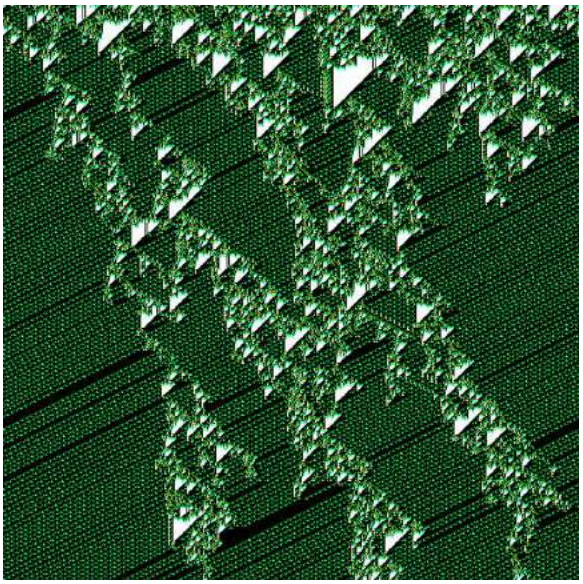


Figure 29. Found 1-D Automaton

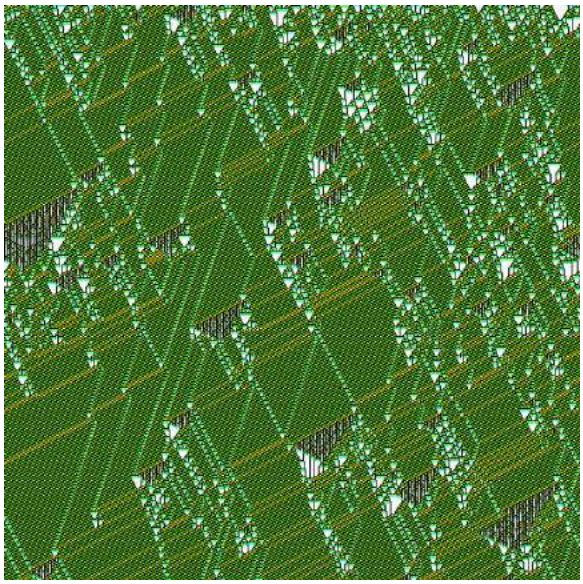


Figure 30. Found 1-D Automaton

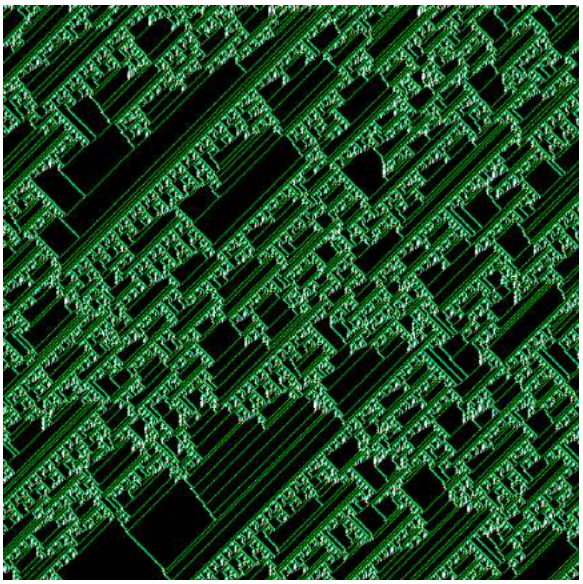


Figure 31. Found 1-D Automaton

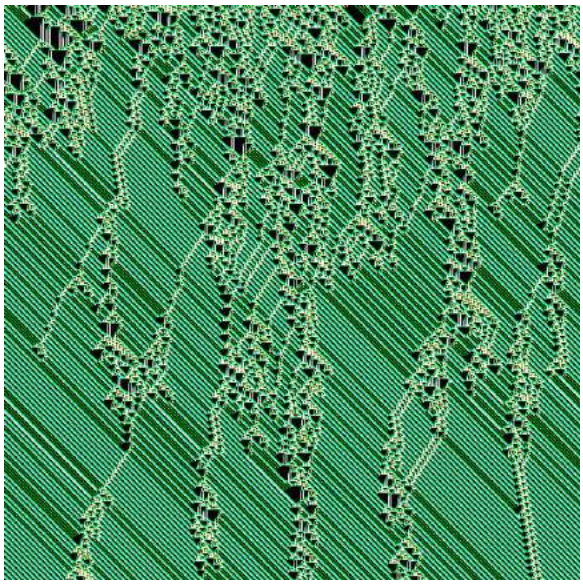


Figure 32. Found 1-D Automaton

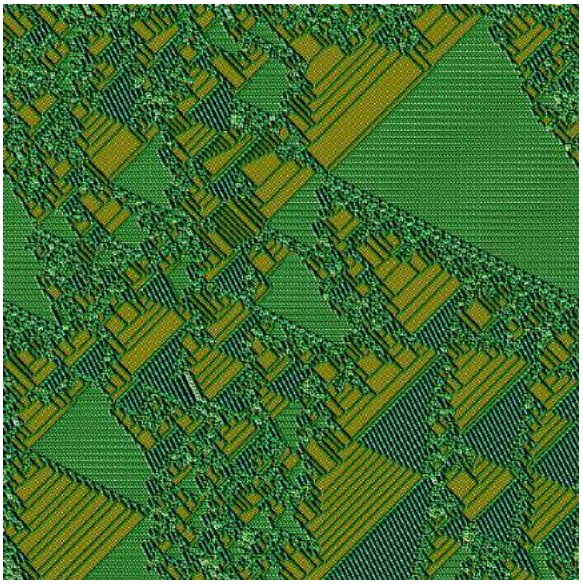


Figure 33. Found 1-D Automaton

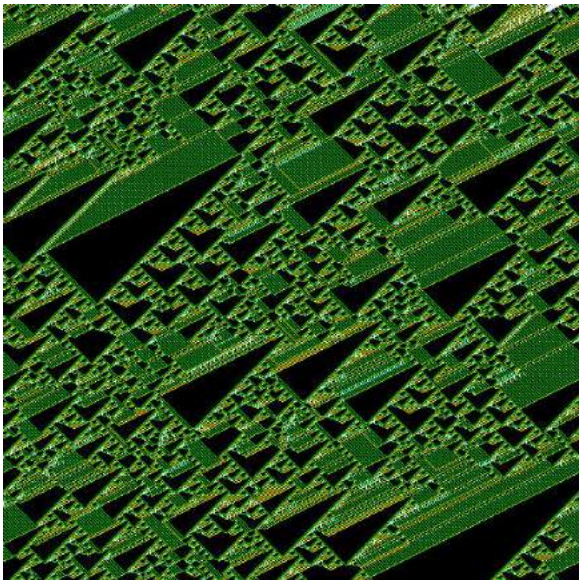


Figure 34. Found 1-D Automaton

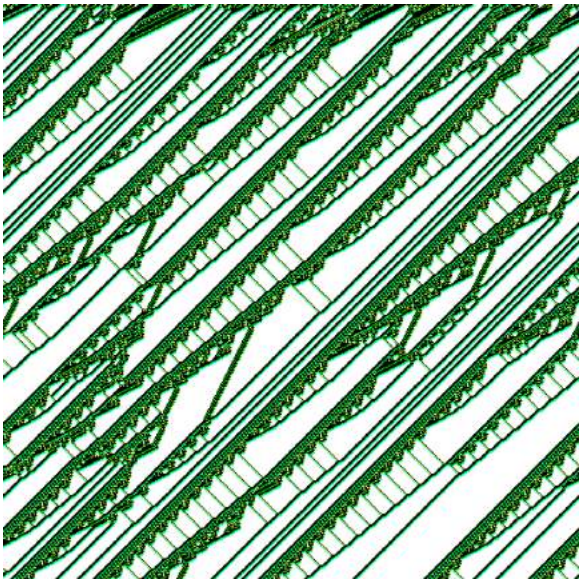


Figure 35. Found 1-D Automaton

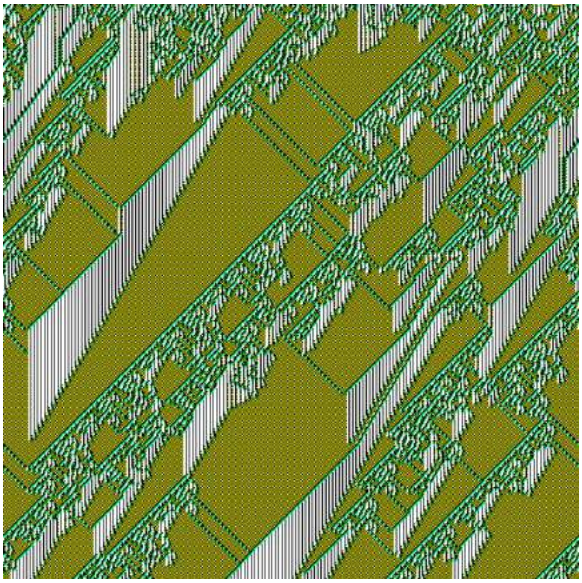


Figure 36. Found 1-D Automaton

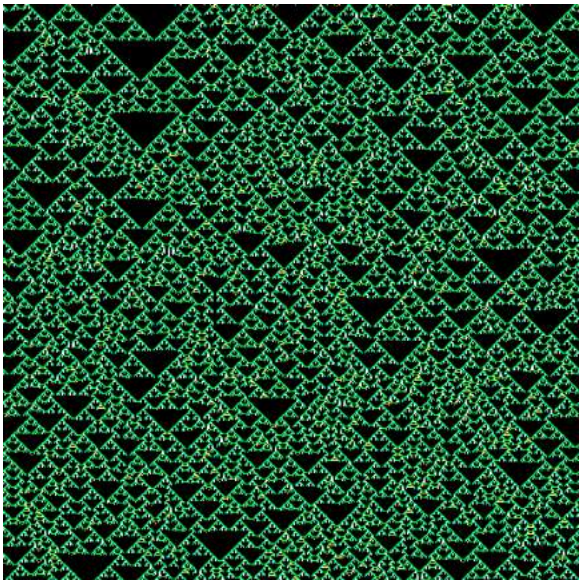


Figure 37. Found 1-D Automaton

We then used the same bounds on input entropy but looked for automata on neighborhoods of radius 3. Figures 38-43 are favorites arising in this manner.

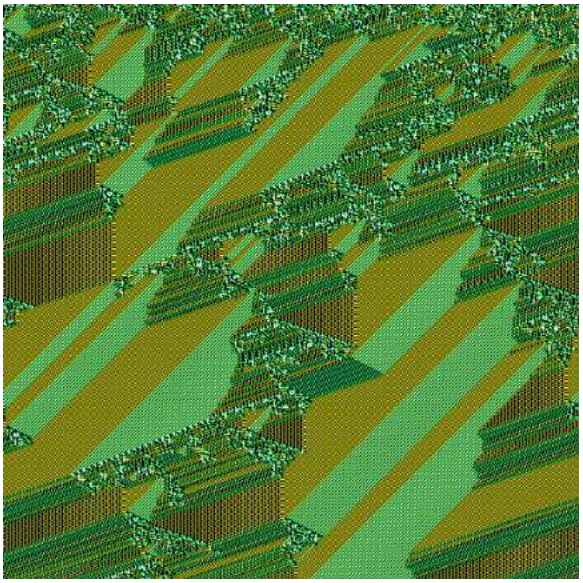


Figure 38. Found 1-D Automaton

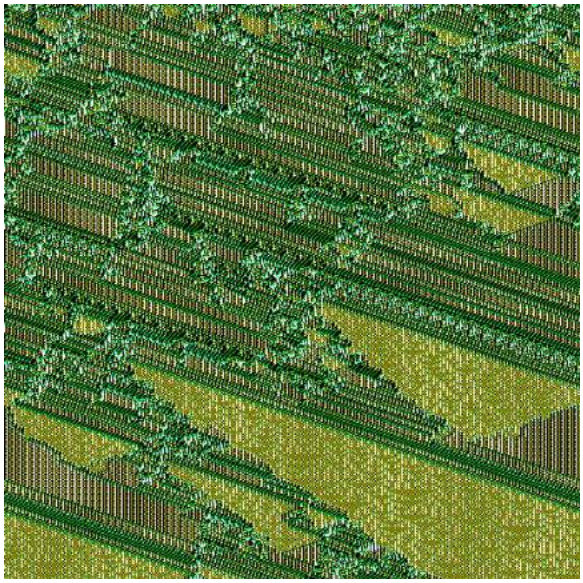


Figure 39. Found 1-D Automaton

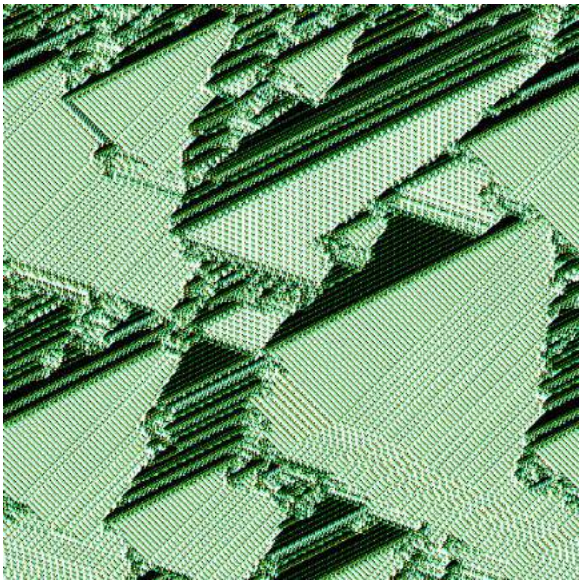


Figure 40. Found 1-D Automaton

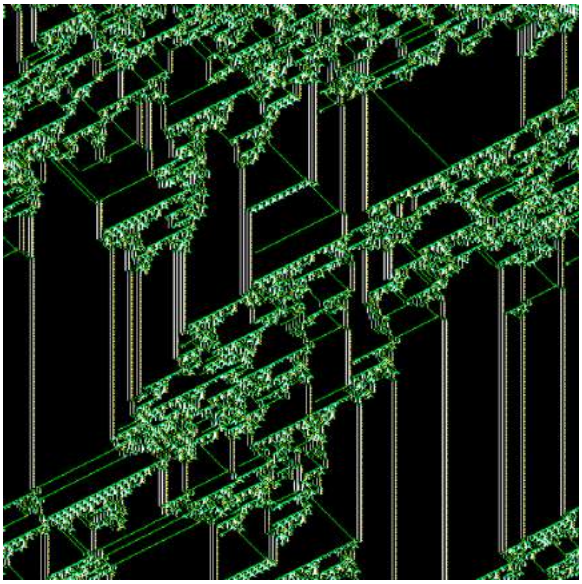


Figure 41. Found 1-D Automaton

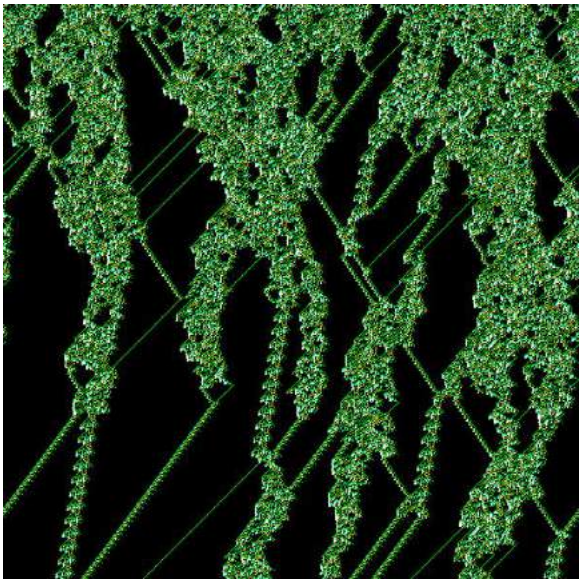


Figure 42. Found 1-D Automaton

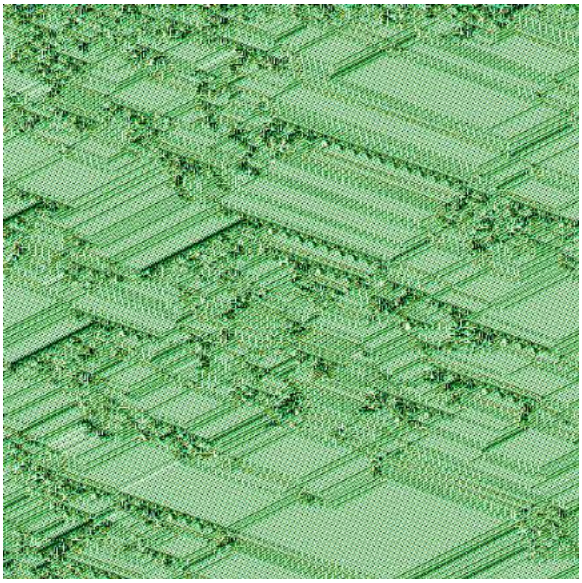


Figure 43. Found 1-D Automaton

Figures 44-47 show some examples with neighborhood radius 4. Note that chaotic regimes within complex behavior seem to be more common as the neighborhoods get wider.

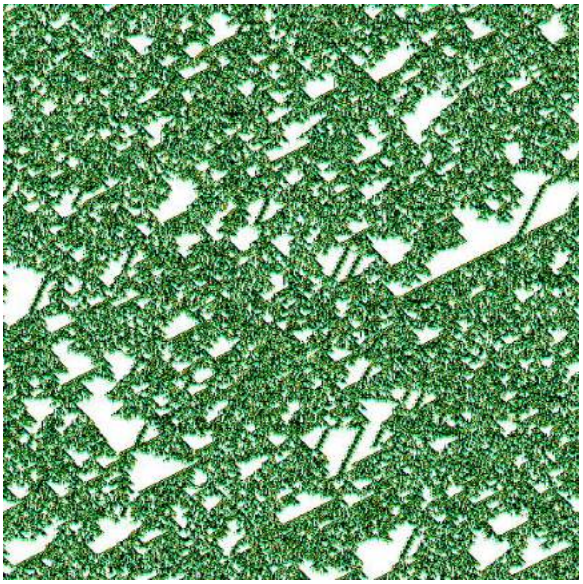


Figure 44. Found 1-D Automaton

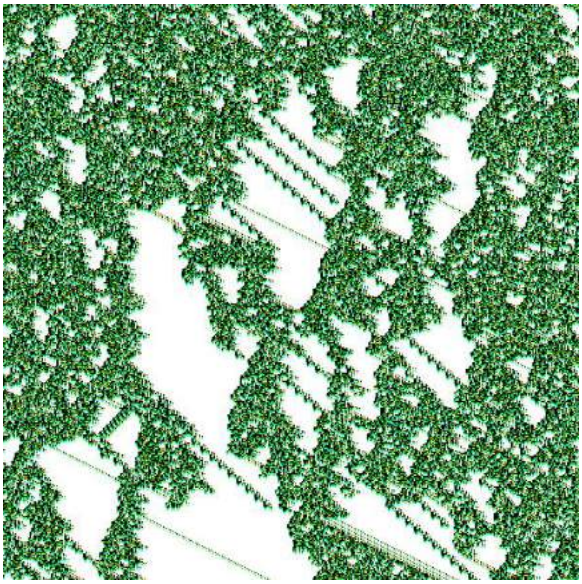


Figure 45. Found 1-D Automaton

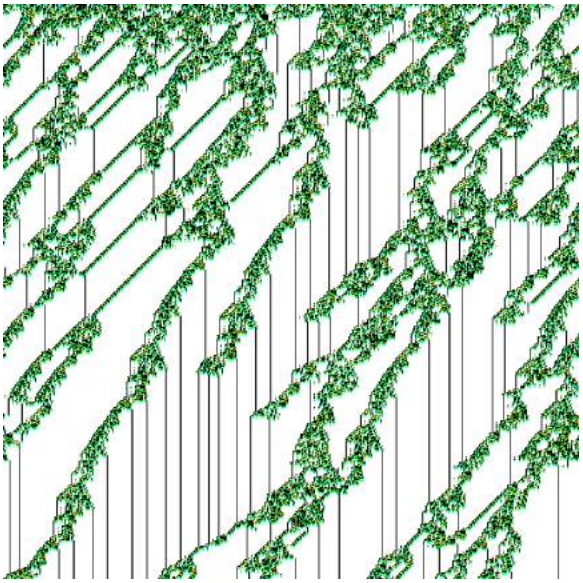


Figure 46. Found 1-D Automaton

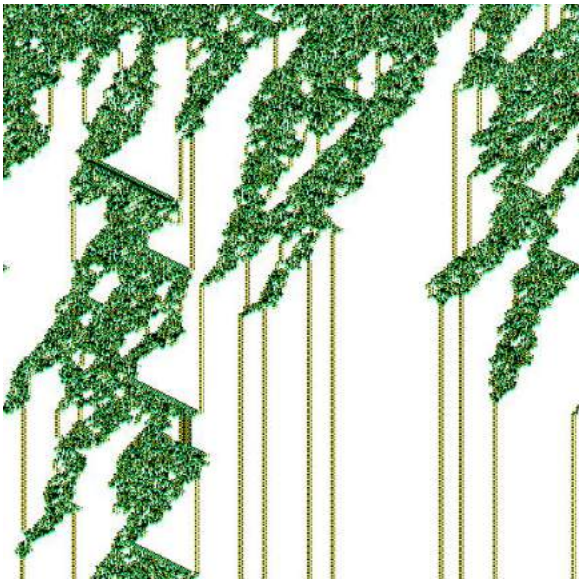


Figure 47. Found 1-D Automaton

Interestingly, overnight runs on neighborhoods of radius 5 resulted in no automata. There does not appear to be a computation limit that is being encountered, but rather the complex automata appear to be too sparse in uniform random Boolean rule space.

We turn briefly to discussing our conjunction for finding these (mostly) complex automata. Basically we run the conjunction `Fc1d` as below. Note that many of the lines of the conjunction are not essential. The average, `A`, and standard deviation, `S`, of the input entropy are assembled for the `x` examples produced. These can be especially helpful for giving guidance for the choices

of parameters of `n`, which specify the minimum standard deviation and minimum average which are useful for designing future experiments. The right argument `y` is a filename prefix for the figures produced. The foreign conjunction `1!:5` is used on the presumption that the filename prefix probably refers to a new folder in an existing folder. The foreign conjunction `5!:5` is used to store the actual version `Fc1d` used along with the values of `n` used since we found ourselves frequently tinkering with the initial and halting conditions. The value of `m` is embedded in the image filenames. Lastly an image gallery is created for the images found.

```
   Fc1d=:2 : 0
:
'mins mina'=.n
A=:S=:i.0
try. 1!:5<path_nm y catch. end.
('Fc1d=:',5!:5<'Fc1d')fwrite (path_nm y),'fc1d.ijs'
(LF,'n=:',":n)fappends (path_nm y),'fc1d.ijs'
for_k. i.x do.
  whilst. tq+.(mina>a) +.mins>s do.
    r=:?(2^1+2*m)#2
    b=:r m Autoevo ?.512#2
NB.    perq=:({: e. }:) _25{.b
    z=:m Ie b
    s=.stddev 256}.z
    a=.avg 256}.z
    tq=.+./((|."0 1~i.@#){:b)e."1 _]_25{.}:b
  end.
  S=:S,s
  A=:A,avg 256}.z
  fn=.y,'_r',(":m),'_',(nfmt k),'_s',(":<.0.5+100*s),'.png'
  t=:0 2 1{"1 ]255*(2}.b)* 1|:3]\ b
  (2 spix t) write_image fn
  ('r=:',":r) fwrite y,(nfmt k),'.ijs'
end.
3 mk_html_gallery 384 384 images_in path_nm y
)

   randomize''

   100 (2) Fc1d 0.12 2 'c:/temp/a/a'
15463
```

A gallery showing Figures 9-47 along with the corresponding rules as **.ijs* scripts appears at [17]. While we have enjoyed sharing some of the illustrations in this section, we encourage the reader to create their own galleries of presumably complex automata with the above tools or variants and then ponder their aesthetics and whether they seem truly complex.

4. Finding General 2-dimensional Automata with Complexity

We next turn to consider the case of general 2-dimensional Boolean automata. Thus, if the radius is `m`, then the width is `w=:1+2*m` and we need to specify a result for every possible `w` by `w` neighborhood. Note that for radius 1, we are using 3 by 3 neighborhoods and thus our rules will require 512 entries. It may not be practical to use a larger radius and we will content ourselves with the `m=:1` case. We load the same scripts as before, however, we need to define 2-dimensional analogs of a few of the utilities that we used. Below, both `Auto2d` and `Ie2` utilize `_3` cut as in `u;._3` in order to apply local rules on all 3 by 3 tessellations. We will illustrate this a bit further along once an illustrative rule is in place.

```
   load '~addons/graphics/fvj4/automata.ijs'
   load 'plot'
   load '~addons/stats/base/base.ijs'

   lauto2d=:lauto ,

   Auto2d=:1 : 0
:
(2#1+2*m)&(x&lauto2d;._3 f.)@:(m nperext2)y
)

   Autoevo2d=:1 : ' m Auto2d^:(i.@#@])'

   Ie2=:1 : 0("2)
-+/(*^.)(#/.~%#) ,/(2#1+2*m) ,;._3 m nperext2 y
)
```

We select a random rule and compute its result on a couple of neighborhoods. We then take a look at two dimensional input entropy `Ie2` on a random configuration and a periodic one.

```
   $r=:?.512#2
512

   32{.r
0 1 0 1 1 0 0 1 1 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 1 0 0 1 0 1

   |:(9#2)#:i.32
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
```

```
   r lauto2d 3 3$0
0

   r lauto2d 8=i.3 3
1

   ]y=:?.8 8$2
0 1 0 1 1 0 0 1
1 1 0 0 0 0 1 0
0 1 0 1 1 0 0 0
0 0 1 0 0 1 0 1
0 0 0 0 0 0 1 0
1 1 1 0 0 1 0 0
0 0 0 0 0 0 1 0
0 0 1 1 1 0 1 1

   3 3 <;._3 y
```

0 1 0 1 1 0 0 1 0	1 0 1 1 0 0 1 0 1	0 1 1 0 0 0 0 1 1	1 1 0 0 0 0 1 1 0	1 0 0 0 0 1 1 0 0	0 0 1 0 1 0 0 0 0
1 1 0 0 1 0 0 0 1	1 0 0 1 0 1 0 1 0	0 0 0 0 1 1 1 0 0	0 0 0 1 1 0 0 0 1	0 0 1 1 0 0 0 1 0	0 1 0 0 0 0 1 0 1
0 1 0 0 0 1 0 0 0	1 0 1 0 1 0 0 0 0	0 1 1 1 0 0 0 0 0	1 1 0 0 0 1 0 0 0	1 0 0 0 1 0 0 0 1	0 0 0 1 0 1 0 1 0
0 0 1 0 0 0 1 1 1	0 1 0 0 0 0 1 1 0	1 0 0 0 0 0 1 0 0	0 0 1 0 0 0 0 0 1	0 1 0 0 0 1 0 1 0	1 0 1 0 1 0 1 0 0
0 0 0 1 1 1 0 0 0	0 0 0 1 1 0 0 0 0	0 0 0 1 0 0 0 0 0	0 0 0 0 0 1 0 0 0	0 0 1 0 1 0 0 0 1	0 1 0 1 0 0 0 1 0
1 1 1 0 0 0 0 0 1	1 1 0 0 0 0 0 1 1	1 0 0 0 0 0 1 1 1	0 0 1 0 0 0 1 1 0	0 1 0 0 0 1 1 0 1	1 0 0 0 1 0 0 1 1

```
   ]Y=:3 3 r&lauto2d ;._3 ]1 nperext2 y
1 0 0 1 1 1 0 1
1 1 0 0 1 1 1 0
```

```
1 0 1 0 0 1 1 0
1 0 1 0 1 1 1 1
0 0 0 1 0 1 1 0
0 0 0 0 1 0 1 1
1 1 0 0 1 1 1 0
0 0 1 1 1 0 1 0

   Y-:r 1 Auto2d y
1
   1 Ie2 ?.256 256$2
6.23425

   1 Ie2 256 256$1 0 0 0 1 0 0
2.03195
```

We now turn to the conjunction that allows us to search for general 2-dimensional automata with complexity. During our first attempt at this we sought examples where the standard deviation was greater than 0.03. We found 100 examples during a week and a half that we were convalescing from a knee replacement. The examples suggest that we should seek higher standard deviation. We also expect any examples found using uniform choices for rule values will be far more difficult than in other sections. Subsequent to that, we found examples far more easily if we biased the rules so that the probability a rule value was 1 was 40%, 30% or 20%. Thus, we recommend the following conjunction and experiment as a starting point. While we will share examples of favorite illustrations we found, the reader is encouraged to experiment and to discover your own examples.

```
   Fc2d=:2 : 0
:
'mins mina'=.n
A=:S=:i.0
try. 1!:5<path_nm y catch. end.
('Fc2d=:',5!:5<'Fc2d')fwrite (path_nm y),'fc2d.ijs'
(LF,'m=:',":m)fappends (path_nm y),'fc2d.ijs'
(LF,'n=:',":n)fappends (path_nm y),'fc2d.ijs'
for_k. i.x do.
  whilst. perq +. (mina>a) +.mins>s do.
NB.    r=:?(2^*:1+2*m)#2
    r=:0.7<?(2^*:1+2*m)#0
    b=:r m Autoevo2d ?.128 128$2
    perq=:({: e. }:) _25{.b
    z=:m Ie2 b
    s=.stddev 28}.z
    a=.avg 28}.z
  end.
  S=:S,s
  A=:A,a
  B=:_3{.b
```

```
  B=:_3{.r m Autoevo2d ?.256 256$2
  t=:0 2 1{"1 ]255*({:B)* 0|: B
  fn=.y,(nfmt k),'s',(nfmt <.0.5+100*s),'.png'
  (2 spix t) write_image fn
  ('r=:',":r) fwrite y,(nfmt k),'.ijs'
end.
3 mk_html_gallery 384 384 images_in path_nm y
)

   randomize''

   100 (1) Fc2d 0.08 0.2 'c:/temp/a/a'
```

The final configuration for several illustrations when the rule probability is 50% are shown in Figures 48-52. Animated *.gif files showing the time evolution may be found at [17] along with the *.ijs files giving the rules. The coloring scheme is as in the previous section in that it is black if the cell is dead but gives some information about whether the cell is alive at the previous two iterates. A moving configuration on a black background will tend to have green on its leading edge and red on its trailing edge. Such features are far more obvious in the animations. Figure 48 evolves into every configuration moving west. Figure 49 is a rare examples where gliders moving in many directions evolve though that is difficult to observe casually. Figure 50 shows south moving droplets on a white background, which is not expected for rules with a bias below 50%. Figure 51 shows a fairly typical starry night stable configuration while Figure 52 shows a rare variant where only double stars develop.

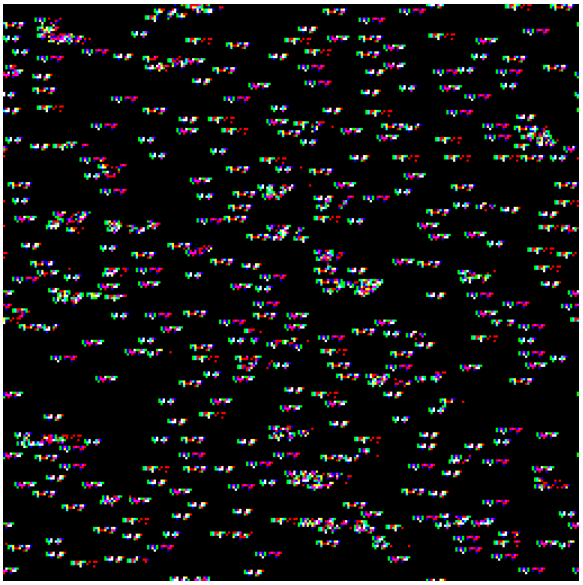


Figure 48. Found 2-dimensional Automaton

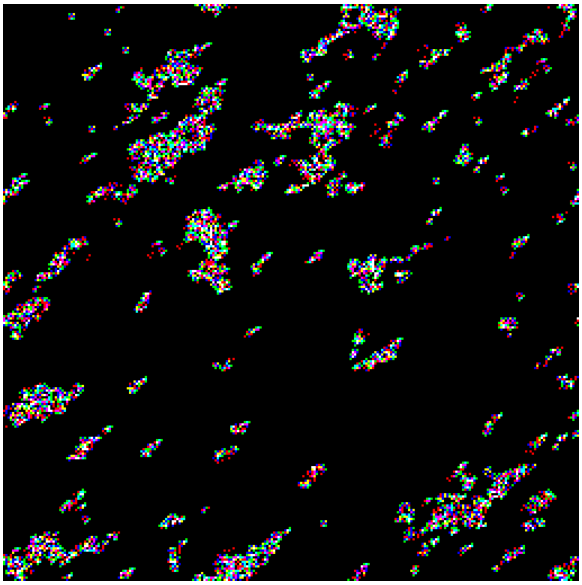


Figure 49. Found 2-D Automaton

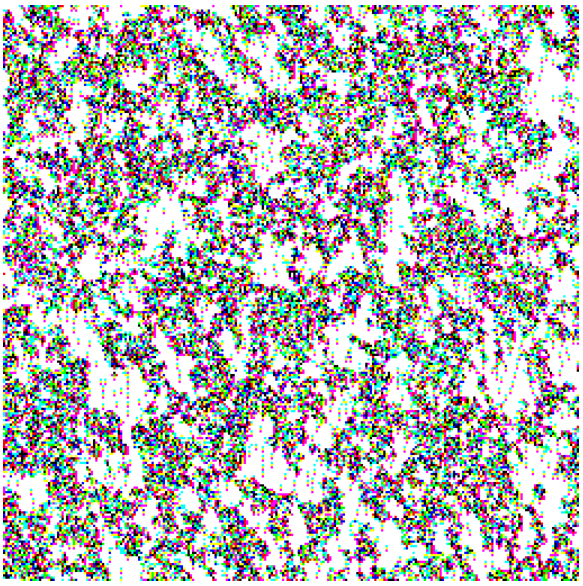


Figure 50. Found 2-D Automaton

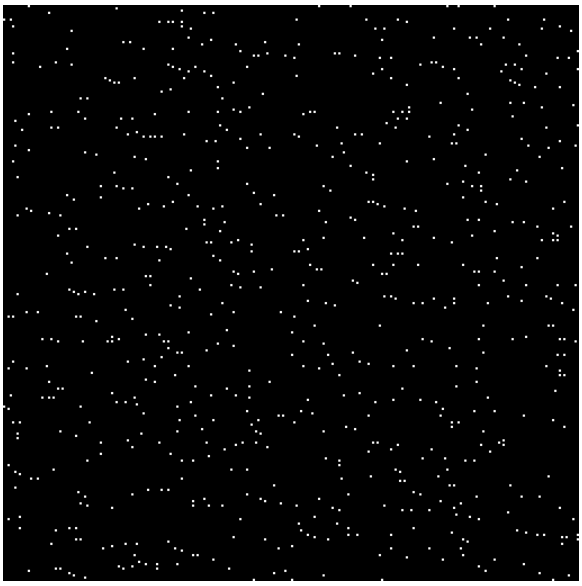


Figure 51. Found 2-D Automaton

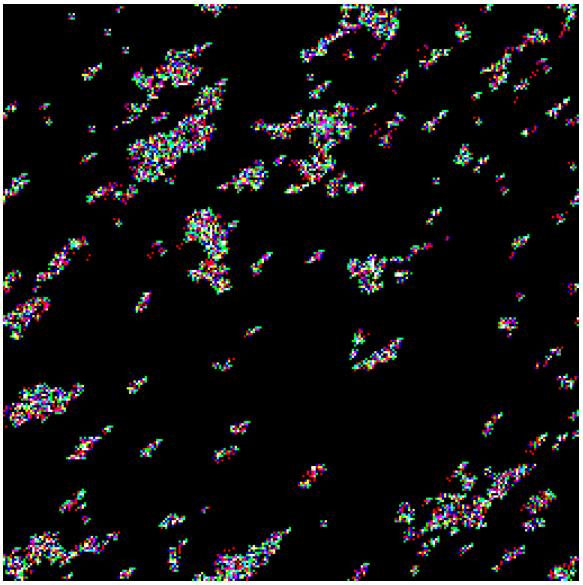


Figure 52. Found 2-D Automaton

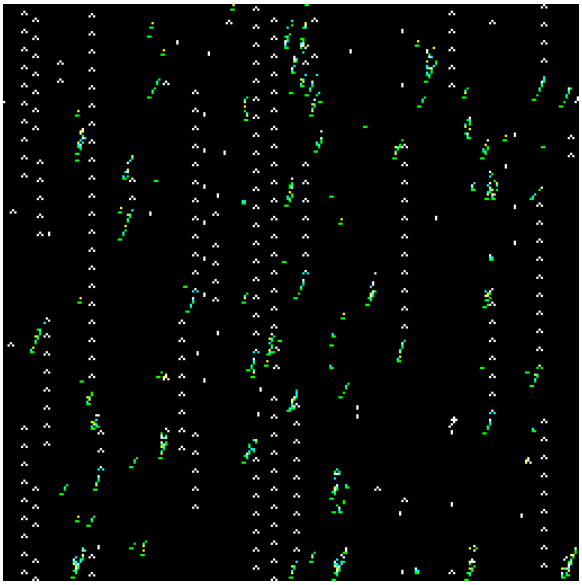


Figure 53. Found 2-D Automaton

In Figures 53-56 we look at some final configurations that arise when the probability of a rule value 1 is 60%. Figure 53 shows south moving configurations leaving behind trails of stable triples, among other features. Figure 54-55 shows configurations that rapidly move north with some intriguing tails that are chaos like. Figure 56 shows simple configurations moving west leaving trails.

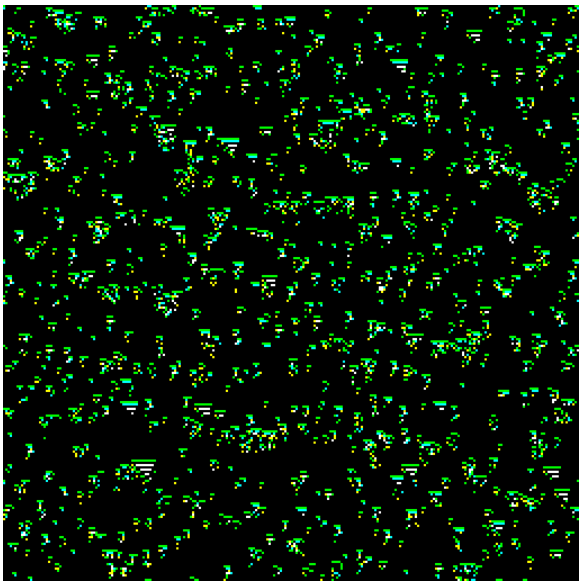


Figure 54. Found 2-D Automaton

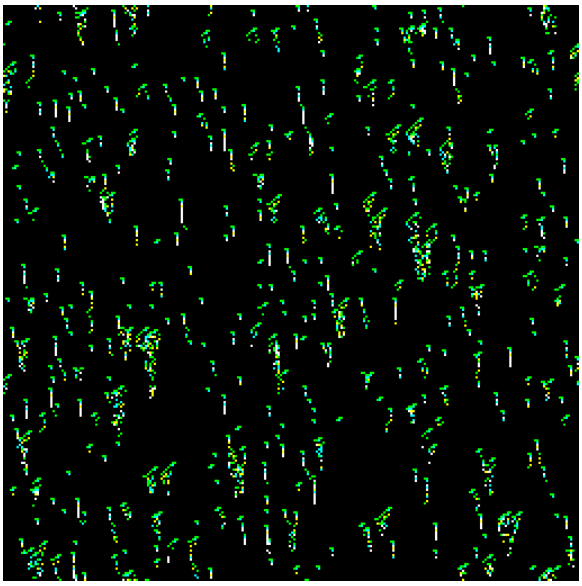


Figure 55. Found 2-D Automaton

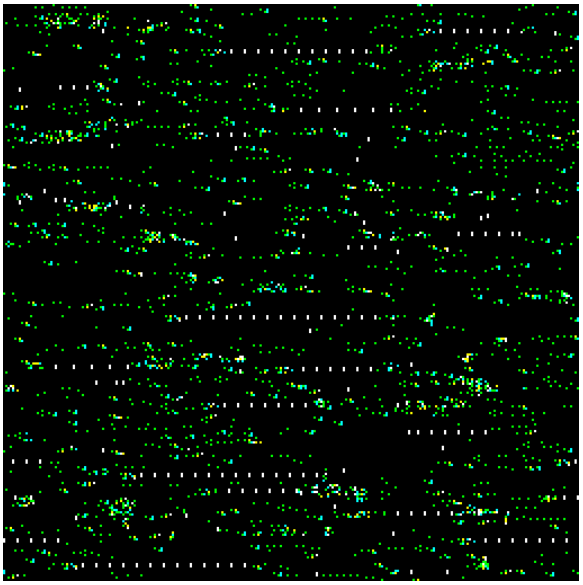


Figure 56. Found 2-D Automaton

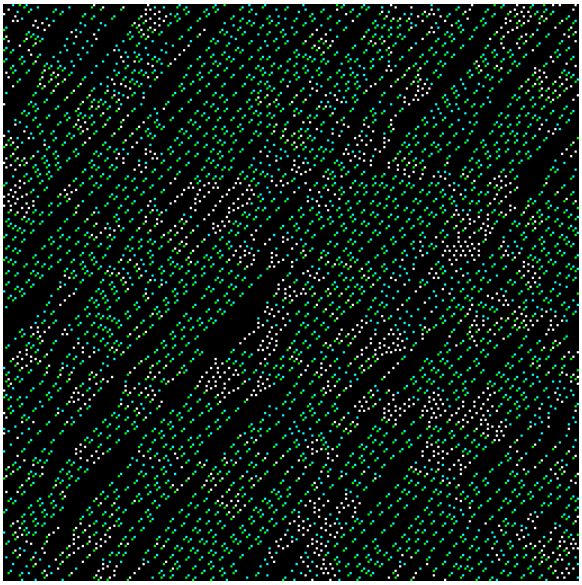


Figure 57. Found 2-D Automaton

Figures 57-61 show rules where the probability was 70%. Figure 57 shows southwest flows interacting with stable trails. In Figure 58 patterned regions self-organize with chaotic behavior along boundaries. In Figures 59-60 lots moves south but stable diagonal patterns self-organize and grow and diminish. Figure 61 seems chaotic but alternate periodic patterns develop.

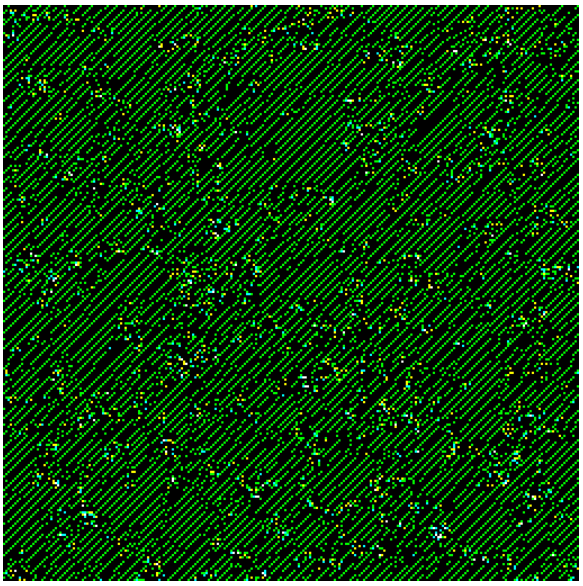


Figure 58. Found 2-D Automaton

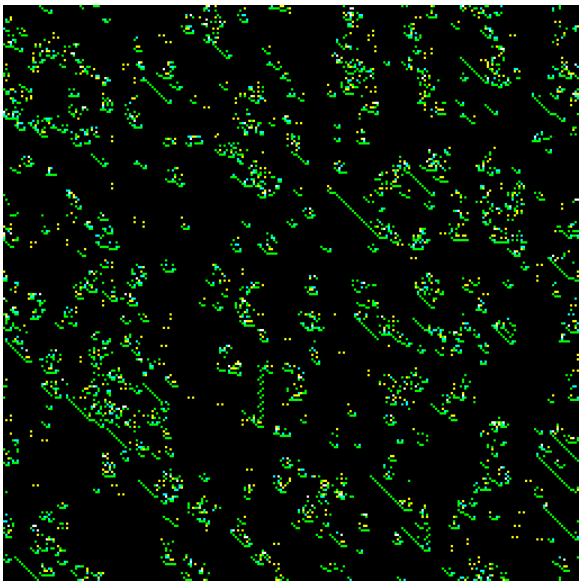


Figure 59. Found 2-D Automaton

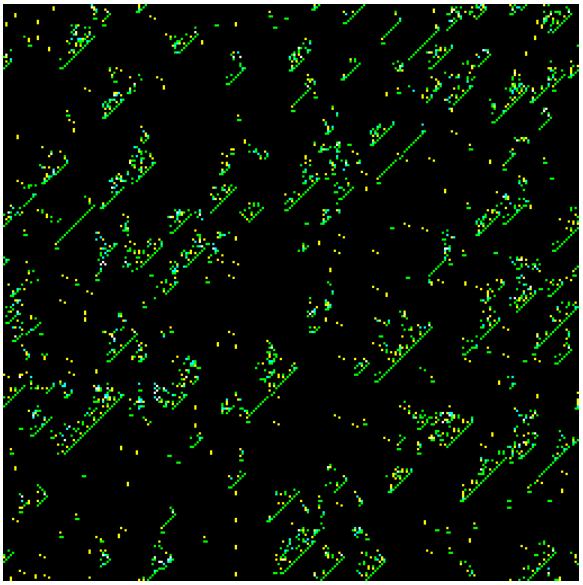


Figure 60. Found 2-D Automaton

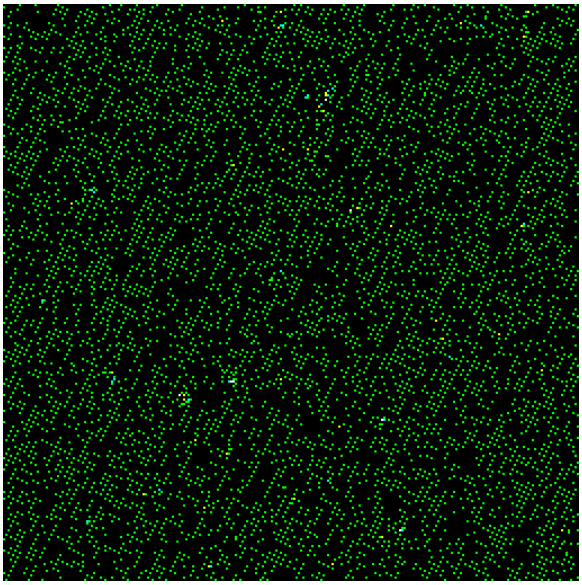


Figure 61. Found 2-D Automaton

Figures 62-66 show configurations when the probability an initial cell is alive is 20%. In Figure 62 southwest flows organize. In Figure 63 horizontal stable regions appear with action on the boundary. In Figure 64 southeast flows develop with trails. In Figure 65 northwest flows with dendritic trails develop. In Figure 66 self-organized regions appear with action occurring on the boundaries.

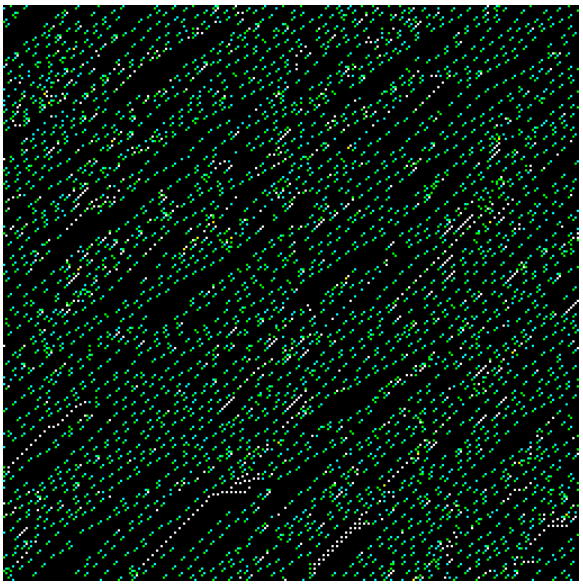


Figure 62. Found 2-D Automaton

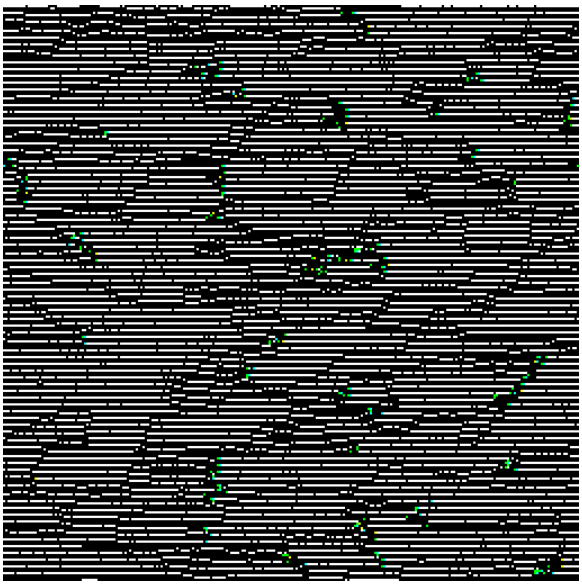


Figure 63. Found 2-D Automaton

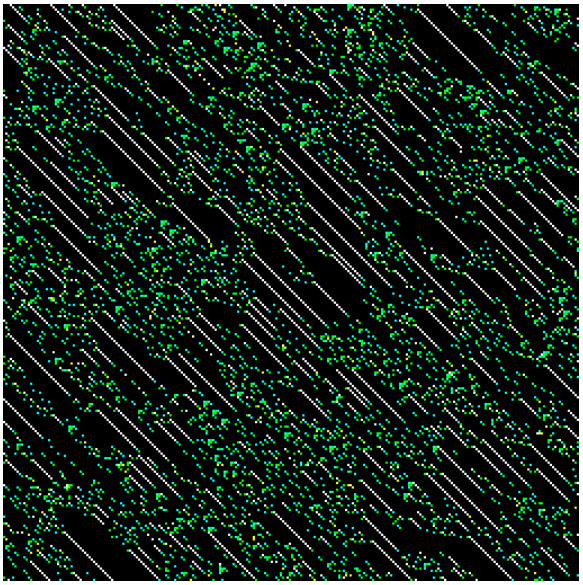


Figure 64. Found 2-D Automaton

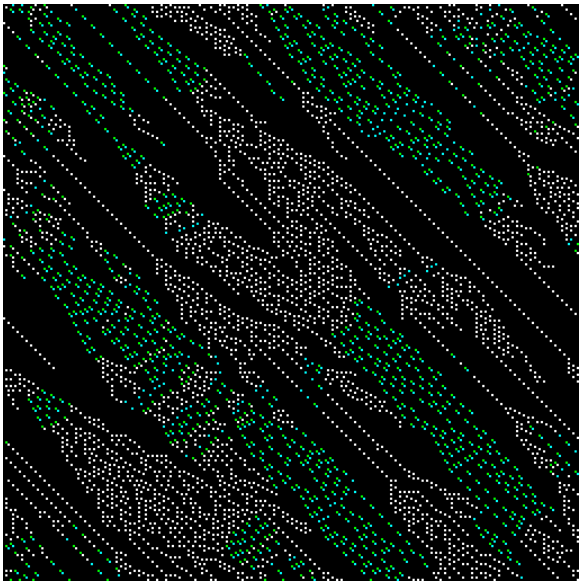


Figure 65. Found 2-D Automaton

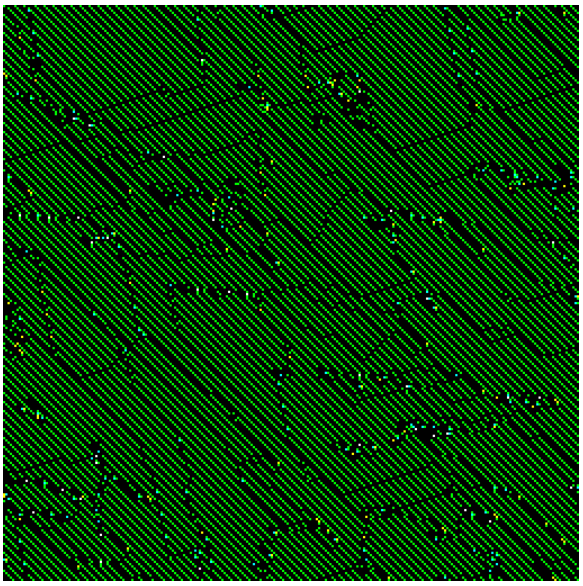


Figure 66. Found 2-D Automaton

While the general random 2-dimensional automata show a great deal of interesting behavior, there is a tendency to want rules that somehow respect the symmetry of the lattice. That is, if a pattern can move south, it seems reasonable to want analogous patterns to move north, east and west; however, this is not a consequence of a general automata. We will see in Section 6 that Larger than Life automata have that feature. Before that, we will consider 1-dimensional Larger than Life automata.

5. Finding Larger than Life 1-dimensional Automata with Complexity

A 1-dimensional Larger than Life automaton [1,3-7,10-12,15-16] depends upon 5 parameters: r, a, b, c, d, such that neighborhoods have radius r and width $1+2r$. If s denotes the number of alive cells in a neighborhood (including the cell itself) then a dead cell becomes alive if $a \leq s \leq b$ and an alive cell remains alive if $c \leq s \leq d$. It turns out this is easily generalized to 2 dimensions and the classic Conway's Game of Life [2,8,9] is a 2-dimensional 1 3 3 3 4 Larger than Life automaton, whence the name. Nevertheless, we will begin with considering the 1-dimensional version.

Below we load some scripts and remind ourselves of the adverb for input entropy. The larger than life automata script (*ltl.ijs*) defines an adverb `LtL1d` that implements one step of the one-dimensional larger than life automaton specified by its adverb argument which specify the 5 larger than life parameters. We then illustrate a small example.

```
   load '~addons/graphics/fvj4/ltl.ijs'
   load 'plot'
   load '~addons/stats/base/base.ijs'

   Ie=:1 : 0("1)
-+/(*^.)(# %~ [: #/.~(1+2*m) ]\])m nperext y
)
```

```
   LtL1d
1 : 0
'r a b c d'=.m
nl=.(a&<: *. <:&b)@:(+/)
rl=.(c&<: *. <:&d)@:(+/)
cen=.r&{
lltl=.(nl`rl@.cen)
(1+2*r)&(lltl;._3 f.)@:(r nperext)
)

   5 <;._3 ]2 nperext 0 0 1 0 1 1 0
```

1 0 0 0 1	0 0 0 1 0	0 0 1 0 1	0 1 0 1 1	1 0 1 1 0	0 1 1 0 0	1 1 0 0 0

```
   2 1 1 3 3 LtL1d 0 0 1 0 1 1 0
0 1 0 0 1 0 0
```

We next display the conjunction `Fc1d_ltl` that can search for good choices of parameters for a specified radius. We suggest a search choice and then turn to sharing some examples that technique found.

```
   Fc1d_ltl=:2 : 0
:
'mins mina maxa'=.n
A=:S=:i.0
try. 1!:5<path_nm y catch. end.
('Fc1d_ltl=:',5!:5<'Fc1d_ltl')fwrite (path_nm y),'fc1d_ltl.ijs'
(LF,'m=:',":m)fappends (path_nm y),'fc1d_ltl.ijs'
(LF,'n=:',":n)fappends (path_nm y),'fc1d_ltl.ijs'
for_k. i.x do.
  whilst. tq+.(a>maxa)+.(mina>a) +.mins>s do.
    p=:m,+/\?4#1+>.-:m
    b=:p LtL1d^:(i.512) ?.512#2
NB.    perq=:({: e. }:) _25{.b
    z=:m Ie b
    s=.stddev 256}.z
    a=.avg 256}.z
    tq=.+./(((|."0 1~i.@#){:b)e."1 _]_25{.}:b
  end.
  S=:S,s
  A=:A,a
  fn=.y,(' _' stringreplace ": p),'_s',(":<.0.5+100*s),'.png'
  t=:0 2 1{"1 ]255*(2}.b)* 1|:3]\ b
  (2 spix t) write_image fn
end.
3 mk_html_gallery 384 384 images_in path_nm y
```

```
)

   randomize ''
_953397380 1215314363 420407457 744247738

   100 (5) Fc1d_ltl 0.18 2.8 _ 'c:\temp\ltl5\ltl5'
```

Note that the parameters, except the radius `m` which is given, are computed by `p=:m,+/\?4#1+>.-:m` so that we get four numbers on average near half the radius and then take the partial sum to get the four required bounds. There is no reason to require that $b \leq c$ so this does not fully explore parameter space.

Just three radius three LtL automata were found using an analogous command. They include the automaton in Figure 67 which seems possibly chaotic while the automaton in Figure 68 is quite complex.

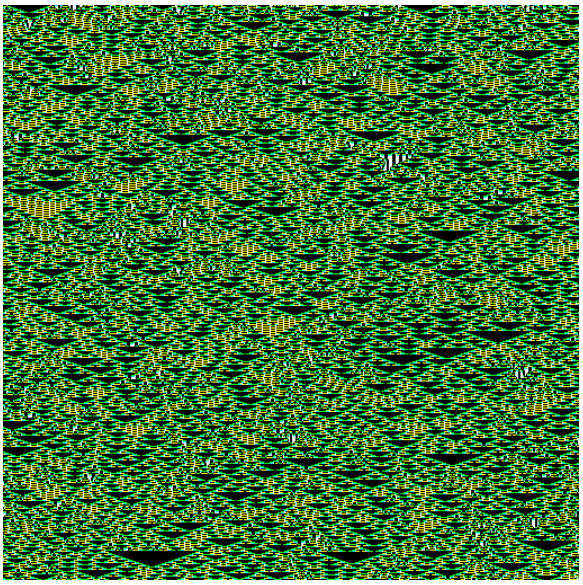


Figure 67. Larger than Life 1-D: 3 1 3 3 4

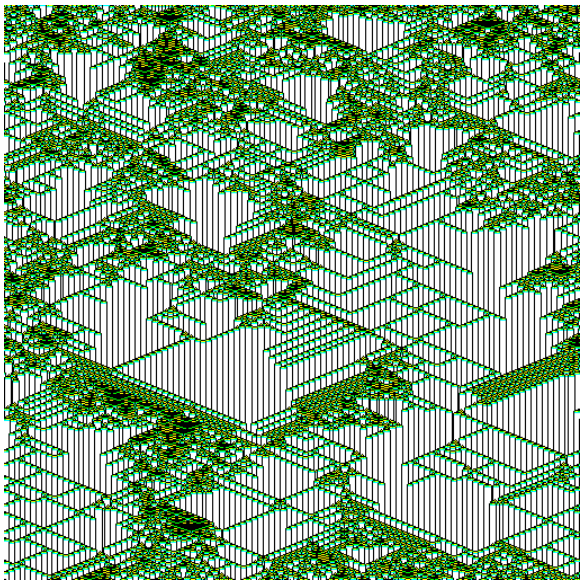


Figure 68. Larger than Life 1-D: 3 1 3 5 6

Once interesting parameters for a Larger than Life automaton have been identified, it is easy to create an array of its time evolution that keeps track of the most recent time that a cell was alive. See [15, 16] for other examples. Below we illustrate this with the 3 1 3 5 6 automaton from Figure 68 and the result is shown in Figure 69. Note trailing "t" in `LtL1dt` which stands for time.

```
   p=:3 1 3 5 6

   b=:p LtL1dt^:(i.512) ?.512#2
```

```
   >./,b
512

   10 10{.b
0 1 0 1 1 0  0  1  1  1
0 0 2 0 0 0  0  0  0  0
3 3 0 3 3 3  0  3  3  3
0 3 0 3 3 3  0  3  0  0
5 0 0 0 3 0  0  0  5  5
0 0 6 6 0 6  6  0  0  0
7 7 0 0 0 0  0  7  7  7
0 0 8 8 8 8  8  0  0  0
9 9 0 8 8 8  0  0  0  0
0 9 0 8 0 0 10 10 10 10

   view_image t=.P254;254<.<.-:1+b
```

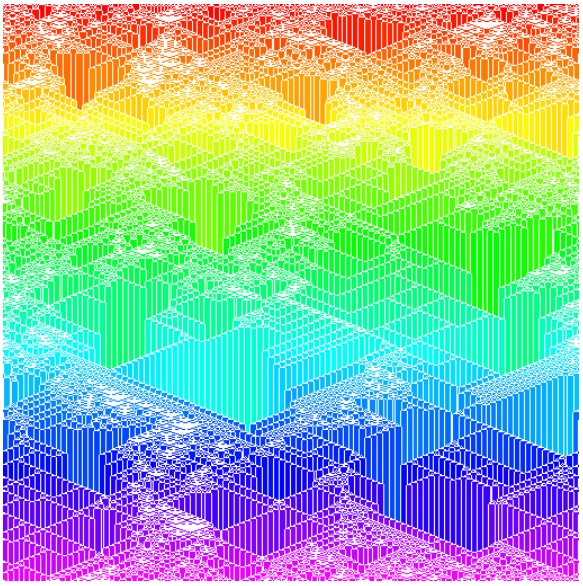


Figure 69. Larger than Life 3 1 3 5 6 by Time to Life

The search for radius 3 illustrations missed the very similar automaton with parameters 3 1 3 5 7. However, it also found the example in Figure 70 which shows regimes with different qualitative behavior. Other favorite examples with radius 4 and 5 appear in Figures 71-75. In Figure 74 notice that because the windows are width 11, information can move more quickly than in the examples with smaller width.

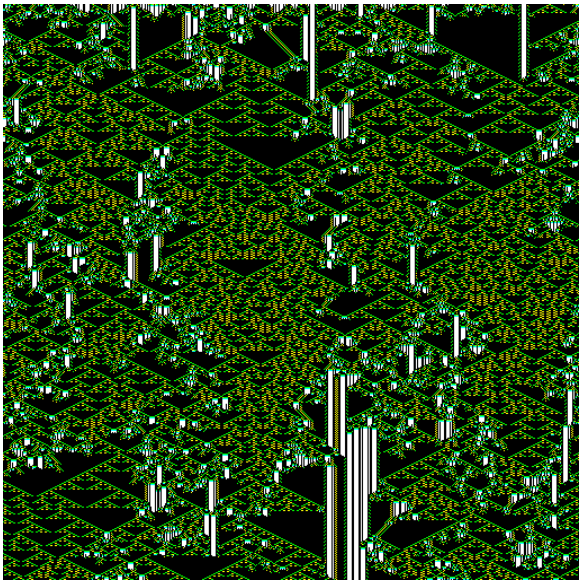


Figure 70. Larger than Life 1-D: 3 2 2 4 5

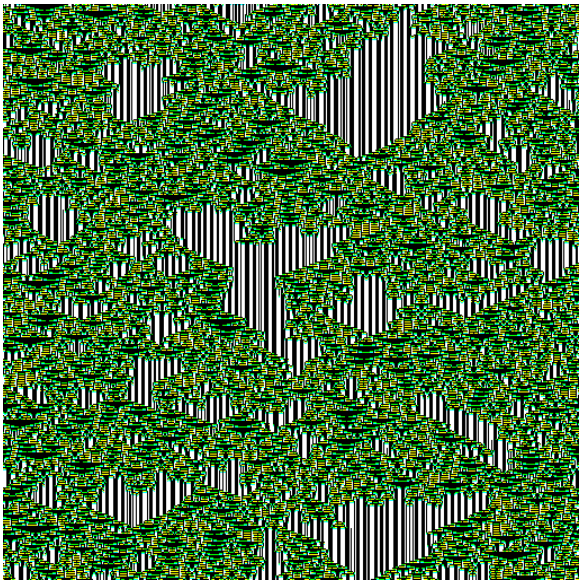


Figure 71. Larger than Life 1-D: 4 1 3 3 5

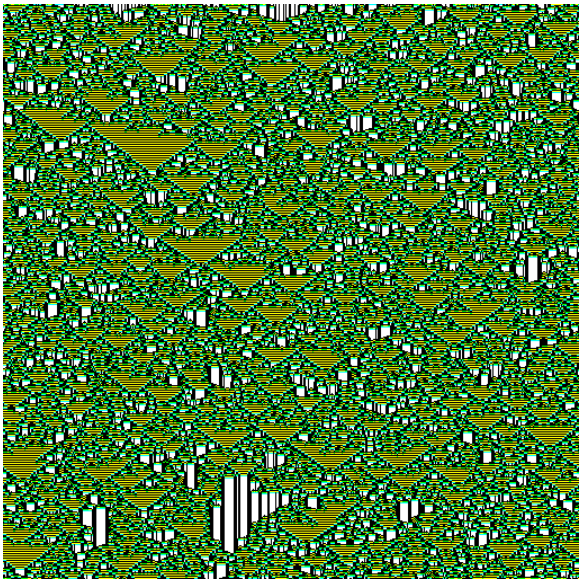


Figure 72. Larger than Life 1-D: 5 0 3 6 8

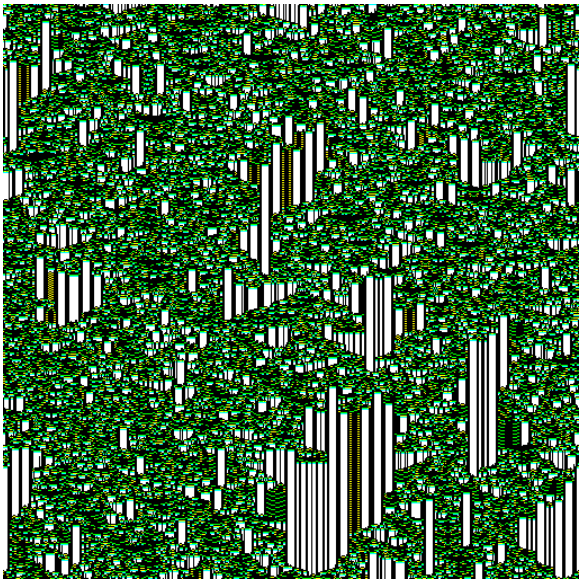


Figure 73. Larger than Life 1-D: 5 2 3 6 8

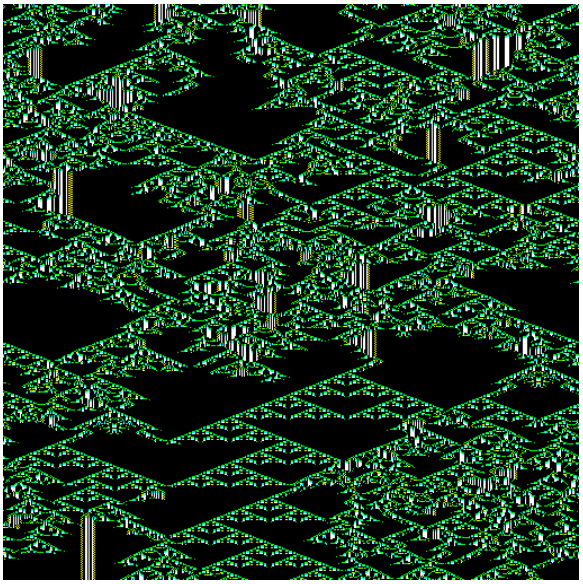


Figure 74. Larger than Life 1-D: 5 3 3 4 5

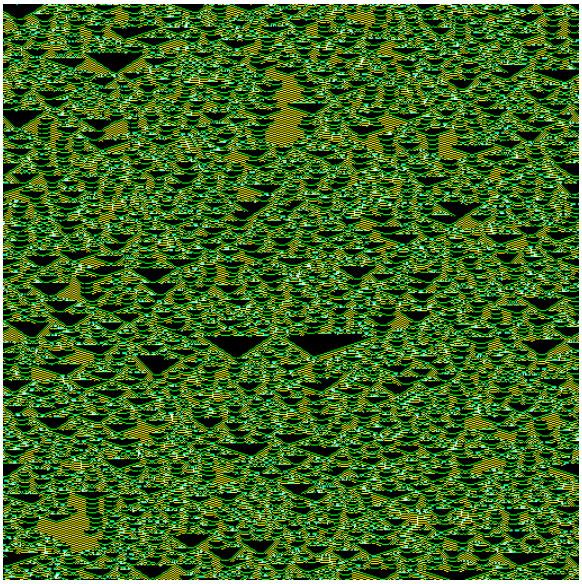


Figure 75. Larger than Life 1-D: 5 3 6 6 6

Figure 76 shows a periodic configuration that slips in. However, the period is large and beyond our test for translations. In fact, the period is 14,160 which is well beyond the image shown. Figures 77-85 show favorite examples when the radius is between 6 and 9. Notice the gliders, especially in Figure 82 and 85.

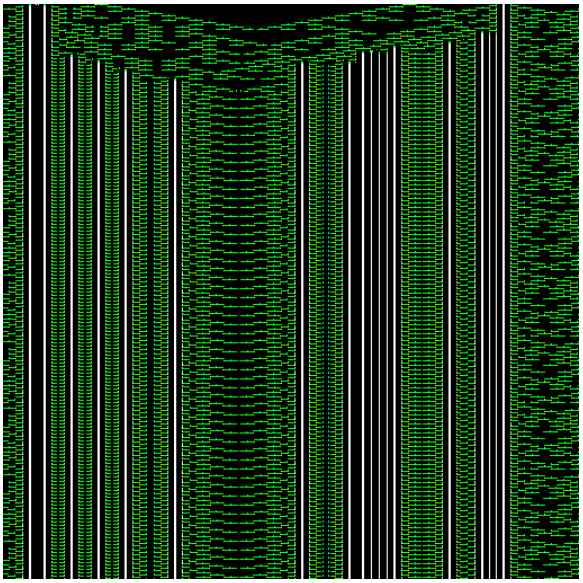


Figure 76. Larger than Life 1-D: 6 1 1 1 3

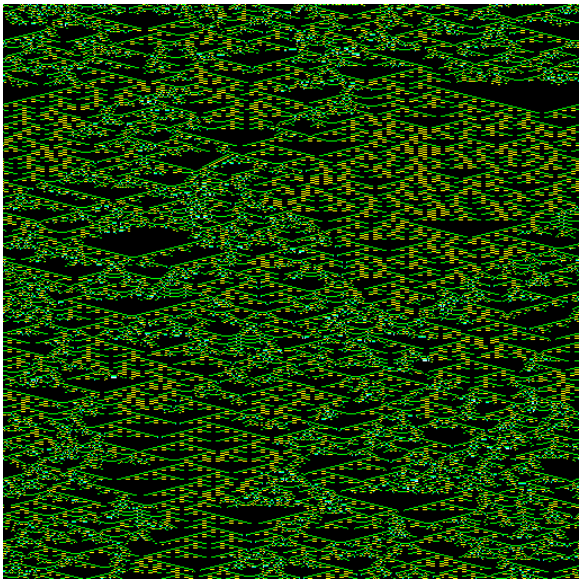


Figure 77. Larger than Life 1-D: 6 3 4 7 7

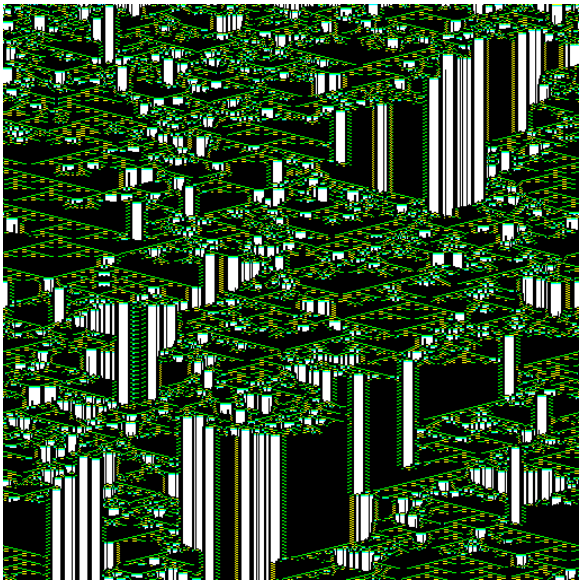


Figure 78. Larger than Life 1-D: 6 3 4 7 10

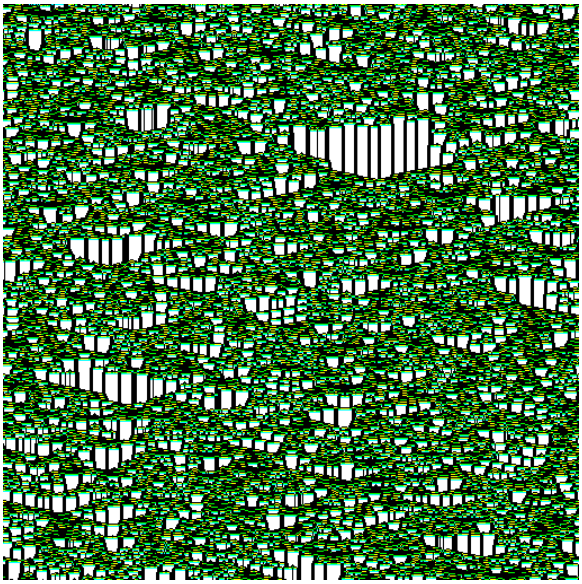


Figure 79. Larger than Life 1-D: 7 2 5 9 12

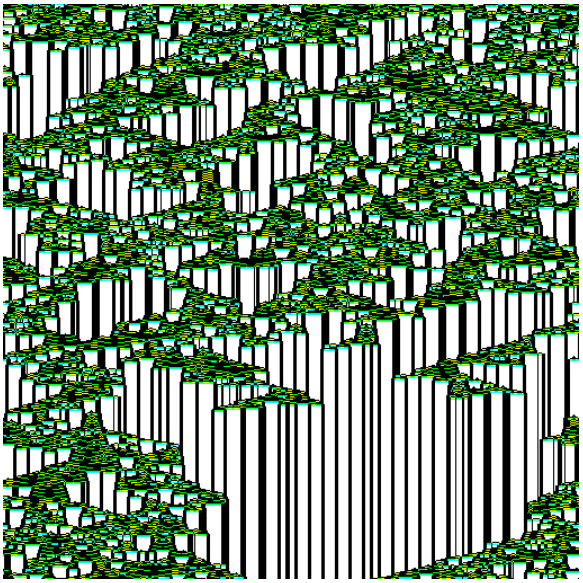


Figure 80. Larger than Life 1-D: 8 3 6 10 14

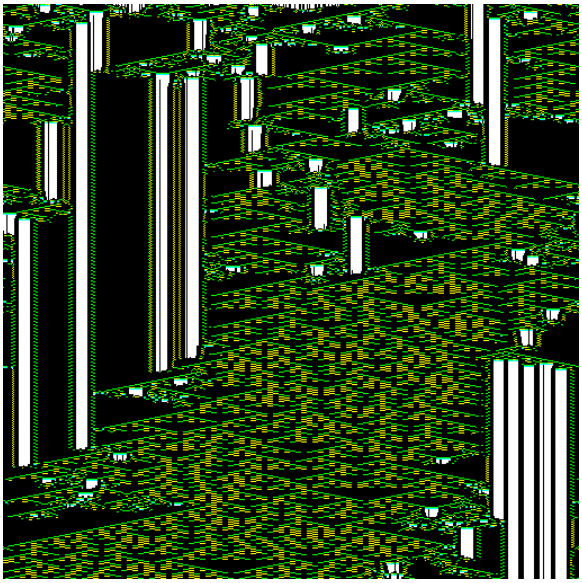


Figure 81. Larger than Life 1-D: 8 4 5 9 13

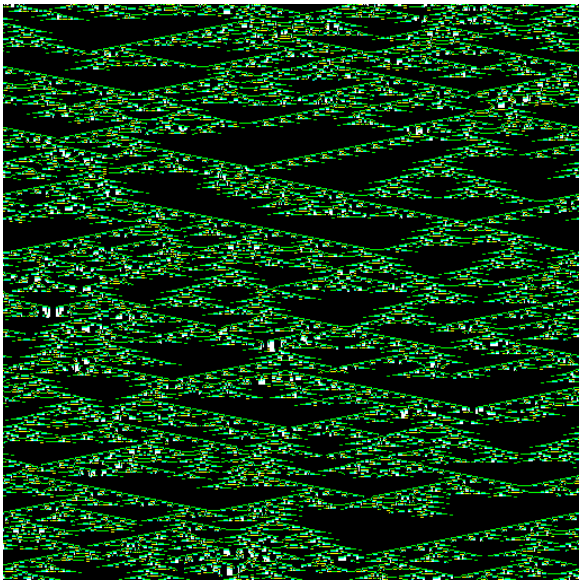


Figure 82. Larger than Life 1-D: 8 4 6 6 8

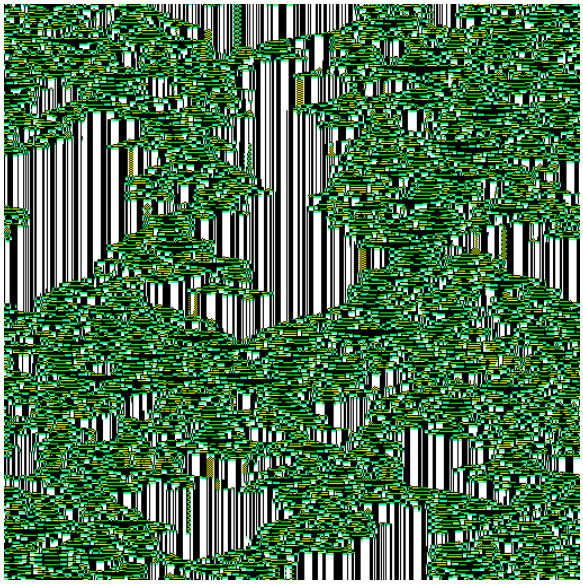


Figure 83. Larger than Life 1-D: 9 3 7 8 11

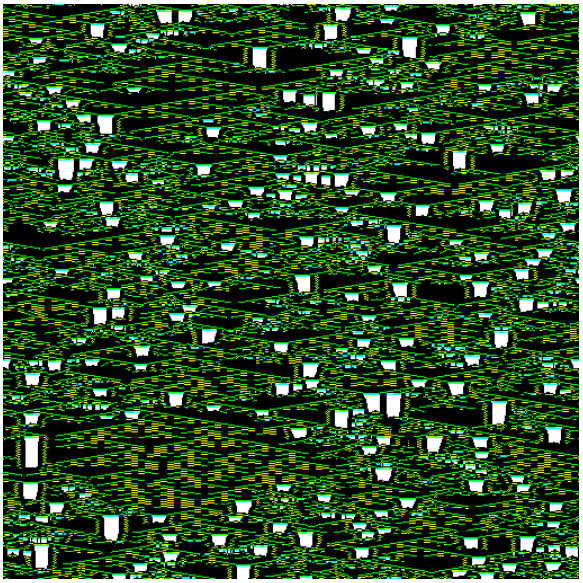


Figure 84. Larger than Life 1-D: 9 4 6 11 15

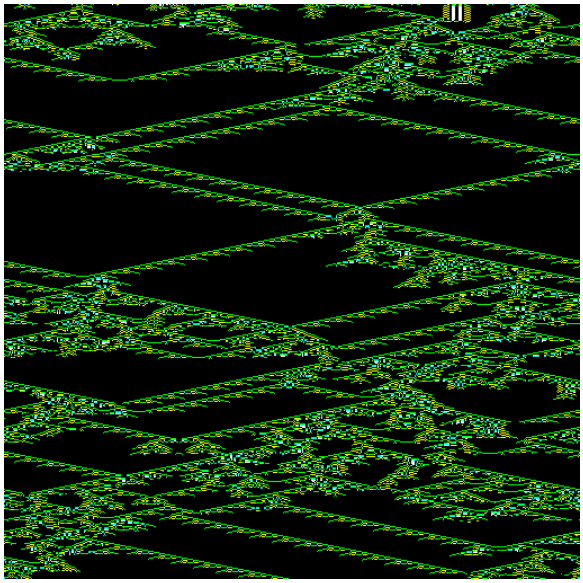


Figure 85. Larger than Life 1-D: 9 5 7 9 10

Favorites with radius 10 to 12 appear in Figures 86-89. Notice rapid movement of information in each case.

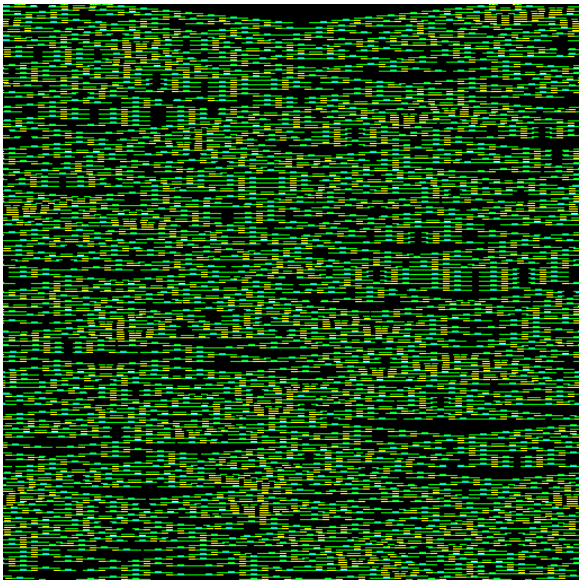


Figure 86. Larger than Life 1-D: 10 1 6 6 6

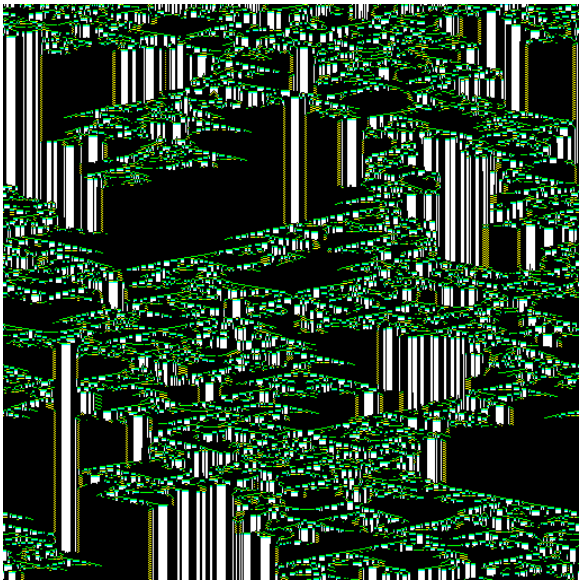


Figure 87. LtL 1-D: 10 5 6 8 11

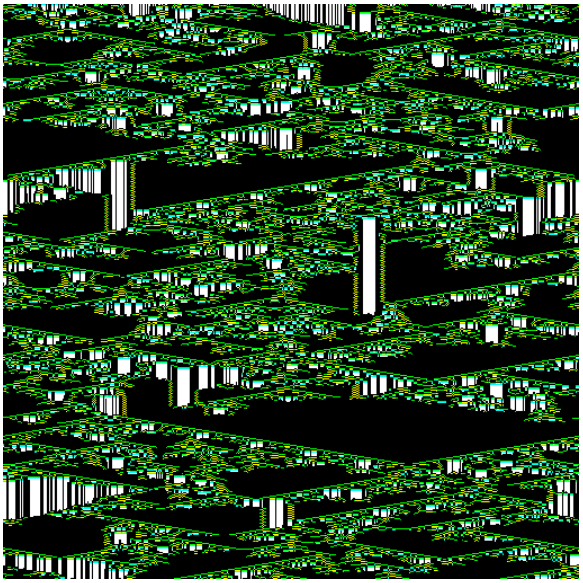


Figure 88. Larger than Life 1-D: 11 6 8 11 15

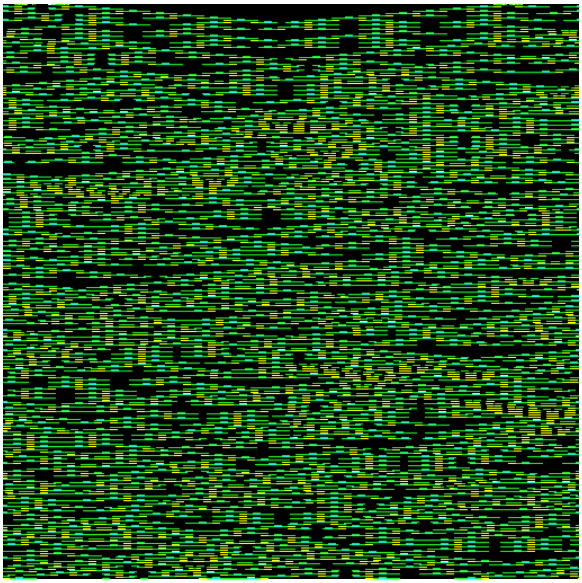


Figure 89. Larger than Life 1-D: 12 1 7 7 7

6. Finding Larger than Life 2-dimensional Automata with Complexity

In order to search for 2-dimensional Larger than life automata that exhibit complex behavior, we load the scripts below as well as define the adverb `Ie2` that we used to compute 2-dimensional input entropy. The adverb `LtL` applies an iteration of the automata with the parameters given as the adverb argument. We then apply the automaton with parameters 1 3 3 3 4 to a small example. As noted before, this is the classic Conway's Game of Life.

```
   load '~addons/graphics/fvj4/ltl.ijs'
   load 'plot'
   load '~addons/stats/base/base.ijs'

   Ie2=:1 : 0("2)
-+/(*^.)(#/.~%#) ,/(2#1+2*m) ,;._3 m nperext2 y
)

   LtL
1 : 0
'r a b c d'=.m
nl=.(a&<: *. <:&b)@:(+/)
rl=.(c&<: *. <:&d)@:(+/)
cen=.(2*r*r+1)&{
lltl=.(nl`rl@.cen)@,
(2#1+2*r)&(lltl;._3 f.)@:(r nperext2)
)
   ]a=:?.4 4$2
0 1 0 1
1 0 0 1
1 1 0 0
0 0 1 0

   3 3<;._3 ]1 nperext2 a
```

0 0 0 1 0 1 1 1 0	0 0 1 0 1 0 1 0 0	0 1 0 1 0 1 0 0 1	1 0 0 0 1 0 0 1 1
1 0 1 1 1 0 0 1 1	0 1 0 1 0 0 1 1 0	1 0 1 0 0 1 1 0 0	0 1 0 0 1 1 0 0 1
1 1 0 0 1 1 0 0 0	1 0 0 1 1 0 0 0 1	0 0 1 1 0 0 0 1 0	0 1 1 0 0 1 1 0 0
0 1 1 0 0 0 1 0 1	1 1 0 0 0 1 0 1 0	1 0 0 0 1 0 1 0 1	0 0 1 1 0 0 0 1 0

```
   1 3 3 3 4 LtL a
0 1 0 1
0 0 0 1
1 1 1 0
0 0 1 1
```

We searched for 2-dimensional LtL automata with complex behavior using `Fc2d_ltl` and illustrated below. Again, we note that our choice of parameters does not fully explore parameter space. We did searches with radii ranging from 2 to 15 as applied to three types of random initial arrays: 128 iterates on 128 by 128 array with probability of 1 being 0.5 (***standard***), 256 iterates on a 256 by 256 array with probability of 1 being 0.5 (***wide***), and 256 iterates on a 256 by 256 array with probability of 1 being 0.3 (***biased***).

```
Fc2d_ltl=:2 : 0
:
'mins mina'=.n
A=:S=:i.0
try. 1!:5<path_nm y catch. end.
('Fc2d_ltl=:',5!:5<'Fc2d_ltl')fwrite (path_nm y),'fc2d_ltl.ijs'
(LF,'m=:',":m)fappends (path_nm y),'fc2d_ltl.ijs'
(LF,'n=:',":n)fappends (path_nm y),'fc2d_ltl.ijs'
for_k. i.x do.
  whilst. perq +. (mina>a) +.mins>s do.
    p=:m,+/\?4#1+*:m
NB.    b=:p LtL^:(i.128) ?.128 128$2  NB. standard
NB.    b=:p LtL^:(i.256) ?.256 256$2  NB. wide
    b=:p LtL^:(i.256) 0.7<?.256 256$0  NB. biased
    perq=:({: e. }:) _25{.b
    z=:m Ie2 64}.b
    s=.stddev z
    a=.avg z
  end.
  S=:S,s
  A=:A,a
  fn=.y,(' _' stringreplace ": p),'_s',(":<.0.5+100*s),'.png'
  (4 spix 0 2 1{"1]255*0|:_3{. b) write_image fn
end.
3 mk_html_gallery 384 384 images_in path_nm y
)

randomize''

50 (2) Fc2d_ltl 0.15 2 'c:\temp\fc2d_ltl_r2\r'
```

Figure 90 shows the 2 1 4 4 7 automata on a standard array. Note that the `Fc2d_ltl` conjunction does not make a dead cell black; it uses all three of the last three iterates with green being the final state, blue the previous and red the one before that. Note the self-organizing regions which often seem quite curve-like. Animations of the evolution of these automata are available at [17]. In some cases the animations show interesting evolution, in other cases they show harsh, rapidly changing frames, but in both cases we can get some sense of how transient

behavior played a role. In Figure 91 we see the standard result of LtL 2 4 4 4 7. Note there is a mix of stable black and white regions with substantial regions where changes occur.

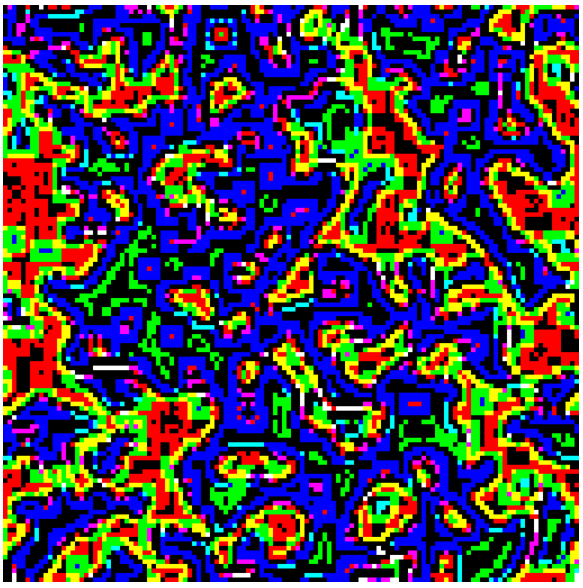


Figure 90. Larger than Life 2-D: 2 1 4 4 7

Figure 91. Larger than Life 2-D: 2 4 4 4 7

If a region oscillates on and off, it will be alternately blue and yellow. Figure 92 shows the LtL 2 0 1 5 7 automata (biased) where that behavior dominates. Figure 93 shows the LtL 2 4 4 8 10 automaton (biased) where most cells are dead. However, patches seem to continue to evolve and some local symmetric patches develop.

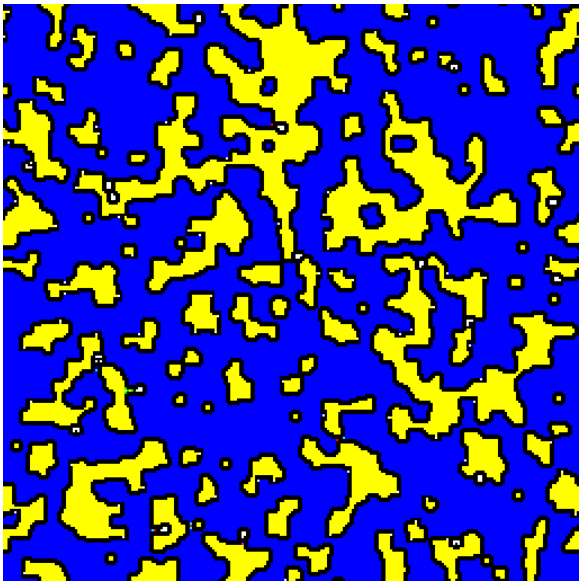


Figure 92. Larger than Life 2-D: 2 0 1 5 7

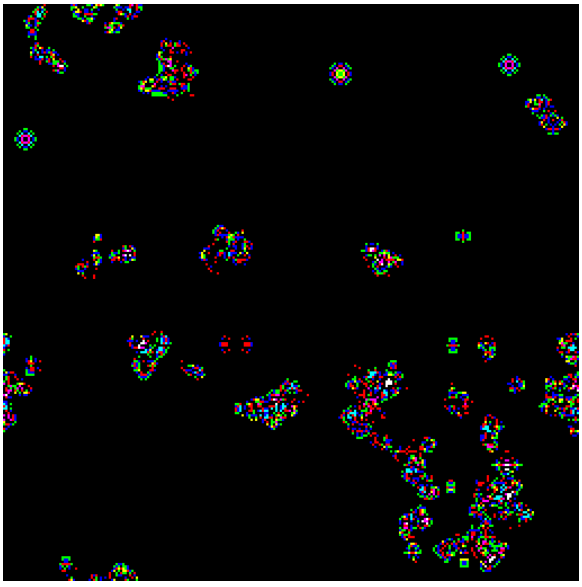


Figure 93. Larger than Life 2-D: 2 4 4 8 10

One of the surprises in [15] was the occurrence of a pattern with global symmetry. Figure 94 and 95 show a standard and a wide version of this. These occur for many radii but actually form a significant, but small, proportion of the examples found when the radius is 3. Note one has black and white temporarily stable regions while the other does not. The reason for the symmetry is that in a small number of iterates, only a few cells survive and they happen to have symmetry and retain it as they grow. We count the number alive cells in each time step for the example in Figure 94 and display the first and last 10. So the entire configuration evolved from a single cell after one iteration.

```
   b=:3 1 10 10 13 LtL^:(i.128) ?. 128 128$2

   10{.+/@,"2 b
8139 1 48 80 140 328 205 924 244 1120

   _10{.+/@,"2 b
4892 4028 5944 3684 6476 3188 5512 3536 4884 3972
```

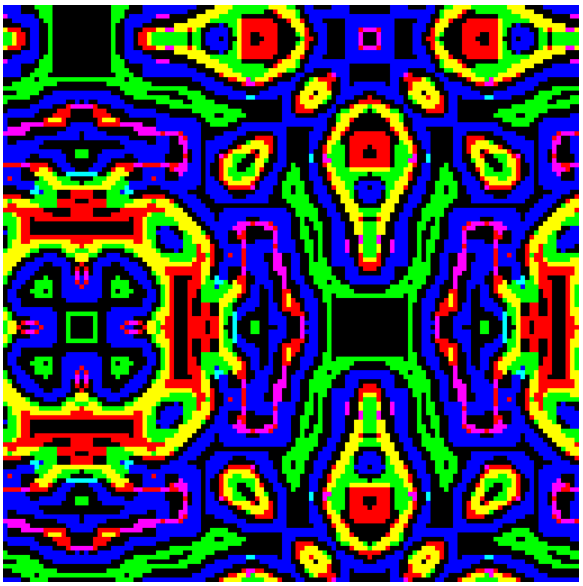


Figure 94. Larger than Life 2-D: 3 1 10 10 13

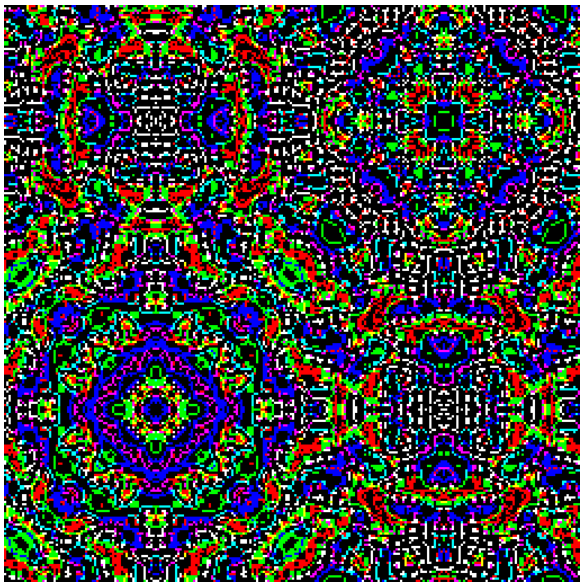


Figure 95. Larger than Life 2-D: 3 5 7 7 14

Figure 96 shows the LtL 3 8 8 9 13 biased automata with large black regions and very granular regions with alive cells. Figure 97 shows the LtL 4 1 12 14 18 biased automata where clear curve-like regions organize and blue and yellow may loosely alternate.

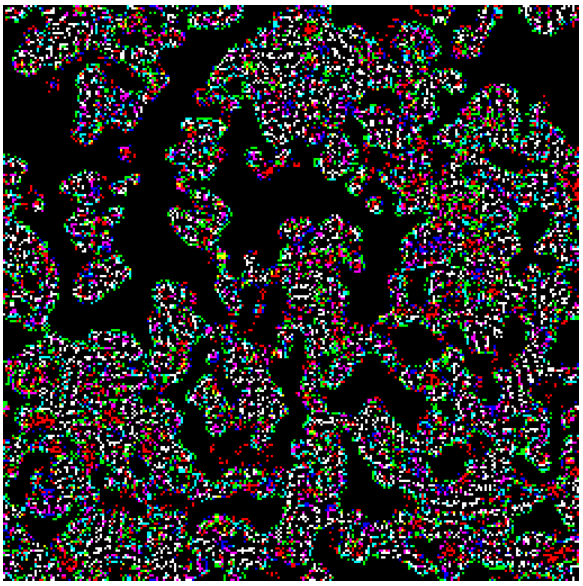


Figure 96. Larger than Life 2-D: 3 8 8 9 13

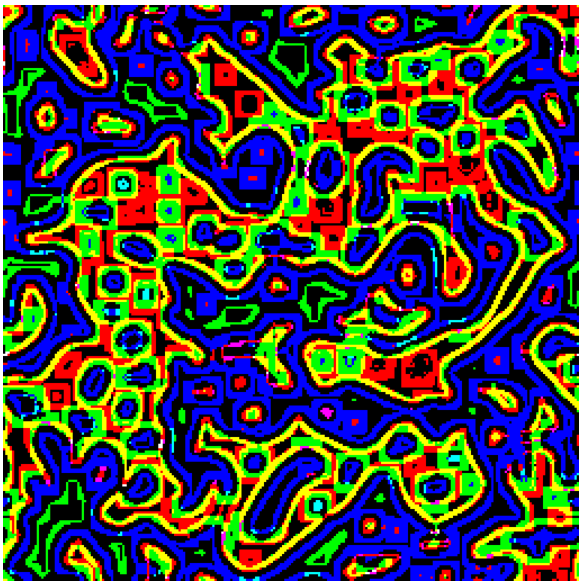


Figure 97. LtL 2-D: 4 1 12 14 18

Figure 98 shows a wide automata that mostly stabilizes into white patches on a black background. Figure 99 shows a wide automata that has self-organized patches with a remarkable amount of cyan, yellow and magenta occurring. Figure 100 shows a biased example with some alternating regions and stable regions where the white on black appears both in solid clumps but also in dusty regions. Figure 101 shows a biased automata where both thin bands and clumps appear. The orientation seems related to the rectangular lattice.

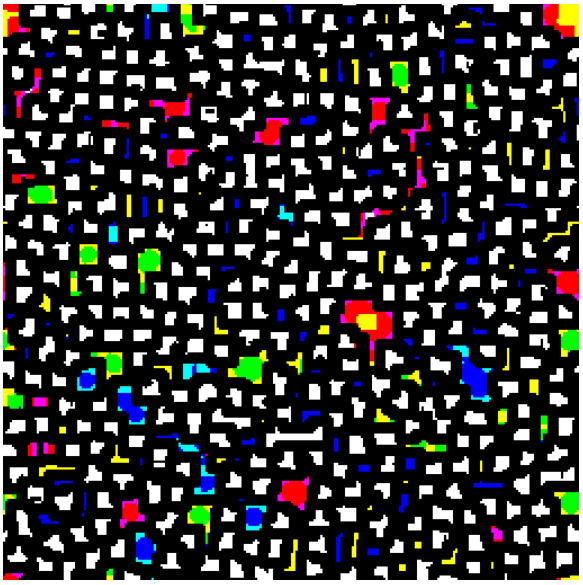


Figure 98. LtL 2-D: 5 0 1 26 50

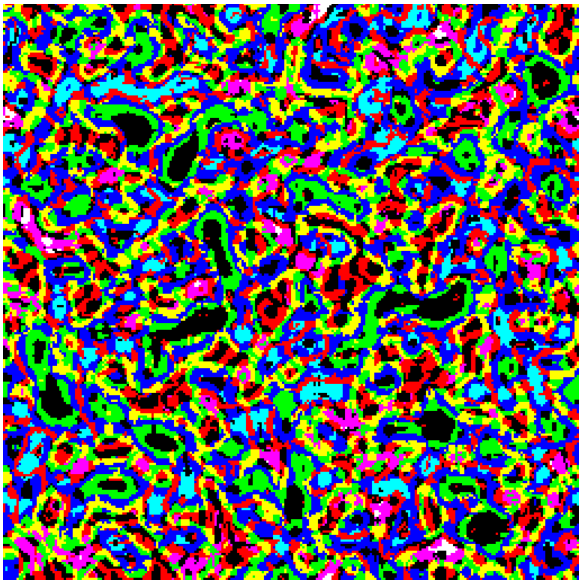


Figure 99. LtL 2-D: 5 25 46 69 89

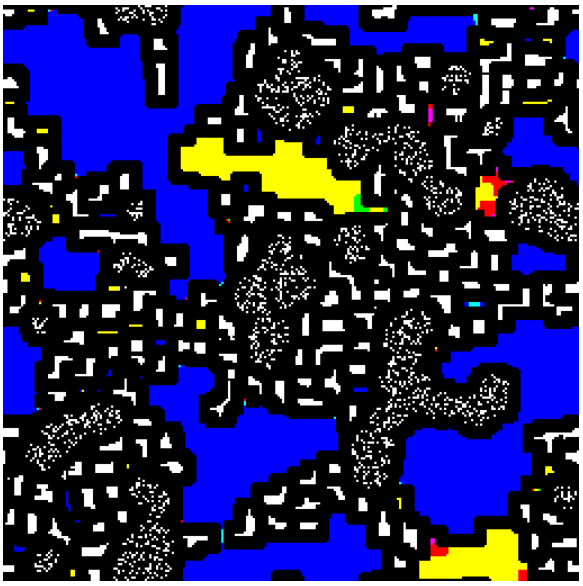


Figure 100. LtL 2-D: 6 0 1 17 47

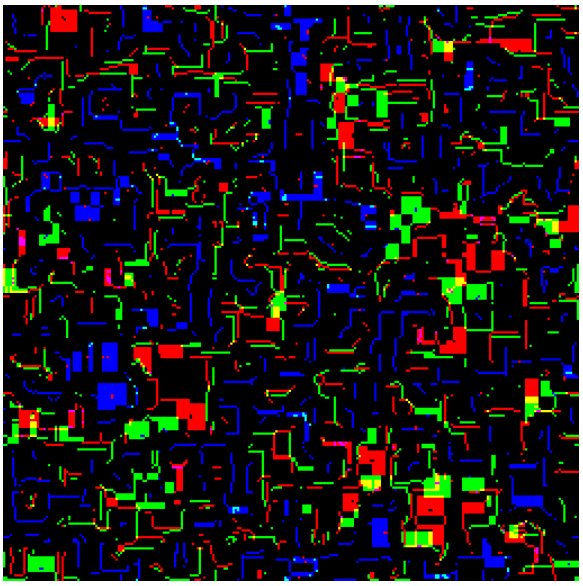


Figure 101. LtL 2-D: 6 2 2 33 35

Figure 102 shows wide LtL 8 3 39 59 120 with self-organized temporarily stable black and white regions and other waves. Figure 103 shows an automata (biased) with mostly thin alive regions that are much less oriented with the lattice. Like the symmetric examples, the automata in Figure 104 nearly dies out before growing into this form. I couldn't resist shifting this image a bit to the right to enhance its appearance as an automaton that reminds of "The Scream".

Figure 102. LtL 2-D: 8 3 39 59 120

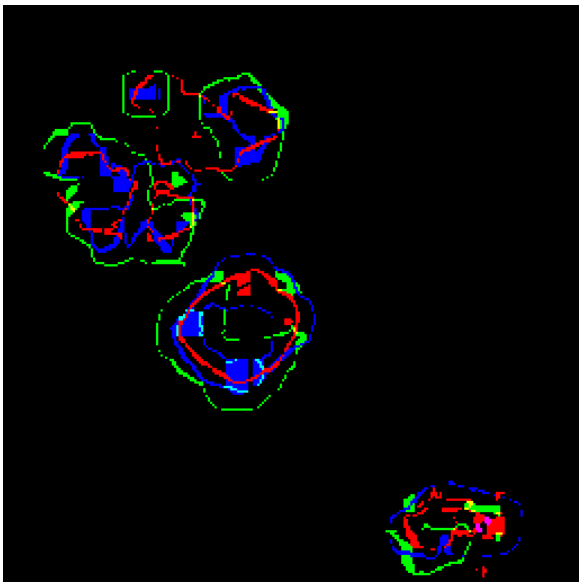


Figure 103. LtL 2-D: 10 31 35 129 141

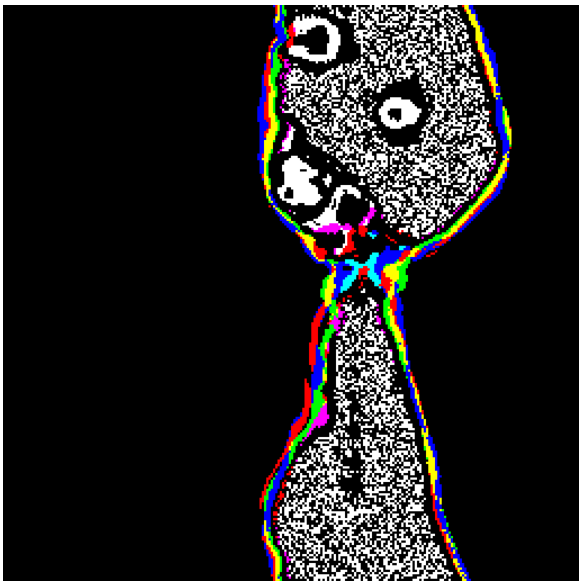


Figure 104. LtL 2-D: 15 194 237 285 491

We saved our favorite 2D LtL automata found by our search until last. The LtL 2 4 4 4 4 (wide) automata shows a mix a behavior on a black background as shown in Figure 105. Again, the cells almost all die out and then v and u shaped configurations move across the screen and eventually generate the well populated image seen in Figure 105. Figures 106-108 show the configurations after 6, 63, and 127 iterations, respectively.

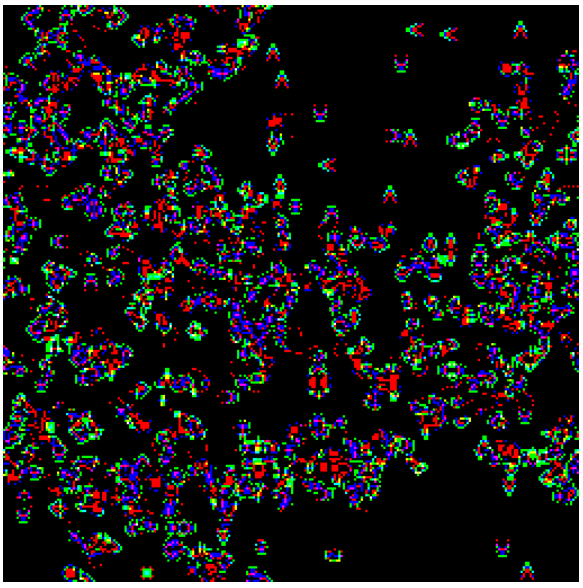


Figure 105. LtL 2-D: 2 4 4 4 4

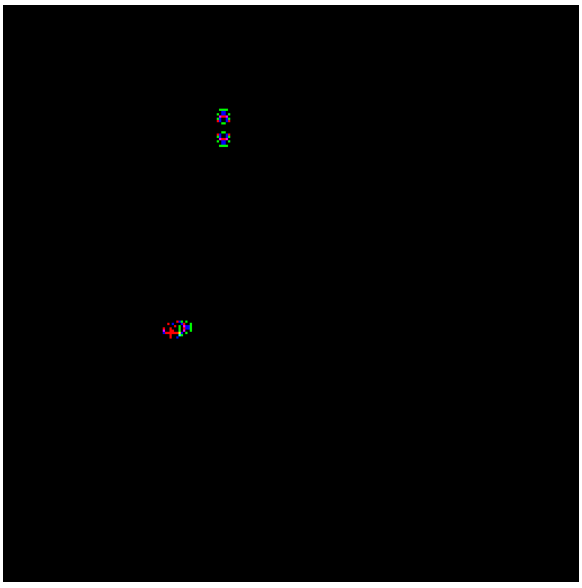


Figure 106. LtL 2 4 4 4 4 after 6 steps

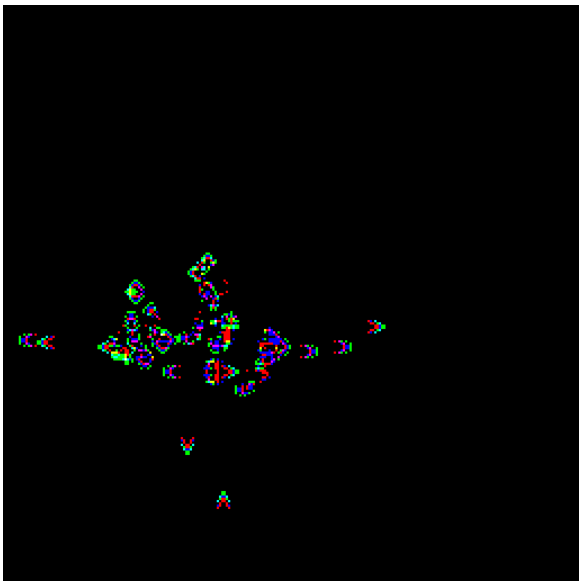


Figure 107. LtL 2 4 4 4 4 after 31 steps

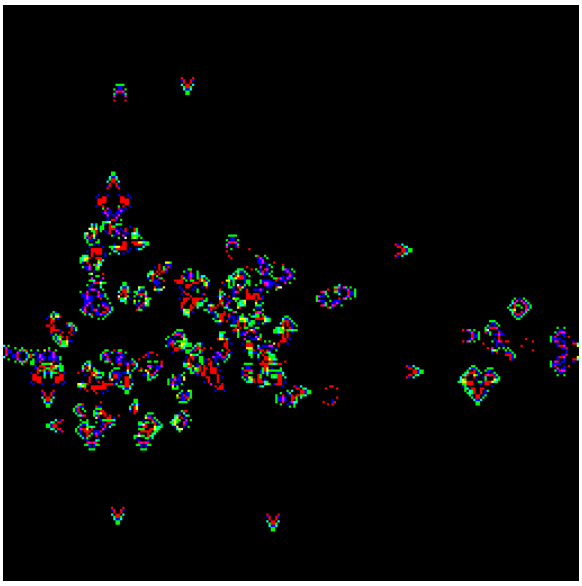


Figure 108. LtL 2 4 4 4 4 after 63 steps

7. Concluding Remarks

We have seen that it is easy to compute input entropy for one and two dimensional automata. The J adverbs to accomplish that are quite concise. We have seen that input entropy can be used to guide searches to find automata that exhibit complex behavior. J utilities that illustrate that included one and two dimensional general automata and one and two dimensional Larger than Life automata. We illustrated our results but encourage readers to find their own automata exhibiting complexity. Once an interesting automata has been found, one need not stop there. One can explore alternate visualization schemes, consider initial conditions other than the default, and explore nearby automata by making small changes to the parameters.

References

[1] A. Adamatzky and G. J. Martinez, editors, *Designing Beauty: the Art of Cellular Automata*, Springer, 2016.

[2] E. R. Berlekamp, J. H. Conway and R. K. Guy, *Winning ways for your mathematical plays, vol 2*, Academic Press, 1982.

[3] K. M. Evans, PhD thesis, Larger than Life: it's so nonlinear, http://www.csun.edu/~kme52026/thesis.html

[4] K. M. Evans, Web page http://www.csun.edu/~kme52026/

[5] K. M. Evans, Larger than life: threshold-range scaling of Life's coherent structures, *Physica D; Nonlinear Phenomena*, **183** (2003) 45-67.

[6] K. M. Evans, Larger than Life's invariant measures, *Electronic Notes in Theoretical Computer Science*, **252** (2009) 55-75. Gallery of Images: http://www.csun.edu/~kme52026/invariant.html

[7] K. M. Evans, Larger than Life, pp 27-34, in [1].

[8] M. Gardner, The fantastic combinations of John Conway's new solitaire “game of life", *Scientific American*, **223** 4 (1970) 120-123.

[9] M. Gardner, On cellular automata, self-replication, the Garden of Eden and the “game life", *Scientific American*, **224** 4 (1971) 112-117.

[10] D. Griffeath, Primordial Soup Kitchen, http://psoup.math.wisc.edu/welcome.html.

[11] D. Griffeath, Self-organization of Random Cellular Automata: Four snapshots. In: *Probability and Phase Transitions*, 49-67, Springer, 1994.

[12] D. Griffeath, Self-organizing two-dimensional cellular automata, pp 1-12, in [1].

[13] A. Ilachinski, *Cellular Automata: a Discrete Universe*, World Scientific, 2001.

[14] C. A. Reiter, *Fractals, Visualization and J, 4th edition*, lulu.com, Part 1: 2016, Part 2: 2017.

[15] C. Reiter, Larger than life automata, *Vector*, in press, http://archive.vector.org.uk/art10501760

[16] C. Reiter, Auxiliary Materials for Larger than Life Automata, http://webbox.lafayette.edu/~reiterc/j/vector/ltl/index.html

[17] C. Reiter, Auxiliary Materials for `$:` https://webbox.lafayette.edu/~reiterc/j/JoJ/fc_ie/index.html

[18] J. L. Schiff, *Cellular Automata: a Discrete View of the World*, Wiley, Inc., 2008.

[19] S. Wolfram, *A New Kind of Science*, Wolfram Media, 2002.

[20] Andrew Wuensche, *Exploring Discrete Dynamics*, *Second Edition*, Luniver Press, 2016.

Calculating Maximum Drawdown

Devon McCormick, CFA

First, a definition [from http://www.investopedia.com/terms/m/maximum-drawdown-mdd.asp]:

DEFINITION of 'Maximum Drawdown (MDD)'

The maximum loss from a peak to a trough of a portfolio, before a new peak is attained. Maximum Drawdown (MDD) is an indicator of downside risk over a specified time period. It can be used both as a stand-alone measure or as an input into other metrics such as "Return over Maximum Drawdown" and Calmar Ratio. Maximum Drawdown is expressed in percentage terms and computed as:

(Trough Value – Peak Value) ÷ Peak Value

BREAKING DOWN 'Maximum Drawdown (MDD)'

Consider an example to understand the concept of maximum drawdown.

Assume an investment portfolio has an initial value of $500,000. The portfolio increases to $750,000 over a period of time, before plunging to $400,000 in a ferocious bear market. It then rebounds to $600,000, before dropping again to $350,000. Subsequently, it more than doubles to $800,000. What is the maximum drawdown?

The maximum drawdown in this case is = ($350,000 – 750,000) / $750,000 = –53.33%

Note the following points:

- The initial peak of $750,000 is used in the MDD calculation. The interim peak of $600,000 is not used, since it does not represent a new high. The new peak of $800,000 is also not used since the original drawdown began from the $750,000 peak.
- The MDD calculation takes into consideration the lowest portfolio value ($350,000 in this case) before a new peak is made, and not just the first drop to $400,000.

MDD should be used in the right perspective to derive the maximum benefit from it. In this regard, particular attention should be paid to the time period being considered. For instance, a hypothetical long-only U.S. fund Gamma has been in existence since 2000, and had a maximum drawdown of -30% in the period ending 2010. While this may seem like a huge loss, note that the S&P 500 had plunged more than 55% from its peak in October 2007 to its trough in March 2009. While other metrics would need to be considered to assess Gamma fund's overall performance, from the viewpoint of MDD, it has outperformed its benchmark by a huge margin.

How might we calculate this in J in an array-oriented fashion? Let's start in the middle: assume we have the peak and trough values:

```
   ((>./-<./)%>./) 2108.63 1867.61 NB. Max drawdown

0.114302
```

That's the easy part. The harder part is finding the correct peak and trough for a given set of numbers.

Taking the numbers from the example above:

```
   vals=. 10000* 50 75 40 60 35 80

   (<./,>./) vals

350000 800000
```

We know these aren't the right numbers. We need to find the peak at each point in the series. This will get us close:

```
   >./\vals

500000 750000 750000 750000 750000 800000
```

Now we can find the location of each new peak:

```
   ]whNewPeak=. 2</\>./\vals

1 0 0 0 1

   #whNewPeak

5

   #vals

6
```

There's a problem: due to the nature of comparing pairs of items, our result will always be one shorter than our starting vector but we'd like them to line up so we have to make a decision. Fortunately, the choice seems clear: we'll assume that the first value counts as a peak under the reasoning that it is the highest value "so far". So,

```
   #whNewPeak=. 1,whNewPeak

6
```

It seems natural, with an eye to the next step of finding the minimum in each intra-peak section, to use `whNewPeak` as a partition vector:

```
   whNewPeak <;.1 vals
+------+---------------------------+------+
|500000|750000 400000 600000 350000|800000|
+------+---------------------------+------+
```

This allows us to find the minimum of each interval along with its corresponding peak:

```
   <./&>whNewPeak<;.1 vals
500000 350000 800000
   whNewPeak#vals
500000 750000 800000
```

So, finding the maximum drawdown should be relatively straightforward: we simply find the maximum difference:

```
   >./(whNewPeak#vals) - <./&>whNewPeak<;.1 vals
400000
```

However, there is a complication: we have to associate this number with its corresponding peak. Also, upon reflection, it would be useful to keep track of the date associated with each of these values so we can show our work.

```
maxDrawdown=: 3 : 0
   whNewPeak=. 1,2</\>./\y
   md=. (whNewPeak#y) - <./&>whNewPeak<;.1 ] y
   nfp=. 0={:whNewPeak            NB. Not final peak (no peak at end)
   md0=. (-nfp)}.md%whNewPeak#y   NB. Drop last if didn't end on new peak
   wsp=. md0 i. >./md0            NB. Where's starting peak of max drawdown?
   spix=. (wsp+0 1){I. whNewPeak  NB. Start, end index of max drawdown
   span=. y{~(<./ + [: i. [: >: [: | -/) spix
   wmin=. (<./spix)+span i. <./span NB. Where minimum was in span
   md=. (>./md0);spix;wmin
NB.EG 'md whsp whmin'=. maxDrawdown 500 750 400 600 350 800
NB.EG 'md whsp whmin'=. maxDrawdown vals=. 100 150 90 120 80 200
)
```

```
   vals{~/:~whsp, whmin

750000 350000 800000

   ]'md whsp whmin'=. maxDrawdown 500 750 400 600 350 800

+--------+---+-+

|0.533333|1 5|4|

+--------+---+-+

   vals{~/:~whsp, whmin

750000 350000 800000
```

A fuller example of usage:

```
'tit1 sp500ix'=: split <;._1&>TAB,&.><;._2 ] LF (] , [ #~ [ ~: [: {: ]) CR-
.~0 : 0

Date  S&P 500 Index - Index Levels - Index Value - USD

01/06/2015  2002.613587

01/07/2015  2025.90105

01/08/2015  2062.143554

01/09/2015  2044.8099

…

01/04/2016  2012.659371

01/05/2016  2016.714426

01/06/2016  1990.262292

)


   tit1

+----+-----------------------------------------------+

|Date|S&P 500 Index - Index Levels - Index Value - USD|

+----+-----------------------------------------------+
```

```
   'dts vals'=. <"1 |:sp500ix

   vals=. ".&>vals

   ]'md whsp whmin'=. maxDrawdown vals

+---------+-----+--+

|0.0364402|37 75|44|

+---------+-----+--+

   sp500ix{~/:whsp,whmin

+----------+----------+

|01/06/2015|2002.613587|

+----------+----------+

|01/08/2015|2062.143554|

+----------+----------+

|01/07/2015|2025.90105 |

+----------+----------+
```

Graphing Drawdown

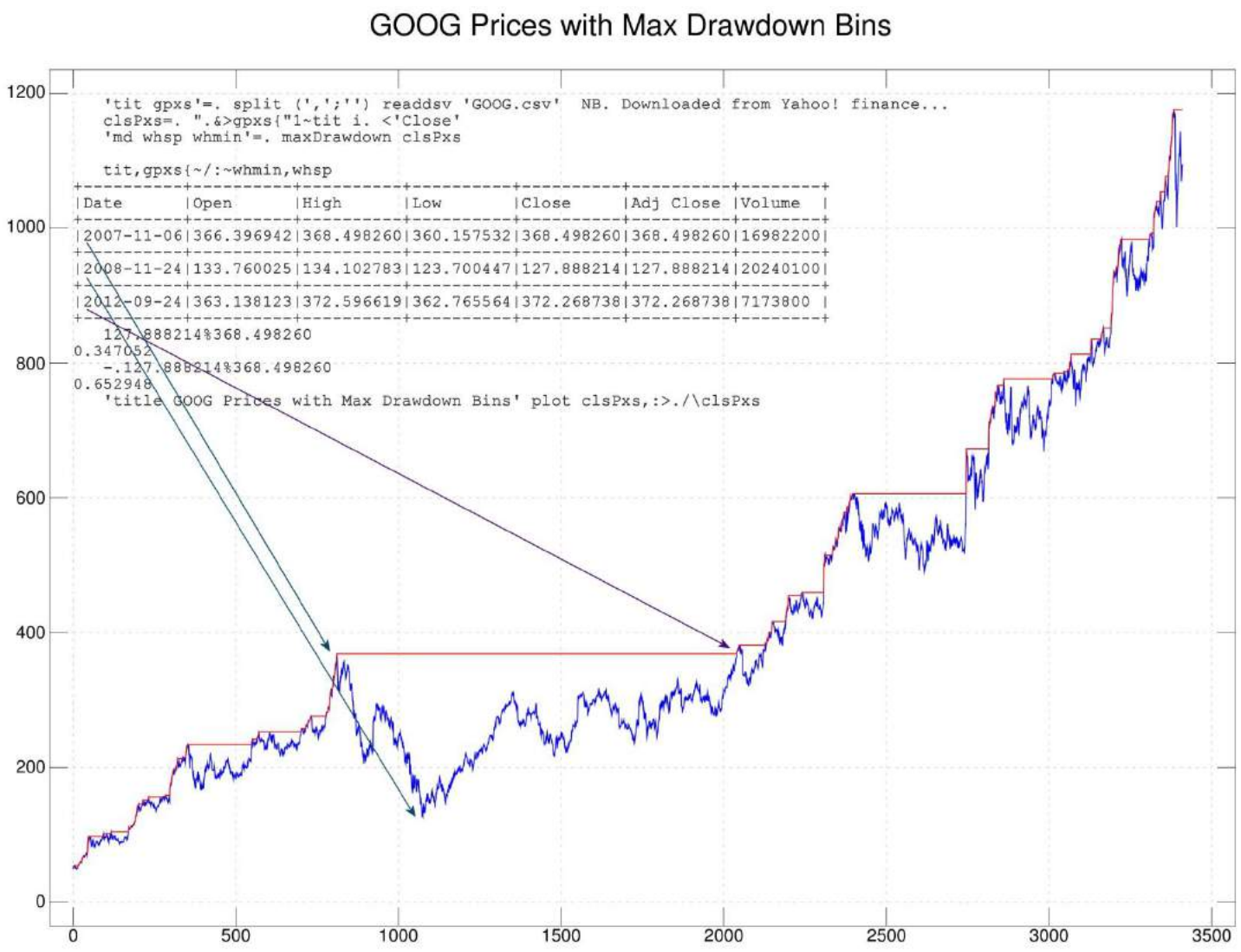
First, let's get some real-life data.

```
   load 'dsv'                                NB. Delimiter-Separated Values

   $googPxs=. (',';'') readdsv 'GOOG.csv'  NB. Downloaded from Yahoo
Finance...

3411 7

   3{.googPxs

+----------+---------+---------+---------+---------+---------+--------+

|Date      |Open     |High     |Low      |Close    |Adj Close|Volume  |

+----------+---------+---------+---------+---------+---------+--------+

|2004-08-19|49.676899|51.693783|47.669952|49.845802|49.845802|44994500|

+----------+---------+---------+---------+---------+---------+--------+

|2004-08-20|50.178635|54.187561|49.925285|53.805050|53.805050|23005800|

+----------+---------+---------+---------+---------+---------+--------+
```

```
   'tit gpxs'=. split googPxs

   tit

+----+----+----+---+-----+---------+------+

|Date|Open|High|Low|Close|Adj Close|Volume|

+----+----+----+---+-----+---------+------+

   clsPxs=. ".&>gpxs{"1~tit i. <'Close'

   $clsPxs

3410

   clsPxs

49.8458 53.8051 54.3465 52.0962 52.6575 53.6063 52.732 50.6754 50.8542
49.8011 50.427 49.6819 50.4618 50.8195 50.8244 52.3247 53.4027 55.3848
55.6381 56.6168 58.3654 59.2943 58.5393 58.8075 60.0196 59.5278 58.7479
63.0201 65.1165 64.3813 65.8616 67.0936 68...

   load 'plot'

   'tit gpxs'=. split (',';'') readdsv 'GOOG.csv'  NB. Downloaded from Yahoo!
finance...

   clsPxs=. ".&>gpxs{"1~tit i. <'Close'

   'md whsp whmin'=. maxDrawdown clsPxs



   md;whsp;whmin

+--------+--------+----+

|0.652948|810 2040|1075|

+--------+--------+----+



   tit,gpxs{~/:~whmin,whsp

+----------+----------+----------+----------+----------+----------+--------+

|Date      |Open      |High      |Low       |Close     |Adj Close |Volume  |

+----------+----------+----------+----------+----------+----------+--------+

|2007-11-06|366.396942|368.498260|360.157532|368.498260|368.498260|16982200|

+----------+----------+----------+----------+----------+----------+--------+
```

```
|2008-11-24|133.760025|134.102783|123.700447|127.888214|127.888214|20240100|

+----------+----------+----------+----------+----------+----------+--------+

|2012-09-24|363.138123|372.596619|362.765564|372.268738|372.268738|7173800 |

+----------+----------+----------+----------+----------+----------+--------+

   127.888214%368.498260     NB. Lowest in drawdown as portion of starting
point.

0.347052

   -.127.888214%368.498260   NB. Drawdown is how far down this took us at
that point.

0.652948

   'pensize 2' plot clsPxs,:>./\clsPxs
```

After drawing and writing on the result of this plot, we get the following.


GOOG Prices with Max Drawdown Bins
'tit gpxs'=. split (',';'') readdsv 'GOOG.csv' NB. Downloaded from Yahoo! finance...
clsPxs=. ".&>gpxs{"1~tit i. <'Close'
'md whsp whmin'=. maxDrawdown clsPxs
tit,gpxs{~/:~whmin,whsp
Date	Open	High	Low	Close	Adj Close	Volume
2007-11-06	366.396942	368.498260	360.157532	368.498260	368.498260	16982200
2008-11-24	133.760025	134.102783	123.700447	127.888214	127.888214	20240100
2012-09-24	363.138123	372.596619	362.765564	372.268738	372.268738	7173800
127.888214%368.498260
0.347052
-.127.888214%368.498260
0.652948
'title GOOG Prices with Max Drawdown Bins' plot clsPxs,:>./\clsPxs
0
200
400
600
800
1000
1200
0
500
1000
1500
2000
2500
3000
3500

A NOTE ON TRUTH-FUNCTIONAL SELF-DUALITY

Schema S_1 is a dual of schema S_0 iff: when the letters of S_0 are given interpretation *I*, yielding value *V*, giving interpretation *-.I* to the letters of S_1 will yield *-.V*. 'a+.b', for example, is dual to 'a*.b'. In APL dialects the most natural representation of a truth-table for a schema in N letters will be an array of shape N#2; so talk of interpretations can be elided into talk of locations: in the truth-tables for the duals 'a<:b', 'a<b'

```
┌─────┬─────┐
│1 1 │0 1 │
│0 1 │0 0 │
└─────┴─────┘
```

the single 0 for 'a<:b' is at 1 0, the single 1 for 'a<b' is at 0 1.

Some schemata (and tables) are self-duals, for example:

```
   ]T=: truarr '(a*.b)+.(a*.c)+.b*.c'
0 0
0 1

0 1
1 1
```

```
Gloc=: $ #: I.@,

Gloc L:0 (1=T);0=T
```

```
┌─────────┬─────────┐
│0 1 1│0 0 0│
│1 0 1│0 0 1│
│1 1 0│0 1 0│
│1 1 1│1 0 0│
└─────────┴─────────┘
```

The indices for the 0s in T are the negations of the indices for the 1s.

It is evident that iff T is self-dual then {.T will be dual to {:T (on any axis). Any string of indices I will select either {.T or {:T, -.I the other. But }.I and }.-.I select opposite values in {.T and {:T; so I and -.I select opposite values in T.

The number of self-duals for tables of N axes, for a few N:

For 0 axes, there are no self-duals.

The 2 tables of 1 axis, corresponding to the schemata 'a' and '-.a', are both self-duals.

For 2 axes, there can be no self-duals. For A to be self-dual, {.A and {:A must be duals; if A is of rank 2, {.A and {:A are of rank 1, and so, if dual, are identical. But then A is not of rank 2.

For 3 axes, there are 10 self-duals (each of the 10 rank 2 functions paired with its dual).

Nollaig MacKenzie