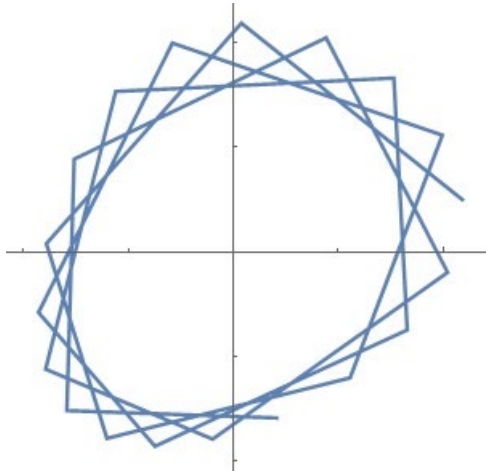
Cellular automata

theory

Notes for CAS

Feijs



Cellular automata:

- Model of interacting agents in fixed grid
- Very simple rules, yet complex behaviour
- Fundamental studies about computation and complexity
- Also useful for simulations in physics and social computation

Good overview: http://slideplayer.com/slide/5674720/

Mathworld: A cellular automaton is a collection of "colored" cells on a grid of specified shape that evolves through a number of discrete time steps according to a set of rules based on the states of neighboring cells.

The rules are then applied iteratively for as many time steps as desired. von Neumann was one of the first people to consider such a model, and incorporated a cellular model into his "universal constructor." Cellular automata were studied in the early 1950s as a possible model for biological systems (Wolfram 2002, p. 48).

Comprehensive studies of cellular automata have been performed by S. Wolfram starting in the 1980s, and Wolfram's fundamental research in the field culminated in the publication of his book *A New Kind of Science* (Wolfram 2002) in which Wolfram presents a gigantic collection of results concerning automata, among which are a number of groundbreaking new discoveries.

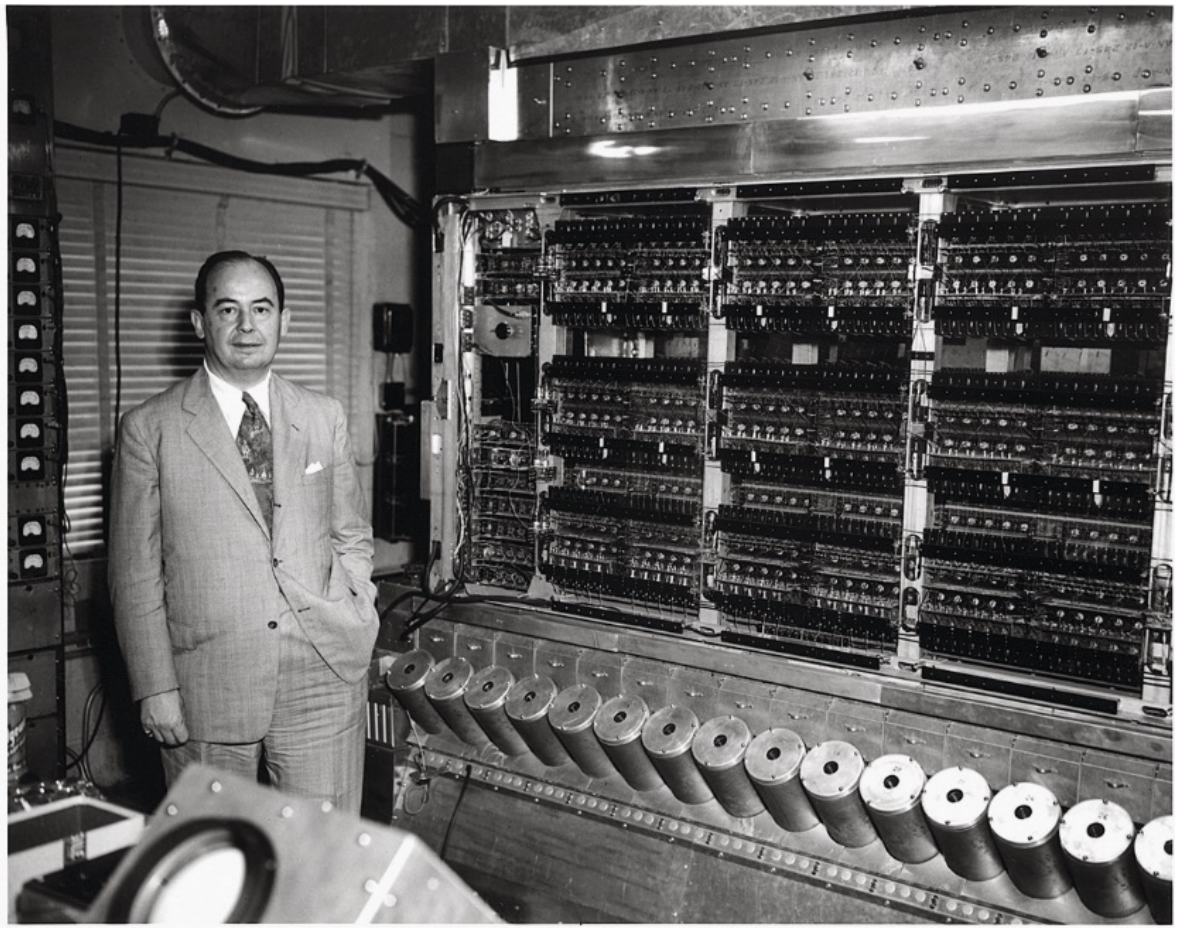
http://www.idemployee.id.tue.nl/g.w.m.rauterberg/lecturenotes/DDM110%20CAS/default.html

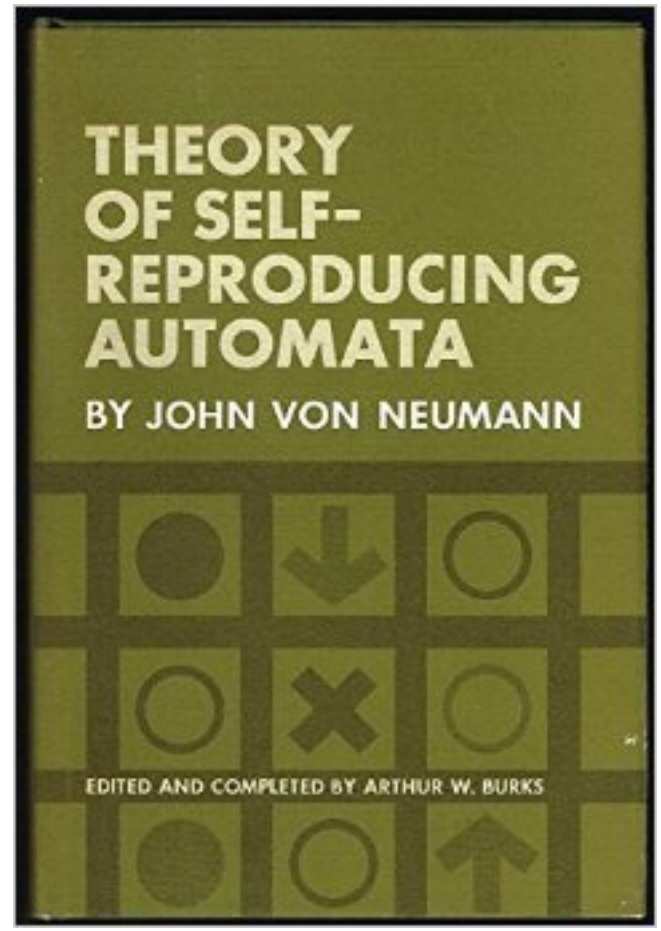
Cellular automata:

- every cell (agent) has finite-state machine behaviour
- neighbour states + own state determine new state
- all cells are updated simultaneously
- the rules are the same for all cells

History:

- John Von Neumann (1948, self-replication)
- John Conway (1970, universality)
- Steven Wolfram (2002, complexity)

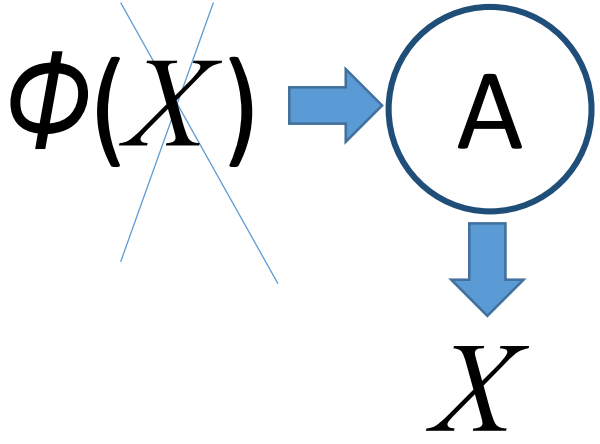


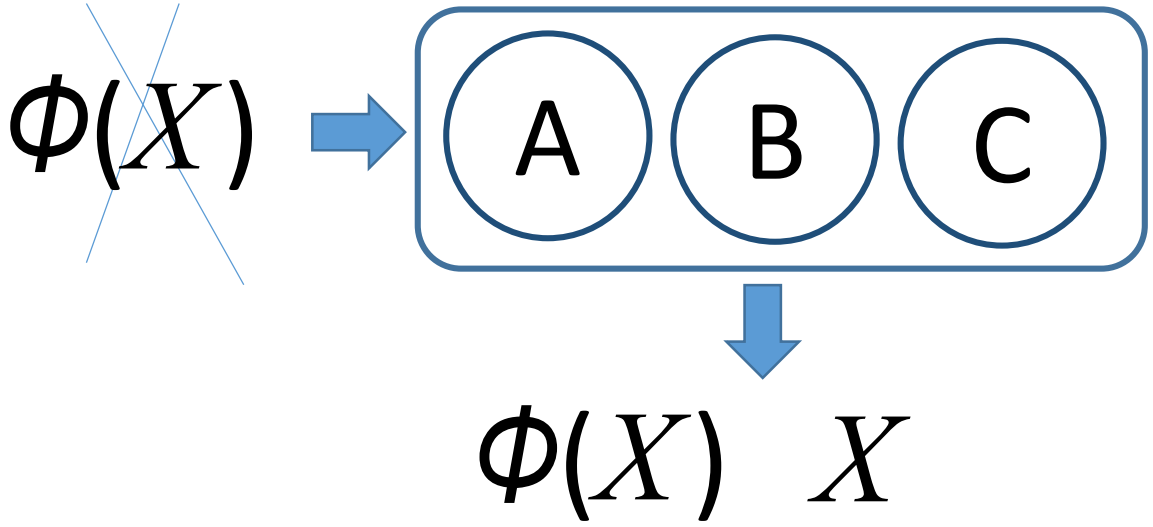

THEORY
OF SELF-
REPRODUCING
AUTOMATA
BY JOHN VON NEUMANN
EDITED AND COMPLETED BY ARTHUR W. BURKS

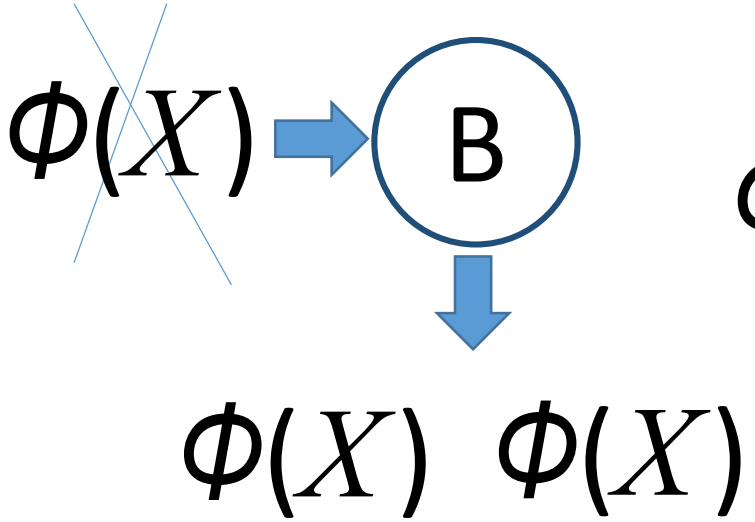
$\Phi(X)$ is a linear description of X

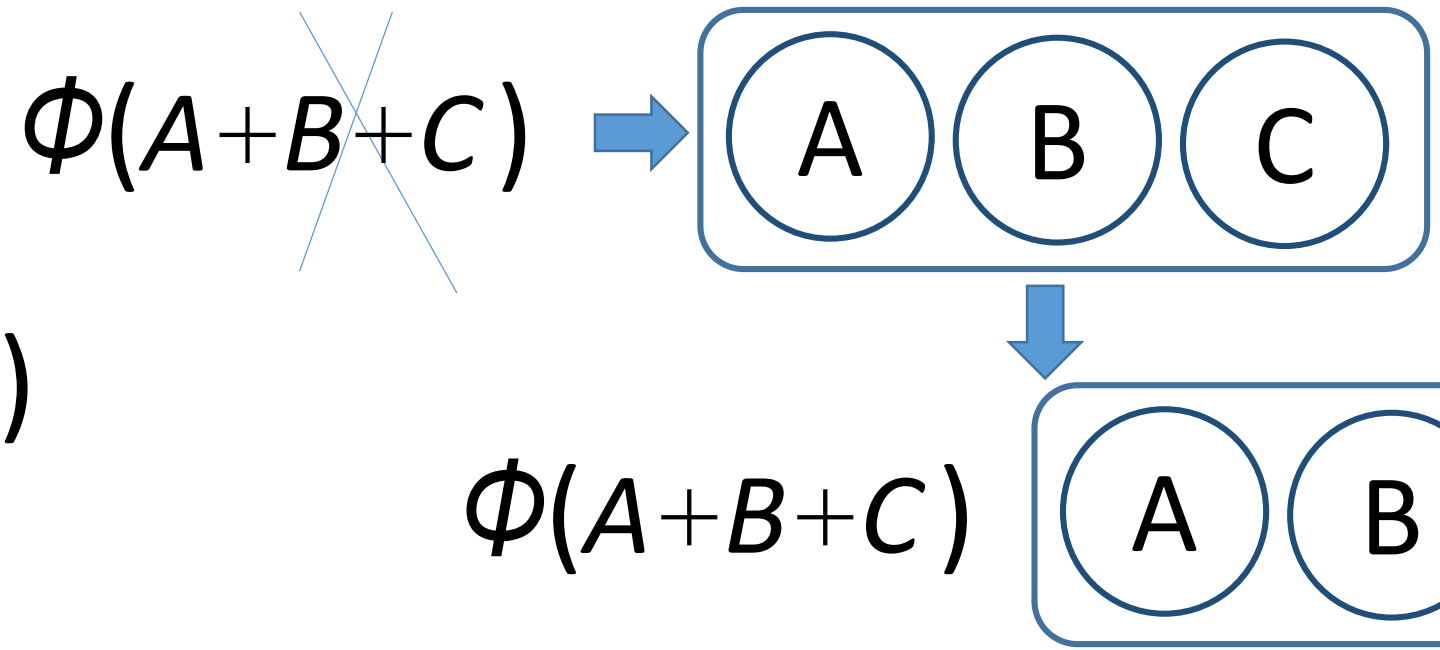
The general constructive automaton A produces only X when a complete description of X is furnished it, and on any reasonable view of what constitutes complexity, this description of X is as complex as X itself. The general copying automaton B produces two copies of $\phi(X)$, but the juxtaposition of two copies of the same thing is in no sense of higher order than the thing itself. Furthermore, the extra unit B is required for this copying.

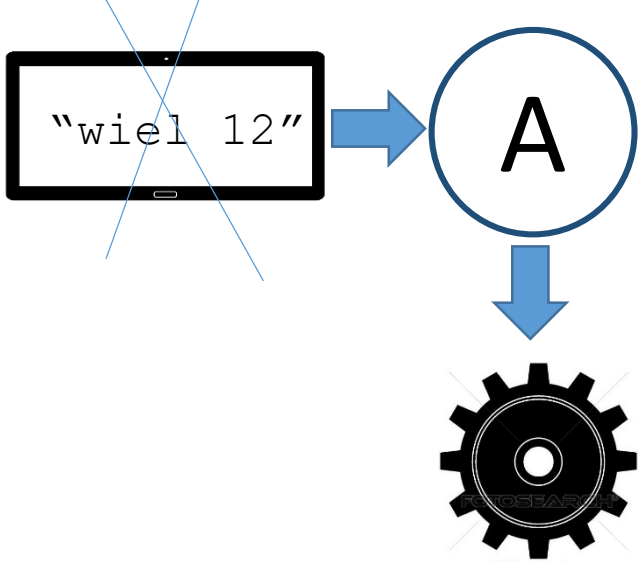
Now we can do the following thing. We can add a certain amount of control equipment C to the automaton $A + B$. The automaton C dominates both A and B, actuating them alternately according to the following pattern. The control C will first cause B to make two copies of $\phi(X)$. The control C will next cause A to construct X at the price of destroying one copy of $\phi(X)$. Finally, the control C will tie X and the remaining copy of $\phi(X)$ together and cut them loose from the complex $(A + B + C)$. At the end the entity $X + \phi(X)$ has been produced.

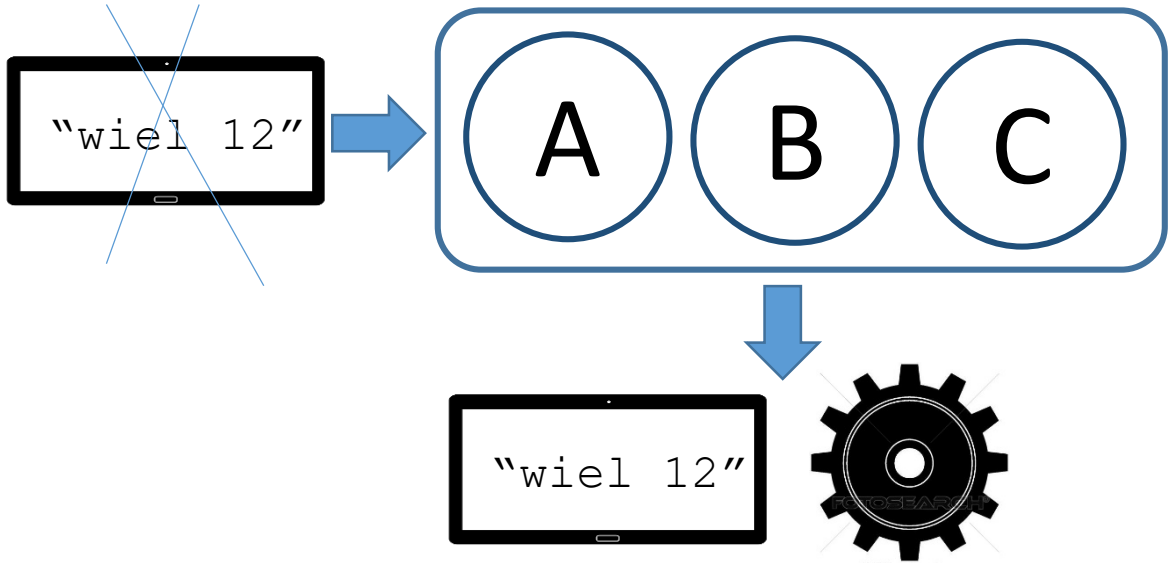

Φ(X)
A
X

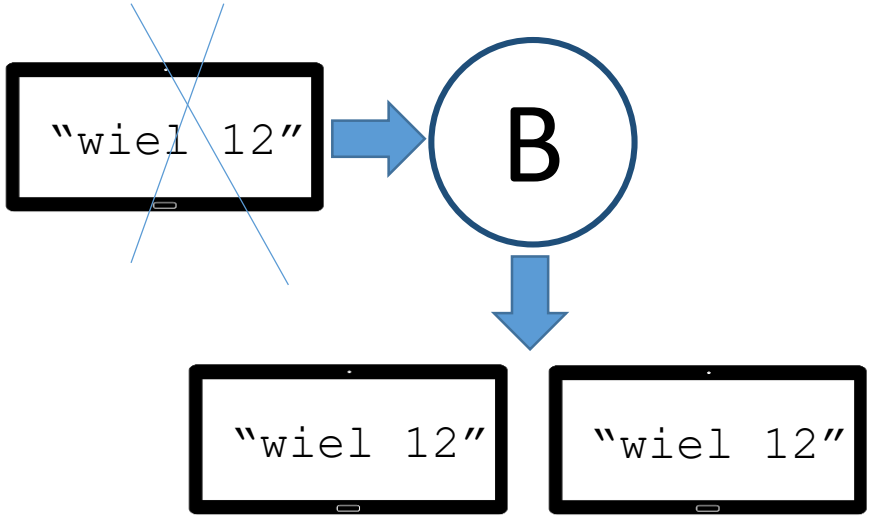

Φ(X)
A
B
C
Φ(X) X

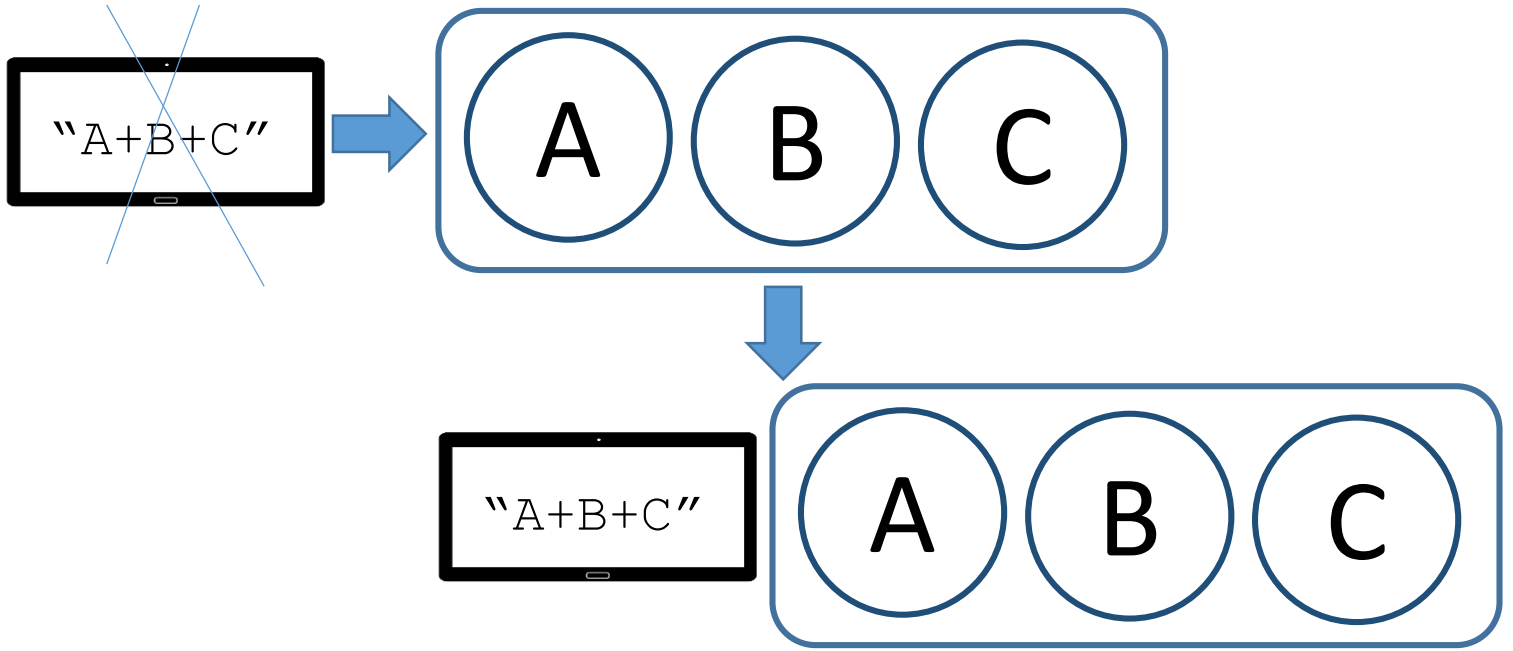

Φ(X)
B
Φ(X) Φ(X)


Φ(A+B+C)
A
B
C
Φ(A+B+C)
A
B
C


“wiel 12”
A


“wiel 12”
A
B
C
“wiel 12”


“wiel 12”
B
“wiel 12”
“wiel 12”


“A+B+C”
A
B
C
“A+B+C”
A
B
C

completely unnecessary here. You could use more complicated looped chains, which would be perfectly good carriers for a code, but it would not be a linear code. There is reason to suspect that our predilection for linear codes, which have a simple, almost temporal sequence, is chiefly a literary habit, corresponding to our not particularly high level of combinatorial cleverness, and that a very efficient language would probably depart from linearity.[3]

There is no great difficulty in giving a complete axiomatic account of how to describe any conceivable automaton in a binary code. Any such description can then be represented by a chain of rigid elements like that of Figure 2. Given any automaton X, let $\phi(X)$ designate the chain which represents X. Once you have done this, you can design a universal machine tool A which, when furnished with such a chain $\phi(X)$, will take it and gradually consume it, at the same time building up the automaton X from the parts floating around freely in the surrounding milieu. All this design is laborious, but it is not difficult in principle, for it's a succession of steps in formal logics. It is not qualitatively different from the type of argumentation with which Turing constructed his universal automaton.

Another thing which one needs is this. I stated earlier that it might be quite complicated to construct a machine which will copy an automaton that is given it, and that it is preferable to proceed, not from original to copy, but from verbal description to copy. I would like to make one exception; I would like to be able to copy linear chains of rigid elements. Now this is very easy. For the real reason it is harder to copy an existing automaton than its description is that the existing automaton does not conform with our habit of linearity, its parts being connected with each other in all possible directions, and it's quite difficult just to check off the pieces that have already been described.[4] But it's not difficult to copy a linear chain of rigid elements. So I will assume that there exists an automaton B which has this property: If you provide B with a description of anything, it consumes it and produces two copies of this description.

Please consider that after I have described these two elementary steps, one may still hold the illusion that I have not broken the principle of the degeneracy of complication. It is still not true that, starting from something, I have made something more subtle and more

[3] [The programming language of flow diagrams, invented by von Neumann, is a possible example. See p. 13 of the Introduction to the present volume.]

[4] [Compare Sec. 1.6.3 of Part II, written about 3 years later. Here von Neumann gives a more fundamental reason for having the constructing automaton work from a description of an automaton rather than from the automaton itself.]

involved. The general constructive automaton A produces only X when a complete description of X is furnished it, and on any reasonable view of what constitutes complexity, this description of X is as complex as X itself. The general copying automaton B produces two copies of $\phi(X)$, but the juxtaposition of two copies of the same thing is in no sense of higher order than the thing itself. Furthermore, the extra unit B is required for this copying.

Now we can do the following thing. We can add a certain amount of control equipment C to the automaton $A + B$. The automaton C dominates both A and B, actuating them alternately according to the following pattern. The control C will first cause B to make two copies of $\phi(X)$. The control C will next cause A to construct X at the price of destroying one copy of $\phi(X)$. Finally, the control C will tie X and the remaining copy of $\phi(X)$ together and cut them loose from the complex $(A + B + C)$. At the end the entity $X + \phi(X)$ has been produced.

Now choose the aggregate $(A + B + C)$ for X. The automaton $(A + B + C) + \phi(A + B + C)$ will produce $(A + B + C) + \phi(A + B + C)$. Hence auto-reproduction has taken place.

[The details are as follows. We are given the universal constructor $(A + B + C)$, to which is attached a description of itself, $\phi(A + B + C)$. Thus the process of self-reproduction starts with $(A + B + C) + \phi(A + B + C)$. Control C directs B to copy the description twice; the result is $(A + B + C) + \phi(A + B + C) + \phi(A + B + C)$. Then C directs A to produce the automaton $A + B + C$ from one copy of the description; the result is $(A + B + C) + (A + B + C) + \phi(A + B + C)$. Finally, C ties the new automaton and its description together and cuts them loose. The final result consists of the two automata $(A + B + C)$ and $(A + B + C) + \phi(A + B + C)$. If B were to copy the description thrice, the process would start with one copy of $(A + B + C) + \phi(A + B + C)$ and terminate with two copies of this automaton. In this way, the universal constructor reproduces itself.]

This is not a vicious circle. It is quite true that I argued with a variable X first, describing what C is supposed to do, and then put something which involved C for X. But I defined A and B exactly, before I ever mentioned this particular X, and I defined C in terms which apply to any X. Therefore, in defining A, B, and C, I did not make use of what X is to be, and I am entitled later on to use an X which refers explicitly to A, B, and C. The process is not circular.

The general constructive automaton A has a certain creative ability, the ability to go from a description of an object to the object. Like-

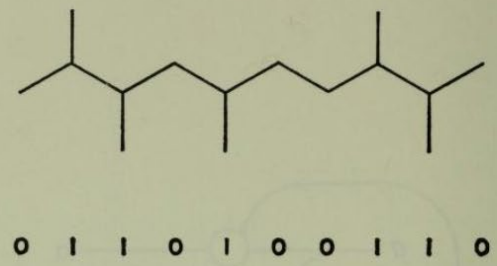


Fig. 2. A binary tape constructed from rigid elements

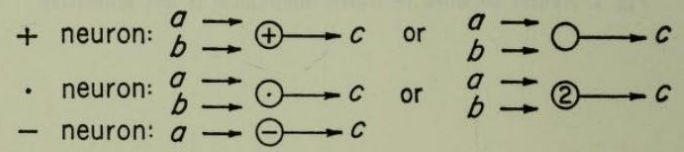
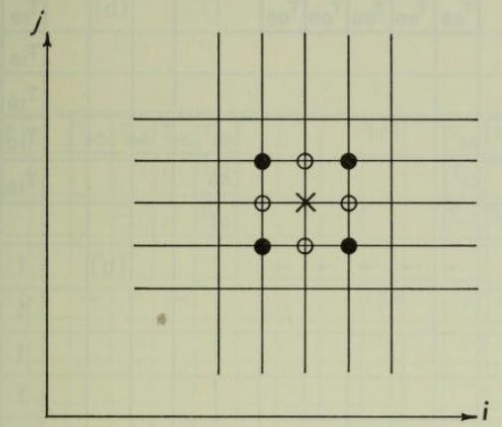
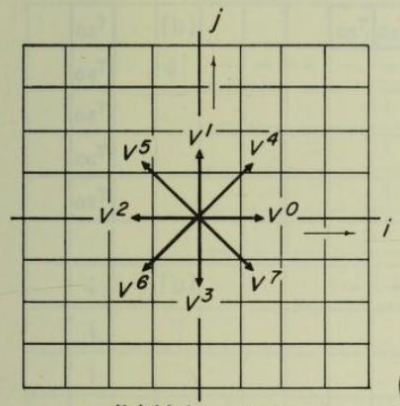


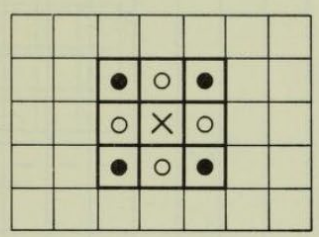
Fig. 3. The basic neurons



(a) Nearest (○) and next nearest (●) neighbors of X



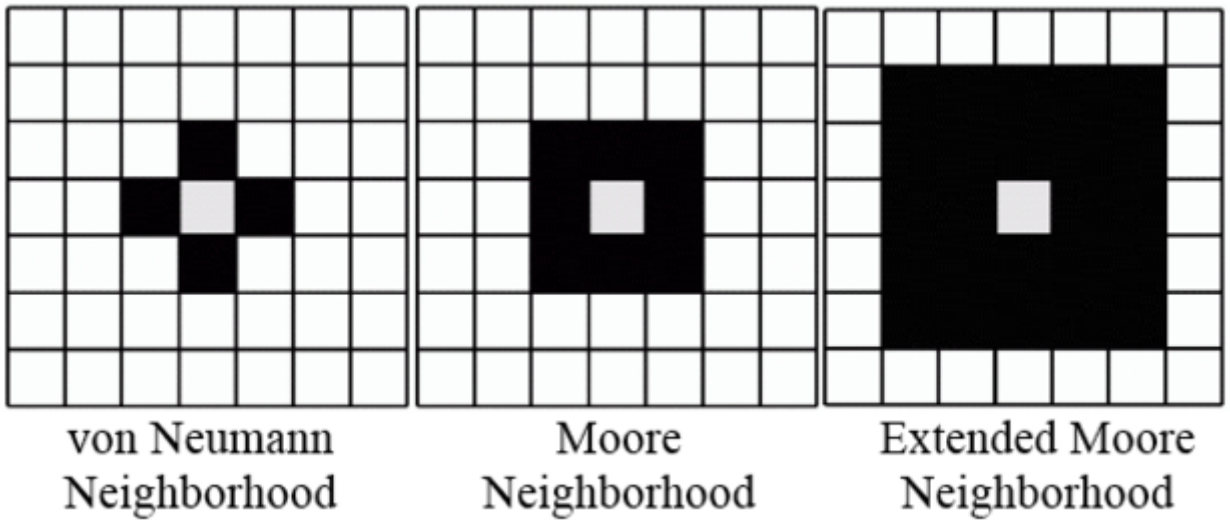
(b) Unit vectors



(c) Nearest (○) and next nearest (●) neighbors of X

Fig. 4. The quadratic lattice

https://www.openabm.org/book/export/html/1949



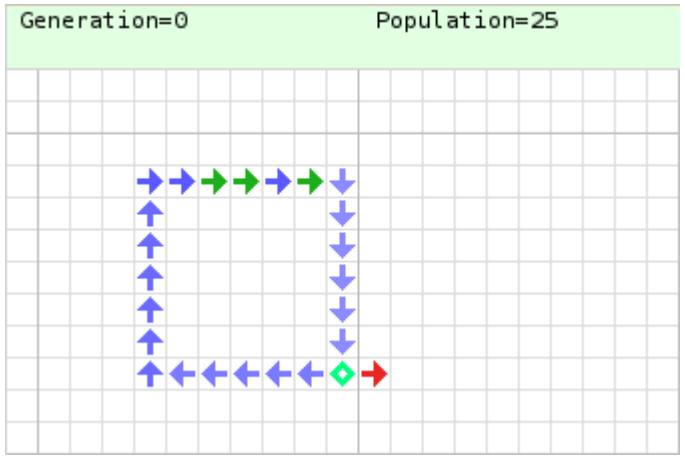
An Implementation of von Neumann's Self-Reproducing Machine

Umberto Pesavento
Princeton University
Class of 2000
Princeton, NJ 08544
pesavent@intercity.shiny.it

Keywords
cellular automata, self-reproduction, universal constructor

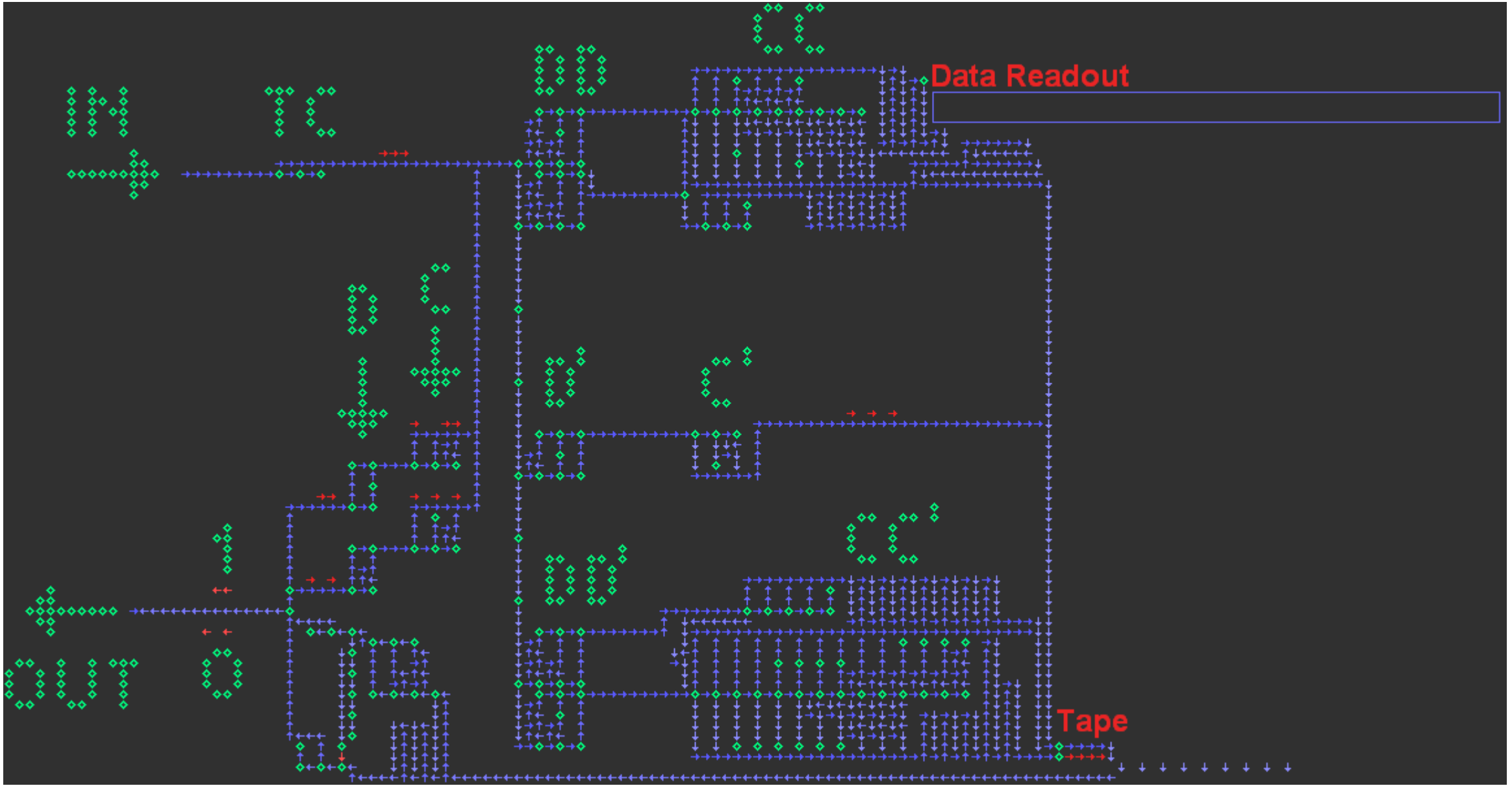
Abstract This article describes in detail an implementation of John von Neumann's *self-reproducing machine*. Self-reproduction is achieved as a special case of construction by a *universal constructor*. The theoretical proof of the existence of such machines was given by John von Neumann in the early 1950s [6], but was first implemented in 1994, by the author in collaboration with R. Nobili. Our implementation relies on an extension of the state-transition rule of von Neumann's original cellular automaton. This extension was introduced to simplify the design of the constructor. The main operations in our constructor can be mapped into operations of von Neumann's machine.

1. a **ground** state **U** (48, 48, 48)
2. the **transition** or **sensitised** states (in 8 substates)
    1. **S** (newly sensitised)
    2. $\mathbf{S_0}$ – (sensitised, having received no input for one cycle)
    3. $\mathbf{S_{00}}$ – (sensitised, having received no input for two cycles)
    4. $\mathbf{S_{000}}$ – (sensitised, having received no input for three cycles)
    5. $\mathbf{S_{01}}$ – (sensitised, having received no input for one cycle and then an input for one cycle)
    6. $\mathbf{S_1}$ – (sensitised, having received an input for one cycle)
    7. $\mathbf{S_{10}}$ – (sensitised, having received an input for one cycle and then no input for one cycle)
    8. $\mathbf{S_{11}}$ – (sensitised, having received input for two cycles)
3. the **confluent** states (in 4 states of excitation)
    1. $\mathbf{C_{00}}$ – quiescent (and will also be quiescent next cycle)
    2. $\mathbf{C_{01}}$ – next-excited (now quiescent, but will be excited next cycle)
    3. $\mathbf{C_{10}}$ – excited (but will be quiescent next cycle)
    4. $\mathbf{C_{11}}$ – excited next-excited (currently excited and will be excited next cycle)
4. the **ordinary transmission** states (in 4 directions, excited or quiescent, making 8 states)
    1. North-directed (excited and quiescent)
    2. South-directed (excited and quiescent)
    3. West-directed (excited and quiescent)
    4. East-directed (excited and quiescent)
5. the **special transmission** states (in 4 directions, excited or quiescent, making 8 states)
    1. North-directed (excited and quiescent)
    2. South-directed (excited and quiescent)
    3. West-directed (excited and quiescent)
    4. East-directed (excited and quiescent)

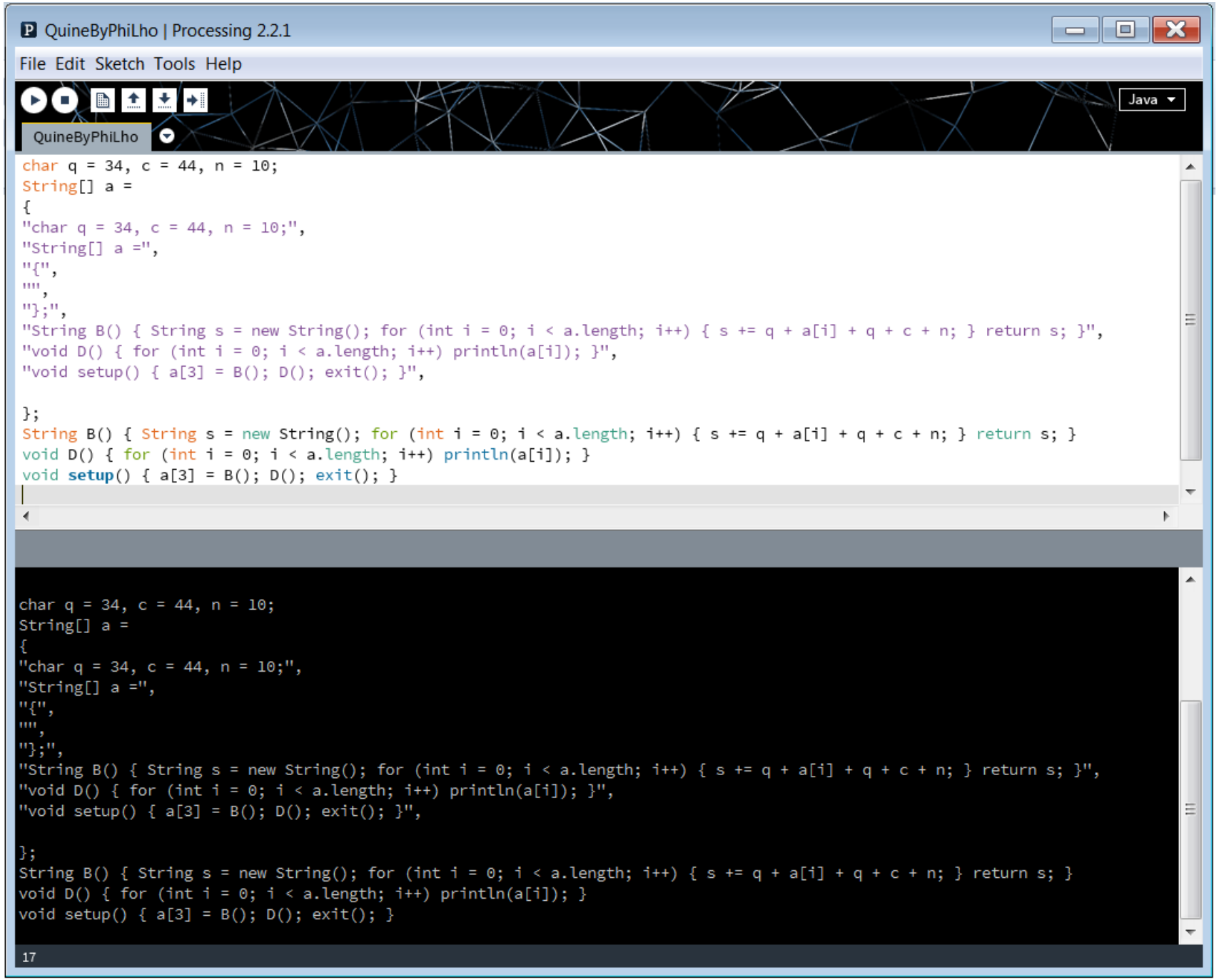


A simple configuration in von Neumann's cellular automaton. A binary signal is passed repeatedly around the blue wire loop, using excited and quiescent *ordinary transmission states*. A confluent cell duplicates the signal onto a length of red wire consisting of *special transmission states*. The signal passes down this wire and constructs a new cell at the end. This particular signal (1011) codes for an east-directed special transmission state, thus extending the red wire by one cell each time. During construction, the new cell passes through several sensitised states, directed by the binary sequence

Source:
https://en.wikipedia.org/wiki/Von_Neumann_cellular_automaton



Source: https://en.wikipedia.org/wiki/Von_Neumann_cellular_automaton

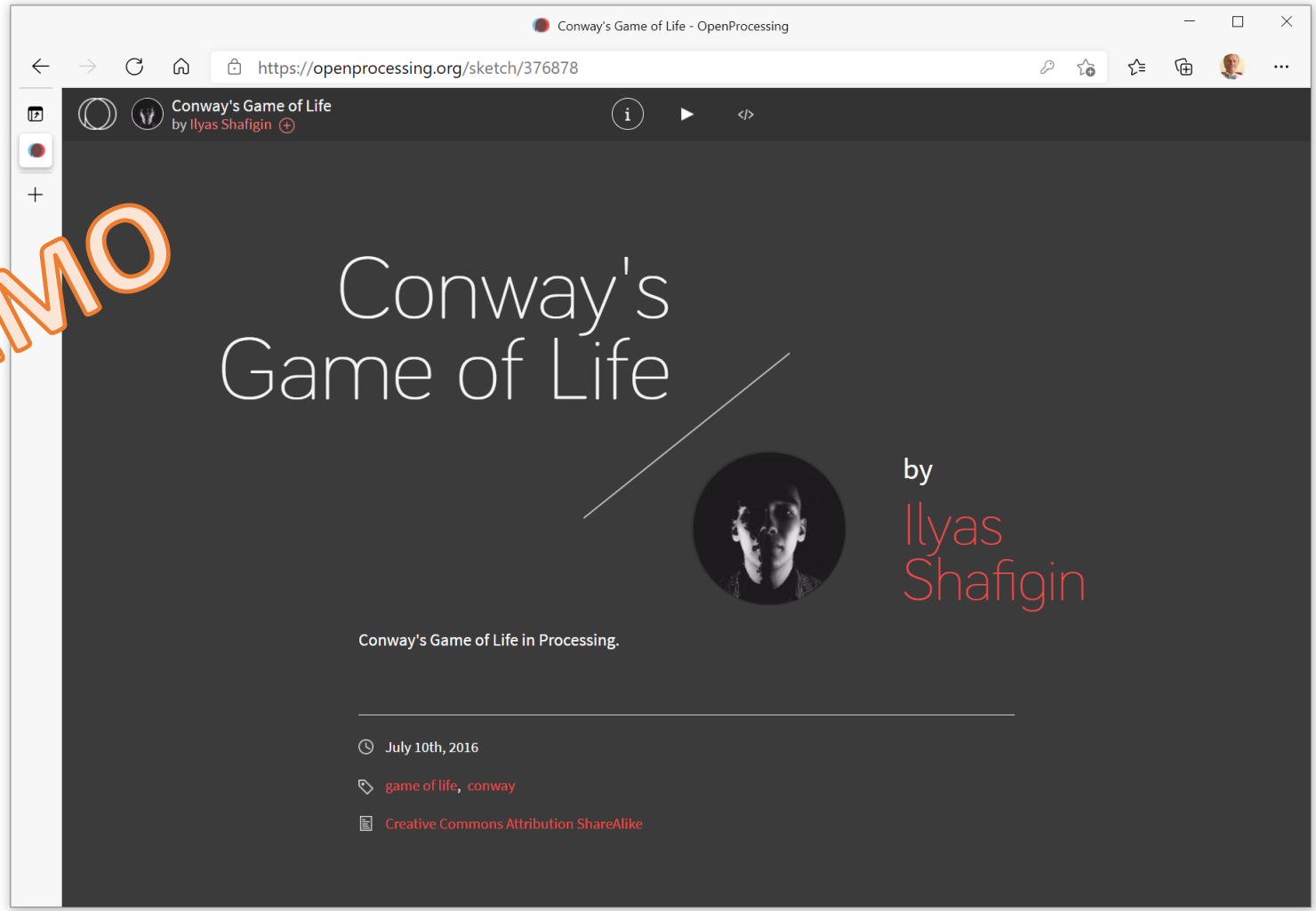


https://processing.org/discourse/beta/num 1243254687.html

Conway’s game of life

- Any live cell with fewer than two live neighbours dies, as if caused by underpopulation.
- Any live cell with two or three live neighbours lives on to the next generation.
- Any live cell with more than three live neighbours dies, as if by overpopulation.
- Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

(https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life)


Conway's Game of Life - OpenProcessing
https://openprocessing.org/sketch/376878
Conway's Game of Life
by Ilyas Shafigin
Conway's Game of Life
by
Ilyas Shafigin
Conway's Game of Life in Processing.
July 10th, 2016
game of life, conway
Creative Commons Attribution ShareAlike

DEMO

Life | Processing 3.3.7

File Edit Sketch Debug Tools Help

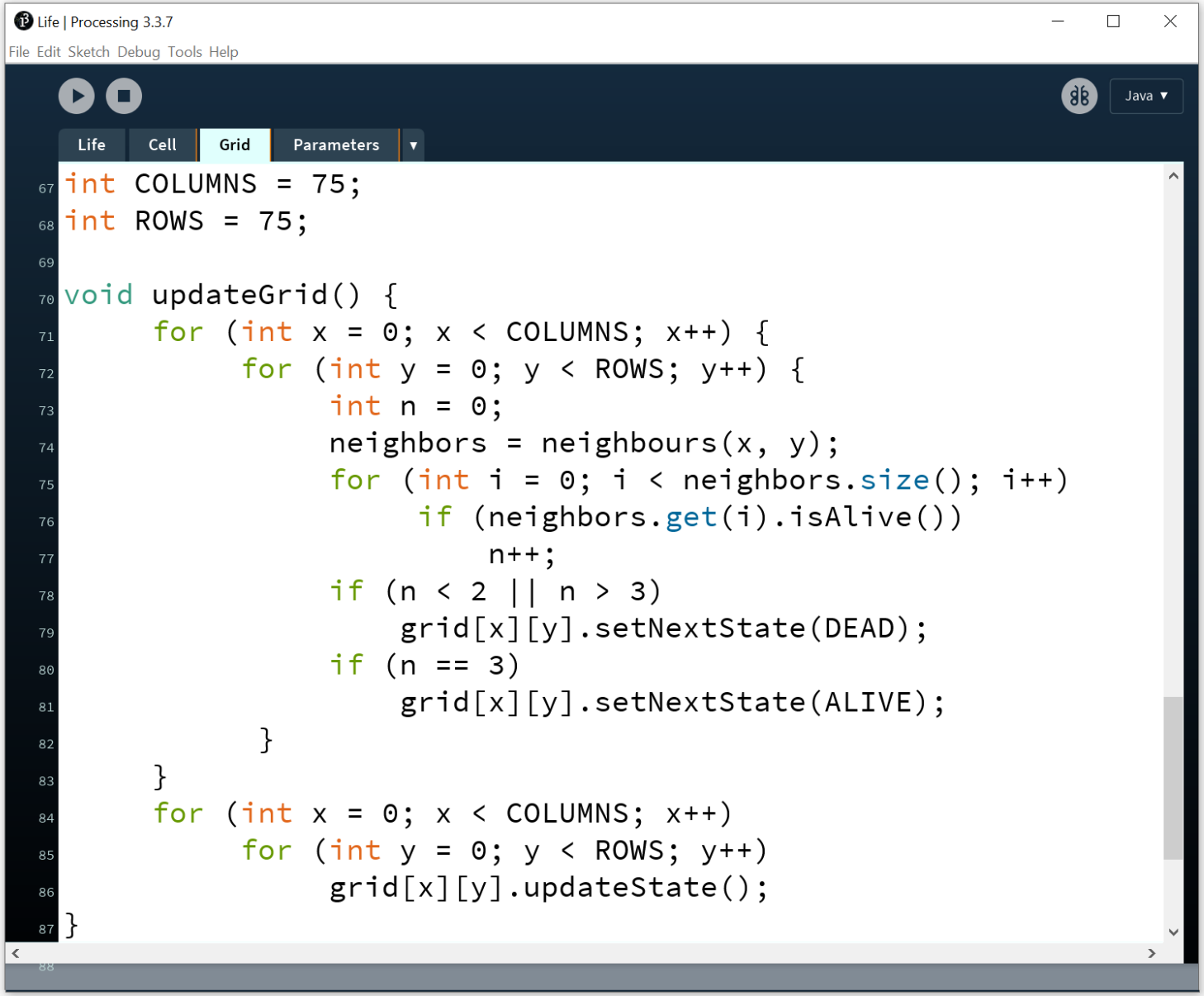
Java ▼

Life | Cell | Grid | Parameters ▾

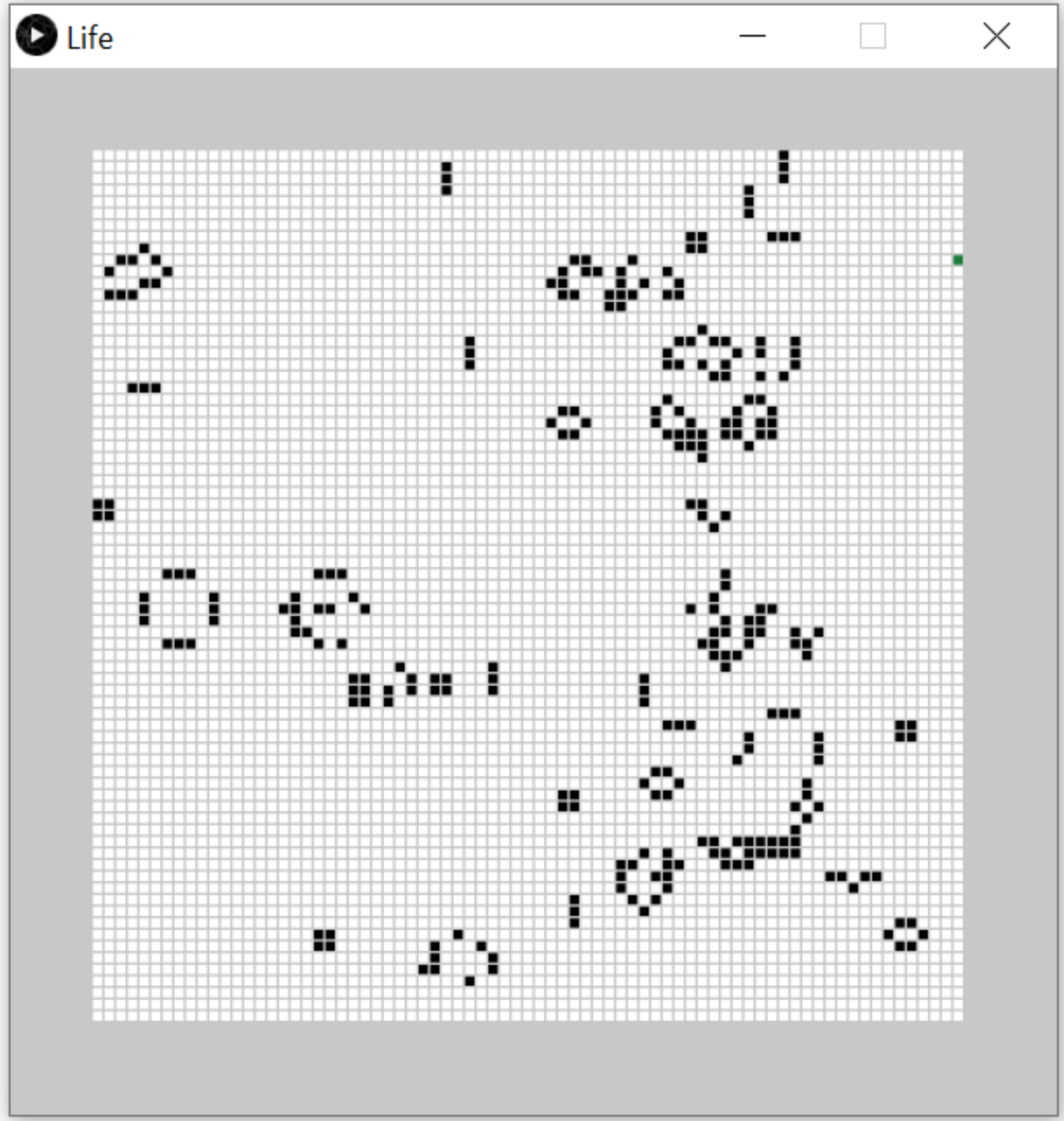
```
static final int DEAD = 0;
static final int ALIVE = 1;

class Cell {
      int state;
      int nextState;
      Cell(int s) {
            setState(s);
      }
      void setState(int s) {
            state = s;
      }
      void setNextState(int s) {
            nextState = s;
      }
      boolean isAlive() {
               return state == ALIVE;
      }
      void updateState() {
            state = nextState;
      }
}
```

`Cell[][] grid;`

Adapted from: Game of Life by Ilyas Shafigin, https://openprocessing.org/sketch/376878



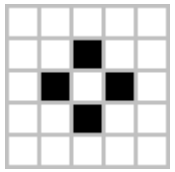
Adapted from: Game of Life by Ilyas Shafigin, https://openprocessing.org/sketch/376878

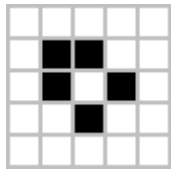

Life

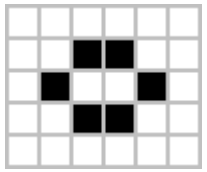

DIGITAL EQUIPMENT CORPORATION

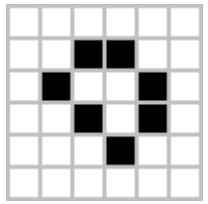
Static

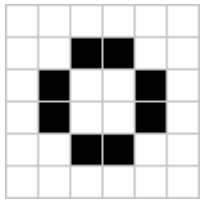

block



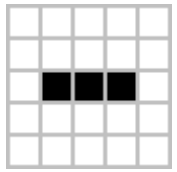


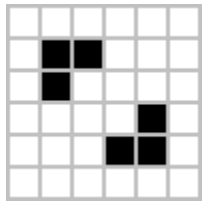

beehive

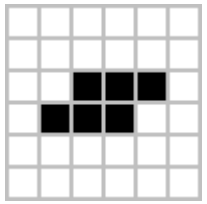


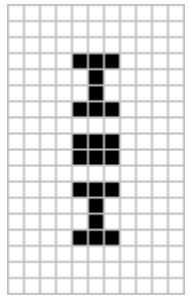

pond

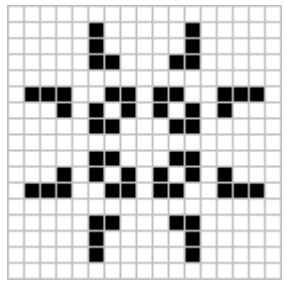
Pulsing


blinker

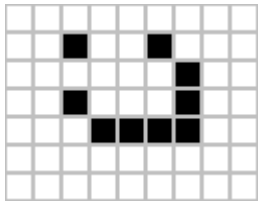


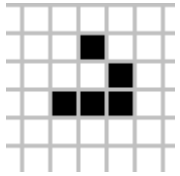


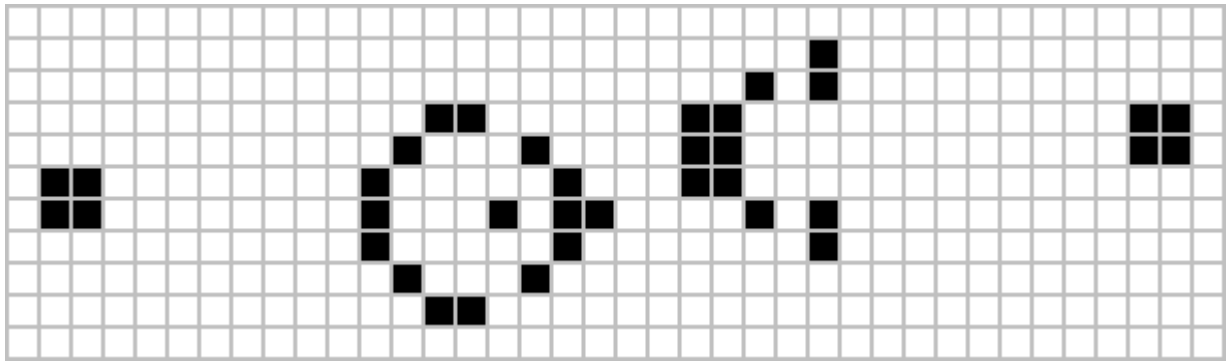




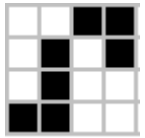
Crawling


lightweight spaceship (c/2)

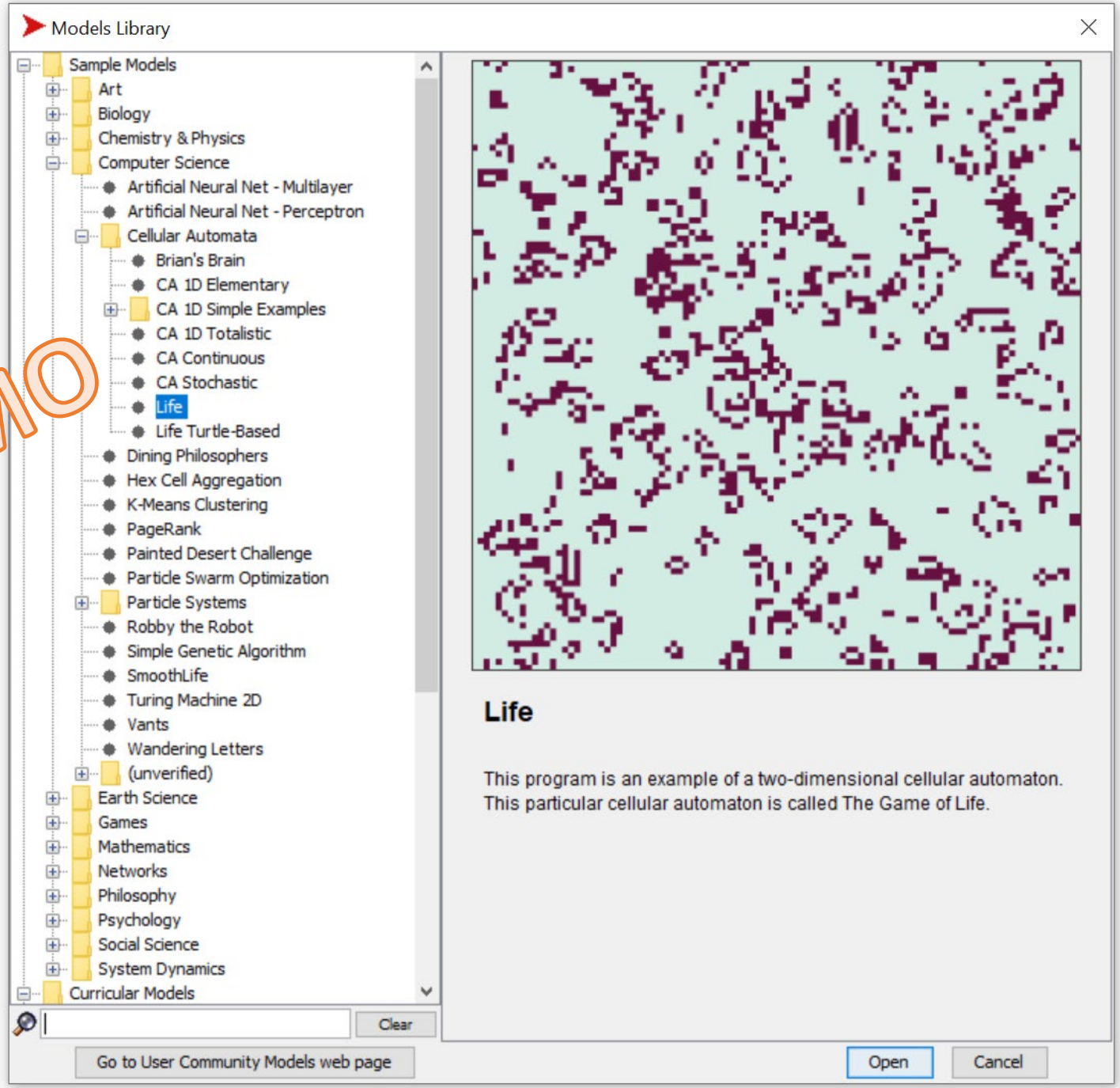

Glider (c/4)

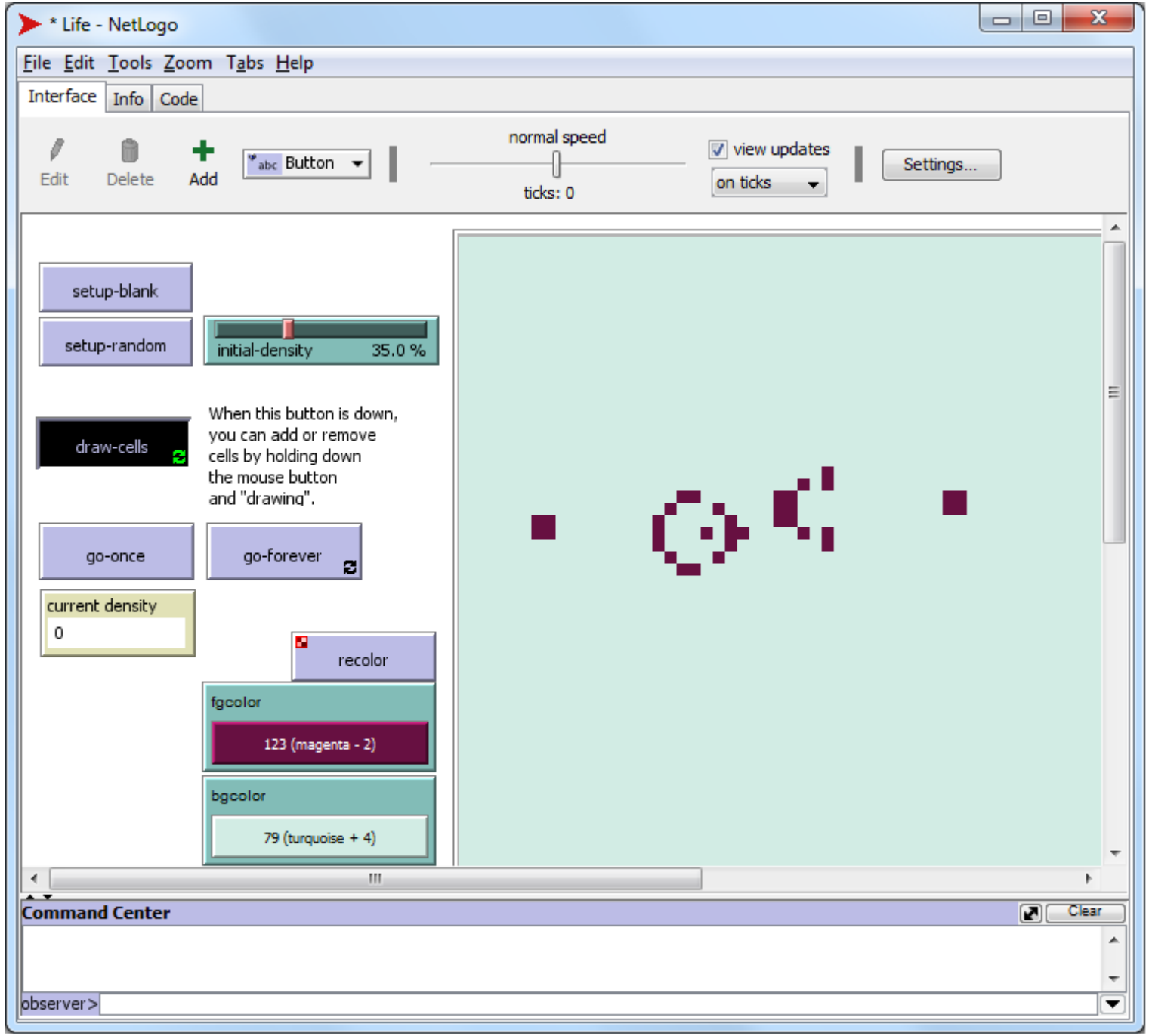


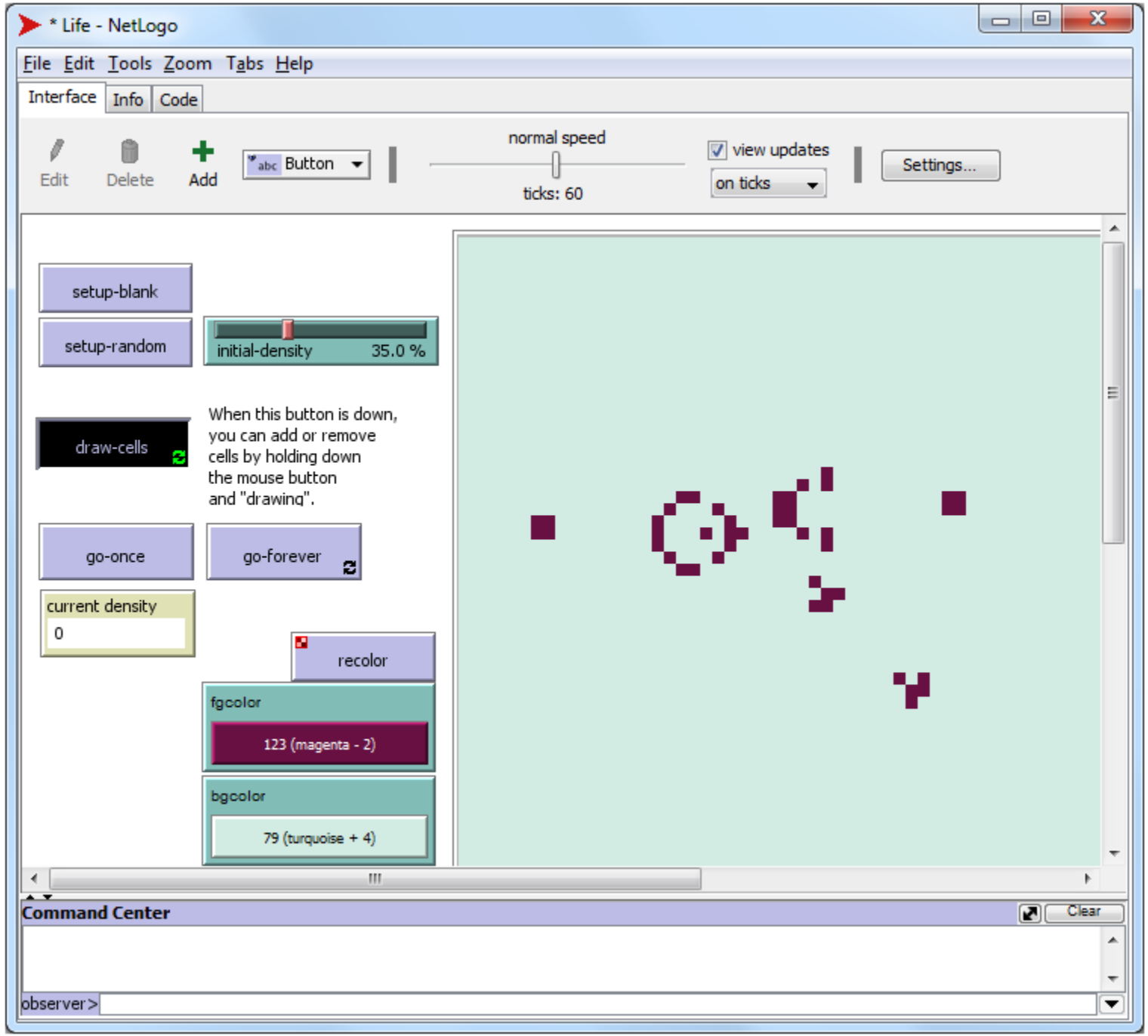
Gosper’s glider gun (wikimedia)



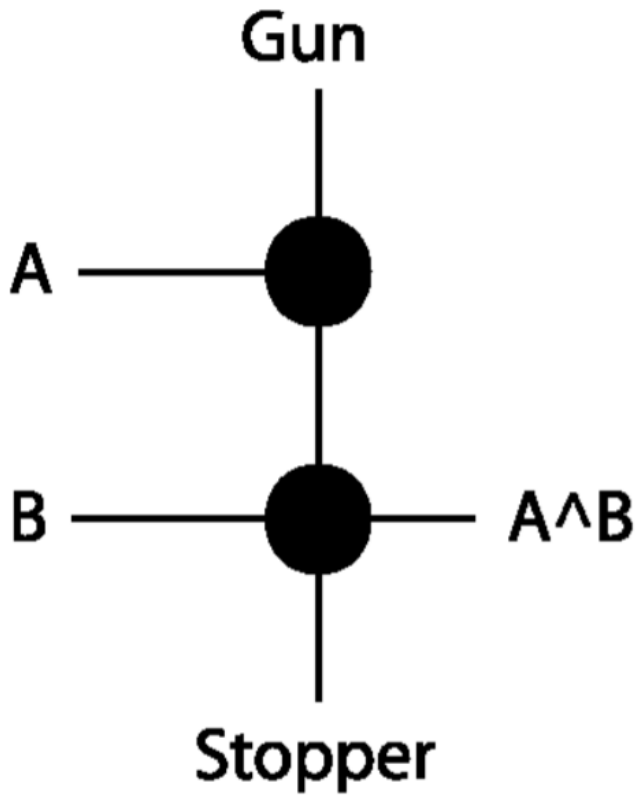
eater,
stopper


Models Library
Sample Models
Art
Biology
Chemistry & Physics
Computer Science
Artificial Neural Net - Multilayer
Artificial Neural Net - Perceptron
Cellular Automata
Brian's Brain
CA 1D Elementary
CA 1D Simple Examples
CA 1D Totalistic
CA Continuous
CA Stochastic
Life
Life Turtle-Based
Dining Philosophers
Hex Cell Aggregation
K-Means Clustering
PageRank
Painted Desert Challenge
Particle Swarm Optimization
Particle Systems
Robby the Robot
Simple Genetic Algorithm
SmoothLife
Turing Machine 2D
Vants
Wandering Letters
(unverified)
Earth Science
Games
Mathematics
Networks
Philosophy
Psychology
Social Science
System Dynamics
Curricular Models
Clear
Life
This program is an example of a two-dimensional cellular automaton.
This particular cellular automaton is called The Game of Life.
Go to User Community Models web page
Open
Cancel

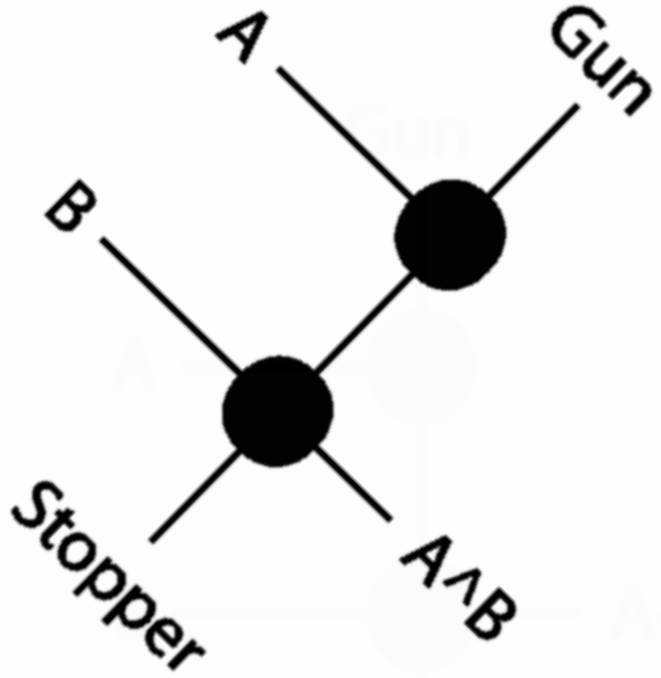

* Life - NetLogo
File Edit Tools Zoom Tabs Help
Interface Info Code
Edit
Delete
Add
Button
normal speed
ticks: 0
view updates
on ticks
Settings...
setup-blank
setup-random
initial-density 35.0 %
draw-cells
When this button is down, you can add or remove cells by holding down the mouse button and "drawing".
go-once
go-forever
current density
0
recolor
fgcolor
123 (magenta - 2)
bgcolor
79 (turquoise + 4)
Command Center
Clear
observer>

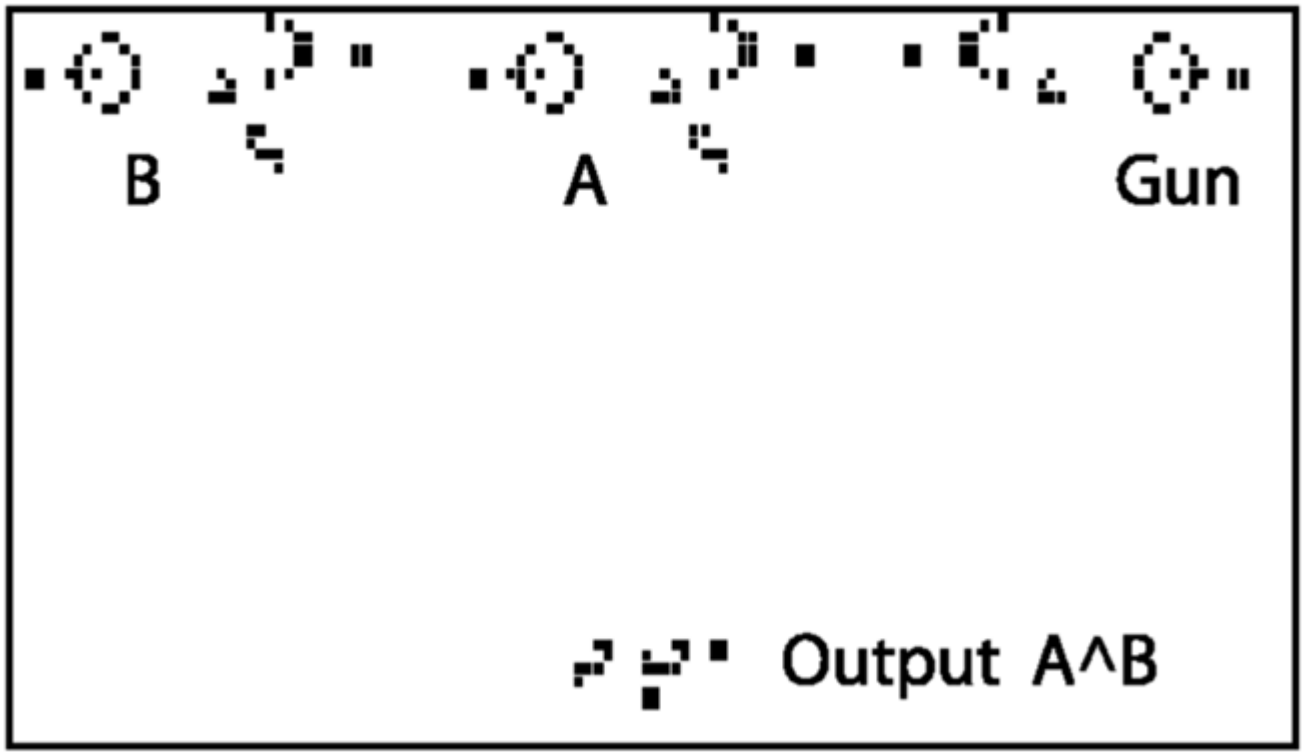

* Life - NetLogo
File Edit Tools Zoom Tabs Help
Interface Info Code
Edit
Delete
Add
Button
normal speed
ticks: 60
view updates
on ticks
Settings...
setup-blank
setup-random
initial-density 35.0 %
draw-cells
When this button is down, you can add or remove cells by holding down the mouse button and "drawing".
go-once
go-forever
current density
0
recolor
fgcolor
123 (magenta - 2)
bgcolor
79 (turquoise + 4)
Command Center
Clear
observer>

The AND-Gate



Source: Jean Philippe Rennard,
Implementation of Logical functions in the game of life,
Collision-based computing, 2002 - Springer,
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.17.9114&rep=rep1&type=pdf



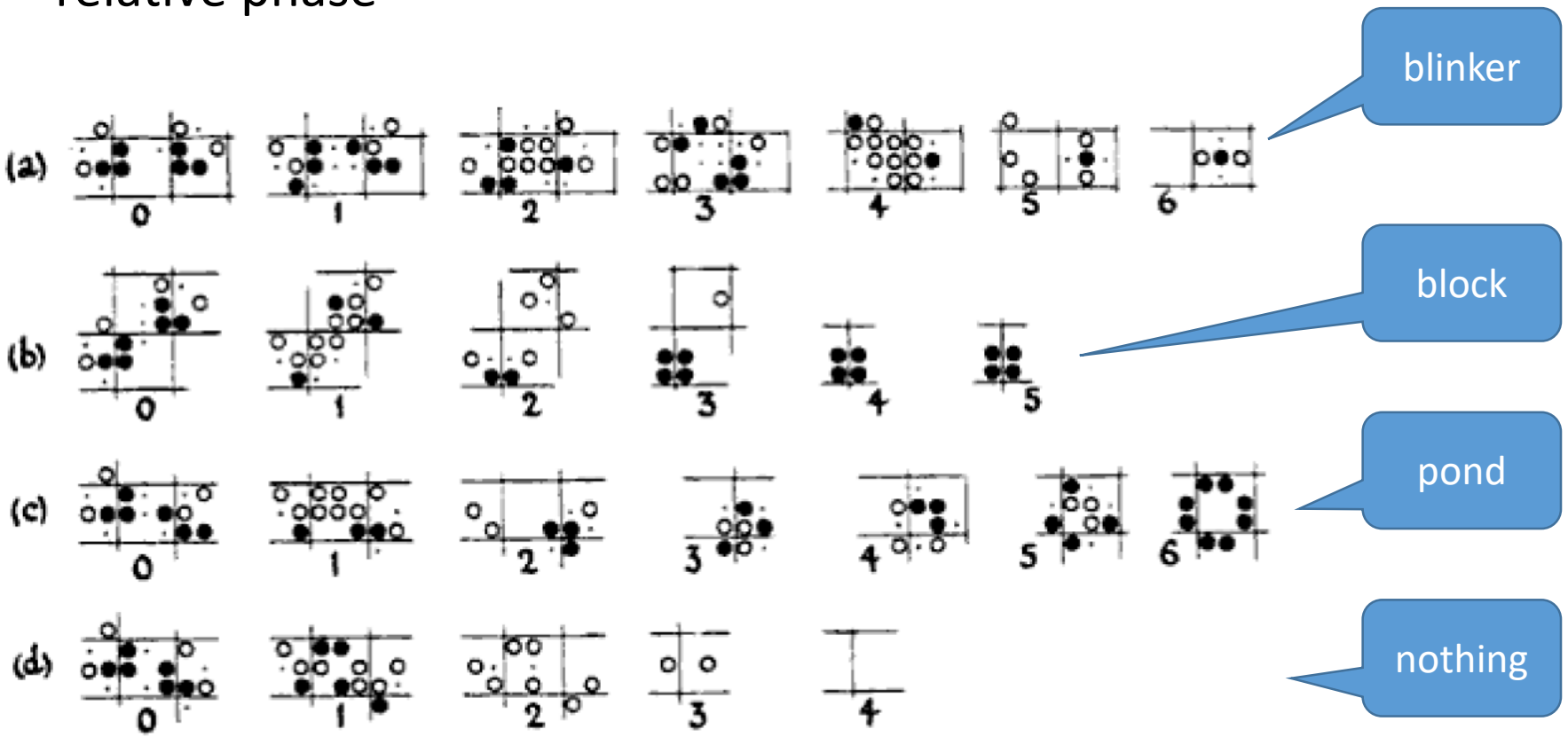


Source: Jean Philippe Rennard,
Implementation of Logical functions in the game of life,
Collision-based computing, 2002 - Springer,
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.17.9114&rep=rep1&type=pdf

Colliding gliders

outcome depends on:

- relative position
- relative phase

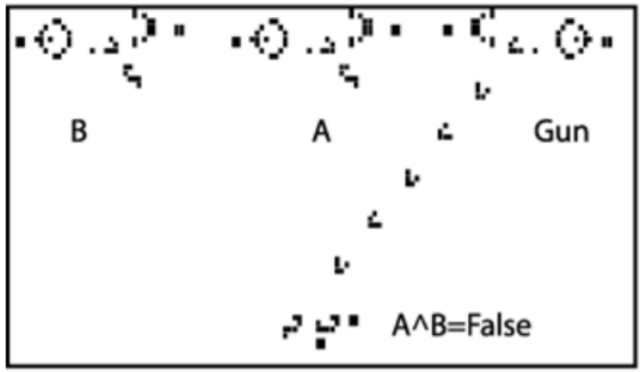


Source: Melissa Gymrek's lecture notes (MIT).
http://web.mit.edu/sp.268/www/2010/lifeSlides.pdf

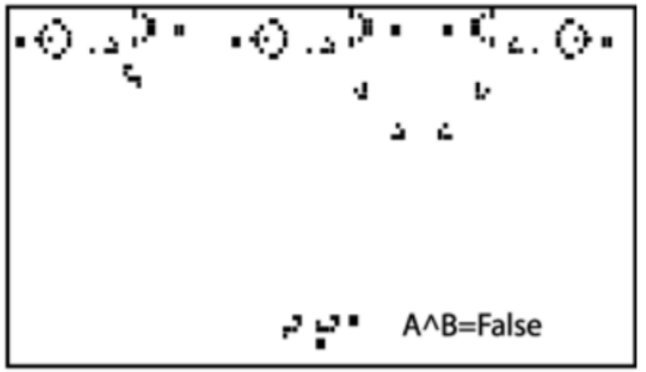
https://www.youtube.com/watch?v=vGWGeund3eA

No glider can activate the output. The result is False.

A False, B False



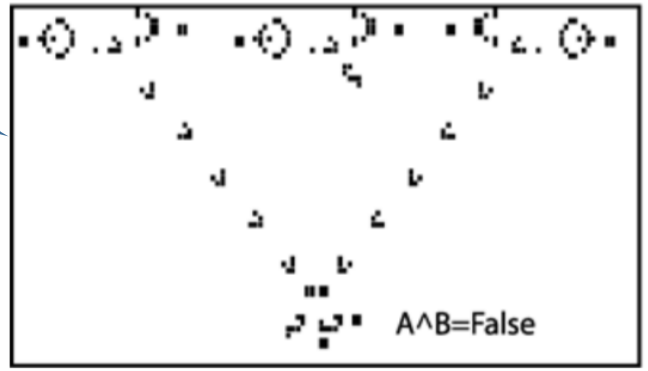
A True, B False



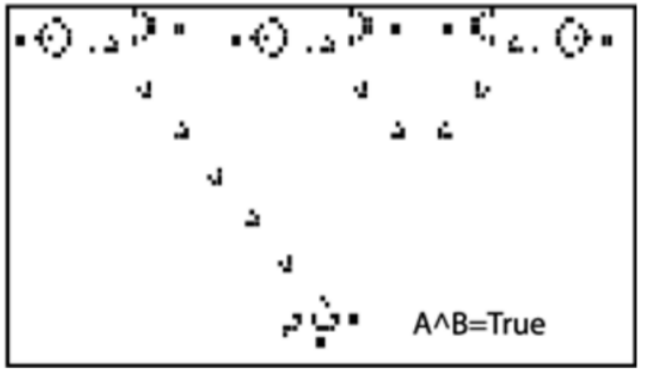
A stops the gun. Since B is false, it cannot activate the output. The result is False.

B cannot join the output since the gun stops it. The result is False.

A False, B True



A True, B True

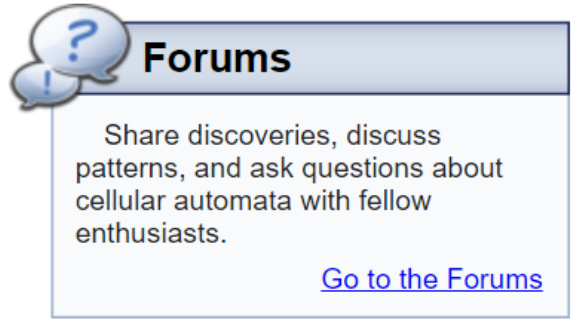


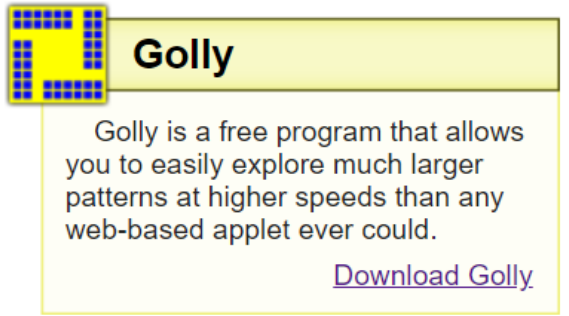
A stops the gun and B can activate the output. The result is True.

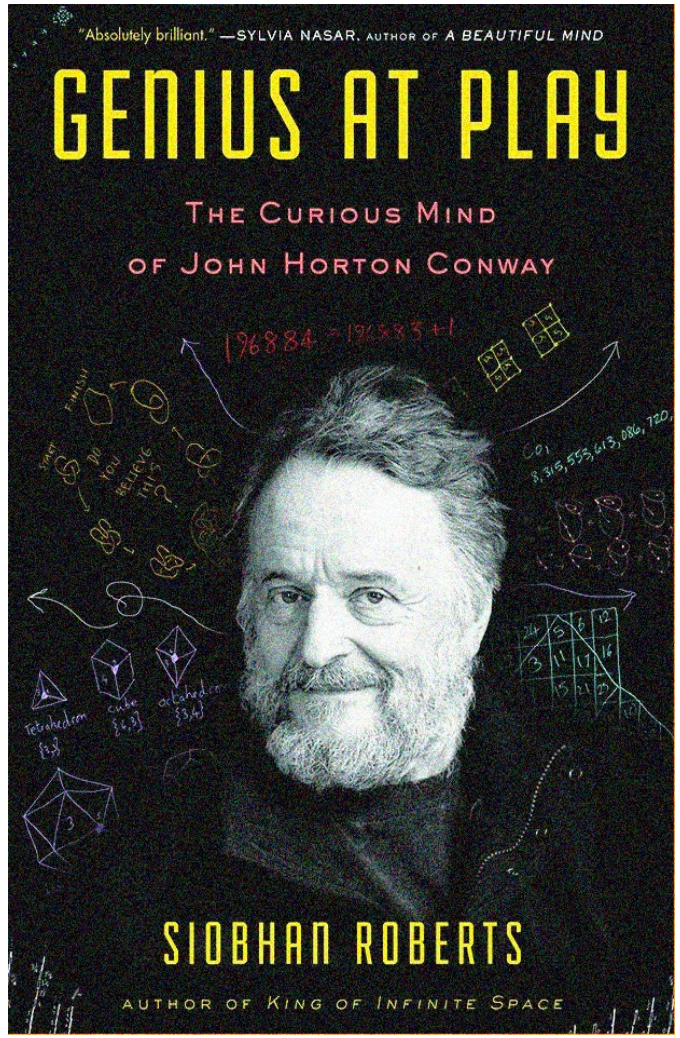
Source: Jean Philippe Rennard, Implementation of Logical functions in the game of life, Collision-based computing, 2002 - Springer, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.17.9114&rep=rep1&type=pdf

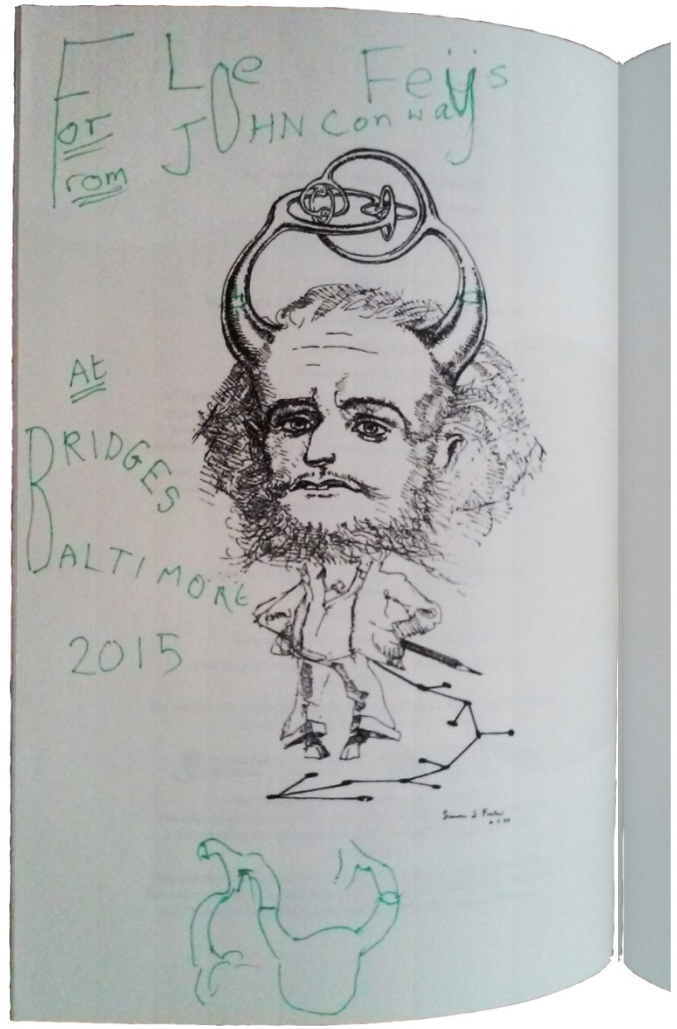
Conway’s game of life: conclusions

- There is a large number of life forms, many still to be discovered
- Boolean gates can be simulated
- Even a universal computer can be simulated (Turing machine)
- There is an active research community (www.conwaylife.com)





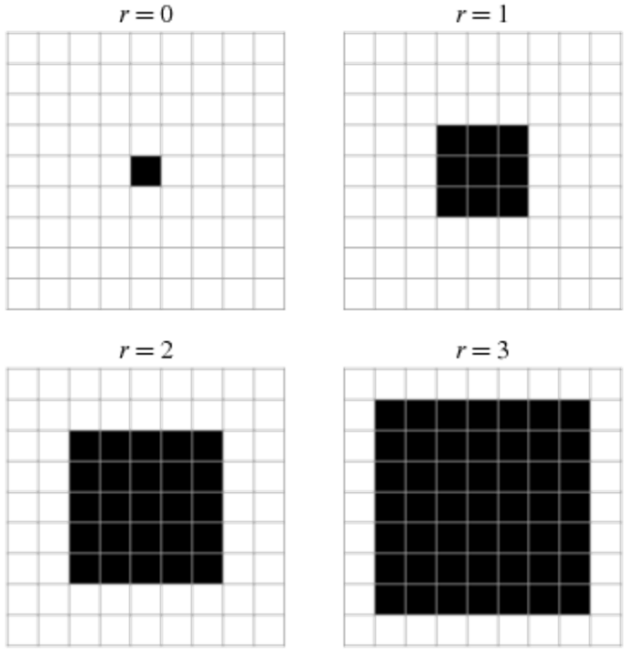




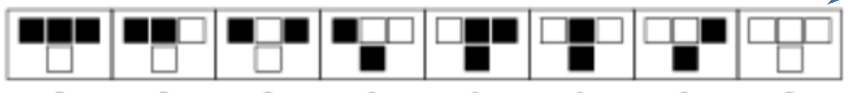
Meeting John Conway at Bridges 2015

Towards elementary rules: one-dimension, r = 1

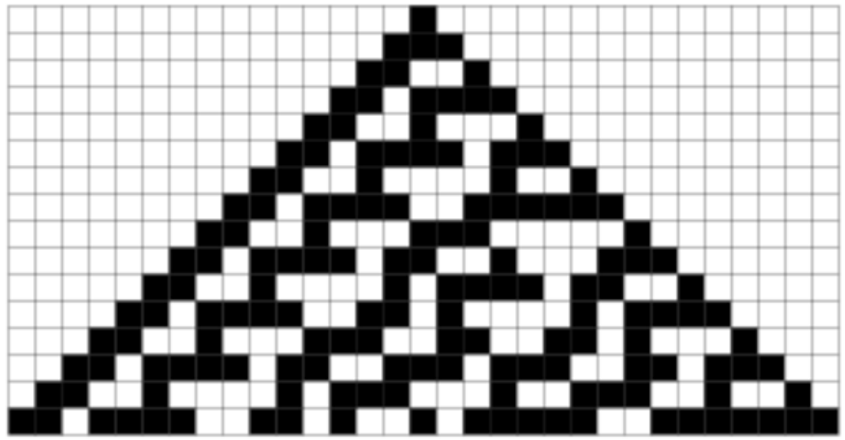
Mathworld: Moore neighbourhood (in two dimensions)



rule 30

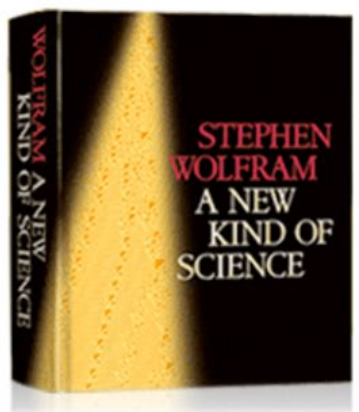


r=1



https://www.wolframscience.com/

Now available for iPad
Download now »



Also available in the classic hardcover edition with art-quality printing »

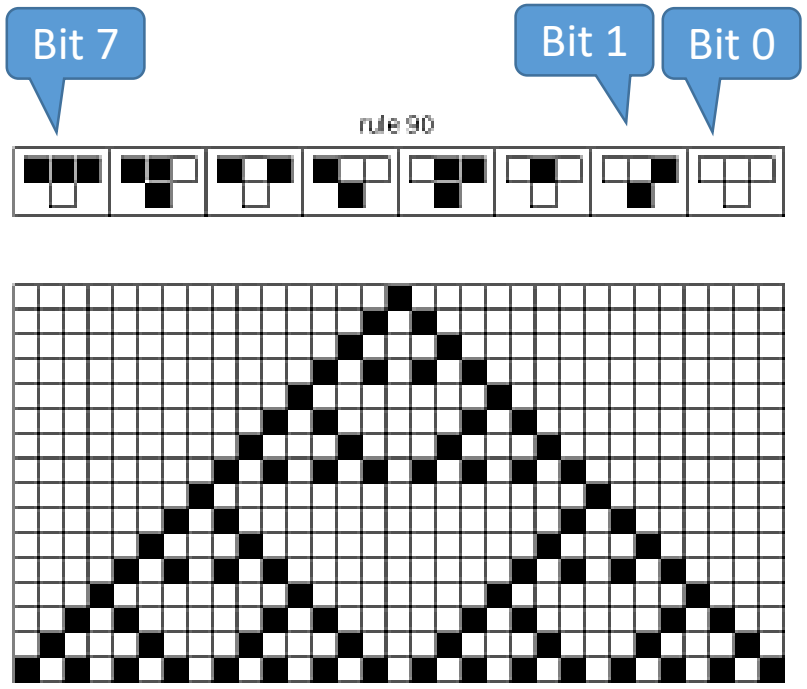
Stephen Wolfram's
A NEW KIND OF SCIENCE | ONLINE

Jump to Page | Look Up in Index | Search


SETI INSTITUTE
SCIENCE COLLOQUIUM
Astronomy & Astrophysics - Astrobiology - Climate & Geoscience - Exoplanets - Planetary Exploration - SETI
Microsoft

Enumeration scheme for two-color r=1 one-dimensional automata

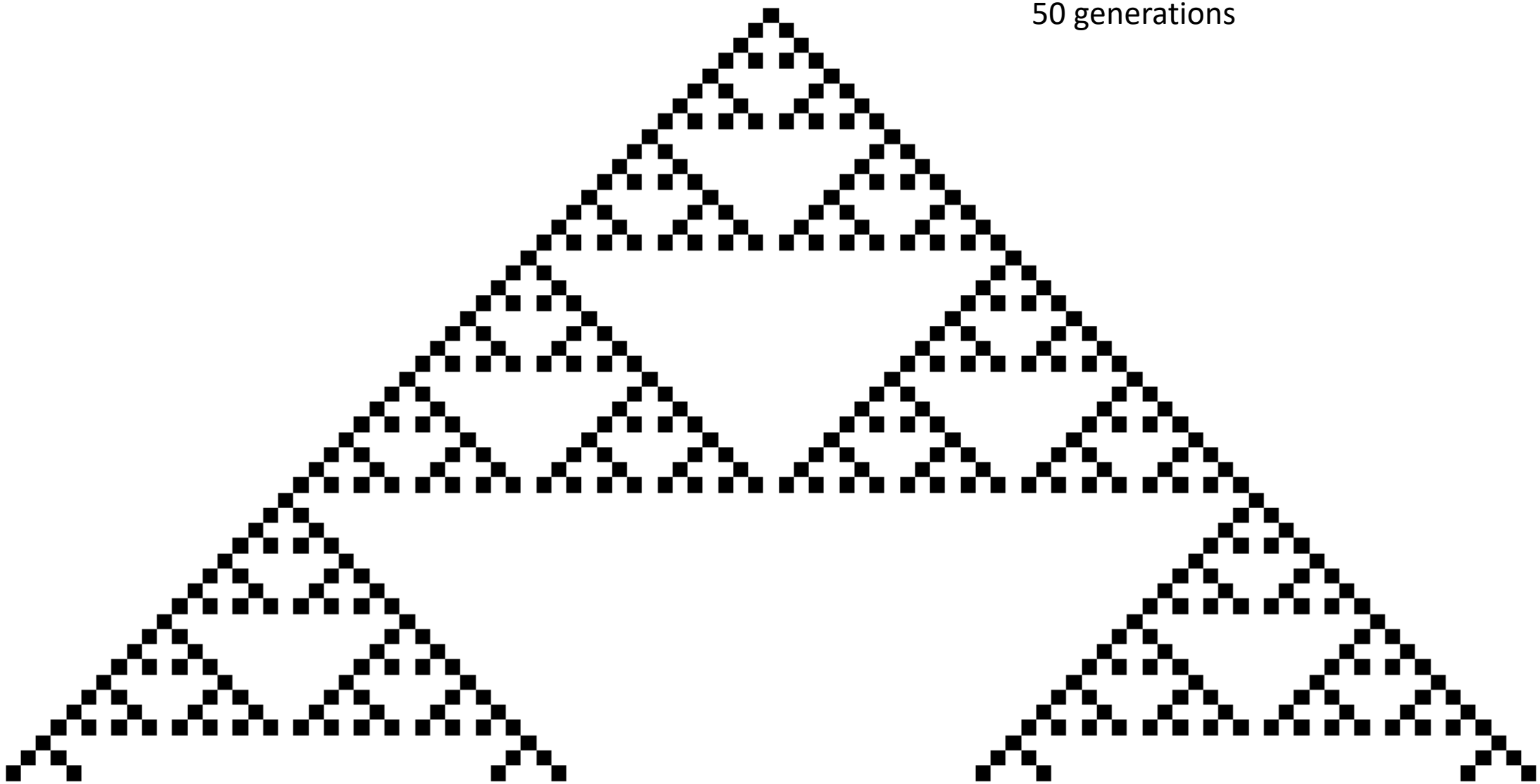
Example 01011010 (binary) = 64 + 16 + 8 + 2 = 90



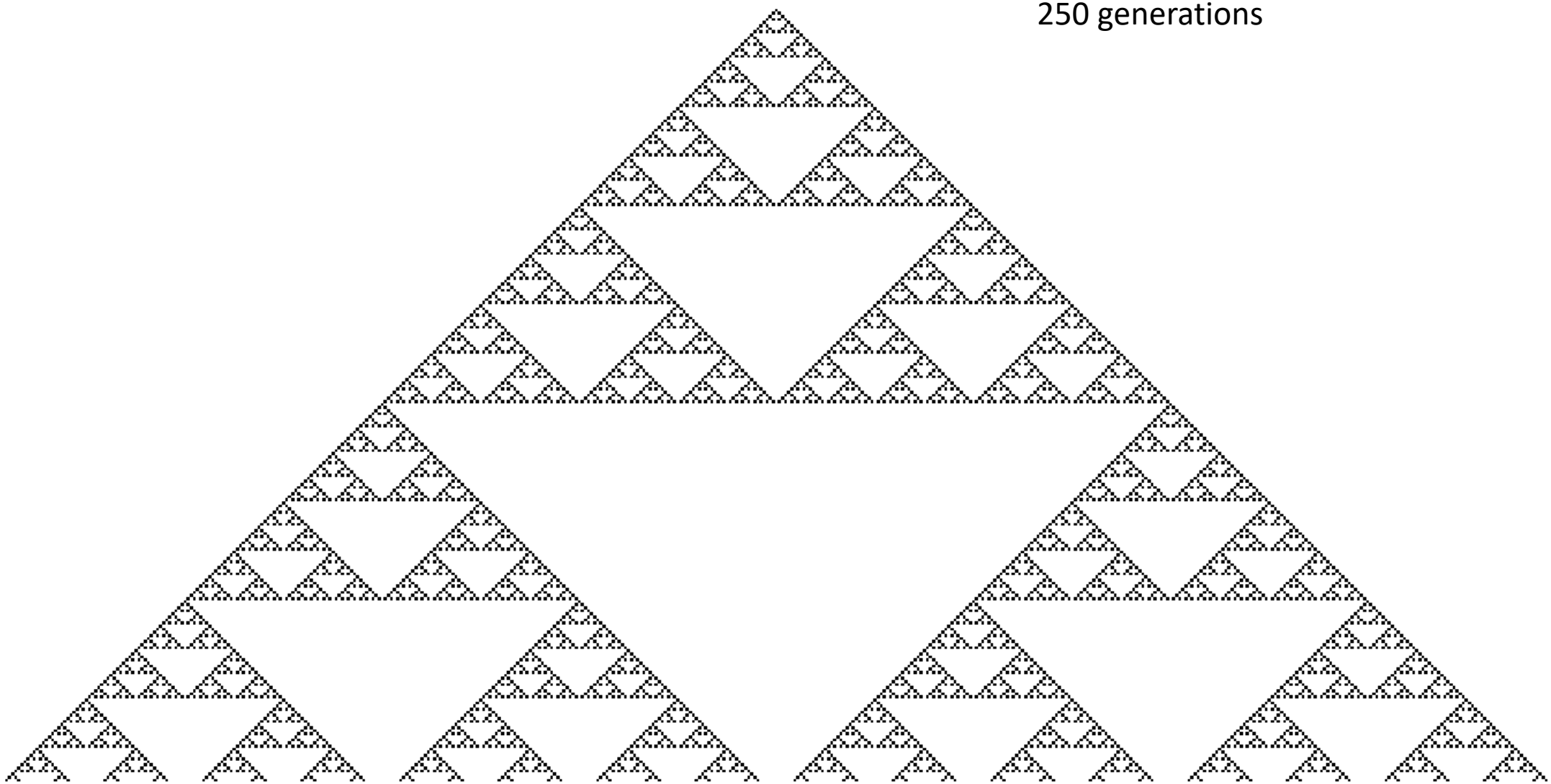
The 2D pattern formed by this rule has fractal dimension log 3 / log 2 ≈ 1.58

Source: NKS

50 generations



250 generations



This is a so-=called Sierpinski triangle, see https://en.wikipedia.org/wiki/Sierpinski_triangle

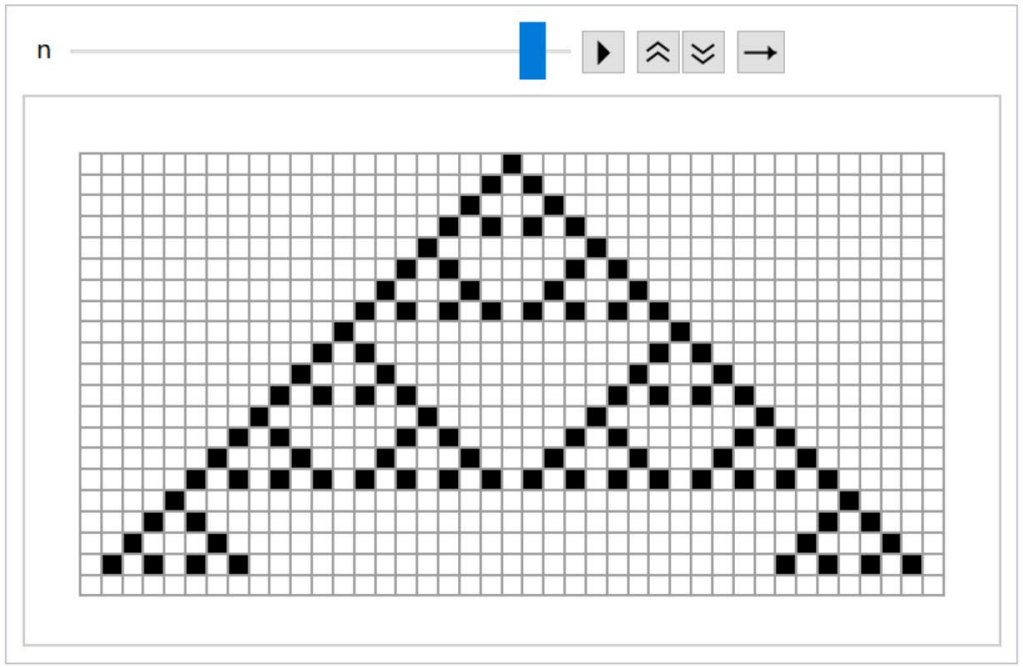
rule90.nb - Wolfram Mathematica 11.0

File Edit Insert Format Cell Graphics Evaluation Palettes Window Help

In[35]:=
```
max = 20;
RulePlot[CellularAutomaton[90]]

Animate[
 ArrayPlot[ArrayPad[CellularAutomaton[90, {{1}, 0}, {n, All}],
   {{0, max - n}, {max - n, max - n}}], Mesh → True], {n, 0, max, 1},
 AnimationRunning → False]
```


Out[3

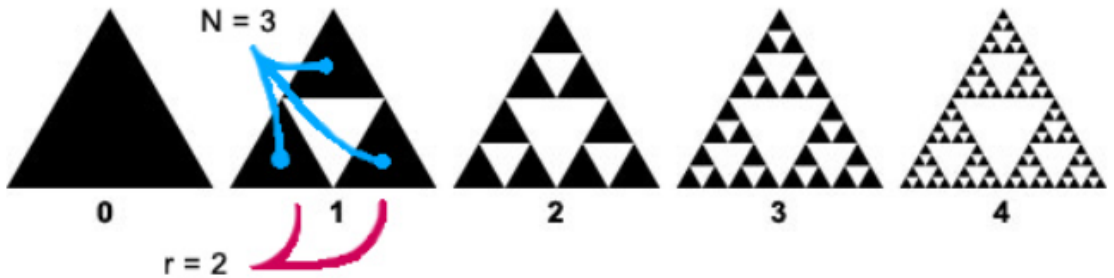
Out[37]=



200%

Fractal Dimension of the Sierpinski Triangle

Let's use the formula for scaling to determine the dimension of the Sierpinski Triangle fractal. First, take a rough guess at what you might think the dimension will be. Less than 1? Between 1 and 2? Greater than 2? Since the Sierpinski Triangle fits in plane but doesn't fill it completely, its dimension should be less than 2. Let's see if this is true.

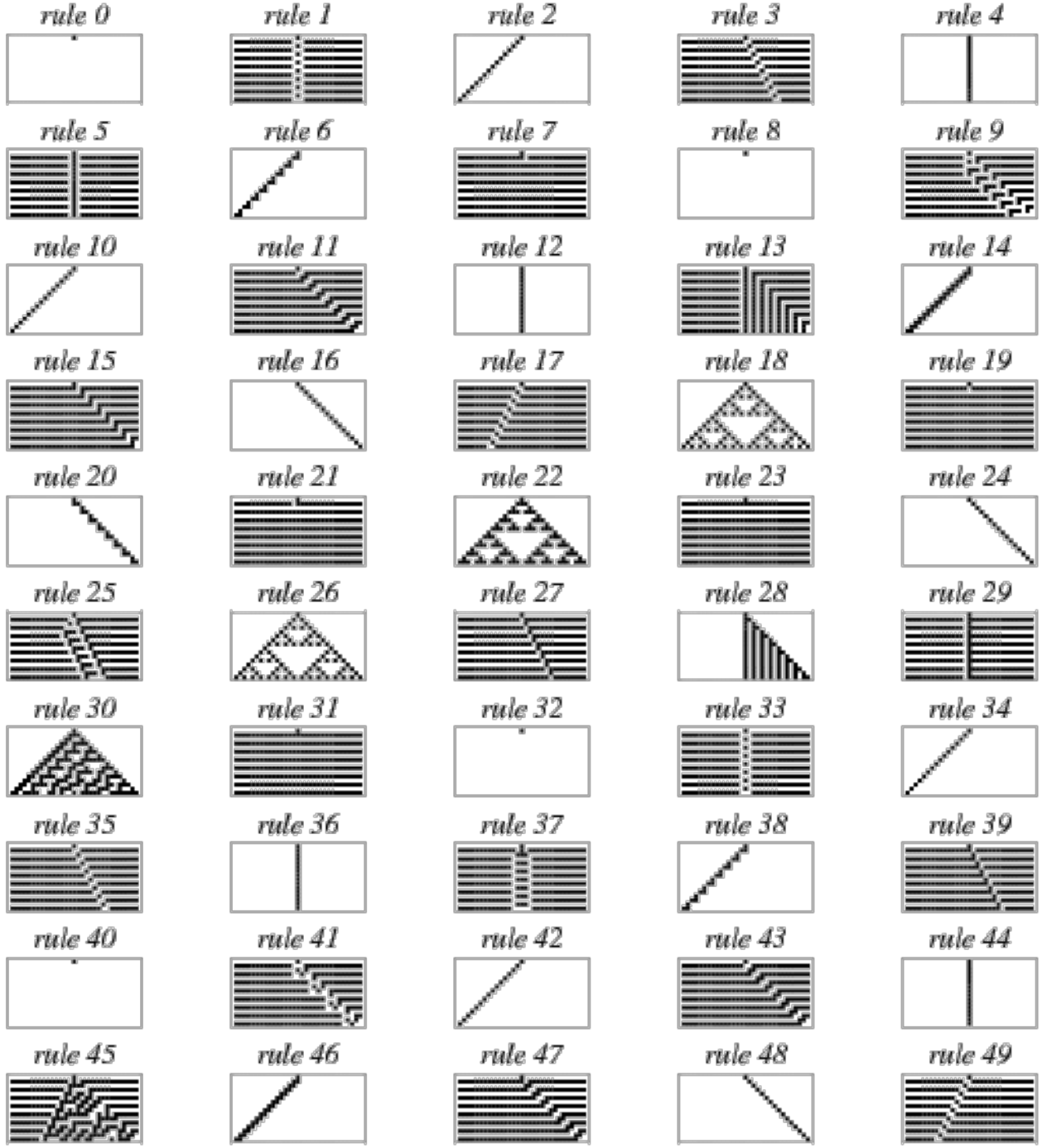


Start with the 0 order triangle in the figure above. The next iteration, order 1, is made up of 3 smaller triangles. And order 2 is made up of 9 triangles. So each iteration of the fractal has 3 times as many triangles, and **N=3**. Next we need to figure out the scaling factor, **r**. How much smaller is each triangle in order 1 than order 0? Look at the edge of each triangle in order 1, and you can see that the edge of each triangle is half the length of the edge of the triangle in order 0. So the scaling factor **r=2**.

That's all we need to know, and we can find the dimension by using the formula:

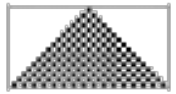
$$D = \frac{\log(N)}{\log(r)} = \frac{\log(3)}{\log(2)} = 1.585$$

http://fractalfoundation.org/OFC/OFC-10-3.html

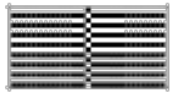

rule 0
rule 1
rule 2
rule 3
rule 4
rule 5
rule 6
rule 7
rule 8
rule 9
rule 10
rule 11
rule 12
rule 13
rule 14
rule 15
rule 16
rule 17
rule 18
rule 19
rule 20
rule 21
rule 22
rule 23
rule 24
rule 25
rule 26
rule 27
rule 28
rule 29
rule 30
rule 31
rule 32
rule 33
rule 34
rule 35
rule 36
rule 37
rule 38
rule 39
rule 40
rule 41
rule 42
rule 43
rule 44
rule 45
rule 46
rule 47
rule 48
rule 49

Source: NKS

rule 50

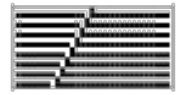


rule 51

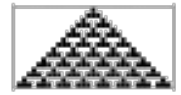


rule 52

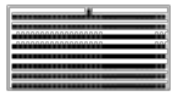
rule 53



rule 54

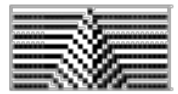


rule 55

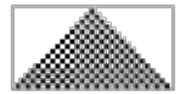


rule 56

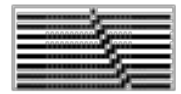
rule 57



rule 58



rule 59

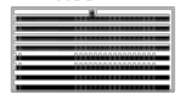


rule 60

rule 61

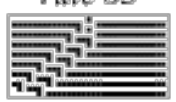
rule 62

rule 63

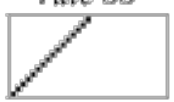


rule 64

rule 65



rule 66

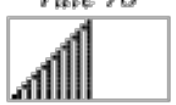


rule 67

rule 68

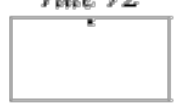
rule 69

rule 70

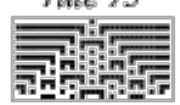


rule 71

rule 72



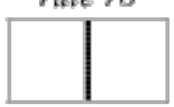
rule 73



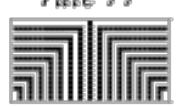
rule 74

rule 75

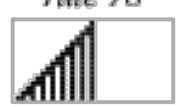
rule 76



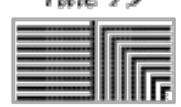
rule 77



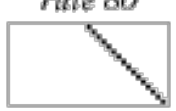
rule 78



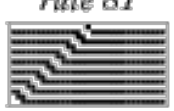
rule 79



rule 80



rule 81



rule 82

rule 83

rule 84

rule 85

rule 86

rule 87

rule 88

rule 89

rule 90

rule 91

rule 92

rule 93

rule 94

rule 95

rule 96

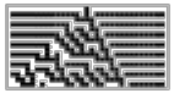
rule 97

rule 98

rule 99

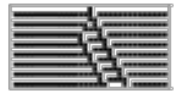
rule 100

rule 101



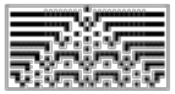
rule 102

rule 103



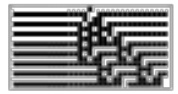
rule 104

rule 105



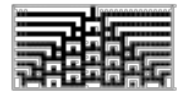
rule 106

rule 107



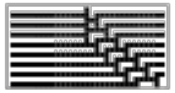
rule 108

rule 109

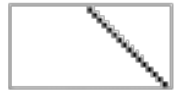


rule 110

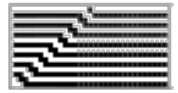
rule 111



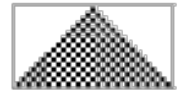
rule 112



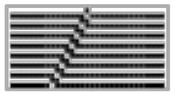
rule 113



rule 114

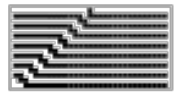


rule 115



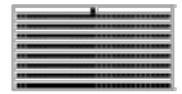
rule 116

rule 117



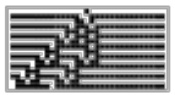
rule 118

rule 119

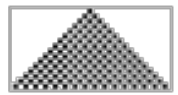


rule 120

rule 121



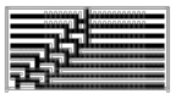
rule 122



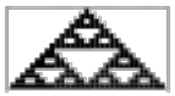
rule 123

rule 124

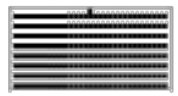
rule 125



rule 126



rule 127



rule 128

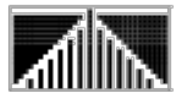
rule 129

rule 130

rule 131

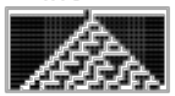
rule 132

rule 133



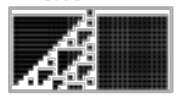
rule 134

rule 135



rule 136

rule 137



rule 138

rule 139

rule 140

rule 141

rule 142

rule 143

rule 144

rule 145

rule 146

rule 147

rule 148

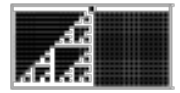
rule 149

rule 150

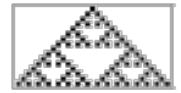
rule 151

rule 152

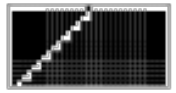
rule 153



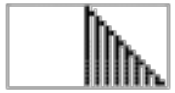
rule 154



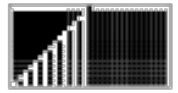
rule 155



rule 156



rule 157



rule 158

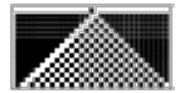
rule 159

rule 160

rule 161

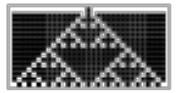
rule 162

rule 163



rule 164

rule 165



rule 166

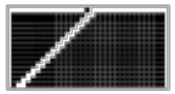
rule 167

rule 168

rule 169

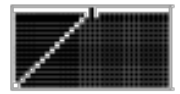
rule 170

rule 171



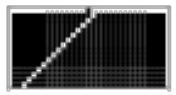
rule 172

rule 173



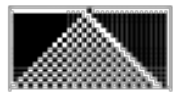
rule 174

rule 175

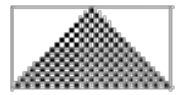


rule 176

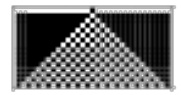
rule 177



rule 178



rule 179



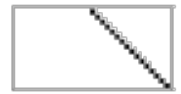
rule 180

rule 181

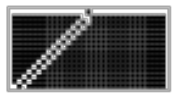
rule 182

rule 183

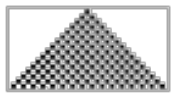
rule 184



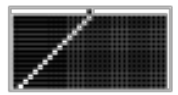
rule 185



rule 186



rule 187



rule 188

rule 189

rule 190

rule 191

rule 192

rule 193

rule 194

rule 195

rule 196

rule 197

rule 198

rule 199

rule 200

rule 201

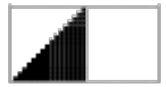
rule 202

rule 203

rule 204

rule 205

rule 206

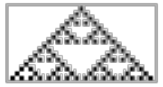


rule 207

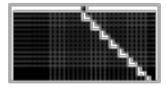
rule 208

rule 209

rule 210

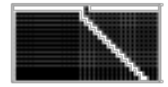


rule 211

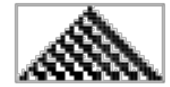


rule 212

rule 213



rule 214

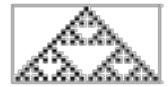


rule 215

rule 216

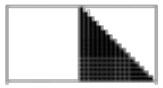
rule 217

rule 218

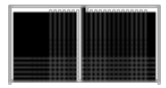


rule 219

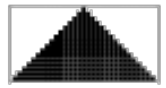
rule 220



rule 221



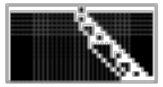
rule 222



rule 223

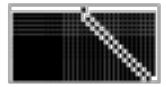
rule 224

rule 225



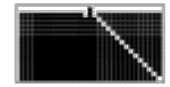
rule 226

rule 227



rule 228

rule 229



rule 230

rule 231

rule 232

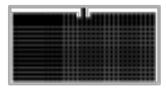
rule 233

rule 234

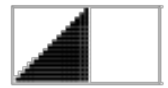
rule 235

rule 236

rule 237



rule 238



rule 239

rule 240

rule 241

rule 242

rule 243

rule 244

rule 245

rule 246

rule 247

rule 248

rule 249

rule 250

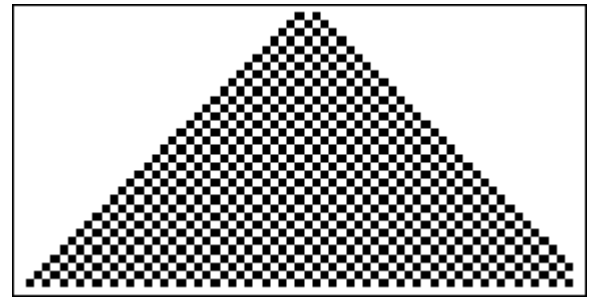
rule 251

rule 252

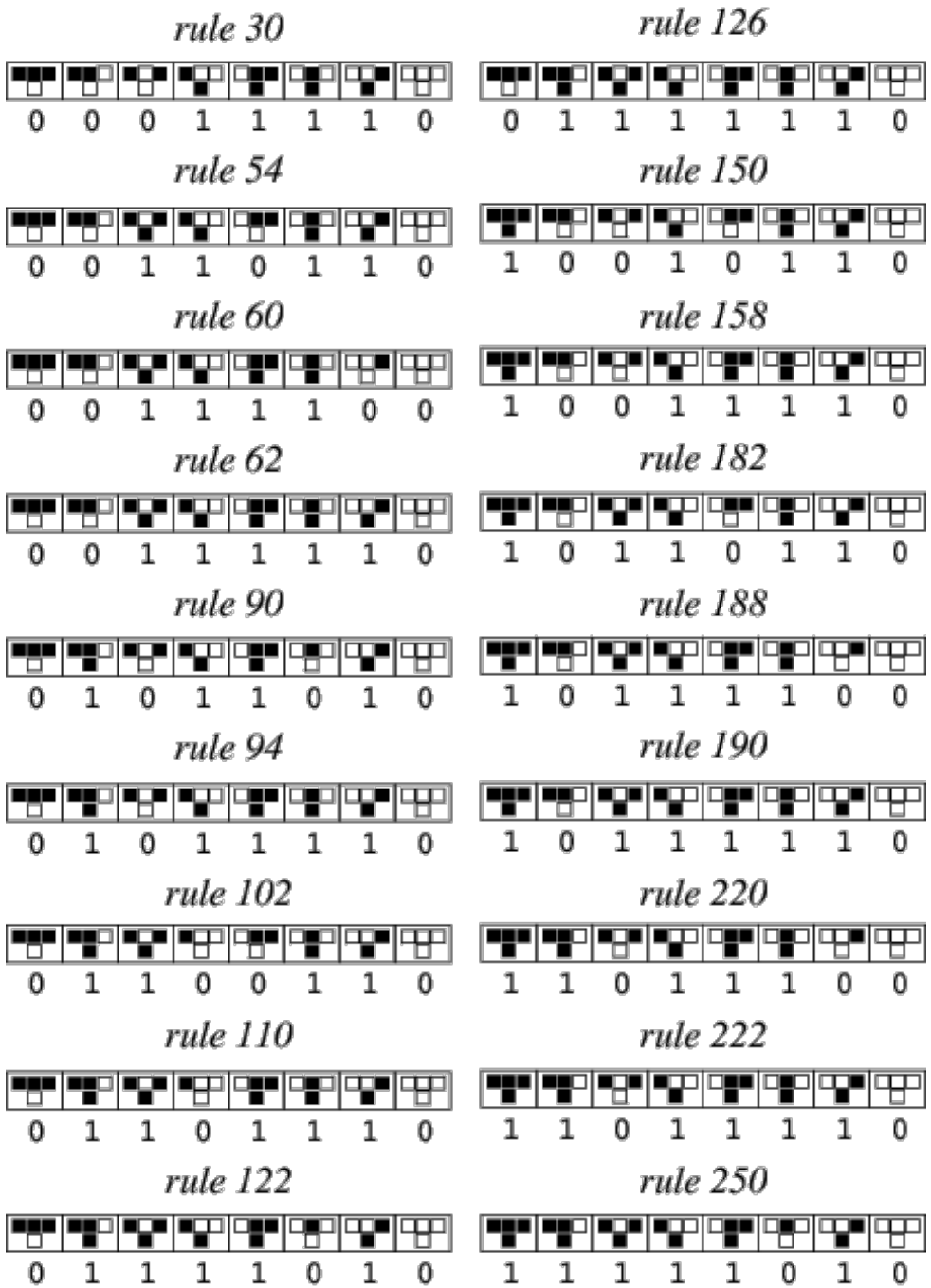
rule 253

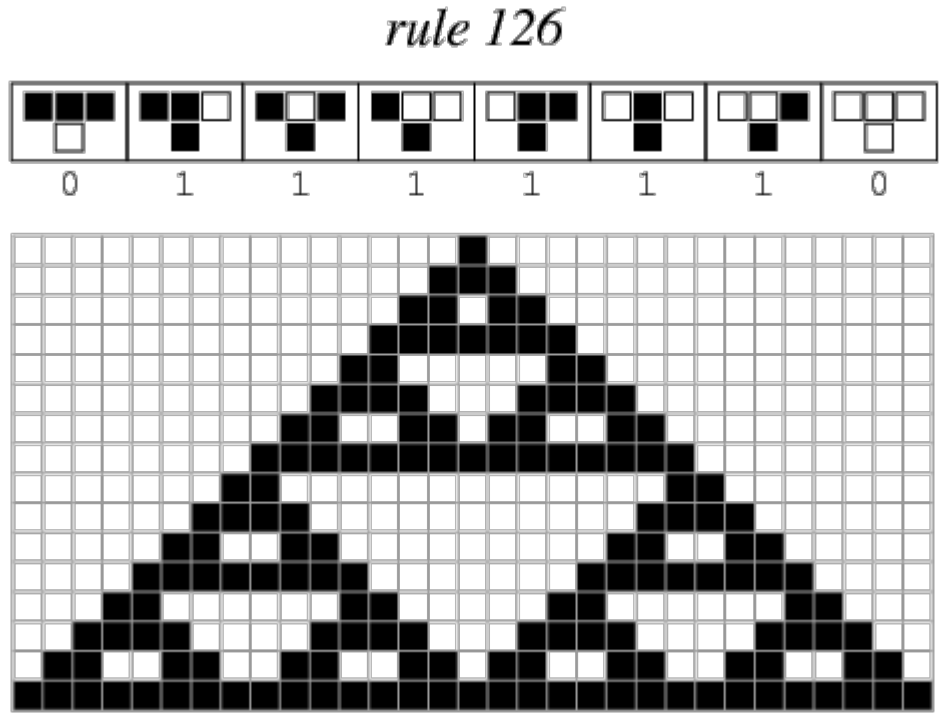
rule 254

rule 255



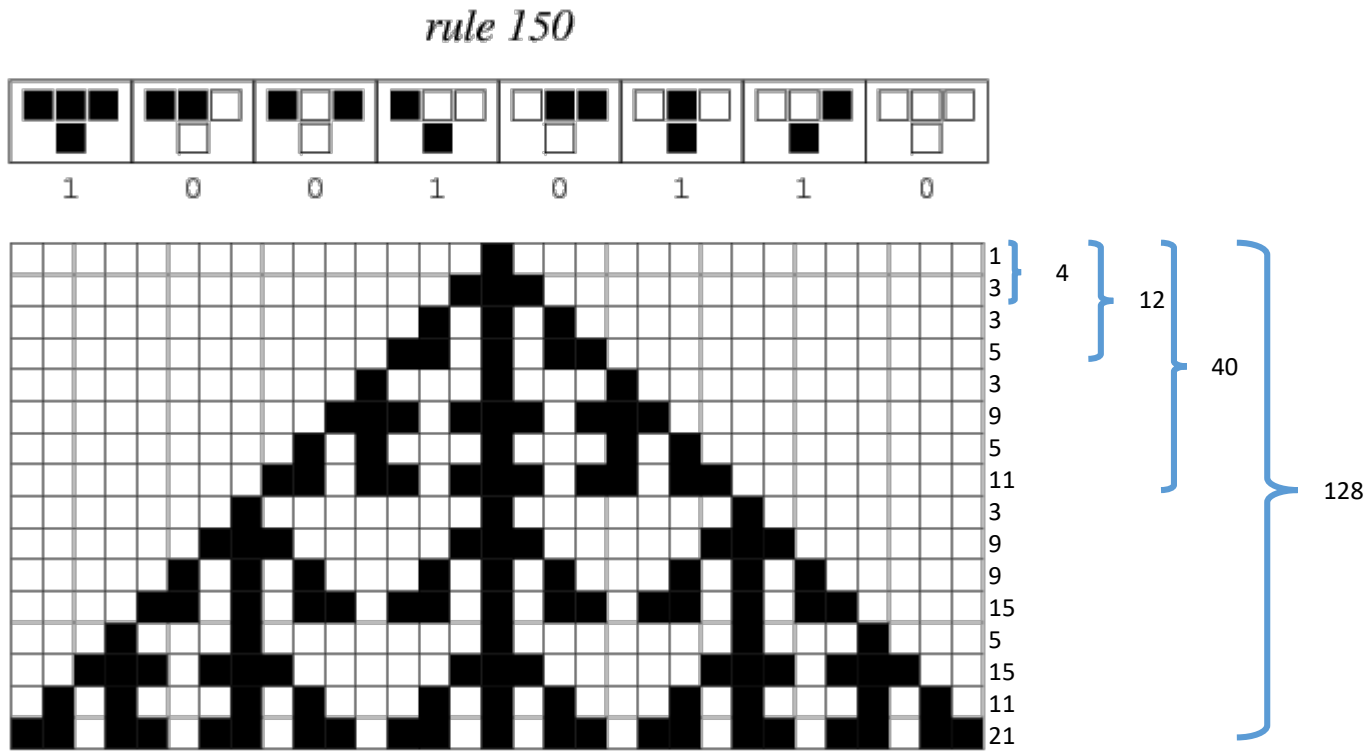
Rule 250 This one is additive, see
http://mathworld.wolfram.com/AdditiveCellularAutomaton.html





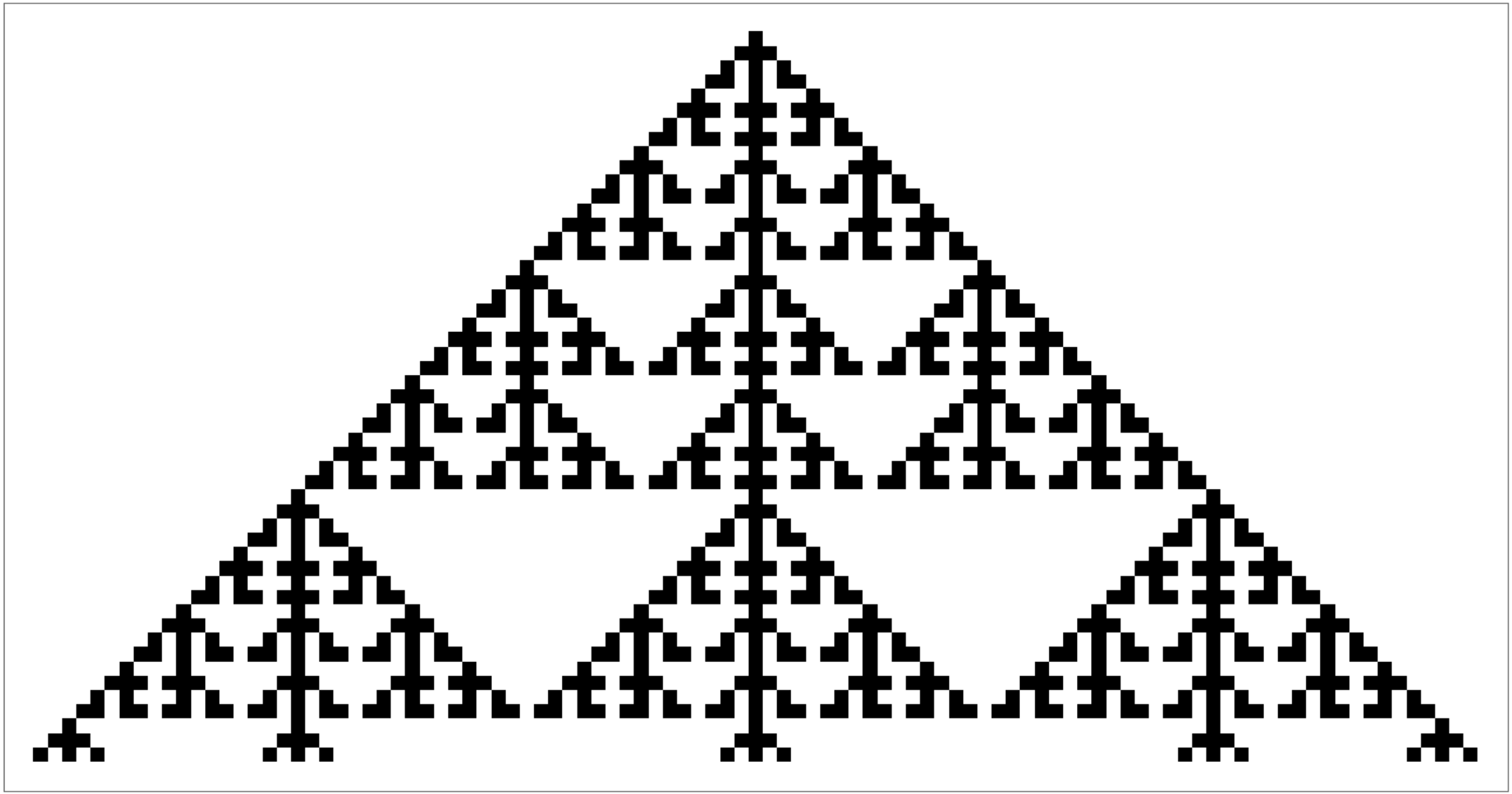
Also Rule 90 is “additive”, check Mathworld: http://mathworld.wolfram.com/AdditiveCellularAutomaton.html

source: NKS



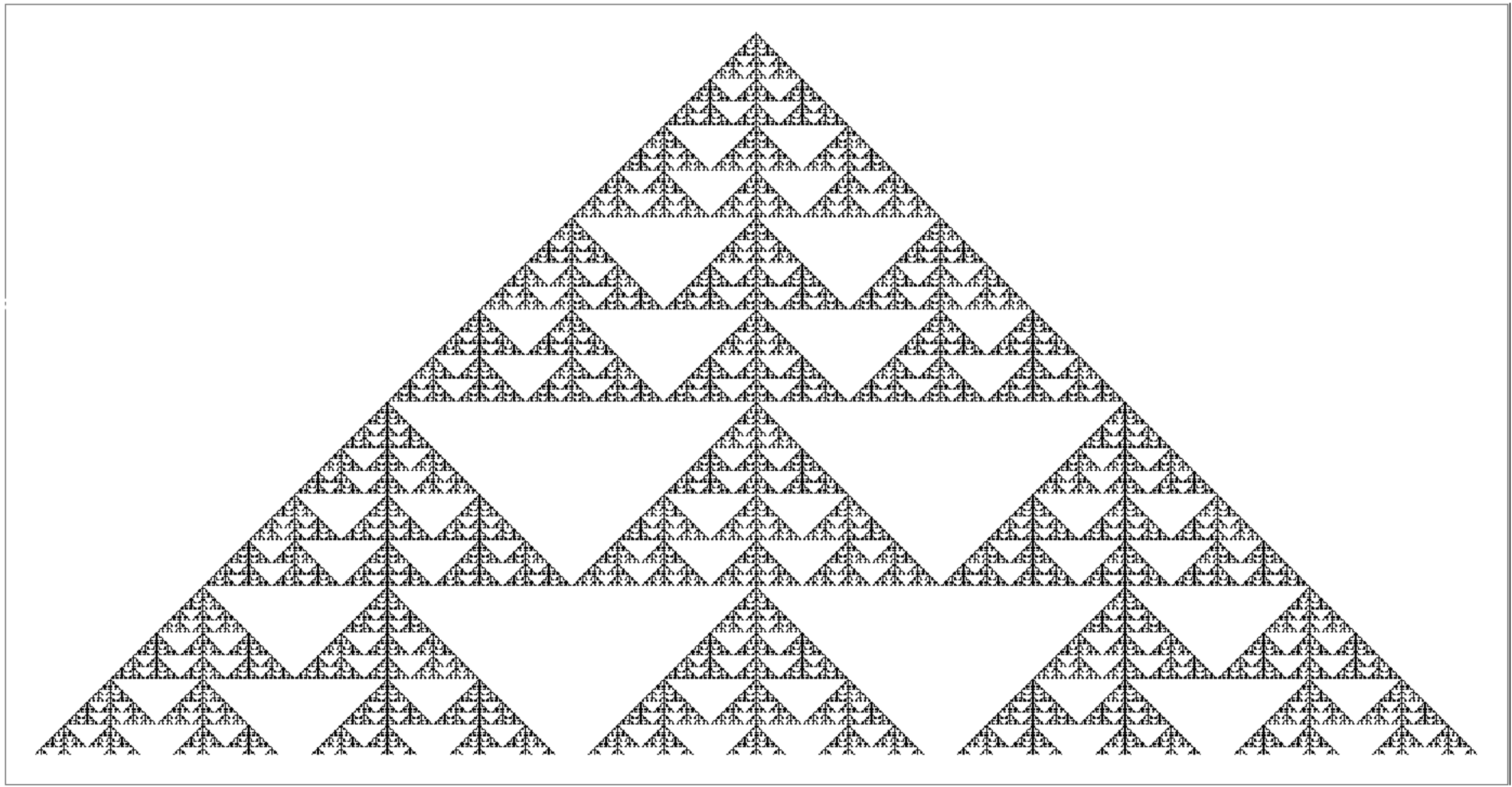
D=1 would give 2, 4, 8, 16, …
D=2 would give 2×2, 4×4, 8×8, 16×16, …
Now we have 2×2, 3×4, 5×8, 8x16, …
Fibonacci numbers: 2,3,5,8,13,21,34, etc.
Limit $F_{n+1}/F_n = (1+\sqrt{5})/2$
This rule has fractal dimension $\log(1+\sqrt{5}) / \log 2 \approx 1.69$

In[81]:=
```
M = 50;
c = CellularAutomaton[150, {{1}, 0}, M];
ArrayPlot[c]
```

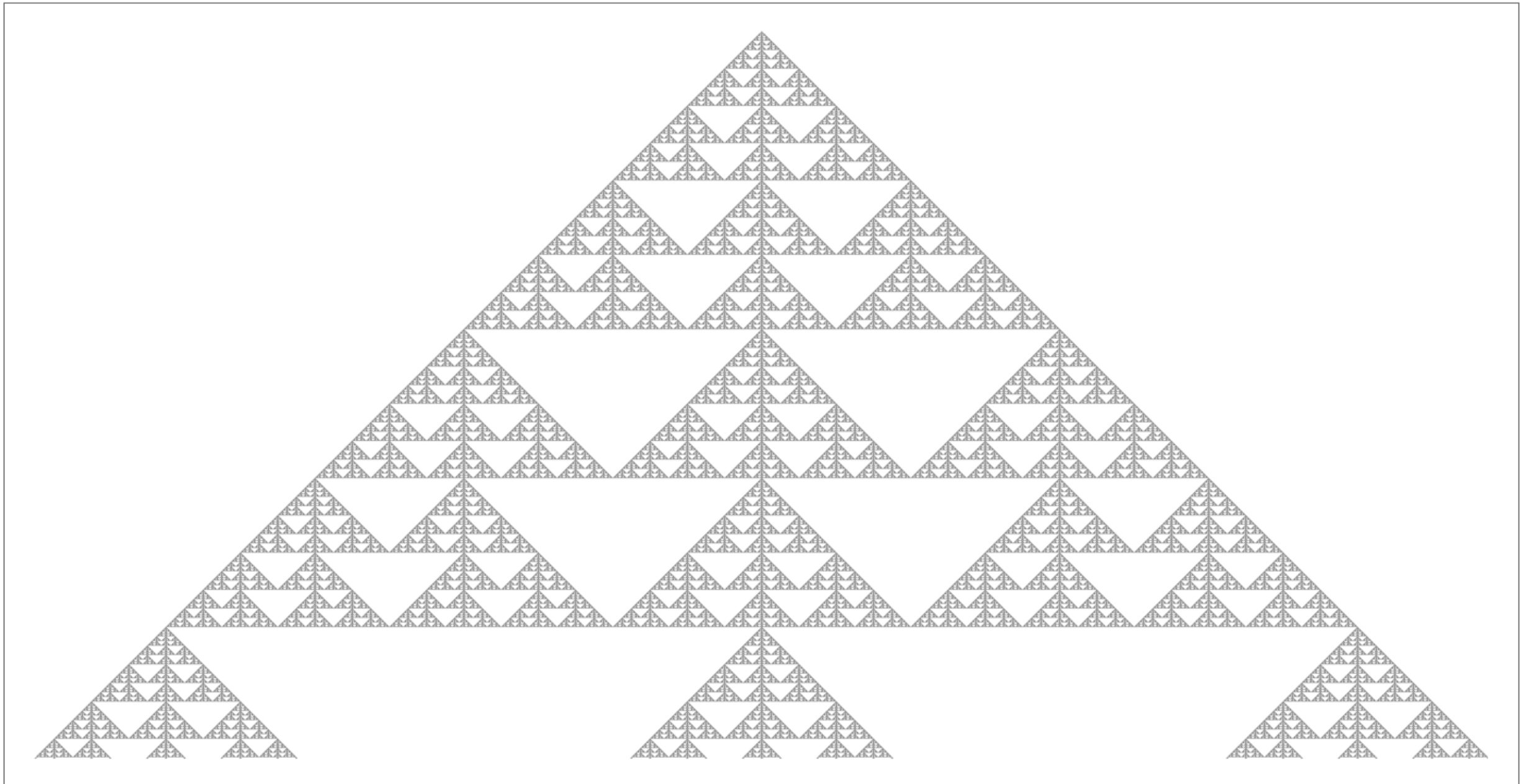


In[78]:=
```
M = 500;
c = CellularAutomaton[150, {{1}, 0}, M];
ArrayPlot[c]
```

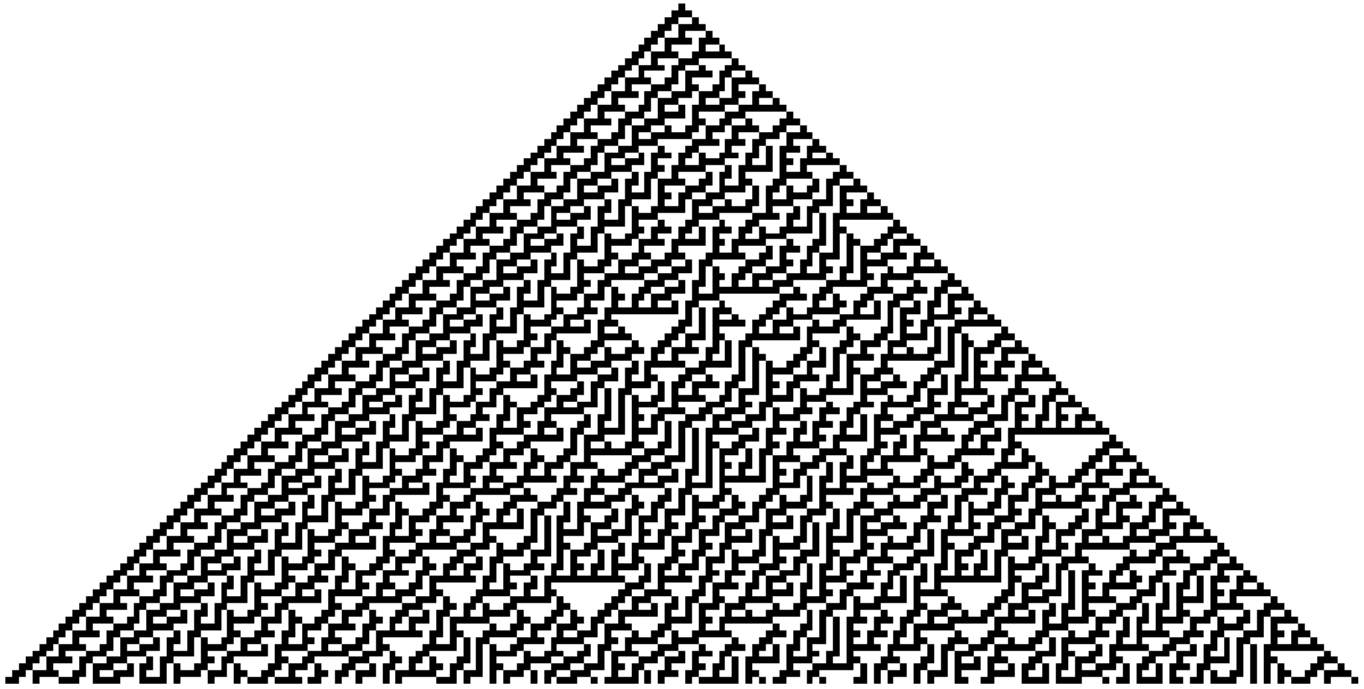
Out[80]=



In[75]:=
```
M = 5000;
c = CellularAutomaton[150, {{1}, 0}, M];
ArrayPlot[c]
```



Rule 30 is chaotic, like a random generator



Conus textile by Richard Ling, Wikimedia

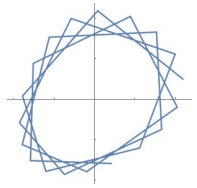
An empirical observation is that cellular automata can be classied according to the complexity and information produced by the behavior of the pattern they produce:

- Class 1 : Fixed; all cells converge to a constant black or white set
- Class 2 : Periodic; repeats the same pattern, like a loop
- Class 3 : Chaotic; pseudo-random
- Class 4 : Complex local structures; exhibits behaviors of both class 2 and class 3; with long lived hard to classify structure.

Source: http://www2.math.uu.se/~david/MCS/lecture3.pdf by David_Sumpter

Wolfram’s one-dimensional automata: conclusions

- Some of the automata have complex behavior
- Some rules are perfect random generators (automaton rule 30)
- Even a universal computer can be simulated (automaton rule 110)
- Wolfram believes our whole universe could be just one big cellular automaton



Cellular automata: research and applications

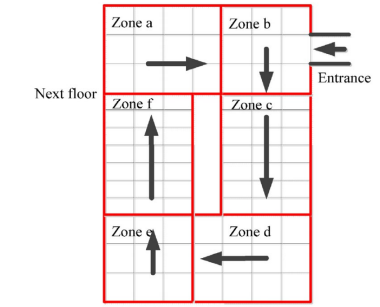
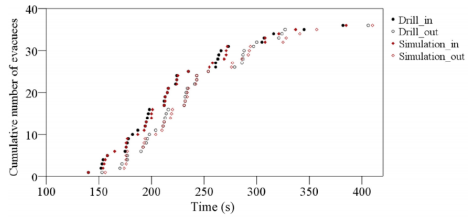


Fig. 1 Grid map and six zones



(a) Cumulative number of evacuees between floors 7 and 5

Fig. 12 Drill and simulation data between floors 7 and 5

- Fundamentals of computation
  - ✓ Von Neumann, Conway, Wolfram
- Models of social behavior
  - ✓ Beltran, F. S., Herrando, S., Estreder, V., Ferreres, D., Adell, M. A., & Ruiz-Soler, M. (2011). Social simulation based on cellular automata: Modeling language shifts. In Cellular Automata—Simplicity Behind Complexity (p. 323). InTech.
  - ✓ Lu, Y., Laffan, S., Pettit, C., & Cao, M. (2020). Land use change simulation and analysis using a vector cellular automata (CA) model: A case study of Ipswich City, Queensland, Australia. Environment and Planning B: Urban Analytics and City Science, 47(9), 1605-1621.
  - ✓ Ding, N., Chen, T., & Zhang, H. (2017, June). Simulation of high-rise building evacuation considering fatigue factor based on cellular automata: A case study in China. In Building Simulation (Vol. 10, No. 3, pp. 407-418). Tsinghua University Press.
- Aesthetics and awareness in design
  - ✓ Lukas, Troy, Loe -> next time