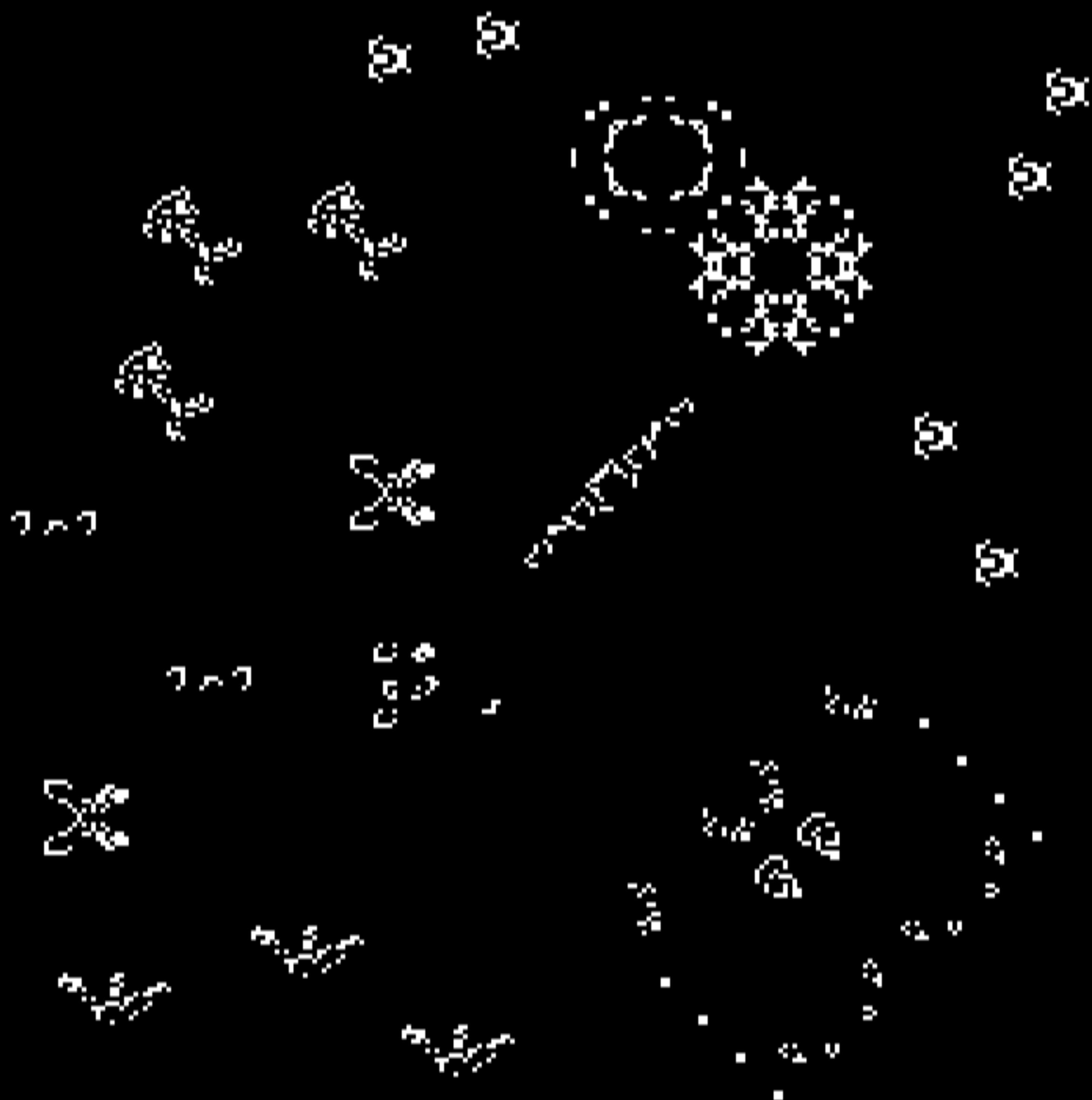
Cellular Automata
Part Two

Two Dimensional CA

- last class we looked at one-dimensional CA
- in particular, elementary cellular automata
- they have proved useful in the study of complexity
- as Rule 110 proved, they can be capable of powerful computation
- but if we want to see lifelike behaviours in CA then we should move on to two-dimensional CA
- and there's no better place to start than the Game of Life…



the game of life

Game of Life

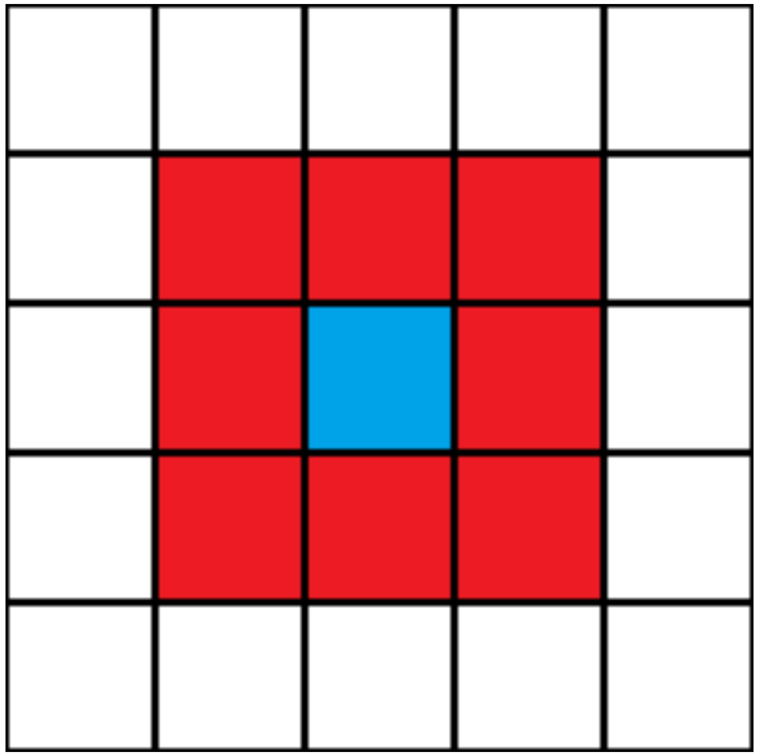
- (or simply 'Life')
- invented by John Conway in 1970
- aim: to define an interesting and unpredictable CA
- rules chosen to meet these criteria:
    1. there should be no explosive growth
    2. there should exist small initial patterns with chaotic, unpredictable outcomes
    3. there should be potential for von Neumann universal constructors (more on this later)
    4. the rules should be as simple as possible
- in effect: it should be a complex system

Game of Life

- was the aim (and criteria) met?
- yes!
- also later proven that Life fulfils John von Neumann's definition of life:
    - an organism is alive if it can
        1. reproduce itself
        2. simulate a Turing Machine
- is von Neumann's definition of life controversial?
- definitely! (although finding a consensus definition for life is proving trickier than we thought)
- but, while nobody would seriously consider the Game of Life to be alive, it does provide a good example of emergence and self-organization

Game of Life: States & Neighbourhood

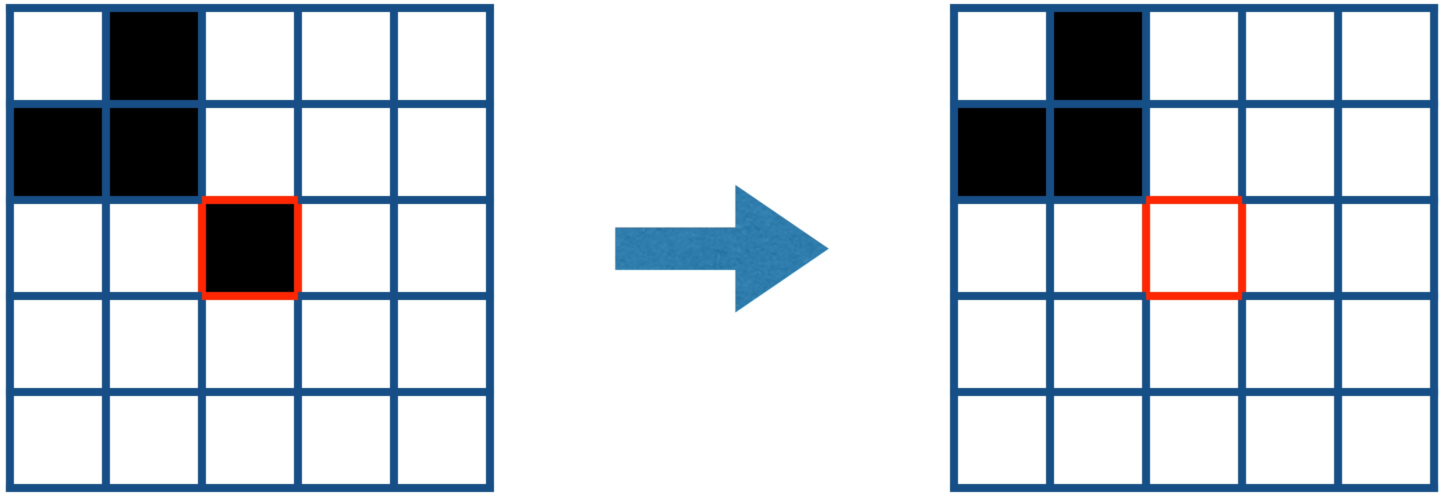
- two cell states:
  - 0 = dead
  - 1 = alive
- uses a Moore neighbourhood:



Game of Life: Rules

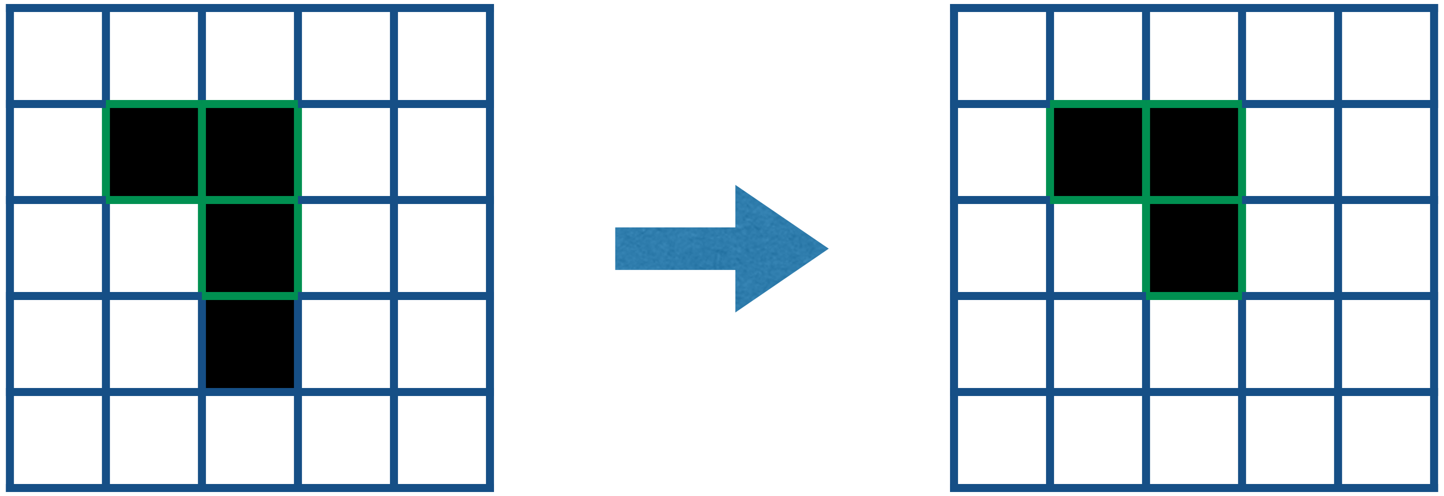
- at each time step:
    1. any live cell with fewer than two live neighbours dies, as if by underpopulation
    2. any live cell with two or three live neighbours lives on to the next generation
    3. any live cell with more than three live neighbours dies, as if by overpopulation
    4. any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction

Game of Life: Underpopulation



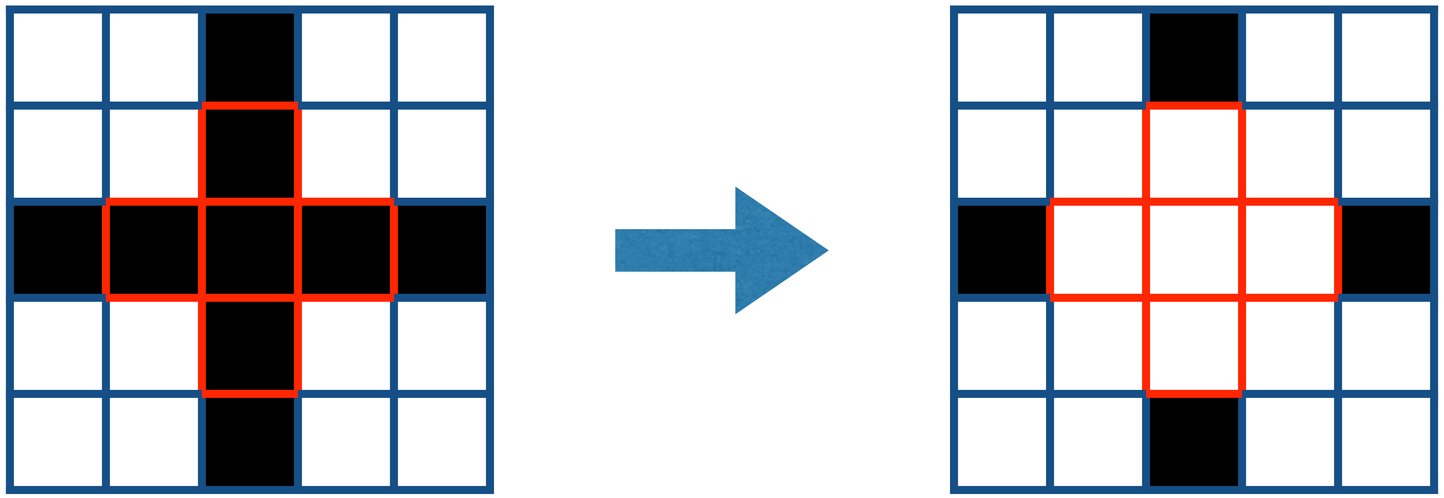
any live cell with fewer than two live neighbours dies,
as if by underpopulation

Game of Life: Survival



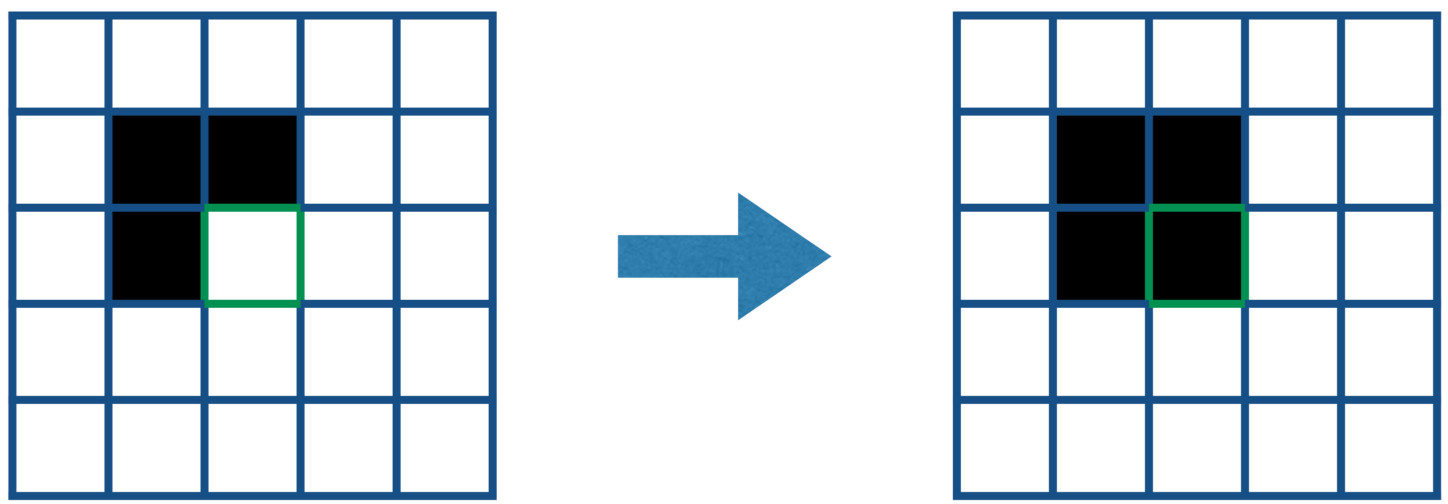
any live cell with two or three live neighbours lives on to the next generation

Game of Life: Overpopulation



any live cell with more than three live neighbours dies, as if by overpopulation

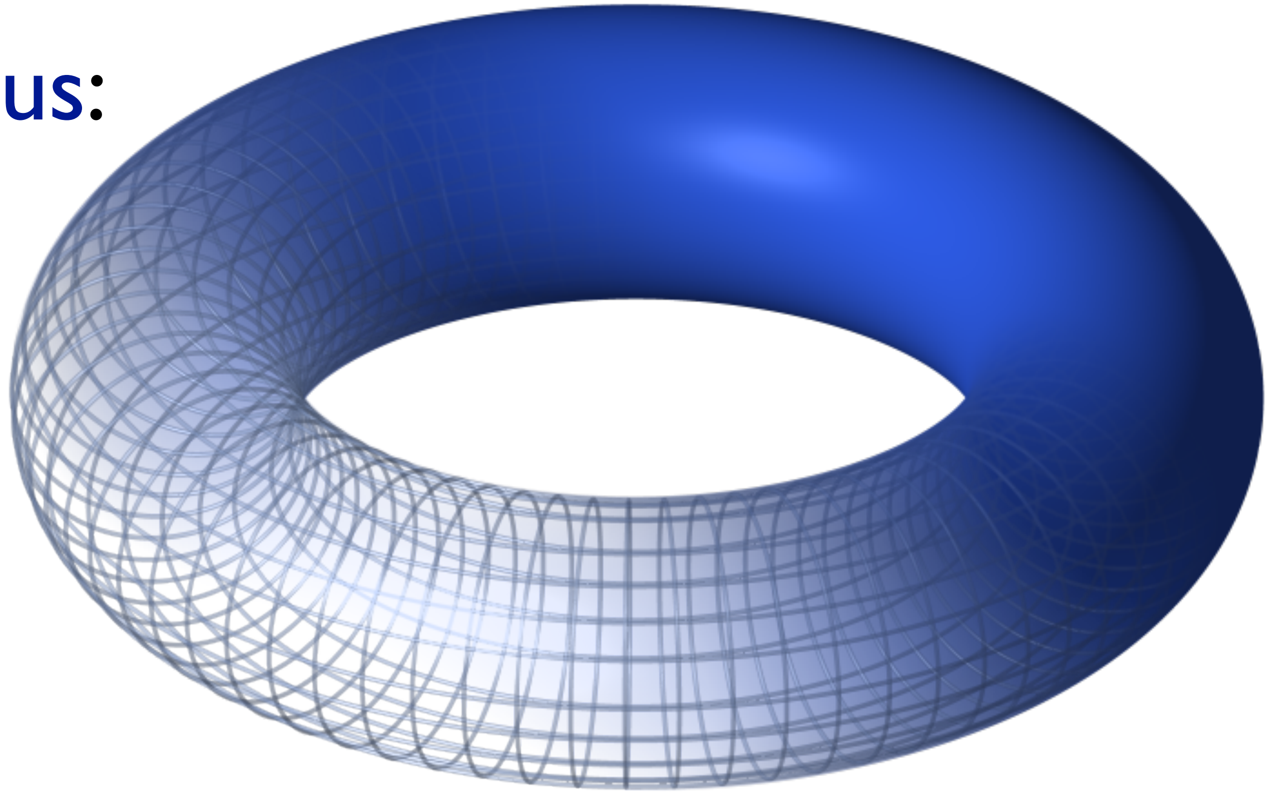
Game of Life: Reproduction



any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction

Game of Life: Topology

- the 2D grid used by Life wraps around, so that:
  - the topmost row of cells neighbours the bottommost row of cells
  - the leftmost column of cells neighbours the rightmost column of cells
- effectively forming a torus:



(this is very common for 2D CA)

Game of Life: Patterns to Look For

- stills (these never change)

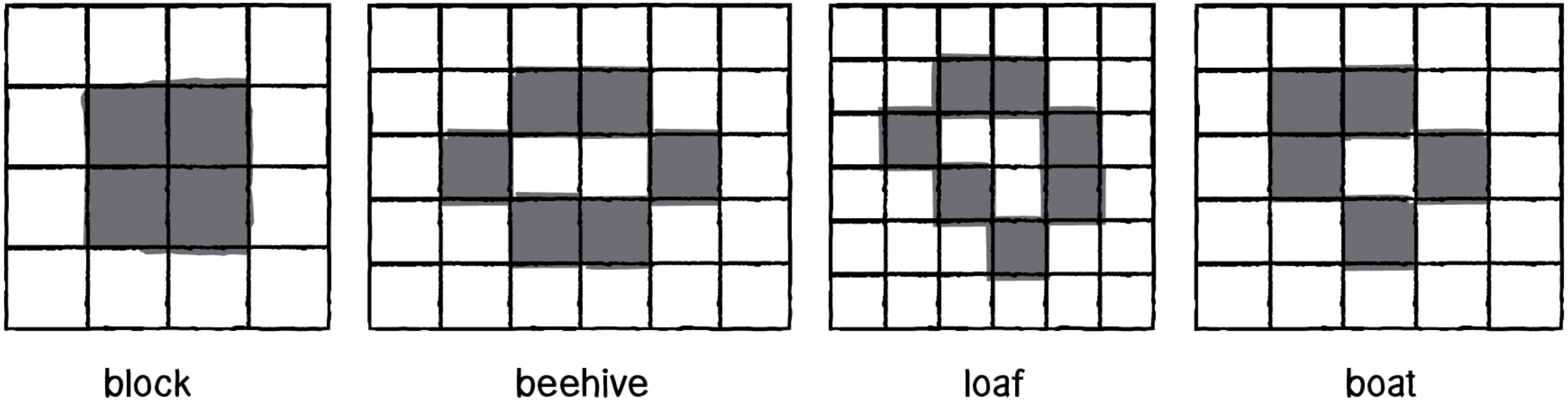


figure 7.24 from The Nature of Code

Game of Life: Patterns to Look For

- oscillators (between two states)

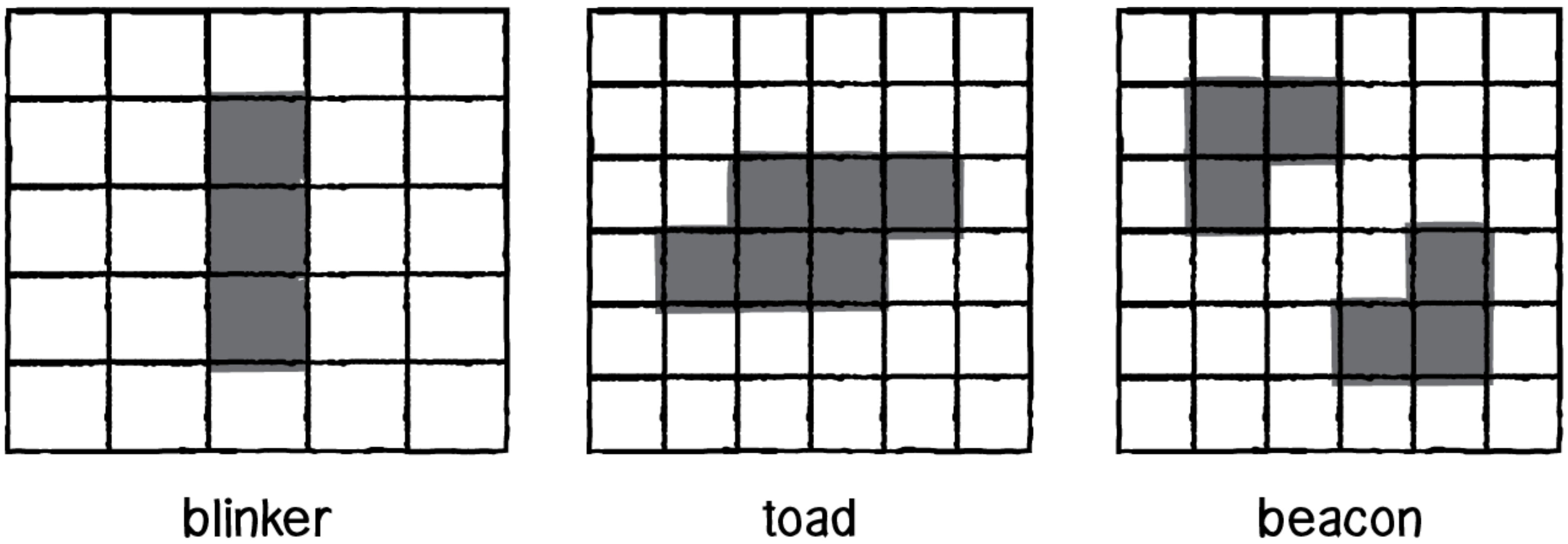


figure 7.25 from The Nature of Code

Game of Life: Patterns to Look For

- spaceships (move about the grid)

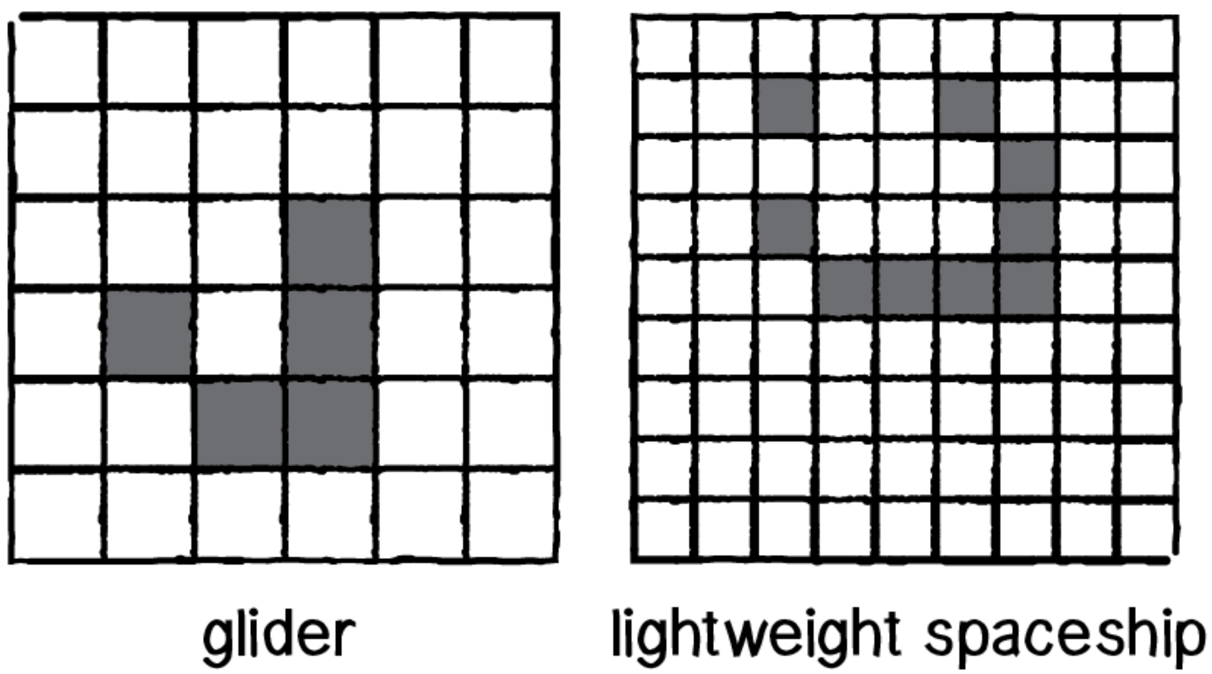


figure 7.26 from The Nature of Code

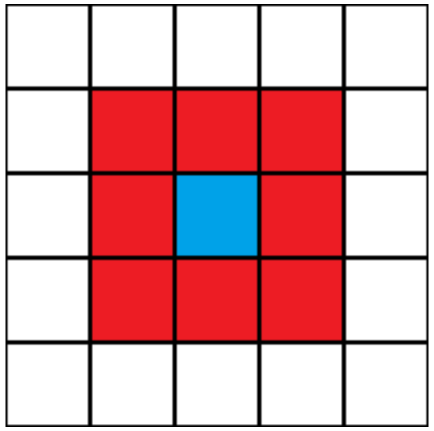
- see the wikipedia page for animations

So Many Possibilities

- so yet again, we've seen that from a very simple set of rules
- and with a very simple binary-state automaton
- that complex behaviour can emerge
- so it probably won't surprise you to learn that Life is capable of universal computation
- which also means that it's undecidable:
    - given an initial pattern and a later pattern, no algorithm exists that can tell whether the later pattern is ever going to appear
- but what might surprise you is how many possible alternatives to Life can be built…

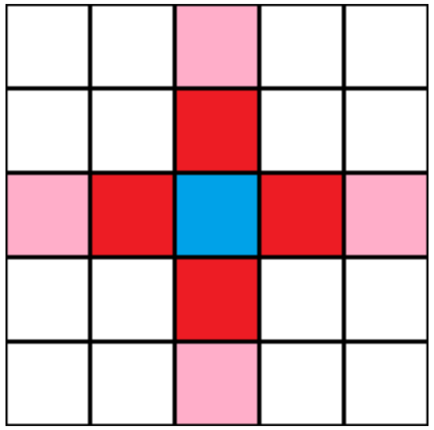
Neighbourhoods

- recall that Life uses a Moore neighbourhood



beginning in the top left, and moving clockwise, we can label these cells NW, N, NE, and so on after the compass points, with C for the centre cell

- (the other common type is the von Neumann neighbourhood)



Rule Table

- Life has 4 written 'rules', which are easily implemented using conditional code
- but, (similar to Elementary CA), we could implement the rules by creating a rule table, which defines the current value of a cell for every possible neighbourhood of previous states:

neighbourhood									new state
NW	N	NE	E	SE	S	SW	W	C	
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	0	0	0	0	0	**1**
...									
1	1	1	1	1	1	1	1	1	**0**

Rule Table

- with a Moore neighbourhood, if there are just 2 states, then the rule table has $2^9 = 512$ rows
- but with 3 states, the rule table has $3^9 = 19683$ rows
- and in general, with k states, the table will have k^9 rows
  - a value that grows very quickly with k
- which makes manual entry of a full rule table impractical
- however, in practice we usually just manually define a small fraction of those rows
- and then set the remainder of the rows to either return a new state of 0, or to leave the state untouched between iterations

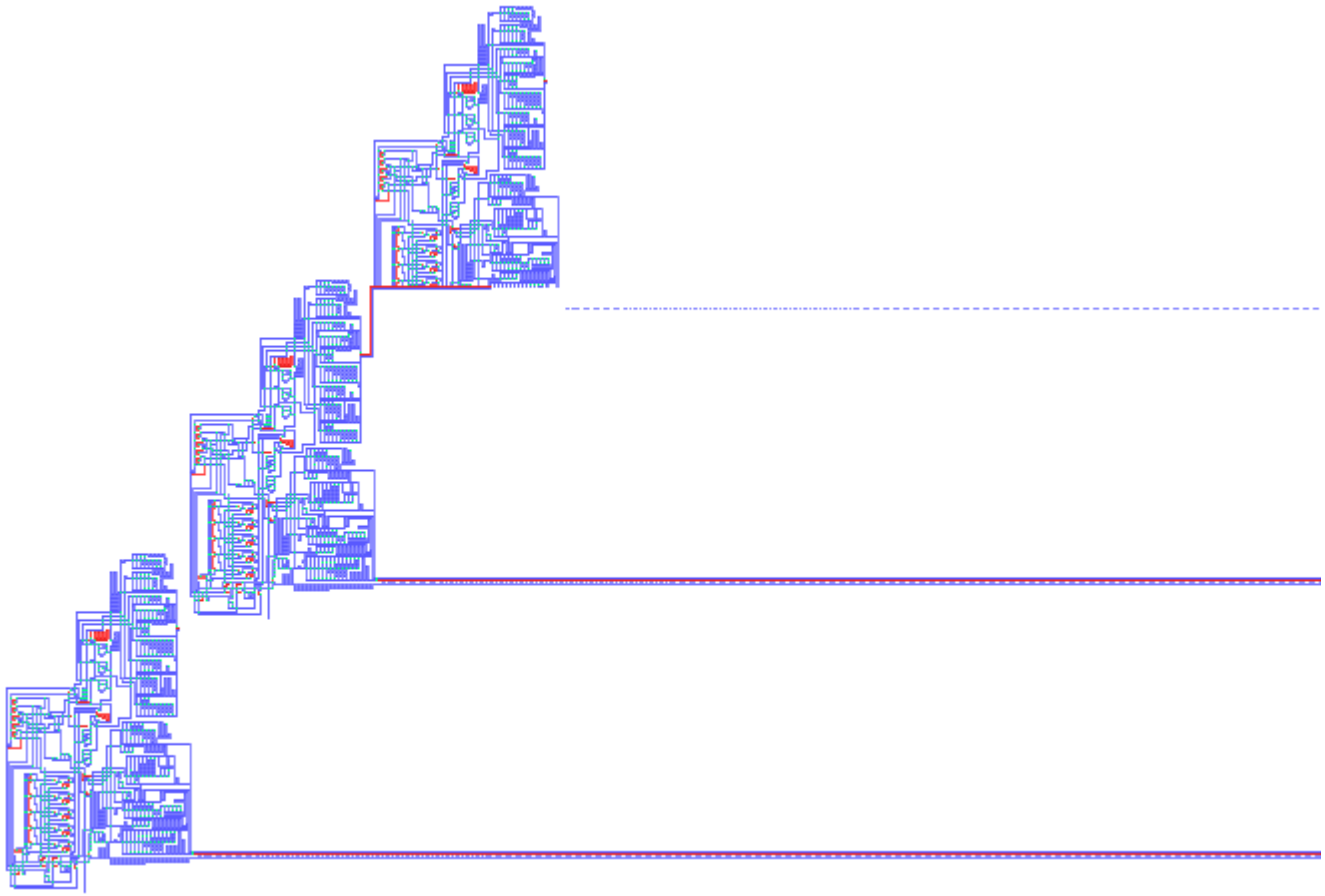
The Set of All Possible Rule Tables

- k^9 is the number of rows needed to define one particular rule table
- so in total there are k^{k^9} possible rule tables for a CA with a Moore neighbourhood
  - a huge number! (with just 2 states that gives 1.34×10^{154} possible rulesets)
- the majority of these will lead to uninteresting behaviour
  - such as rapid convergence to a steady state, or chaos
- but it leaves a lot of potential to discover interesting rulesets, or rulesets than can perform a useful function
  - perhaps by some kind of evolutionary search
  - more on this later in the course

Cellular Automata: A Potted History

- recall earlier that one of Conway's criteria for Life is that it has the potential for the presence of von Neumann universal constructors
- and that's where the story of cellular automata really begins…

von Neumann Universal Constructor



"What kind of logical organization is sufficient for an automaton to be able to reproduce itself?"

von Neumann Universal Constructor

- designed by John von Neumann in the 1940s
- on paper
  - because there was no computer to use!
- initially based on the idea of one robot building another robot
  - an impractical and costly idea
- his colleague Stanlislaw Ulam suggested using a discrete system for creating a reductionist model for self-replication
- an abstract machine which, when run, will replicate itself

von Neumann Universal Constructor

- 29-state, 2D cellular automaton
- contains a machine consisting of 3 parts:

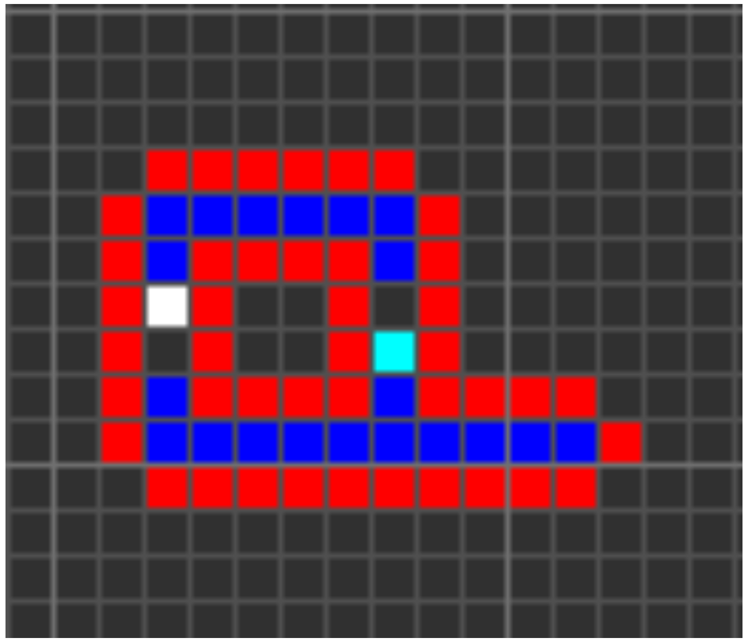
1. a blueprint for itself
2. a mechanism that can read any blueprint and construct the machine specified by that blueprint
3. a mechanism that can make copies of any blueprint

von Neumann Universal Constructor

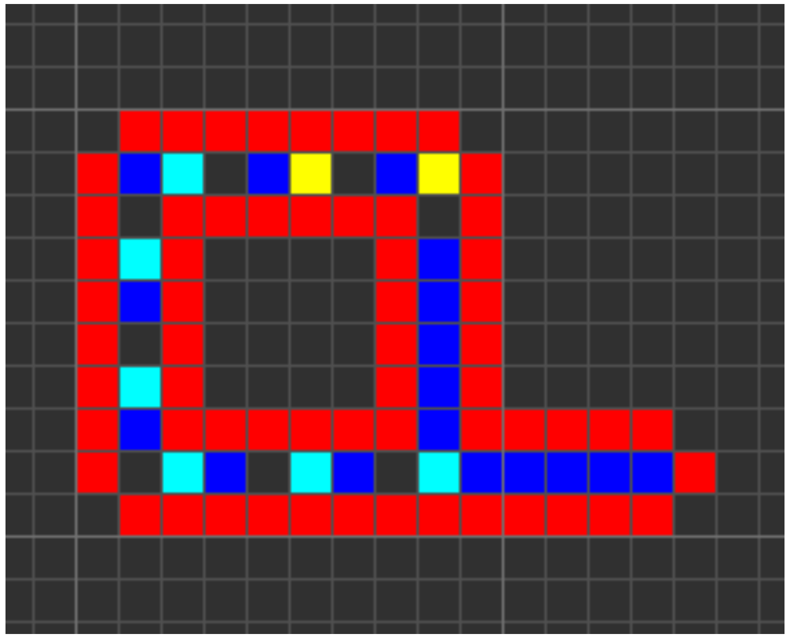
- a key insight is that the machine has two roles:
    - it plays an active role in the construction of a new copy
    - and is the target of the copying process
- compare this to biological DNA
- or to a computer virus
- add some random mutation in the copying process, and some selection pressure, and the Universal Constructor - in principle - has the potential for open-ended evolution
- however - in practice - it is far too fragile
    - most mutations would lead to disintegration

Following von Neumann

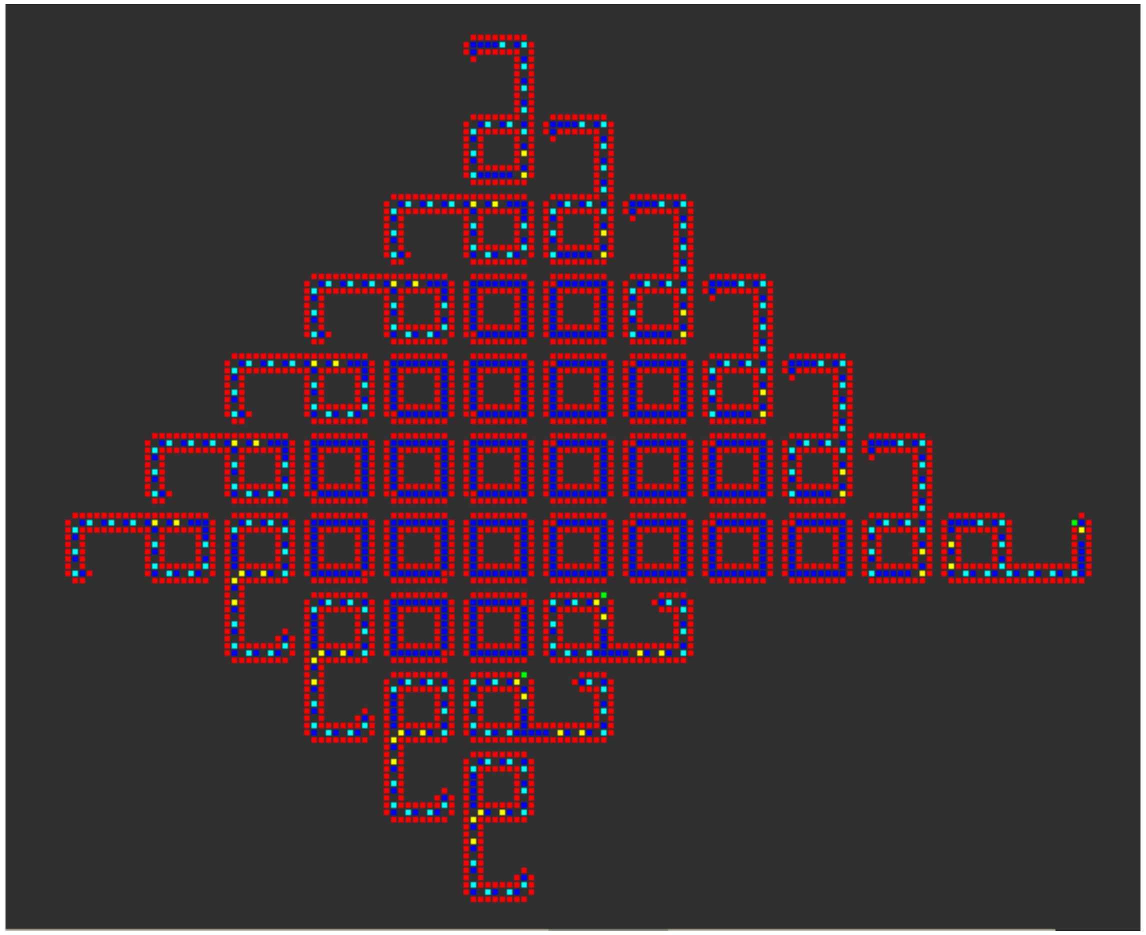
- CA recognised as a type of nonlinear dynamical system worthy of study
  - see Langton's paper *Computation at the Edge of Chaos* in Brightspace
- based on von Neumann's work, EF Codd produced an 8-state CA capable of self replication
  - however, a replicator in this CA requires 283,126,588 cells
- this led to a search for systems with smaller-sized replicators



Langton Loops



“70-70-70-70-70-70-40-40”

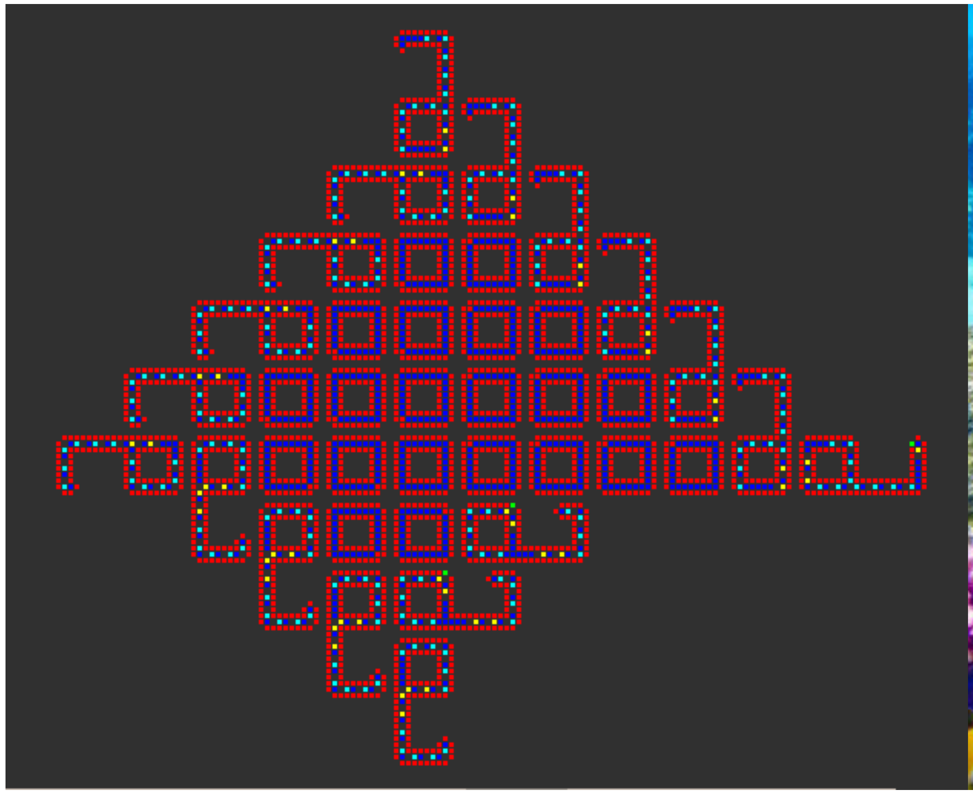


Langton Loops

- created in 1984 by Christopher Langton
- removed the universality condition of von Neumann and Codd's works
- was able to reduce the size of the replicator to just 86 cells
- like Codd's work, the 'instructions' or 'genome' to are encoded within a sheath
- these cause the arm ('pseudopod') to:
    - extend from the parent loop
    - turn four times to create a new loop
    - inject the genome into the daughter loop

Langton's Loop

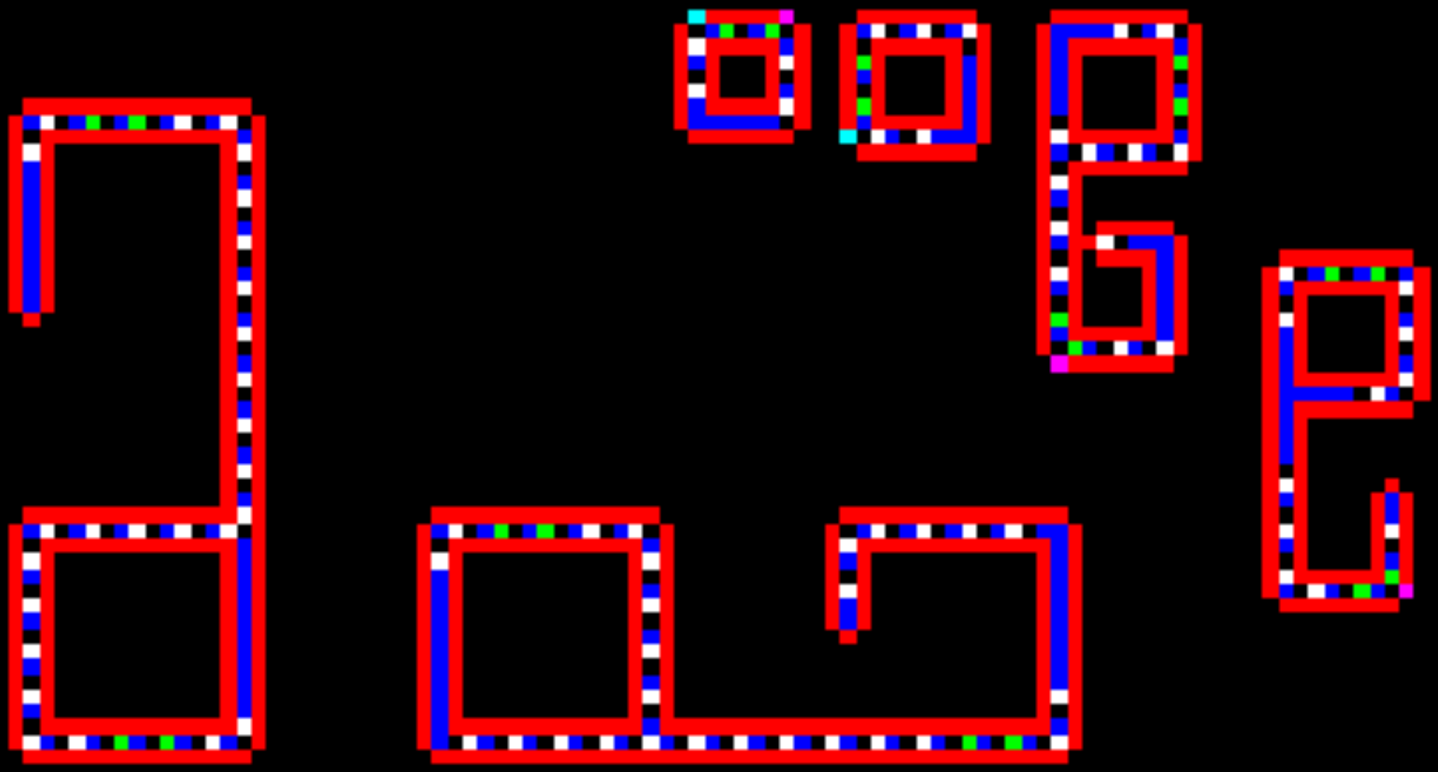
- a problem with Langton's loop is that when the arm encounters another loop it stops operating
- leaving 'dead' loops in the grid
- these tend to build up to form coral like colonies




SDSR Loop, Evoloop

- Hiroki Sayama solved this problem with the SDSR Loop:
  - "structure dissolving, self-replicating"
- a slight adjustment of Langton's Loop causes the entire loop to dissolve (go to the 'quiescent' state 0) when it collides with another loop
  - removing the 'dead' colony
- This was further improved with his Evoloop, which, allows loop size to mutate when a loop collides with another loop…

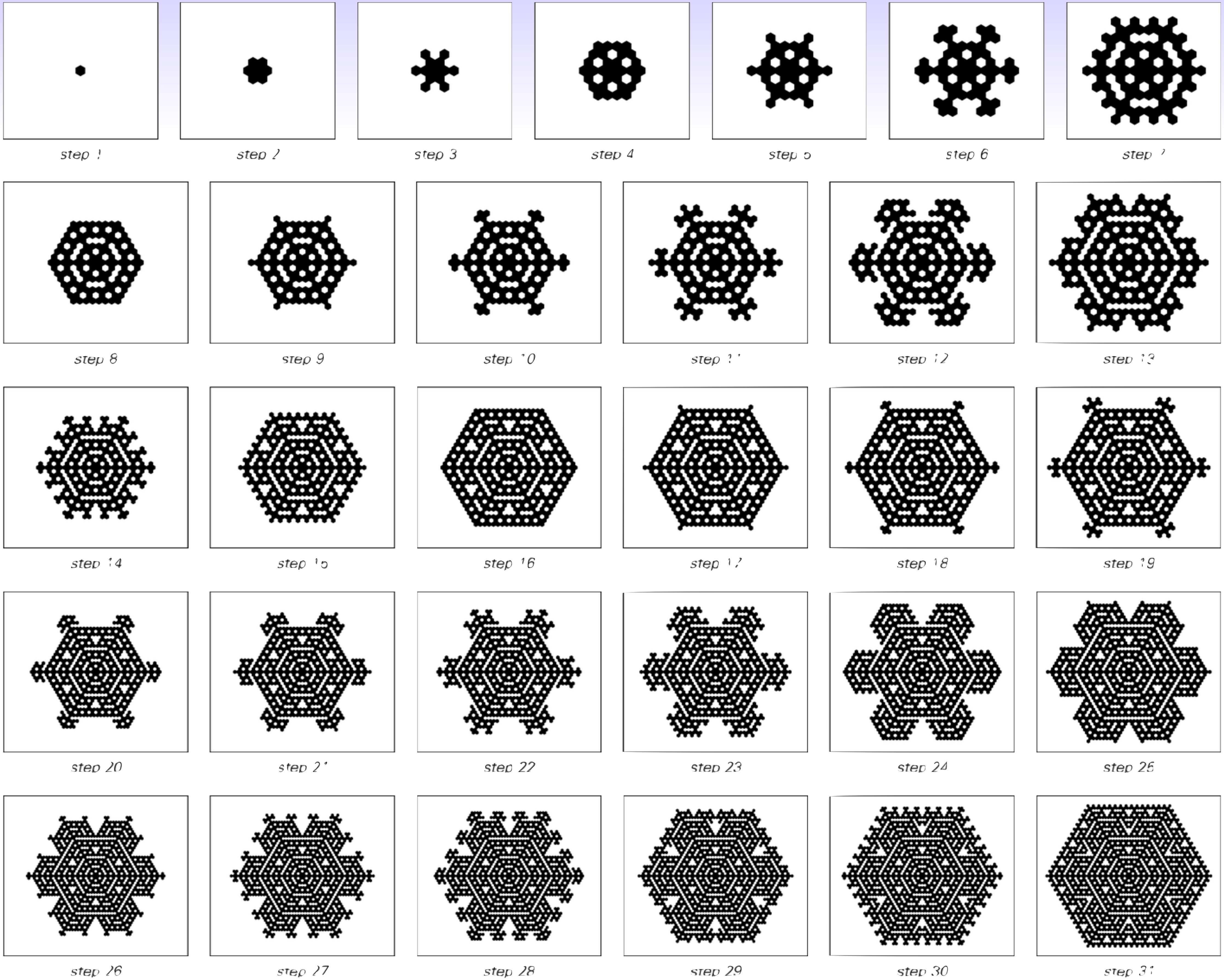
Evoloops



a brief guide

Other Types of Cellular Automata

- given the basic definition of a cellular automaton (see first set of slides), there are many different types that can and do exist
- such as hexagonal CA
- which can bear remarkable resemblance to a particular natural phenomenon…



A New Kind of Science, p371

Reading & References

- required reading:
  - The Game of Life in Scientific American
  - Elementary Cellular Automata at Wolfram Mathworld
- required tasks:
  - familiarize yourself with the course's Brightspace shell
  - download, try out and play with the code in RESOURCES > Cellular Automata
- highly recommended reading:
  - Cellular Automata in the Nature of Code
  - A New Kind of Science by Stephen Wolfram (free book!)
  - Conway's Game of Life Wikipedia entry
  - Evoloop by Hiroki Sayama
- cool stuff:
  - golly is a free-to-download Game of Life simulator - and more!