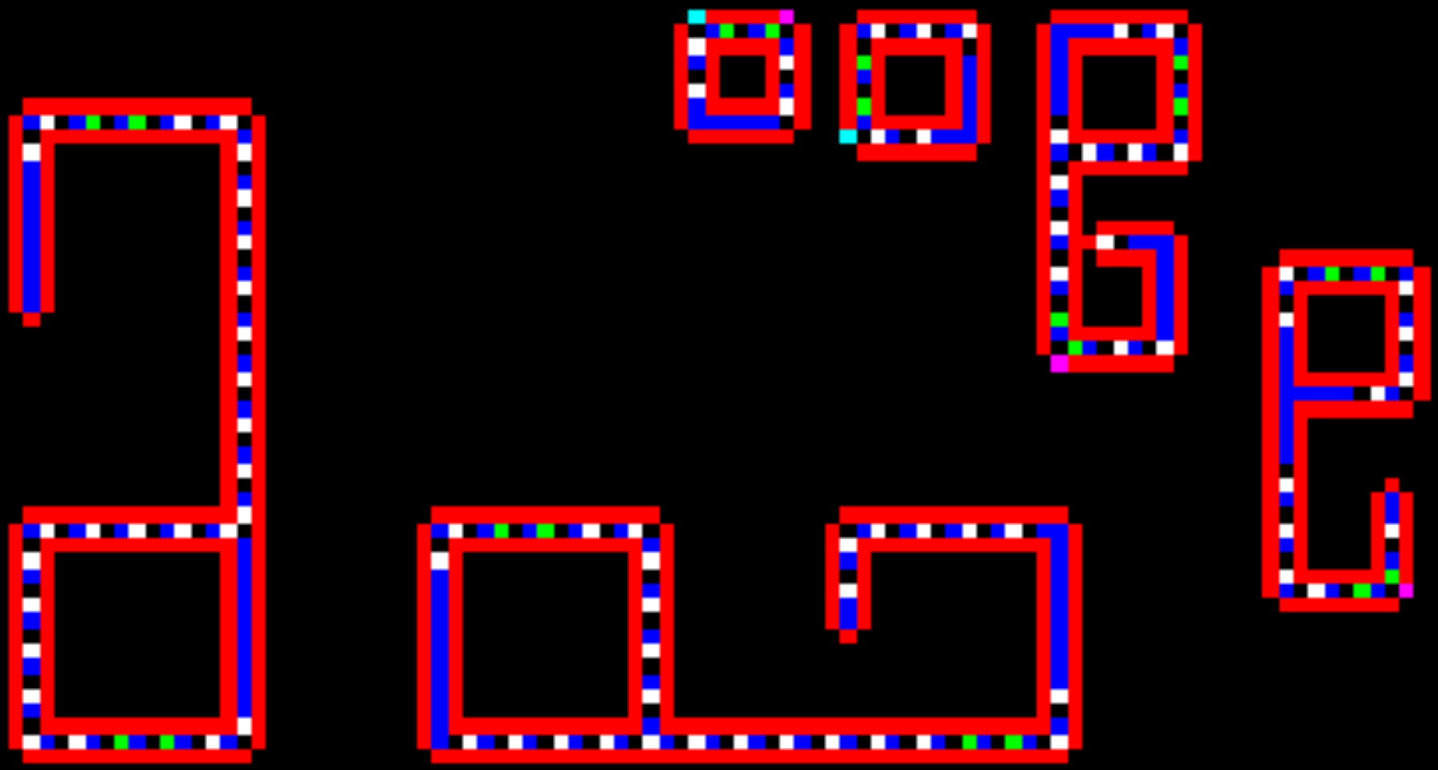
Cellular Automata
Part One



evoloops

A cellular automaton (CA) consists of…

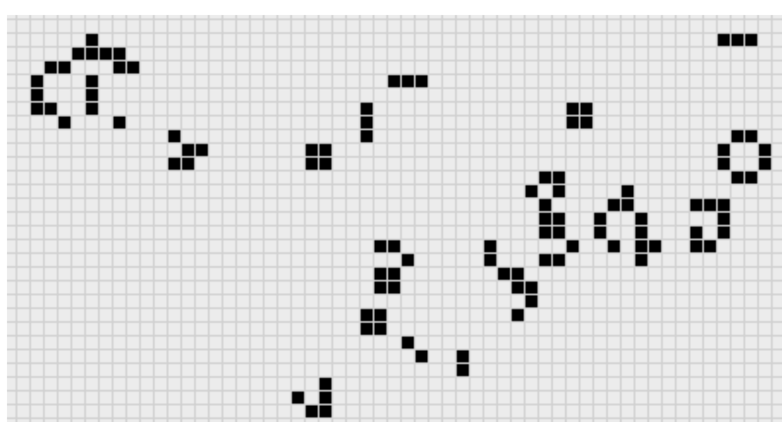
- a regular grid of cells,
  - can be any number of dimensions
- each cell has:
  - a finite state at any given time
  - a defined neighbourhood of other cells
- transition rules that describe how a cell changes its state over time:
  - based upon the current state of itself and its neighbouring cells

Many, Many Variations

- the grid can be any finite number of dimensions, but is typically one or two-dimensional:
  - a 1-dimensional row

1	0	1	0	1	1	1	0	0	1

  - a 2-dimensional square lattice



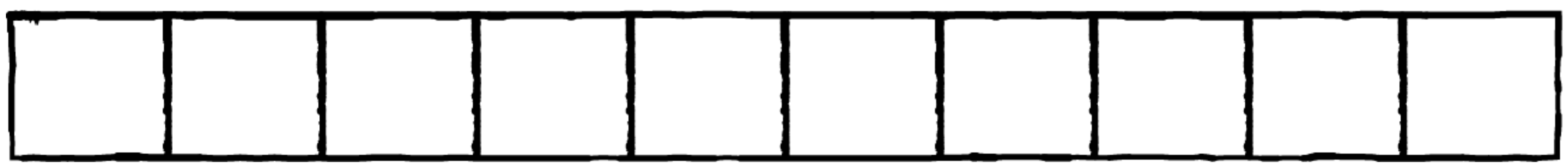
- other forms such as rings and hexagonal lattices have also been used

Many, Many Variations

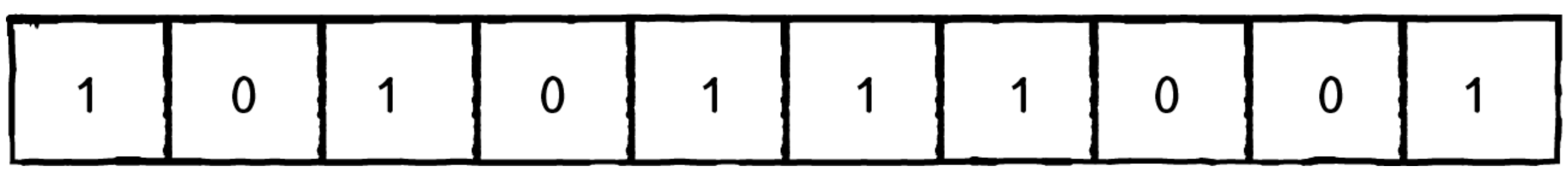
- any finite number of states can be used for cell values
- and a cell's neighbourhood can be as wide or as complicated as we desire
- so a huge number of CA models have been discovered or created
    - some to perform or attempt specific tasks
    - and some to research complexity
- but right now, to see how CA work, we'll focus on the simplest form of CA, the Elementary Cellular Automaton
- because as it turns out, this automaton is just as powerful and expressive as all the others…

Elementary Cellular Automata

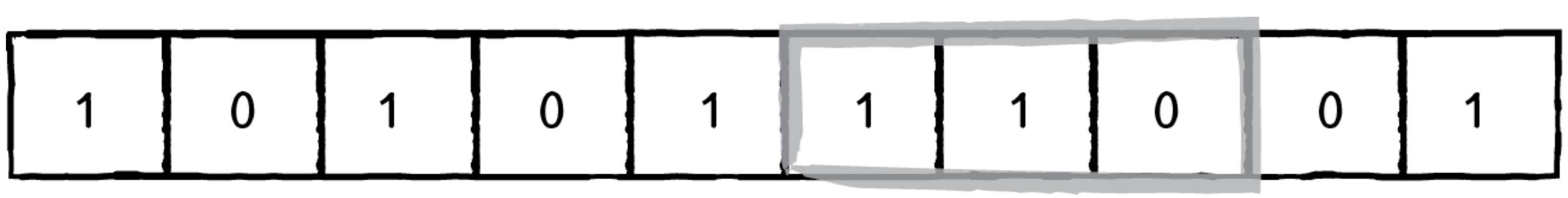
- a simple, one-dimensional line of cells:



- there are just two states:



- each cells neighbourhood consists of itself, and it’s immediate left and right neighbours:



figures 7.2/3/4 from The Nature of Code

Elementary Cellular Automata

- the states change over time, synchronously:

generation 0	1	0	1	0	1	1	1	0	0	1
generation 1	0	0	0	0	1	0	1	1	1	0
generation 2	0	0	0	1	0	0	1	0	1	1

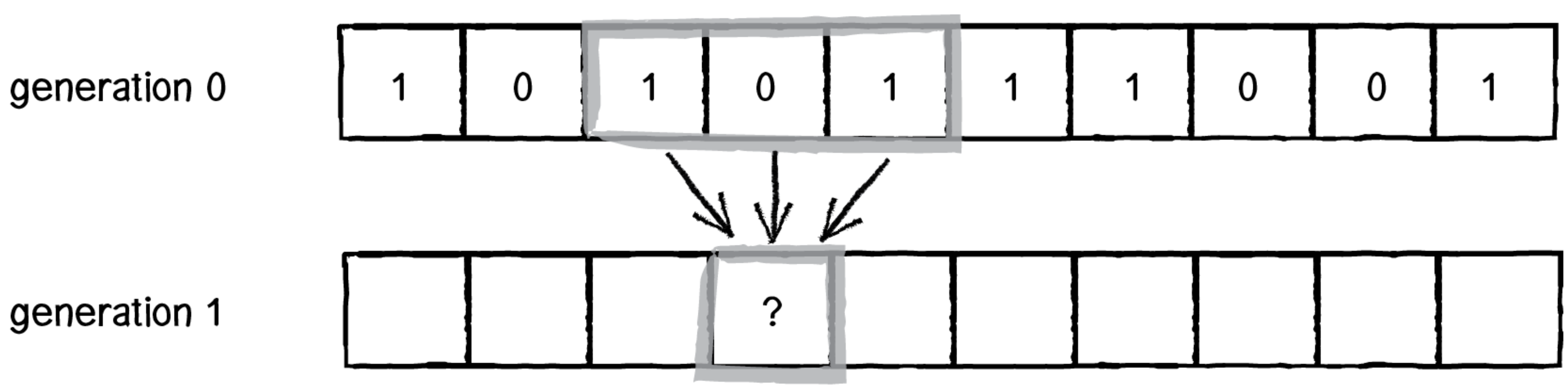
- but how?

Transition Rules

- transition rules determine a cell's next state:

  `CELL state at time t =`

  `f(CELL neighbourhood at time t - 1)`



Transition Rules

- transition rules are defined for every possible neighbourhood:

1 1 1	1 1 0	1 0 1	1 0 0	0 1 1	0 1 0	0 0 1	0 0 0
⬇	⬇	⬇	⬇	⬇	⬇	⬇	⬇
0	1	0	1	1	0	1	0

- together these make a ruleset, commonly called a Rule
- this ruleset is known as "Rule 90"
  - can you figure out why?
  - how many different Rules are there?

Rule 90

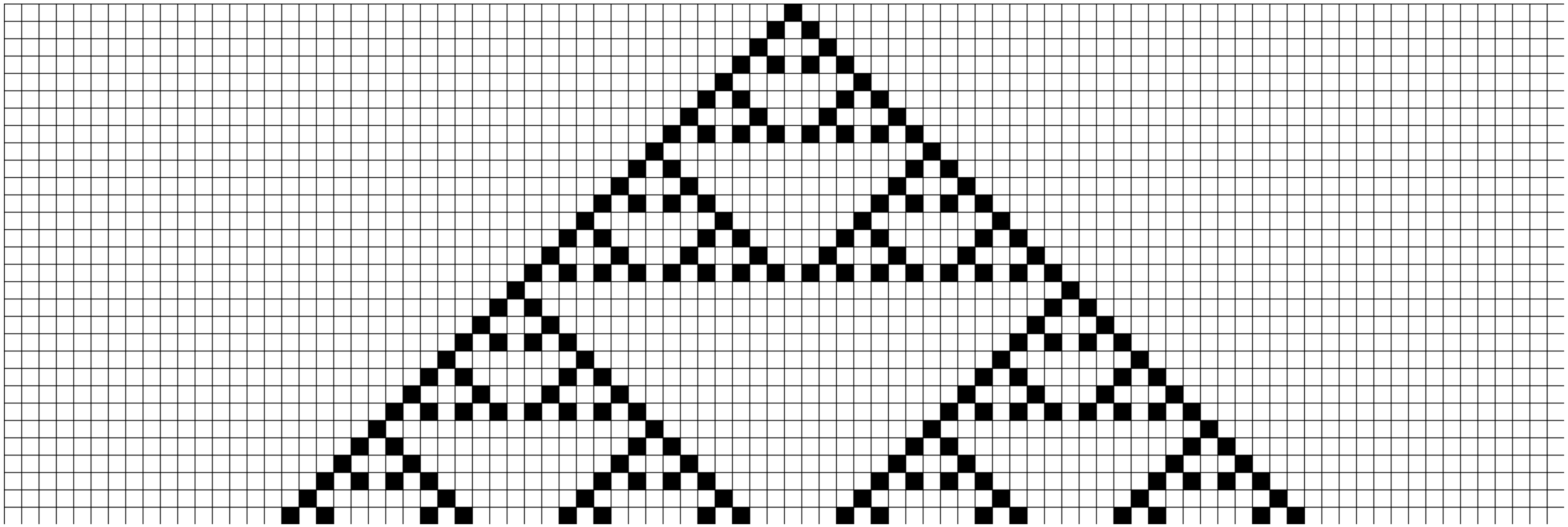


figure 7.12 from The Nature of Code

- in this visualization the first row represents generation 0
- the next row is generation 1
- and so on
- black squares represent 1, and white squares represent 0

Rule 90

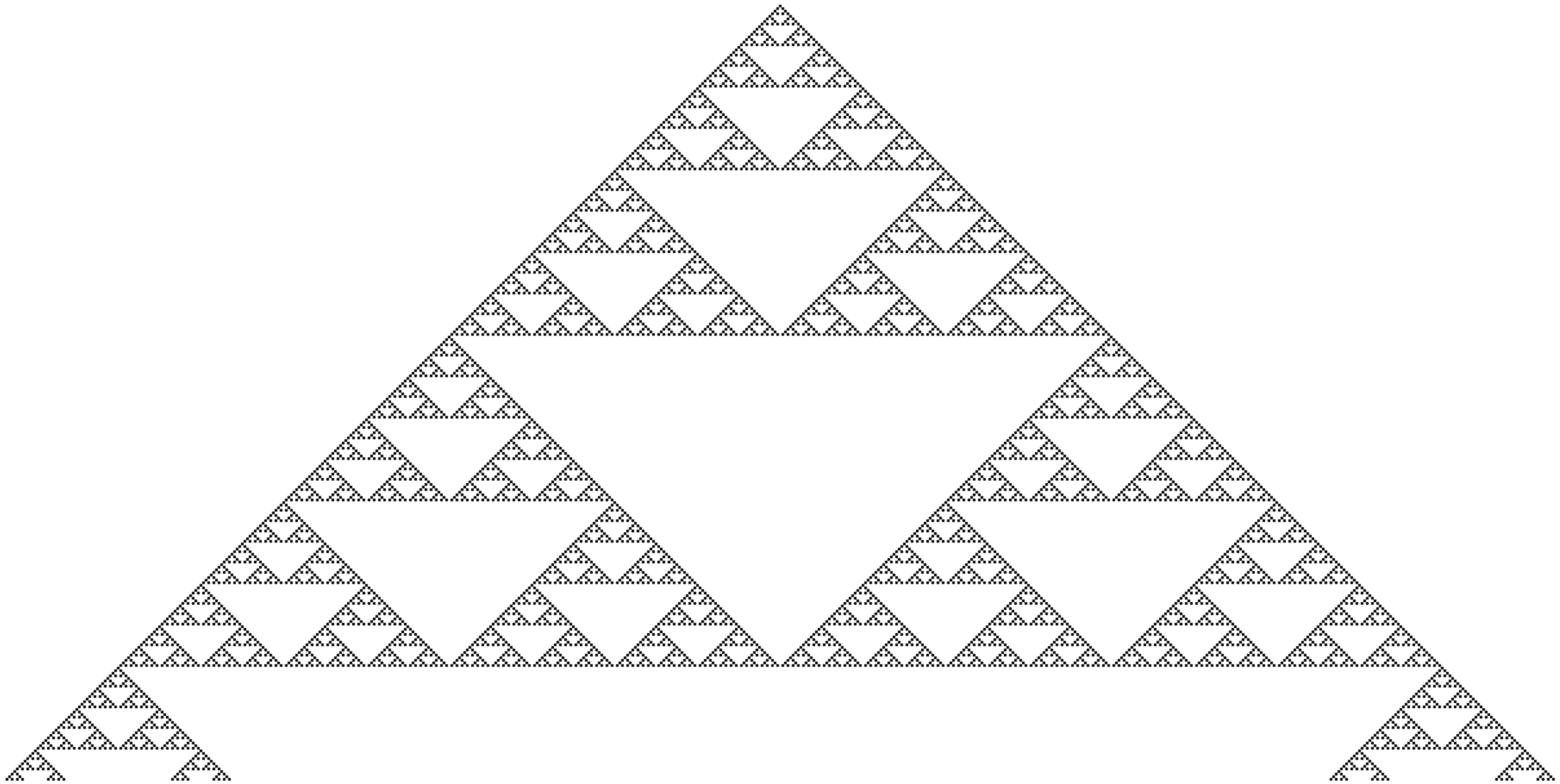
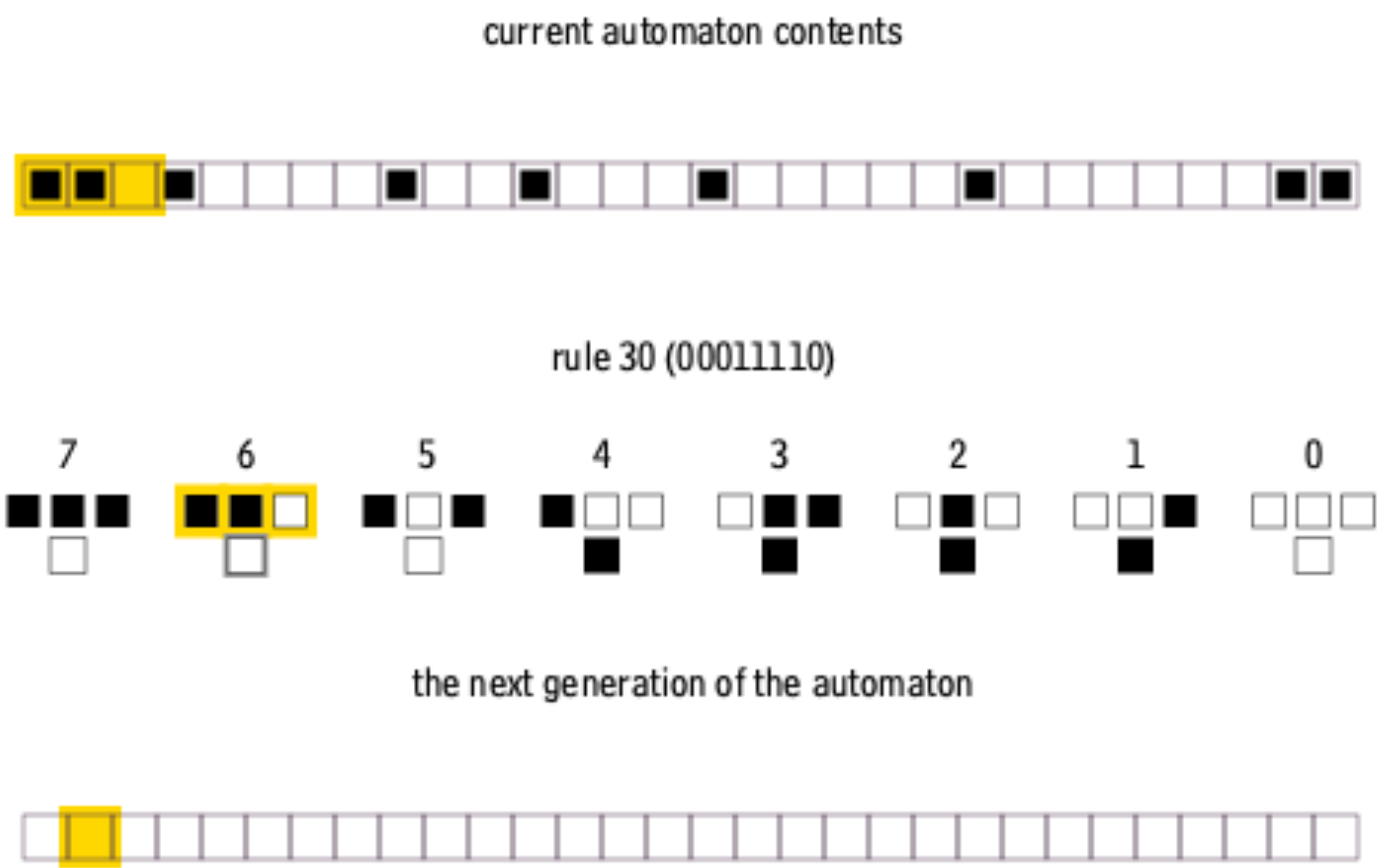


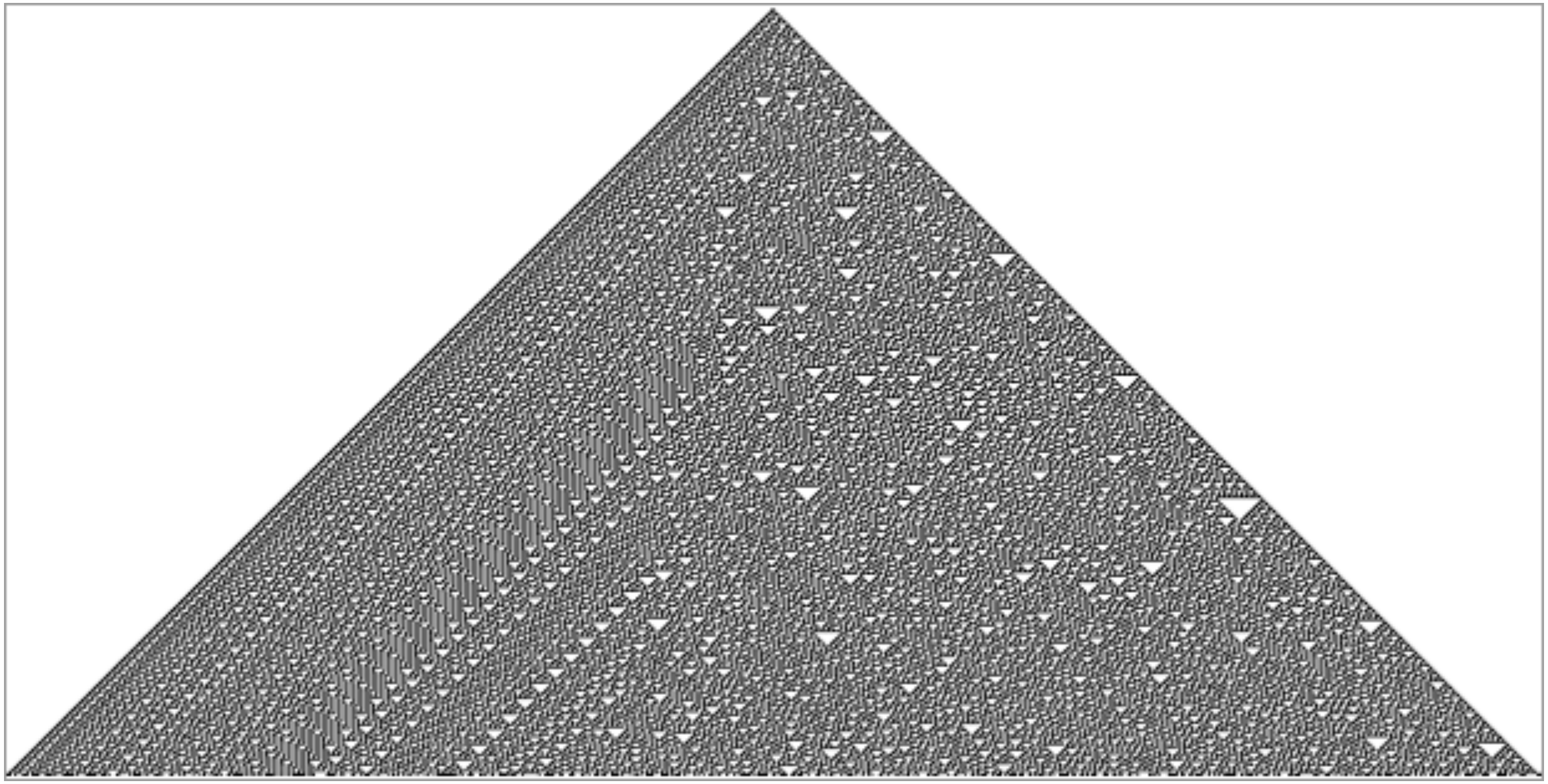
figure 7.13 from The Nature of Code

- we can run the CA for many iterations and observe the self-similar (fractal) nature of its structure
- in this case, the Sierpiński triangle

Rule 30

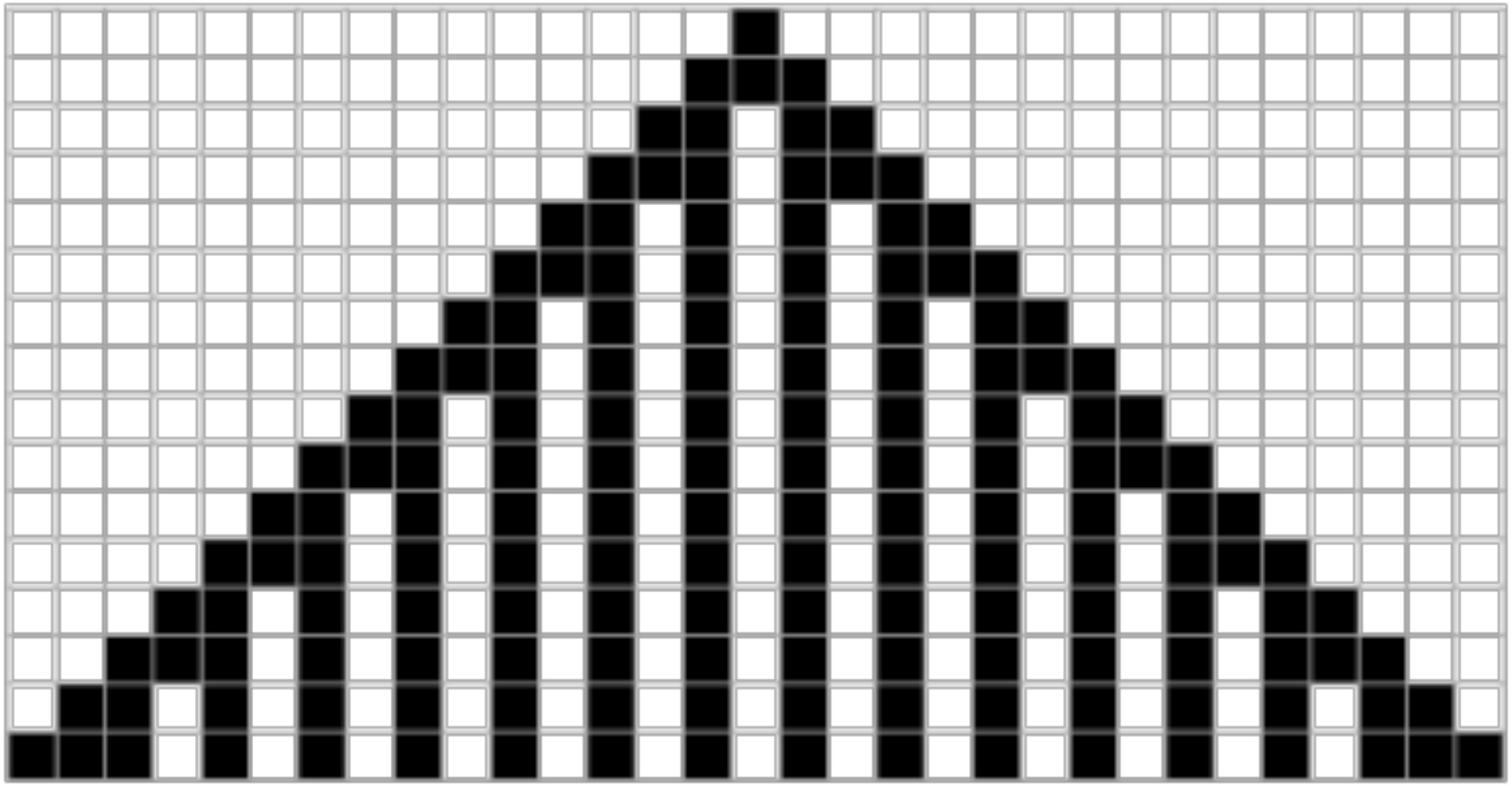

current automaton contents
rule 30 (00011110)
7
6
5
4
3
2
1
0
the next generation of the automaton

Rule 30



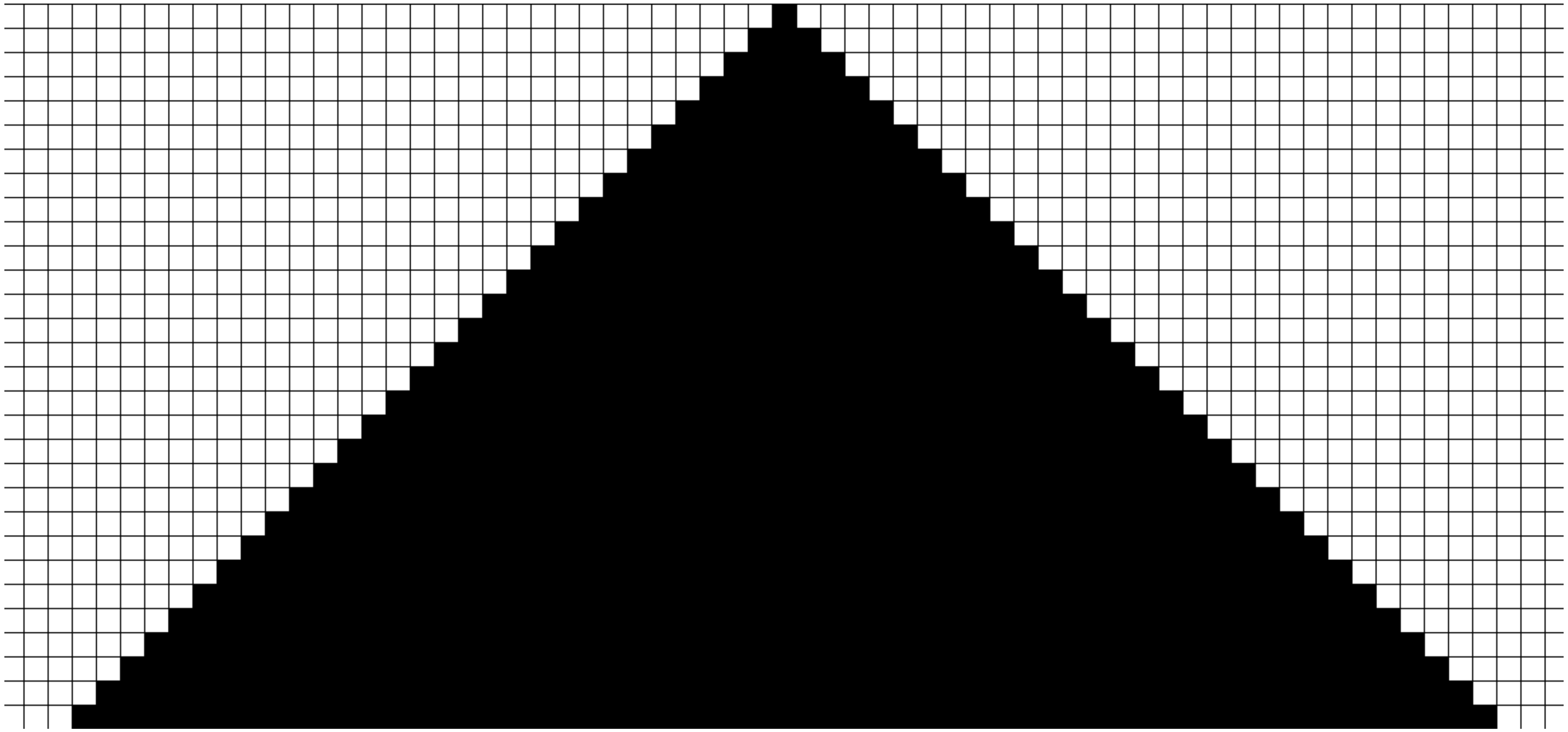
- same initial configuration as Rule 90
    - a single cell set to ‘1’
- but a very different behaviour

Rule 94



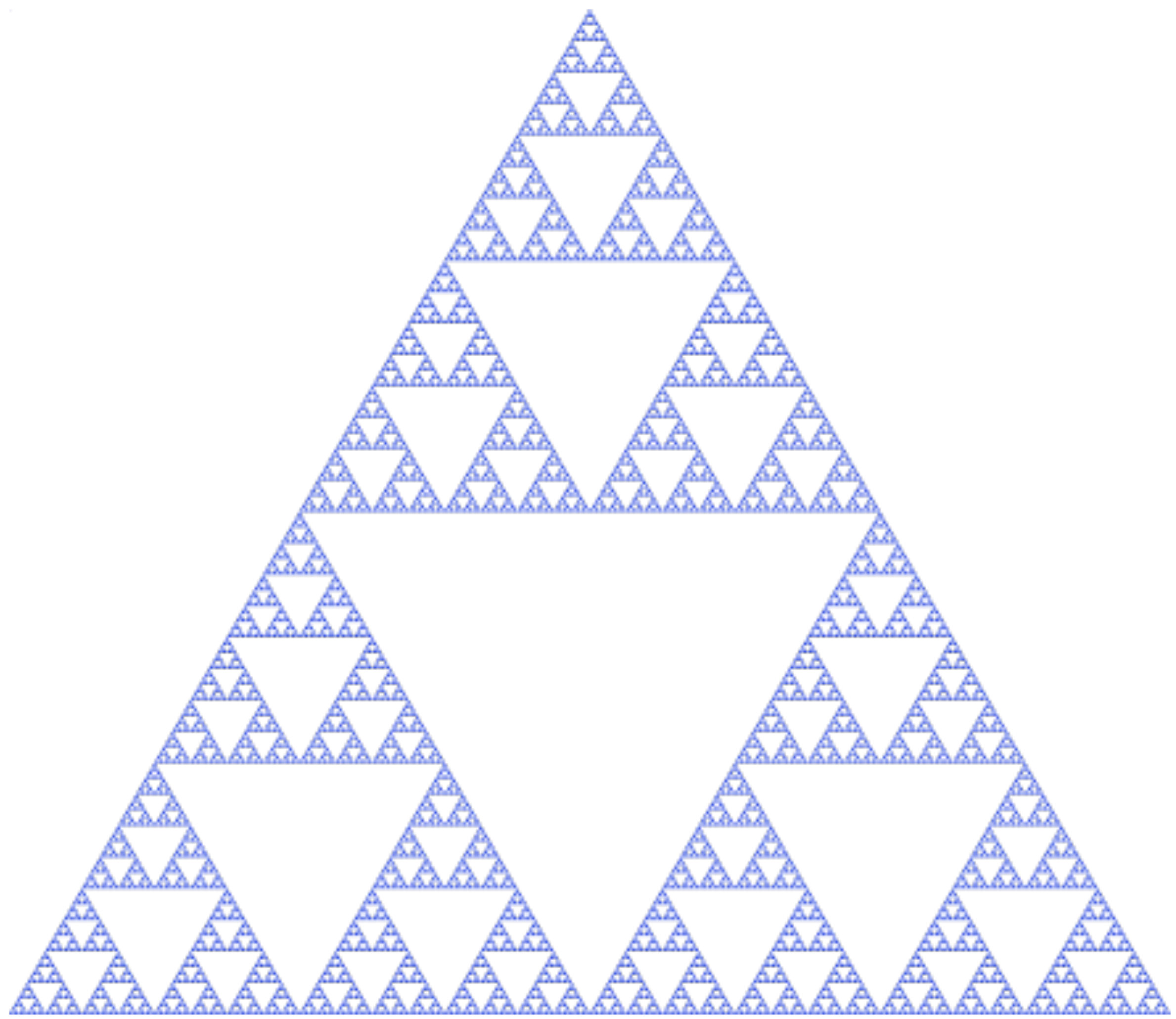
- ...again different

Rule 222



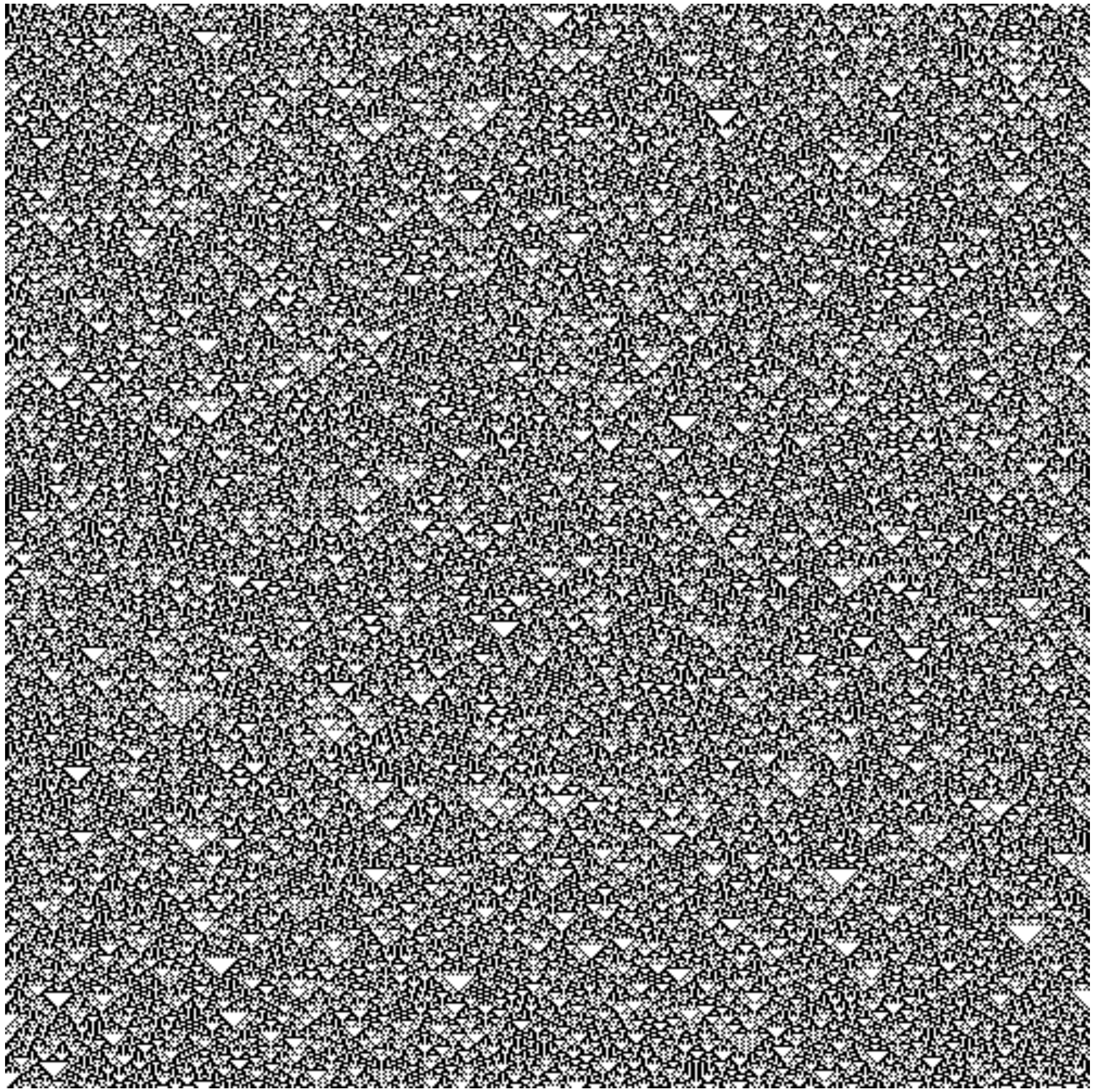
- …and again
- but it's not just the Rule that determines the outcome

Sensitivity to Initial Conditions



Compare Rule 90 seeded with a single '1'…

Sensitivity to Initial Conditions



… to Rule 90 seeded with a random initial configuration

Sensitivity to Initial Conditions

- so the behaviour of a cellular automaton is determined by a combination of its Rule and its initial configuration
- but some Rules always lead to predictable, often dull outcomes, regardless of the initial condition
  - consider: what will Rule 0 always do?
- while others, such as Rule 30 and Rule 90, sometimes lead to more interesting behaviour
- so Stephen Wolfram set about classifying each of the Rules…

Wolfram Classification

- Wolfram divided the range of outcomes into four classes:
    1. Uniformity
    2. Repetition
    3. Random
    4. Complexity

Wolfram Classification

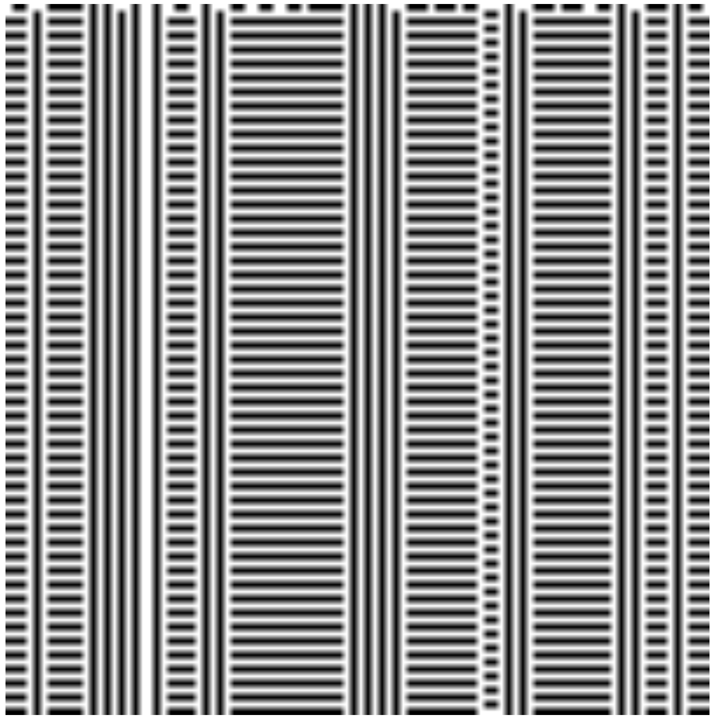
Class 1: Uniformity

- after some number of generations every cell state will remain constant
- example: Rule 232

Wolfram Classification

Class 2: Repetition

- after some number of generations the cell states oscillate in a regular pattern
- example: Rule 5



Wolfram Classification

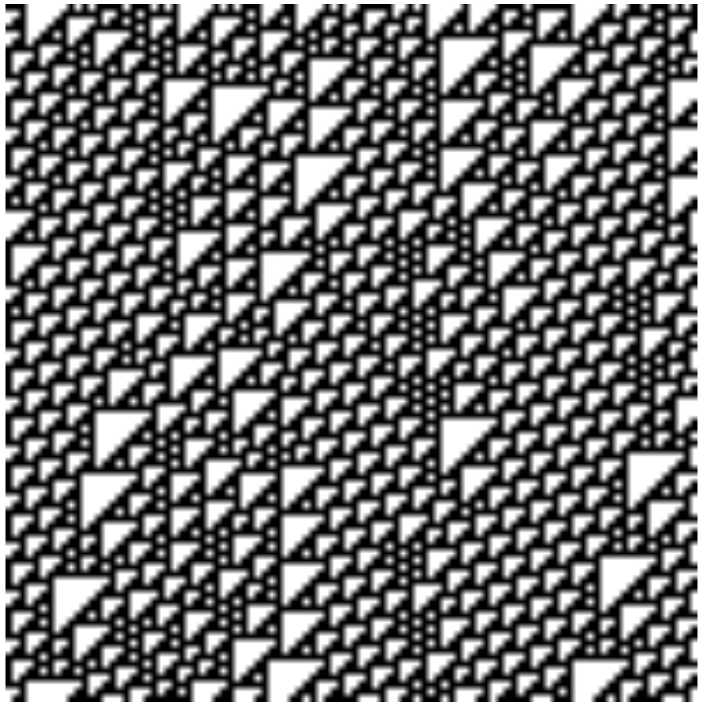
Class 3: Random

- the cell states have no discernible pattern
- example: Rule 30

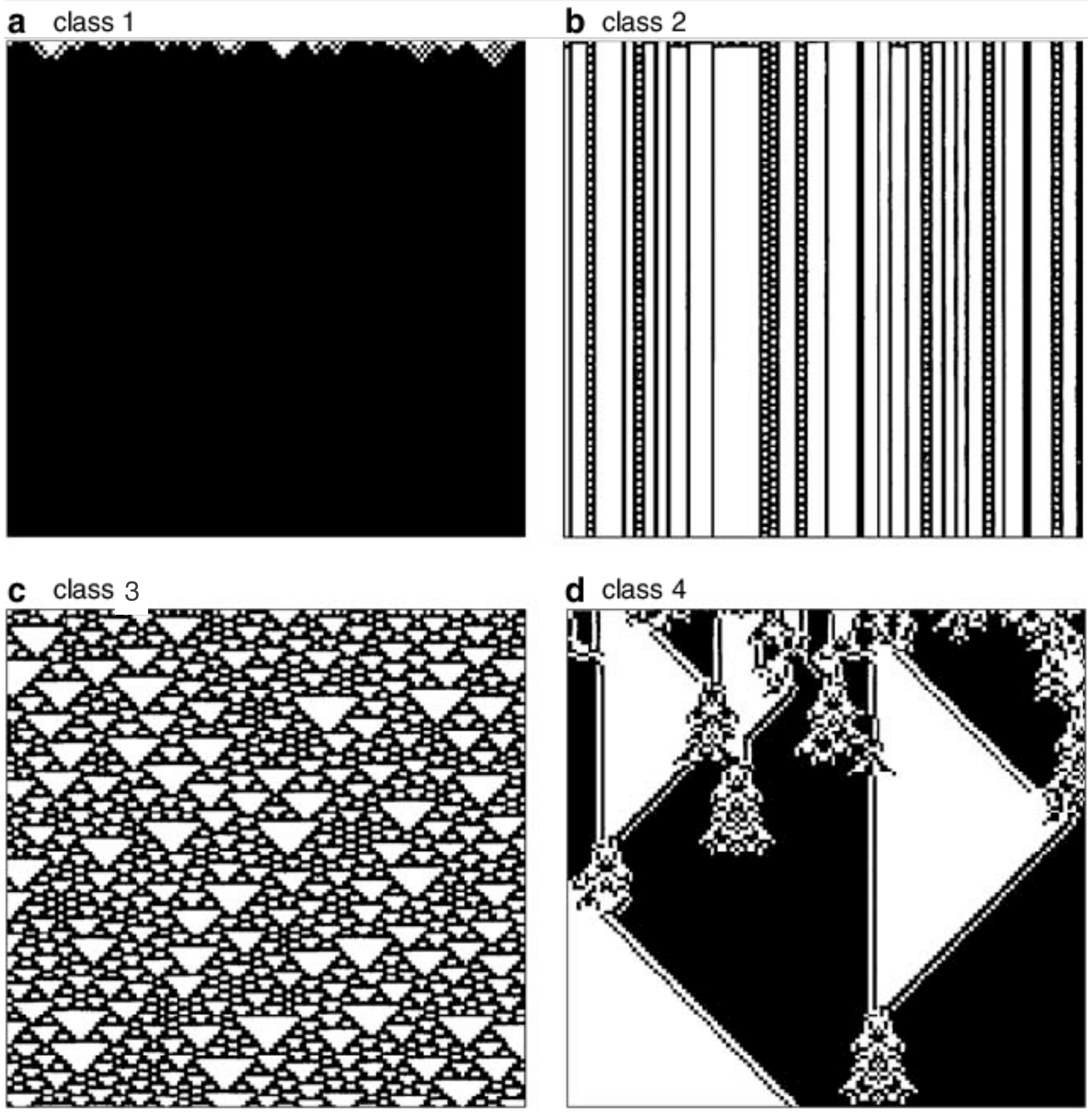
Wolfram Classification

Class 4: Complexity

- the cell states exhibit the properties of complex systems
- example: Rule 110
- (more on this later)



Wolfram Classification



Wolfram Classification

- Wolfram's classification is not without its difficulties
- in particular deciding if a Rule belongs to Class 3 or Class 4
- so how did he decide?
- surprisingly, he judged by eye!
- as you can imagine, that particular method has its critics
- but the boundary between Class 3 and Class 4

  - between random and complex behaviour -

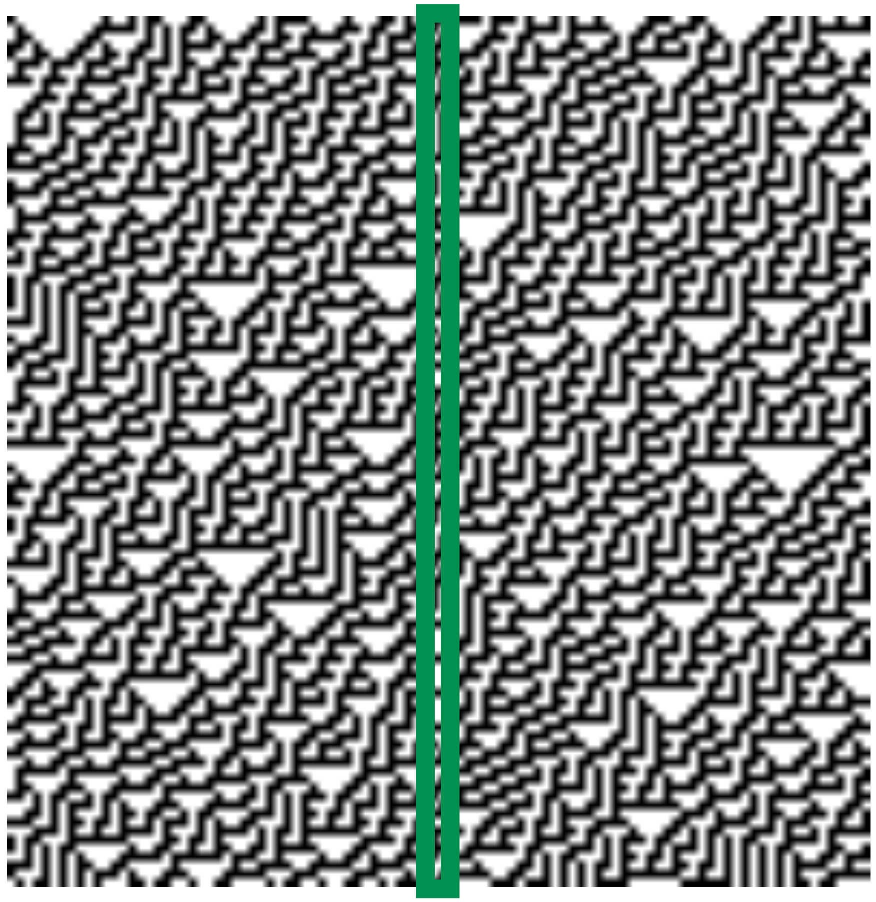
  - between order and chaos -
- is, of course, a fruitful area of study

Rule 30

- recall that Rule 30 is in Class 3 - ‘Random’
- saying that a deterministic system, whose rules are very well understood, exhibits ‘random’ behaviour might seem to be a contradiction
- and yet it’s not
- the output of Rule 30 passes every test for randomness!
- it was used in Mathematica as the random number generator

Rule 30

- if we begin at an arbitrary iteration and read out the value of the centre cell state each iteration, we have no way of predicting what the next value will be

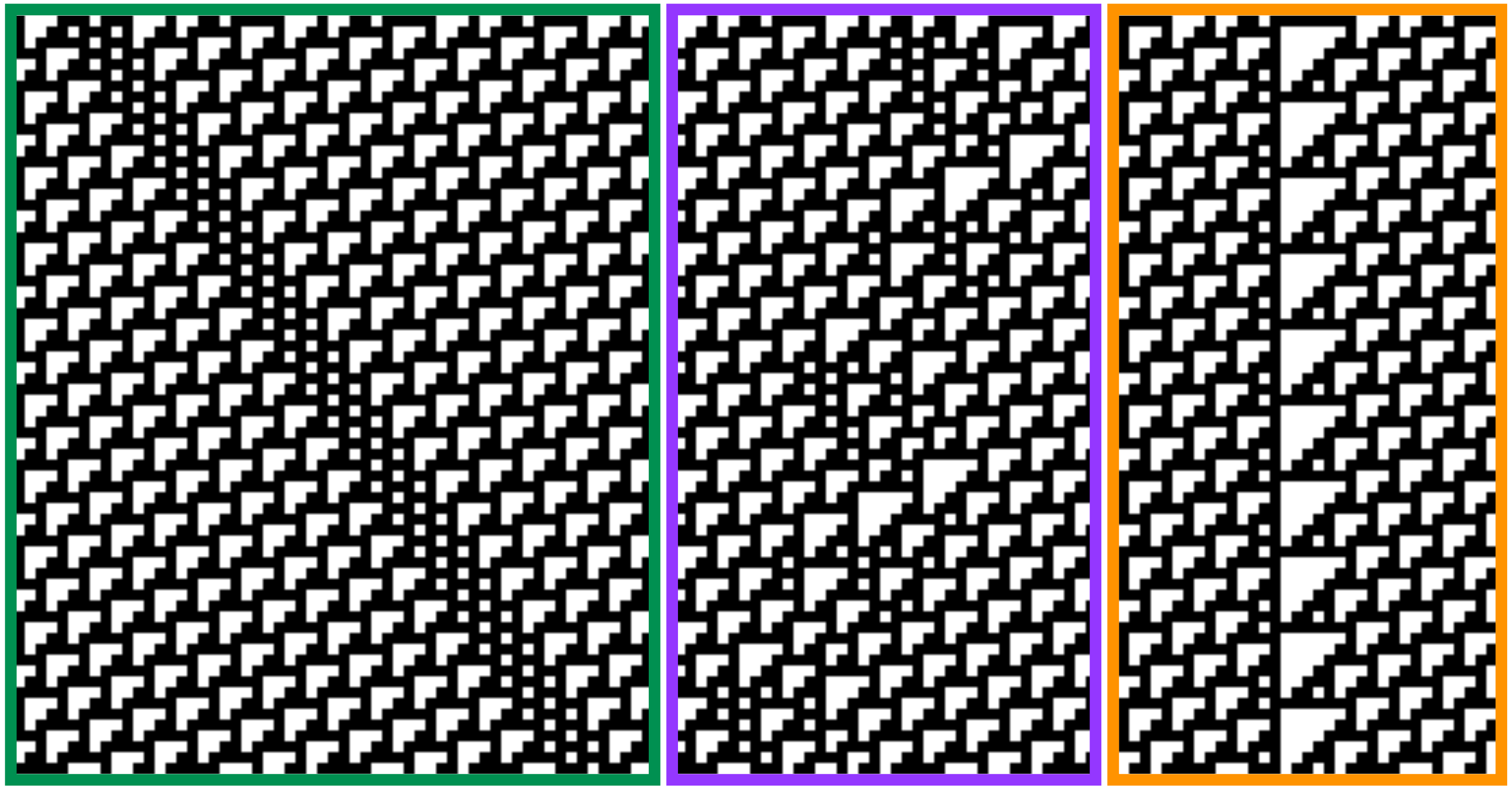


Rule 110

1 1 1	1 1 0	1 0 1	1 0 0	0 1 1	0 1 0	0 0 1	0 0 0
⬇	⬇	⬇	⬇	⬇	⬇	⬇	⬇
0	1	1	0	1	1	1	0

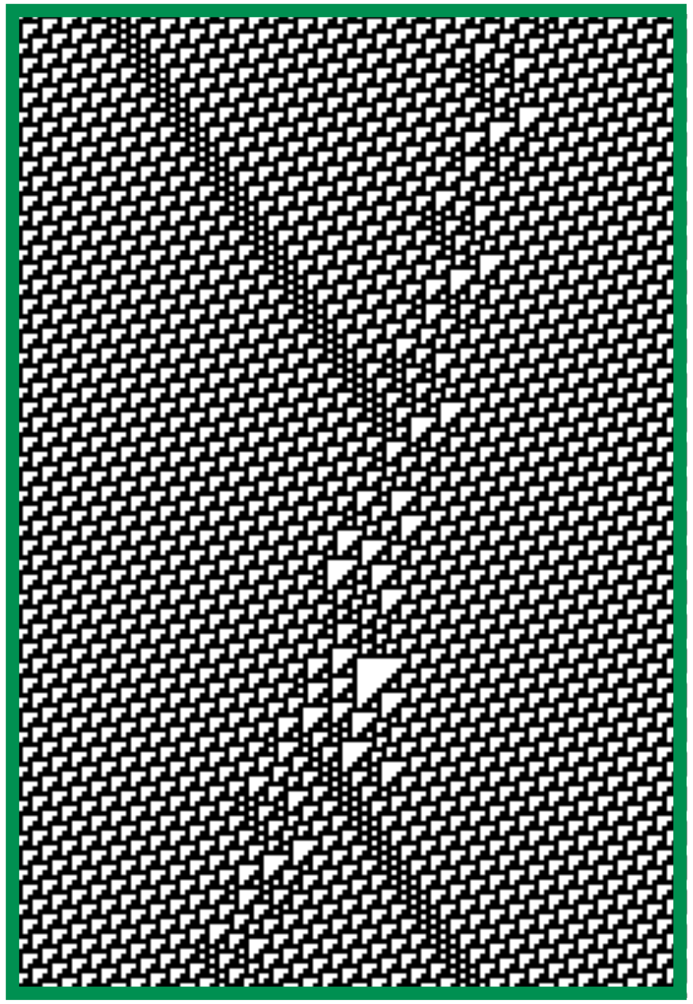
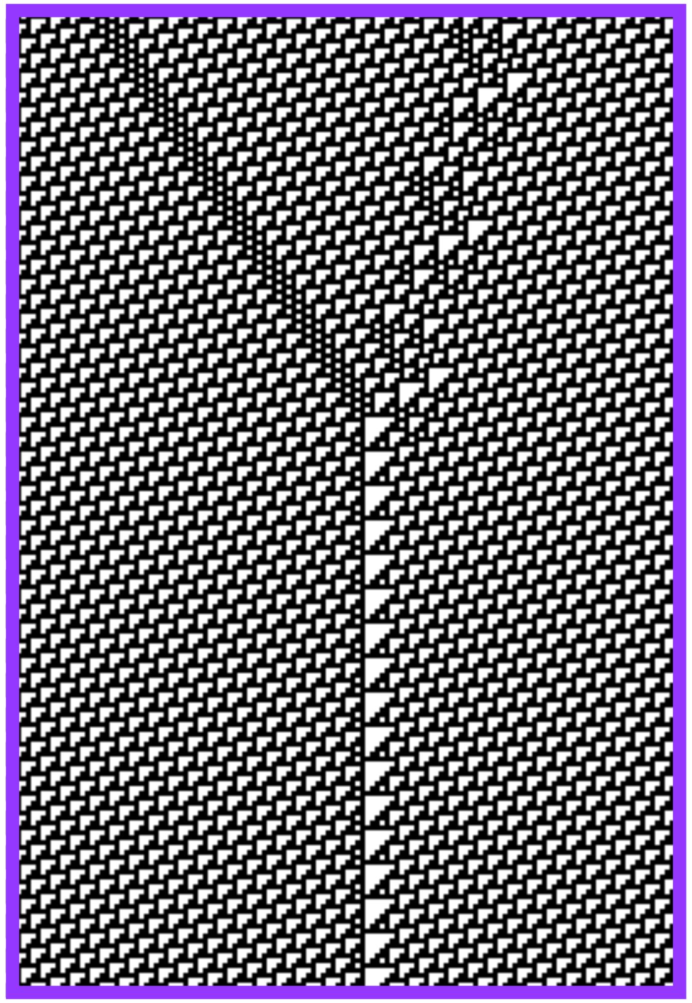
- in 2004, Matthew Cook proved that Rule 110 is Turing Complete
- and therefore capable of universal computation
- computation is achieved using a finite number of localized patterns embedded within an infinitely repeating background pattern

Rule 110



- three localized patterns used for universal computation:
- left: shifts to the right two cells and repeats every three generations
- centre: shifts left eight cells and repeats every thirty generations
- right: remains stationary and repeats every seven generations

Rule 110

- left: two structures pass through each other without interaction
- right: two structures interact to form a new third structure

Rule 110

- using Rule 110 for universal computation is, of course, thoroughly impractical
  - it is difficult to configure and slow
- but, on the whole, elementary CA show how a variety of computations can be performed by very simple systems
- perhaps systems that could be implemented physically…
- perhaps by living things…

CA in Nature

- the seashell *conus textile*
- pattern resembles Rule 30
- pigment cells reside in a narrow band along the shell's lip
- each cell secretes pigments according to the activating and inhibiting activity of its neighbour pigment cells

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!)