Nature Inspired Computing

K.Dimopoulos

1D Cellular Automata

What Is a Cellular Automaton?

A cellular automaton (cellular automata plural, or CA for short) is a model of a system of cell objects with the following characteristics:

- The cells live on a grid. (I'll include examples in both one and two dimensions in this chapter, though a CA can exist in any finite number of dimensions.)
- Each cell has a state, though a cell's state can vary over time. The number of possible states is typically finite. The simplest example has the two possibilities of 1 and 0 (otherwise referred to as on and off, or alive and dead).
- Each cell has a neighborhood. This can be defined in any number of ways, but it's typically all the cells adjacent to that cell.

CA don't refer to biological cells (although you'll see how CA can mimic lifelike behavior and have applications in biology). Instead, they simply represent discrete units in a grid, similar to the cells in a spreadsheet (as in Microsoft Excel).

The development of CA systems is typically attributed to Stanisław Ulam and John von Neumann, who were both researchers at the Los Alamos National Laboratory in New Mexico in the 1940s.

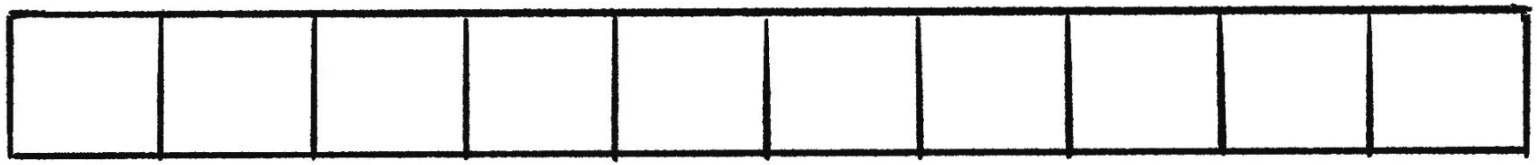
Wolfram's CA

Perhaps the most significant (and lengthy) scientific work studying CA arrived in 2002: Stephen Wolfram's 1,280-page A New Kind of Science. Available in its entirety for free online, Wolfram's book discusses how CA aren't simply neat tricks but are relevant to the study of biology, chemistry, physics, and all branches of science.

What's the simplest CA you can imagine? For Wolfram, an elementary CA has three key elements:

- Grid
- States
- Neighborhood

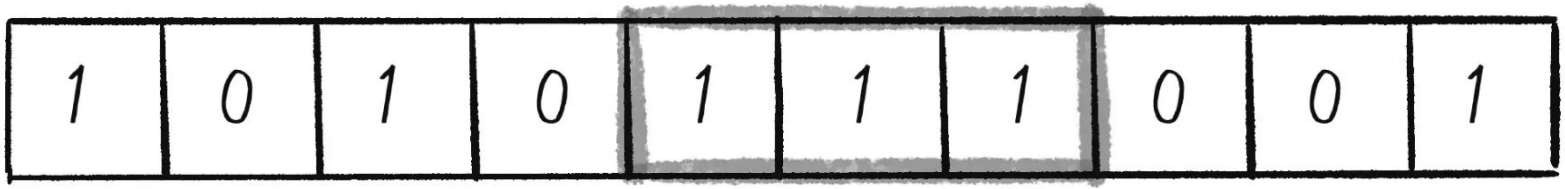
The simplest grid would be 1D: a line of cells



Neighbours

The simplest neighborhood in one dimension for any given cell would be the cell itself and its two adjacent neighbors: one to the left and one to the right.

I have a line of cells, each with an initial state, and each with two neighbors. The exciting thing is, even with this simplest CA imaginable, the properties of complex systems can emerge.

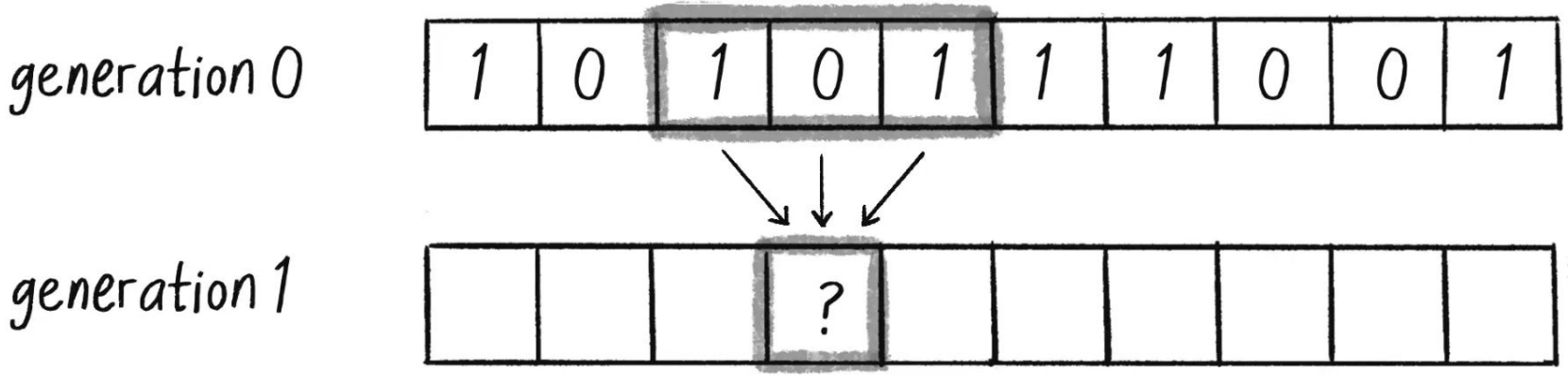


Change with time

This CA is developing across a series of discrete time steps, which could also be called generations.

Given the states of the cells at time equals 0 (or generation 0), how do I compute the states for all cells at generation 1? And then how do I get from generation 1 to generation 2? And so on and so forth.

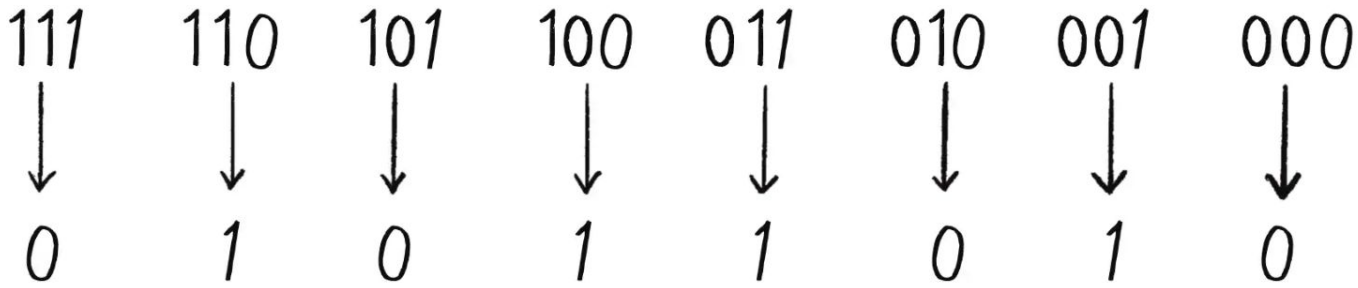
A cell's new state is a function of all the states in the cell's neighborhood at the previous generation (time t-1). A new state value is calculated by looking at the previous generation's neighbor states:



CA ruleset

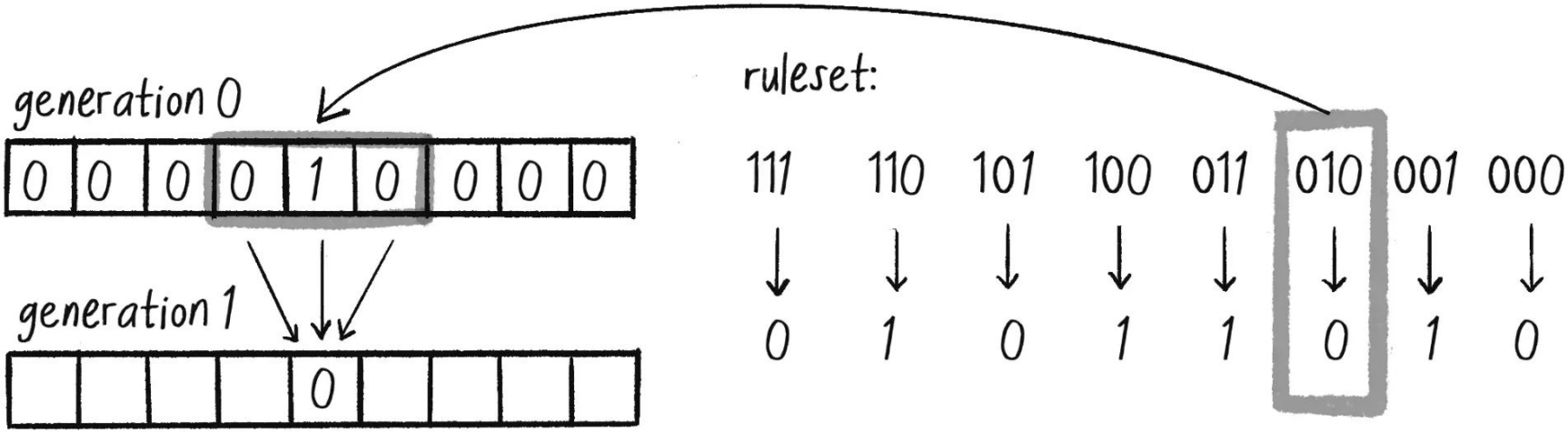
New states are determined by predefined rules that account for every possible configuration of a cell and its neighbors. These rules are known collectively as a **ruleset**.

Once all the possible neighborhood configurations are defined, an outcome (new state value: 0 or 1) is specified for each configuration. In Wolfram's original notation and other common references, these configurations are written in descending order. As we have a neighborhood of 3, each with 2 possible states, the number os combinations is $2^3 = 8$. Below is an example:



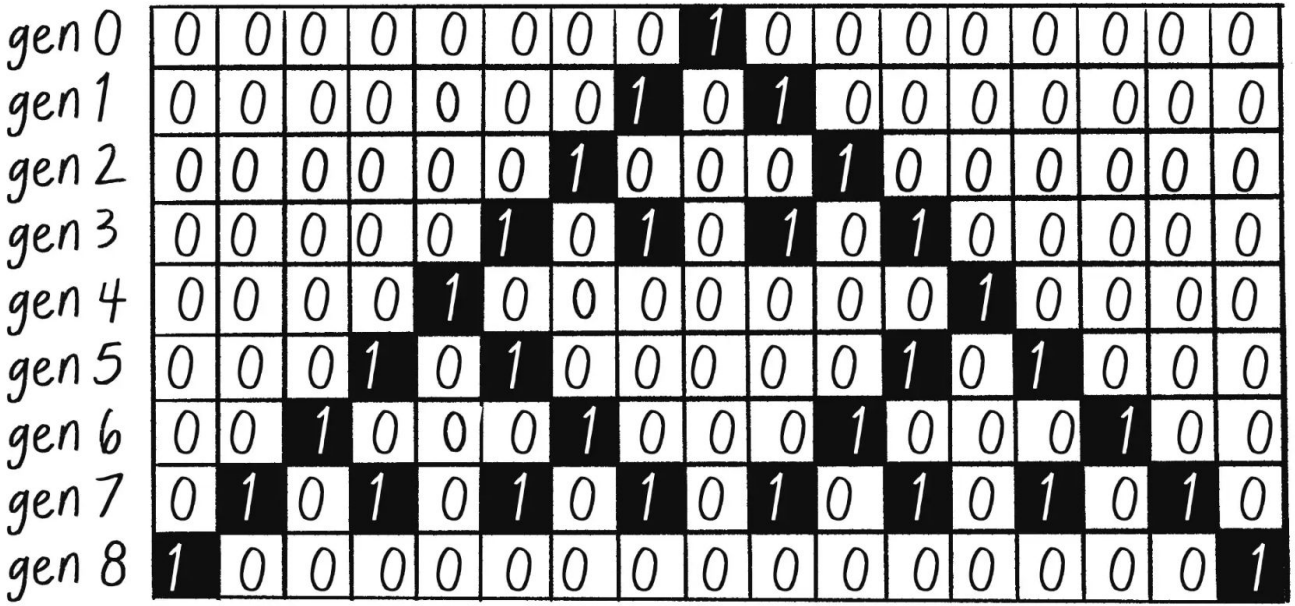
Applying the ruleset

Based on the ruleset in the previous slide, how do the cells change from generation 0 to generation 1? Below it shows how the center cell, with a neighborhood of 010, switches from a 1 to a 0. Try applying the ruleset to the remaining cells to fill in the rest of the generation 1 states.



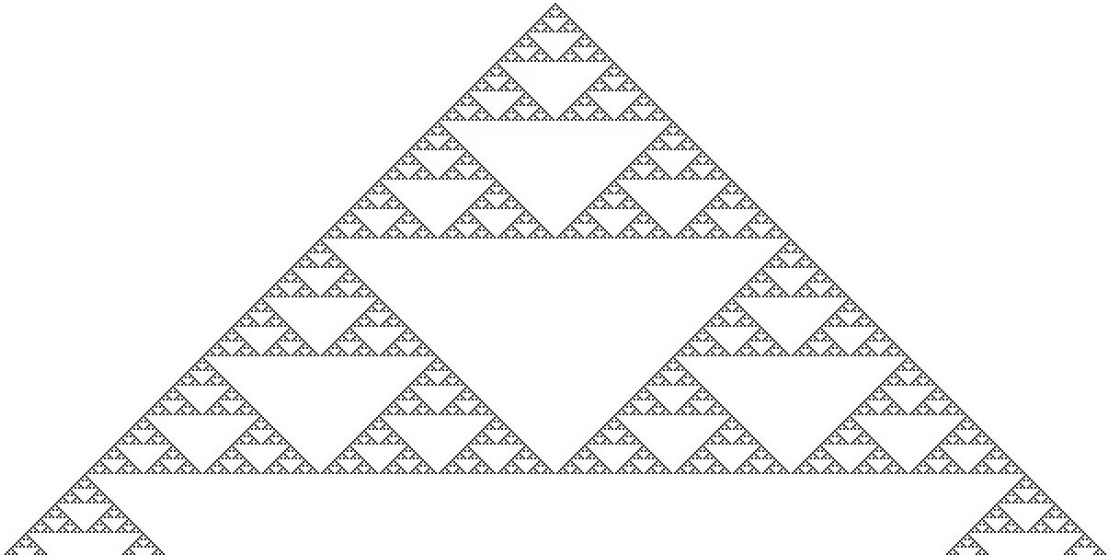
Pattern emergence

With this switch from numerical representations to visual forms, the fascinating dynamics and patterns of CA will come into view! To show them even more clearly, instead of drawing one generation at a time, I'll also start stacking the generations, with each new generation appearing below the previous one, as shown below:



Fractals

The low-resolution shape that emerged is the **Sierpiński triangle**. Named after the Polish mathematician Wacław Sierpiński, it's a famous example of a fractal. Fractals are patterns in which the same shapes repeat themselves at different scales.



Why this pattern?

Of course, this particular result didn't happen by accident. This set of rules in Figure is known to produce this pattern. The mere act of defining a ruleset doesn't guarantee visually exciting results. In fact, for a 1D CA in which each cell can have two possible states, there are exactly 256 (2^8) possible rulesets to choose from, and only a handful are on par with the Sierpiński triangle.

Since the eight possible inputs are the same no matter what, potential shorthand for indicating a ruleset is to specify just the outputs, writing them as a sequence of eight 0s or 1s—in other words, an 8-bit binary number. For example, the ruleset before could be written as 01011010. also known as rule 90.

What kind of patterns can we do?

Most of the 256 elementary rulesets don’t produce compelling outcomes.

However, it’s still quite incredible that even just a few of these rulesets—simple systems of cells with only two possible states—can produce fascinating patterns seen every day in nature.

For example, next picture shows a snail shell resembling Wolfram’s rule 30. This demonstrates how valuable CAs can be in simulation and pattern generation.

Handling the edges

How should we handle the cell on the edge that doesn't have a neighbor to both its left and it's right? Here are three possible solutions to this problem:

1. **Edges remain constant.** This is perhaps the simplest solution. Don't bother to evaluate the edges, and always leave their state value constant (0 or 1).
2. **Edges wrap around.** Think of the CA as a strip of paper, and turn that strip of paper into a ring. The cell on the left edge is a neighbor of the cell on the right edge, and vice versa. This can create the appearance of an infinite grid and is probably the most commonly used solution.
3. **Edges have different neighborhoods and rules.** If I wanted to, I could treat the edge cells differently and create rules for cells that have a neighborhood of two instead of three. You may want to do this in some circumstances, but in this case, it's going to be a lot of extra lines of code for little benefit.

Wolfram Classification

The vast majority of elementary CA rulesets produce visually uninspiring results, while some result in wondrously complex patterns like those found in nature. Wolfram has divided the range of outcomes into four classes:

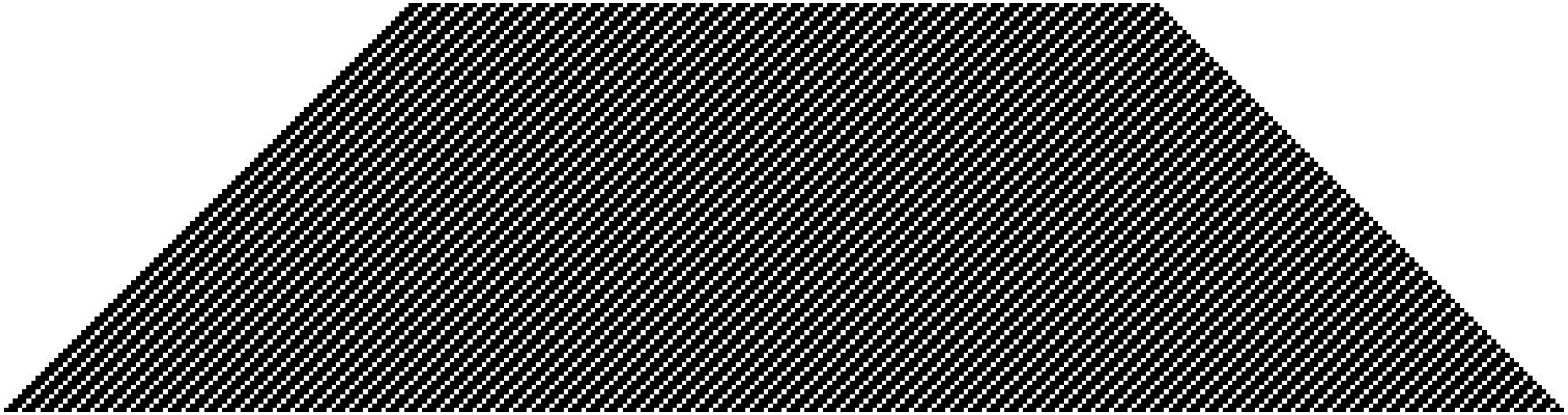
- Class 1: Uniformity
- Class 2: Repetition
- Class 3: Random
- Class 4: Complexity

Class 1: Uniformity

Class 1 CAs end up, after a certain number of generations, with every cell constant. This isn't terribly exciting to watch. Rule 222 is a class 1 CA; if you run it for enough generations, every cell will eventually become and remain black

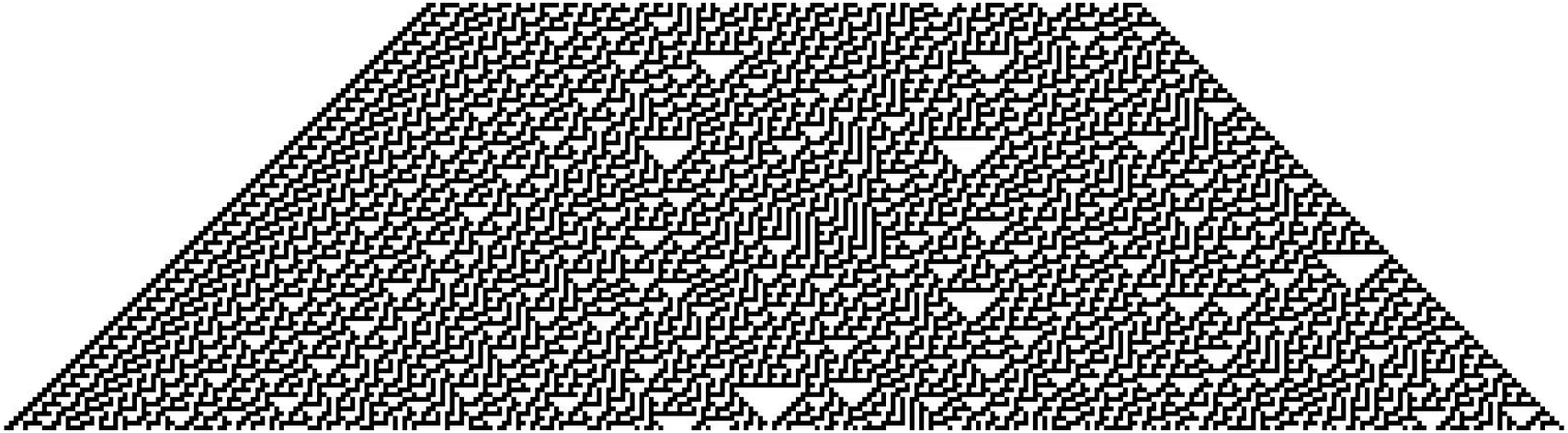
Class 2: Repetition

Like class 1 CAs, class 2 CAs remain stable, but the cell states aren't constant. Instead, they oscillate in a repeating pattern of 0s and 1s. In rule 190, each cell follows the sequence `11101110111011101110`



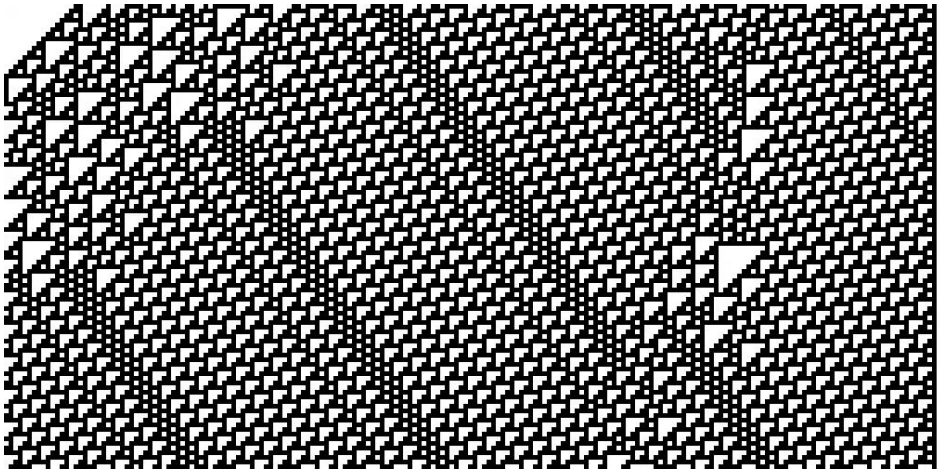
Class 3: Random

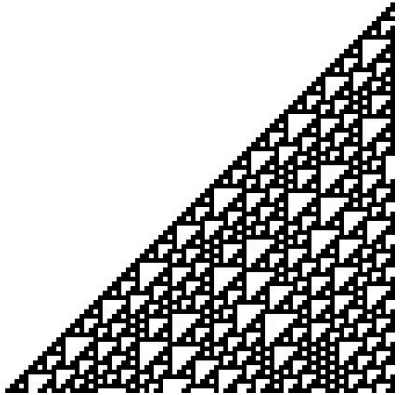
Class 3 CAs appear random and have no easily discernible pattern. In fact, rule 30 is used as a random-number generator in Wolfram's Mathematica software. Again, you can feel amazed that such a simple system with simple rules can descend into a chaotic and random pattern.



Class 4: Complexity

Class 4 CAs can be thought of as a mix between class 2 and class 3. You can find repetitive, oscillating patterns inside the CA, but where and when these patterns appear is unpredictable and seemingly random. If a class 3 CA wowed you, then a class 4 like rule 110 should really blow your mind!





The Game of Life

Conway's Game of Life

In 1970, Martin Gardner wrote a Scientific American article that documented mathematician John Conway's new Game of Life, describing it as recreational mathematics: "To play life you must have a fairly large checkerboard and a plentiful supply of flat counters of two colors. It is possible to work with pencil and graph paper but it is much easier, particularly for beginners, to use counters and a board."

The Game of Life has become something of a computational cliché, as myriad projects display the game on LEDs, screens, projection surfaces, and so on. But practicing building the system with code is still valuable for a few reasons.

Conway’s goals

Unlike von Neumann, who created an extraordinarily complex system of states and rules, Conway wanted to achieve a similar lifelike result with the simplest set of rules possible. Let’s look at how Gardner outlined Conway’s goals.

1. There should be no initial pattern for which there is a simple proof that the population can grow without limit.
2. There should be initial patterns that apparently do grow without limit.
3. There should be simple initial patterns that grow and change for a considerable period of time before coming to an end in three possible ways: fading away completely (from overcrowding or becoming too sparse), settling into a stable configuration that remains unchanged thereafter, or entering an oscillating phase in which they repeat an endless cycle of two or more periods.

This essentially describes a Wolfram class 4 CA. The CA should be patterned but unpredictable over time, eventually settling into a uniform or oscillating state. In other words, though Conway didn’t use this terminology, the Game of Life should have all the properties of a complex system.

The Structure of the Game

Instead of a line of cells, I now have a 2D matrix of cells. As with the elementary CA, the possible states are 0 or 1. In this case, however, since the system is all about life, 0 means “dead” and 1 means “alive.”

Since the Game of Life is 2D, each cell’s neighborhood has now expanded. If a neighbor is an adjacent cell, a neighborhood is now nine cells instead of three

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

The Rules of the Game

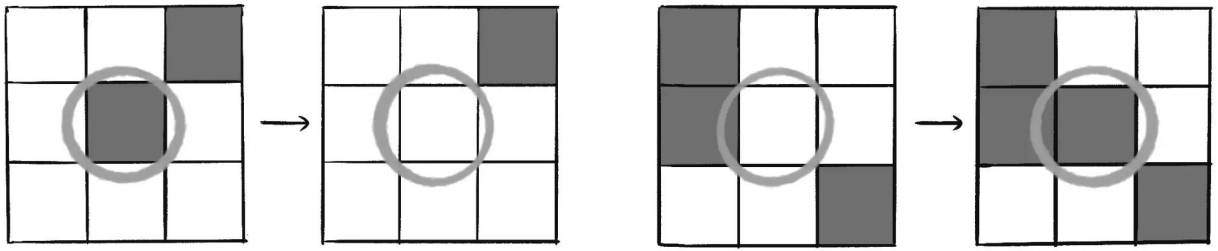
With three cells, a 3-bit number had eight possible configurations. With nine cells, there are 9 bits, or 512 possible neighborhoods. In most cases, defining an outcome for every single possibility would be impractical.

The Game of Life gets around this problem by defining a set of rules according to general characteristics of the neighborhood: Is the neighborhood overpopulated with life, surrounded by death, or just right? Here are the rules of life:

1. Death
2. Birth
3. Stasis

Death, Birth and Stasis

1. **Death:** If a cell is alive (state = 1), it will die (state becomes 0) under the following circumstances:
    - **Overpopulation:** If the cell has four or more living neighbors, it dies.
    - **Loneliness:** If the cell has one or fewer living neighbors, it dies.
2. **Birth:** If a cell is dead (state = 0), it will come to life (state becomes 1) when it has exactly three living neighbors (no more, no less).
3. **Stasis:** In all other cases, the cell's state doesn't change. Two scenarios are possible:
    - Staying alive: If a cell is alive and has exactly two or three live neighbors, it stays alive.
    - Staying dead: If a cell is dead and has anything other than three live neighbors, it stays dead.

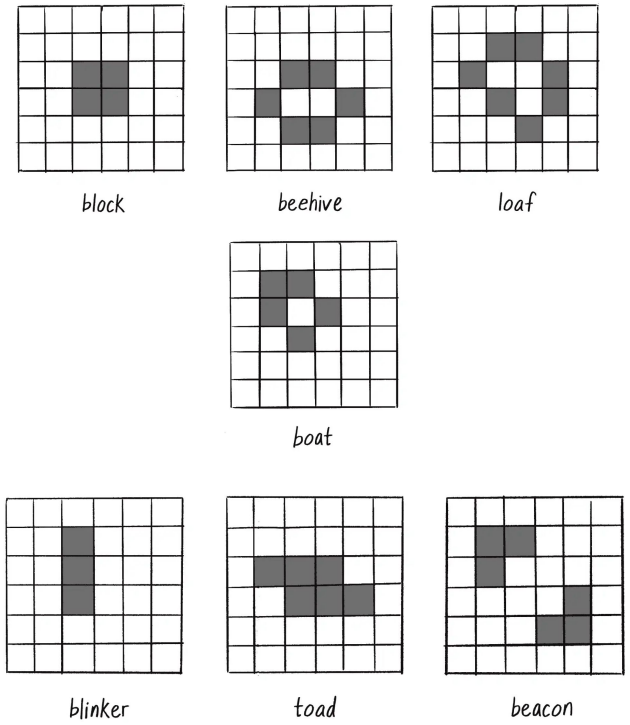


Visualising the Game of Life

With the elementary CA, we visualized many generations at once, stacked as rows in a 2D grid. A more typical way to visualize the Game of Life is to treat each generation as a single frame in an animation. This way, instead of viewing all the generations at once, you see them one at a time, and the result resembles rapidly developing bacteria in a petri dish.

One of the exciting aspects of the Game of Life is that some known initial patterns yield intriguing results. For example, the patterns shown here remain static and never change.

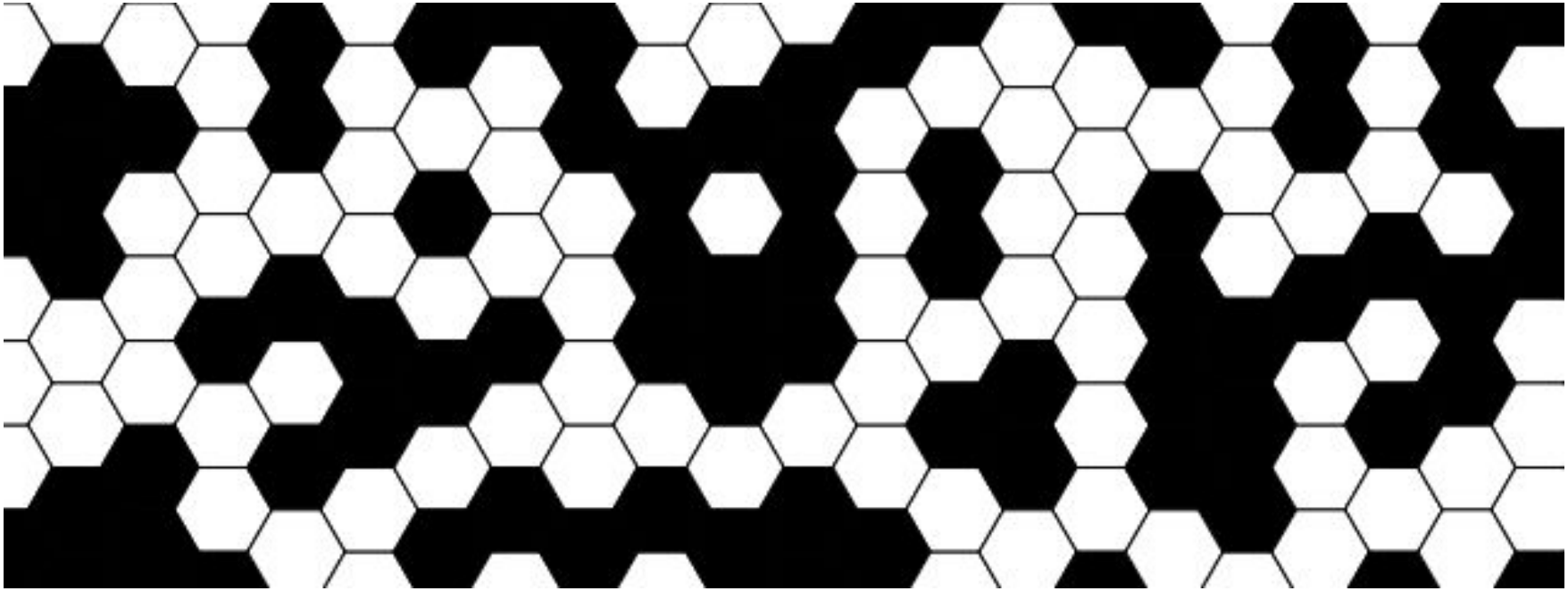
Others oscillate between two states, and others appear to be moving



Variations on Traditional CA

Non Rectangular Grids

There’s no particular reason to limit yourself to placing your cells in a rectangular grid. What happens if you design a CA with another type of shape?



Others

Probabilistic: The rules of a CA don't necessarily have to define an exact outcome:

- Overpopulation: If the cell has four or more living neighbors, it has an 80 percent chance of dying.
- Loneliness: If the cell has one or fewer living neighbors, it has a 60 percent chance of dying.

Continuous: What if the cell's state could be any floating-point number from 0 to 1?

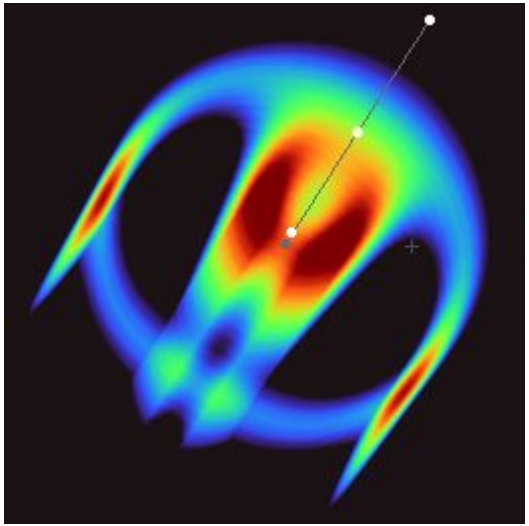
Historical: What if we use an array to keep track of a cell's state history over a longer period? This relates to the idea of a complex adaptive system, one that has the ability to change its rules over time by learning from its history.

Lenia

What is Lenia

Lenia is a family of cellular automata created by Bert Wang-Chak Chan. It is intended to be a continuous generalization of Conway's Game of Life, with continuous states, space and time. As a consequence of its continuous, high-resolution domain, the complex autonomous patterns ("lifeforms" or "spaceships") generated in Lenia are described as differing from those appearing in other cellular automata, being "geometric, metameric, fuzzy, resilient, adaptive, and rule-generic".

Lenia won the 2018 Virtual Creatures Contest at the Genetic and Evolutionary Computation Conference in Kyoto, an honorable mention for the ALIFE Art Award at ALIFE 2018 in Tokyo, and Outstanding Publication of 2019 by the International Society for Artificial Life (ISAL)



Differences with Conway's Game of Life

The Game of Life may be regarded as a special case of discrete Lenia.

Lenia has more than two states of each cell.

In Lenia's case, the neighborhood is instead a ball of radius R centered on the cell

The calculation of the new state is more complex and comes in two flavours: discrete and continuous

Over 400 "species" of "life" have been discovered in Lenia, displaying "self-organization, self-repair, bilateral and radial symmetries, locomotive dynamics, and sometimes chaotic nature". Chan has created a taxonomy for these patterns.

Wang-Chak Chan, B., "Lenia - Biology of Artificial Life", Complex Systems, 2019, 28(3), 251-286.