# Chapter 4

# The Game of Life

How can we define life? There is no simple or unanimous definition of life. All living systems share some characteristics, though: They exhibit complex emergent behaviors, can replicate themselves, and present self-regulating mechanisms that avoid exponential growth or extinction.

In the 1940s, John von Neumann defined life as an entity that can reproduce itself and simulate a Turing machine. Together with Stanislaw Ulam, he developed the concept of cellular automata, which are a grid of cells, each with a definite state (e.g., on or off, 1 or 0) that changes every time step on the basis of simple, fixed rules, equal for all the cells, applied simultaneously to the whole grid [1]. Later research by Stephen Wolfram on cellular automata classifies them in four different categories [2]: cellular automata that evolve towards homogeneity (class 1), that evolve towards an oscillating or periodic condition (class 2), that

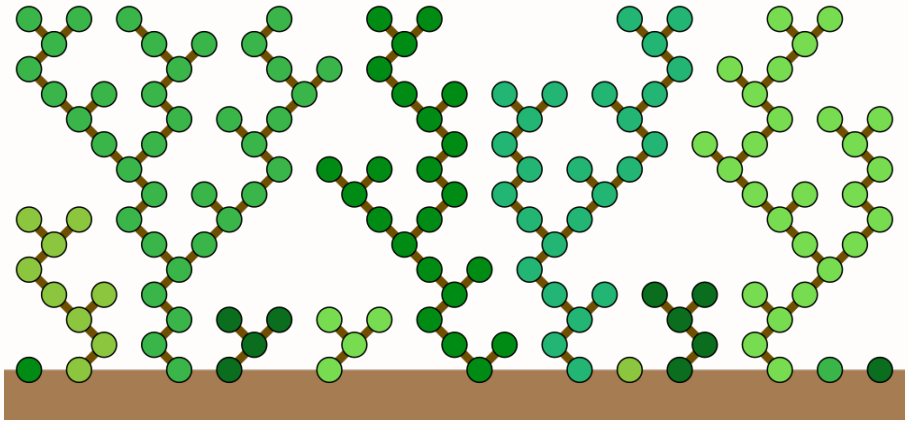

Figure 4.1: **Cellular automata and emerging complexity.** The stunted trees of this image are in reality the evolution of the so-called Rule 90, a one-dimensional cellular automaton. Though simple, even one-dimensional cellular automata can give rise to complex emerging behavior, and can be interpreted to represent the evolution of several natural system, like the growth of a forest of stunted trees. *Source: Rule_90_trees by David Eppstein in the Public Domain.*

display chaotic behaviors (class 3), and that display complex emergent behaviors (class 4).

In 1970, John H. Conway was experimenting with two-dimensional cellular automata and testing different sets of rules. He was looking for some set of rules that could result in an unpredictable complex behavior. The rules he was after should have the following characteristics: They should not give rise to exponential growth, they should allow configurations with unpredictable chaotic outcomes, they should potentially satisfy the von Neumann condition of universal constructor (i.e., to be a self-replicating machine), and they should be the simplest possible rules. The game was first published in 1970 in "Scientific American", in Martin Gardner's mathematical games section [3]. Since its appearance, the *Game of Life*, as it came to be known, has attracted a lot of interest and has been thoroughly studied. It has shown features of emergence of complex behavior and self-organization. It has later been shown to be Turing-complete and capable of simulating a universal constructor or any other Turing machine. Researchers and enthusiasts are actively bringing forward this research nowadays. It has even recently been featured in the "New York Times" [4], in the occasion of the 50 years of Game of Life and 6 months after the death of John Conway, who was about 80 years old and died of covid-19.

In this chapter, we first introduce a simple one-dimensional cellular automaton, which has the advantage of having sets of rules that can be enumerated from 0 to 255; we then focus our attention on some interesting rules (namely, Rule 184, Rule 30, Rule 90, and Rule 110, Fig. 4.1) mentioning their properties and applications. Then, we introduce the definition and properties of the Game of Life, studying its evolution in a few paradigmatic classical cases. We investigate some variations in the rules to understand how sensitive the model is and how different are the outcomes it produces. We also introduce and simulate another two-dimensional cellular automaton, the *majority rule*, which finds application in the study of elections where voters must choose between two candidates, or in modeling of markets with two competing products.

**Example codes:** Example Python scripts related to this chapter can be found on:
https://github.com/softmatterlab/SOCS/tree/main/Chapter_04_Game_of_Life
Readers are welcome to participate in the discussions related to this chapter on:
https://github.com/softmatterlab/SOCS/discussions/12

## 4.1 One-dimensional cellular automata

One-dimensional cellular automata have been studied systematically by Stephen Wolfram [2]. Even though limited to one single dimension and apparently simple, they feature remarkable properties and they have found important applications [5, 6, 7]. The universe of a one-dimensional cellular automaton is composed by cells arranged on an infinite line. Each cell has two neighbors, on the left and on the right. There are, therefore, only $2^3 = 8$ possibilities when considering the state of a cell and its two nearest neighbors. Starting from a given initial configuration, i.e., the *parent generation*, the states of the cells evolve generation after generation. Each automaton is defined by the rules to determine the state of the central cell given the states of the cell itself and its first neighbors: therefore, there are only $2^8 = 256$ different one-dimensional cellular automata. Each of these one-dimensional cellular automata

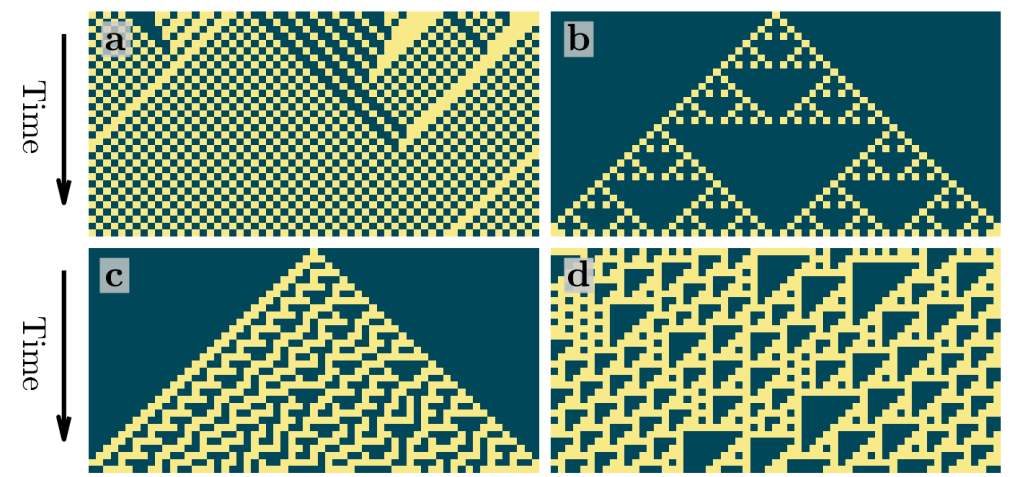


Figure 4.2: **One-dimensional cellular automata.** Examples of time evolution of cellular automata. The yellow cells represent 1s, and the beige cells represent 0s. Each row represents the state of the cellular automaton at a given time, and the time evolution can be seen going down the rows. **a** Rule 184 is a class-2 cellular automaton, because its evolution tends to a state that moves each generation of one step in time (e.g., traffic flow on a single lane, particle deposition on a surface, ballistic annihilation of particles). **b** Rule 90 is a class-4 cellular automaton, because it can show self-similarity and emergent complex behavior. **c** Rule 30 is a class-3 cellular automaton, because it shows a chaotic behavior. It can be used in random number generators. **d** Rule 110 is a very peculiar and unique cellular automaton. It can show emergent complex behavior and chaotic behavior as well. This cellular automaton has been shown to be equivalent to a Turing machine.

can be identified by the decimal number from 0 to 256 corresponding to the final outcomes of the action of the automaton, when acting, respectively, on the base of states 111, 110, 101, 100, 011, 010, 001, 000. Often, the sequence of generations for a one-dimensional cellular automaton is represented on a two-dimensional diagram. Each row represents a generation, starting from the parent generation set in the top row and proceeding downwards with generation 1, 2, 3, and so on (see Fig. 4.2 for some examples of representation). In this section, we will describe four of the most interesting one-dimensional cellular automata, those known as Rule 184, Rule 90, Rule 30, and Rule 110.

**Rule 184.** This is an example of a class-2 cellular automaton, i.e., a cellular automaton that evolves from any initial patterns towards regular patterns that shift of one step each generation. The automaton is defined by the following rules [8]:

| **Current pattern** | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
|---|---|---|---|---|---|---|---|---|
| **New value for center cell** | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |

(4.1)

A typical sequence of generations for rule 184 is given in Fig. 4.2a. This set of rules can be intuitively described as follows: At each step, if a cell with value 1 has a cell with value

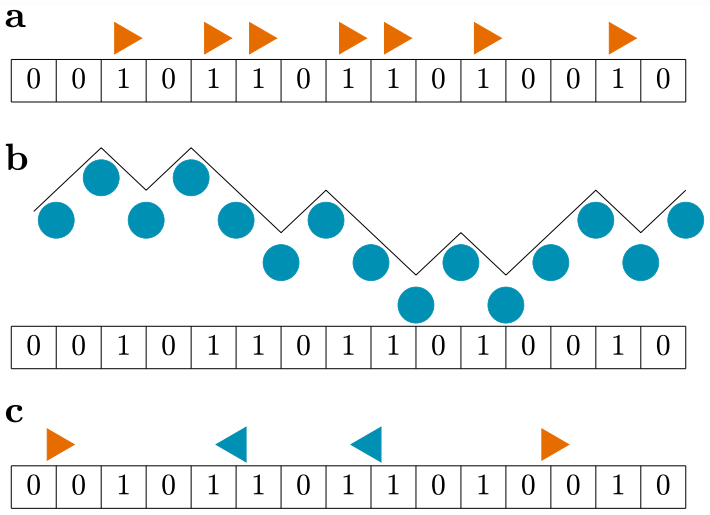


Figure 4.3: **Rule 184.** Rule 184 can be used to represent the time evolution of systems with simple one-dimensional dynamics, such as: **a** Traffic flow in a single lane: Each 1 represents a car trying to go to the right. Only cars with an empty place in front of them will be able to move. **b** Particle deposition: The 0s and 1s represents the boundary of a surface where particles (blue circles) are deposited. The boundary follows the profile of particle accumulation going up and down. In the next generation, new particles can deposit only on the local minima of the boundary, and the boundary is updated accordingly. **c** Ballistic annihilation: Contiguous regions with multiple 0s represent a particle going to the right, while contiguous regions with multiple 1s represent a particle going to the left. Each time step, the particles travel one cell. When they meet, they annihilate, generating an empty space (sequence of alternating single 1-s and 0-es). Interestingly, the ballistic annihilation interpretation can also be used for content-free language parsing: An open parenthesis is a particle travelling to the right, and a closed parenthesis is a particle travelling to the left. A well-structured expression is made by an equal number of open and closed parentheses; therefore, it is possible to check the syntax of the parentheses in a regular expression by letting its representation evolve with rule 184, and see if it arrives to a state where all the parentheses have annihilated.

0 immediately to its right, the 1 moves rightwards leaving a 0 behind. A 1 with another 1 to its right remains in place, while a 0 that does not have a 1 to its left stays a 0. Rule 184 can represent the evolution of several systems. For example, it can provide a simple model for the traffic flow in a single lane (Fig. 4.3a), for the deposition of particles on an irregular surface (Fig. 4.3b), and for the ballistic annihilation of particles in one dimension (Fig. 4.3c).

**Rule 90.** This is an example of a class-4 cellular automaton, which gives rise to emergent complex behavior. It is defined by the following rules [9]:

| **Current pattern** | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
|---|---|---|---|---|---|---|---|---|
| **New value for center cell** | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |

(4.2)

The state of each cell is determined solely by the states of its neighbors, disregarding the state of the cell itself. The rule is actually implementing a XOR (exclusive or) of the state of the neighboring cells. It can describe the growth of a forest of stunted trees (see Fig. 4.1). Also, it features self-replication: For example, when starting from an initial condition of with a single 1, the time evolution of Rule 90 leads to a fractal pattern (the *Sierpinski triangle*, Fig. 4.2b).

**Rule 30.** This is an example of a class-3 cellular automaton, whose evolution leads to a chaotic pattern. This cellular automaton is defined by the following rules [10]:

| **Current pattern** | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
|---|---|---|---|---|---|---|---|---|
| **New value for center cell** | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |

(4.3)

A typical sequence of generations for Rule 30 is shown in Fig. 4.2c. This cellular automaton exhibits features of aperiodicity and chaotic behavior, which have led to its application in random number generation and cryptography [11]. Furthermore, its time evolution represented in two dimensions resembles the pattern found on shells, like those of *Conus textile* sea snails.

**Rule 110.** This unique cellular automaton can be considered a class-4 cellular automaton, but it also shares features of a class-3 cellular automaton. It is defined by the following rules [12]:

| **Current pattern** | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
|---|---|---|---|---|---|---|---|---|
| **New value for center cell** | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |

(4.4)

This cellular automaton is unique for its behavior at the boundary between stability and chaos, and for its characteristic of *universality*, i.e., it is Turing complete, which means that it can be used to simulate any Turing machine, as demonstrated by Matthew Cook [13].

**Exercise 4.1: One-dimensional cellular automata.** Write a code that implements the one-dimensional cellular automata described in the text above. Given the number of the rule, it should produce an animation of the corresponding cellular automaton.

**a.** Run the simulation for the cellular automata described in the text, namely Rule 184, Rule 90, Rule 30, and Rule 110. *[Hint:Compare your outcome with Fig. 4.2.]*

**b.** Experiment with other rules! Try to find cellular automata corresponding to each of the four classes identified by Stephen Wolfram.

## 4.2 Conway's Game of Life

The set of rules of Conway's Game of Life are pretty simple [3]. On an infinite two-dimensional board, the cells have only two states, 1 (or *on*, *alive*) and 0 (or *off*, *dead*).

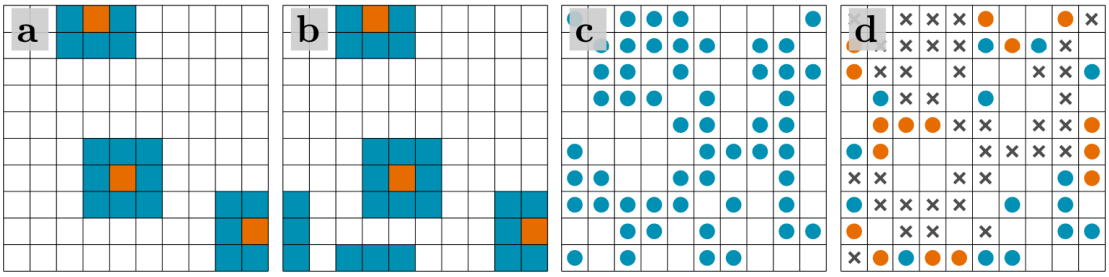


Figure 4.4: **Game of life rules.** A cell (orange) and its first Moore's neighbors (cyan) **a** without and **b** with periodic boundary conditions. **c** Example of a generation and **d** of its subsequent generation obtained applying the rules 4.5: Cells with crosses have died, cells in orange have been newly born, and cells in cyan survived from the previous generation.

.

Initially, at generation 0, a certain number of cells is alive and the rest is dead. The next generation is calculated from the former generation by applying the following rules:

Dead cell

| **Number of alive neighbors** | **Next generation** |
|---|---|
| $1, 2$ | remains dead |
| $3$ | a new alive cell is born |
| $4, 5, 6, 7, 8$ | remains dead |

(4.5)

Alive cell

| **Number of alive neighbors** | **Next generation** |
|---|---|
| $1$ | dies (loneliness) |
| $2, 3$ | remains alive |
| $4, 5, 6, 7, 8$ | dies (overpopulation) |

When considering the number of neighbors, the *Moore neighborhood* is employed (Figs. 4.4a and 4.4b).

**Exercise 4.2: Game of life.** Implement the Game of Life on a square grid $10 \times 10$. Start from a random configuration and proceed for 20 generations.

**a.** Structure your program using a function that, for every generation, finds the number of alive neighbors for each cell, and a function that calculates the next generation on the basis of the rules 4.5. Assume non-periodic boundary conditions (i.e., a cell on the boundary has less than 8 neighboring cells, as shown in Fig. 4.4a).

**b.** Visualize your output for each generation (Figs. 4.4c-d) and check that every cell is updated correctly from one generation to the following one.

**c.** Modify your program to find the number of neighbors of each cell for the case of periodic boundary condition (Fig. 4.4b).

Let us start by applying the rules 4.5 to some simple configurations. If, for example, we start from the *block*, shown in Fig. 4.5a, we find that the next generation is exactly equal to

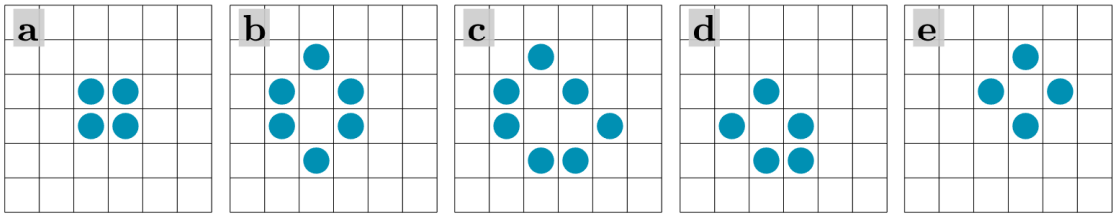


Figure 4.5: **Still life.** Examples of still life: **a** the block, **b** the beehive, **c** the loaf, **d** the boat, and **e** the tub. The next generation originated by these shapes does not change. Can you think of other still-life examples?

.

the parent generation. Shapes that have this property are called *still life*. In Fig. 4.5 you can find the most common still life examples.

**Exercise 4.3: Time evolution of a still life.** For each still life represented in Fig. 4.5, write a function that reproduces its time evolution.

If, instead, we start from the *blinker* (Fig. 4.6a), the next generation (Fig. 4.6b) is different from the parent, but the second generation returns the original form (Fig. 4.6c). This kind of periodical configurations are called *oscillators*. Fig. 4.6 represents the most common oscillators with period two. There are also oscillators with period greater than two. Can you think of some examples?

**Exercise 4.4: Time evolution of an oscillator.** For each oscillator represented in Fig. 4.6, write a function that reproduces its time evolution.

Still lifes and oscillators are configurations that do not move. The simplest configuration capable of motion on the Game of Life universe is the *glider* (Fig. 4.7). As you can see, after four generations the glider has regained its initial form, but its position is shifted diagonally of one step. Its velocity is $v = \frac{1}{4}c$, where $c$ is the velocity of one cell per generation. In general, if after $N$ generations a shape has regained its original form but shifted by $M$ cells, then its velocity is $v = \frac{M}{N}c$. A velocity of $c$, i.e., one cell per second, cannot be obtained by any object (why? and how close is it possible to get?).

**Exercise 4.5: Time evolution of a glider.** Implement the glider represented in Fig. 4.7.

**a.** Test your function by checking the time evolution of the glider between five successive generations. In which direction does your glider move?

**b.** Implement a glider moving along for each diagonal direction.

Gliders are not the only movable objects in the Game of Life universe. There are several other objects, called *spaceships* that move. Each spaceship has its own period and its own

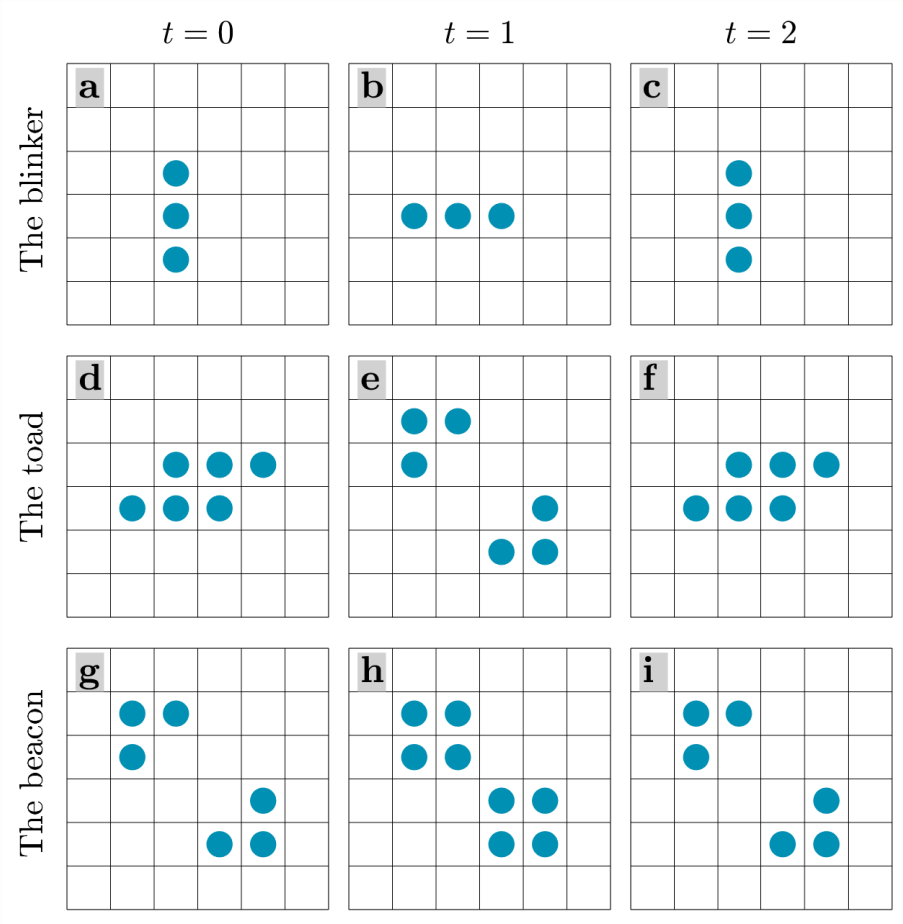


Figure 4.6: **Oscillators.** Examples of oscillators: **a**-**c** the blinker, **d**-**f** the toad, and **g**-**i** the beacon. Three generations are shown, indicated by the timestamp. After two generation, the initial shape repeats itself.

.

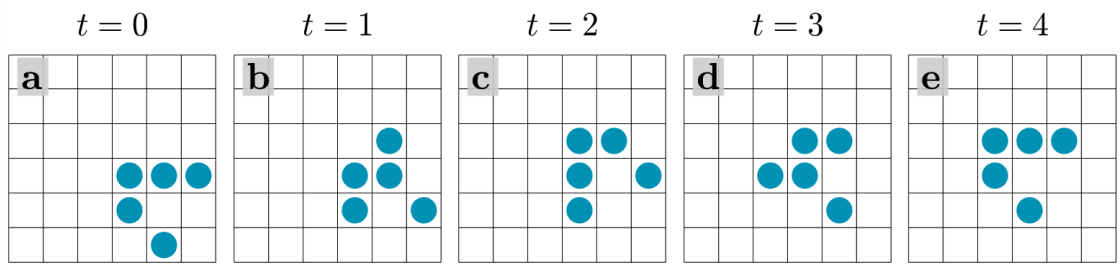


Figure 4.7: **Glider.** The glider regains its original shape after four generations, but it is shifted diagonally with respect to initial position.

.

velocity. Gliders, among the spaceships, have a particular importance because they are very simple objects and it has been shown that, for any stable configuration, there is a finite number of gliders coming from a faraway distance that can build that configuration starting from an empty board, and there is also a constructive procedure to determine the initial configuration of the gliders [14, 15, 16].

**Exercise 4.6: A quest for new oscillators and gliders.** How can you find new oscillators, gliders, spaceships, contained in a square $N \times N$? One way is to take a board with size $3N \times 3N$, generate a random configuration $N \times N$ at its center, and let it evolve for $K$ generations. If the generation $k$ is the first generation to be identical to the starting configuration, then you have found an oscillator with period $k$. If the generation $k$ is shifted version of the initial configuration, shifted by $s_x$ cells along the $x$ direction and $s_y$ cells along the $y$ direction, then you have found a spaceship with velocity $v = \frac{s}{k}c$, with $s = \sqrt{(s_x^2 + s_y^2)}$.

**a.** Implement a function that, given a configuration, translates it of $s_x$ cells along the $x$ direction and $s_y$ cells along the $y$ direction, and a function that checks if two configurations are identical.

**b.** Implement the algorithm to look for new oscillators and spaceships, starting from a random configuration.

**c.** How easy is it to find a new oscillator or a new spaceship using this procedure? Is a generic random configuration easily evolving to a stable one? How can you improve the algorithm to find new spaceships or oscillators?

Initially, John H. Conway was convinced that there could be no configuration giving rise to a continuous growth of the number of alive cells. However, this conjecture proved to be wrong: A team led by Bill Gosper found a configuration called the *Gosper glider gun* (Fig. 4.8), which produces its first glider at the $15^{\text{th}}$ generation and then one every 30 generations. After that discovery, several other growing configurations have been found. The taxonomy of the entities in the Game of Life includes now objects like *guns* (objects shooting new gliders), *puffer trains* (objects leaving a trail of debris behind them), *rakes* (objects producing spaceships), *breeders* (objects leaving behind a trail of guns), and so on. Also, the Game of Life has been shown to be Turing complete [17].

**Exercise 4.7: Alternative rules for the Game of Life.** Modify the rules 4.5 and explore the features of the ensuing modified Game of Life.

**a.** Starting from a random configuration, how does the evolution look like? Do the modified rules give rise to emergent complexity? or does the system evolve towards a stable pattern? Does the system evolve towards overpopulation or towards extinction?

**b.** Explore several possibilities. Find at least one non-trivial set of rules leading to extinction and one leading to a fixed or oscillating stable pattern.

**c.** How likely is that complexity emerges with a random set of rules?

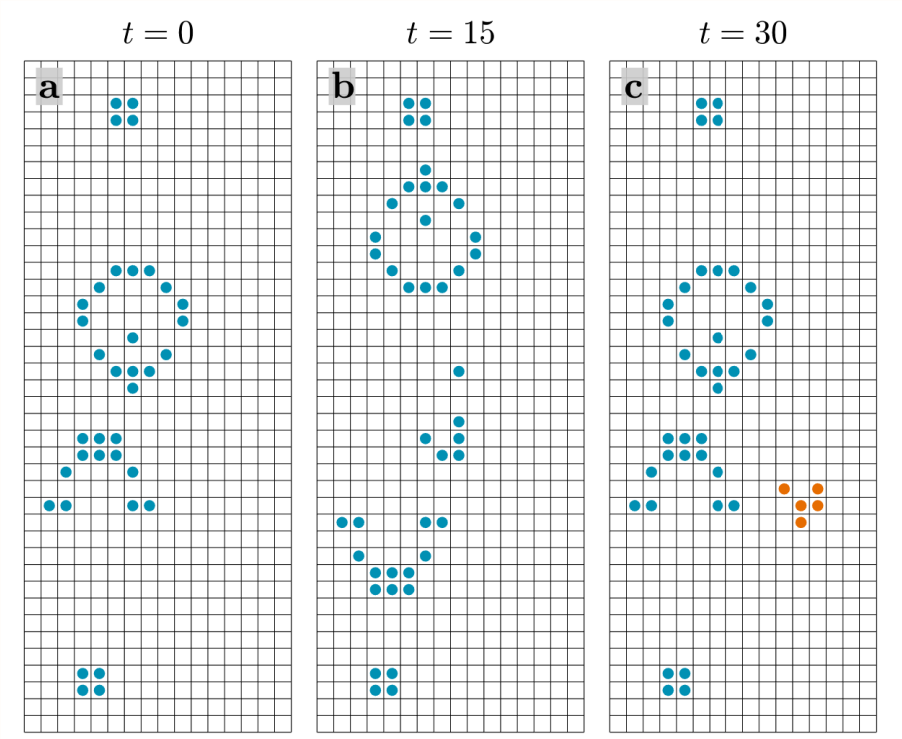


Figure 4.8: **Gosper glider gun. a** Parent generation, **b** $15^{\text{th}}$ generation, and **c** $30^{\text{th}}$ generation of a Gosper glider gun. At the $15^{\text{th}}$ generation a glider is emitted. Comparing **a** and **c**, the two configurations are identical, with the exception of the generated glider (drawn in orange in **c**), which in the meanwhile has translated diagonally towards South-East. The Gosper glider gun is ready to generate another glider at the $45^{\text{th}}$ generation, and the process continues forever.

.

## 4.3 Majority rule

Another interesting set of rules is the so-called *majority rule* [18]. This rule does not give rise to complex dynamics, but it has interesting applications to elections where voters must choose between two parties (or to markets where consumers must choose between two competing brands). Thus, we can suppose that each cell represents an elector, and the state 0 or 1 represent a vote for one of the two parties. The voting process is iterative, and each elector can reconsider their vote on the basis of the votes of their neighbors. Each iteration, electors change their vote to the vote expressed by the majority of their Moore's first neighbors. If no majority exists, then the electors preserve their former vote. The rules in this system are

therefore the following:

| Number of neighbors voting 1 | Next generation |
|---|---|
| $0,1,2,3$ | vote 0 |
| 4 | vote does not change |
| $5,6,7,8$ | vote 1 |

(4.6)

**Exercise 4.8: Majority rule.** Modify your code for the Game of Life by introducing the majority rule 4.6. Start with a square board $100 \times 100$ with a random configuration where the initial fraction of 1 votes is $p$. For example, start with $p = 0.45$.

**a.** Find the final state of the system: After how many iterations the configuration becomes stable? What is the outcome of the vote? How are the two domains of voters spatially distributed?

**b.** Run your simulation for different initial distributions of votes by varying $p$. Record the outcome of each election, after the configuration stabilizes. Record also

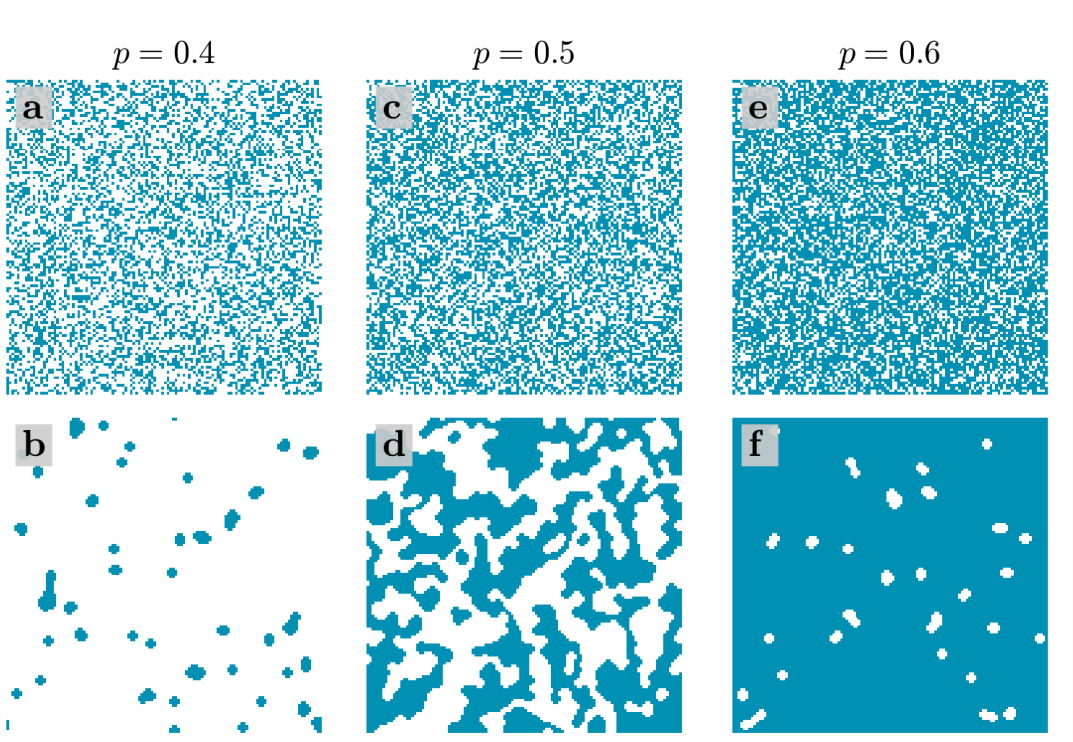


Figure 4.9: **Majority rule.** Initial (upper row) and final (lower row) states for a majority rule game with an initial distribution of votes 1 with a probability **a**-**b** $p = 0.4$, **c**-**d** $p = 0.5$, **e**-**f** $p = 0.6$. One can see that the vote remains balanced with $p = 0.5$. For $p \neq 0.5$, instead, the majority rule leads the vote to a stable outcome where the disproportion between the two factions is much larger than the initial one.

.

the number of iterations needed to stabilize. Compare your results with those shown in Fig. 4.9.

Related to the majority rule, it is worth mentioning the *majority problem*, which is the problem of finding a one-dimensional cellular automaton correctly classifying a majority voting, i.e., to decide whether the majority of the votes is 1 or 0 on the base of local rules only. As Fig. 4.9 suggests, the majority rule seems to be a very good candidate to classify the outcome of a binary election, and it works very well when the initial proportion of 0s and 1s is not too close to 50%. However, it has been shown that it is not a perfect classifier [19], especially in the case of a partition of votes that is very close to 50%. For example, for configurations that differ by only one vote and the deciding vote is surrounded by opposite votes, the deciding vote is flipped by the majority rule. Moreover, it has been shown that it is impossible to have a perfect majority classifier, either deterministic or stochastic, for any dimension [20].

## 4.4 Further readings

The literature on the Game of Life and cellular automata is vast. Online resources for simulations are very easily found. Among the many, it is worth mentioning the websites LifeWiki [21] and ScholarPedia [22], a website demonstrating various cellular automata [23], and a resource for the hexagonal Game of Life [24].

The first public appearance of Conway's Game of Life is in an article of Martin Gardner on Scientific American [3]. Moreover, important are the contributions of Stephen Wolfram to the field of cellular automata and artificial life and complexity. The highly recommended Refs. [25, 2] provide the context of the research on cellular automata, its possible applications, and its open challenges.

Regarding the historical and theoretical context of the mathematical definition of life and the foundation of the theory of self-replicating machines and cellular automata, one can refer to Refs. [1, 26]. It is worth reading the article about the contributions of John von Neumann [27] to the field of artificial life. This article also gives an all-round picture of the scientist and person.

## Bibliography

[1] J. Von Neumann and A. W. Burks. *Theory of Self-Reproducing Automata*. University of Illinois Press, 1966.

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

[3] M. Gardner. Mathematical games – the fantastic combinations of John Conway's new solitaire game 'life'. *Sci. Am.*, 223:120, 1970.

[4] S. Roberts. The lasting lessons of john conway's game of life. https://www.nytimes.com/2020/12/28/science/math-conway-game-of-life.html, 2020.

[5] S. Wolfram. Statistical mechanics of cellular automata. *Rev. Mod. Phys.*, 55:601, 1983.

[6] S. Wolfram. Cellular automata as models of complexity. *Nature*, 311:419, 1984.

[7] S. Wolfram. Cellular automaton fluids 1: Basic theory. *J. Stat. Phys.*, 45:471, 1986.

[8] S. Wolfram. rule 184 - wolfram—alpha. https://www.wolframalpha.com/input/?i=rule+184.

[9] S. Wolfram. rule 90 - wolfram—alpha. https://www.wolframalpha.com/input/?i=rule+90.

[10] S. Wolfram. rule 30 - wolfram—alpha. https://www.wolframalpha.com/input/?i=rule+30.

[11] S. Wolfram. Random sequence generation by cellular automata. *Adv. Appl. Math.*, 7:123, 1986.

[12] S. Wolfram. rule 110 - wolfram—alpha. https://www.wolframalpha.com/input/?i=rule+110.

[13] M. Cook. Universality in elementary cellular automata. *Complex Syst.*, 15:1, 2004.

[14] Conwaylife, glider synthesis. https://conwaylife.com/wiki/Glider_synthesis.

[15] A. Adamatzky. *Game of Life Cellular Automata*. Springer, 2010.

[16] E. R. Berlekamp, J. H. Conway, and R. K. Guy. *Winning Ways for Your Mathematical Plays*. Taylor and Francis, 2004.

[17] A. Adamatzky, editor. *Collision-Based Computing*. Springer-Verlag, Berlin, Heidelberg, 2001.

[18] B. N. Gärtner and Zehmakan A. Color war: Cellular automata with majority-rule. In F. Drewes, C. Martín-Vide, and Truthe B., editors, *Language and Automata Theory and Applications.*, volume 10168 of *Lecture Notes in Computer Science*, page 393. Springer, Cham., 2017.

[19] M. Land and R. K. Belew. No perfect two-state cellular automata for density classification exists. *Phys. Rev. Lett.*, 74:5148, 1995.

[20] A. Bušić, N. Fatès, J. Mairesse, and I. Marcovici. Density classification on infinite lattices and trees. *Electron. J. Probab.*, 18:1, 2013.

[21] Lifewiki. https://www.conwaylife.com/wiki/Main_Page.

[22] Scholarpedia. http://www.scholarpedia.org/article/Game_of_Life.

[23] J. Wezorek. A gallery of cellular automata created via artificial selection using lifelike. http://jwezorek.com/CA/ca_gallery.html.

[24] S. Ingram, J. Lee, and A. Arjunakani. Hexagonal game of life. https://arunarjunakani.github.io/HexagonalGameOfLife/.

[25] S. Wolfram. Twenty problems in the theory of cellular automata. *Phys. Scr.*, T9:170, 1985.

[26] G. J. Chaitin. To a matematical definition of life. *ACM SICACT News*, page 12, 1970.

[27] P. Marchal. John vonNeumann: The founding father of artificial life. *Artif. Life*, 4:229, 1998.

[28] C. Bays. Candidates for the game of life in three dimensions. *Complex Syst.*, 1:373, 1987.

[29] C. Bays. A note about the discovery of many new rules for the game of three-dimensional life. *Complex Syst.*, 16:381, 2006.