

# Natural Computing SoSe25







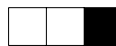
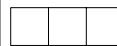








Exercise Sheet 1 — May 15, 2025

Thomas Gabor, Maximilian Zorn











## 1 Cellular Automata

In the lecture we have seen and defined 1D cellular automata (CA) with only two states (0 or 1), which evolve over generations according to a predefined rule.

(i) As you might recall from the lecture, a cellular automaton's function  $f$  is often given as the template below, which shows the inputs to  $f$  on the top row and the respective output on the bottom row. Give a concise mathematical definition for  $f$  according to the specific template below.

(ii) For the following initial configuration, draw one configuration that cannot result from the initial configuration through any cellular automaton following the template of task (i). State your reasoning.

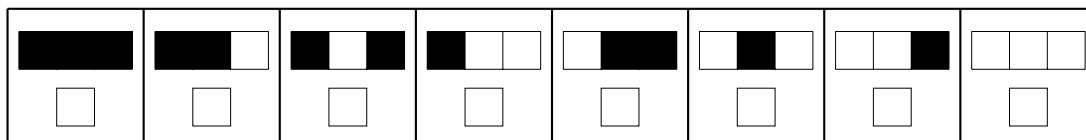
				
				

**Wolfram code** is a scheme proposed by Stephen Wolfram and the de facto standard for describing elementary cellular automata. A *code* assigns to each template an integer from 0 to 255 as *rule*  $n \in [0; 255] \subset \mathbb{N}$  where each digit in  $n$ 's binary representation encodes one output of  $f$  so that the digit for the base value 128 encodes the output of  $f(1, 1, 1)$ , 64 for  $f(1, 1, 0)$ , and so on, sorted by decreasing numerical value. This number is taken to be the *rulestring* of the automaton.<sup>1</sup>

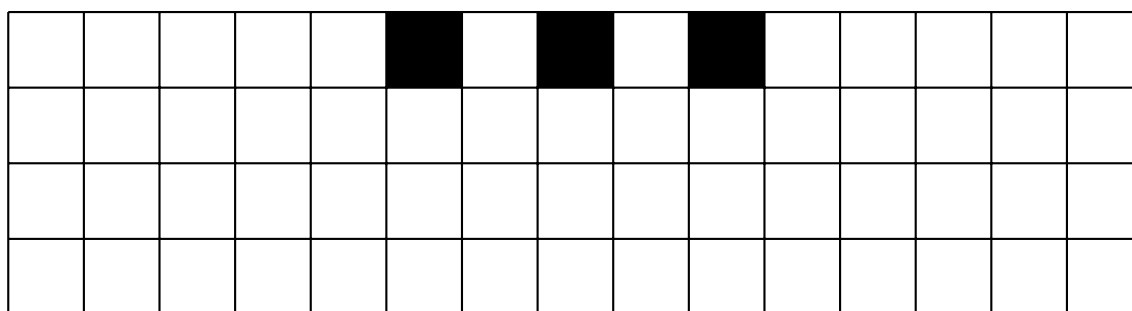
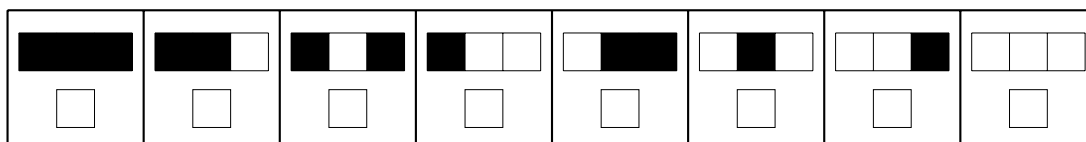
(iii) Given the rulestring 255, fill in the rule template below. With a three-cell neighborhood, each with two possible states, in how many ways can we configure the rule-template and in how many ways can we set the CA initial configurations for a board of width  $w$ ?

Possible rule configurations:

Possible initial board states:

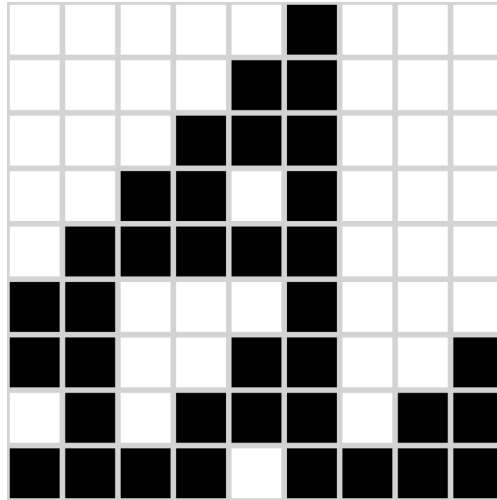


(iv) Fill in the rulestring 102, then evolve this initial configuration by hand for three generations.



<sup>1</sup>[https://en.wikipedia.org/wiki/Wolfram\\_code](https://en.wikipedia.org/wiki/Wolfram_code)

(v) Observe the evolution of a new cellular automaton below, showing one generation per row. Give a full definition of this automaton's function  $f$ . You may use the empty rule template printed below. Is the given evolution sufficient to fully define  $f$ ? Why or why not?



■ ■ ■ ■ ■	■ ■ ■ ■ □	■ □ ■ ■ ■	■ □ □ □ □	□ ■ ■ ■ ■	□ ■ ■ □ □	□ □ □ ■ ■	□ □ □ □ □
□	□	□	□	□	□	□	□

## 2 Wolfram's CA Game

Let's put the basics of Task 1 into practice and code a 1d CA evolution game. For a code template consider the included `ca_game.py` file.

(i) In the `*** marked sections ***`, code the rule application and neighborhood check to complete your 1D cellular automaton game. You may consult the respective chapter in “The Nature of Code”<sup>2</sup> if necessary, but you will have to translate Java to Python. If your code is correct, a `pygame` window will open that repeatedly completes the evolution.

(ii) In your rule application, is it possible to directly update the `cells` list in place? What happens if you do so?

(iii) (Bonus) *In a cellular automaton, a **Garden of Eden** is a configuration that has no predecessor. It can be the initial configuration of the automaton but cannot arise in any other way. [...] [F]or any Garden of Eden there is a finite pattern (a subset of cells and their states, called an **orphan**) with the same property of having no predecessor, no matter how the remaining cells are filled in. A configuration of the whole automaton is a Garden of Eden if and only if it contains an orphan.*<sup>3</sup>

For 1D cellular automata the existence of Gardens of Eden may be checked by brute force. Change your code to iterate every possible initial state for all 256 rule numbers, keeping track of all states seen per each rule number. (You may reduce the number of cells by increasing `cell_size=50`.) How many evolution steps from every initial state do you need to check for a Garden of Eden? Output for each rule number, how many Gardens of Eden you have found. Why are there trivially no “general Gardens of Eden” that cannot be reached by any rule?

---

<sup>2</sup><https://natureofcode.com/book/chapter-7-cellular-automata/>

<sup>3</sup>[https://en.wikipedia.org/wiki/Garden\\_of\\_Eden\\_\(cellular\\_automaton\)](https://en.wikipedia.org/wiki/Garden_of_Eden_(cellular_automaton))