SHARIF UNIVERSITY OF TECHNOLOGY
PHYSICS DEPARTMENT

# Problem Set 6: Cellular Automata

Student name: *Ali Fayyaz - 98100967*

Course: *Complex Systems - (Spring 2023)*
Due date: *June 8, 2023*

### Exercise 18

**Cellular Automata and Wolfram's Classification**

As you may know, we can write 256 rules as an 8-bit number. Using 1 bit flips in the binary representation of rule number 110, we can come up with 8 images. Generate these images (take the length of the system to be 201 and simulate for 300 time steps). Explain how these CAs are classified according to Wolfram's Classification.

**Answer.** A function was defined to generate a cellular automaton with periodic boundary conditions as follows:

- **CA_generator():**

  This function takes as input a string of 0's and 1's as the initial condition of the automaton, number of time steps to simulate the automaton, and the rule in its binary representation. It returns a matrix, each row of which contains one time step of the simulation.

The function was used alongside *matplotlib.pyplot.imshow()* to visualize the evolution of automatons through time. To do so, for each rule, as specified in the question, two initial conditions were studied and plotted: the first initial condition consisted of all cells being *off* (or *dead*), except for the middle cell, the other initial condition began with a random *generation zero*, which means some of the 201 cells were *on*, with a probability of $\frac{1}{2}$, and the others were *off*.

Wolfram classifications are as follows:

1. **Class One:**

   Cellular automata which rapidly converge to a uniform state: Rules 78 and 238

2. **Class Two:**

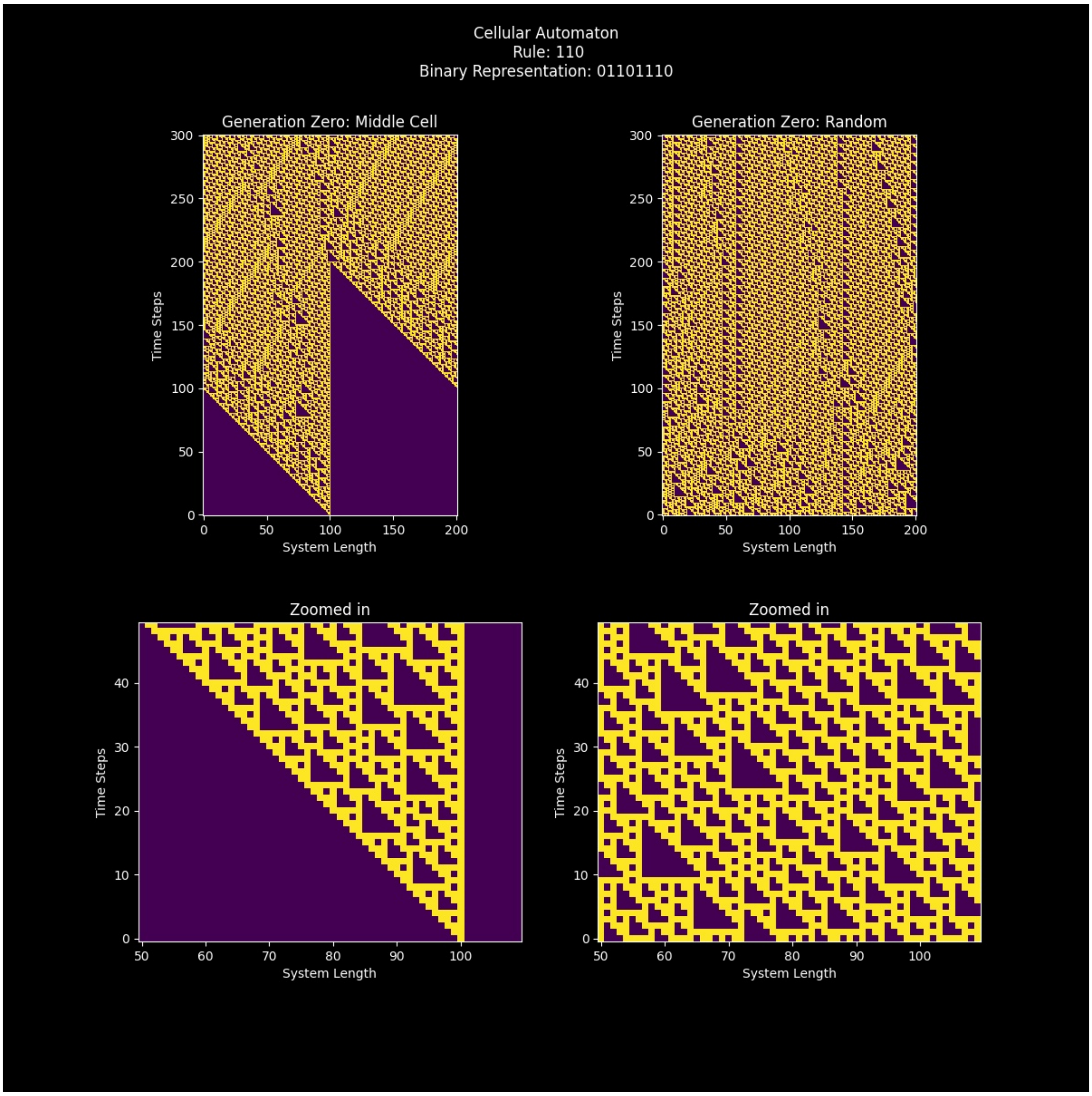
   Cellular automata which rapidly converge to a repetitive or stable state: Rules 46 and 108

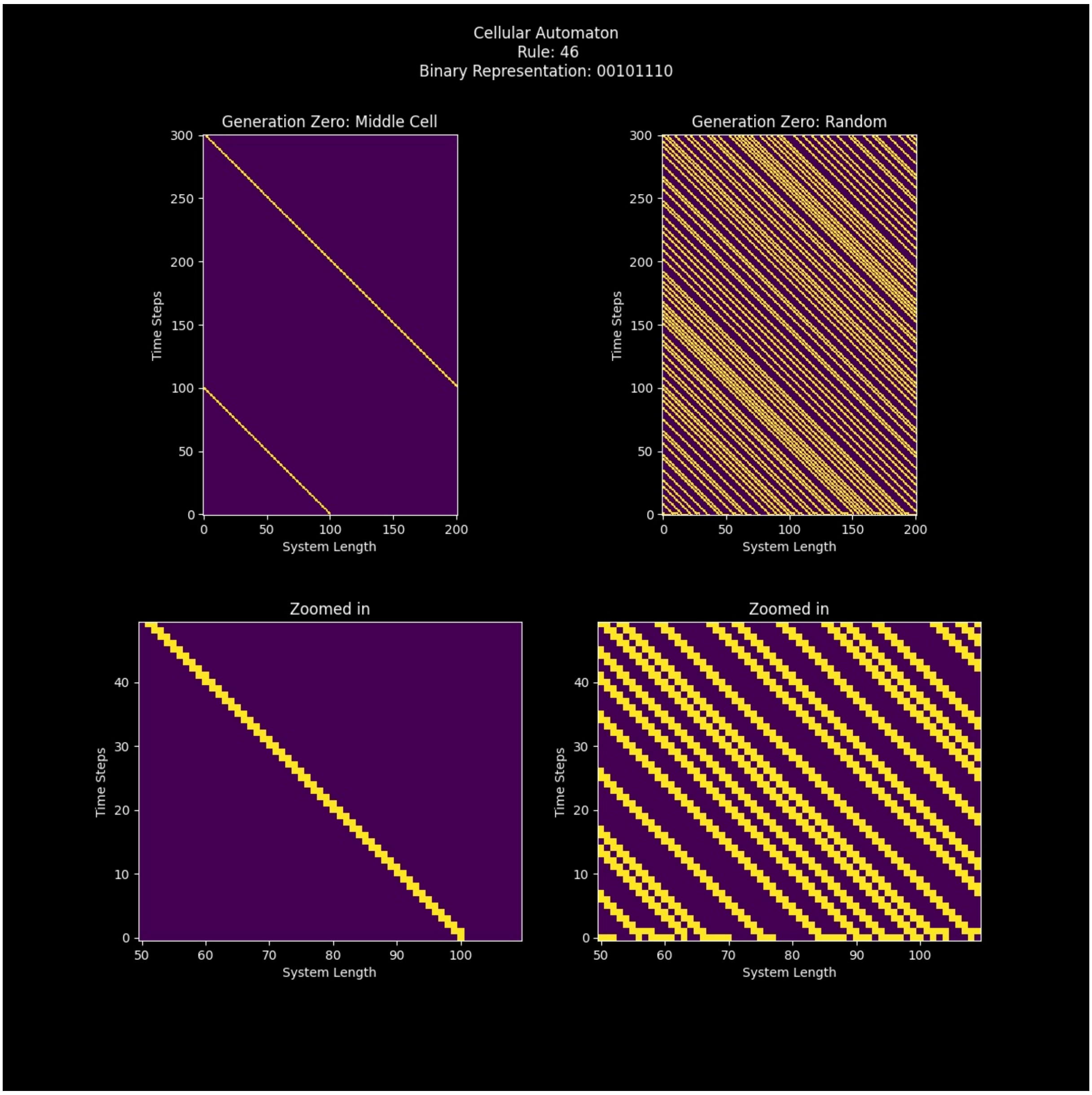
3. **Class Three:**

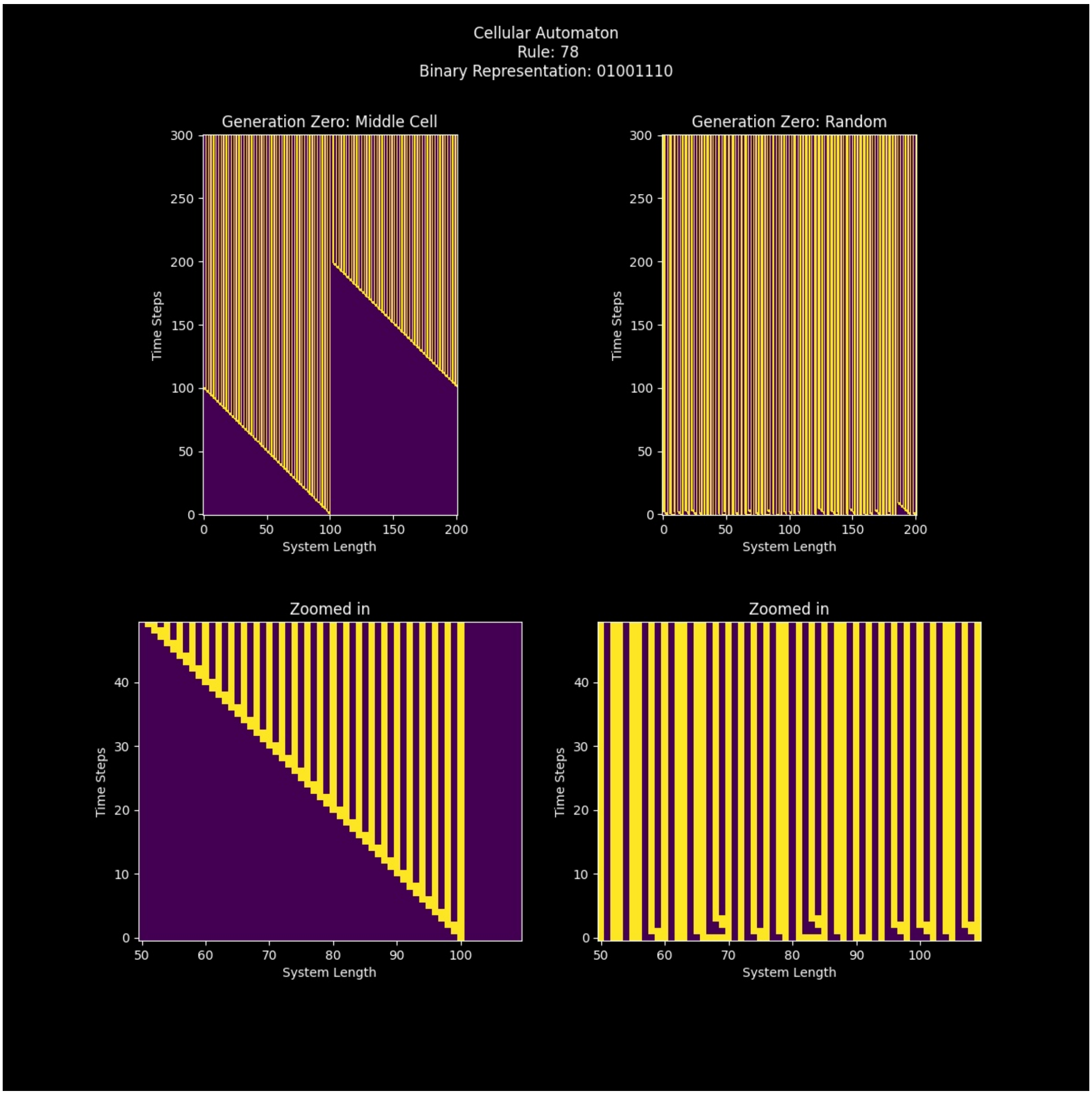
   Cellular automata which appear to remain in a random state: Rules 102, 106, 111 and 126

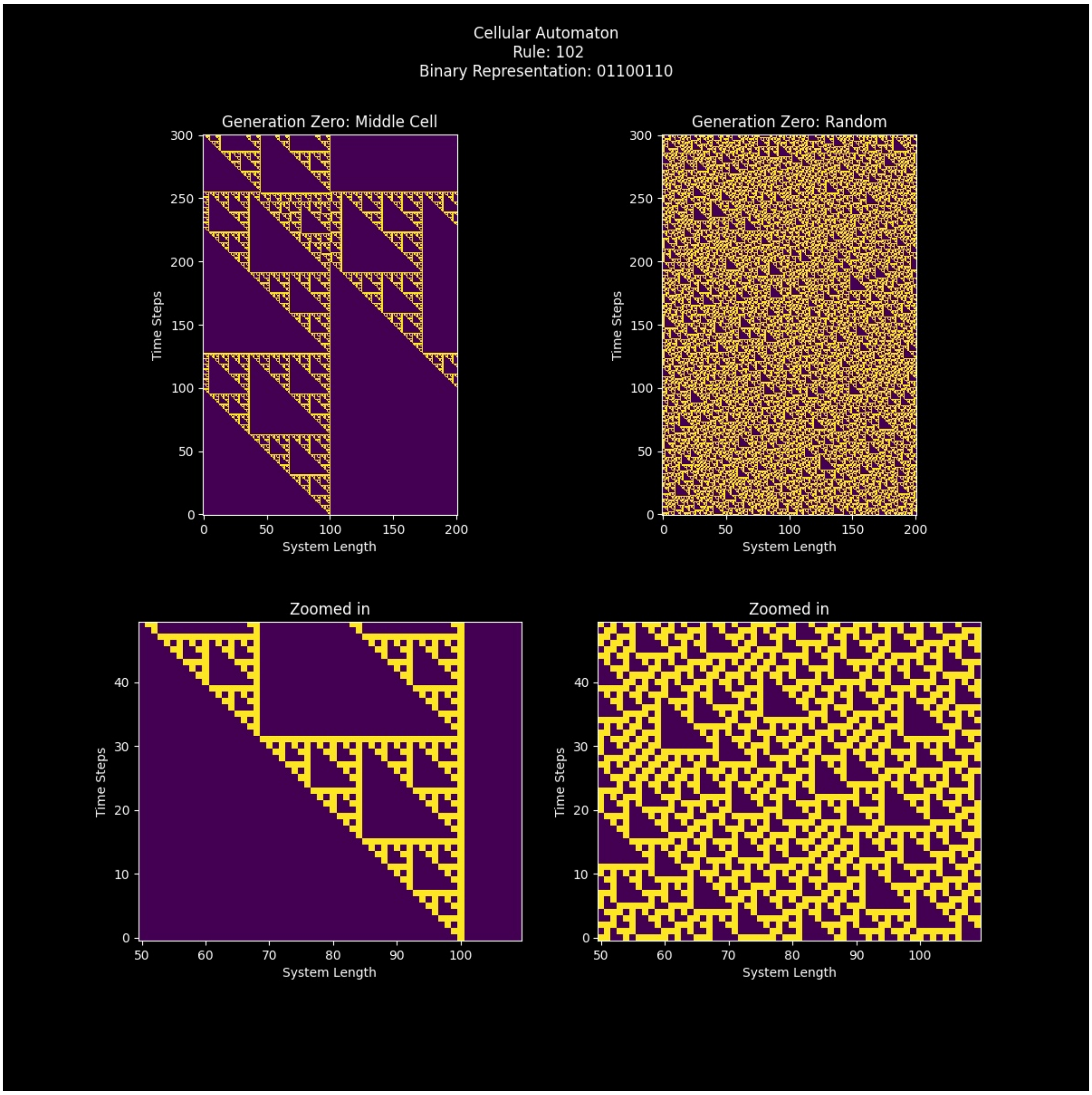
4. **Class Four:**

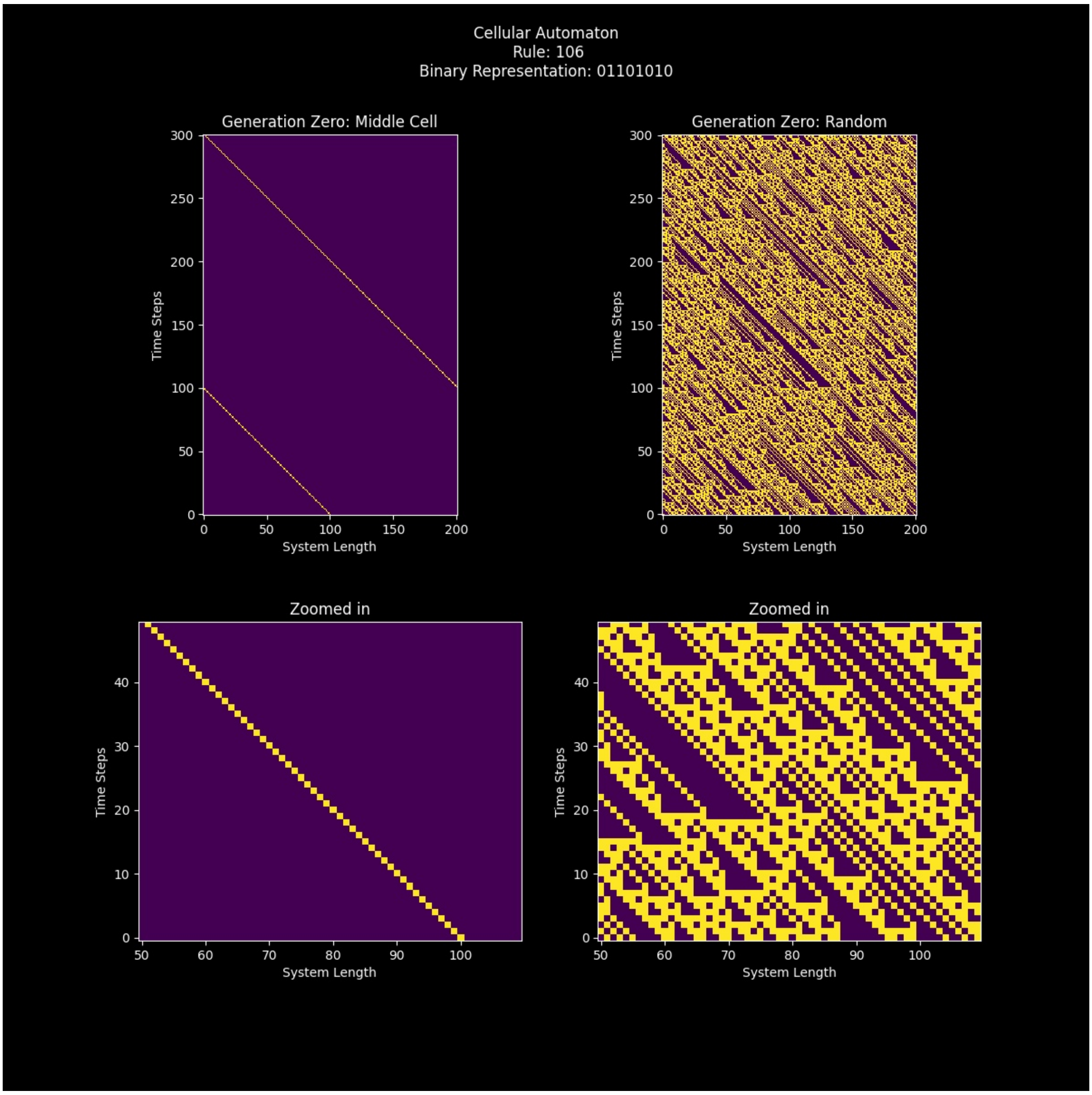
   Cellular automata which form areas of repetitive or stable states, but also form structures that interact with each other in complicated ways: Rules 110



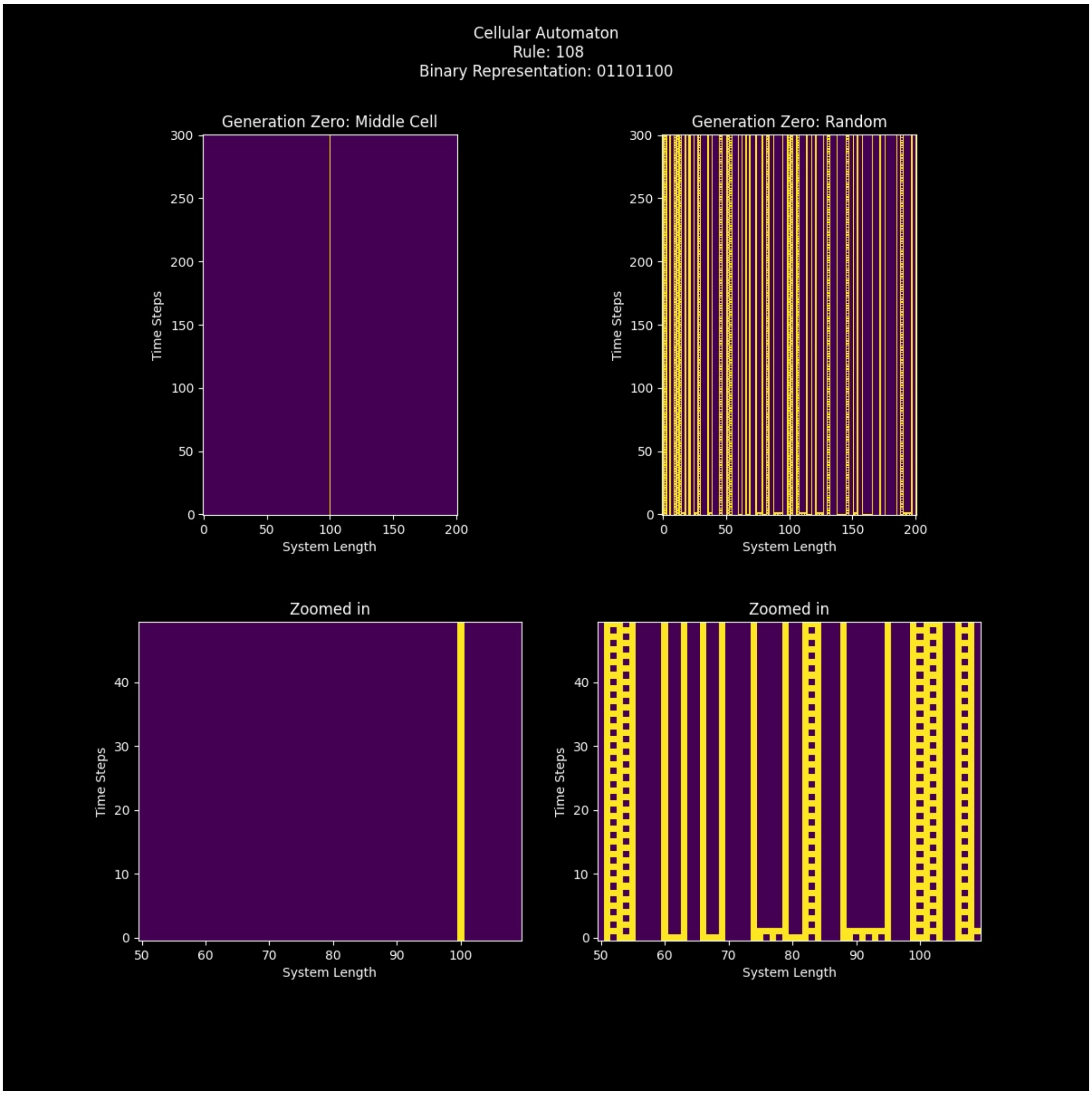

Cellular Automaton
Rule: 46
Binary Representation: 00101110
Generation Zero: Middle Cell
Generation Zero: Random
Zoomed in
Zoomed in
Time Steps
System Length


Cellular Automaton
Rule: 78
Binary Representation: 01001110
Generation Zero: Middle Cell
Generation Zero: Random
Zoomed in
Zoomed in
Time Steps
System Length
0
50
100
150
200
250
300
10
20
30
40
60
70
80
90

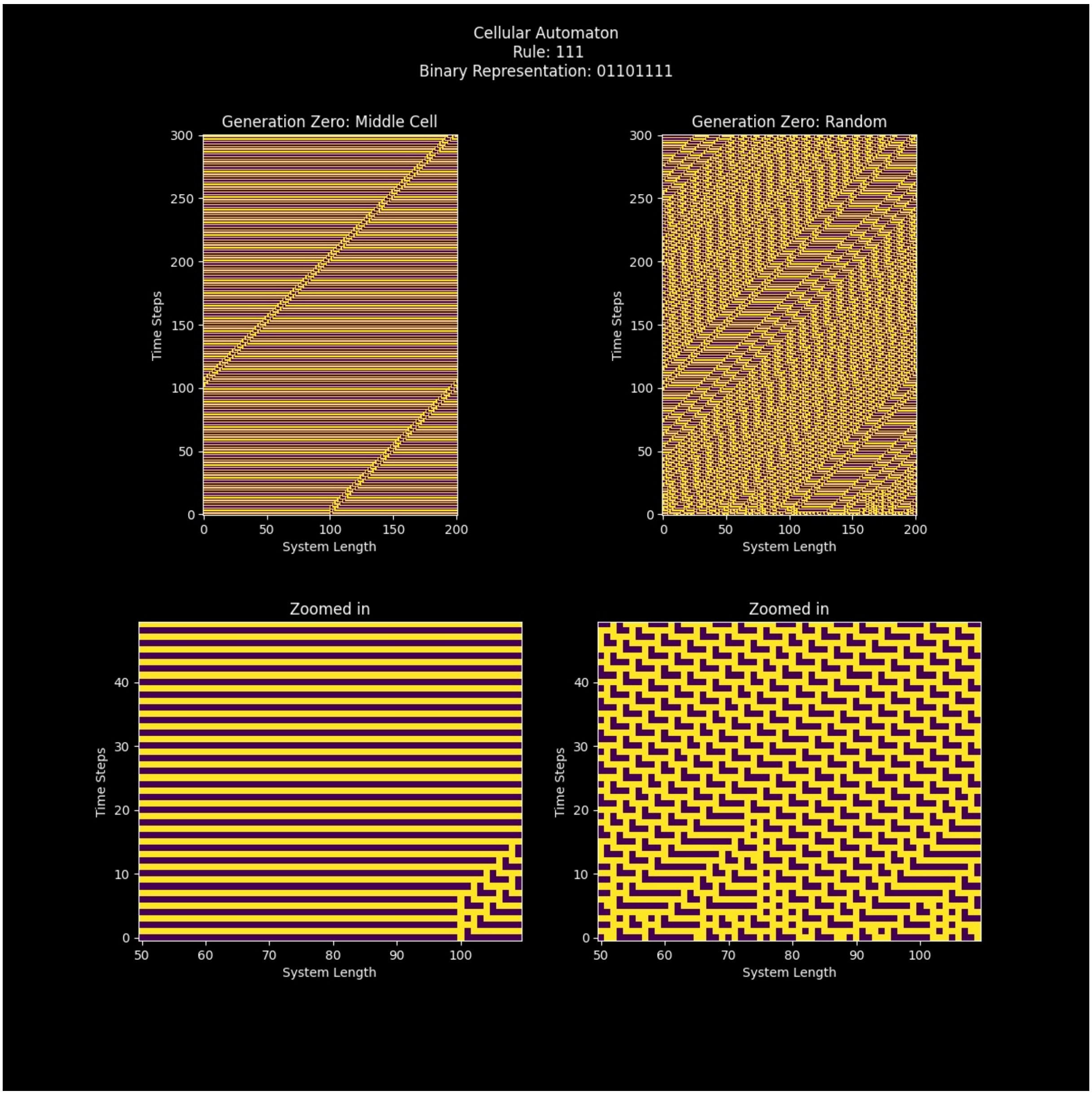

Cellular Automaton
Rule: 102
Binary Representation: 01100110
Generation Zero: Middle Cell
Generation Zero: Random
Time Steps
System Length
Zoomed in
Zoomed in
Time Steps
System Length



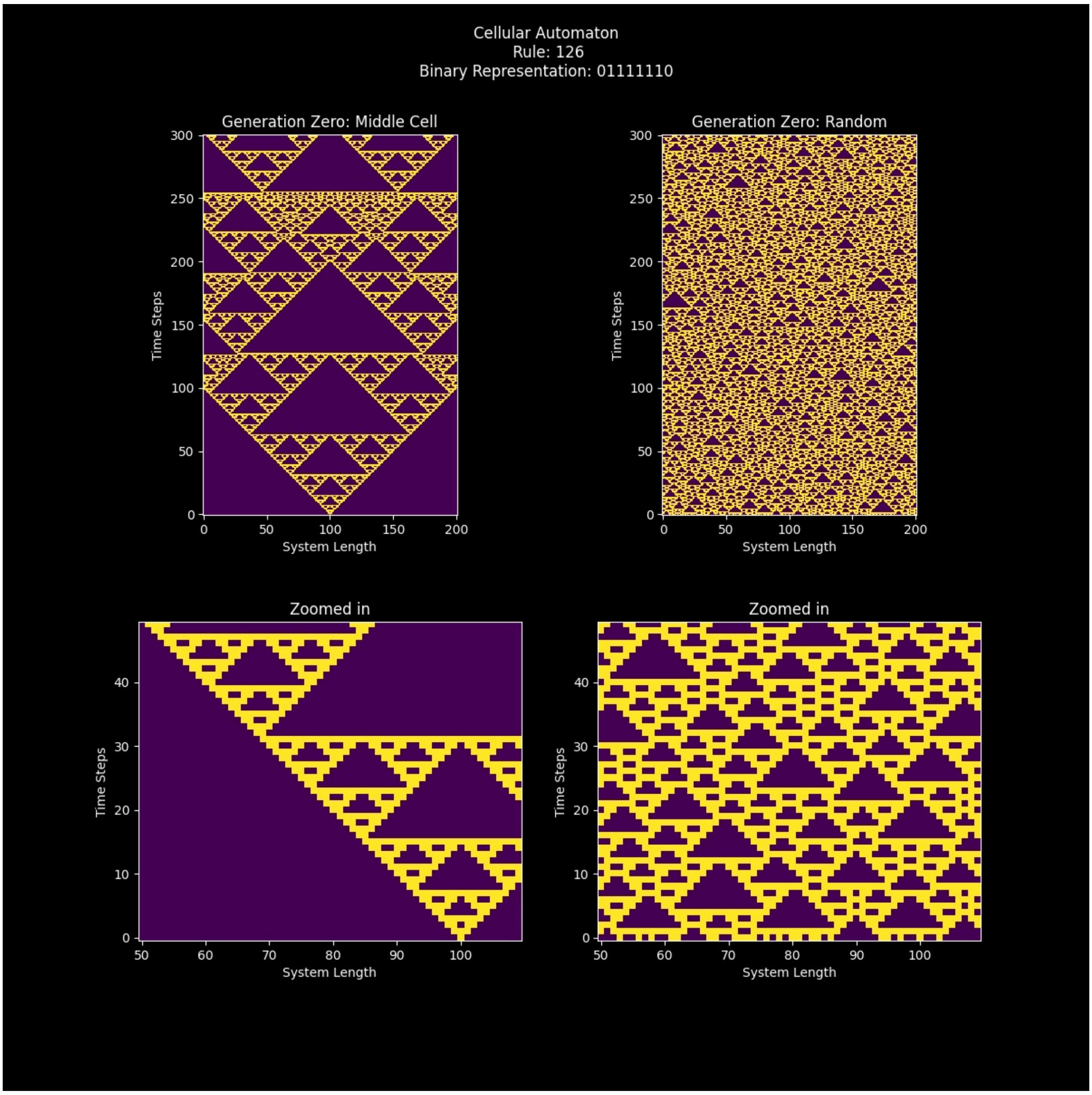
Cellular Automaton
Rule: 106
Binary Representation: 01101010
Generation Zero: Middle Cell
Generation Zero: Random
Zoomed in
Zoomed in
Time Steps
System Length

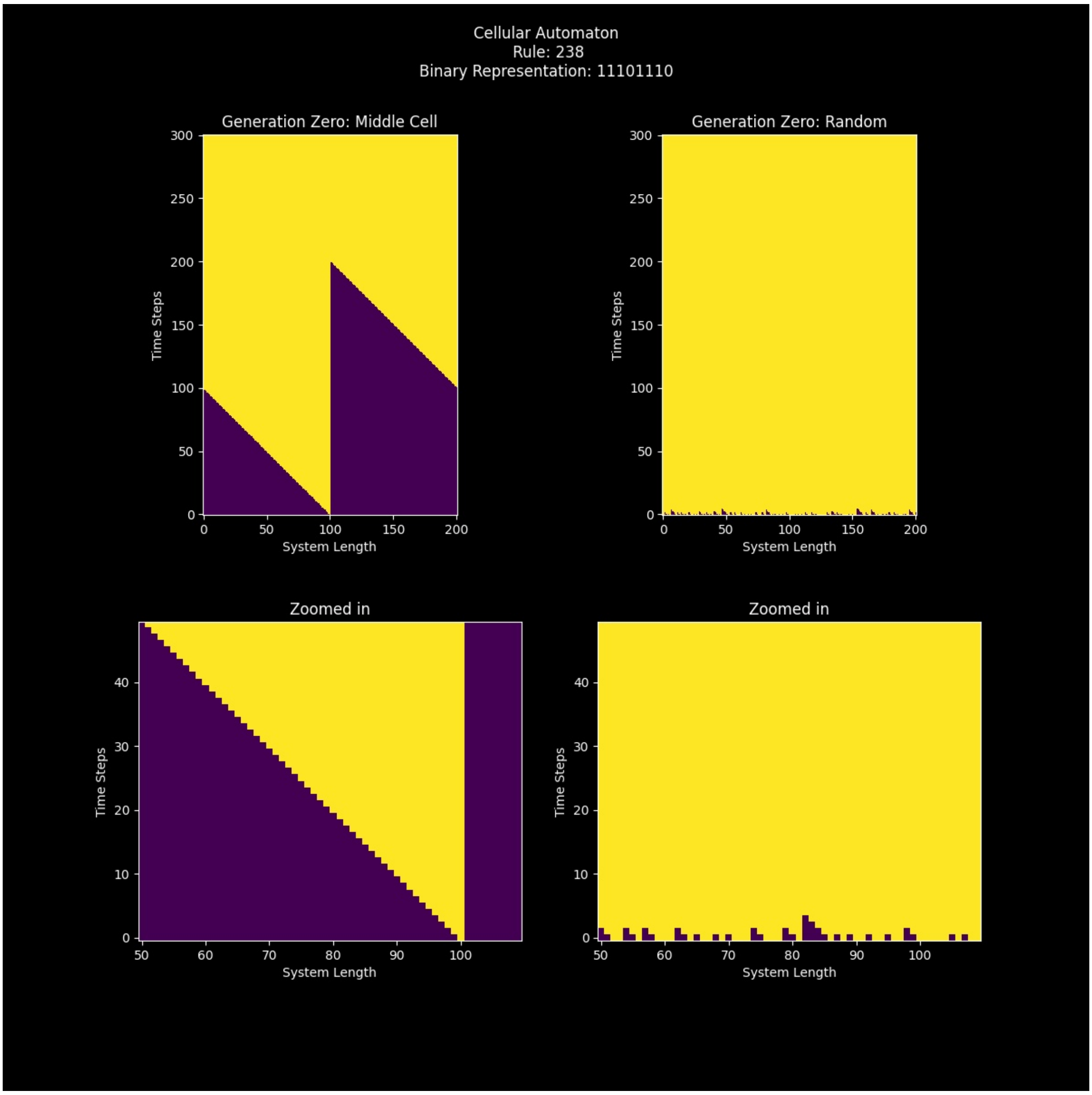


Cellular Automaton
Rule: 108
Binary Representation: 01101100
Generation Zero: Middle Cell
Generation Zero: Random
Zoomed in
Zoomed in
Time Steps
System Length



Cellular Automaton
Rule: 111
Binary Representation: 01101111
Generation Zero: Middle Cell
Generation Zero: Random
Zoomed in
Zoomed in
Time Steps
System Length


Cellular Automaton
Rule: 126
Binary Representation: 01111110
Generation Zero: Middle Cell
Generation Zero: Random
Time Steps
System Length
Zoomed in
Zoomed in




Sources:
1. Wikipedia: Cellular automaton
2. matplotlib blog posts: Eitan Lees (2020); Elementary Cellular Automata