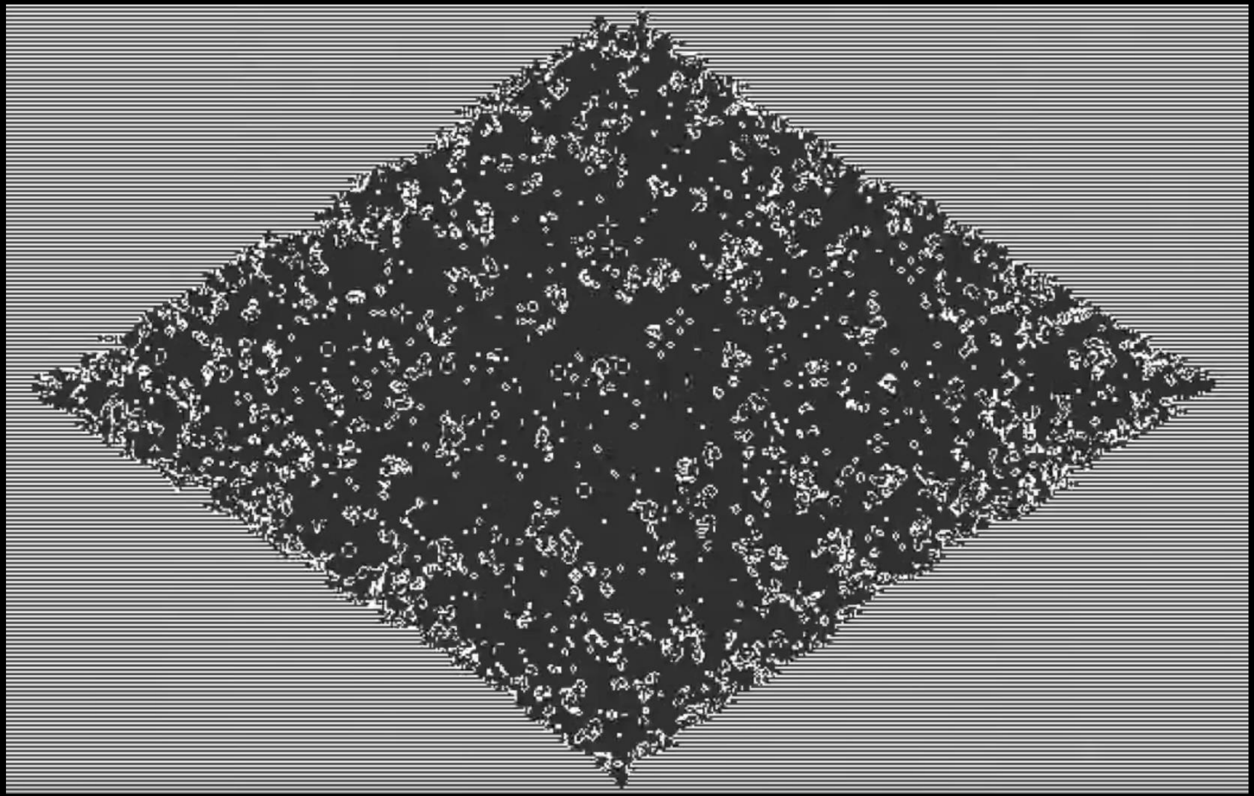
Computer modeling of physical phenomena



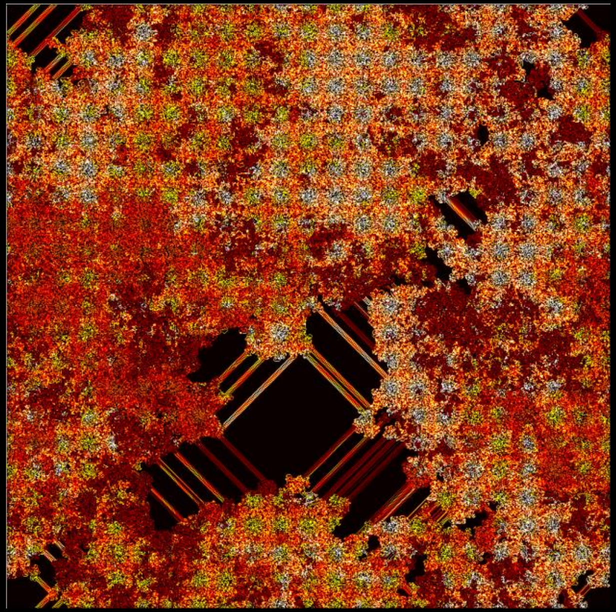
27-28.02.2024

Lab 1: Game of Life

Task 1

Langton's Ant

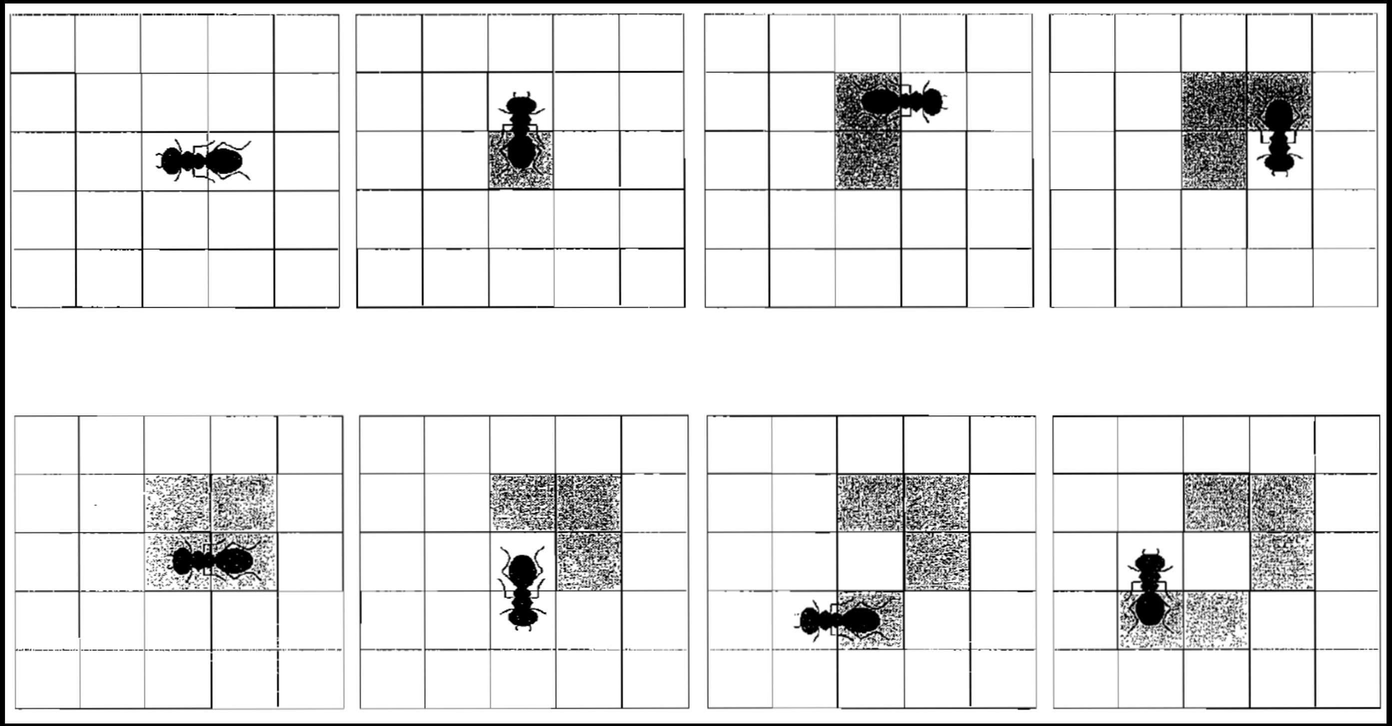
Create a code that simulates Langton's Ant. Start small, with a grid of size 100x100 filled with zeros and make 10000 ant steps (use periodic boundary conditions). If that works, try a large grid (e.g. 1000x1000) and simulate four ants for 1000000 steps. Mark ants with different colours. Use *numba* for speed and plot the final grid.



Task 1

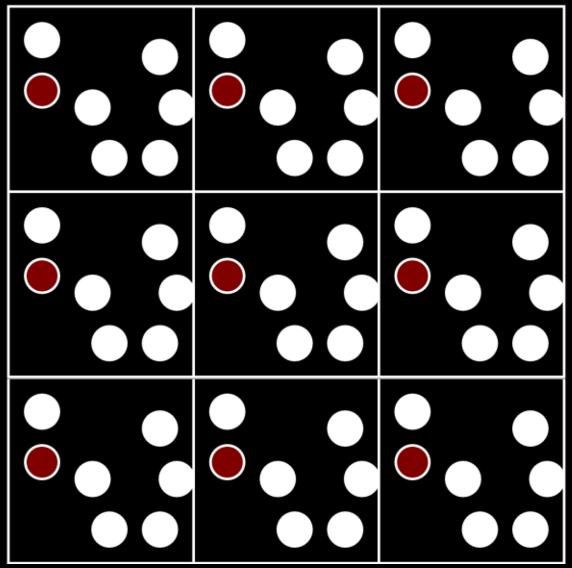
Langton's Ant

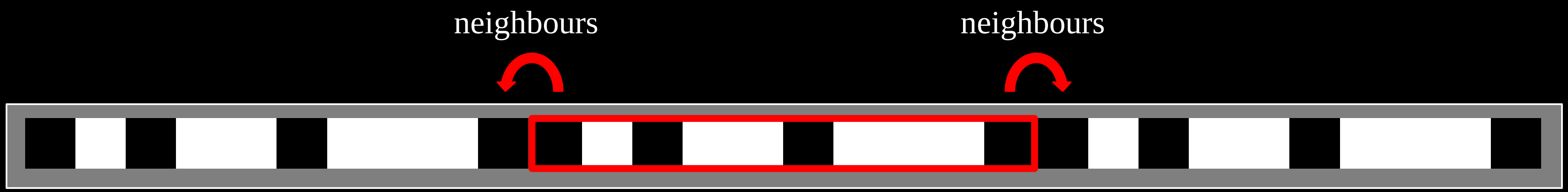
1) The ant changes any white cell it lands on to black, then turns 90º to the right and moves one cell forward.
2) The ant changes any black cell it lands on to white, then turns 90º to the left and moves one cell forward.



Periodic Boundary Conditions (PBC)

Surround the system with its replicas.

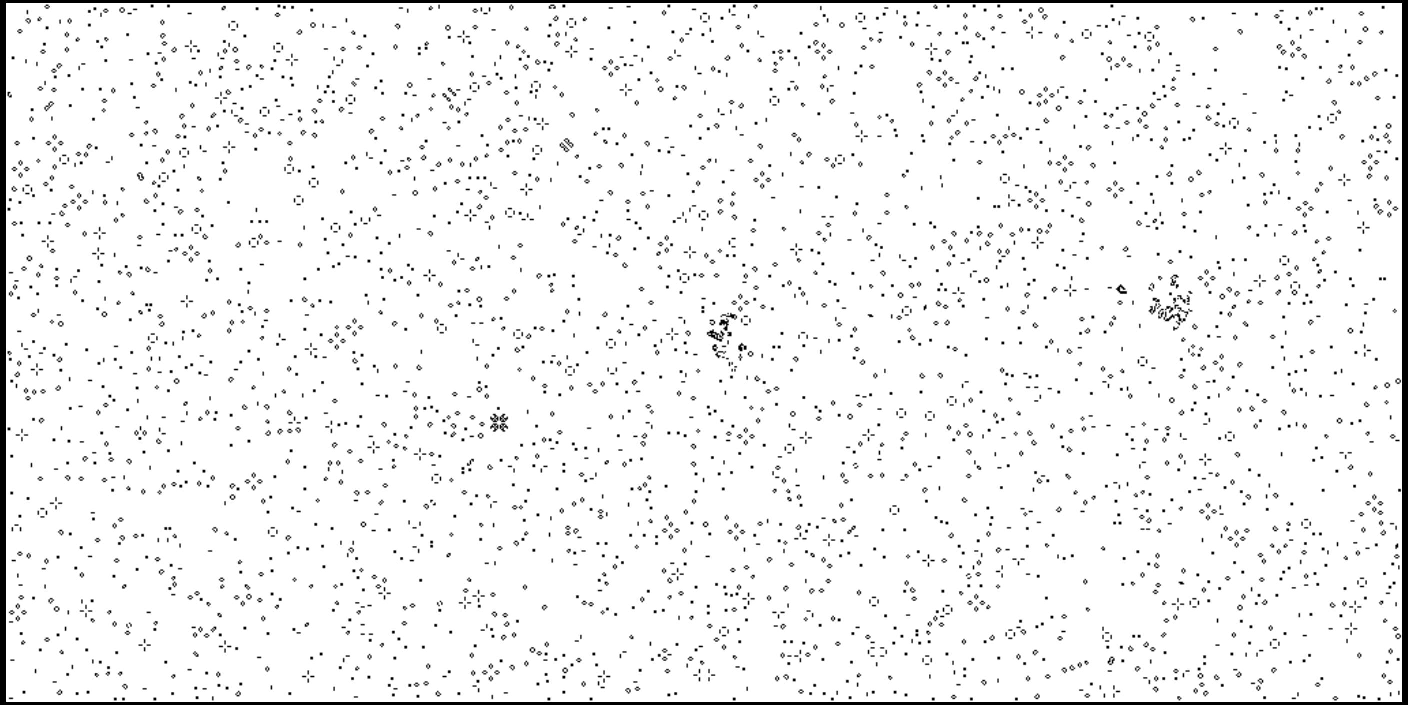




Task 2 v.1

Game of Life

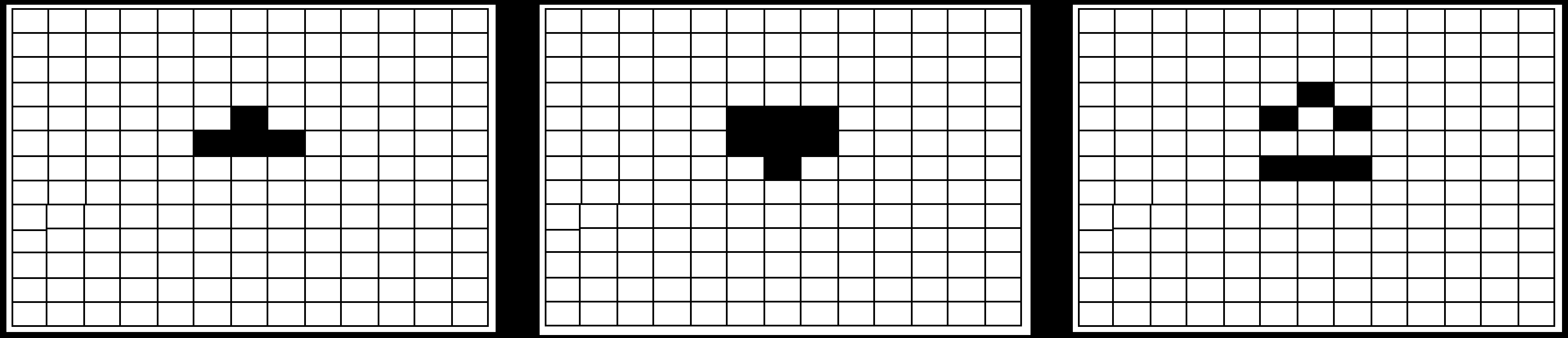
Write a game of life simulator, which would take any initial configuration and iterate it according to GoL rules. Try it out on a random initial pattern, e.g. *np.random.randint(0,2,(256,512))* and iterate it to the steady state; make a corresponding movie.



Task 2 v.1

Game of Life

1) A dead cell becomes alive at the next generation if exactly three of its (Moore) neighbours are alive.
2) A live cell at the next generation remains alive if either two or three of its neighbours are alive but otherwise it dies.



initial state | step 1 | step 2

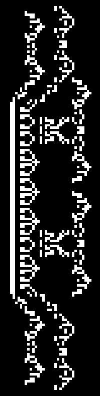
Task 2 v.1

Game of Life

Boundary conditions: put the layer of white (0) cells around your system and keep them at 0.

Vectorize - do not use loops at all! Otherwise you will wait forever...

Plot e.g. using *plt.imshow*.



Task 2 v.2

Dog Bone

Simulate a `dog bone` 2D cellular automaton. Work on a grid of size 101x101 and start with a single black cell in the middle. Use np.roll for checking the neighbourhood. To reproduce the configuration below, perform 47 steps. After that you can explore larger systems and longer time steps and observe the complex patterns.

Task 2 v.2

Dog Bone

1) A cell becomes alive at the next generation if it was a neighbour (in the von Neumann sense) of exactly one live cell of the current generation.
2) All cells that are two generations old are required to die.

That means: $(n+1)$th generation is derived from the nth generation. The nth generation survives and the $(n-1)$th generation is erased.

Extra task v.1

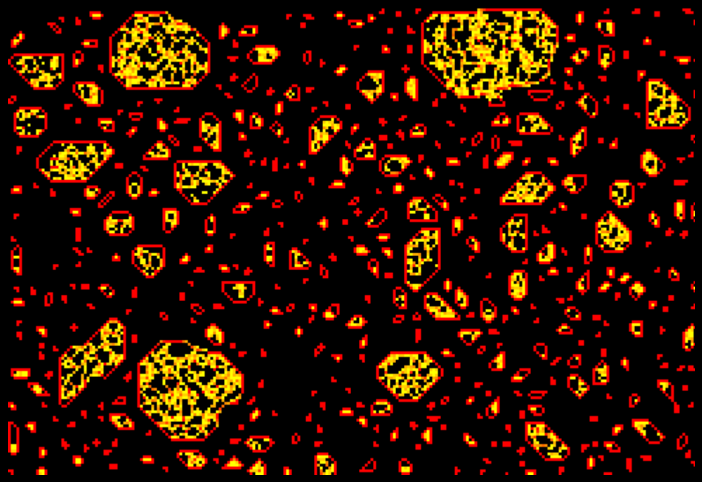
General Totalistic 2D CA

Create a code that can simulate any totalistic (dependent only on the numer of neighbours alive) 2D cellular automaton. Totalistic automata are coded in a form (S/B), e.g. (1234/1). Pursue the possible (and sensible) evolutions, for comparison (and inspiration for the interesting rules) you can use *http://www.mirekw.com/ca/rullex_life.html*. Use closed BC.

(S/B)

no. of live neighbors needed for a cell to survive

no. of live neighbors needed for a cell to be born

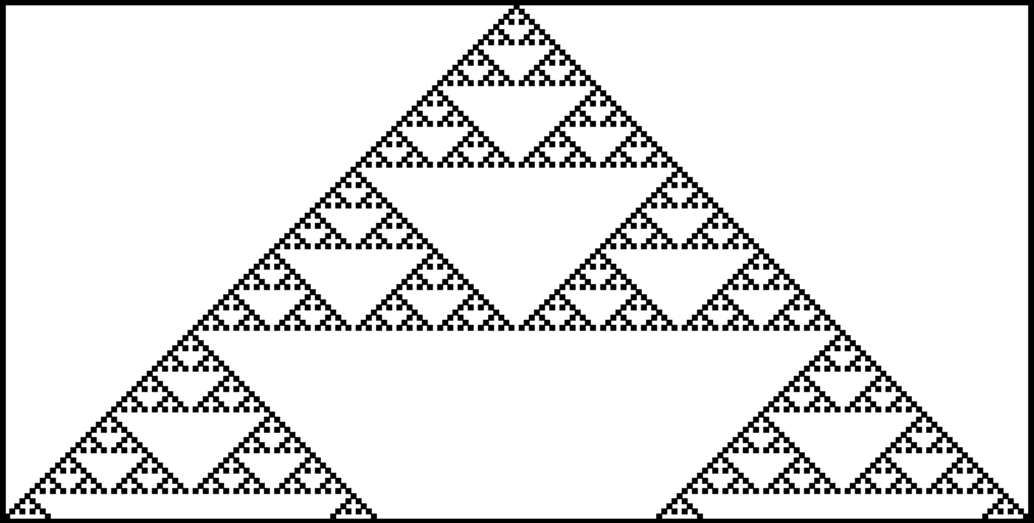


Extra task v.2

General 1D Cellular Automaton

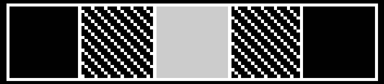
Only if you haven’t done it before – but then very highly recommended!

Create a code that can simulate any 1D cellular automaton (it should take only the rule number as parameter). Check out the evolution for multiple rules.

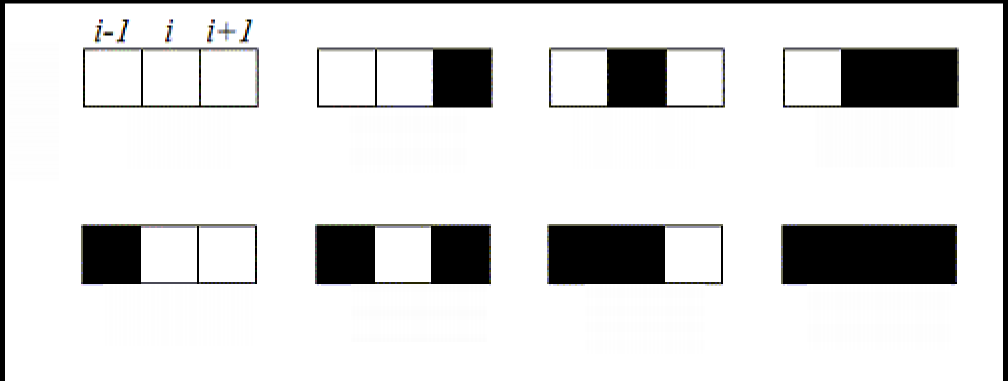


1D cellular automata

Let's consider 1D cellular automata with neighbourhood consisting of two closest cells.

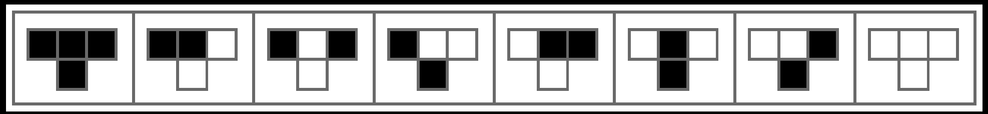


This neighbourhood may take one of 8 states:



Transition rule determines the state of the middle cell in each of these cases.

Example

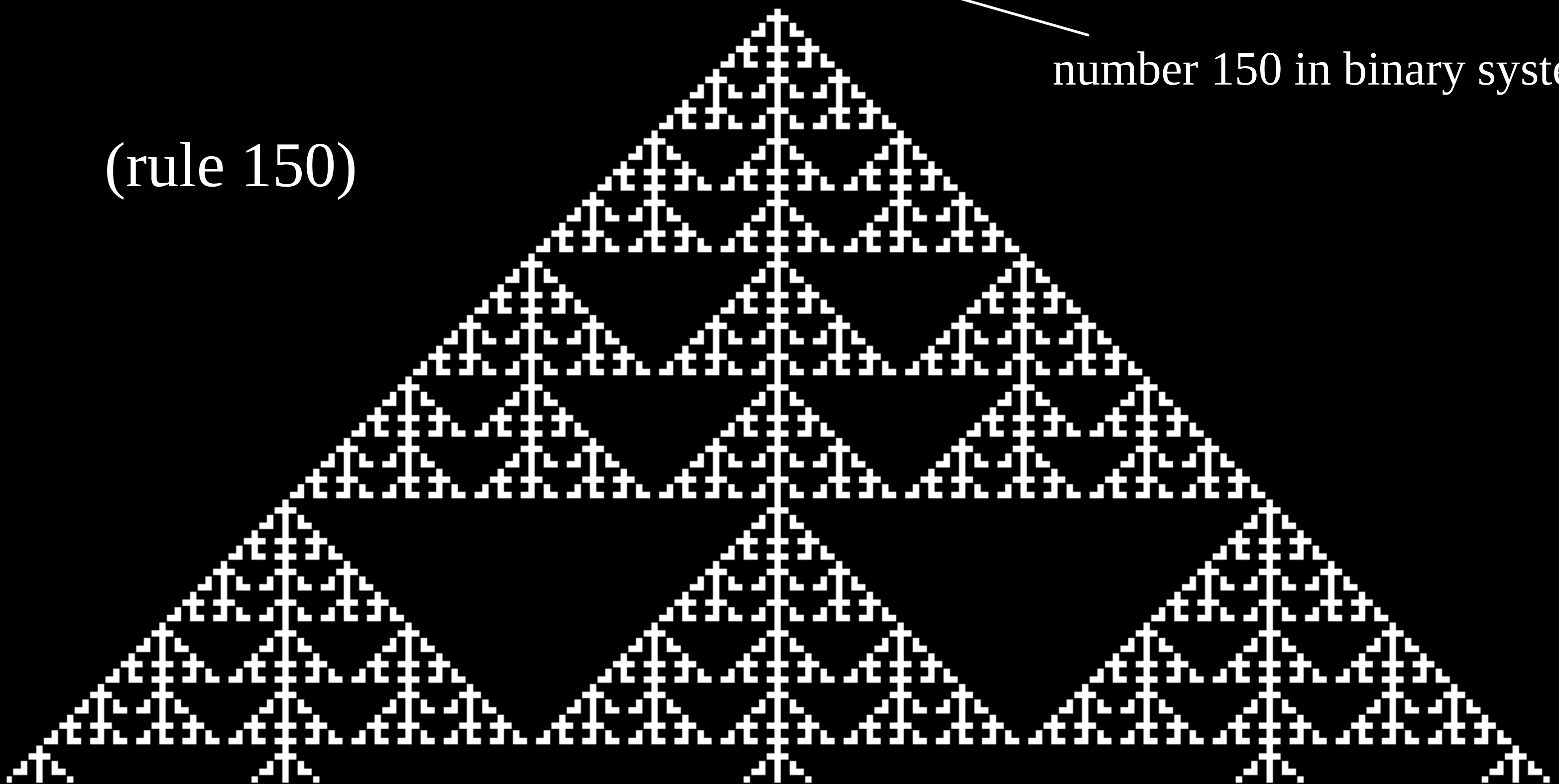


in total 256 different rules!

111	110	101	100	011	010	001	000
1	0	0	1	0	1	1	0

number 150 in binary system

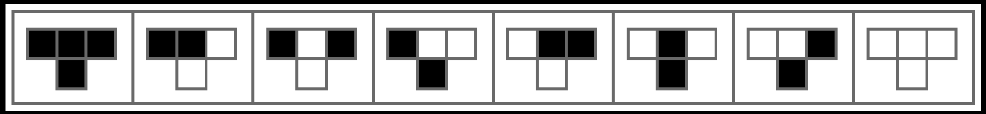
(rule 150)



Extra task v.2 – code template

1. Load the rule number and translate it to the transition rule (translating should happen automatically – the program takes only the number and calculates the rule itself).

rule 150



2. Initialize the first row with one middle black cells surrounded by 100 white cells.

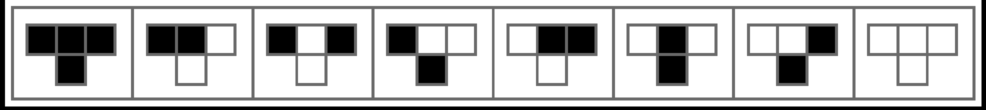
3. Don’t use loops in space! Use *np.roll* for checking the neighbourhood (with periodic boundary conditions).

4. Evolve in time for T = 1000 steps, collecting the rows for each iteration in one huge array, which we plot.

Extra task v.2 – tips

To translate the rule, you can use e.g. *np.binary_repr*.

Remember the sequence of the neighbourhoods in the rule!

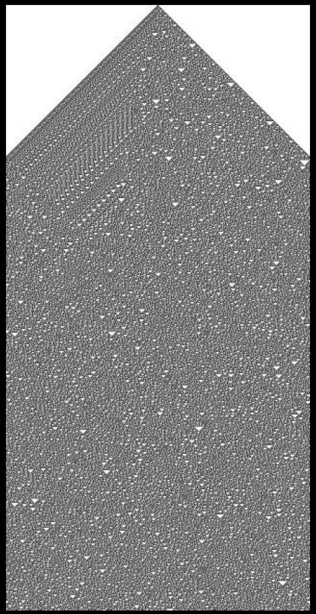
binary value	111	110	101	100	011	010	001	000
decimal value	7	6	5	4	3	2	1	0
color code								
rule 150	1	0	0	1	0	1	1	0

For applying the rule, consider the following example:

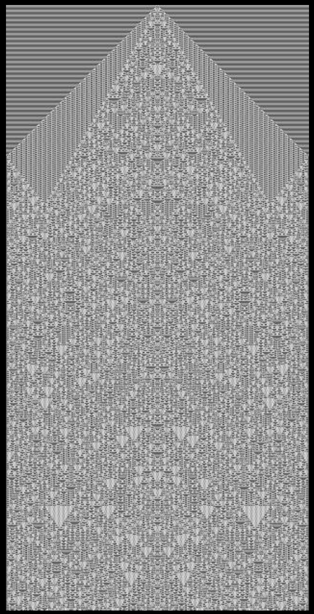
```
rule = np.array([0, 1, 1, 0, 1, 0, 0, 1])
neighbourhood_value = np.array([0, 2, 4, 5, 7])
print(rule[neighbourhood_value])
[0 1 1 0 1]
```

Drawing

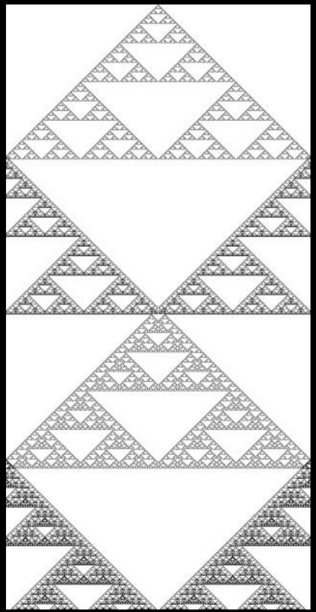
```python
from PIL import Image, ImageDraw
img = Image.new("RGB",(width, height),(255,255,255))
draw = ImageDraw.Draw(img)
for y in range(height):
   for x in range(width):
       if data[y][x]:
           draw.point((x,y),(0,0,0))
img.save(f"ca{rulenr}.PNG")
```



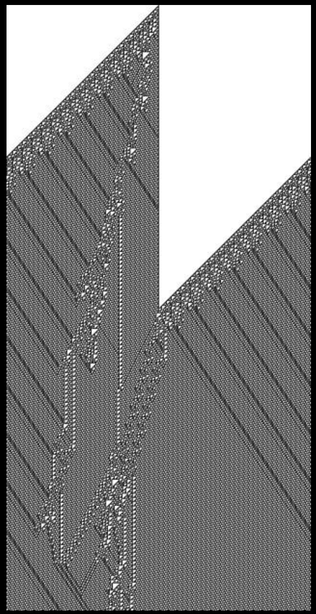
30



73



90



110

Points

1. Langton’s Ant on a small grid – 0.25 p.
   Multiple ants on a large grid – 0.25 p.

2. Animation from Game of Life simulator or Dog Bone – 0.5 p.

3.* General totalistic 2D CA or general 1D CA – 0.2 p.