

cellular automata & Agent-based modeling

{'Summer School' Bayonne': 2024}



mario.gellrich@zhaw.ch

Dr. Mario Gellrich

Cellular automata

- What is a cellular automaton?
- Motivation to develop cellular automata
- Application areas
- Components of a cellular automaton
- Cellular automaton example: Conway's Game of Life

What is a cellular automaton?



Image Credit: <https://towardsdatascience.com/cellular-automaton-and-deep-learning-2bf7c57139b3>

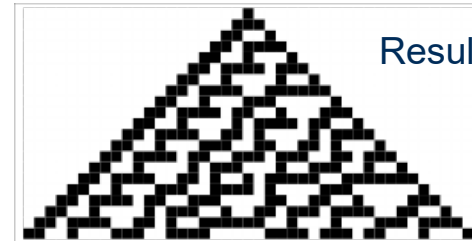
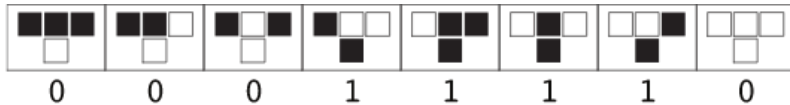
A **cellular automaton** (pl. cellular automata) is a discrete model of computation studied in automata theory.

What is a cellular automaton?

A cellular automaton is defined by three key elements:

- Collection of cells each having a state
- The cells are distributed on a grid with a predefined shape
- The cells update their state according to a rule based on the neighboring cells

Rule



Resulting pattern

Image Credit: <https://mathworld.wolfram.com/ElementaryCellularAutomaton.html>

Motivation to develop cellular automata

Cellular automata can be used to investigate patterns and its underlying rules

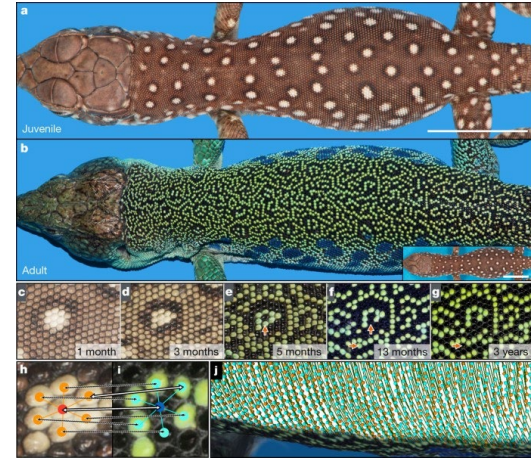
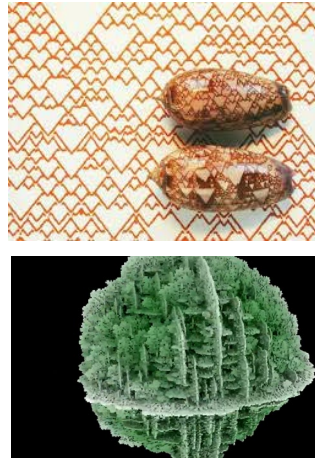
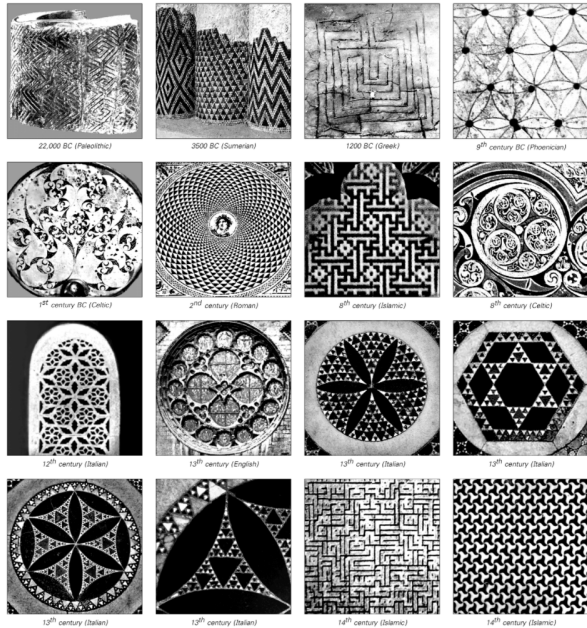
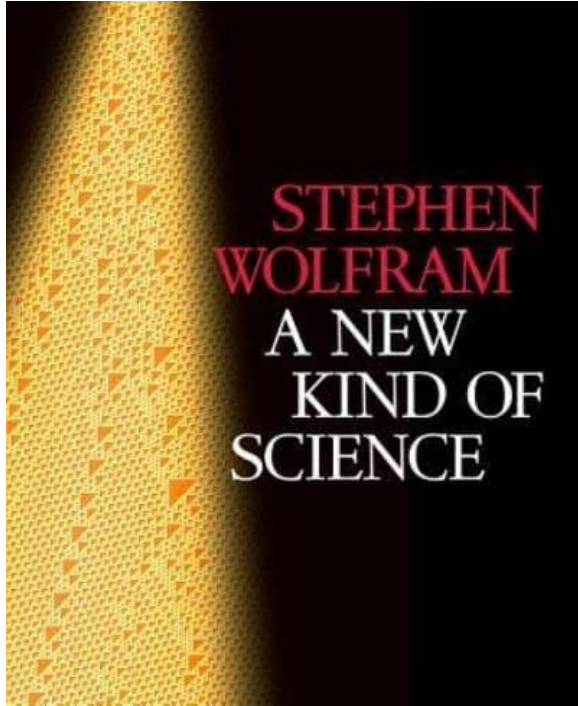


Image Credit:
<https://www.wolframscience.com/nks/>
<https://www.pinterest.com/wonderscience/cellular-automata-wstv/>
<https://www.nature.com/articles/nature22031>
<https://www.youtube.com/watch?v=W-n510Pca0>

Motivation to develop cellular automata



From 1992 to 2002, Stephen Wolfram worked on his controversial book «**A New Kind of Science**» which presents an empirical study of simple computational systems. Additionally, it argues that for fundamental reasons these types of systems, rather than traditional mathematics, are needed to model and understand complexity in nature.

Application areas of cellular automata

Application areas of cellular automata

- Biology
- Chemistry
- Physics
- Computer science, coding, and communication
- Generative art and music (see example right)

Open Access Article

Generative Music with Partitioned Quantum Cellular Automata

by  Eduardo Reck Miranda ^{1,2,*}  and  Hari Shaji ¹ 

¹ Interdisciplinary Centre for Computer Music Research (ICCMR), University of Plymouth, Plymouth PL4 8AA, UK

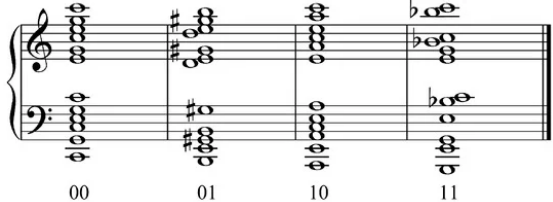
² Quantinuum, Partnership House, Carlisle Place, London SW1P 1BX, UK









* Author to whom correspondence should be addressed.

Appl. Sci. **2023**, *13*(4), 2401; <https://doi.org/10.3390/app13042401>

Received: 3 January 2023 / Revised: 31 January 2023 / Accepted: 3 February 2023 /

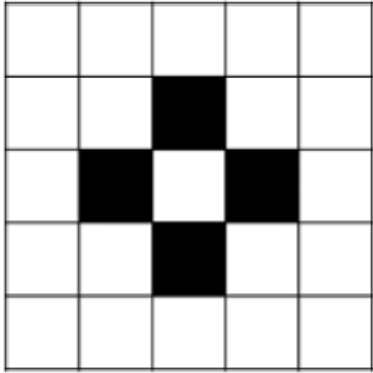
Published: 13 February 2023



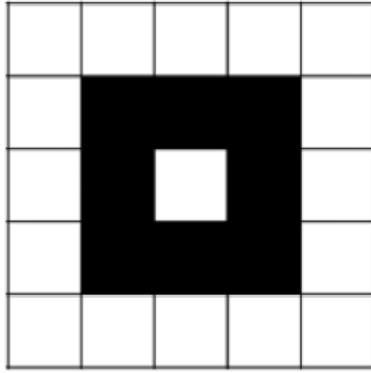
	= 000		= 100
	= 001		= 101
	= 010		= 110
	= 011		= 111

Components of a cellular automaton: Neighborhood

Neighborhood in a cellular automaton



a)



b)

a) von Neumann neighborhood

b) Moore neighborhood

Image Credit: <https://www.researchgate.net>

Components of a cellular automaton: Update Rules

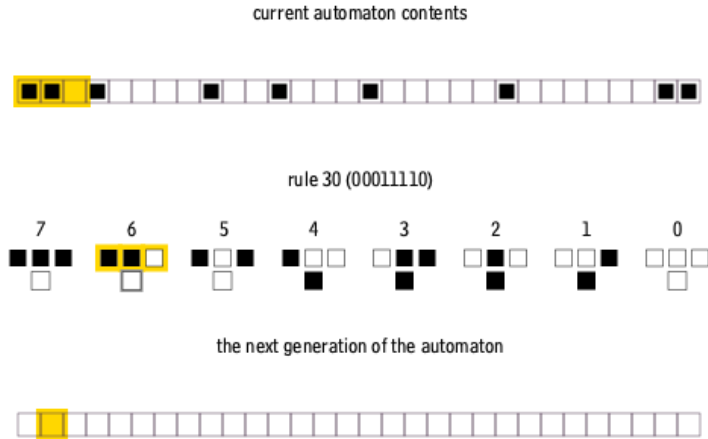


Image Credit: https://en.wikipedia.org/wiki/Cellular_automaton

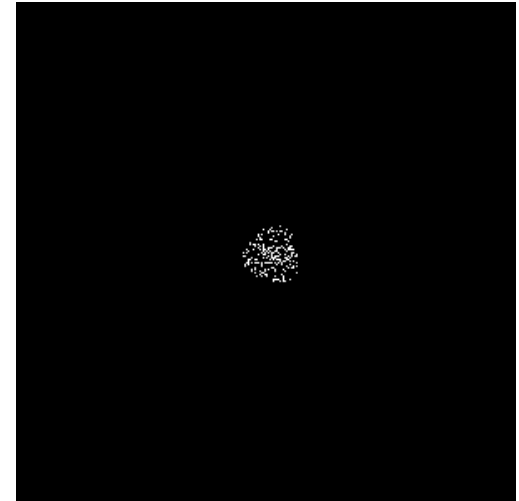
One-dimensional cellular automaton with two possible states per cell, and a cell's neighbors defined as the adjacent cells on either side of it. A cell and its two neighbors form a neighborhood of 3 cells, so there are $2^3 = 8$ possible patterns for a neighborhood. A rule consists of deciding, for each pattern, whether the cell will be a 1 or a 0 in the next generation. There are then $2^8 = 256$ possible rules.

Components of a cellular automaton: Specific rules

Specific rules (for details, follow the links below)

- [Brian's Brain](#)
- [Codd's cellular automaton](#)
- [CoDi](#)
- [Conway's game of life](#)
- [Day and Night](#)
- [Langton's ant](#)
- [Langton's loops](#)
- [Nobili cellular automata](#)
- [Rule 90](#)
- [Rule 184](#)
- [Seeds](#)
- [Turmite](#)
- [Von Neumann cellular automaton](#)
- [Wireworld](#)

Rule: Brian's Brain



Cellular automaton example: Conway's Game of Life

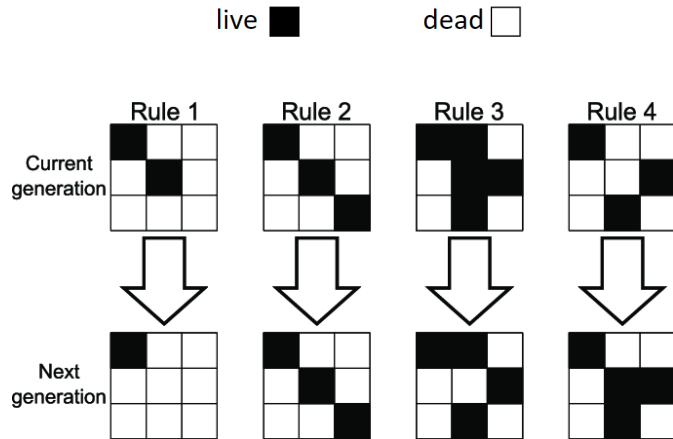


Conway's Game of Life is a cellular automaton devised by the British mathematician John Horton Conway in 1970. It is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input.

Image Credit: https://en.wikipedia.org/wiki/Cellular_automaton

Cellular automaton example: Conway's Game of Life












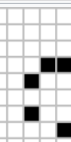


Rules of Conway's Game of Life



1. Any live cell with fewer than two live neighbours dies, as if by underpopulation.
2. Any live cell with two or three live neighbours lives on to the next generation.
3. Any live cell with more than three live neighbours dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

Image Credit: https://www.researchgate.net/figure/Rules-of-Conways-Game-of-Life_fig5_339605473

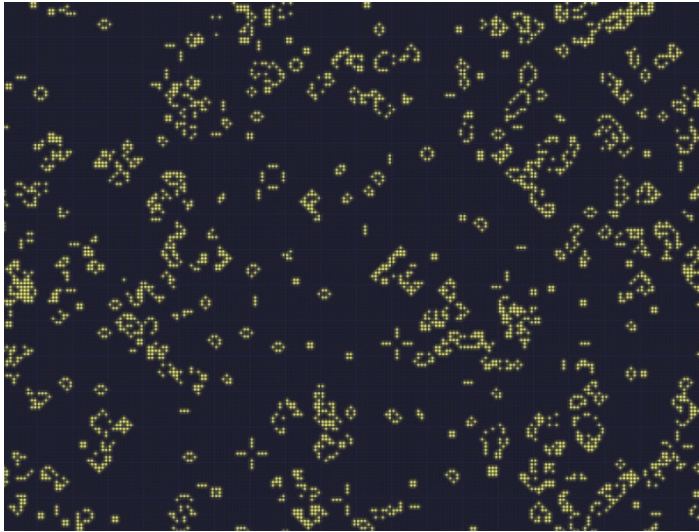
Cellular automaton example: Conway's Game of Life

Still lifes		Oscillators		Spaceships	
Block		Blinker (period 2)		Glider	
Beehive		Toad (period 2)		Light-weight spaceship (LWSS)	
Loaf		Beacon (period 2)		Middle-weight spaceship (MWSS)	
Boat		Pulsar (period 3)		Heavy-weight spaceship (HWSS)	
Tub		Pentadecathlon (period 15)			

Many different **types of patterns** occur in the Game of Life, which are classified according to their behavior. Common pattern types include **still lifes**, which do not change from one generation to the next; **oscillators**, which return to their initial state after a finite number of generations; and **spaceships**, which translate themselves across the grid.

Image Credit: https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

Cellular automaton example: Conway's Game of Life



https://beltoforion.de/en/game_of_life

RANDOMIZED

R-PENTOMINO

GLIDER

SPACESHIP

TRANS QUEEN BEE SHUTTLE

QUEEN BEE LOOP

PARTIAL QUEEN BEE LOOP

TAGALONG FOR TWO LWSS

GLIDER DUPLICATOR

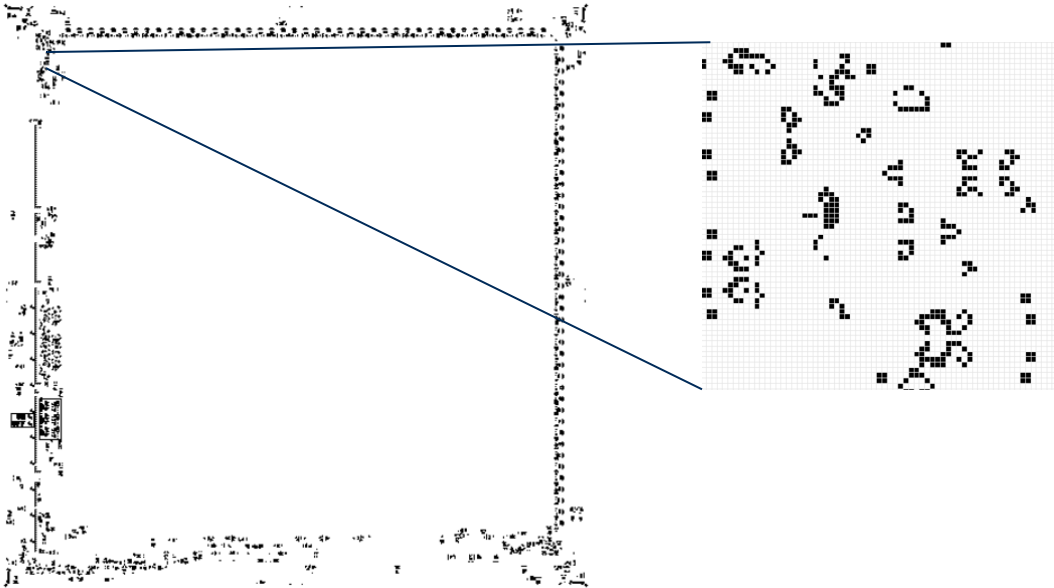
TWOGUN

NEWGUN

BLOCK-LAYING SWITCH ENGINE

40514M

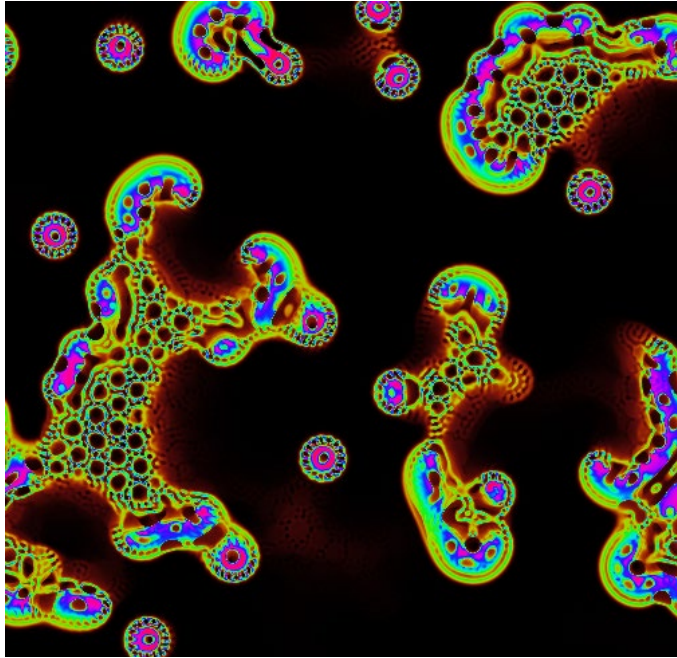
Cellular automaton example: Conway's Game of Life



In an **Octa-Metapixel** each cell is represented by a small metapixel, which is itself a small cellular automaton.

Image Credit: https://beltoforion.de/en/game_of_life

Cellular automaton example: Conway's Game of Life



In **Smooth-Life**, Conway's Game of Life is generalized to a continuous domain.

Image Credit: <https://sourceforge.net/projects/smoothlife>

Cellular automaton example: 3D cellular automata

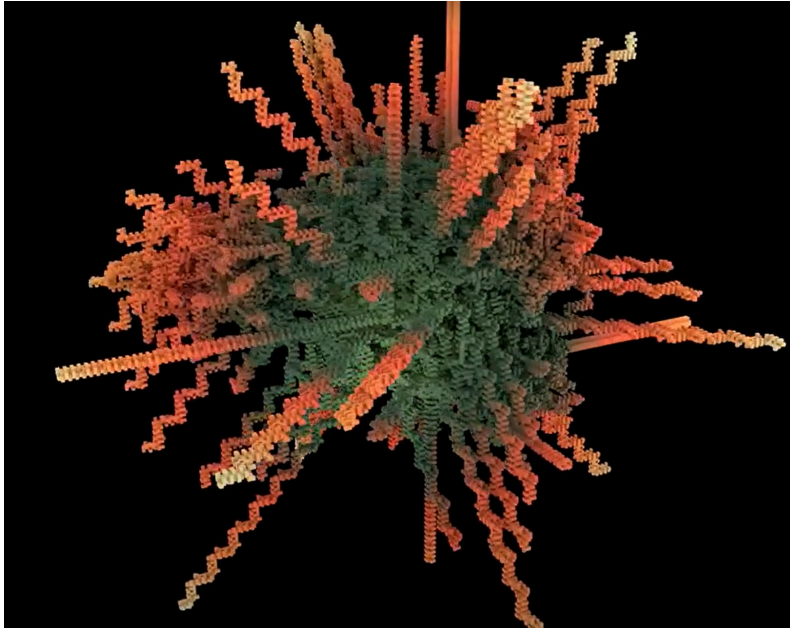


Image credit: https://youtu.be/_W-n510Pca0

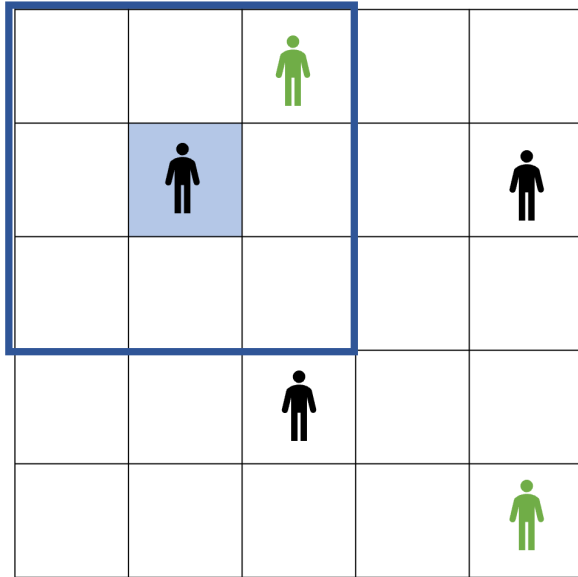
3D cellular automata are extensions of 1D and 2D cellular automata into the third dimension. Rather than just checking neighbor cells in the X and Y directions, the Z direction is also included.

To see examples, follow this [Link](#).

Agent based modeling

- What is an agent-based model?
- Components of an agent-based model
- Agent-based modeling examples
 - Simulation of swarm behavior
 - Forest fire simulation
 - Schelling's model of segregation
 - Animal disease simulation
- Creating an agent-based model from scratch

What is an agent-based model?



An **agent-based model (ABM)** is a computational model for simulating the actions and interactions of autonomous agents in order to understand the behavior of a system and what governs its outcomes.

Image Credit: <https://datasciencechalktalk.wordpress.com>

Components of an agent-based model

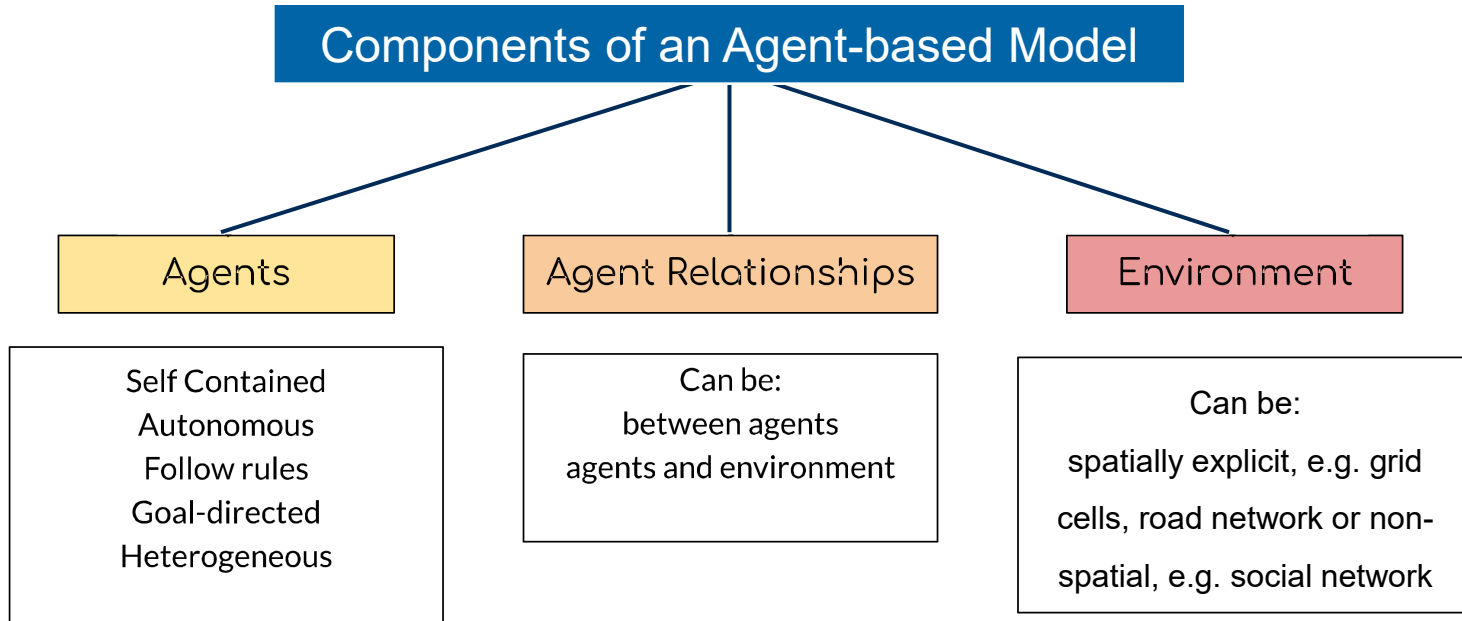
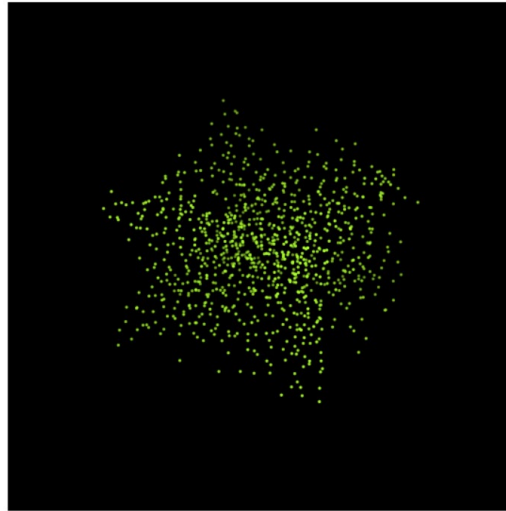


Image adapted from: <https://www.mdpi.com/2076-2607/9/2/417/htm>

Agent-based modeling examples: Simulation of swarm behavior

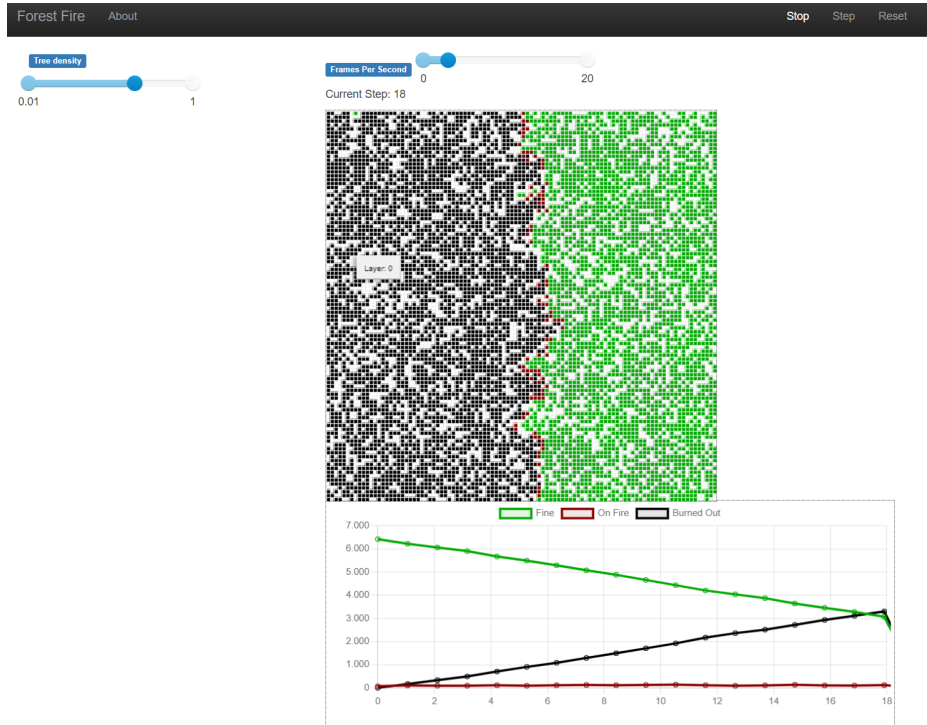
```
parameters3D = {  
    'size': 50,  
    'seed': 123,  
    'steps': 200,  
    'ndim': 3,  
    'population': 1000,  
    'inner_radius': 3,  
    'outer_radius': 10,  
    'border_distance': 10,  
    'cohesion_strength': 0.005,  
    'seperation_strength': 0.1,  
    'alignment_strength': 0.3,  
    'border_strength': 0.5,  
    'wall_avoidance_distance': 5,  
    'wall_avoidance_strength': 0.3  
}
```

Boids Flocking Model 3D (t=10)



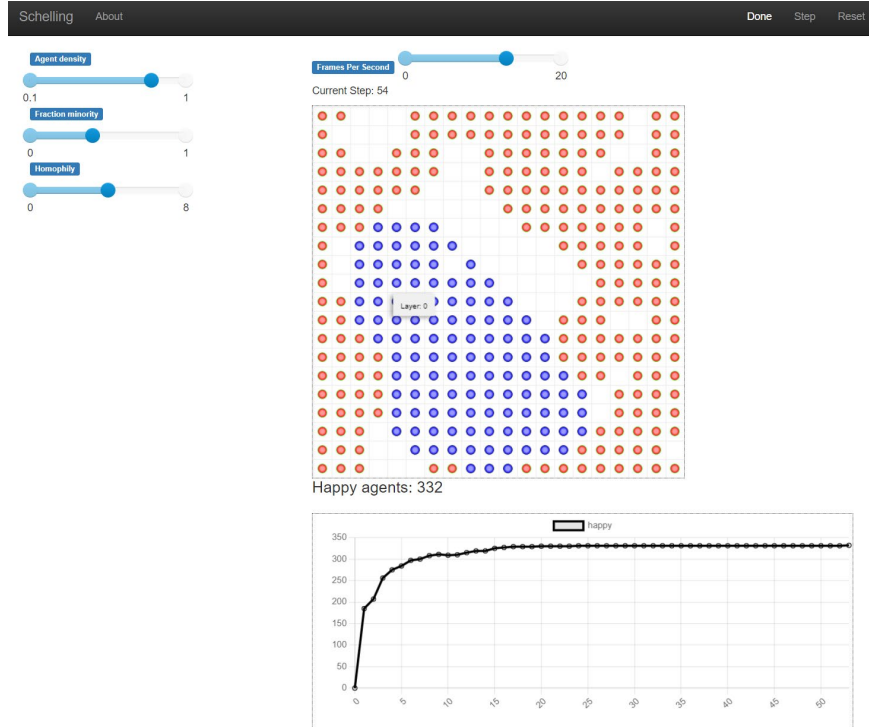
In this example, agents are **animals organized in a swarm** like certain bird-, fish- or insect species. Model parameters are related e.g. to the size of population, radius of the swarm and separation of individual agents. Such simulations can be used to investigate **collective behavior in complex systems**.

Agent-based modeling examples: Forest fire simulation



In this example, **agents are trees** in a forest which will be destroyed by a forest fire or not. The only parameter in this simulation model is the tree density. This agent-based simulation model can be used to identify the optimal tree density in areas with high wild fire risk to prevent future damages.

Agent-based modeling examples: Schelling's model of segregation



Schelling's model of segregation is an agent-based model developed by economist Thomas Schelling. The model demonstrate that having people with "mild" in-group preference towards their own group could lead to a highly segregated society via de facto segregation. The parameters in the example left are agent density, fraction minority and homophily.

Creating an agent-based model from scratch

Example Model Description

1. There are some number of agents.
2. All agents begin with 1 unit of money.
3. At every step of the model, an agent gives 1 unit of money (if available) to some other agent.

Creating an agent-based model from scratch

Load required libraries

```
1 from mesa import Agent, Model
2 from mesa.time import RandomActivation
3 from mesa.space import MultiGrid
4 from mesa.datacollection import DataCollector
```

Basis: mesa library

Create an agent class

```
1 # MoneyAgent class
2 class MoneyAgent(Agent):
3     """An agent with fixed initial wealth."""
4
5     def __init__(self, unique_id, model):
6         super().__init__(unique_id, model)
7         self.wealth = 1
8
9     # Method to move the agent on the grid
10    def move(self):
11        possible_steps = self.model.grid.get_neighborhood(
12            self.pos, moore=True, include_center=False
13        )
14        new_position = self.random.choice(possible_steps)
15        self.model.grid.move_agent(self, new_position)
16
17    # Method to give money to other agents
18    def give_money(self):
19        # Count the number of agents in neighboring cells
20        cellmates = self.model.grid.get_cell_list_contents([self.pos])
21        if len(cellmates) > 1:
22            other = self.random.choice(cellmates)
23            other.wealth += 1
24            self.wealth -= 1
25
26    # Step method (action the agent takes when it is activated by the schedule)
27    def step(self):
28        self.move()
29        if self.wealth > 0:
30            self.give_money()
```

Agent class with methods

Creating an agent-based model from scratch

Create a model class

```
1  # MoneyModel class
2  class MoneyModel(Model):
3      """A model with some number of agents."""
4
5      def __init__(self, N, width, height):
6          self.num_agents = N
7          self.grid = MultiGrid(width, height, True)
8          # Schedule (controls the order in which agents are activated)
9          self.schedule = RandomActivation(self)
10
11         # Create agents using MoneyAgent class
12         for i in range(self.num_agents):
13             a = MoneyAgent(i, self)
14             self.schedule.add(a)
15             # Add the agent to a random grid cell
16             x = self.random.randrange(self.grid.width)
17             y = self.random.randrange(self.grid.height)
18             self.grid.place_agent(a, (x, y))
19
20         # Data collector (for Gini coefficient see function compute_gini())
21         self.datacollector = DataCollector(
22             model_reporters={"Gini": compute_gini}, agent_reporters={"Wealth": "wealth"}
23         )
24
25         def step(self):
26             self.datacollector.collect(self)
27             self.schedule.step()
```

Model class with methods

Creating an agent-based model from scratch

Function for the calculation of the Gini coefficient (measure of wealth inequality)

```
1 # Function to calculate the Gini coefficient
2 def compute_gini(model):
3     agent_wealths = [agent.wealth for agent in model.schedule.agents]
4     x = sorted(agent_wealths)
5     N = model.num_agents
6     B = sum(xi * (N - i) for i, xi in enumerate(x)) / (N * sum(x))
7     return 1 + (1 / N) - 2 * B
```

Create a model and define the model parameters

```
1 # Create a model with 300 agents on a 20x20 grid, and run it for 100 steps
2 model = MoneyModel(300, 20, 20)
3 for i in range(100):
4     model.step()
```

*The **Gini coefficient** is a measure of the distribution of income across a population. Values are usually between 0 and 1. A higher Gini coefficient indicates greater inequality, with high-income individuals receiving much larger percentages of the total income of the population. Likewise, a lower Gini coefficient means greater equality.

Calculate wealth inequality
using the Gini coefficient*

Create a model with:

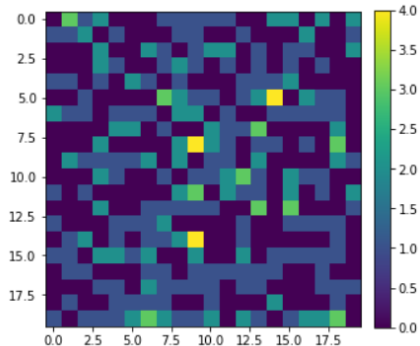
- 300 agents
- 20x20 grid (400 grid cells)
- 100 steps

Creating an agent-based model from scratch

Visualize the number of agents on the grid

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 agent_counts = np.zeros((model.grid.width, model.grid.height))
5
6 for cell in model.grid.coord_iter():
7     cell_content, x, y = cell
8     agent_count = len(cell_content)
9     agent_counts[x][y] = agent_count
10
11 plt.figure(figsize=(5,5))
12 plt.imshow(agent_counts, interpolation="nearest")
13 plt.colorbar(fraction=0.046, pad=0.04)
```

<matplotlib.colorbar.Colorbar at 0x1724b61f9d0>



Number of agents on the 20x20 grid.

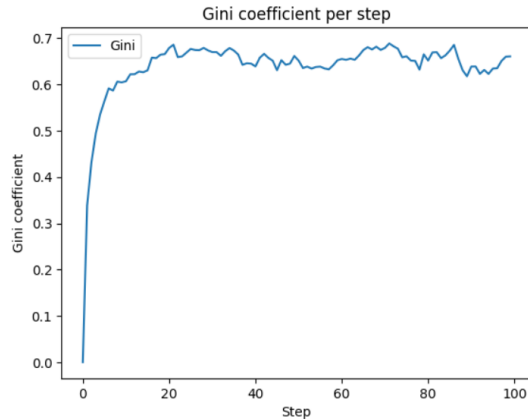
Creating an agent-based model from scratch

Plot Gini coefficient per step

+ Code + Markdown

```
gini = model.datacollector.get_model_vars_dataframe()
gini.plot(xlabel='Step',
          ylabel='Gini coefficient',
          title='Gini coefficient per step')
```

<Axes: title={'center': 'Gini coefficient per step'}, xlabel='Step', ylabel='Gini coefficient'>



The Gini coefficient increases during model run showing that the wealth inequality increases over time.

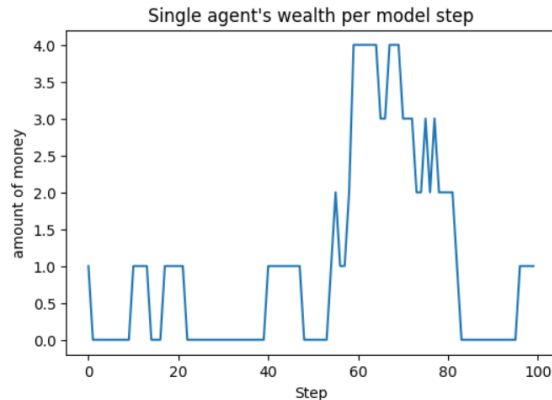
Creating an agent-based model from scratch

Line chart of a single agent's wealth per model step

```
# Wealth of a single agent
one_agent_wealth = agent_wealth.xs(5, level="AgentID")
one_agent_wealth.Wealth.plot(figsize=(6,4),
                              xlabel='Step',
                              ylabel='amount of money',
                              title="Single agent's wealth per model step")
```

✓ 0.1s

<Axes: title={'center': 'Single agent's wealth per model step'}, xlabel='Step', ylabel='amount of money'>



The line chart shows a single agent's wealth for each model iteration.

Creating an agent-based model from scratch

Distribution of wealth after model run

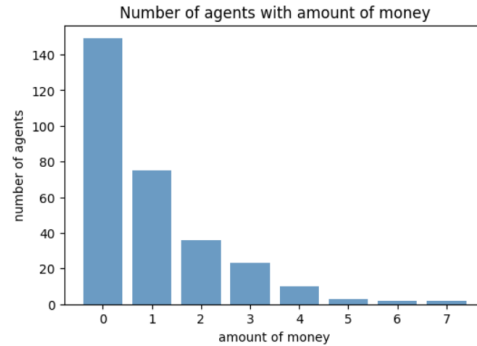
```
# Distribution of wealth after model run
end_wealth = agent_wealth.xs(n_steps - 1, level="Step")["Wealth"]
end_wealth

# Values
df_bar = end_wealth.value_counts()
num_agents = list(df_bar)
wealth = list(df_bar.index.values)
y_pos = np.arange(len(wealth))

# Bar chart
fig, ax = plt.subplots(figsize=(6,4))

ax.bar(y_pos, num_agents, align='center', color='steelblue', alpha=0.8)
ax.set_title('Number of agents with amount of money')
ax.set_xlabel('amount of money')
ax.set_ylabel('number of agents')

# Show graph
plt.show()
```



The bar chart summarizes the wealth of all agents after model run. It shows that a small number of agents holds much more money than most other agents leading to inequality (which is expressed by a lower Gini coefficient).

Learning Objectives (lessons and exercises)

You can ...

- explain what a cellular automaton is
- explain and name the motivation to use cellular automata
- explain the components of a cellular automaton
- explain rules of a cellular automaton using the example of Conway's Game of Life
- explain what an agent-based model is and how it works
- explain the components of an agent-based model

Literature and useful resources for further reading

- Ilievski, V. (2021). Simple but Stunning: Animated Cellular Automata in Python. Link: <https://towardsdatascience.com/simple-but-stunning-animated-cellular-automata-in-python-c912e0c156a9>
- Lees, E. (2020). Elementary Cellular Automata. <https://matplotlib.org/matplotlibblog/posts/elementary-cellular-automata>
- Mesa: Agent-based modeling in Python 3+ (n.d.). <https://mesa.readthedocs.io/en/latest>
- Wolfram, S. A New Kind of Science. Champaign, IL: Wolfram Media, pp. 23-60, 112, and 865-866, 2002.