University of Stuttgart
Germany

Florian Schröder

Can Julia win the Game of Life?

July 25, 2025

Supervisor: Gerasimos Chourdakis

What's that?

Overview

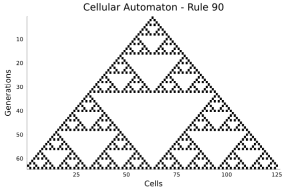


Figure: 1D Cellular Automata

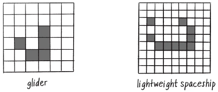


Figure: 2D Cellular Automata

Figure: Julia Implementation[1]

[1] `https://commons.wikimedia.org/wiki/File:Julia_Programming_Language_Logo.svg`

What are Cellular Automata?

- Discrete models: grid of cells, each with a state
- Simple, local rules $\rightarrow$ complex global behavior
- Used for simulating complex systems (urban, physics, biology)
- Example: Conway's Game of Life

1D Cellular Automata: Theory

- Cells in a 1D array, each with a state (e.g., 0 or 1)
- Neighborhood: e.g. cell itself + left/right neighbors
- Update rules: Neighborhood $\rightarrow$ next state

1D CA: Ruleset

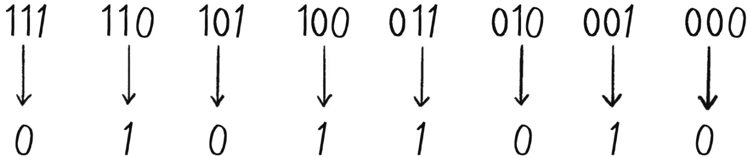


Figure: Example: Mapping neighborhood to next state (here rule number 90)

1D CA: Visualization

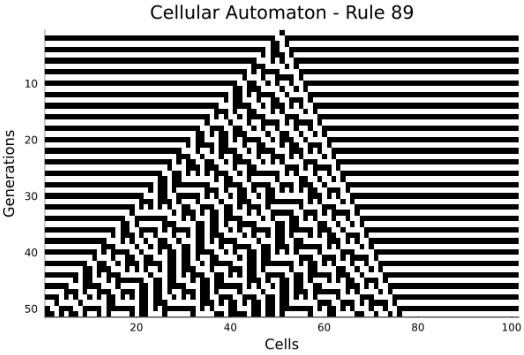


Figure: Visualization of generations as rows in a 2D grid

Wolfram Classification

- **Class 1:** Uniformity (stable)
- **Class 2:** Repetition (periodic)
- **Class 3:** Random (chaotic)
- **Class 4:** Complexity (mix of order/chaos)

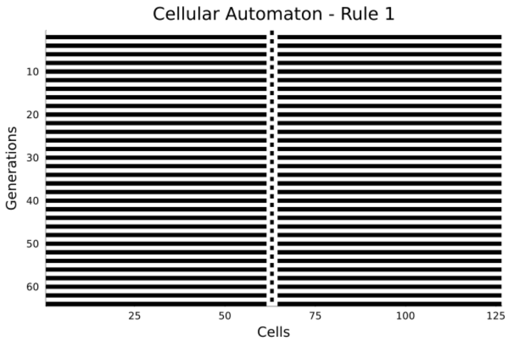


Figure: Rule 1: An example of Class 2 (repetitive)

2D Cellular Automata: Game of Life

- Grid of cells, each with 8 neighbors
- **Rules:**
  - Birth: exactly 3 alive neighbors
  - Survival: 2 or 3 alive neighbors
  - Death: < 2 or > 3 alive neighbors

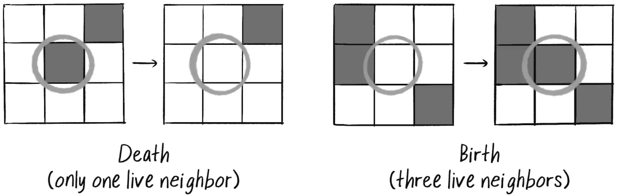
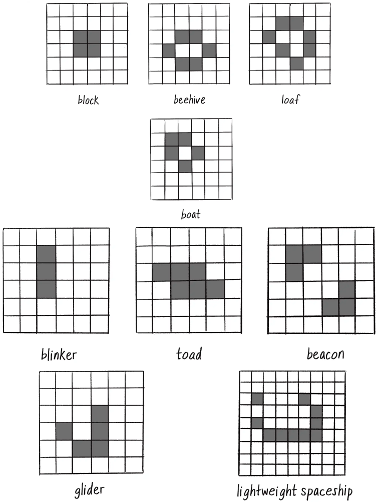


Figure: Examples of scenarios in the Game of Life[2]

[2] https://natureofcode.com/static/99dd5b32b72ce094d5a77f749c2ab9f0/3ca65/07_ca_28.webp

Game of Life: Patterns

- **Stable:** Do not change
- **Oscillators:** Repeat after *n* steps
- **Spaceships:** Move across grid
- **Guns:** Emit other patterns



1D CA in Julia - apply_rule function

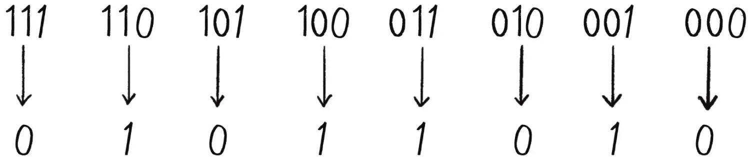


Figure: Example: Mapping neighborhood to next state (here rule number 90)

Listing 1: apply_rule function for 1D CA

```
function apply_rule(left::Int, center::Int, right::Int, rule_number::Int)::Int
    neighborhood_value = left * 4 + center * 2 + right * 1
    return (rule_number >> neighborhood_value) & 1
end
```

Rest of 1D CA Implementation

- **Initialization:** Create initial state (e.g., random)
- **Update:** Apply rules to each cell
- **Visualization:** Display generations as rows in a 2D grid

Game of Life in Julia

- **Initialization:** Create intial state (e.g., random)
- **Update:** Apply Game of Life rules to each cell
- **Visualization:** Display grid as a row of 2D images or a GIF

Extending Game of Life: Infection Simulation

- **Infection Model:** Cells can be healthy, infected, dead/not existing
- **Rules:**
  - Infected cells infect neighbors
  - Infected cells can die
  - Healthy cells can reproduce
- **Visualization:** Display grid with different colors for each state

Why Julia for Simulation?

- **Performance:** Fast, optimized for numerical computing
- **Syntax:** Concise, math-like, less verbose than JavaScript
- **Ecosystem:** Rich packages for simulation and visualization

Discussion

- Cellular automata are used in simulations like Lattice Boltzmann for fluid dynamics.
- LBM is efficient and parallelizable due to local interactions.
- PDEs simulate continuous systems but need complex numerical methods.
- Cellular automata use simple, discrete rules; not direct PDE replacements.
- Time steps and update rules are common in many simulation methods.

Conclusions / Relation to seminar

- Covered cellular automata basics, especially Game of Life.
- Showed Julia implementation and visualization.
- Simple rules yield complex behavior.
- Julia is fast and flexible for simulations.
- Cellular automata help study emergent phenomena.

Thanks

Thank you for your attention!