Game of Life with Customizable Rules and Non-Binary States

Tim LaVake
Rutgers University
Piscataway, NJ, USA
tjl142@scarletmail.rutgers.edu

Arpeet Barvalia
Rutgers University
Piscataway, NJ, USA
aab353@scarletmail.rutgers.edu

Danny Watson
Rutgers University
Piscataway, NJ, USA
dmw277@scarletmail.rutgers.edu

Abstract - This project presents an interactive framework to simulate and visualize Conway's Game of Life using Python. The implementation spans three levels of complexity, starting with the classical rules of Conway's Game of Life, progressing to user-defined rule configurations, and culminating in simulations of multi-state systems. These models enable users to explore emergent behaviors, test alternative cellular automata rules, and simulate real-world processes like disease spread. With dynamic visualizations, customizable inputs, and data persistence in CSV and JSON formats, this project demonstrates how computational tools can model complex systems efficiently.

Keywords - Game of Life, Non-Binary States, NumPy, Matplotlib, CSV, JSON, Python Programming

I. Project Description

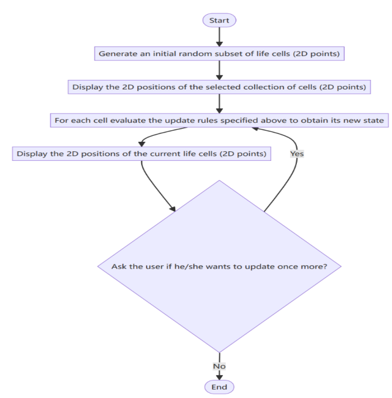
Conway's Game of Life is a cellular automaton that illustrates how simple, local rules can result in intricate, emergent global patterns. Each cell in a grid toggles between alive and dead states based on the states of its neighbors. This project aimed to implement the Game of Life framework while enabling customizations and extending the model to multi-state systems. The framework is implemented in Python using libraries like NumPy for computational efficiency and Matplotlib for visualizations. It supports user-defined input parameters such as grid size, probabilities for initial cell states, and update rules. Outputs include interactive visualizations of the grid over iterations, saved grid states in CSV files, and rules stored in JSON files. These features make the simulation accessible and reproducible for both educational and research purposes.

II. Input/Output Format Specification

Input - The project allows users to define the grid size by specifying the number of rows and columns. Users can also configure the initial states and their respective probabilities, along with a JSON file to specify the rules for state transitions. The simulation runs for a user-defined number of iterations, with an option to save the initial and final states as separate CSV files for analysis and reference.

Output - The project features real-time visualization of grid updates, with dynamic displays generated after each iteration using Matplotlib. Additionally, the framework saves the initial and final grid states as CSV files, ensuring the data is preserved for further analysis and reproducibility.

Implementation Highlights:



The grids are initialized according to user-defined probabilities for each state, ensuring flexibility in starting conditions. Once initialized and an alive cell was on the border, the grid expands by adding a row and column. Visualization is handled through Matplotlib, rendering grids with distinct colors to represent different states for clear and intuitive displays. Cell updates occur simultaneously based on neighbor conditions and user-defined rules, allowing for complex behavior modeling. The framework leverages NumPy for optimized computations, enabling efficient handling of large grids and ensuring smooth performance.

III. Implementation

- Designed the grid as a 2D array where each cell represents a live or dead state.
- Initialized the simulation with a pre-defined grid pattern.
- Optimized the simulation to handle grids of various sizes efficiently.
- To enhance performance, the grid operations are vectorized using NumPy, ensuring that even large grid configurations can be processed efficiently without loops.
- Visualization was implemented using Matplotlib's animation feature, creating a seamless progression of generations. The animation highlights the evolving patterns and behaviors of the cell grid.

IIII. Sample Input/Outputs

Sample Input:

Grid Size: 20x20
States: [0, 1, 2]
Probabilities: [0.5, 0.3, 0.2]
Rules File: rules.json

```
{
 "0": [{"turn_to": 0}],
 "1": [{
  "neighbor_to": {
   "if": [{"at_least": 1, "at_most": 9, "type": 2}],
   "then": {"probability": [{"value": 0.25, "then":
          {"turn_to": 2}}]}
  }
 }],
 "2": [{"probability": [{"value": 0.5, "then":
      {"turn_to": 0}}]}]
}
```

Sample Output:

Initial Grid State Visualization:

- Black (0): Removed
- Green (1): Susceptible
- Red (2): Infected

Updated Grid State Visualization:
After 10 iterations, cells transition based on the defined rules, illustrating dynamic changes in states:

- Infected (Red) cells have spread to neighboring Susceptible (Green) cells.
- Some cells transitioned to removed (Black) state based on recovery rules.

V. Programming Language and Libraries Used

Python:

- NumPy: For initializing and updating grid states.
- Matplotlib: For rendering state visualizations.
- JSON: For loading user-defined rules.

VI. Conclusions

This project successfully implemented Conway's Game of Life and extended its scope to user-defined rules and multi-state systems. The use of Python and its libraries made the framework efficient and accessible. Key achievements included the interactive visualization of cellular dynamics, the ability to experiment with alternative rules, and the extension to multi-state systems for modeling real-world phenomena. The framework demonstrates the power of cellular automata to model complex systems and provides a foundation for further exploration. Potential improvements include developing a graphical user interface, parallelizing computations for larger grids, and integrating machine learning techniques to discover novel patterns.

Potential Applications:

Epidemiology: Modeling the spread and containment of diseases through Susceptible-Infected-Recovered (SIR) dynamics. Ecosystems: Simulating predator-prey relationships or plant growth patterns. Urban Planning: Exploring traffic flow and population density models.

Future Enhancements:

Adding performance optimizations for massive grids using GPU acceleration. Developing an interactive GUI for intuitive configuration and real-time visualization. Expanding support for additional state types and more complex neighbor interaction rules.

Sample Inputs/Outputs (Task 1)

Inputs:

10
Enter # of rows in initial grid (ex: 20): (Press 'Enter' to confirm or 'Escape' to cancel)

10
Enter # of columns in initial grid (ex: 20): (Press 'Enter' to confirm or 'Escape' to cancel)

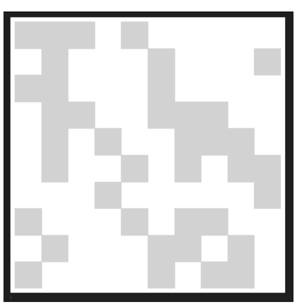
0.4
Enter probability of a cell being alive (0.0 to 1.0, ex: 0.2): (Press 'Enter' to confirm or 'Escape' to cancel)

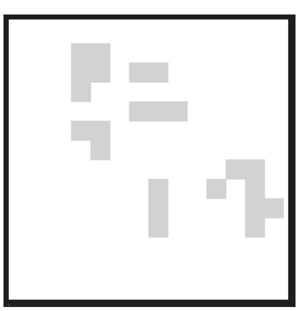
5
Enter number of iterations to update (ex: 10): (Press 'Enter' to confirm or 'Escape' to cancel)

no
Do you want to perform more iterations? (input yes/no): (Press 'Enter' to confirm or 'Escape' to cancel)

yes
Do you want to save the initial and final states of the grid? (input yes/no): (Press 'Enter' to confirm or 'Escape' to cancel)

Outputs (Initial and Final Grids):





Sample Inputs/Outputs (Task 2)

Inputs:

10
Enter # of rows in initial grid (ex: 20): (Press 'Enter' to confirm or 'Escape' to cancel)

10
Enter # of columns in initial grid (ex: 20): (Press 'Enter' to confirm or 'Escape' to cancel)

0.4
Enter probability of a cell being alive (0.0 to 1.0, ex: 0.2): (Press 'Enter' to confirm or 'Escape' to cancel)

1
Enter b1 (if a dead cell has at least b1 neighbors and at most b2 neighbors, then it becomes alive, ex: 3): (Press 'Enter' to confirm or 'Escape' to cancel)

1
Enter b2 (if a dead cell has at least b1 neighbors and at most b2 neighbors, then it becomes alive, ex: 3): (Press 'Enter' to confirm or 'Escape' to cancel)

2
Enter d1 (if an alive cell has at least d1 neighbors and at most d2 neighbors, then it stays alive, ex: 2): (Press 'Enter' to confirm or 'Escape' to cancel)

3
Enter d2 (if an alive cell has at least d1 neighbors and at most d2 neighbors, then it stays alive, ex: 2): (Press 'Enter' to confirm or 'Escape' to cancel)

7
Enter number of iterations to update (ex: 10): (Press 'Enter' to confirm or 'Escape' to cancel)

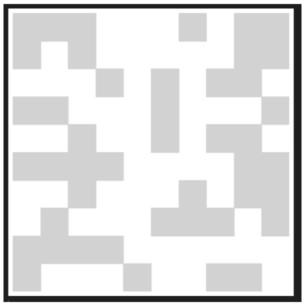
yes
Do you want to perform more iterations? (input yes/no): (Press 'Enter' to confirm or 'Escape' to cancel)

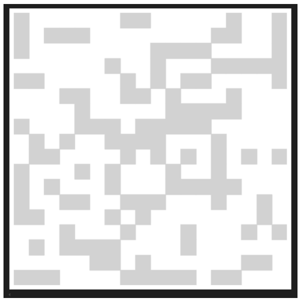
1
Enter number of iterations to update (ex: 10): (Press 'Enter' to confirm or 'Escape' to cancel)

no
Do you want to perform more iterations? (input yes/no): (Press 'Enter' to confirm or 'Escape' to cancel)

no
Do you want to save the initial and final states of the grid? (input yes/no): (Press 'Enter' to confirm or 'Escape' to cancel)

Outputs (Initial and Final Grids (on next page)):





Sample Inputs/Outputs (Task 3)

Inputs:

8
Enter # of rows in initial grid (ex: 20): (Press 'Enter' to confirm or 'Escape' to cancel)

7
Enter # of rows in initial grid (ex: 20): (Press 'Enter' to confirm or 'Escape' to cancel)

0,1,2
Enter possible cell states (seperate each state by a ',', ex: 0,1,2): (Press 'Enter' to confirm or 'Escape' to cancel)

0.5,0.3,0.2
Enter probabilities for each state (seperate each probability by a ',', inputs must sum to 1, ex: 0.5,0.3,0.2): (Press 'Enter' to confirm or 'Escape' to cancel)

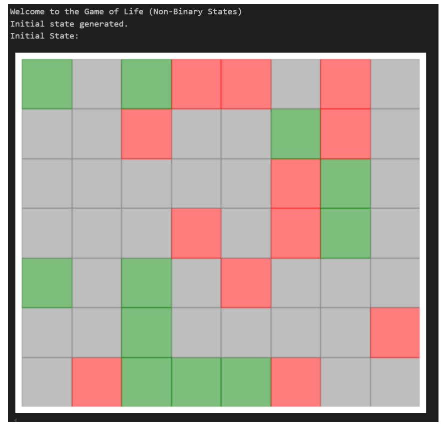
C:\Users\Danny\Downloads\rules.json
Enter the path to the JSON file with rules: (Press 'Enter' to confirm or 'Escape' to cancel)

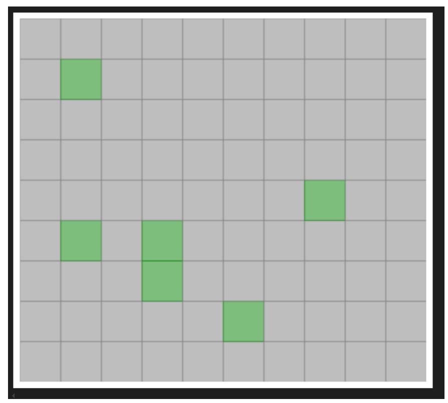
11
Enter number of iterations to update (ex: 10): (Press 'Enter' to confirm or 'Escape' to cancel)

no
Do you want to perform more iterations? (input yes/no): (Press 'Enter' to confirm or 'Escape' to cancel)

no
Do you want to save the initial and final states of the grid? (input yes/no): (Press 'Enter' to confirm or 'Escape' to cancel)

Outputs (Initial and Final Grids):





References

Gardner, M. "The Fantastic Combinations of John Conway's New Solitaire Game 'Life'." Scientific American, 1970.

NumPy Documentation: https://numpy.org/doc/

Matplotlib Documentation: https://matplotlib.org/stable/contents.html

Conway's Game of Life Resources: https://playgameoflife.com

Gardner, Martin (October 1970). "The fantastic combinations of

John Conway's new solitaire game 'life'" (PDF). Mathematical

Games. Scientific American. Vol. 223, no. 4. pp. 120–123.

doi:10.1038/scientificamerican1070-120. JSTOR 24927642.

Berlekamp, E. R.; Conway, John Horton; Guy, R. K. (2001–2004).

Winning Ways for your Mathematical Plays (2nd ed.). A K Peters

Ltd.

Izhikevich, Eugene M.; Conway, John H.; Seth, Anil (2015-06-21).

"Game of Life". Scholarpedia. 10 (6): 1816.

Bibcode:2015SchpJ..10.1816I. doi:10.4249/scholarpedia.1816.

ISSN 1941-6016.

"NaiveLife Emulated: A reading-order simulation of Life".

ConwayLife.com. 24 May 2020.

Goucher, Adam. "Re: Thread For Your Accidental Discoveries".

ConwayLife.com.

Ian07. "Re: Strange spaceship that is supposed to be impossible

and infinite cell spread". ConwayLife.com. "I'm pretty sure this is

because you've accidentally created an implementation of

what's sometimes known as NaiveLife (as it's a common mistake

made by many people coding CGoL for the first time):"

Brown, Nico; Cheng, Carson; Jacobi, Tanner; Karpovich, Maia;

Merzenich, Matthias; Raucci, David; Riley, Mitchell (5 December

2023). "Conway's Game of Life is Omniperiodic".

arXiv:2312.02799 [math.CO].

"LifeWiki:Game of Life Status page - LifeWiki". conwaylife.com.

Stone, Alex (2024-01-18). "Math's 'Game of Life' Reveals LongSought Repeating Patterns". Quanta Magazine.

"Conway's Game of Life". Rosetta Code. June 7, 2024.