Game Of Life Adaptation

Krisha Patel
Rutgers University, NJ, USA
ksp162@rutgers.edu

Alex Dodson
Rutgers University, NJ, USA
awd46@rutgers.edu

Julian Esquivel
Rutgers University, NJ, USA
jse84@rutgers.edu

***Abstract*— We developed three variations of cellular automata using Python in combination with the NumPy and Matplotlib libraries. The variations are as follows: (1) John Conway's original Game of Life, (2) a customizable binary Game of Life allowing user-defined rules, and (3) a non-binary version modeled after the SIR epidemiological framework (Susceptible, Infected, Removed). The binary games operate within a two-state system where cells are either "alive" or "dead." In contrast, the SIR-inspired version introduces a multi-state dynamic, enabling more complex simulations. Versions 2 and 3 allow users to create their own rules, leading to interesting patterns. This project was given by James Abello and Haoyang Zhang for CS210.**

I. Project Description

John Conway's "Game of Life" is a zero-player cellular automaton developed in 1970, where cells on a grid evolve based on simple rules: they can live, die, or reproduce depending on the states of neighboring cells. Despite its simplicity, the Game of Life demonstrates how complex patterns and behaviors can emerge from basic interactions, including oscillators, gliders, and self-replicating structures. It has captivated mathematicians, computer scientists, and hobbyists alike, inspiring studies in complexity theory, artificial life, and computational universality. Beyond academic significance, it popularized the concept of emergent behavior, showcasing how intricate systems can arise from simple rules—a principle with implications across disciplines, from biology to artificial intelligence.[1]

Our project explores two variations of this famous automation (in addition to the original). The first variation replicates the original game, but instead of it being a zero-player automation, it is now a one-player automation. We created a customizable version of the Game of Life that allows users to input their own rules, offering flexibility for experimenting with different patterns and dynamics within the binary framework. These implementations provide insight into emergent behavior and the complex interactions arising from simple local rules.

In addition to the binary user model, we introduced a non-binary cellular automaton inspired by the SIR epidemiological framework, which classifies cells as Susceptible, Infected, or Removed. This model expands the traditional binary approach to include multiple states, simulating processes that evolve in a more nuanced manner. By transitioning beyond "alive" or "dead" states, the SIR-inspired Game of Life demonstrates how cellular automata can be adapted to represent real-world phenomena, such as the spread of disease. This project utilizes the NumPy and Matplotlib libraries in Python for implementation and visualization. Overall, this project highlights the versatility of cellular automata and their potential applications in scientific modeling and simulation.

A. Division of Labor

This project took approximately 7 hours to complete, not including this paper and corresponding videos. Alex Dodson completed Task 1 and half of Task 2. Julian Esquival completed Task 2 and worked on the baseline for Task 3. Krisha Patel completed Task 3 and adjusted Tasks 1 and 2 to ensure the program could work with a multitude of test cases. Additionally, she worked on this project description documentation. Descriptions of each group member's task and sample scenarios can be found below.

B. Task 1 - The Original

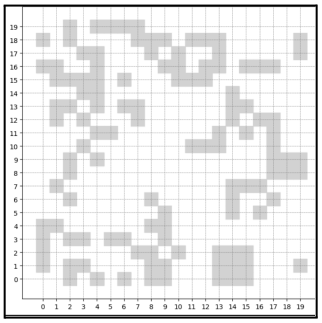

Figure 1.

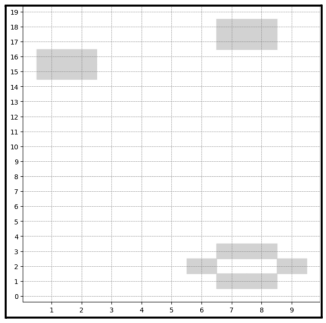

Figure 2.

Task 1 replicates John Conway's "Game of Life." To run this program, the user needs to specify the number of rows and columns and the probability of a cell being alive. Figure 1 is an example output of the initial state with rows = columns = 20 and the likelihood of a cell being alive equal to 0.30. This initial state is achieved using NumPy to generate a row x column array that randomly contains a value between 0 and 1. Then, the value in each cell is compared to the probability of a cell being alive, and the color gray is assigned if it is alive, and white is assigned if it is not. To determine whether the any cell should live or die, these basic rules are followed:

1. If a dead cell has at least 3 neighbors alive, then it comes back to life.
2. If an alive cell has at least 2 and at most 3 neighbors alive, then it continues to be alive.
3. If an alive cell has less than 2 or more than 3 neighbors alive, then the cell dies of loneliness/overcrowding. [2]

Running 100 iterations of this simulation results in the permanent structure shown in Figure 2. The user can view each display after each iteration or just a few display outputs from the beginning. Many different patterns can emerge from the Game of Life, such as oscillators, gliders, and self-replicating structures. Oscillators are formed when a pattern repeats itself after a fixed number of repetitions. Figures 4 and 5 show an example of an oscillating pattern that was achieved after 120 repetitions on a simulation with a 0.25 rate of being alive.

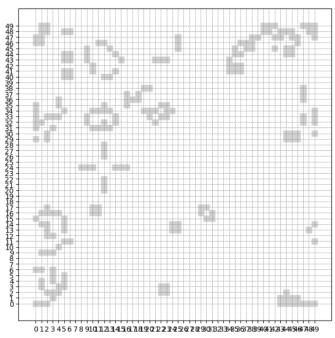

Figure 3.

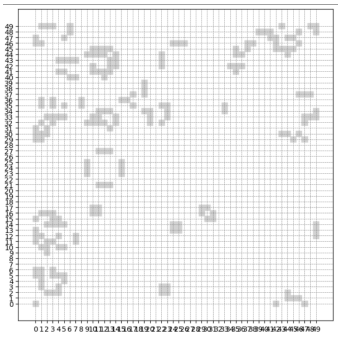

Figure 4.

C. *Task 2 - The Customizable*

In the second task, we incorporate further user interaction by allowing users to create rules. This is done by prompting the user to enter four integers, d_1, d_2, b_1, and b_2, between 0 and 8 (because each cell has only eight neighbors) in addition to the inputs from Task 1. The rules are as follows:

1. If a dead cell has at least b_1 and at most b_2 neighbors alive, then it comes back to life.
2. if a alive cell has at least d_1 and at most d_2 neighbors alive, then it continues to be alive.
3. If a living cell has less than d_1 or more than d_2 neighbors alive, then the cell dies.

Notice that if $b_1 = b_2 = d_2 = 3$ and $d_1 = 2$, the rules are the basic rules from Task 1.

This variation requires the user to be more creative when designing rules, at least if they want to see a pattern emerge. If they use random numbers without much thought behind them, there is a chance that the simulation will fizz out and all cells die. Figure 3 shows an example of this, using the following set of rules: $b_1 = 4$, $b_2 = 6$, $d_1 = 5$, $d_2 = 6$, probability of life = 0.27, rows = 10, columns = 20.

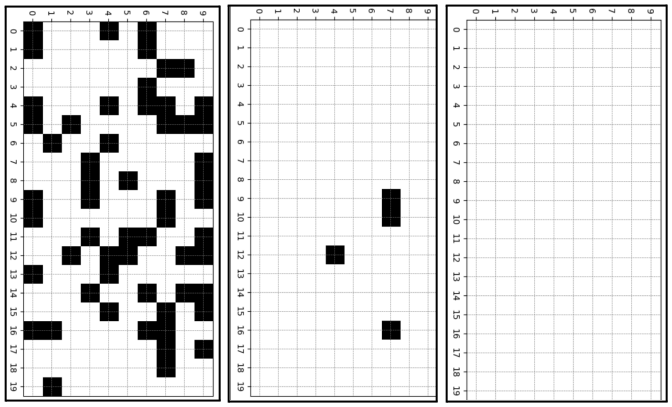

Figure 5.

D. *Task 3 - SIR*

Task 3 differs from the others because it introduces the concept of non-binary cells. This means cells can have multiple states, particularly susceptible, infected, and removed. Nodes start out as 'susceptible,' except for a specific few, which start out as 'infected.' Each iteration allows the infected nodes to pass their infection to a susceptible neighbor. The chance that the susceptible neighbor cell becomes afflicted with the infection is equal to the transmission rate, which the user can specify. Infected nodes then transition to the removed state. A cell that is considered "removed" cannot infect other cells or become infected itself. For this task, the user inputs the following:

1. rows and columns (as integers)
2. The probability of a cell being removed
3. The probability of a cell being susceptible
4. The probability of a cell being infected
5. The probability of a disease spreading to its neighbors, called the transmission rate
6. The probability of a cell recovering- recovery rate
7. The number of neighbors it takes for a susceptible cell to get infected (integer from 0-8)

Please note that inputs 2-4 must be decimal values that add up to 1. Additionally, every input aside from 1 and 8 must be decimal values in between 0 and 1. An example input could be rows = columns = 10, removed probability - 0.2, susceptible probability = infected probability = 0.4, transmission rate = .25, and recovery rate, 0.50. Figure 6 shows the initial state created by those rules. Red represents infected cells, blue represents susceptible cells, and white represents removed cells.

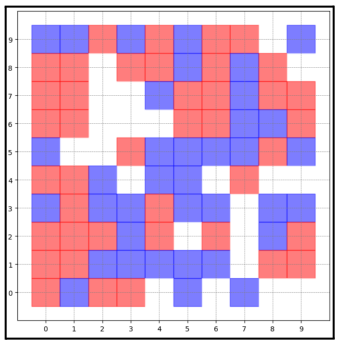

Figure 6.

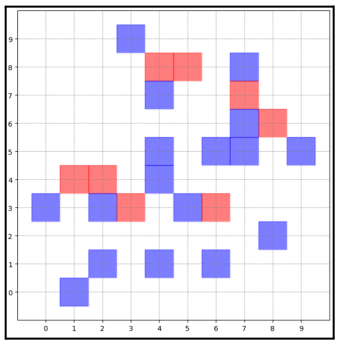

Figure 7.

Figure 7 shows the grid after 3 iterations.

II. Bonus - Interface

To enhance user experience and visualization, Alex Dodson utilized Matplotlib to create an interactive display that dynamically updates as the simulation progresses. This eliminates the need for users to scroll through individual iterations, allowing them to observe patterns and dynamics in real-time seamlessly. By animating the grid within a single visualization window, users can focus on the evolution of the system without distraction, making it more intuitive and engaging.

This real-time interface also significantly improves the presentation of our work, especially in scenarios like our demonstration video. It provides a clear and immediate understanding of how the rules and states influence the simulation, making it an invaluable tool for showcasing the potential of cellular automata to both technical and non-technical audiences.

III. Applications for the Non-CS/DS User

While the Game of Life is a computer program created by a mathematician, the applications of this simulation are not limited to mathematics or computer/data science. This is because this simulation has multitudes of applications. The basic version of Game of Life can be used in:

1. Habitat Simulation – by modeling how species interact within constrained spaces and environments.
2. Neural Networks– by creating simple models that can help neuroscientists understand how neurons in the brain fire and interact over time.

The SIR (susceptible, infected, removed) version of this game can be used for other scenarios. For example:

1. The spread of infectious diseases throughout a population. In this situation, a 'removed' cell would be

someone who passed away or has immunity.

2. A wildfire across a wild landscape. A ‘removed’ cell would be a burned tree in this situation.

Even the spread of inventions or ideas can be modeled with a Game of Life simulation– particularly, a SIS (Susceptible, Infected, Susceptible) simulation. The Game of Life can be adapted to fit many different situations. Take the spread of a philosophy or political idea, for example. The idea could spread in a certain region, die out, and remerge a few years later [3]. This does not follow the SIR method used in task 3 because the people/cells never really become removed, but they always remain susceptible. Thus, we adopt a new version of Game of Life– SIS.

IV. Conclusion

This project demonstrates the power and versatility of John Conway’s ‘Game of Life’ by showing how cellular automation can be a tool for modeling and simulation. By implementing this program, we explored how simple rules can produce complex and often unpredictable patterns. The customizable two-state (alive or dead) version provided users with a hands-on way to experiment with emergent behavior. At the same time, the SIR-inspired model extended the framework into multiple states, allowing cells to transition from susceptible to infected to removed.

These implementations underscore the broad applicability of cellular automata beyond computer science, including ecology, neuroscience, and social sciences. The ability to simulate processes like disease spread, habitat interactions, and idea propagation highlights the relevance of these models in addressing modern challenges.

Through this project, we gained valuable insights into emergent behavior, rule-based systems, and computational modeling. The combination of Python, NumPy, and Matplotlib allowed for efficient computation and intuitive visualization, enhancing the depth of exploration and the accessibility of the results.

V. References

[1] Martin, E. (n.d.). *Play John Conway's Game of Life*. Play Game of Life. https://playgameoflife.com/info

[2] YouTube. (2012). *Stephen Hawking's The Meaning of Life (John Conway's Game of Life segment)*. YouTube. https://www.youtube.com/watch?v=CgOcEZinQ2I

[3] Simler, K. (2019). *Going critical*. Melting Asphalt. https://meltingasphalt.com/interactive/going-critical/