University at Buffalo

CSE 368

Fall 2024

Evolving Patterns in Conway's Game of Life to Reproduce User-Defined Images

Elisha Amiri (elishaam@buffalo.edu)

Animesh Saha (asaha5@buffalo.edu)

Seohyeon Kim (skim243@buffalo.edu)

Jacob Szczudlik (jacobszc@buffalo.edu)

Introduction

Conway's Game of Life is a well-known cellular automaton illustrating how simple, local rules can yield complex emergent behavior. Traditionally, Conway's Game of Life begins with an initial pattern of "alive" (ON) and "dead" (OFF) cells on a grid, and each iteration updates this pattern according to fixed rules governing cell survival, birth, and death. Our project sought to extend this concept by steering the evolving patterns toward a user-defined target image, effectively using the game's cellular automaton rules to recreate a given image starting from random initial conditions.

Instead of relying solely on the predefined rules of Conway's Game of Life, we introduced a mechanism that gradually biases the evolving grid toward the target pattern, effectively blending the classical emergent complexity with a directed form of pattern formation. Initially, we considered applying reinforcement learning to guide the game's evolution toward a target image, but as our project evolved, we shifted to a more deterministic approach: each generation, the algorithm compares the current grid to the target image and gently nudges cells to match that pattern. This approach maintains the spirit of Conway's rules as a baseline, while adding a probabilistic adjustment that becomes more aggressive over time, ensuring the final grid configuration aligns closely with the user's chosen image.

The solution leverages Python for rapid prototyping, PIL (Python Imaging Library) for image processing, NumPy for efficient numerical computation, and Matplotlib for real-time visualization and animation. We also implemented an interactive user interface using Tkinter, allowing users to upload their own images or select from a set of

predefined images, preview their choice, and run the animation seamlessly. This interface made the system accessible and user-friendly, transforming what began as a console-driven script into a visually appealing and intuitive application.

Section 1 - Technologies/Techniques Explored

- Pillow(PIL): We used this library for handling the image. For our project, this library allows us to load, resize and convert the image into binary grids easily. It was brand new to us. We found out about it while searching image manipulation techniques on Google. We were looking for ways to efficiently process images for our project, and we found Pillow would be suitable for our project.

- NumPy: We used this library for handling the grid as a 2D array and performing operations(for Conway's game of Life rules). We already knew about this, because we learned about this from the lecture. From the lecture and PA3, we learned about efficiency to handle the computationally update for Conway’s Game of Life. That’s why we decided to use this for our project.

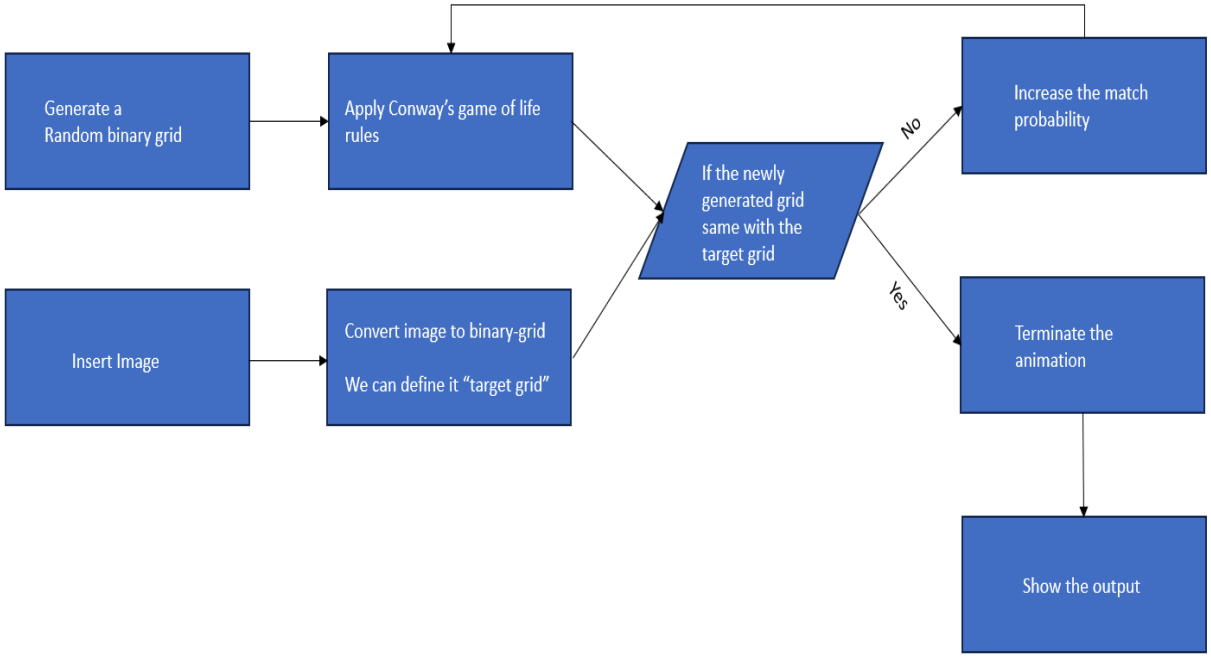
- Matplotlib : We used Matplotlib for the animation purpose. To display the grid view on screen, this framework was useful. We were aware of basic functionality of matplotlib and its usage but using this in our project was a bit challenging for us at first specifically with larger grids since we needed to manage frame updates efficiently. The funcAnimation in matplotlib provided a framework to update the visualization in a frame by frame rendering loop.

- Tkinter: We used this library to create a GUI for our project. It allowed us to build an interface where users can upload the image or select image from preselected options. Initially, our program required to set the image path directly into the code, making it not user-friendly. So we decided to implement a feature that allows users to easily upload their own image through the GUI. We knew that Tkinter is used for creating user interfaces(UI), but it was our first time actually implementing it in a project

- Conway’s game of Life rules: We used the rules of Conway’s game of life in our project and using the rule we created our program which can generate images following the set of rules of Conway’s game of life. The rule says there can be two states of a cell: Dead or Alive. If the cell has fewer than two live neighbors, it dies. If it has more than three live neighbors, it also dies. It will survive only if it has 2 or 3 live neighbors. The dead cell will become alive if it has 3 live neighbors. We used the basic rule of Conway's game of life and we created an algorithm to effectively use the rule to generate a picture. We came to know about the rule from the programming assignment 1 of CSE220: Systems Programming where we implemented the game’s rules and created a program which follows the rules. Later, in lecture 368, we got to know a bit of the background of the game. We faced challenges in implementing the rules and automating it since the algorithm was complex where we had to design a convergence logic while following the rules.

Section 2 - Methods/Methodology

Overall algorithm



First, the user will select or upload an image. Image will be converted to a binary grid(It represents the target pattern. Each cells classified as “ON” or “OFF” based on their brightness. Also a random binary grid is generated as the initial state for Conway’s game of life Comway’s game of life rules are applied with probabilistic adjustment to align with the target image. Target grid convergence: At each frame, our algorithm compared the current grid with the target grid. For this we

used a dynamic match probability, where the probability increases with each frame where match probability = min(1, 0.1 +frameNum *0.02).

1) Helper functions

We used 8 helper functions as follows:

a. image_to_grid function converts an inserted image to a binary grid (black and white/ ON and OFF) based on its brightness. We opened and resized the image by using PIL image.Then we converted the image into a Numpy array (for using 2D arrays efficiently).If the grid grayscale is less than 128, the value of the grid should be ON(1 = White), otherwise, its value is OFF (0 = Black)
b. random_grid function generates a random initial grid to start the game. It generates a random size x size array values between 0 and 1.For example, if the value is more than 0.5 then, it will be ON (white) and if it is less than 0.5, it will be OFF (black)
c. Update function is our main functionality of the project. Here we have -

   1) Checked if current grid matches the target grid, if so it will stop the animation by using Matplotilib

   2) Applied Conway's game of Life rules to update the state of each cell in the grid. For each cell, we calculate the sum of its neighbors and use this

to determine if the cell will survive, die or be born. Apply Conway's rules are as follows:

- If the current cell is alive(ON) :
    a. smaller than 2 neighbor: Die,
    b. 2 or 3 neighbors: alive
    c. more than 3 neighbor: die.
- If the current cell is dead(OFF) :
    a. 3 neighbor: born

3) Gradually increasing the match_probability of matching cells.Randomly updating cells to align with the target grid The match_probability starts at 0.1 and increase linearly with each animation frame

d. process_and_animate function initializes and runs the animation for the game. What we did in this function -
    1) Converts the input image into a binary grid by using image_to_grid()
    2) Initialize the starting grid by using random_grid()
    3) Set up some figures (Animation duration and timing, matplotilib)
    4) Display the animation
e. display_delected_image function is for loading, resizing and displaying a preview of a selected image. When the user upload the image, it is displayed in the preview section

f. choose_preselected_image function handles the selection and validation of a preselected image. It retrieves the image path based on the selected image name
g. upload_image function allows the user to upload an image file using a file dialog. And stores the image path for further use during the animation process
h. start_animation function starts the animation. It checks if the selected image exists. Then close the tkinter window and then starts the animation by calling the process_and_animate()

3) How probability works

We used probabilistic adjustments. This algorithm follows some form of reinforcement learning where each frame tries to maximize alignment to generate desired outcome. Our variable named match_probability tries to increase the probability of each generated grid to be like a targeted grid. If the targeted grid is not achieved, our algorithm keeps generating new grids with higher probability of success.

We think this algorithm is rooted in AI principles such as optimization loops. This dynamic adjustment is similar to a simplified reward system. A higher match probability increases the likelihood of achieving the goal. It ensures that natural attributes ariging from Conway's game of life rules are not immediately ignored,

and that they can be progressively aligned with the goals while maintaining direction toward the desired outcome.

Difficulty what we faced

- Initially, we were not able to generate the image accurately and the result was always either completely different than we expected so we couldn’t find the result due to the infinite loop of generation. These issues were mostly because of issues with our match probability. So, instead of using a fixed probability, we chose for a dynamic probability that increased with each iteration.

Successes

- We have managed to make our algorithm work mostly accurately. Our algorithm program is able to generate some complex images using the methods discussed above. We also introduced taking inputs from users and generating the requested image using our algorithm. Finally, we have been able to blend dynamic probability with Conway's game of life together.

Failure

- Sometimes, it generates oversimplified patterns due to not having fine-tuning. This results in a bit of an inaccurate outcome.

For example, when we used a photo of white-furred dog, the conversion to the binary grid caused the details to be lost, resulting in a grid that did not accurately represent the original image. And when we tested with a triangle image, the algorithm failed to generate the output correctly, due to simplified the shapes into binary values.

The reason for this problem is binary conversion depends on brightness, which can oversimplify images with subtle detail.

Things from class that helped out this process

- We learned the basics of numpy and reinforcement learning from class which initially helped us set up the program and proceed with implementing our algorithm efficiently.

Section 3 - Results

For the most part I would say that the result of this project was a success. Our initial goal was to take a famous example of what an early AI project might look like, that being a game that when given a simple rule set can go through some sort of decision making process to give you complex results, and add some sort of probability distribution to that decision making to see how much we can influence the outcome in unique ways. To that end we were successful.

```
match_probability = min(1, 0.1 + frameNum * 0.02)
    for i in range(size):
        for j in range(size):
            if np.random.rand() < match_probability:
                new_grid[i, j] = target_grid[i, j]
    img_plot.set_data(new_grid)
    grid[:] = new_grid[:]
    return img_plot
```

This match_probibilty is the main driver of probabilistic decision making in our project. Before i talk about the results ill give a brief overview of the idea. Based on the frame, this function will update our 2d grid of binaries. Copying the values from our desired grid determined by a base probability that increases with each frame in an attempt to match the desired grid as best as possible. Effectively it's attempting to make decisions based on probabilities that increase as needed. This resulted in success for simple cases and failure for some more complicated images. The algorithm seems to struggle with more contrasting images with varying brightness. When you only have 1 way to process an image, which is based on brightness or darkness, there's no way to get back on track when unexpected values are put in the grid.

Given a few images to test the algorithm with, you will clearly see that as image complexity increases, quality of outcome decreases, as would be expected. We would need many more layers to be able to increase the likelihood of replicating the desired image. However this seems fitting given a project based on a simple game based on binary decisions. In order to increase the likelihood of success, we would need to stay away form the nature of conway's game of life.

Conclusion

Given this was a very interesting project and I learned alot from it as well as from the beginning of class until now. When we were first assigned and AI project at the beginning of the semester, I was really unsure what that would entail, as I'm sure the rest of the class was as well. It sounded scary and overwhelming. But as time went on and we learned a little bit more about the tools that already exist for doing things like image processing, language prediction, and probability distributions, I started to understand that we just needed to use the tools at hand to do something cool with these tools. I think we all learned a little bit about image processing and certainly a lot about the python tools that exist. I had no idea there were already so many simple to use AI tools that exist in numpy, matplotlib, and other libraries. I personally picked up skills in using these libraries to help generate results based on probability, and display them in a variety of ways, as well as just overall code optimization. When you work with AI tools that can have so many parameters and so many loops, its important to do things as efficiently as possible.

Overall the project was a good experience. It was a good balance between learning something new about a complicated topic, but not being too overwhelming. Given a wide umbrella to work under made it possible to come up with a project that was a good learning experience, but didn't keep anyone up at night. I know that in the future these skills will come in handy both with higher education as well as in a job implementing AI systems.

References

animeshsaha0906/conwayGameOfLifeAI - GItHub repo

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90-95.

Matplotlib Documentation:

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html

https://matplotlib.org/stable/tutorials/pyplot.html

https://matplotlib.org/stable/api/animation_api.html

Pillow (PIL) Documentation: https://pillow.readthedocs.io/en/stable/

Conway, J. H. (1970). The Game of Life. Scientific American, 223, 4, 4–5.

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

Tkinter documentation:https://docs.python.org/3/library/tkinter.html