SIMULATING THE CONWAY GAME OF LIFE USING 2D ARRAYS (GRIDS)

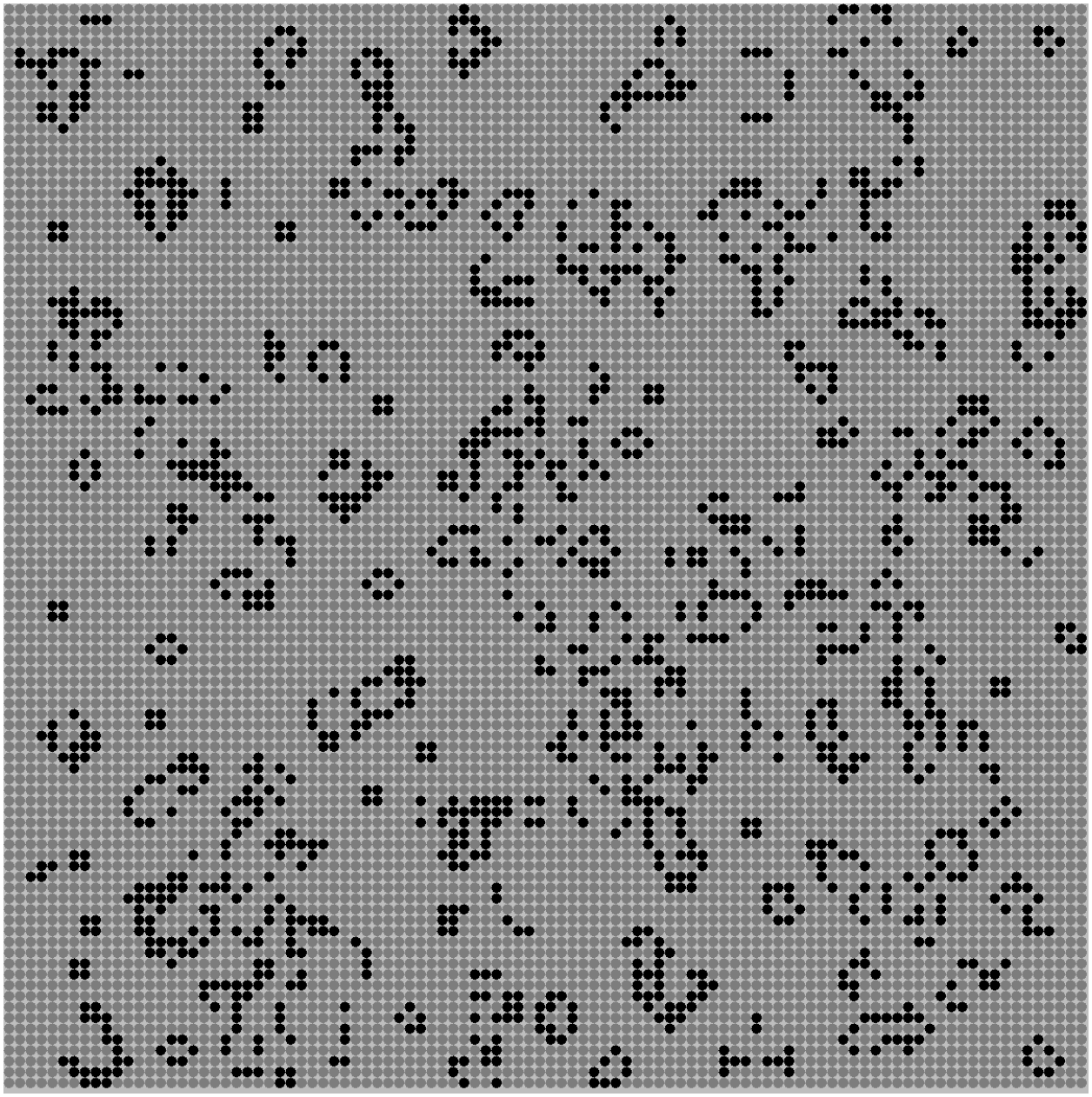
Abstract:

In this project, I explored the concepts of 2D Grids (Arrays) in Java to simulate the Conway Game of Life. In the game, the state of each cell in the grid is continuously updated based on its neighboring cells, with each cell either being "alive" or "dead." The grid's size and state evolve dynamically as the game progresses. The use of arrays in this project was crucial as they allowed me to model the grid, where each cell's state is represented and updated at every iteration. The size of the grid could be manipulated to fit the changing conditions of the game at any point in time. I simulated this dynamic evolution using arrays, where cells could transition between alive and dead states depending on their neighbors. One of the major takeaways from this project was understanding how simple rules of grids can lead to complex patterns of behavior, demonstrating the power of 2D arrays and iteration in simulating evolving systems, especially in games like sudoku and this specific example.

Results:

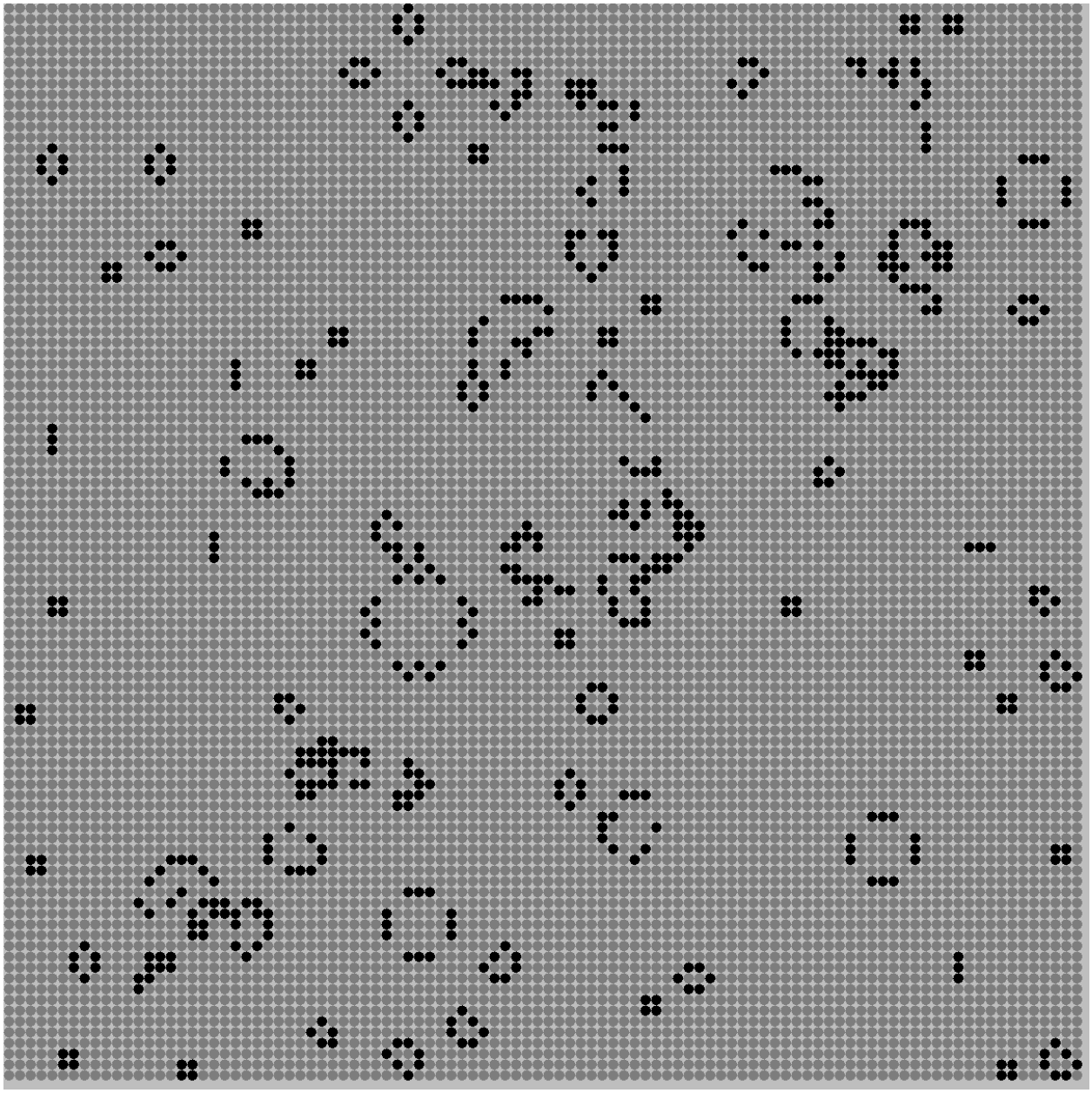
In my experiments, I created a new class called the LifeSimulation that enabled me to run a number of rounds for my game for specific sizes of the columns. My LifeSimulation code was modelled to update the states of the cell (according to the rules of the game) after 250ms. Each iteration was set to run for 250 ms each time. I also used a command line argument to control my grid size and the number of steps in the simulation. For my random initial conditions, I tested my LifeSimulation program for a grid size of 100 x 100 and an initial chance of living of 0.25.

I got my initial board state :



Initial Board State

Then, the program ran an iteration of the program after 250ms (which is quite fast for me to get a quite timely picture).



Board State after One update

I also got a video of the animation of the game for about a minute or two. Check it here : Simulation Video .mov

Exploration

Hypothesis : The number of living cells after 1000 rounds depends on the initial probability of a cell being alive. Moderate initial probabilities (around 0.2 to 0.6) lead to the most surviving cells, while very low (0.0-0.1) or very high (0.8-1.0) probabilities cause most cells to die out. Larger grids support more stable patterns, leading to higher survival rates compared to smaller grids.

I set up my experiment by varying the initial status chance from 0.1 to 1.0 (increment of 0.1 every step) under three different conditions of the grid (50 by 50, 100 by 100, 150 by 150). I wanted to show that even if the grid size varies, proportionately the same number of cells would live and the number of cells would actually increase. The independent variables are the grid size and the initial chance of a cell being alive. The dependent variable is the average number of cells living after 1000 rounds

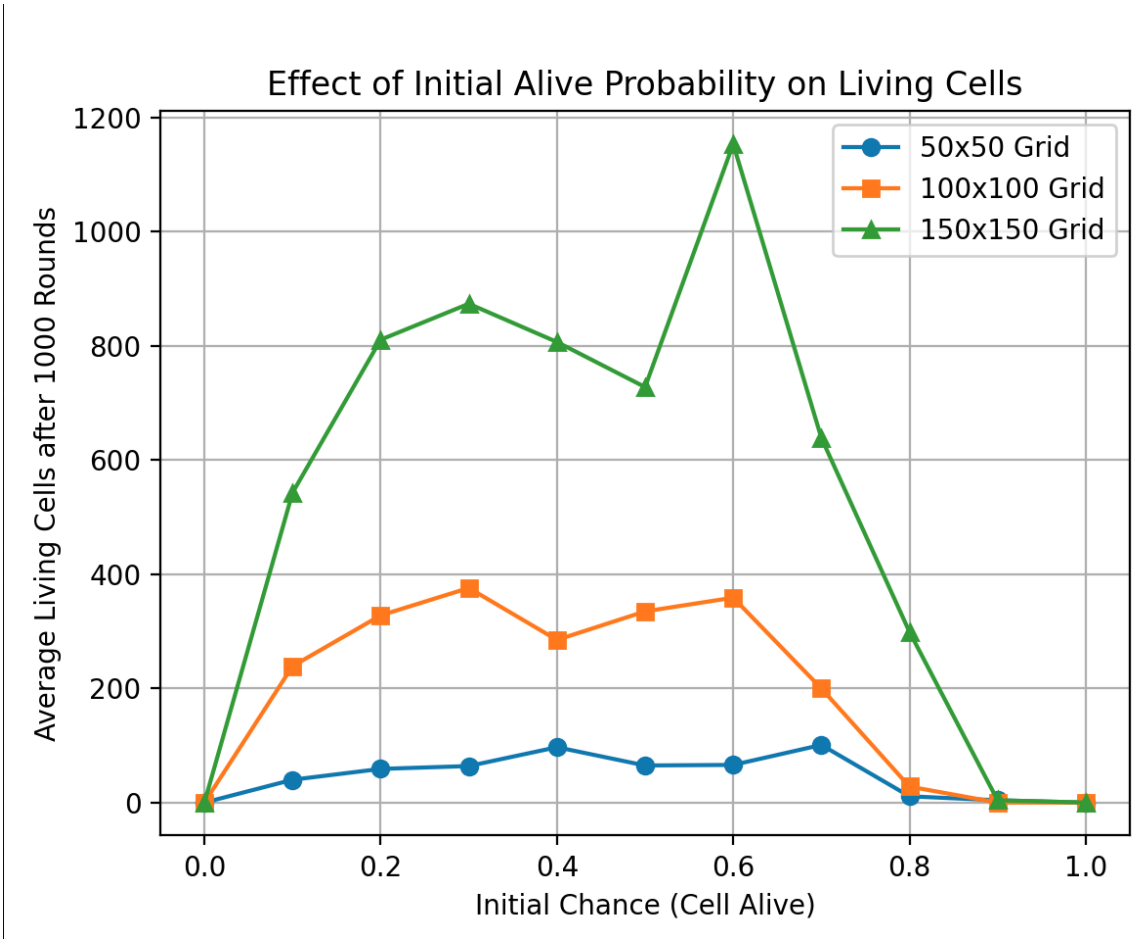
For this task, I used my command line parameters to receive arguments into my LifeSimulation main function. **The first command line argument is the number of rows, second, number of columns, the third, the initial chance that a cell is alive**. I then changed/ remodelled my code for the LifeSimulation to return the number of living cells based on the initial given chance and the grid dimensions provided on the command line. I also set the advance function to be in a loop so that it could run for 1000 rounds for each test case.

For example for the first test case: I ran **java -ea LifeSimulation 50 50 0**

Initial Chance (Cell Alive)	**Average Number of Living Cells after 1000 rounds (50 x 50 Grid)**	**Average Number of Living Cells after 1000 rounds (100 x 100 Grid)**	**Average Number of Living Cells after 1000 rounds (150 x 150 Grid)**
0	0	0	0
0.1	40	239	543
0.2	59	328	811
0.3	64	376	874
0.4	97	285	807
0.5	65	335	728
0.6	66	359	1154
0.7	101	200	639

0.8	11	28	299
0.9	4	0	4
1.0	0	0	000

The Average Number of Living Cells from The Simulation of The Conway Game of Life for 1000 Rounds for Various Grid Size and Initial Chance of Living



A Graph of Average Number of Living Cells after 1000 rounds against the Initial Chance

From the result of the experiments, extremely low initial probabilities (0.0-0.1) result in very few surviving cells because there aren't enough starting structures to sustain growth. Similarly, very high

initial probabilities (0.8-1.0) lead to overcrowding, which quickly stabilizes into extinction. The middle range (0.2-0.6) allows for the formation of stable patterns, leading to the highest number of surviving cells, especially in larger grids where more complex structures can persist.

Extensions:

For my extension, I introduced a way to analyze how different grid sizes and initial cell probabilities impact the long-term stability of living cells in the simulation. By modifying the program by allowing the user to control the LifeSimulaiton main method with command line parameters, I allowed for systematic experimentation with different grid dimensions and probability values to observe patterns in how the simulation evolves over 1000 iterations. Specifically, I adjusted the initialization process to support running multiple trials and collecting statistical data on the number of living cells after a long period. This was done by modifying the LifeSimulation class to store and analyze results across different runs, making it easier to track trends in the game's evolution.

This extension enhances the program by providing insights into how initial conditions shape the final state of the simulation. Instead of simply running a visual simulation, users can now dynamically test different configurations and analyze the outcomes through collected data.

To run this extension, users can execute the program with different grid sizes and initial probabilities using the command:

java LifeSimulation <rows> <columns> <chance> <Number of iterations>

For my first instance, I got a video of running the simulation by putting all the following parameters

java LifeSimulation 100 100 0.3 1000

Check the video here: Extension Simulation .mov

See the ExtensionREADME.txt file in the folder extension for more details

Acknowledgements:

I consulted a lot of Youtube videos on the use of ArrayList, 2D Arrays, Java Graphics , implementation and related data structures. I also went to the professor's office hours and the TA hours (Ben) for a better explanation on 2D lists, java problem solving logic, and related java syntax. I

also studied the code from the class notes to get a better understanding of the related concepts. I also discussed code concepts with Emmanuel Buabeng during my project testing and some implementations.