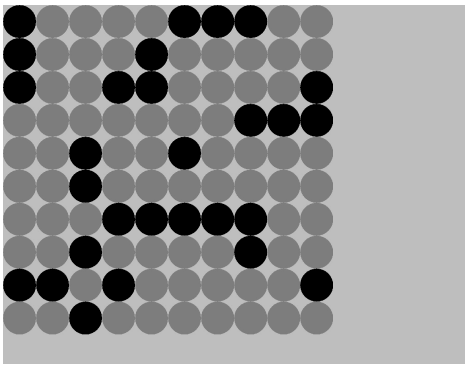
CS231 Project 2: Conway's Game of Life

September 18, 2023

Abstract

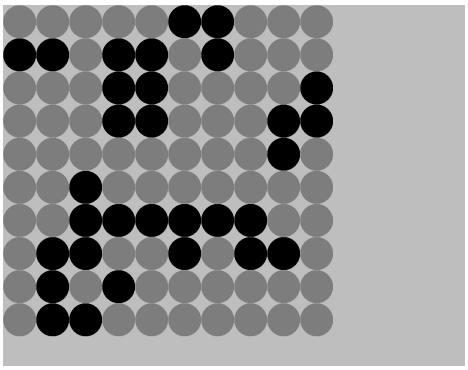
This project is meant to teach CS Students how to build and manipulate 2 Dimensional Arrays (known as 2DArrays) in Java. This is done through simulating Conway's Game of Life, a simulation of entities on a 2D grid. The entities will interact with each other to determine how they change over time.

Results



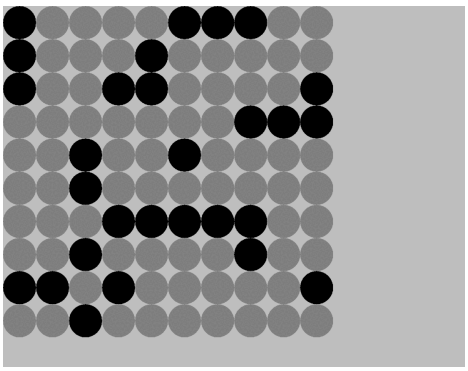
Initial Board State

This is the initial state of a 10x10 grid of cells that each have had a 25% chance of starting alive. For every update that follows, each of the cells will evaluate what kind of state it would be based on its neighbors from its current update.



Board State After 1st Update

A lot more cells become alive due to reproduction. However, this will most likely lead to the overall population's downfall, as the overpopulation of these cells will cause mutual death.



Board State After ~19 Updates

Total number of alive cells gradually decreases.

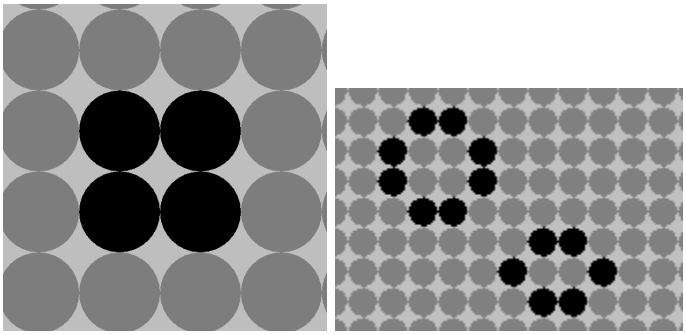
Discussion

Overall, looking at the results of Conway's Game of life, they do make sense and explain how the game plays. As shown from the images/video that I got in the LifeSimulation.java, the user has an input for how many alive cells the landscape will initially start with, but then after the input, the game is all based on randomness. The landscape will drastically differ depending on if there are too many cells alive, causing overpopulation, or if there are dead cells near alive ones, resulting in reproduction.

Exploration

How do the parameters of grid dimensions and initial status chance affect the number of living cells after a sufficiently long time (say, 1000 rounds)?

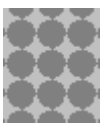
Hypothesis: Having more grid dimensions means more space for each cell cluster to branch off, making it more likely to create an infinite system (that lasts forever) such as these:



The best initial status chance has a sweet spot, like a parabola there's a higher amount of living cells the closer the chance is to 50%, and lower if the amount is closer to either 0% or 100%.

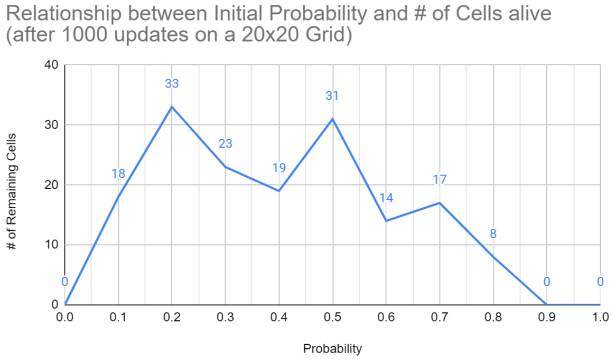
Explanations:

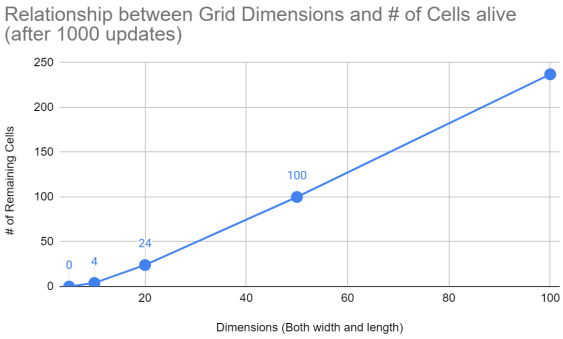
Generally, having bigger grid dimensions means more empty cells, which can increase the potential number of living cells at a time. When evaluating neighbors, cells consider every space around them. These include the ones horizontally, vertically, and diagonally adjacent to them, for a max of 8.

 For a cell to stay alive, they need to have 2 or 3 alive neighbors, and with 8 potential spots, it's very likely that a cluster of cells could approach it from any direction and revive/keep it alive.

However, for some spaces, such as the ones on a wall, there  are no further cells at a specific direction, for a max of 5 neighbors. This lowers the chances of having the opportunity of being revived.

This is even worse for corner cells,  , with them having only 3 to determine whether they are alive or not.



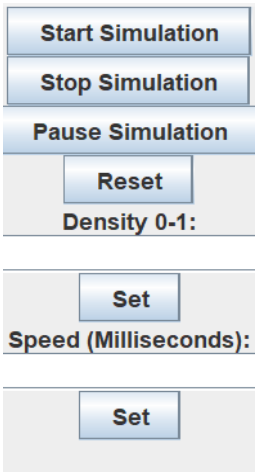


This hypothesis is backed up by further testing, to which the results are displayed on the graph(s).

Extensions

GUI()

Having to constantly run the code just to see another simulation run was tedious, the buttons implemented are meant to run as many simulations as possible without having to rerun the code.



Start Simulation: Resumes the simulation if paused (The simulation starts on default).

Stop Simulation: Closes the program.

Pause Simulation: Stops the simulation on whatever update it is on.

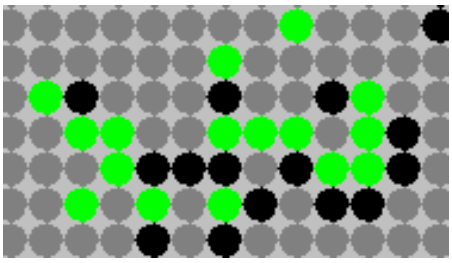
Reset: Triggers the reset() method in landscape, recreates the landscape based on the parameters given.

Density (0-1): This changes the probability of an initial cell starting as alive, from 0 to 1. Pressing Set applies this, and resets the simulation automatically.

Speed (Milliseconds): This changes the time that the system stalls for in between each update. This helps observe the simulation at whatever speed is best. Pressing Set applies this.

Infection

Nothing stays dead forever. When a Cell dies, there is a 1% chance that it may reanimate as a zombie, and it has its own unique set of rules.



When a Cell **dies** (alive to dead) it has a **1%** chance of turning into a zombie.

As long as there's at least one **alive** cell, it will continue to stay alive, otherwise it will **die** (gray cell)

An **alive** cell that comes in contact with a **zombie** will become a zombie itself.

This effectively gets rid of patterns that may cause the simulation to loop to be eliminated as the repeated shuffle between dead and alive will eventually spawn a zombie, and a field of all zombies won't sustain itself.

References/Acknowledgements

I asked for input from Prof. Lage about what to include in this report, and a TA (I didn't get his name) for troubleshooting on printing the wrong simulation, and/or neighbor interaction. The code sent in the original project profile was wrong, according to Prof. Bender it didn't really matter.
