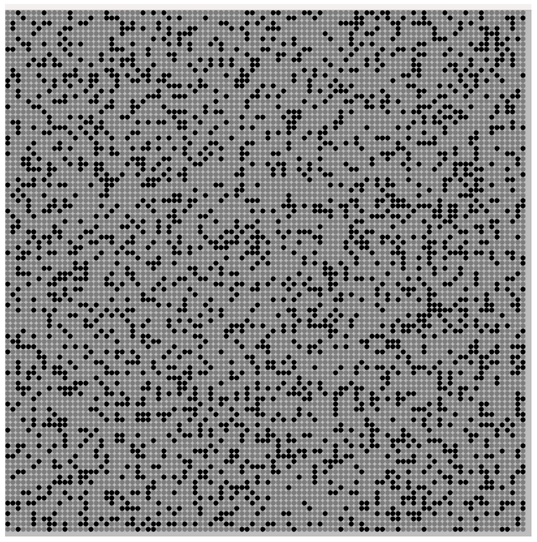
Simulating Conway’s Game of Life using Java

Abstract:

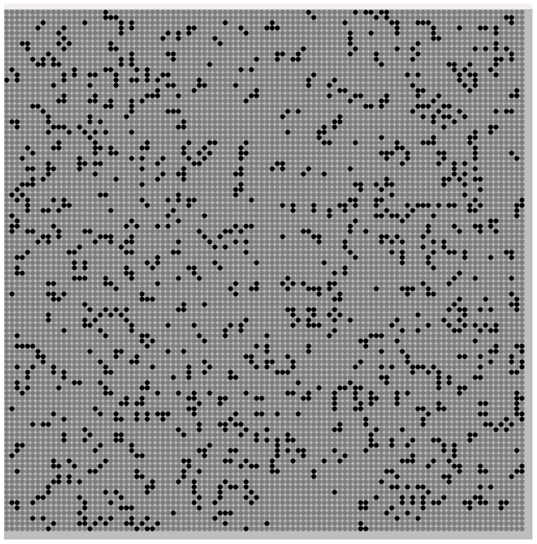
The real world problem I explored in this project is Conway’s Game of Life, which is a cellular automaton that runs on a few mathematical rules. To better understand the problem in this project, I created 3 different classes called Cell, Landscape, and LifeSimulation and built the simulation from there. The core CS concepts that I used in this project were loops, lists, and space and memory usage. My key findings from this project were figuring out how to save memory space and still make the simulation run efficiently.

Results:

The experiments I ran in this code are baseline simulation, which generated random initial conditions in order to randomize the simulation each time I ran it. I also tried to identify infinite loops every time it ran in case the simulation would be unable to stop running. I did this by checking the grids against their past grids to make sure that nothing was repeating again. The metrics I am reporting for the outcomes of this experiment are the living cell counts per simulation and the chance probability.



Picture 1: This is a picture of the board’s initial state when the simulation is started.



Picture 2: This is a picture of the board’s state following one update past its initial state.

Video: https://youtube.com/shorts/_eMOvY5pTT4?feature=share

Above is a video of my simulation.

The outcome of my experiments successfully ran and simulated Conway’s Game of Life. They showed the prevented infinite loops and optimized memory space.

Extensions:

I didn’t do an extension for this project.

Reflection:

1. I checked the grid after every step in the simulation. I also tested the simulation multiple times with different initial conditions. With respect to storage, the grids have to be stored every time which makes it not as efficient as it could be. The runtime of the methods is O(1) on average.
2. a) It would make the most sense to implement a Stack instead of a Queue because if you want to use a backwards function in a Queue, it would be much more difficult because it uses first in, first out, so it would be much less efficient than using a Stack, which uses last in, first out. Backward navigation would be much easier with a Stack because of the efficiency of popping the last item.

   b) Storing the grids again would take up a lot of storage and would need to create a brand new grid every single time you went backward and forwards again. Although it may be easier and less time consuming to store the grids, it would be better to recalculate the grids again even with the little bit of extra time it takes.

Acknowledgements:

I worked with Kamalani on this project and I also went to TA hours on Wednesday and Sunday.