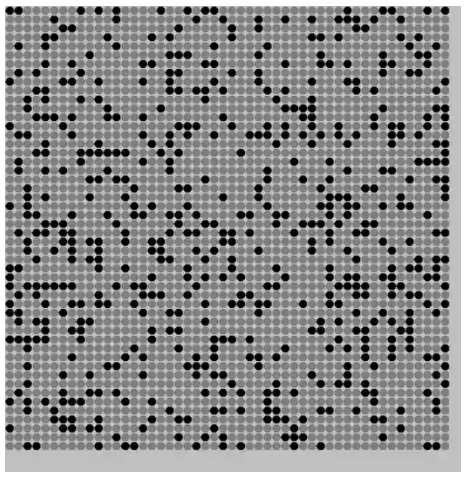
Title: Game Of Life
Author: Eli Giglietti
Section A

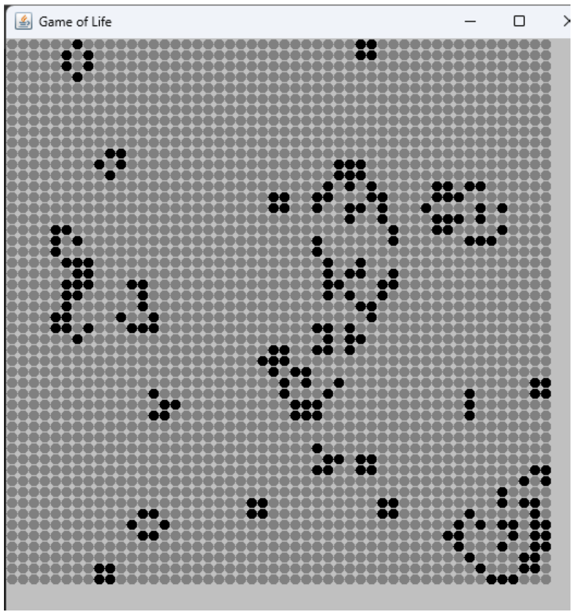
Abstract:

In this project we simulated what would happen if some cells were alive and some were dead and they changed states depending on how many alive cells were around them. The main data structure we used were 2d arrays. We used them to store a grid of cells that contained information on their state. The first class was the cell class where we made a cell. The cell had a 1 or 0 if it was alive or dead, and can change states. The next class was the landscape class where we had a grid of cell objects. The landscape class can change the state of the cells based on the cells' neighbors. There was also a pre-made landscape display class that displayed the landscape using the Graphics java import. Lastly, we had a simulation class that ran through many iterations of the landscape and updated the cell states each iteration.

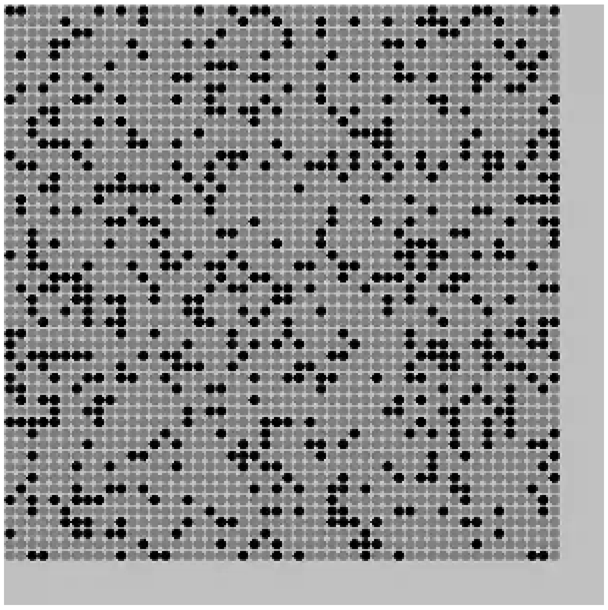
Results:



This is the initial landscape for my simulation. The rows and columns are both 50 long and the chance was 25%. The darker circles are the alive ones and the gray circles are dead. Over the loop, they constantly change.



This is the final result of the landscape after the 100 iteration long simulation. It advanced, and then displayed the new landscape 100 times
.



This is the gif I made for the game. At first I tried to run 500 iterations, but the gif file was too large to put into a google doc, so I had to cut it down to 100 iterations.

Exploration:
*I did this part in my LifeSimulationExtension.java file as it is easier to change the values.

My hypothesis is that the lower the initial status chance is, the fewer alive cells there will be at the end. However, I don't think the grid size will affect the number of cells alive at the end.

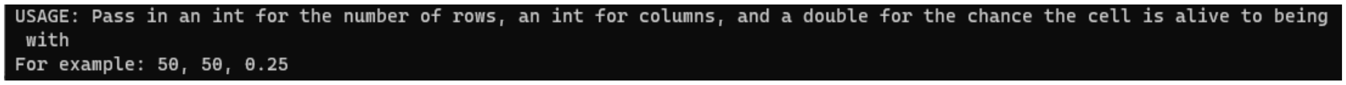
Grid Size	Initial Chance	Average number of alive cells at end	Percent Alive
20x20	0.25	11	2.75%
20x20	0.5	17	4.25%
20x20	0.75	7.2	1.8%
50x50	0.25	93.6	3.744%
50x50	0.5	86.6	3.464%
50x50	0.25	70.8	2.832%
100x100	0.25	351.8	3.518%
100x100	0.5	369.6	3.696%
100x100	0.75	185.4	1.854%

For this part I got 5 values for the number of cells alive at the end after 1000 iterations, and then found the average. The results seem to show that the higher percent of alive cells you start with, the fewer alive cells there are at the end. In each case, the 75% alive rate ended with the fewest alive cells. However, for the 50x50 and 100x100, when the initial rate was 50%, the most cells were left alive at the end. I am assuming that there is some sweet spot between 75% and 25% where the most cells are left alive consistently. Further, the 20x20 has the most variation, but I think I would need more data points to back that up.

Overall, my hypothesis seems to be wrong and the number of alive cells at the end seems to be a function of the updateState method where the cell stays alive if it has two or three alive neighbors and becomes alive if it has three alive neighbors.

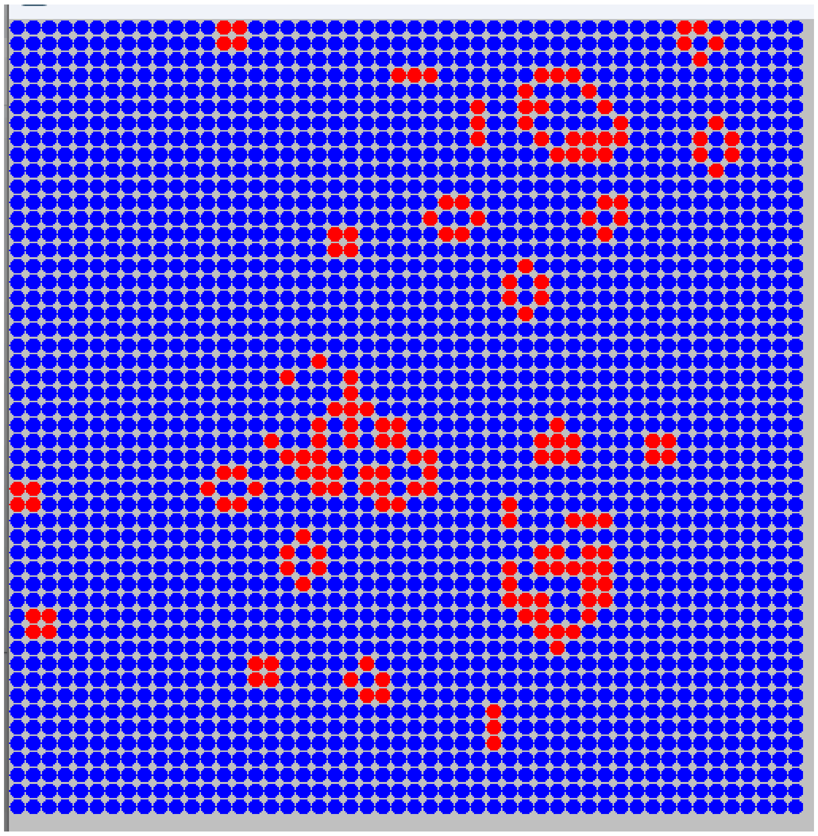
Extensions:
*This is in LifeSimulationExtention

For my first extension I did command line arguments. As implementing the command line arguments wasn't that hard, making sure they were correct from the user was. I needed to implement two different usage conditions. Firstly, if the user doesn't put in 3 arguments, then you get a usage statement. Secondly, these arguments need to be the correct type. At first, I tried to change the main parameters to take integers instead of strings, but that didn't work. So, I had to change the strings into integers and doubles, using parseInt and parsDouble. Then I had to check if they were actually integers and doubles, and if not, print a usage statement and return nothing.



```
USAGE: Pass in an int for the number of rows, an int for columns, and a double for the chance the cell is alive to being
 with
For example: 50, 50, 0.25
```

*This is in landscapeDisplay
For my second extension I changed the color of the cells. I did this by making a new method in landscape display that changed the alive cells to red and dead cells blue. I then implemented this method into the paintComponent method and commented out the scape.draw.



Acknowledgements:

Ryan Mogauro
Cynthia
Liam Cotter