Lab Report Week 3: Conway's Game of Life!

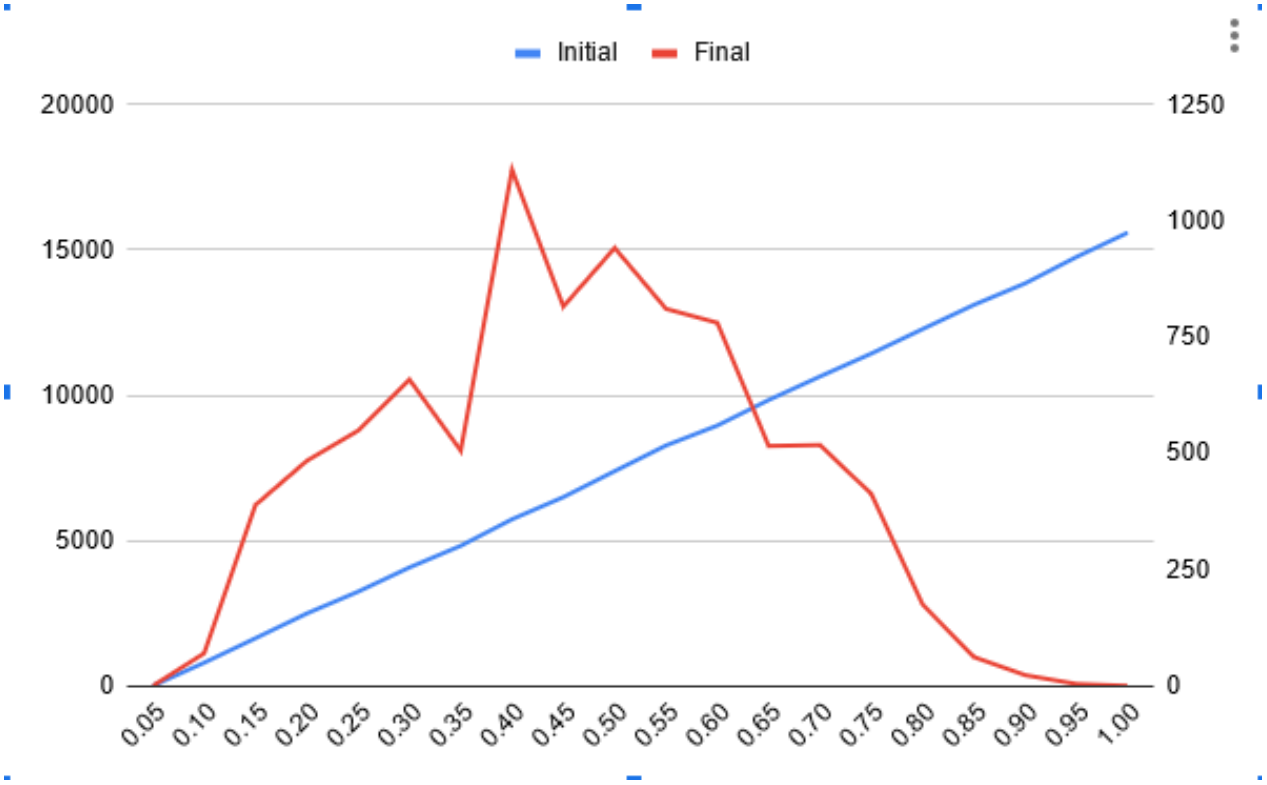
Abstract

The objective of this project is to implement Conway's Game of Life in Java. Following given templates I modified and extended upon needs. The project explores using the Grid data structure and extend upon OOP in Java. I ended up using the `Queue` interface in Java as a `LinkedList` type to implement my Project idea, and I also modified heavily the drawing code in `LandscapeDisplay.java` to my needs, of which I understood a lot more.

Results

Exploration results

Following the instructions I ran the game on a 127x127 grid, `chance` increasing on [0;1] with interval 0.05. I collected the data `java LifeSimulation > res.txt` and ploted the data from res.txt, resulting in the figure below:



It seems for `chance` around [0.4;0.65] the grid has the most life after 1000 advances. Which make sense in the original theory of the game of life, with the rule being set to account for underpopulation, overpopulation and reproduction.

Extension 1: Historical Data!

You might have noticed the lingering marks of the old population on the landscape as red marks (graphic!) that slowly fades into the darkness. This is a really cool visual effect I found from here and wanted to replicate. To replicate this effect, I needed to: 1. Create a `Queue` that stores a certain number of historical landscapes (set to 20 arbitrarily in `Landscape` class constructor), and 2. draw the historical landscapes in fading red to the window.

The first task is relative simple, I created a new `Queue<Cell[][]>` called historicalLandscapes in the Landscape class, `poll`(remove) the oldest/farthest landscape from the queue `offer` (insert) the current (second newest) landscape to the queue before advancing the grid. I then in my `draw()` method loop through the queue and draw the landscapes with increasingly red (from black) color (the order of the queue is old -> new landscapes). I soon realized to make the smooth fading effect, the drawing of historical landscapes must happen at a faster FPS than the advancing of the current Landscape. To fix this, I added a counter called `whenToAdvance` in the Landscape display class, restricting the advancing rate to half the rate of screen update.

Extension 2: I am God and I spread life.

You might have also noticed the empty board if you run `java LandscapeDisplay`. To create life, simply click or drag the mouse and where the mouse touched new Alive Cells will born. This is also inspired from the same site as before.

To implement this, I needed 2 things: 1. A way to catch when the mouse clicks/drags and 2. A way to translate the mouse computer on the screen to the cell it touched on the grid.

For 1. I simply relied on the Java documentation. I added MouseListener that catches for when the mouse interacts and pass the interaction's data to a new method in `Landscape` class that is `setCellAlive`. For 2. The data passed to the `setCellAlive` method are the ratios between the mouse's position and the dimension of the window, resulting in a `double` from 0 to 1 that will be multiplied with the number of rows and columns of the grid to give back the cell I touched. I simply then set the cell's state to alive via `Cell:setAlive(boolean)`;

Reflection and Acknowledgements

I discussed with no one during my project but consulted a lot of websites in my looking for ideas. Although I pursued an extension idea of GPU accelearting by parallel computing the

neighbors sum and landscape update I decided not to do it for this project realzing how complicated (or high learning curve) this idea is.

All my extension ideas came from here, I did not sample their code but tried to replicate it completely by myself.

References:

https://docs.oracle.com/javase/8/docs/api/java/awt/Component.html#addMouseMotionListener-java.awt.event.MouseMotionListener-
https://docs.oracle.com/javase/8/docs/api/java/awt/event/MouseEvent.html
https://www.geeksforgeeks.org/queue-interface-java/
https://docs.oracle.com/javase/8/docs/api/java/awt/Container.html
https://docs.oracle.com/javase/8/docs/api/javax/swing/JPanel.html
https://docs.oracle.com/javase/7/docs/api/javax/swing/JComponent.html
https://docs.oracle.com/javase/8/docs/api/java/awt/Window.html#pack--
https://docs.oracle.com/javase/7/docs/api/java/awt/Color.html
https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html