Game of Life 220019540 & 220004294 & 220023971
Tutors: Alan Miller & David Harris-Birtill

Overview

Our task was to create a replica of Conway's Game Of Life, a game which 'plays itself'. To achieve this, we used Java Swing to create a user interface which will display the game and include intuitive features to save the current status of the game and alter the game conditions.

We based our game on core rules which determine how the life of a cell will play out, including the requirements for it to live, become alive, and die.

Our git repository is located at /cs/group/cs1006-p2-group22/gameoflife.

Design

Game Rules

One of the most well-known features of Conway's *Game Of Life* are the rules that are associated with them. The game logic behind this program is what causes the grid to produce interesting patterns with the cells.

For the game logic, we have a class named GameOfLife that, as an object, represents the game. We first started by using a HashMap of integers, which represent indices on the board. We also have a game settings record (For the neighbour variables, board size and toroidal toggle), a step counter, and user action history. By default, you can create the class as an object by providing it the Settings record. It then initialises the rest of the variables to a default state.

Another way to create a game is by deserializing it from a file. It takes either a text file or a binary file with signature "g22". The text file follows the standard laid out in the specification, and the binary file writes to an output stream that first writes the file signature, serialises the objects and then compresses them, and does the same in reverse when reading the file.

When advancing (what we called stepping in the code) the game, we first get the coordinates of alive cells and then add the neighbouring cells around it. This list of coordinates are all thrown into a stream and then deduplicated. Moreover, the neighbouring cell finder is able to filter out cells based on if the board is toroidal. When reading the stream, we get another neighbour count for each cell and then use this and the board's settings to set the new cell state for the index of the cell, and after every coordinate is processed, set the new state to the current state, and advance the step counter.

A novel feature of our implementation is that we also have a way of reversing the game. When there is a step backwards, the board class runs every single step up to that point, and then caches a number (currently 100 in the code at the time of writing) of board states up to that point. It also reads the user's action history to see if any cells were toggled from step 0 to the current step.

User Interface

One of the core parts of this project was to create a user-friendly interface, which allows easy-to-access controls that excel in comparison to working with a terminal and command line. Java Swing was the feature that we utilised most, creating windows that allow us to make our own interfaces.

To create a sensible user interface we first needed to outline the options required within the program that the user will be able to choose from:

- ➔ Play / Pause
- ➔ Step
- ➔ Load / Save
- ➔ Game Options

From this list, we decided that the best way to organise these options would be to create a drop-down menu of the load and save options, while keeping the play / pause and step options visible always for the user to be able to interact with them often. We also decided on exposing a speed control slider in the main interface below the play / pause button to provide easy access to modifying the simulation speed, with increment and decrement buttons beside it. The current delay in milliseconds can be seen by hovering over the slider in the tooltip text.

Our design also features additional information in a status bar along the bottom of the frame, including a play or stop symbol showing if the simulation is running and a current step counter in the bottom-left, and information about the selected game settings in the bottom-right. The settings information contains the number of cell neighbours to stay alive (SA), neighbours to become alive (BA), the grid size, and whether the grid is toroidal or non-toroidal.

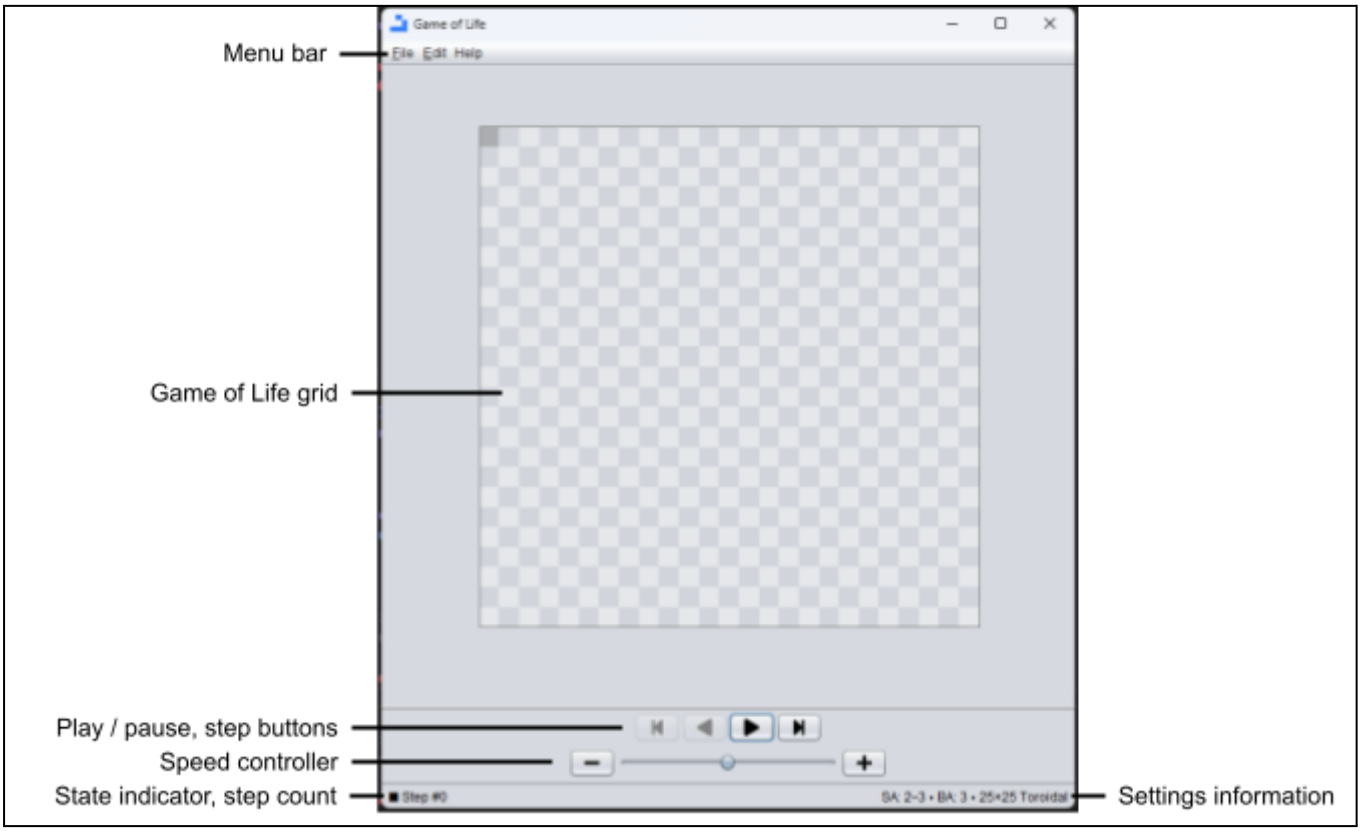

Figure: Final main interface for our project

New Game Options

Within the menu bar, games can be opened, saved, or created. Creating a game using “New” in the File menu will open a settings menu which allows a user to configure the Game of Life to their choosing by showing options for both the board and the behaviour of cells. The data is validated before a new game is created, and if validation fails the invalid input will be outlined in a red border and tooltip text will be added explaining the problem to the user.

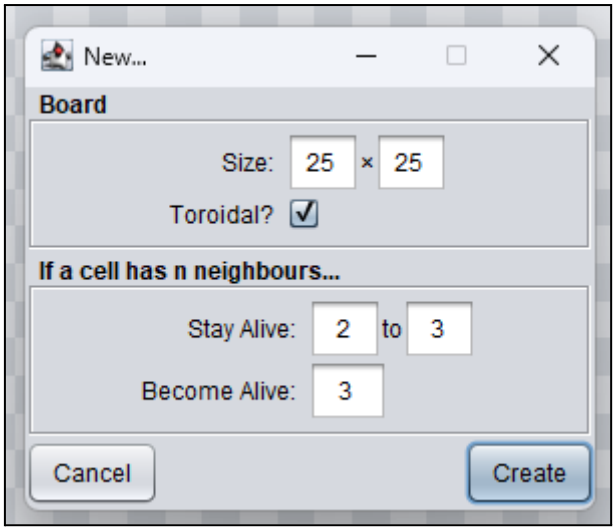

Figure: New game options interface

Icons

Where possible, we decided to use icons instead of text to label buttons within our user interface. These allow for the interface to be easily understood, even if English is not a user’s primary language, while also allowing for the state of certain buttons (e.g. the play / pause button) to be easily inferred. Buttons are also labelled with tooltip text that appears on hover for further elaboration. All icons used within our project, aside from the glider icon used in the title bar, come from the free *Font Awesome (https://fontawesome.com/)* icon set. An example of a few of the icons we used can be seen in the figure below.

Figure: Example button icons

The Grid

For our implementation of the Game of Life grid itself, we opted for a zoomable and pannable grid that is controlled using the mouse. Cells can be toggled between alive and dead with a simple left click (which pauses the simulation if it is currently running), while the grid can be panned by holding middle or right click and dragging. The grid can be zoomed by scrolling the mouse wheel.

A unique feature of our grid is that we are also able to represent the age of a cell through its colour - cells gradually transition from blue to yellow depending on the number of steps that it has survived for. Once a cell has survived for at least 50 steps, it will stay yellow.

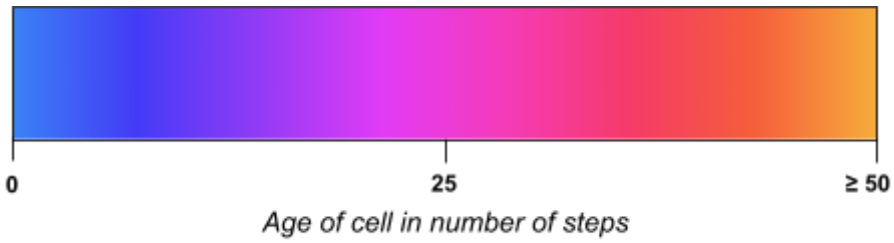

Figure: Gradient showing colour of cell based on its age

Testing

To ensure that our program was running smoothly, we implemented multiple tests to allow us to make sure everything is working accordingly.

Game Option Validation

The inputs in the new game creation window are validated before a new game is allowed to be created to ensure that they are reasonable. Failing validation will outline the invalid input with a red border and display a reason for the validation's failure in tooltip text.

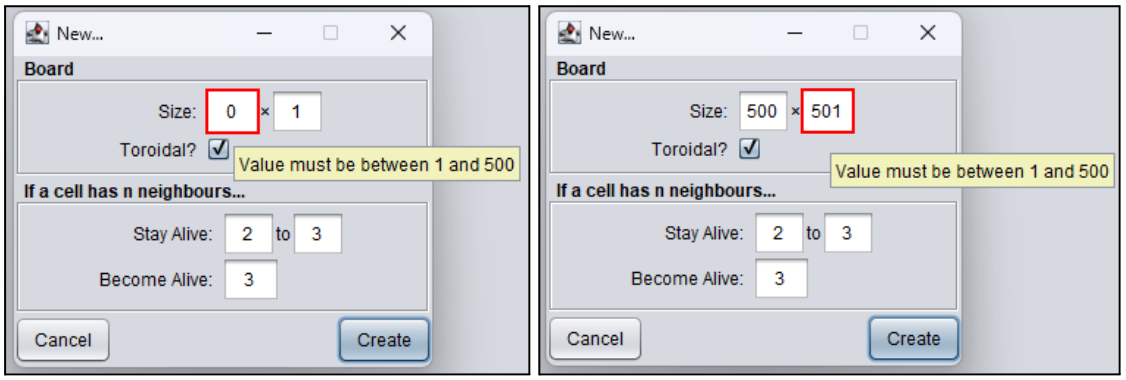

Figure: Board size inputs successfully rejected when either value not between 1 and 500

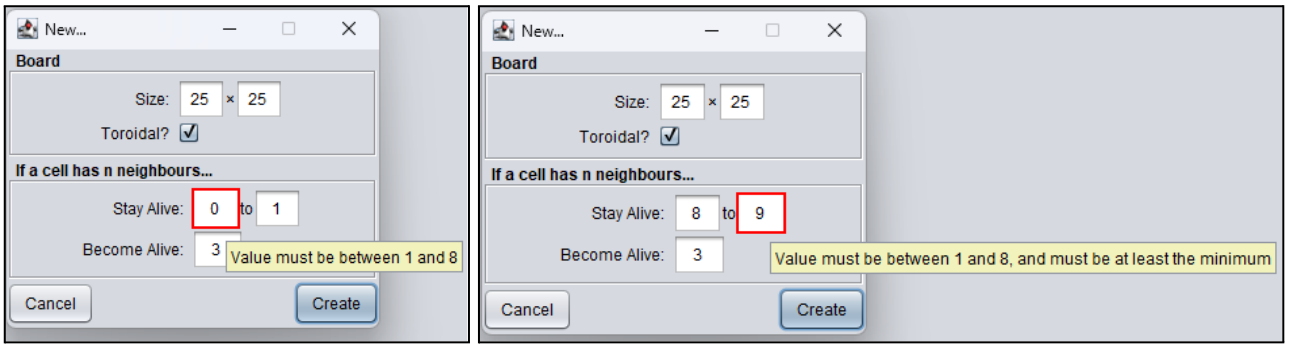

Figure: Stay alive inputs successfully rejected when either value not between 1 and 8

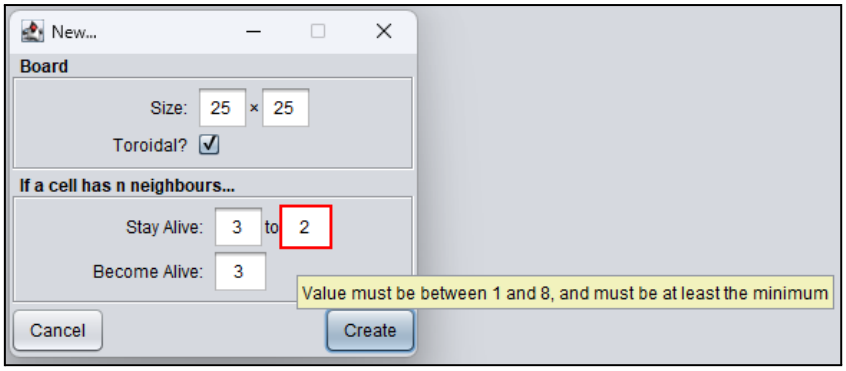

Figure: Upper bound of stay alive input successfully rejected when less than the lower bound

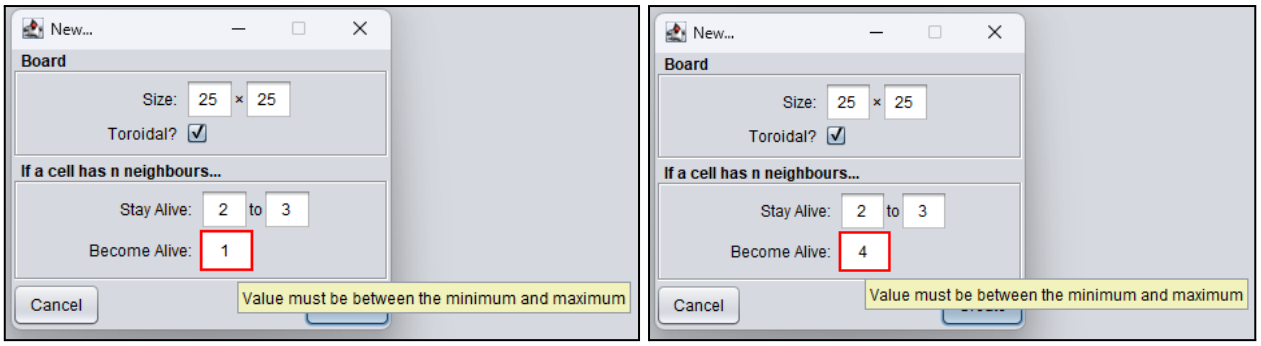

Figure: Become alive input successfully rejected when not between bounds of stay alive inputs

Game Board Testing

We employed further testing of the game board itself to ensure its functionality. This included writing a series of tests in TestBoard.java that verified logic associated with the Game of Life was correctly implemented. In the following test table, “default settings” is defined as the standard Game of Life rules (SA: 2-3, BA: 3) on a 25x25 toroidal board. All of the test assertions passed successfully.

Test name	**Description**	**Pre-conditions**	**Expectation**
testOscillator()	Test if the 3 tall oscillator oscillates	Default settings, alive cells at (3,3), (3,4), (3,5)	Step 0: Cells correctly placed at (3,3), (3,4), (3,5) Step 1: Alive cells at (2,4), (3,4), (4,4) Step 2: Cells back in same position as step 0
testSingleCell()	See if a single cell disappears and leaves the board empty	Default settings, alive cell at (10,10)	Step 0: Cell correctly placed at (10,10) Step 1: Grid is empty
testStepBack()	Test cell respawning when stepping	Default settings, alive cell at (10,10)	Step 0: Cell correctly placed at (10,10) Step 1: Grid is empty Step 0: Cell at (10,10)

	backwards		Step 1: Grid is empty Step 2: Grid is empty Step 1: Grid is empty Step 0: Cell at (10,10)
testNonToroidal()	3 tall oscillator disappears on a non-toroidal board edge.	SA: 2-3, BA: 3, 5x5 non-toroidal board, alive cells at (4,0), (4,1), (4,2)	Step 0: Cells correctly placed at (4,0), (4,1), (4,2) Step 2: Grid is empty

Conclusion

We believe that our program was able to successfully fit the requirements of the specification, providing a Game Of Life simulation with additional functionality added within the game, all implemented into an intuitive game interface, allowing the user to interact with the game accordingly.

Given additional time, we would have liked to have implemented a mechanism for opening example files directly from the “File” menu within the program itself - and also displaying its description HTML file in a popup window. We were, however, unable to do this as we unfortunately had some issues with loading the files using relative paths, so this code remains commented out in our submission.

Extra Features

To go further into this project, we implemented a feature which includes assigning a colour to a cell based on how many steps it has survived. This aspect is unique to the program and an intuitive way to gain better feedback and understanding of how cells interact and survive within the game.

Further expanding on this, we also implemented scalability within our game, found when creating a new window in the game. Giving more functionality and customisation to the user is a fun way of making our program more interactive and entertaining, while still maintaining the rules of the original Game of Life.

Moreover, we have implemented some extra functions in the menu, different save files, and many things to help increase usability and how nice the program looks.

Evaluation

Recreating Conway’s *Game of Life* was a very interesting look into the concepts of birth, survival and death of each cell. By implementing these rules, we could see that many things arise, such as stable configurations, oscillating patterns, and glider-like movements, all detailed within our src folder. The Game of Life is often used as an example of behaviour in complex systems, and has been studied even outside the world of computer science, also including mathematics and biology.

There were a few problems that we encountered in creating this program, such as the board files saving in the wrong location, and the board not appearing on screen when we zoomed or panned, but we managed to resolve all of these issues that we encountered through use of debugging.

All in all, this program was a very interesting project to work on, in which we learned loads about Java GUI, game rule implementation, customisation, and the *Game of Life*.

Word count: **1880**