## I: Basic info

### Student name:

Brianna D'Urso

### Date of project creation:

9-13-2024

### Last update:

9-16-2024

### Statement:

I (Brianna D'Urso) confirm that this project is my own work.

## II: Project Description

* This project implements Conway's Game of Life in both a console playable version as well as a GUI playable version. The console version first asks the user for the size of the board they would like to create. By default, each of the cells will be set to the dead 0 state. The user will then be asked if they would like to play the GUI version of the game; if they input y, the GUI version of the game will start. If they enter n, the console version of the game will start. If anything else is input, the user will be asked to input y or n.

### GUI version:

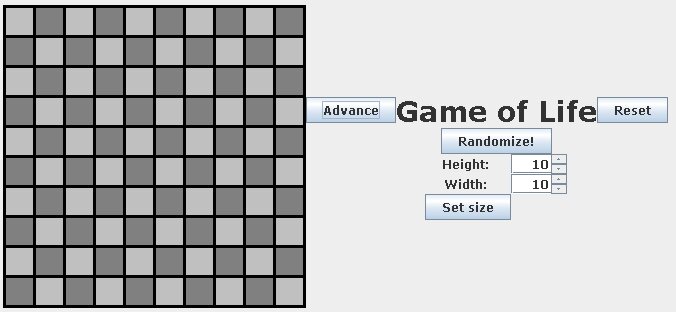
* For the GUI version of the game, the user may at any point click on any of the cells in order to "flip" its state, swapping a 1 for a 0 and vice versa. The user may also at any point click the Advance button in order to advance the state of the game to the next generation. The user may click the Reset button to reset the state of the board to be all 0s. The user may exit the game at any time by exiting out of the window of the GUI. In this version, the user is also able to randomize the cell states by clicking the Random button. Below the Random button is a set of two areas for the user to enter or select a number between 1 and 100, and below those a button that says Set Size. These inputs allow the user to set the size of the board during runtime.

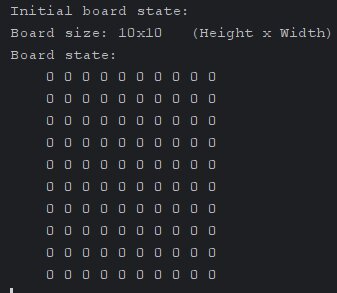
### Console version:

* For the console version of the game, the user will first be asked if they would like to randomize the board. If the user inputs y, the board will be randomized and the program will move on to gameplay. If the user inputs n, the user will be asked if they would like to set the state of one of the cells on the board. If the user inputs y, then they are prompted to input the coordinates and state (1 being alive and 0 being dead) in the format <row>,<column>,<state> (e.g. 3,2,1 to set the cell at the 3rd row 2nd column to the alive state) which they would like to set the cell at the coordinates to. They may repeat this any number of times until they input n when prompted if they would like to set the state of one of the cells on the board. Upon doing so, the game board will be printed out into the console, where 0s represent dead cells and 1s represent live cells. The user may then input . in order to advance to the next generation. The user may exit the console version of the game by entering anything other than . into the console.

## IV: Test cases

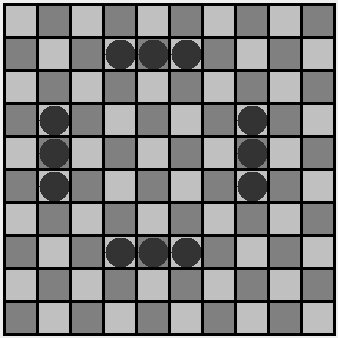
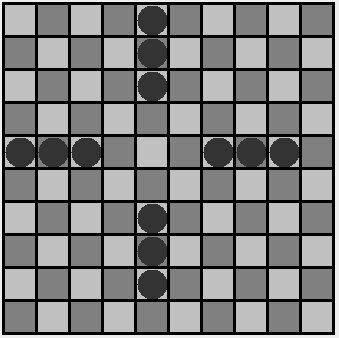
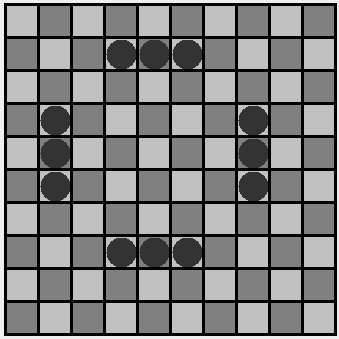
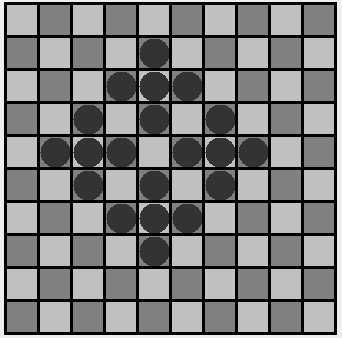
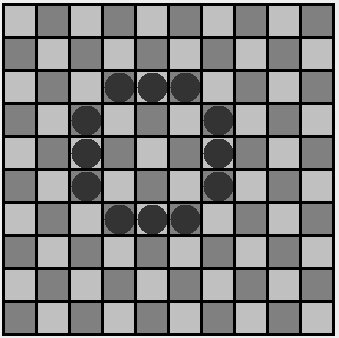
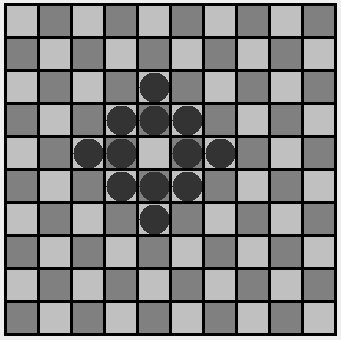
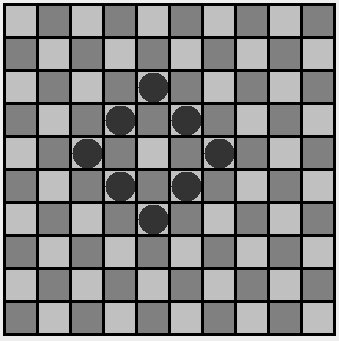
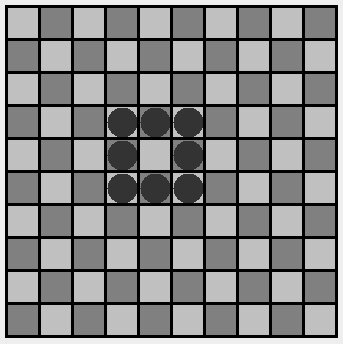
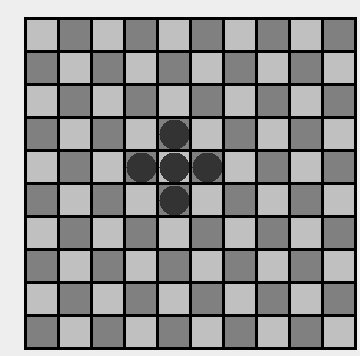
### 1. Initialization:





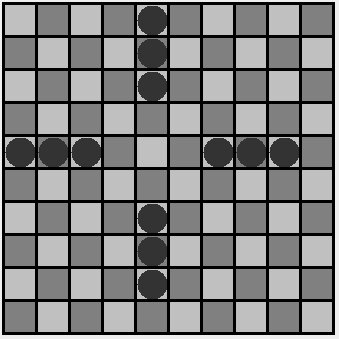
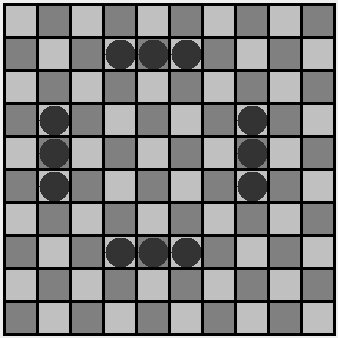
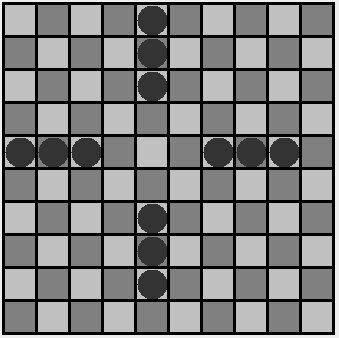
### 2. Next Generation:

The following pictures under this (Next Generation) section depict an in-series sequence of generations as determined by the program and displayed in the GUI. As can be seen by these pictures, the program is functioning as intended, updating each of the cells according to the rules of the game of life between each successive generation. ![[./Pictures/next\_gen\_1.jpg]]



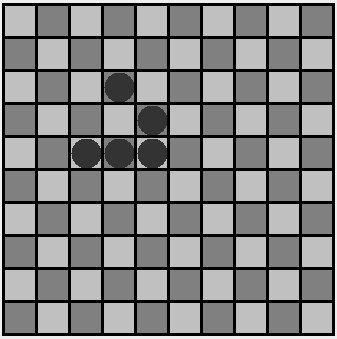
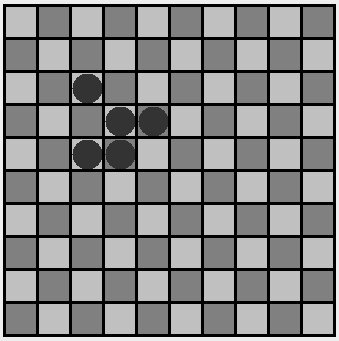
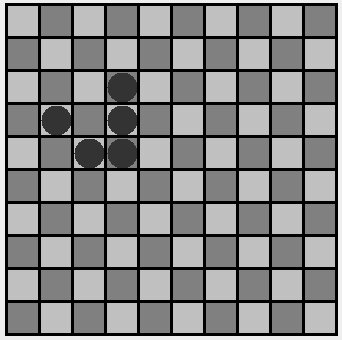
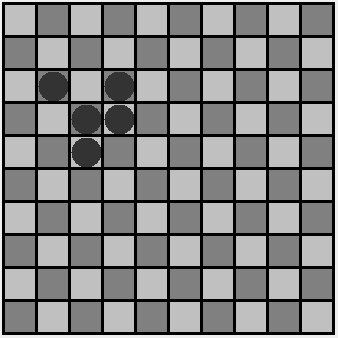
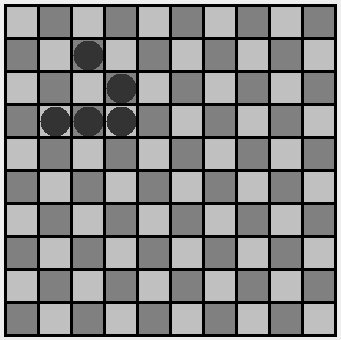
### 3. Blinker Pattern:

When calculating the next generation from the last of the below images, we can see that the blinker on the left should keep its center, lose its left and right side, and gain a top and bottom. Looking at the first of the below images in comparison to the second sequentially, we can see that this is the case.



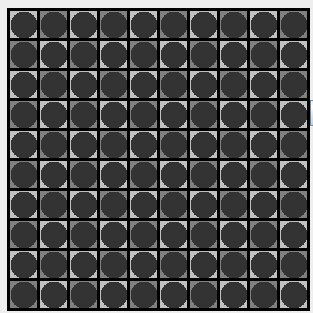
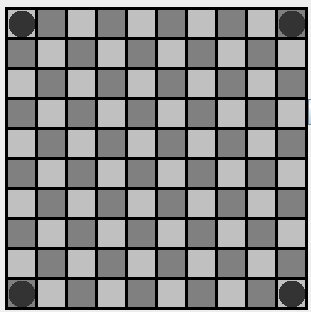
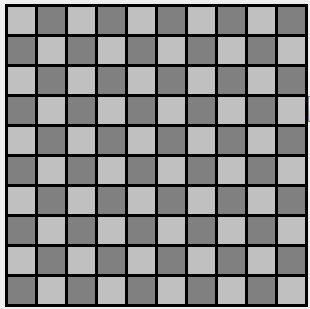
### 4. Glider Pattern:

The glider pattern is a pattern of cells which, after several successive generations without interference, reforms itself in its entirety one step below and one step to the right of its initial position. As can be seen in the below sequence of images, upon inputting a glider and advancing to the next generation several times, we can see that the glider reforms itself one tile down and one to the right, as expected.

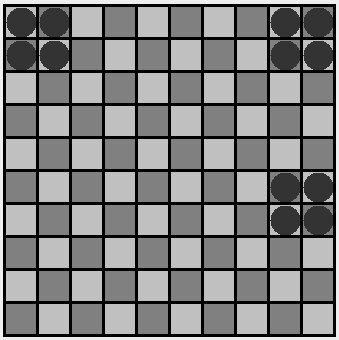
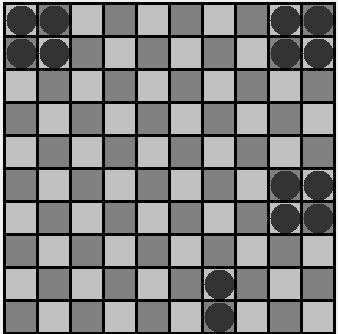
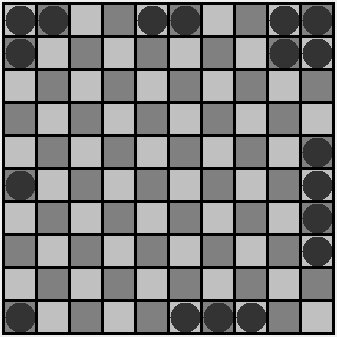


### 5. Edge Cases:

The following is a list of several edge cases, as well as pictures to depict their behavior in the running program.

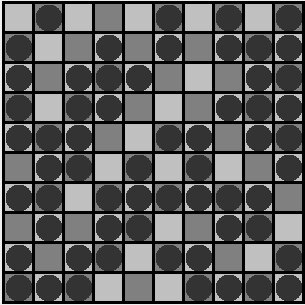
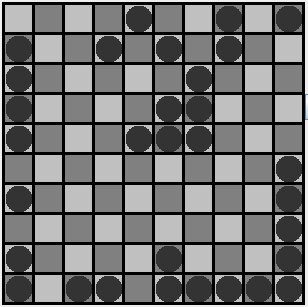
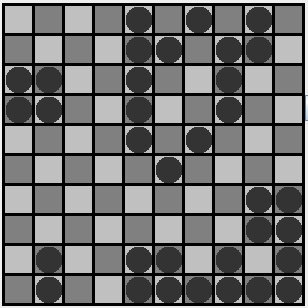
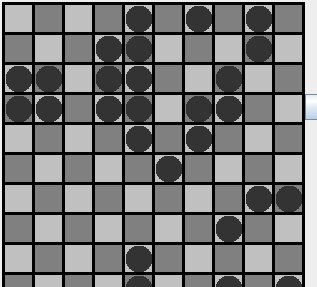
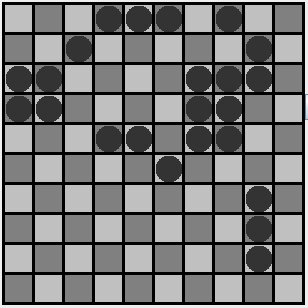
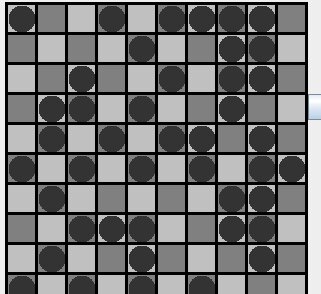
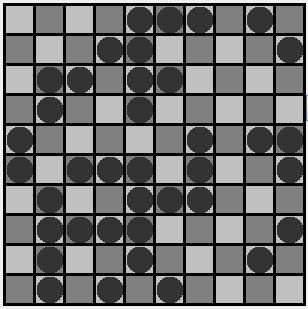
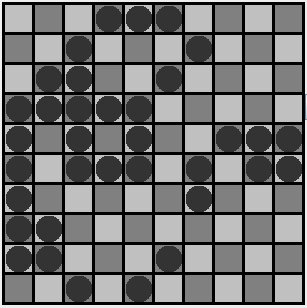
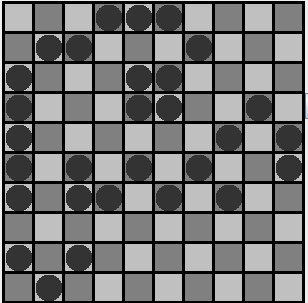
* When there is a full board, upon advancing to the next generation, only the corners should stay alive. Those corners should be dead by the next generation, and by the generation after that, nothing should happen, since there are no live cells.
  + 
  + 
  + 
* Creating a board with a size of 1x1, filling the cell, then progressing to the next generation should result in a dead cell on the next step, which is the behavior seen in the images below which were taken by carrying out this process.
  + 
  + 
* Creating a board of size 2x1 or 1x2, filling the cells, then progressing to the next generation should behave similarly, as can be seen in the below images.
  + 
  + 
  + 
  + 
* Creating a wide board with a height of 1, filling it, then proceeding through several generations should result in the cells on the edges of the board dying while the cells in the middle live, since those in the middle are each surrounded by two cells while those on the edges are only surrounded by one cell. This behavior can be seen in the images below.
  + 
  + 
  + 
  + 
* Creating a 2x2 board and filling it with live cells and then advancing to the next generation should result in a second identical generation, since each of those cells are surrounded by 3 cells when in a 2x2 configuration, thus they stay alive indefinitely. This behavior can be seen in the images below.
  + 
  + 

### 6. Grid Boundaries:

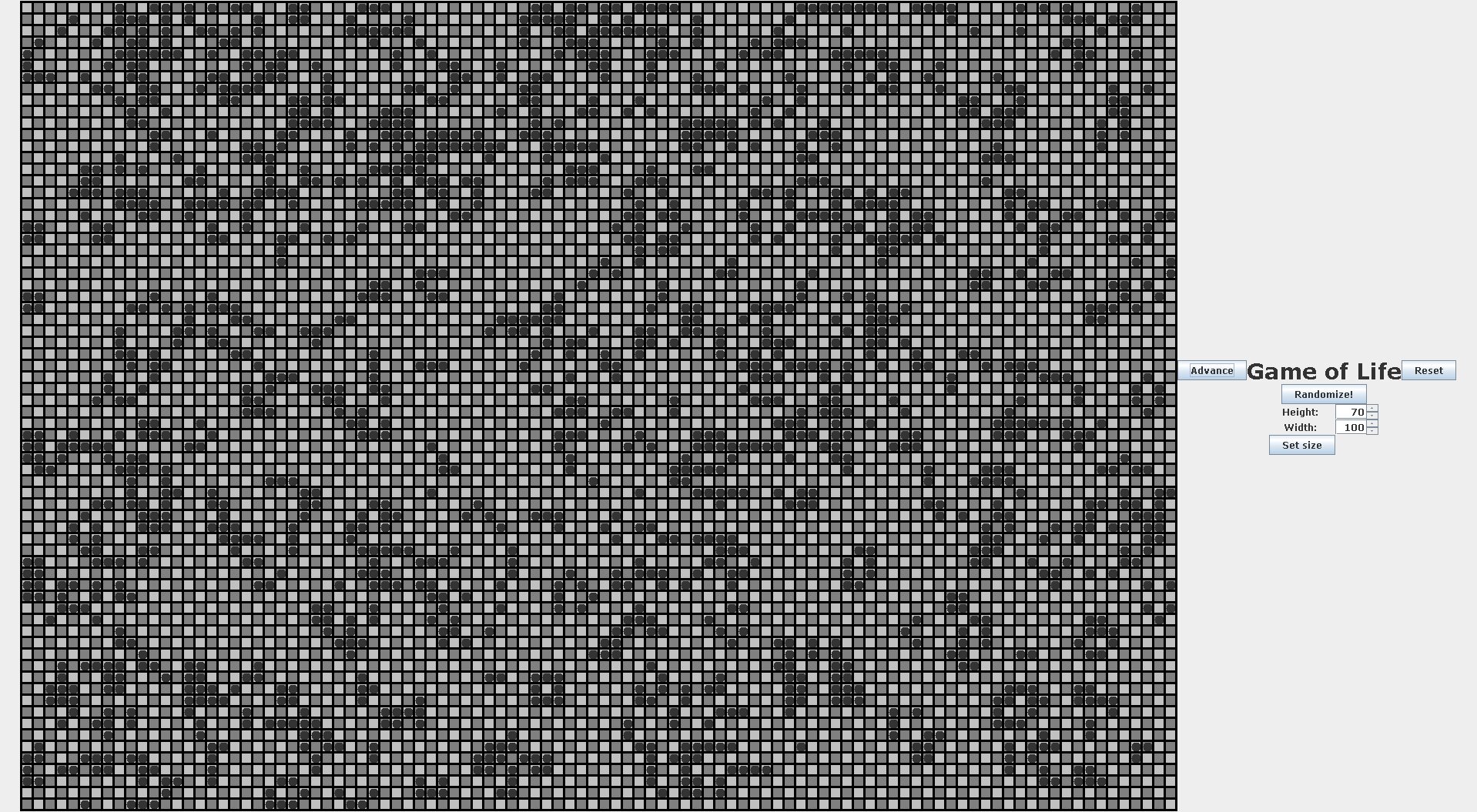
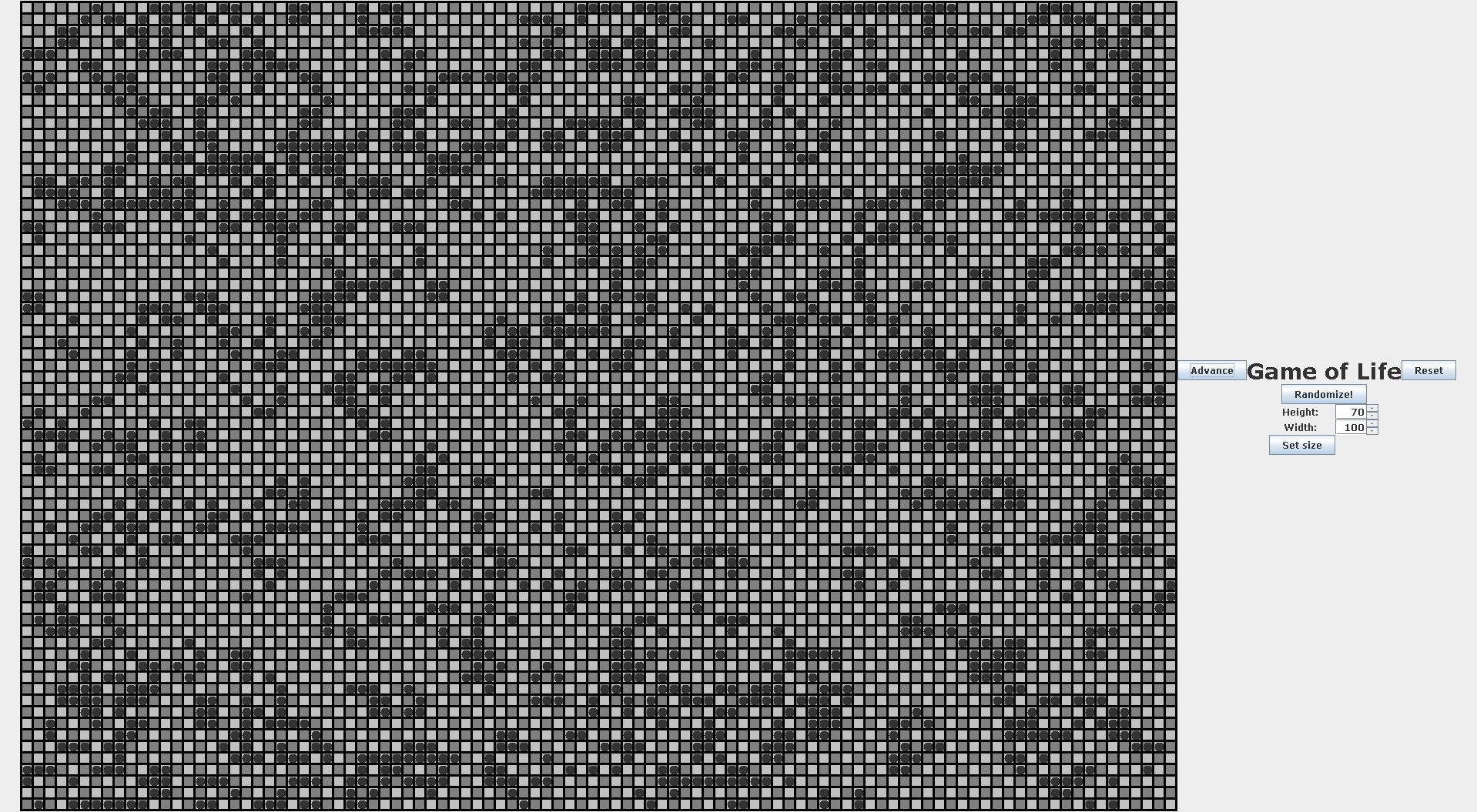
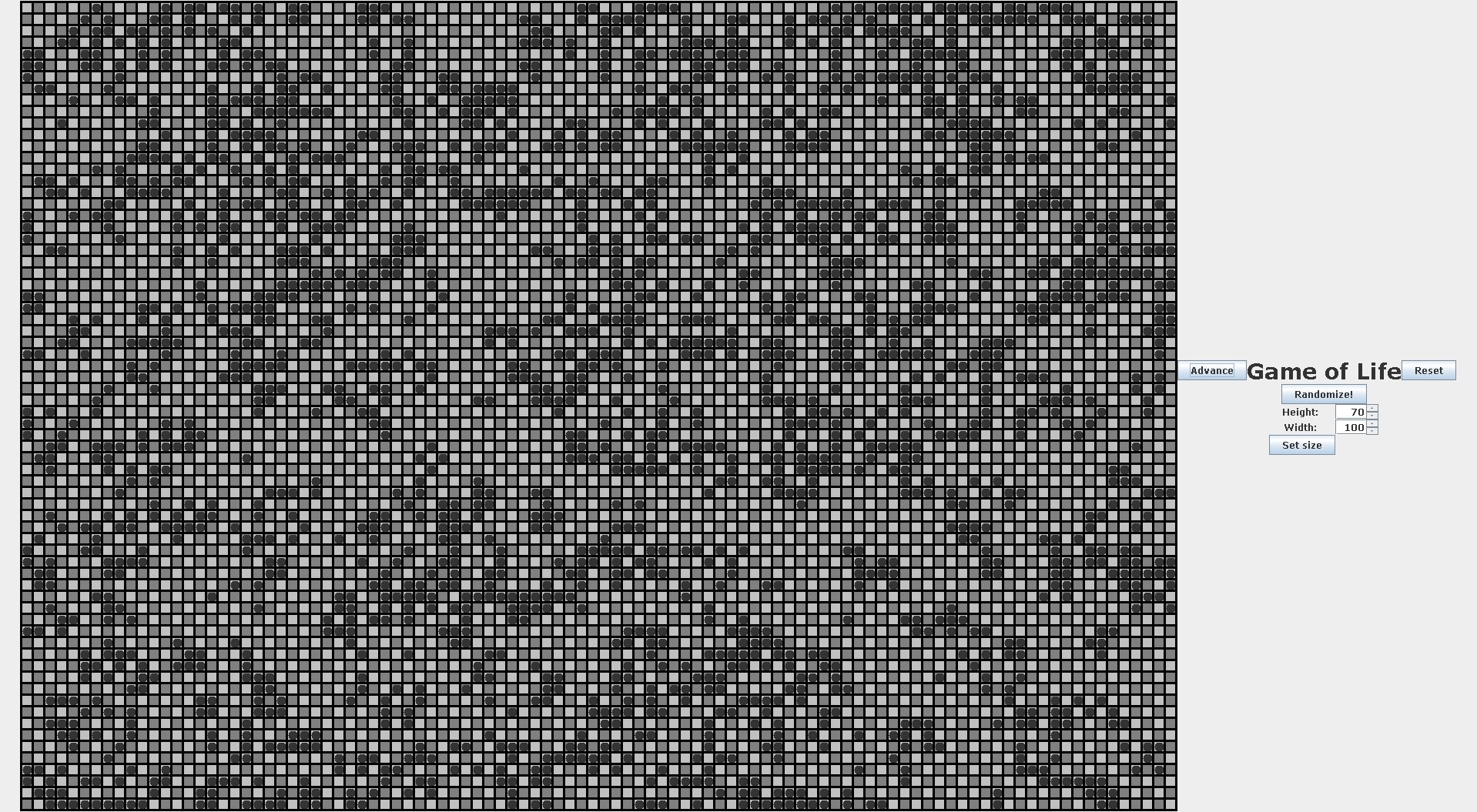
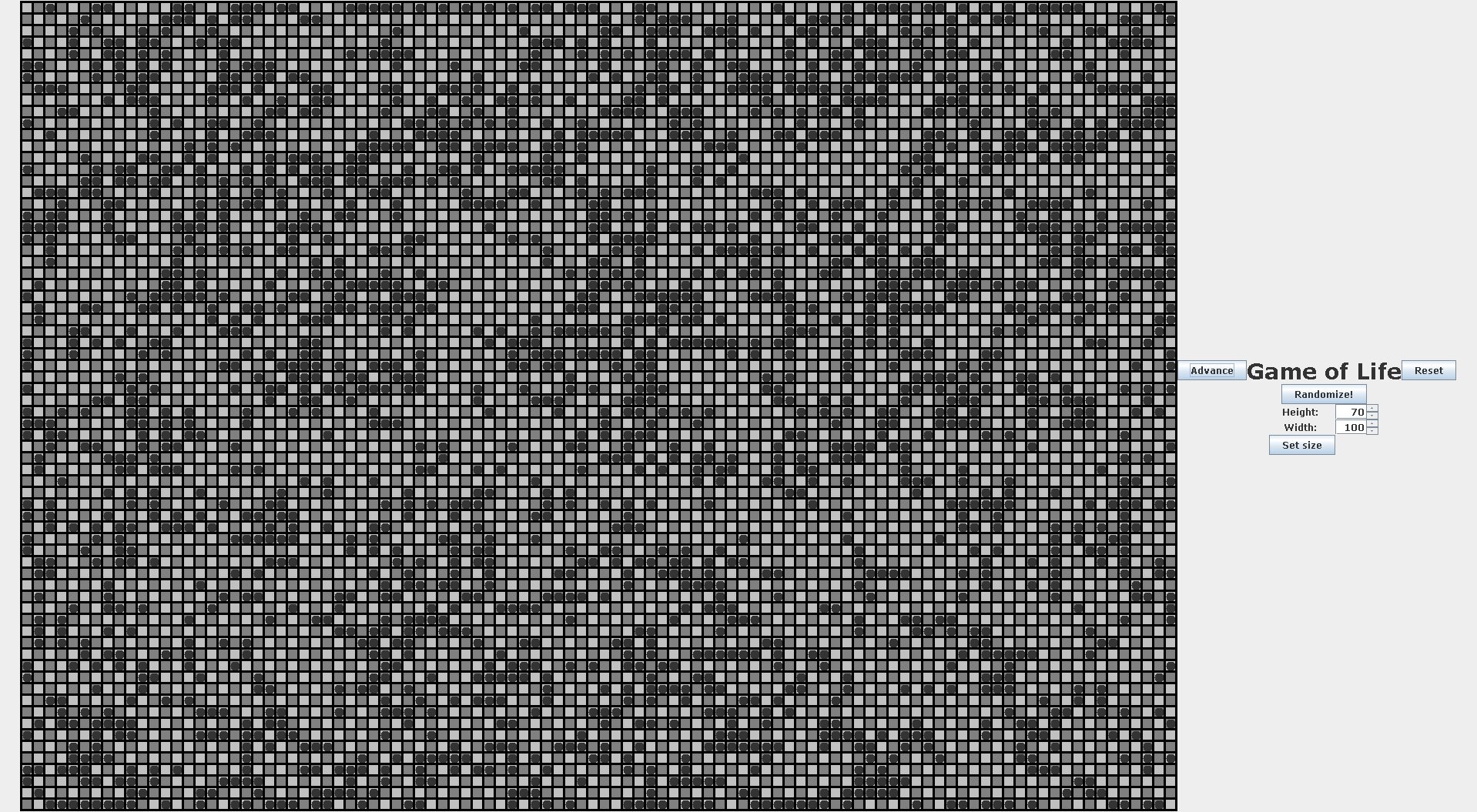
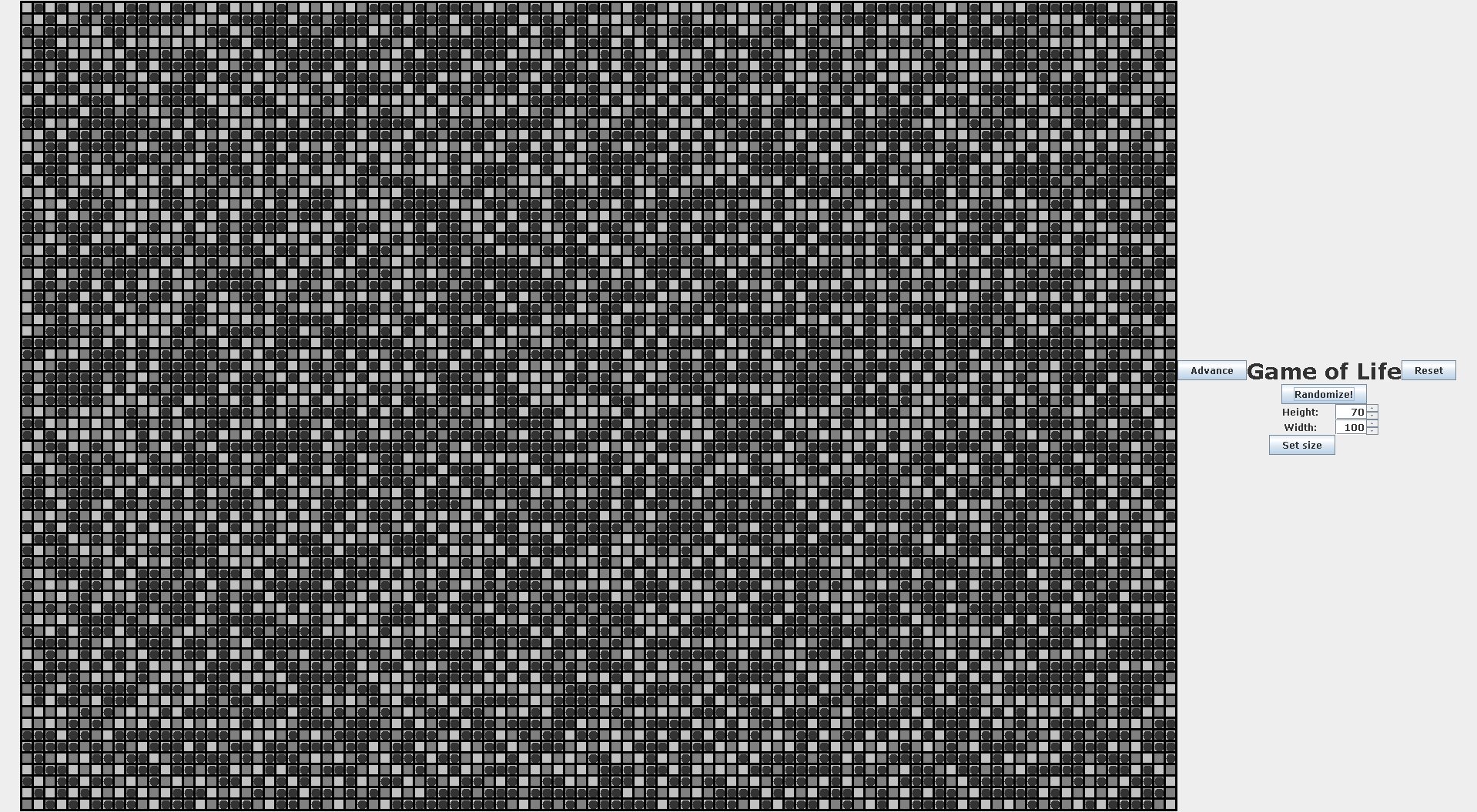
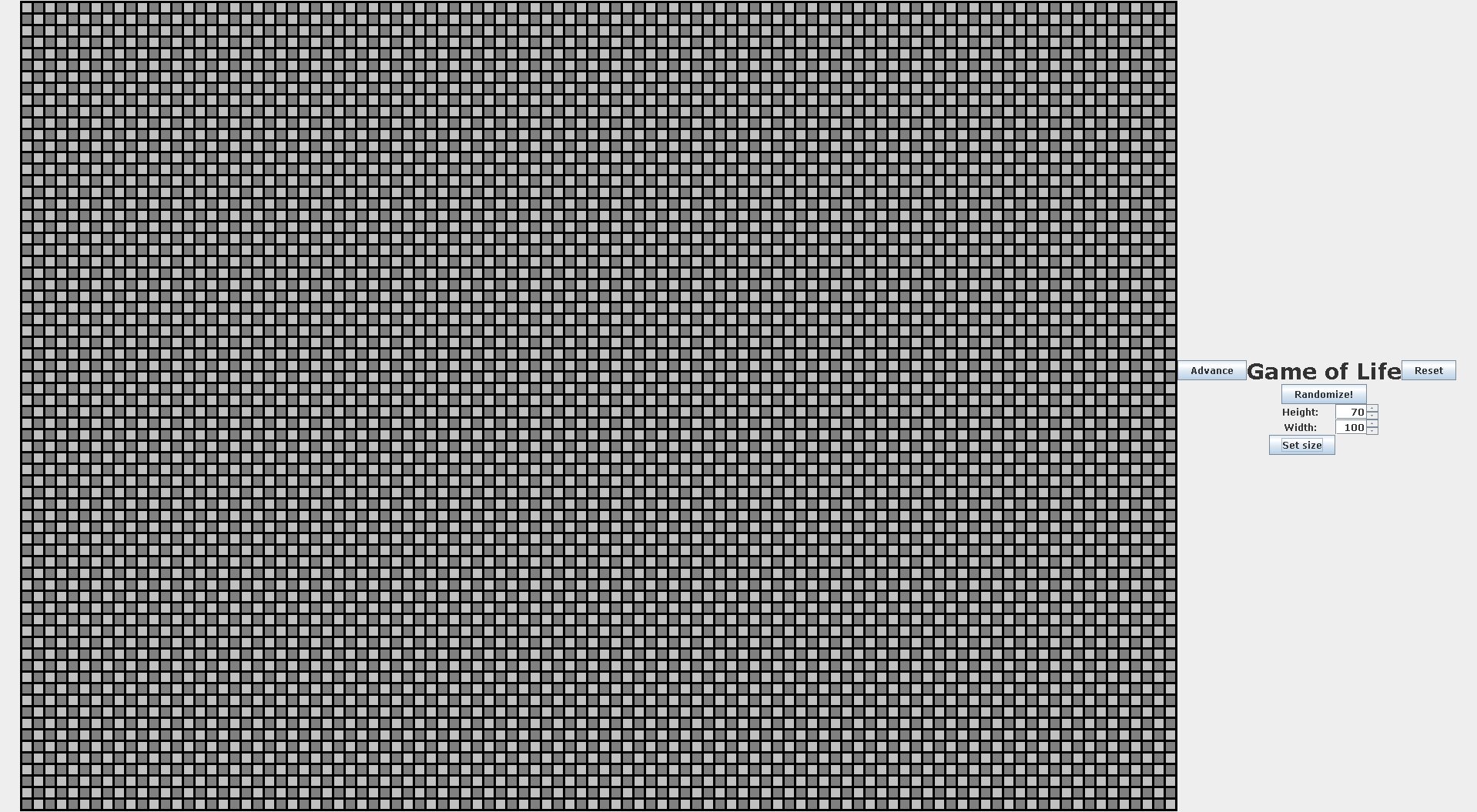


### 7. Random Patterns:

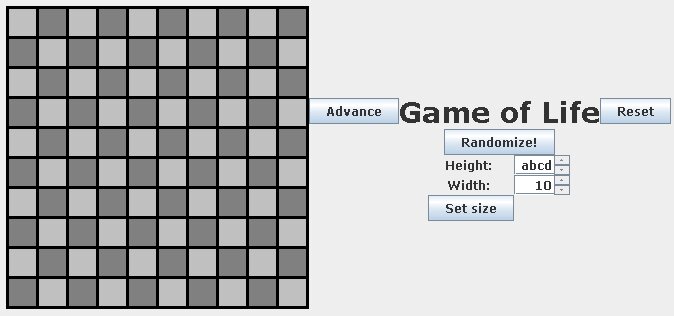
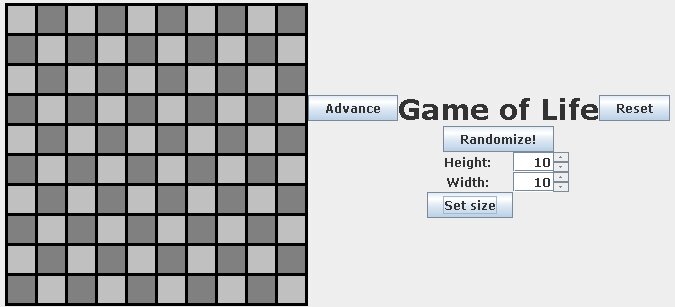
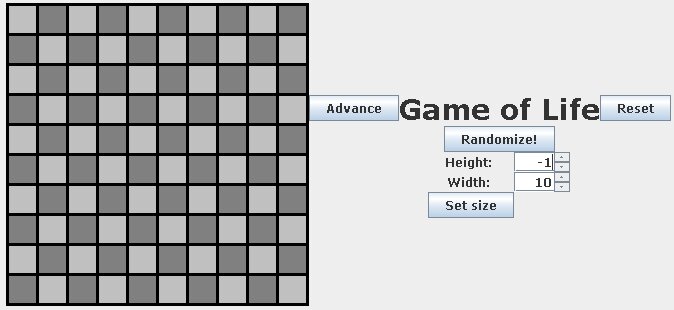
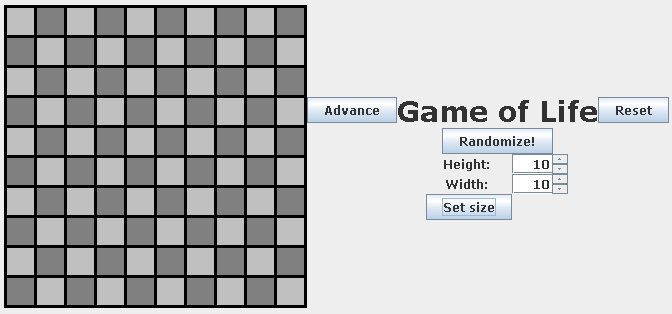
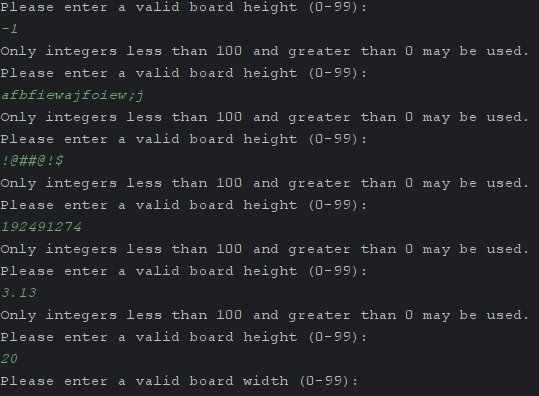
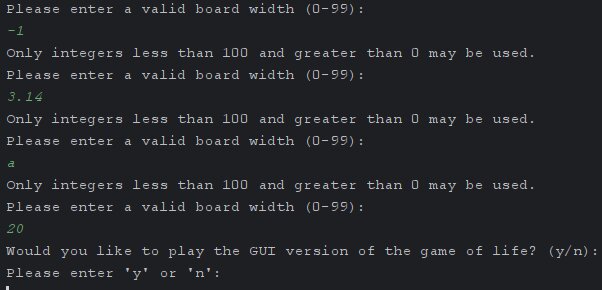
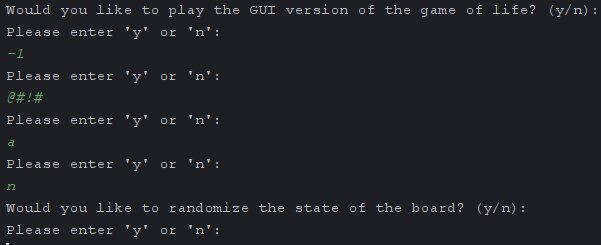
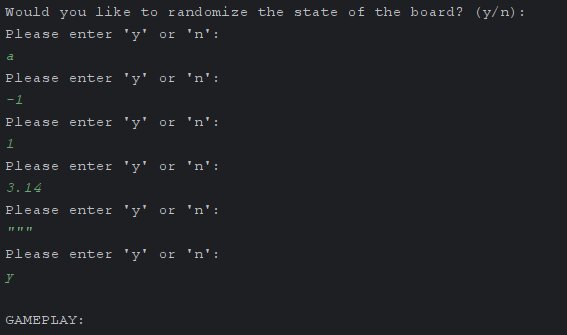
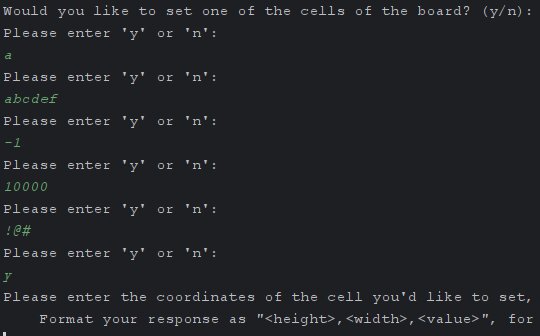
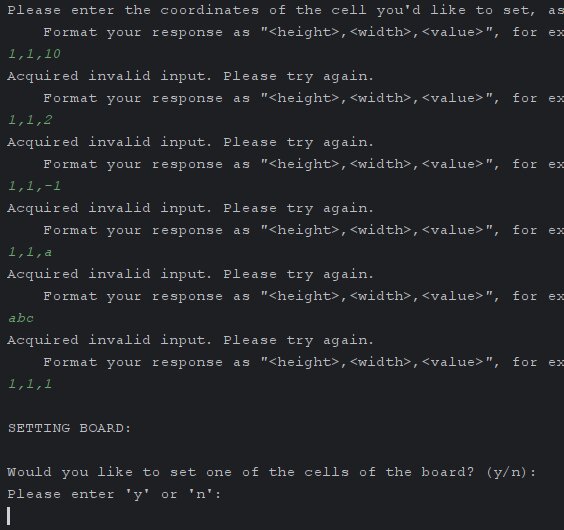
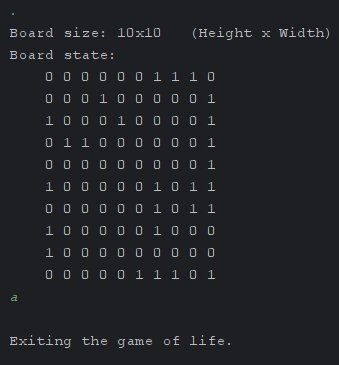
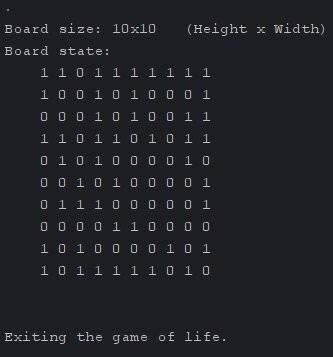
The following two series of images depict the progression of a board with a randomized initial state. As can be seen in the sequences, cells surrounded by more than 3 neighbors as well as cells surrounded by 1 or less neighbors die in the following generation, while cells surrounded by 2 or 3 cells will are alive in the following generation.

* Sequence 1:
  + 
  + 
  + 
  + 
  + 
* Sequence 2
  + 
  + 
  + 
  + 

### 8. Performance:



### 9. Invalid Inputs:

* GUI: Invalid height/width inputs:
  + 
  + 
  + 
  + 
* Console:
  + Entering something aside from an integer between 1 and 99 when entering board height should cause the program to ask the user for a new input, as seen below.
    - 
  + The same should be the case when asking for the width of the board, as seen below.
    - 
  + When asked to play the GUI version of life, if the user input is anything other than a y or a n, the user will be asked to input something new, as seen below.
    - 
  + The same is the case when the user is asked if they'd like to randomize the state of the board, as seen below.
    - 
  + The case is also the same when the user is initially asked if they'd like to set the state of one of the cells on the board, which only happens if they didn't decide to randomize the state of the board, as seen below.
    - 
  + When the user enters an invalid value for a cell during the cell setting process, they will be asked to try again, as seen below.
    - 
  + Entering a new line or anything other than . during the main gameplay loop should result in a clean exiting of the program, which we can see in the below images.
    - 
    - 

## V: Reflection:

I feel that overall this project went fairly well, however there are a couple of things that I think would be best done differently if I were to do it again. One thing I wish could have been done differently would have been to start sooner. I started quite late on this project, since I had a lot of other homework due earlier than this project was due, however if I did start this project earlier I believe it would have been able to be more polished, as well as include more features, such as a functionality which allows users to have the program automatically move on to the next generation after an input time period. The development of this project helped solidify some of my understanding of a few of the more fundamental concepts regarding object oriented programming and the use of the Java programming language, such as . For example, I learned how to create interactive GUI programs via the use of the Java Swing API. I also