Programming Assignment: Game of Life, Graphical Interface

Objectives

- Render (draw) the contents of a two-dimensional array.
- Use JButtons to control the processing of an application.

Overview

This is the second phase in the implementation of the Game of Life. In the first phase, Next Generation, for the Minimal Level, you created a string-based output for the Game of Life board. For the Standard Level, you implemented the ability to calculate the next generation of the Game of Life. This was strictly data manipulation.

The core task for this programming assignment is creating a graphical output component for the Game of Life, which essentially replaces the System.out.println - toString output that we used in the Next Generation phase.

Video Examples of the Levels

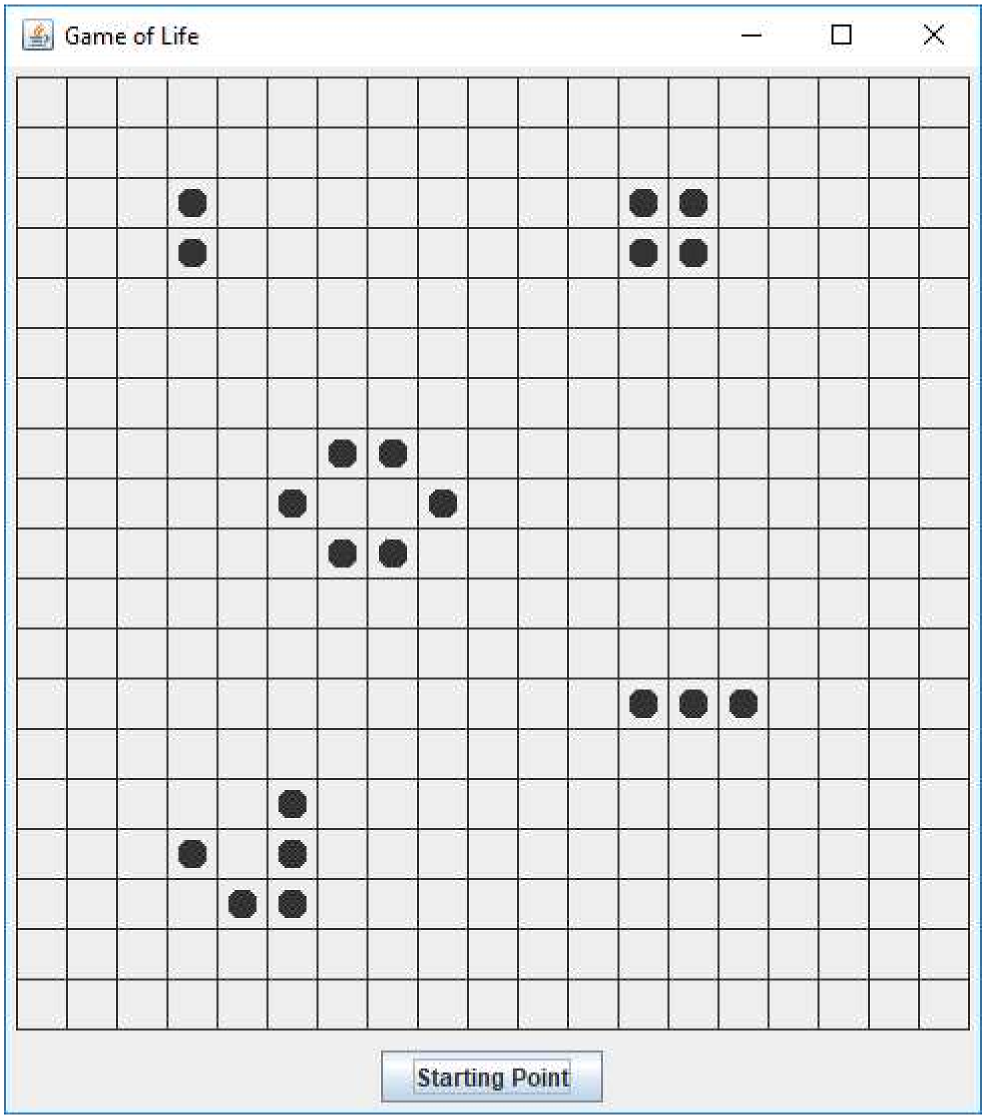
Here are video examples of the three different levels for this PA.

Background Material

- Next Generation
- Examples

Minimal Level

This is an example of what the Minimal Level application will look like as it begins.



Graphical Output Component

Create a graphical component. Make it a subclass of javax.swing.JPanel. This is an opaque component. Its default behavior is to render itself as a blank rectangle of its background color. Name this class **GameOfLifeBoard**.

The board is 19 rows of 19 cells. The cells shall have a border, 1 pixel thick on each of the four sides. Use a simple graphical shape to represent the live cells. The dead cells shall remain "empty".

The constructor for GameOfLifeBoard shall take a single parameter, the GameOfLife object which it represents. This GameOfLife object needs to be maintained within the GameOfLifeBoard. Use a private field to store the reference to this object. This will let GameOfLifeBoard call the methods of GameOfLife, most notably getCellState.

It is easiest if you select a size for the cell. Here are some suggestions based on what was used to create the video samples. The cells are 25x25. That is, there are 24 "blank" pixels between each of the vertical and horizontal border lines. The ALIVE cells are represented by a circle with a 15-pixels diameter. The circles are offset by five pixels vertically and horizontally to center them in the cell. Set the preferred size (setPreferredSize method) of GameOfLifeBoard so that is will display the whole board, including the border lines.

Override the paintComponent method of GameOfLifeBoard to draw the border lines and represent the life cells with the chosen graphical element.

Application

The GameOfLifeBoard needs to put into a top-level container to be visible.

Create an application method (main). This can be in the GameOfLifeBoard class. In main, create a JFrame. You can (should) set the location and title of the window. The size will be set by calling pack. This is why the GameOfLifeBoard needs ot set its preferred size. Remember to set the default close operation so that the application will end when the window is closed.

Instantiate a MyGameOfLife object. This is the core of the game. Pass this into the GameOfLifeBoard constructor as you create the graphical output object. To get a little empty space around the board, put it into a javax.swing.JPanel. Remember the default layout manager for JPanel is java.awt.FlowLayout. This layout manager tries to make the components their preferred size and adds a little empty space around each of the components. Place the JPanel which contains the GameOfLifeBoard object into the center (default position) of the JFrame. (Remember the default layout manager for a JFrame is BorderLayout.)

Set the MyGameOfLife object with live cells. Leverage (copy and edit) the setBoard code from TestGameOfLife.

Create a toolbar with a single JButton instance in it. Label the JButton with the text "Next Generation". The toolbar shall be a second JPanel instance. Add the JButton to the toolbar (JPanel) and place the toolbar in the window. The sample videos have the toolbar added to SOUTH.

Add an action listener to the toolbar button that calls nextGeneration on the GameOfLife object and then repaint on the GameOfLifeBoard object. The handler needs access to both of these objects. This will dictate how the code needs to be structured. Since the GameOfLife object is stored as a field in GameOfLifeBoard, the actionPerformed override can be a method of GameOfLifeBoard. The JButton for Next Generation is local to main, though.

At the end of the application method, make the JFrame object visible and pack it to set its size.

Tada, you should have a working Minimal Level for this programming assignment.

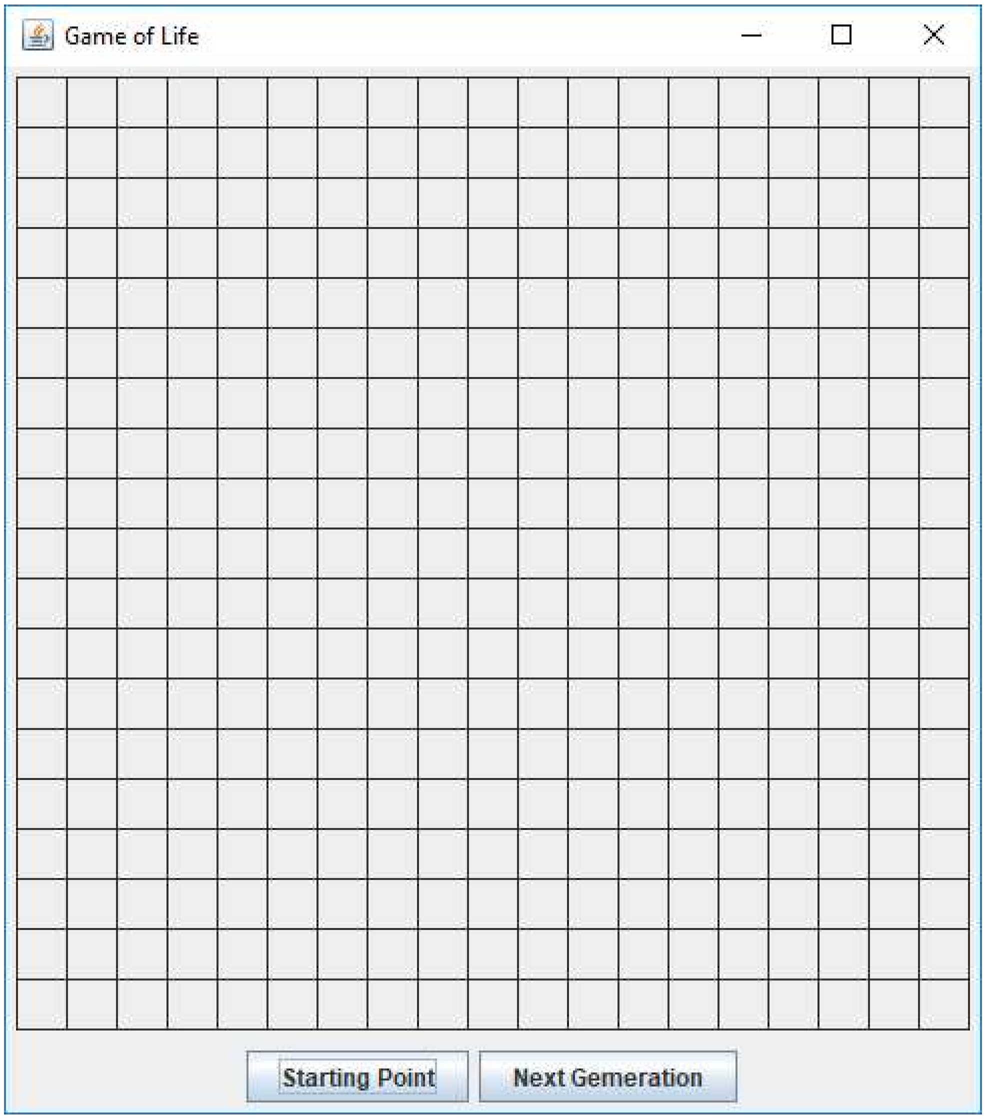
Standard Level

For the standard level, there are a few minor updates.

The GameOfLifeBoard will begin with all the cells dead. The is the default configuration for MyGameOfLife, since it uses an int array to store the state of the board, and int arrays are initialized to the value zero. This is the constant value DEAD in GameOfLife.

Now the toolbar contains two buttons: Starting Point and Next Generation.

This is an example of what the Standard Level application will look like as it begins.



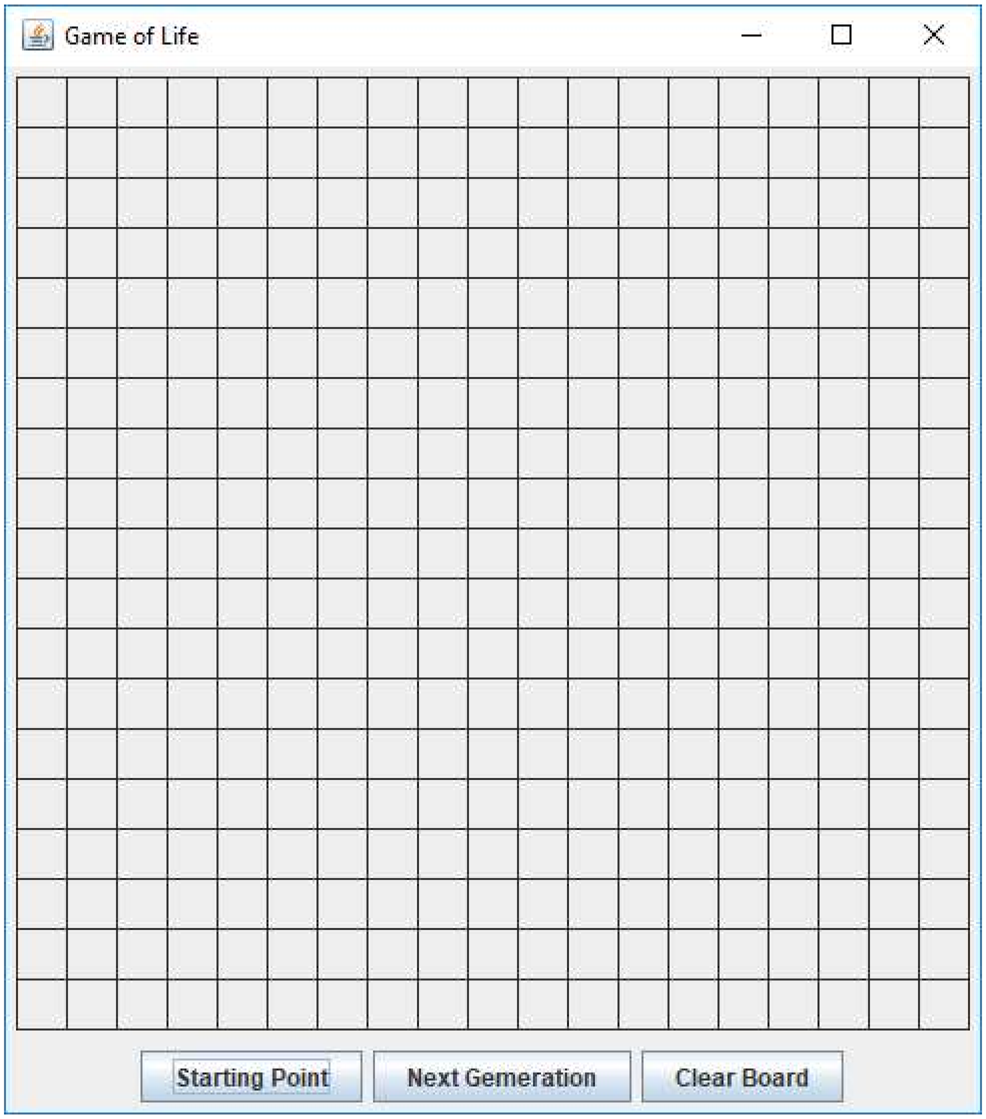
The change is the addition of the second JButton to the toolbar, Starting Point.

It is more straight-forward to create two separate implementations of ActionListener for these two buttons. Actually, the Next Generation handler has already been completed as part of the Minimal Level. Both handlers need to access both the MyGameOfLife instance and the GameOfLifeBoard object. Since each implementation of ActionListener can only have one override of actionPerformed, to create separate implementations, you will need to use an inner class for at least one implementation of ActionListener. You cam restructure your code to make two inner classes, if you like the parallel structure in your code. You could even use anonymous inner classes in main to implement these handlers.

Challenge Level

For the Challenge level, there are two enhancements.

This is an example of what the Standard Level application will look like as it begins.



The addition of the Clear Board button is relatively trivial. The work of the Challenge Level is implementing a MouseListener for GameOfLifeBoard. When the mouse button is clicked, the state of the corresponding cell will toggle. This is the mouseClicked method in MouseListener.

The java.awt.event.MouseEvent class has getX and getY methods that give the coordinates of the mouse action within the component. A bit of algebra will translate from the x- and y-coordinates to the row and column of the cell. Then toggling the cell state is trivial.

```
if(board.getCellState(row, column) == board.ALIVE) {
    board.setCellState(row, columm, board.DEAD);
} else {
    board.setCellState(row, column, board.ALIVE);
}
```

Notice that the constant values ALIVE and DEAD are inherited by MyGameOfLife.

Remember to repaint the graphic component after the cell state toggles.

Written Report

Write a report in which you describe:

- how did you go about starting this project?

- what works and what doesn't?
- the surprises or problems you encountered while implementing this application
- the most important thing(s) you learned from this assignment
- what you would do differently next time

I expect a clear report that has some thought in it. It will be easiest if you take notes about the process as you work on the assignment, rather than waiting till the end to start the written report.

Submission

- Two (2) Java source code files
  - MyGameOfLife.java (could be the same as PA1)
  - GameOfLifeBoard.java (new file, application)
- One (1) ASCII, plain-text report file (.txt)

Grading Rubric

Functionality: 8 points
Minimal Level - 5 points
 - GameOfLifeBoard Is-A JPanel, sets default size, draws border lines and live cells
 - application runs, displaying a board (19x19 grid) and a functional Next Generation button in a toolbar
Standard Level - 8 points
 - Minimal Level functionality
 - two JButtons: Starting Point and Next Generation with separate implementations of ActionListener
Challenge Level - 10 points
 - Standard Level functionality
 - mouse click handler on GameOfLifeBoard to toggle ALIVE/DEAD state
Note: Only adding the Clear Board button will earn no Challenge Level points.

Style: 2 points
 - conforms to Homework Guidelines regarding style

Documentation: 3 points
 - JavaDoc comments on all public and protected elements
 - a separate comment on each class, field, method (can be JavaDoc comment)
 - JavaDoc comments generate no warning or error messages
 - each JavaDoc comment include a brief description in addition to required tags
 - method / constructor internal comments to explain functionality
 - report addresses the questions posed.

Back