CS 1331 Homework 7

Due: Friday, October 23, 2009

Introduction

This assignment will give you practice in developing a larger, multi-class application and in constructing a more complex graphical interface involving mouse selections.

You'll notice that the style of this homework is different from the previous homeworks. Rather than giving step by step programming instructions, this homework describes how the program should behave. Feedback about the two styles is appreciated (email me directly at ahumesky3@gatech.edu, prepend the subject with "[cs1331]").

The Game of Life

The "Game of Life" (not to be confused with the Parker Brothers edition) is a well-known mathematical simulation that can mimic some very complex behavior. The game tries to mimic the life cycle of cells.

The rules for the game are really very simple:

- The game is played on a rectangular board.
- The board is made up of squares.
- Each square can be either empty or occupied (contains a cell).
- To start the game, the user selects which squares are initially occupied. This is the zeroith (start) generation.
- For each generation, the next generation is computed. The transition from one generation to another is called a turn.
- In each turn the next generation is computed as follows:
    - If an empty square is surrounded by exactly 3 neighbors, a new cell is born (that square becomes occupied).
    - A cell dies of overcrowding if it is surrounded by 4 or more neighbors (that square becomes empty).
    - A cell dies of loneliness if it has zero or one neighbors (that square becomes empty).
    - If a cell has 2 neighbors it remains as it is (square stays either empty or occupied)

Neighbors are defined as any adjacent square left, right, up, down or diagonally. The diagram below depicts all the neighbors of a center cell.

```
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
| 7 | 8 | 9 |
+---+---+---+
```

```
So a cell at 5, would need to check its neighbors at
1,2,3,4,6,7,8,9.
```

If a cell is at a border, then treat all locations off-board as empty.

Here are a few sample illustrations to help demonstrate the rules. A C means the square is occupied by a cell.

```
+---+---+---+
| C |   |   |
+---+---+---+
|   | C |   |
+---+---+---+
|   |   |   |
+---+---+---+
This center cell would die (become an empty square) in the next
generation since it only has 1 neighbor.
```

```
+---+---+---+
|   |   |   |
+---+---+---+
| C | C | C |
+---+---+---+
| C |   | C |
+---+---+---+
This center cell would die (become an empty square) in the next
generation since it has 4 neighbors.
```

```
+---+---+---+
| C |   |   |
+---+---+---+
|   |   |   |
+---+---+---+
| C |   | C |
+---+---+---+
 This center cell would be born (become an occupied square) in
the next generation since it has exactly 3 neighbors.
```

Program Requirements

Your assignment is to program the Game of Life subject to the rules described above. The game advances one turn per press of the Step button.

Here is one possible architecture:

- GameOfLife.java (Main startup class, makes the JFrame)
- GamePanel.java (GUI, displays everything)
- Square.java (Represents one square on the grid)
- CellModel.java (Holds a generation)

The only official grading criteria with regards to code organization is that you follow standard OOP principles and that you submit a file called GameOfLife.java with a main method, but other than that, you may implement it how you see fit (note that if your program is all in a single class, that’s probably not good OO programming). See the Hints section below if you want guidance.

User Interface

Your program should run as a GUI application in a JFrame. You should have a JPanel-derived class containing at least 2 buttons (Step, and Clear), a JLabel, and a drawing area.

The buttons consist of:

- Step: Advance the game one turn, calculate and show the next generation. This operates the simulation in single step mode.
- Clear: Clear all the cells, Make all squares unoccupied

You should display the current generation number (turn number) in a JLabel.

In the drawing area, you should display the grid representing the squares that hold the cells. The grid size should be 20 x 20. You can choose the cell size, but make it big enough so that you can easily click any particular cell.

When starting the game, the user should be allowed to click with the mouse inside each cell to make that cell occupied or empty. Clicking on an empty cell makes it occupied. Clicking on an occupied cell makes it empty. Once the user presses the Step button, mouse clicks in the grid area are ignored until the clear button is pressed.

When a square is occupied it should be filled in with the color of your choice. All squares with the same state should be the same color. (For the ambitious: newly born cells in each generation can be colored with a different color from the older cells in a generation).

Hints

Note again that you are free to organize your code and classes in whichever way you like as long as it's consistent with OOP and produces the interface described above. Here’s a suggested implementation:

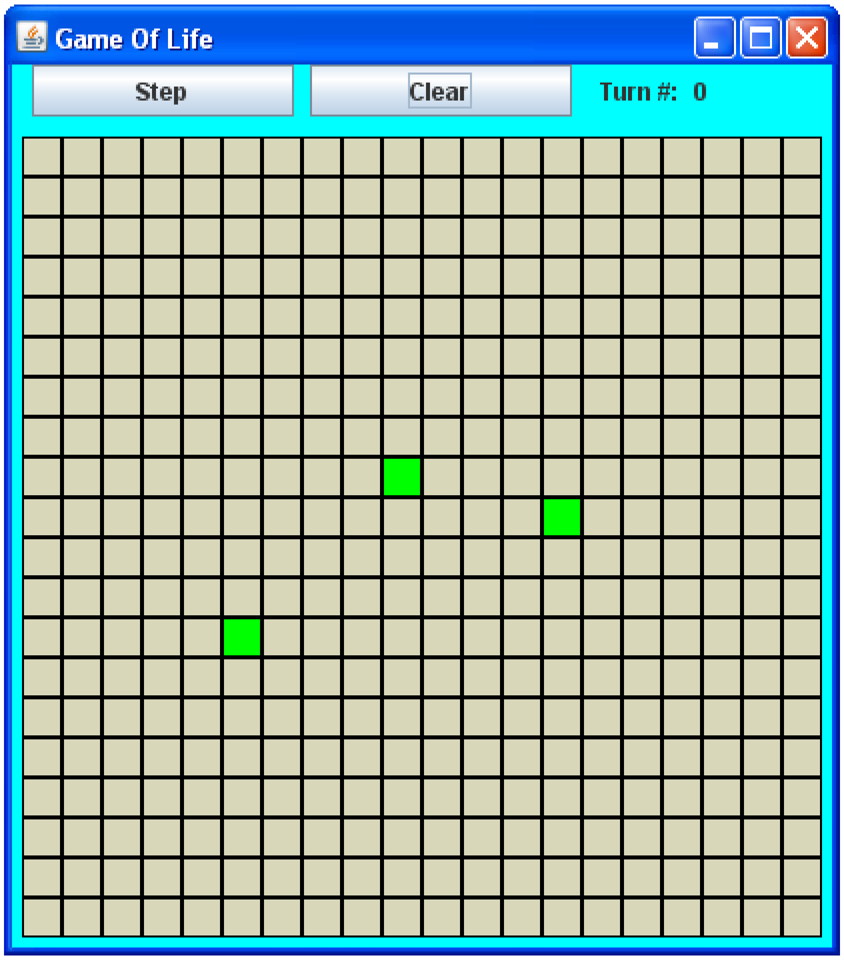
- Square represents a square on a grid. It would be useful if a Square object knew its own status (occupied or empty).
- CellModel should hold information about a generation, meaning it needs some kind of representation of the board of Squares (maybe a 2D array?)
- CellModel should have a step() method which advances the generation. This way, your panel's button simply needs to call step() and the timer has nothing but a call to step() as well, followed by an update of the GUI.
- Your GamePanel should have a CellModel object that it uses to hold data (remember that CellModel holds information about a generation). Whenever the user clicks the Step button, the GamePanel should advance the CellModel one step (using the step() method),

then check each square in the model and update the GUI appropriately, changing colors of cells that have changed status.

- CellModel will need to perform a lot of work on each cell to advance a step (find out how many neighbors a cell has, determine if this makes it occupied or empty, etc.) It would make your code more organized to split this functionality into different methods.

Here's a sample implementation of the game of life: http://www.bitstorm.org/gameoflife/

Here's a screenshot of one implementation. Yours might be different.



Some special start cell sequences

Here are some interesting starting sequences. You can also use these to test your application.

The glider. The glider is cool because it repeats itself every few generations and seems to move across the screen. It will move down and right every 4 generations.

```
Generation 0 (Start)

+---+---+---+---+---+
|   | C |   |   |   |
+---+---+---+---+---+
|   |   | C |   |   |
+---+---+---+---+---+
| C | C | C |   |   |
+---+---+---+---+---+
|   |   |   |   |   |
+---+---+---+---+---+
```

```
Generation 1

+---+---+---+---+---+
|   |   |   |   |   |
+---+---+---+---+---+
| C |   | C |   |   |
+---+---+---+---+---+
|   | C | C |   |   |
+---+---+---+---+---+
|   | C |   |   |   |
+---+---+---+---+---+
```

```
Generation 2

+---+---+---+---+---+
|   |   |   |   |   |
+---+---+---+---+---+
|   |   | C |   |   |
+---+---+---+---+---+
| C |   | C |   |   |
+---+---+---+---+---+
|   | C | C |   |   |
+---+---+---+---+---+
```

```
Generation 3

+---+---+---+---+---+
|   |   |   |   |   |
+---+---+---+---+---+
|   | C |   |   |   |
+---+---+---+---+---+
|   |   | C | C |   |
+---+---+---+---+---+
|   | C | C |   |   |
+---+---+---+---+---+
```

```
Generation 4

+---+---+---+---+---+
|   |   |   |   |   |
+---+---+---+---+---+
|   |   | C |   |   |
+---+---+---+---+---+
|   |   |   | C |   |
+---+---+---+---+---+
|   | C | C | C |   |
+---+---+---+---+---+
```

Turn-in Procedures

After you have finished your program, turn the files in via T-Square.

- GameOfLife.java (Main startup class)
- Any other files needed to run your program

Don’t forget to Javadoc your classes and non-trivial methods and your collaboration statement.