v2 13 Oct 2016

6 Homework Assignment

The Game of Life

The Game of Life was devised by the British mathematician John Horton Conway in 1970. The "game" actually describes how an initial configuration evolves over time according to simple, precise rules. Despite the simple and precise rules, the evolution from the initial configurations is hard to predict and quite similar looking initial configurations may give rise to quite different developments.

The universe of the Game of Life is a two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, alive or dead. Every cell interacts with its maximally eight neighbors, the immediately adjacent cells in orthogonal and diagonal direction. Cells on the border have less than eight neighbors. At each step in time, the following transitions occur:

1. Any live cell with fewer than two live neighbors dies, as if by loneliness.
2. Any live cell with more than three live neighbors dies, as if by overcrowding.
3. Any live cell with two or three live neighbors lives, unchanged, to the next generation.
4. Any dead cell with exactly three live neighbors comes to life.

The initial generation is read from standard input. A new generation is created by applying the above rules on every cell, using the current generation, as if the rules were applied simultaneously on all cells.

Write a program using the following classes. Add methods and variables as mentioned below, including the mentioned names and types, and use them. This is a minimum, more are allowed, of course. **Stick to the names and types mentioned here.** Peach uses these methods and variables to test your submission. Deviations will cause us trouble and you a lower grade.

Do not access instance variables directly from other classes; use methods instead.

6.1 Cell

An object of the class Cell represents one cell, living or dead, in the grid. The class Cell has a boolean instance variable `alive` that is `true` when the cell is alive and `false` when the cell is dead. Furthermore, it has an `int` variable `numNeighbors` that represents the number of alive neighboring cells.

The class Cell is a subclass of JLabel and implements the interface MouseListener.

Methods in Cell:

- `void setAlive(boolean state)` changes the alive/dead state of the cell;
- `boolean isAlive()` returns the alive/dead state of the cell;

- `void setNumNeighbors(int n)` sets `numNeighbors` of the cell to `n`;
- `void update()` takes the cell to the next generation: it changes the instance variable `alive` according to the rules above, using the value of the instance variables `numNeighbors` and `alive`.

6.2 GameOfLife

This is the main class of the application. There will be one object of this class in your application.

It starts up as follows.

It will create a GUI with a row of button(s) and a panel that shows the grid of cells. It will read the initial configuration from a file on disk. If the file is not available, it will use a fixed initial configuration. After this, it will show the initial configuration. When the user presses the start button, an animation of successive generations will be shown. Pressing the stop button will pause the animation and pressing the start button will resume the animation. It is allowed to use one button for both starting and stopping, provided that the button is appropriately labeled.

Instance variables of GameOfLife

- `Cell[][] grid` contains the grid of cells that represents the universe of the game of life;
- `String birthFilename` is the name of the file where the initial generation is stored. This file should be in the same folder as your Java files. Set this variable to `birth.txt`.

Methods of GameOfLife

- `void calculateNumNeighbors()` calculates the number of living neighbors of each cell and sets `numNeighbors`; it will *not* update the dead/alive state of the cells.
- `void readInitial()` creates the grid and reads the initial generation from the file with the name stored in cd birthFilename.
- `void nextGeneration()` updates the grid to the next generation, using the values of `numNeighbors` in the cells.

6.3 File format

The configuration file is a text file consisting of:

1. a line with two integers with minimal value 2 that represent the number of rows and the number of columns of the grid, in that order;
2. h lines consisting of w characters that are either `*` (for a live cell) or `.` (for a dead cell). The characters are separated by spaces. Note that this makes it possible to read the values for the cells by successive calls to `next()` on a `Scanner` object.

Example of file contents

```
13 15
. . . . . . . . . . . . . . * .
. . . . . . . . . . . . . * . .
. . . . . . . . . . . . . * * *
. . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . .
* * * * * * * * . . . . . . . .
. . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . .
```

6.4 Output

6.5 Animation

Using a Timer object, the screen should show successive generations of the grid with a reasonable amount of time between them. The interval should be larger than 0.1 second and less than 1 second.

One or two buttons will start c.q. resume and resume the animation.

6.6 Extensions

The basic version of the Game of Life will earn you a maximum of 7.5 points. More points can be earned with the following extensions. The extensions can be done in any order.

If you have implemented extensions, mention them on top of the file with the GameOfLife class which extensions you have implemented. E.g., `//Extensions:  1, 3`

Show this text also in a label somewhere in your GameOfLife application.

6.6.1 Interactive live and dead setting

Pressing the left mouse button on a cell will turn that cell from dead to alive or vice versa. This will be possible when the animation is paused. It is allowed that it is also possible to change the alive/dead state of cells during the animation.

This will earn you 1 point extra when implemented well.

6.6.2 Dragging

When pressing the mouse button on a cell and then dragging the mouse (moving while keeping the button pressed) will have the following effect. The first cell will change from dead to alive or vice

versa and the other cells that are passed by the mouse will get the same value as the first cell.

This will earn you 0.5 point when implemented well.

6.6.3 Fading

Suppose your cells are black when alive and white when dead. (You may choose other colors, but we use these colors for the explanation here.) When a cell dies it will not immediately turn white, but it will slowly fade to white during 10 successive generations. On the first generation that the cell is dead it will have a color gray that is quite different from black, e.g., in the middle between black and white. After that, it will fade in 9 steps to white.

The rules of life and death are not changed. A gray cell is counted as dead. When a gray cell has exactly three live neighbors, it will come alive and turn black immediately.

You are required to implement this using a subclass of Cell named `FadingCell`. A FadingCell will have one or more additional instance variables that keep track of the "age" of the cell, i.e., the number of generations that the cell is dead. Furthermore, several methods of Cell will have to be overridden in FadingCell.

This will earn you 2 points when implemented well.

6.7 Submission

In the submitted version, **use no output**.

6.8 Example

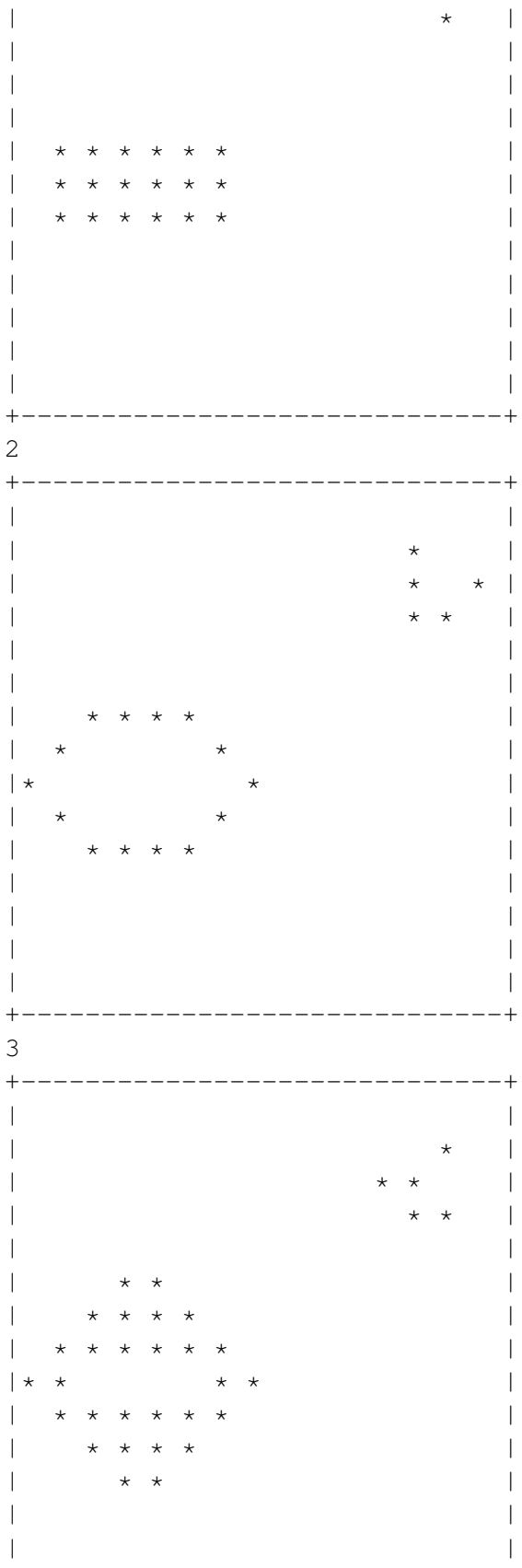
The file

```
15 15
. . . . . . . . . . . . . . *  .
. . . . . . . . . . . . . . *  .  .
. . . . . . . . . . . . . . *  *  *
. . . . . . . . . . . . . . .  .  .
. . . . . . . . . . . . . . .  .  .
. . . . . . . . . . . . . . .  .  .
. . . . . . . . . . . . . . .  .  .
. . . . . . . . . . . . . . .  .  .
* * * * * * * * . . . . . . .  .  .
. . . . . . . . . . . . . . .  .  .
. . . . . . . . . . . . . . .  .  .
. . . . . . . . . . . . . . .  .  .
. . . . . . . . . . . . . . .  .  .
. . . . . . . . . . . . . . .  .  .
. . . . . . . . . . . . . . .  .  .
```

should give show the following generations (here shown in textual form and numbered, for practical purposes; in your application, they should show as an animation of colored rectangles).

Note that in the top right corner sits a so-called *glider*, a shape that moves diagonally across the board.

```
0
+------------------------------+
|                            *    |
|                          *      |
|                          * * *  |
|                                 |
|                                 |
|                                 |
|                                 |
|                                 |
|* * * * * * * *                  |
|                                 |
|                                 |
|                                 |
|                                 |
|                                 |
|                                 |
+------------------------------+
1
+------------------------------+
|                                 |
|                          *    * |
|                          * *    |
```


2
3