Programming Project 5

Due: Thursday, 4/1/15 at 11:59 pm

Conway's Game of Life

For this programming project, write a Java program that will simulate the population growth algorithm as described by John Conway's Game of Life. A good web page describing this can be found on the Wikipedia at: http://en.wikipedia.org/wiki/Conway's_Game_of_Life. The rules of this simulation show how an organism could grow or shrink from one generation to the next. The universe of the Game of Life is an infinite 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 eight neighbors, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

1. Any live cell with less than two live neighbors dies, as if caused by under-population.
2. Any live cell with two or three live neighbors lives onto the next generation.
3. Any live cell with more than three live neighbors dies, as if by overcrowding.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

All of the above transitions occur "at once" in the simulation. So your program will need two instances of the internal grid. One grid will hold the data for generation N, while the program computes generation N+1. After each generation is known, the alive positions in that generation are to be displayed. The storage structure for the internal grids is to be a dynamic 2D array.

This program will use the java GridDisplay class to display each generation which can be found in the file GD2.java. We will use one color to show the alive positions in the grid and other color to show the dead positions in the grid. Using the color so black and white will work just fine (or pick two other colors that you like).

General Description of Algorithm for This Project

- Read Data from File creating and initializing back-end grid A for generation 1
- Create back-end grid B for later use
- Create an instance of the GridDisplay class based on the row and columnvalues from the data file
- Set the colors in the GridDisplay instance based on the alive/dead positions in grid A
- loop (forever)
    - for each position in grid B
        - determine if that position will be alive/dead in generation N+1 based on the generation N data in grid A
    - Copy the information from grid B to grid A
    - Set the colors in the GridDisplay instance based on the alive/dead positions in grid A

Notes on Program Input

We will follow a number of input ideas used with the maze program, Project 2. The input will come from an input file whose name is given via the command line argument. This input file will contain the grid size and the initial alive position for generation 1 in the simulation. The file will always contains two integer values per line of input:

- The first valid line gives the size of the 2-D grid (the number of rows given by the first number, then the number of columns given by the second number), valid values are >= 1
- The remaining valid lines in the file give the coordinates of the initial alive positions in the grid. **The first value on the line will be the row index. The second value on the line will be the column index.**

The following shows an example of such an input file. The coordinates are given with the row listed first and the column listed second. A grid of NxM has rows numbered from 1 to N and columns number from 1 to M.

```
10 20
5 1
4 2
3 3
1 10
2 9
3 8
4 7
5 6
6 5
7 4
8 3
```

The above input creates a grid with 11 initial alive positions.

You may assume that the input will always have two integer values per line; however, the values may be out of range. If an invalid value is given on an input line, print a descriptive error message to STANDARD ERROR (using something along the lines of System.err.print() or System.err.println()), ignore those two input values and continue processing input using the next line of input. Any value of zero or less is invalid in this program. For a coordinate value in the grid, valid value range from 1 to the maximum row size or column size. Since errors may exist:

- The size of the grid is on the first valid line of input.

Input with invalid values is shown below. The comments are each line are not part of the input.

```
10 0          => Invalid: Grid sizes must be greater than 0
15 7          => Grid becomes size 15 x 7
10 20         => Invalid: column 20 is outside range from 1 to 7
5 1
24 2          => Invalid: row 24 is outside of range from 1 to 15
3 3
1 10          => Invalid: column 10 is outside range from 1 to 7
2 9           => Invalid: column 9 is outside range from 1 to 7
3 8           => Invalid: column 8 is outside range from 1 to 7
4 7
```

```
5 6
5 1            => Invalid? Position is already specified
6 5
7 4
8 3
```

Notes on the Game of Life Grid

When referring to the 8 neighbors, those positions will be the ones immediately above, below, left, right or diagonal of the current position. So for position x,y its 8 neighbors are at:

- x-1, y-1
- x-1, y
- x-1, y+1
- x, y-1
- x, y+1
- x+1, y-1
- x+1, y
- x+1, y+1

The original description of this program uses an infinite grid. That doesn't really work. Some implementations use a "wrap-around" grid where the top row is adjacent to the bottom row and the left column is adjacent to the right column. **We will not use either of these, instead we will use a standard fixed size grid.** All locations outside of the grid are considered as dead. Note, you are more than welcome to include an extra row and column on each end of the grid as we did with the Maze Program in Project 2. Thus a grid of size 10x20 would really contain 12 rows and 22 columns (as was done in the original maze project). This would allow your program to not have to check for the special boundary cases.

GridDisplay API

You will find a class GridDisplay in the file GD2.java. This class will allow you to display a 2-D grid of any size. You can also place any character at any grid position and color any grid position. The GridDisplay API (Application Program Interface) is as follows:

- public GridDisplay(int rows, int cols)
  This constructor will take two integers which will set up the rows and columns for the grid. Each grid position will initially display the space character and will be WHITE.

- public void setChar (int row, int col, char c)
  This method will display the character given in the third parameter as the grid position specified by the row and column parameter values given. This methods is not being used for this project (so you can ignore it).

- public void setColor (int row, int col, Color c)
  This method will display the color given in the third parameter as the grid position specified by the row and column parameter values given.

The file GridDisplay class also contains the following static method which will cause the code to sleep for the number of milliseconds specified by the parameter. This method can allow the changes made to the GridDisplay grid to appear as animation. This may needed in case the screen gets updated too fast so that no one can see which changes occurred when. Note if things

need to be slowed down, do the sleep after one generation has been displayed and before the next generation is determined.

- public static void mySleep(int milliseconds)

Note that the GridDisplay class is to only be used the output for your program. See the sample code in GD2.java for examples on how to use the GridDisplay class.

Internal Storage Structure

Your program MUST use a dynamic 2-D array to create a back-end storage structure for the generation grids. A 2-D array in Java is created by creating an array of arrays. This is described in:

http://www.willamette.edu/~gorr/classes/cs231/lectures/chapter9/arrays2d.htm

There are many other places to find out about 2-D arrays (however, the Head First Java text did decide to only briefly discuss them).

Program Submission

You are to submit the programs for this lab via the Assignments Page in Blackboard. To help the TA, name your file with your net-id and the assignment name, like:

- ptroy1LabX.c