Assignment 8 - Game of Life

Description

For this assignment you will create a world teeming with little organisms. The world will be constructed as a two-dimensional array with **M** rows and **N** columns. Each **M** by **N** cell is either (a) occupied by an organism (represented by **X**) or (b) vacant.

Every cell has eight (8) neighbors (top, bottom, left, right, four (4) diagonals) **except** for those cells along the boundary (the edge of the world).

Initially, there is a given population of organisms in the world, but with each successive generation, organisms may reproduce or die based on the following criteria:

- Each organism of the current generation survives to the next generation if, and only if, it has two (2) or three (3) neighbors (a neighbor is an organism that lives in a neighboring cell).
- Organisms with less than two (2) neighbors die from loneliness whereas organisms with more than three (3) neighbors die from overcrowding. These dead organisms leave behind vacant cells in the next generation.
- Each vacant cell in the current generation becomes occupied by a new organism in the next generation if, and only if, it has exactly three (3) neighbors. Otherwise, it remains vacant in the next generation.

For example, given the following population for the zeroth generation:

```
      X X
XXX         XX     XXXX
      X X   XX
```

The next generation would be:

```
 X                 XX
 X           XX    XX
 X           XX    XX
```

The generation after that would be:

```
                   XX
XXX          XX   X  X
             XX    XX
```

Note: The input files to be provided will represent vacant spaces with dots.

Your assignment is write a program to play this Game Of Life. Your program should read the initial world from a file (see below) and then prompt the user to generate a new generation or terminate the program. The program should terminate automatically if the world becomes entirely vacant OR if successive generations will not yield any changes to the current generation. These instances will occur with **life3.dat** and **life4.dat**, respectively.

Design and Implementation

Use two (2) two-dimensional arrays of type **char** to store the old and new generations. The new generation should be created based on information from the old generation. If you use only one (1) array, you will find that your old generation is overwritten by the new one before you have finished using all the information you need from it.

The .dat files provided have worlds of size 25x75. To keep things simple, assume that **M=25** and **N=75** (i.e., the world has 25 rows and 75 columns), and define these in a **final** statement before declaring your array variables.

Data files are attached so you may test your program with different initial worlds, but you should experiment with your own worlds as well! Your program should initially prompt the user to enter a filename (use a **String** variable) similar to the below:

```
Enter a filename: file0.dat
```

Watch your edge cases (literally). Remember that every cell has eight neighbors except for those along the edge of the world. What's the most efficient way to handle these border cells? How could you treat these border cells as if they **also** had eight (8) neighbors?

HINT! (if you want it)

> You should create a "border" of cells that always stay empty by declaring your arrays from 0 to M+1 and from 0 to N+1. This way, the cells on the edge will also have eight neighboring cells and won't need special treatment.

Read the data and assign it to a two-dimensional array using two (2) nested **for loops**. The outer loop reads a string consisting of the data on a given line of the input file. The inner loop assigns the data to a given row of the array. In order to read the file, you will have to use Scanner. Please refer to the attached source code for an example of how this might work.

Include at least **two (2) non-void methods**:

(i) One should take a world and the coordinates of a cell and return the number of neighbors (organisms in neighboring cells) for that the cell.

(ii) The other should take a generation array and return a boolean value that tells whether or not the world represented by the array is empty.

Print all generations to the screen and **include the generation number as well**. The user should be prompted upon each successive generation to either generate a new generation or terminate the program. Make sure you check for invalid input here.

Formatting and Style

Each world should print neatly to the screen so you can clearly see at a glance which cells are neighboring each other and thus, which organisms should live and die based on the above- referenced criteria. Make sure you're adhering to proper style guidelines with respect to indentation for nested for-loops and your control flow statements.
Short and concise method/variable names are always preferred for ease of reference.

Grading Rubric

f

Points	Criteria
2	Properly implements file reader via Scanner
3	Implements at least two (2) non-void methods
3	Iterates each generation properly; prints each generation to screen with generation number
1	Catches invalid input at menu
1	Good style demonstrated in the code; sensible formatting