1 The game

You will write a program to play Minesweeper. It is best learned by playing it, for which there are many web-based and easily installable versions of the game (there is even a version that you can play on remus/romulus called KMines). However, its rules are rather simple:

1. On an $n \times n$ grid of cells\(^1\) some number of mines are randomly placed.
2. All cells are initially covered (or hidden).
3. The player chooses one cell at a time either to reveal or flag.
   - If a cell containing a mine is revealed, the player immediately loses the game.
   - If a cell not containing a mine is revealed, then that cell shows the number of mines contained in the immediately adjacent cells—that is, the eight cells surrounding that one, known as its neighbors.
   - If a cell not containing a mine is revealed, and if that cell has zero neighbors containing mines, then all of the neighbors are recursively revealed as well.
   - If a cell is flagged, then the user is asserting that the cell contains a mine, and thus the flag prevents the cell from being revealed.
   - If a selected cell is already flagged, selecting it to be flagged again removes the flag. That is, flagging a cell toggles the presence of the flag on that cell.
4. If the user reveals all of the unmined cells and flags all of the mined ones, then the player wins.

2 Your assignment

2.1 Getting started

Open a terminal, create a directory for this project, change into it, and grab some starting source code:

```
$ mkdir project3
$ cd project3
$ wget -nv -i https://goo.gl/2rvhqe
$ emacs Cell.java &
$ emacs Minesweeper.java &
```

\(^1\)The grid doesn’t need to be a square, but for simplicity, ours will be.
You should look first at Cell.java. This file contains the Cell class, which defines what a single cell in the grid contains. You will see each Cell object contains its count of live neighbors, and then a few boolean variables to store the cell’s state: mined or not; visible (revealed) or hidden; flagged or not. You will then see a number of methods for setting and accessing this state about each cell.

Then, examine Minesweeper.java. It looks like a normal one of our programs in that it is a collection of static methods, including a main() method. It contains a few methods already written...

- **main()**: Does the usual: checks that the program input was valid, calls a method to populate the Minesweeper board, and calls a method to play the game.

- **showGrid()**: Prints a two-dimensional array of Cell objects.

- **getCommand()**: The user interface. Get a command from the user as to what move to make next, ensuring that the input is a valid operation (reveal or flag) on valid grid coordinates.

- A couple of small, obvious helper methods.

... and two methods you must fill in, along with any helper methods you wish to create and use:

- **populate()**: Create the initial grid of Cell objects, randomly placing the mines and setting the live-neighbor counts.

- **play()**: Given a grid, obtain the user’s move from getCommand() and carry that out until the user loses (reveals a mine) or wins (reveals all non-mines and flags all mines).

### 2.2 Compiling and running the program

To compile your program, you need to compile all of your .java files. You can do this in a single command as follows:

```
$ javac *.java
```

The * indicates that you are referring to all files that fit this pattern; in this cases, all files that end in the extension .java.

To run the program, you then type the command:

```
$ java Minesweeper 4 5
```

Here the first input parameter to the program indicates the number of mines that you want to place, and the second input parameter indicates the size of the grid.
3 Submitting your work

Submit your Minesweeper.java and Cell.java source code files with the CS submission system, using one of the two methods:

- **Web-based:** Visit the submission system web page.
- **Command-line based:** Use the cssubmit command at your shell prompt.

This assignment is due on Wednesday, December 13, 11:59 pm.