Project One — Minesweeper

January 16, 2019

The project involves implementing a version of the Minesweeper game. *Your solution must implement the behavior described in this handout even if it differs from your understanding of how the game is normally played.*

1 Classes

1.1 Gameboard

Your `Gameboard` class will have a two-dimensional array (an 8-by-8 gird) representing the board containing the mines. The `Gameboard` class will contain a public `run()` method that conducts the playing of the game. Your implementation will be character-based. (See section 2 below.)

1.2 Cell

Each square on the game board will be represented by an instance of a class named `Cell`; i.e. the grid is a 2-D array of objects of class `Cell`. In other words, the `Cell` class needs a field that is a `String` or a `char` to represent the character in that cell of the grid. The `Cell` class may need additional fields to facilitate the playing of the a game. Each cell will display one of the following characters

- a hyphen. A hyphen means that a guess for that cell has not yet been made.
- a digit. A digit means that a guess has been made that the cell does not contain a mine and the guess was correct. In this case, the digit represents the number adjacent cells that contain a mine. By adjacent I mean the up to eight cells that are touching the cell in question. The eight cells include the four cells that are directly north, south, east, and west, as well as the four corner cells that are northeast, southeast, southwest, and northwest. Note the digit is a character, not an `int`.
- the character M. An M means that a guess has been made that the cell does contain a mine and the guess was correct. In this case, the M indicates that the cell contains a mine.

1.3 Minesweeper

The `Minesweeper` class contains the `main()` method. Your `main()` should create an instance of the `Gameboard` class and call it's `run()` method. All other program logic will be implemented in the `Gameboard` class.

2 Game Behavior

The initial guessed grid is shown below

```
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
```

Initially no guesses have been made, so all the cells have hyphens in them as shown above.

The game prompts the user to pick a particular cell (by specifying the row and column) and then to guess whether that cell contains a mine.

- If the guess is correct and was that the cell contains a mine, then the grid entry for that cell changes to the character M and the game prompts for a new guess of the content of a cell.
- If the guess is correct and was that the cell does not contain a mine, then the grid entry for that cell changes to a digit and the game prompts for a new guess of the content of a cell. The digit indicates how many adjacent cells contain mines. By "adjacent" I include the squares whose corners touch the current square. For example, a square in the middle of the grid has eight adjacent squares (three above, two on its sides, and three below).
- If the guess is not correct, then the game is over and the user loses. If you guess that the cell has a mine and it does not, then the guess is incorrect. If you guess that the cell does not have a mine and it does, then the guess is incorrect.

The user wins the game if the user can correctly guess every cell. The first incorrect guess causes the game to end with the user losing.

After each guess, the game redisplays the grid with the guessed cell content made visible. An example is below after six guesses have been made (five correct guesses of no mine at a cell and one correct guess of a mine at a cell).

The guessed grid for the example is shown below

```
0 - - 1 M - - -
- - - - - - - -
- - - - - - 2 -
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - 2 - 0 - - -
- - - - - - - -
```

Your program must also give the user the option to peek and see the location of all the mines. Allowing peeking is needed for you and me to determine whether your game is working correctly. Below is an example output if the user said "yes" to the peek prompt for the grid immediately above.

The locations of all the mines are shown

```
0 - - 1 M - - -
- - - - - - - M
- M M - - - 2 -
- - - - - - M -
M - - - - - - -
- - M - - - - -
M M 2 - 0 - - -
- - - - - - - M
```

Comments

Here are a few other comments.

1. Your program must use ten mines. Don't use a magic number; use a named constant.
2. Your program must place the mines randomly at the start of the game using the `java.util.Random` class.
3. Your solution needs to allow the user to play multiple games with a prompt in-between. Your solution needs to keep prompting the user to enter "yes" or "no" to the prompt of whether to play another game.
4. Your solution needs to keep prompting the user to enter "yes" or "no" to the prompt of whether a particular square holds a mine or not.
5. Your solution needs to keep prompting the user to enter a valid row number and a valid column number to the prompts of the row and column of the next square being guessed.
6. Suggestion: To simplify the coding of checking adjacent squares when along the boundary, consider creating a 10-by-10 array which has the mines 8-by-8 grid inside. Initialize the outermost rows and columns in the 10-by-10 array to have zero mines. This way even grid square [0][0] has eight adjacent array elements.

3 Random Mine Placement

The `java.util.Random` class has a `nextInt` method as in

```
Random generator = new Random();
int number = generator.nextInt(27);
```

which causes `number` to have an integer between 0 and 27 including 0 and excluding 27 (so including 0 through 26).

Placing the mines randomly is a bit tricky. Placing the first mine is easy. It randomly goes in any of the 64 cells. The problem starts with placement of the second mine. It should randomly be placed in any of the 63 (not 64) cells that are unoccupied. In particularly it should not be placed in the cell that already contains a mine. Similarly, the third mine needs to be placed in any of the 62 (not 63 or 64) cells that are unoccupied, and so on.

The easiest way to correctly randomly place the mines is, for each mine, to randomly generate a pair of numbers each between 0 and 7. See if that position is occupied by a mine. If it is not

occupied by a mine, then place the new mine there. If it is occupied by a mine, then loop back and again randomly generate another pair of numbers between 0 and 7 and check again. Continue until the position chosen is not occupied.

4 Submission

1. The project is due by 11:59 pm on Friday, February 1.
2. The project is worth 50 points. Late submissions will be accepted with three points off per day late (not counting weekends and holidays) up to a maximum of 24 late points. So from 11:55 pm on Friday to 11:55 pm on the next Monday is one day late, and so on. The last possible day to submit a project will be when the solution is posted on the course website.
3. If your program does not compile, the score is zero.
4. Zip your source files in a folder and submit to Moodle in the usual fashion.
5. You may talk with other students in the class about the concepts involved in doing the project, but anything involving actual code needs to be your own. In other words, you can not show your code to any other student and you can not look at the code of another student.
6. Your solution must follow the course coding conventions as posted on the course website.