Lab #4 – MineSweeper

Total Points: 50

**Introduction**

The Object-Oriented Language Development Company (OLD Co.) would like to hire you to do some game development for them. For this project they need you to write a critical class that is necessary to complete their new, and previously unheard of, game called MineSweeper. The figure below shows a screen shot of their MineSweeper game:



The MineSweeper game is played on a grid of cells, each of which potentially contains a mine. The objective of the game is to uncover all of the cells which do not contain mines and to place a flag on all of the cells which do contain mines.

Initially all of the cells on the board are covered and appear to be raised. Left clicking on a cell with the mouse will uncover a cell. If the uncovered cell contains a mine, the game is over. If the uncovered cell does not contain a mine, the cell will either become blank or will be filled with a number. The appearance of a number in an uncovered cell indicates the number of mines that are contained in adjacent cells. In MineSweeper, “adjacent” cells can be horizontal, vertical, or diagonal neighbors. For example, in the figure above there is a mine in exactly one cell adjacent to each cell containing a 1. Similarly, there are exactly two mines adjacent to the cell which contains a 2.

When a player determines that a cell contains a mine, the player can place a flag on that cell by right clicking the mouse on the cell. In the figure above a flag has been placed on one cell which is thought to contain a mine. Once a flag is placed on a cell, that cell cannot be uncovered unless the flag is first removed by right clicking on the cell again. This protects the player from accidentally uncovering a cell which is thought to contain a mine.

The game ends when the player either: (i) loses by uncovering a mine, or (ii) wins by placing a flag on all cells which contain a mine and uncovering all of the remaining cells. The difficulty of the game can be changed by selecting a different level of play from the "Level" menu. The higher the level of play, the larger the board becomes and the more mines it contains.

**Getting Started**

Open the Lab 4 starter code in Eclipse, by following the instructions on the course How-to page as usual.

Expand the src folder of the Lab-4 project, and open the package named MineSweeper.

Run the MineSweeper program contained in this package (i.e. the main method in the MineSweeper class.)

Running the MineSweeper program will display a window that looks like this:



Note however that the MineSweeper game does not work. This is because as mentioned in the introduction, the implementation of a critical class is missing!

Remember, you can and should, push your changes to GitHub frequently, and definitely at the end of each work session before you log out.

**Design**

You will be responsible for implementing the MineSweeperBoard class. This class lies at the heart of the game. It is responsible for keeping track of the contents of the board as the game is being played including the locations of mines, which cells have been uncovered and which cells have been flagged. The MineSweeperBoard class tracks the board contents by maintaining a two-dimensional array of int values. The value in each cell of the array is a representation of what appears at each cell on the playing board. The following table gives a list of the int values (and the names of associated class constants that have been defined in MineSweeperBoard) that may appear in the array:

|  |  |  |
| --- | --- | --- |
| **Cell Value** | **Constant** | **Meaning** |
| -1 | COVERED\_CELL | An array entry containing the value COVERED\_CELL indicates an empty cell on the board that has not yet been uncovered by the player. |
| -2 | MINE | An array entry containing the value MINE indicates a cell containing a mine which has neither been uncovered by the player nor has it had a flag placed on it. |
| -3 | FLAG | An array entry containing the value FLAG indicates a cell that does not contain a mine, but which has had a flag (incorrectly) placed on it. |
| -4 | FLAGGED\_MINE | An array entry containing the value FLAGGED\_MINE indicates a cell that does contain a mine and has had a flag (correctly) placed on it. |
| -5 | UNCOVERED\_MINE | An array entry containing the value UNCOVERED\_MINE indicates a cell containing a mine which has been uncovered by the player. |
| 0-8 | n/a | An array entry containing a value between 0 and 8 indicates an empty cell which has been uncovered. The specific value between 0 and 8 indicates the number of mines which appear in the adjacent cells. |

To help clarify the meaning of these values consider the following example. When a new MineSweeperBoard is created it will contain only COVERED\_CELLs (-1) and MINEs (-2). The MINEs will be placed randomly in the array. For example, if the new array has 3 rows and 4 columns it might look as follows:


*A newly created array representing a board with no cells
uncovered and mines at location (0,0) and (2,1).*

If the player were to left click on the upper right-hand cell of the board then the MineSweeperBoard class would change cell (0,3) of its array to a zero as shown below:


*The array representing the board after the player left-clicked
on the cell in the upper-right corner.*

Cell (0,3) was changed to a zero reflecting the fact that the cell has been uncovered and that there are no mines adjacent to this cell. When the Mine Sweeper game sees a 0 in this location it will display a blank uncovered cell in the upper-right corner.

If the player were now to click in the center cell in the left-most column, the MineSweeperBoard would change cell (1,0) in its array to a 2, indicating that there are two mines adjacent to that cell. When the Mine Sweeper game see a 2 in this location it will display an uncovered cell containing the number 2. The array representing the board would now appear as follows:


*The array representing the board after the player left-clicked
on the cell in the center of the left-most column.*

If the player were now to right click on the cell in the lower-left corner of the board to place a flag on that cell, the MineSweeperBoard would change cell (2,0) of its array to FLAG. Recall that the value FLAG indicates that a flag has been (incorrectly) placed on a cell which does not contain a mine. The array representing the board would now appear as shown below:


*The array representing the board after the player right-clicked
on the cell in the lower-left corner.*

If the player now right clicks on the cell in the upper-left corner of the board, the MineSweeperBoard would change cell (0,0) of its array to a FLAGGED\_MINE. The value FLAGGED\_MINE indicates that a flag has been (correctly) placed on a cell containing a mine. The array representing the board would now appear as shown below:


*The array representing the board after the player right-clicked
on the cell in the upper-left corner.*

Imagine now that player, having decided that the cell in the lower-left corner of the board should not be flagged, right-clicks again on the lower-left corner cell. The MineSweeperBoard must now change cell (2,0) of its array back to COVERED\_CELL. Similarly, if the player were to decide to remove the flag from the upper-left corner, the MineSweeperBoard class would have to change cell (0,0) of its array back to a MINE. Finally, if after removing the flag from the upper-left corner of the board, the poor unsuspecting player left clicks that cell, the MineSweeperBoard will change cell (0,0) to an UNCOVERED\_MINE, causing the game to end.

**The Assignment**

Your assignment for this lab is to **complete** the implementation of the MineSweeperBoard class and to **test** your implementation. The steps outlined below give the recommended process for completing these tasks.

* 1. Add a field definition for the two-dimensional array that will hold the state of the board as described above.
	2. Complete the implementation of the first constructor (the one with no parameters). This constructor will build a board that has mines in known locations. Having a board for which you know where the mines are located will help when you test the other methods in the MineSweeperBoard class.
	3. Skip the implementation of the second constructor (the one with one parameter). It is best to come back to it at the end.
	4. Complete the getRows(), getColumns(), getNumMines() and getCell() methods.
	5. Create a JUnit Test Case for the MineSweeperBoard class.
	6. Create a test method for the first constructor that checks that it created a board of the correct size, with the correct number of mines and that the mines are placed in the correct locations.
	7. Run the MineSweeper program. The board should appear with the size defined in your constructor. You will not yet be able to play the game, but as you implement each of the following methods you will be able to run the game and see the effects of the code that you have added. Be sure to run the game after you write and test each method to see its effect.
	8. Continue implementing and testing each of the methods in the MineSweeperBoard class. The best way to do this is to implement one method and then test it, ensuring that it is correct before going onto the next method. Note: you may need/want to add additional fields to your class to help with the implementation of some of the other methods. Be sure to test each method thoroughly.
	9. When all of the methods in the MineSweeperBoard class have been implemented and tested, return to the implementation of the second constructor. In order to implement the second constructor, you will need to be able to generate random numbers as the locations for the mines. The java.util.Random class provides a way to generate random integers in a specified range. Consider the following snippet of code:

Random rnd = new Random();

int x = rnd.nextInt(10); // x is a random integer from 0 to 9.

int y = rnd.nextInt(57); // y is a random integer from 0 to 56.

Thus, you can create a java.util.Random object in your constructor and use the nextInt method to generate a random row and column number for each mine. Note that you will need to import java.util.Random.

* 1. Create a test method for the second constructor. This test should ensure that the board is the correct size and contains the correct number of mines. Since the mines are placed randomly it is not possible to check if they are placed in the "correct" locations.
	2. Play the game and have fun!

**Submitting your solution**

As usual, push your code to GitHub regularly for backup purposes and push your final version to submit the assignment. In addition, as usual, submit your lab report via Moodle. The lab report for this lab will consist only of the self-assessment report.