Minesweeper Project (200 Points)

Michael Efram
Healdsburg High School

Introduction

This is the assignment for Minesweeper. You have a Java Class named **Minesweeper** and a Java Application named **MinesweeperApplication** that you should add to an IntelliJ Project. As you work through this project, you will complete an interface and concrete classes. Your solution will demonstrate graphics, the use of 2D arrays, and finally recursion (challenge). Prior to beginning this project, you should play the game of Minesweeper to see how the game is played.

Overview

The project utilizes a Java Frame to implement the game. You should notice that the project extends `Frame` and implements `MouseListener` and `WindowListener`.

Step 1: Run the program.

You should notice that when you click on a square on the grid that it changes color. This is done implementing the `MouseListener` interface and in particular, utilizing the `MouseClicked` method. When the user clicks on a square in the grid, the program code checks for the particular row and column and saves it to the fields, `rowClicked` and `colClicked`. Look through the existing program code to see how this has been done.

Step 2: Change the number of rows and columns and adjust the size of each cell.

Notice that in the fields there are constants set up for the number of cells in the row/column, the overall width and height of the game, and the (calculated) value of the cell width/height. The game board then can change with the simple change of `SIZE`. Change size to 9 so that it is like the “beginner” version of the original Minesweeper game. You may experiment with other sizes if you like and this can be changed dynamically (done later in this assignment).

Step 3: Create Parallel 2D Array for placement of Mines.

Also in the fields, there is a 2D array of `Rectangle` objects, `theGrid`, that is used for the drawing the grid in the game. You will be using the idea of “parallel” 2D arrays in this project and creating several “layers”. For this layer, create and initialize a 2D array of `int` values. You may call it `theMines`. Please see how `theGrid` was created and initialized for help.

Step 4: Randomly place mines in 2D Array.

The "beginner" level of Minesweeper has 10 mines. Other levels have more. Create a constant field that will hold the number of mines (just like SIZE held the number of rows/columns). In the `Constructor` method, randomly place mines in the 2D array `theMines`.

Helpful Hint: use a **while** loop (i.e. while(count < MINES)) to place mines in a random row/column. A mine can be the `int` value of -1. You must check if the row/column already has a mine and do not count it if it does

Step 5: Calculate Values for all other cells in Mine Array.

After you place the mines, you need to calculate the values for the other cells. In the game of Minesweeper, the value is the number of mines adjacent to the cell.

Helpful Hint: Traverse the 2D array `theMines` and for each cell that is NOT a mine, count how many adjacent cells ARE mines (contain value of -1). Careful about not checking cells that are out-of-bounds (causing an IndexOutOfBounds exception).

Step 6: Test your program to verify all cells have correct value.

Right now, your program might run fine or appear to run fine, but it may not be correct. Temporarily add the following line of code into the nested for loop in the `paint`() method.

```
g.drawString("" + theMines[row][col],
             theGrid[row][col].x + CELL_WIDTH/2,
             theGrid[row][col].y + CELL_HEIGHT/2);
```

This should display the values for each cell in the 2D array `theMines`. You can visually check to make sure that you have correctly calculated the cell values. Of course, you will remove or comment out this line of code after you have checked for correctness.

Step 7: Create Parallel 2D Array for Display.

In the game of Minesweeper, as the player clicks on cells, the value of each cell is revealed or a bomb is displayed and the game ends (don't worry about the automatic clearing right now, that will be done recursively).

So, either the cell value is revealed or it isn't. This sounds like a great place to use the `boolean` data type. Create and initialize a 2D array of `boolean` values (parallel to the other 2D arrays) and initialize each value to `false`. You can call this 2D array `showMe`.

Step 8: Enable display of each cell that has been clicked.

Right now, the `mouseClicked()` method sets one row/col to be displayed. You need to replace that program code to set the `boolean` value in the 2D array `showMe` to `true` if the player clicks on the cell.

In the nested for loop in the `paint()` method, if `showMe[row][col]` is `true`, you need to display the `int` value in `theMines` (you did this above when you tested your mine placement above). If you would like it to look nicer, you will have to use `g.fillRect(…)` with different colors depending on the value of `showMe[row][col]`. (use `setColor(…)` to do this)

Here is how to use `fillRect` so that you do not overwrite the rectangle boundaries:

```
g.fillRect(theGrid[row][col].x+1,
           theGrid[row][col].y+1,
           theGrid[row][col].width-2,
           theGrid[row][col].height-2);
```

Remove the program code that filled the Rectangle where the user clicked.

Add the program code to end the game if the player clicks on a mine. One way to do this is to make a `boolean` field named `gameOver` and do not allow the player to do anything if `gameOver` is `true`.

Challenge: Allow for recursive clearing of cells that are not touching a mine.

1) Replace the line of program code in `mouseClicked()` that sets `showMe[row][col]` with a call to a private helper method, called `clearCells(…)` passing the row and column as arguments.

2) Create a private method with the following header:
   ```
   private void clearCells(int row, int col)
   ```

3) In the method `clearCells`:

```
if the row/column is in-bounds and showMe[row][col] is false
    set showMe[row][col] to true
    if theMines[row][col] == 0
        recursively call clearCells for each adjacent cell
        (hint: 8 calls to clearCells)
```

Run your program and watch the power of recursion!! Pretty cool that all of that can be done with so very few lines of program code ☺

Extra Credit: Allow the user to dynamically change the dimensions of the game and how many mines and/or allow for right-clicking to clear mines.

No instructions provided.