CS1 – Practical Session 6: An exam question

Kurt Driessens

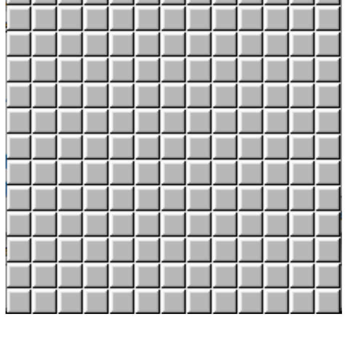
September 25th, 2018

In this 6th practical session, you will be introduced to the level of complexity you can expect at the exam. Given that this means that the complexity of these assignments is a bit higher than those from previous labs, it might be better to first finish previous labs to get the training needed for solving the assignment below.
There are 3 assignments, all in the context of a simple but popular computer game. The third assignment might be too hard at the moment, as it really is a method that should be implemented "recursively". It might be easier (and I actually recommend) to wait until the lab session on October 1st to solve that part.
On EleUM is a set of classes that will allow you to implement and test your methods as part of a game that you can actually play. MineSweeper.java holds a skeleton for your implementation, including a main method to test it. Simply uncomment the part of the main method that you want to test, compile and enjoy.

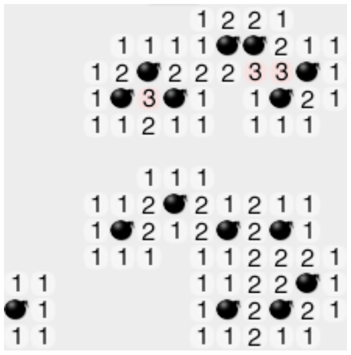
Don't forget to upload your java-file to get credit for this assignment! The deadline is set for the end of week 5 to give you sufficient time. You DON'T have to wait until then to submit.

Context:
The game "Minesweeper" is a single-player video game. The object of the game is to clear an abstract minefield without detonating a mine. The game has been written for many platforms in use today, including the Minesweeper for the Windows platform, which has come bundled with versions of the operating system from 3.1 and during quite a number of versions after.

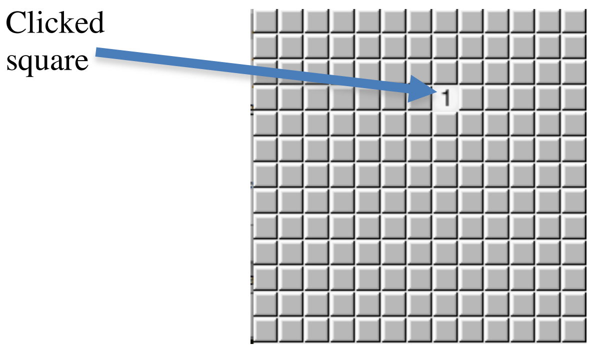
The game board consists of a number of squares under which bombs are hidden at random locations.



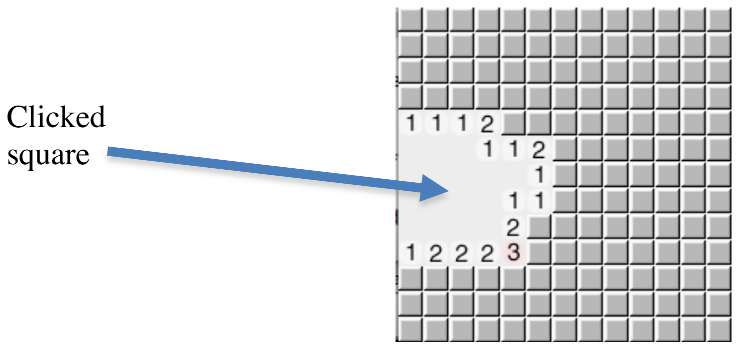
The goal of the game is to open all squares under which there are no bombs hidden. Suspect bomb locations can be marked. The figure below shows a finished game.



The game is played by left clicking squares under which the player expects no bombs. The game helps the player locate the bombs with hints that appear on clicked squares. The number that appears as a hint is the number of bombs located in the 8 squares surrounding the square clicked. The figure below shows the game board after clicking one square on which no bomb is located, but where there is a bomb located on one of the 8 adjacent squares.



When the player clicks on a square without a bomb, but which also has no surrounding squares with bombs located on them, the game opens an area that extends to squares that are bomb adjacent. The figure below shows the game board after such an initial click.



Your task is to implement 3 methods used in this game.

1. Write a method that takes the width and height of the board as parameters and returns a two-dimensional integer array that represents the gameboard and holds 10 bombs at random locations. All empty squares should contain the value 0, while squares that do hold a bomb contain the value 9. Pay attention to the random placement of bombs. All squares should have the same probability to get a bomb assigned to them, no square can contain more than 1 bomb and no more and no less than 10 bombs must be placed on the board.

```
public static int[][] makeBoard(int height, int width) {

        ...

}
```

2. Write a method that computes the hints (i.e. the number surrounding bombs for each square) given a game board initialized as asked in the previous method. This means that all zero values in the two dimensional array have to be replaced by the count of "9" values surrounding them.

For example, if the array coming into this method looks as follows:

0	9	0	0	0
0	0	0	9	9
0	0	0	0	0
9	0	0	0	0
0	0	0	0	9

The method should change the array into the following:

1	9	2	2	2
1	1	2	9	9
1	1	1	2	2
9	1	0	1	1
1	1	0	1	9

```
public static void computeHints(int[][] board) {

        ...

}
```

3. (You can wait with this until after the recursion lecture.) Write a method that processes the player's click. The method takes two integers and two 2-dimensional arrays as parameters. The first two parameters are the coordinates of the clicked square. The third parameter, i.e. the integer matrix, contains the hints as they are computed by the previous method. Squares holding a bomb still hold the value "9". The fourth parameter is a matrix of booleans of the same size as the gameboard, indicating which cells have already been opened by previous clicks.

If the square was already open, the method should return true and do nothing else. If the clicked square contains a bomb, the method should return false as the game is over. Otherwise the method should return true.
As a side effect all the squares that are opened by the click – be it just the square clicked on or the area surrounding it if the square itself has no neighboring bombs (See the figures on page 2) – should get the value true in the boolean matrix.

It is recommended (but not required) to implement this recursively.

```
public static boolean clicked(int x, int y, int[][] board, boolean[][] open) {

        ...

}
```