

ITRW222

PRACTICAL 3 MINESWEEPER

At least 4 hours of work!! Please start today!

We are ready now to create the data structure for the game. You will create new class for the Cell in the minesweeper game. Since this practical follows the previous one, and there are another four to come, you need to strictly use the variable names I present below. YOU WILL LOOSE MARKS IF YOU USE DIFFERENT DEFINITIONS O DO NOT FOLLOW THE INSTRUCTIONS.

The most important aspect is that you have a deep understanding of the class structure. Note the following:

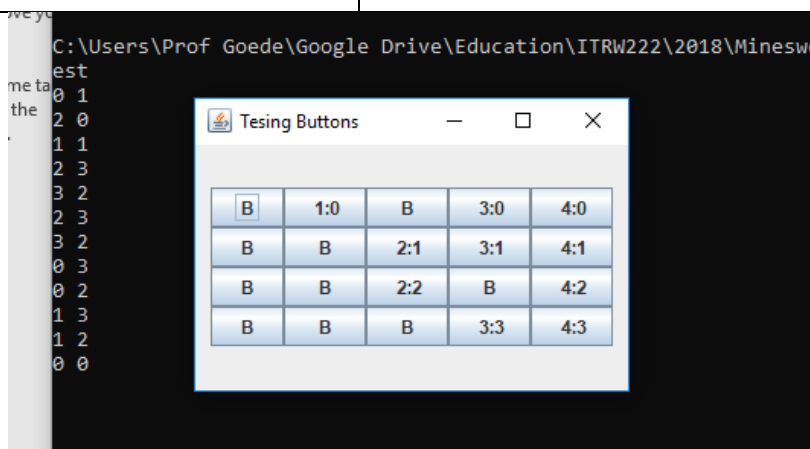
- Each square on the game is actually 2 things. It is the GUI component you programmed in Practical 1 and 2. BUT it is also an object of the Cell class that stores the game information.
- Each cell can either be a bomb or not a bomb.
- If it is not a bomb it should store a number equal to the number of bombs in the 3x3 neighbourhood of the specific cell. If you do not understand this, you need to play a downloaded minesweeper game more often! You will implement this in PRAC 4!
- The game player controls the status of a cell:
 - It can be revealed or not – all cells start as not-revealed. The user will reveal a cell with a left mouse click.
 - It can be flagged as a bomb. The user flags a cell with a right click
 - This button input processing comes in PRAC 5 – So relax for now!
- The game stops:
 - When a user revealed a bomb.
 - When all the cells are correctly flagged or revealed. (This is hard to program – think about it in the meantime)
 - This comes in PRAC 6!

In practical 3 you need to do the following:

1. Create the Cell class to store all the information of the cell explained above, using the class definitions in the table
2. Create methods for your GUI class to
 - a. Create your field[][] to hold the cell objects. The method called: `public void initialiseField()` should create an object of the Cell class for every entry of the *field* 2D-array.
 - b. `public void displayField()`. The method link your data 2D-array with your GUI 2D-array. The coordinates are linked – `grid[2][2]` should display `field[2][2]`. In this practical display a 'B' if a cell is a bomb (see below) or else the position of the cell as you did in practical 2.
 - c. `PlantBombs()`. We use a random number generator to set the Boolean variable in SOME OF the MScells to true – thereby planting the bombs.
 - `bombRow = randomNumbers.nextInt(ROWS);`
//produce random number from 0-ROWS-1
 - `bombCol = randomNumbers.nextInt(COLS);`

- It is possible to still use command screen output in the background of your GUI, so you can add: `System.out.println(bombRow + " " + bombCol);` to see if it is working!
- Two basic rules – you need to end up with the correct number of BOMBS – specified at the top of your program! You cannot plant a bomb on bomb!

CLASS 1 – The Cell:	CLASS 2 - The Minefield: THESE ARE EXTENTIONS TO YOUR GUI FROM PRACTICAL 2
<pre> public class MScell { private boolean revealed; private boolean bomb; private int value; private boolean flagged; // constructor to initialise //variables public MScell () // accessor and mutators public void setRevealed () public boolean isRevealed() public void setFlagged () public boolean isFlagged() public void setBomb() public boolean isBomb() public void setValue(int v) public int getValue() public String toString() // mark BOMB! /*This method is crucial! Here we overwrite the java method – Think carefully what each cell should display – it involves a few if-statements*/ } </pre>	<pre> public class MineField { private MScell field[][]; private final int BOMBS; // constructor already in practical 1-2 public void initialiseField() public void plantBombs() // Plant BOMBS number of Random bombs public void displayField() /* you need to change the text on your buttons/labels to work according to the rules of the game, */ } </pre> <p>NOTE: The size of the declared array should come from the ROWS and COLS values you made earlier and now you need to add another variable for the number of BOMBS.</p>



ENJOY! THE FUN STARTS NOW!