CP1300 SP51 2015 Assignment Details

Task:

You are to plan and then code the “True/False Game” as described by the information provided below. The assignment is worth 30% of your overall grade for CP1300. You are expected to work the assignment based on the following coding requirements and expected gameplay features.

Gameplay Features:

The “True/False game” is played on a 2D board of cells. The content of each cell is initially unknown by the player. The game is played by revealing the contents of each cell. A cell might be empty or contain a single flag. There are two types of flag: a true-flag, and a false-flag. A true-flag is a “treasure” the player wants to find, whereas a false-flag must never be revealed. Finding a false-flag causes the player to immediately lose the game. The player only wins the game if they reveal all cells except those that contain false-flags.

When a new game starts the board is initialised with a number of randomly placed flags. The number of flags depends on the difficulty level and the size of the board. So after the board is setup, some cells will be empty and some will contain a flag. There are 3 difficulty levels: “easy”, “medium”, and “hard”. The board size is determined by a row value between 4 - 10 inclusive and a column value between 4 - 10 inclusive. The number of flags assigned to the cells of a new board is decided as follows:

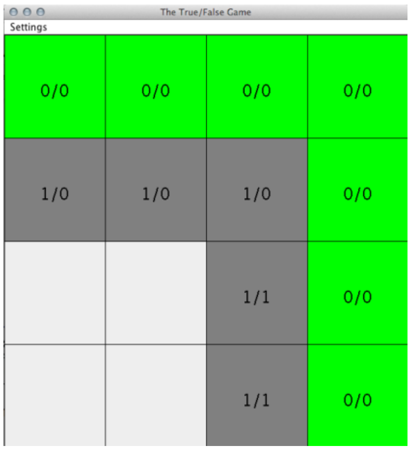
Difficulty level	**Percentage of false-flags**	**Percentage of true-flags**
Easy	5% of the board	1% of the board
Medium	10% of the board	5% of the board
Hard	20% of the board	10% of the board

Note: changing the size of the board does not change the difficulty level, and a new board must have at least 1 false-flag and 1 true-flag after it has been setup for gameplay.

During gameplay, the player is free to click on any cell on the board to reveal its content. Clicking on a cell that has already been revealed has no effect. If the player clicks on a cell that contains a true-flag then that flag is revealed (i.e. the player found a treasure). If the player clicks on an empty cell then all nearby cells that are also empty are revealed (we expect you to use a *recursive method* to implement this behaviour). Note: if the game is already over and the player clicks on the board then a popup message should be displayed to inform the player the game is over.

The player can change the size of the board at any time. When this happens, the game is restarted with a new board of randomly placed flags. Moreover, when the program is started the board should have 10x10 cells and the difficulty level should be “easy”.

The **Field** class is used to represent the board. The **Cell** class is used to represent the cells in the board. The **GameDisplay** class is a resizable custom **JPanel** that displays the contents of the **Cell** objects currently on the **Field**. If a **Cell** object contains a flag then **GameDisplay** displays “T” or “F” for that cell depending on which flag the **Cell** object contains. If the **Cell** object does not contain a flag then **GameDisplay** will display a text string for that cell of the form “digit / digit”. The first digit is the number of neighbouring **Cell** objects that contain false-flags. The second digit is the number of neighbouring **Cell** objects that contain true-flags. Let “neighbouring cells” be those that surround a particular cell on the board. Here is a simple example:



This board has 4x4 cells. The green cells are empty and have been revealed by the player. The light-grey cells have not yet had their content revealed. The dark-grey cells have been revealed and are empty but they have neighbouring cells that contain flags. Cells that display “1/0” have one neighbouring cell with a false-flag and zero with a true-flag. Cells that display “1/1” have one neighbouring cell with a false-flag and one with a true-flag.

Note: more examples are available in the subject outline and a 10min screencast demo of the finished program.

So in this particular example, it is safe to click on the bottom left cell. Beyond that can you tell which other cells are safe to reveal? You should realise that it is possible for an empty cell to have between 0 to 8 neighbouring cells that contain flags.

Coding Requirements:

Your code must include the following classes:

- **App** - the main executable class. Use it to create and initialise the main program objects, and to control program behaviour.
- **SizeDialog** - a custom **JDialog** subclass that allows the player to change the size of the game board.
- **MainFrame** - a custom **JFrame** subclass that handles general setup, creation of the menu-bar, and the layout of the **GameDisplay**.
- **GameDisplay** - a custom **JPanel** subclass that displays the game board.
- **Field** - a class that models the game board as a 2D primitive array of **Cell** objects.
- **Cell** - a class that represents the contents of a single cell in the game.
- **TestField** - a class used to implement “manual unit testing” for **Field** and **Cell**.

Ensure that your assignment code covers the following list of requirements:

- Write your program using Java code that will work with JDK 1.7.
- Your code should have at least 2 packages to hold the classes.
- The **App** class should be placed in the default package.
- The source code text should have a consistent layout with respect to indentation and whitespace - making it easy to read.
- The code should include comments that help the reader understand the more complex parts of the code.
- A set of “manual unit tests” should be created that demonstrate the stability of the **Field** and **Cell** classes.
- The names of variables in the code should be well-chosen so that they appropriately express the meaning of the values/objects they contain.
- Where appropriate, a class should use helper methods.

- The **Field** and **Cell** classes model the gameplay as defined above. Whereas the **GameDisplay** class only visualises the gameplay.
- The menu-bar added to **MainFrame** should be provided that allows the board size and the difficulty level to be changed.
- The custom dialog class **SizeDialog** should be used to configure the size of the game board.
- Informative popup dialogs should be used to provide information to the player during game-play (don't use custom classes).
- The GUI should allow the player to click on cells to play the game.

Suggestions for working on your assignment:

Looking at a computer screen for hours does not help you solve coding problems. When dealing with a difficult part of the assignment, consider sketching your ideas on a piece of paper. This helps you develop those ideas without the distraction of writing code and thinking about syntax. Consider implementing the **Field** class using a primitive 2D array of **Cell** objects - nested **for-loops** will be useful. Also, nested **for-each-loops** are convenient for interacting with existing **Cell** objects. It's a good idea to pre-compute the count of nearby flags for each cell on the board. Helper methods come in handy for making code easier to understand and develop.

Due:

Submit your assignment by **5:00pm 16th September 2015**. Submissions received after this date will incur late penalties as described in the subject outline.

Submission Requirements:

Submit your assignment by uploading it to the dropbox on LearnJCU for CP1300. Create a zip file of your entire Eclipse project including all Java files. Please name the zip file using your last-name and first-name. For example, if your name is Jack Smith, then the file you submit is Smith_Jack.zip.

Integrity:

The work you submit for this assignment must be your own. You are allowed to discuss the assignment with other students and get assistance from your peers, but you may not do anyone else's work for them and you may not get anyone else to do any part of your work.

When marking your work, if we detect that it is too similar to another student's work then we will follow the university procedures for handling plagiarism. If you require assistance with the assignment, please talk to your lecturer or tutor.