Victoria University of Wellington
School of Engineering and Computer Science

SWEN221: Software Development

Mid-term Test (worth 10% of overall mark)

Minesweeper

In this test you will be working on a program for checking games of *minesweeper*. The program reads in a file which represents a game of *minesweeper* and checks that it follows the rules of the game correctly. An example game file is the following:

```
5,6
2 (1,1) (2,2)
3 E(0,0) F(1,1) E(1,0)
```

The first line gives the width and height of the board in that order. In this case, the board is `5` squares wide and `6` squares deep. The second line begins with the number of bombs, followed by their positions on the board. The final line begins with the number of moves, followed by a string encoding each move. The valid move strings are:

- `E(x,y)` — indicates the player exposed the square at position `(x,y)`
- `W(x,y)` — indicates the player exposed the square(s) at position `(x,y)` and there were no blank squares remaining afterwards. In such case, the player "wins" the game.
- `L(x,y)` — indicates the player exposed a bomb at position `(x,y)` and, hence, lost the game.
- `F(x,y)` — indicates the player flagged the square at position `(x,y)`
- `U(x,y)` — indicates the player unflagged the square at position `(x,y)`

Like arrays in Java, the x- and y-components of a position start from `0`. For example, the top-left position on the board is `(0,0)`. **The rules of the game will be explained in the following pages**.

Download

You can download the code provided for the minesweeper game from here:

`http://ecs.victoria.ac.nz/~djp/files/tt2012.jar`

You will find several Java source files, including a JUnit test file.

Submission

You should submit your solutions through the usual assignment submission system. Please make sure you submit to the correct session. The URL for submission is:

`http://ecs.victoria.ac.nz/cgi-bin/auth/submit?course=SWEN221`

Late submissions will get zero marks (unless you have arranged this with us, which will only be in exceptional circumstances).

PLEASE TURN OVER

1 Debugging the Parser (worth 10%)

You should begin by importing the code provided into Eclipse and running the JUnit tests. You will find that the first five tests fail. You will find information regarding these errors is printed into the Eclipse console, which you may find helpful.

There is one bug in Parser.java, and your aim is to find and correct the mistake. Upon completing this, you should find that tests validFile_1 , . . ., validFile_5 now pass. **NOTE:** you should also find that many or all of the other tests now fail!

2 Invalid Board Positions (worth 15%)

This part concerns the following requirements for the minesweeper game:

1. All bombs should be placed on valid board positions.
2. All moves should be for valid board positions.

You should find that some or all of the tests invalidFile_6 , ..., invalidFile_11 currently fail. This is because Parser.java does not check that a position read from the file is valid (i.e. that the x component is less than the board width, and the y component is less than the board height). Your aim is to add code into Parser.java which throws a SyntaxError when an invalid position is read.

3 Exposing Squares (worth 15%)

This part concerns the following requirements for the minesweeper game:

3. An exposing move `E(x,y)` requires the square at position `(x,y)` is not already exposed.
4. An exposing move `E(x,y)` requires the square at position `(x,y)` is not flagged.

You should find that some or all of the tests invalidFile_12 , ..., invalidFile_15 currently fail. This is because the method apply(Game) in ExposeMove.java is empty. Your aim is to implement this method appropriately. The method should throw a SyntaxError if one of the above requirements is not met. Once finished, you should find that tests invalidFile_12 , ..., invalidFile_15 now pass (except invalidFile_14).

4 Flagging Squares (worth 20%)

This part concerns the following requirement for the minesweeper game:

5. A flagging move `F(x,y)` requires the square at position `(x,y)` is not already flagged.
6. An unflagging move `U(x,y)` requires the square at position `(x,y)` is flagged.

You should find that some or all of the tests invalidFile_16 , ..., invalidFile_20 currently fail. This is because the method apply(Game) in FlagMove.java is empty. Your aim is to implement this method appropriately. The method should throw a SyntaxError if one of the above requirements is not met. Once finished, you should find that tests invalidFile_16 , ..., invalidFile_20 now pass.

PLEASE TURN OVER

5 Winning & Losing (worth 20%)

This part concerns the following requirement for the minesweeper game:

7. A losing move L(x,y) requires the square at position `(x,y)` contains a bomb.
8. A winning move W(x,y) requires that the square at position `(x,y)` does not contain a bomb.
9. A winning move W(x,y) requires that there are no hidden blank squares after the square at position `(x,y)` is exposed.

You should find that some or all of the tests invalidFile_21 , ..., invalidFile_24 currently fail. This is because the method apply(Game) in file EndMove.java has not been implemented. Your aim is to implement this method appropriately. The method should throw a SyntaxError if one of the above requirements is not met. Once finished, you should find that tests invalidFile_21 , ..., invalidFile_24 now pass.

6 Recursive Exposure (worth 20%)

The final (and most challenging) requirement for the minesweeper game is the following:

10. When a move `E(x,y)` exposes a blank square with no adjacent bombs (i.e. `numberOfBombsAround==0`), every adjacent blank square is recursively exposed, and so on.

NOTE: For any square `(x,y)` the *adjacent squares* have the following locations: `(x-1,y-1)`, `(x,y-1)`, `(x+1,y-1)`, `(x-1,y)`, `(x+1,y)`, `(x-1,y+1)`, `(x,y+1)` and `(x+1,y+1)`. Since some of these may be invalid board positions, there are *at most* 8 adjacent squares. However, in some cases, there will be less (e.g. for a corner square).

You should find that some or all of the tests validFile_25 , ..., invalidFile_30 currently fail. Your aim is to implement recursive exposure, such that tests validFile_25 , ..., invalidFile_30 now pass.