Test Documentation

Madeleine Ekblom

December 7, 2012

JUnit tests

The logic in the game, eg. the class Game, are tested in three different JUnit tests. In the first one, GameTest, the constructor is tested. There are three different kinds of wrong inputs: the mines are more than the number of squares, negative input values and an empty constructor. In the second JUnit test, GameTest2, the help methods that are used to create a text version of the gameboard are tested. The things that are tested are: the number of mines are correct and if the numbers are correctly placed. The third JUnit test, GameTest3, checks that the main method, createGame, generates random gameboards.

In the GameMovesTest, the logic for the game is tested. It checks that the methods openButtons, markWithFlag and unMarkSquare works properly. The test checks that the helpmatrices flagMatrix and help keeps update when a method is called.

In the HighScoreTest, the reading and writing of the highscore files works. First it checks that each level returns the correct file in the getFile method.

Interface tests

To test the user interface, I created a class called ButtonsTestMain, which is a simple version of the game. In ButtonsTestMain, I tried different ways of creating the buttons. Firstly, by naming the buttons as b1, b2, b3 and b4, and then realising that it is impossible to create a 100 buttons like that, and secondly by creating a matrix of JButtons.

After creating the buttons, it was time to make the buttons openable. Under a button there can be an empty square, a number or a mine. Opening a mine or a number doesn't cause any problems, but if the square is empty it must open all the other buttons until it is not an empty one. This is done with recursion. I tested this by creating gameboards of different sizes that had no mines, and trying to open all buttons by pressing one button (different buttons.). When it worked with no mines, I added mines and checked again if it worked. When I knew it worked I could create the real main class for the game.

After that I added some more functions. To test the markFlag - method, I just checked that it marked the correct button, and unmarked the button if the right button was pressed again. I did this several time.

To check whether the lose and win functions worked, I played the game on beginner level. From the tests I assumed the functions were done properly, and did the sames thing for different gameboard levels, and got the results I wanted.

When a new gameboard is created after either losing or winning, the previous window doesn't close. I see this as a problem, but haven't come up with a way to fix it.

Highscore list

The highscore lists are not totally reliable, since you can modify (for example by removing #) in some way that makes the file unreadable. The class highscore is tested by hand in ReadFileMain. First I tested that it is possible to write something into specific text file and then read the file. I did this by looping and printed out the results. Since it is results that the file reads, it means that it has to be a String (a name) in the first column and a String that can be converted to an integer (time given in whole seconds) in the second column. The columns are separated by #. It will cause a problem reading the file if it is not a string line. The sorting algortithm sortList is tested by hand and in JUnit tests. It sorts the results in an ascending order according to the times (second column). The method getTop5 will return the first five results from the sorted list.

The highscore list doesn't update meanwhile the program is running; instead it updates when the program restarts.