SOFE 3980U: Software Quality
Assignment 1: Choosing the right software process to build good-quality software, and ensuring quality using test automation

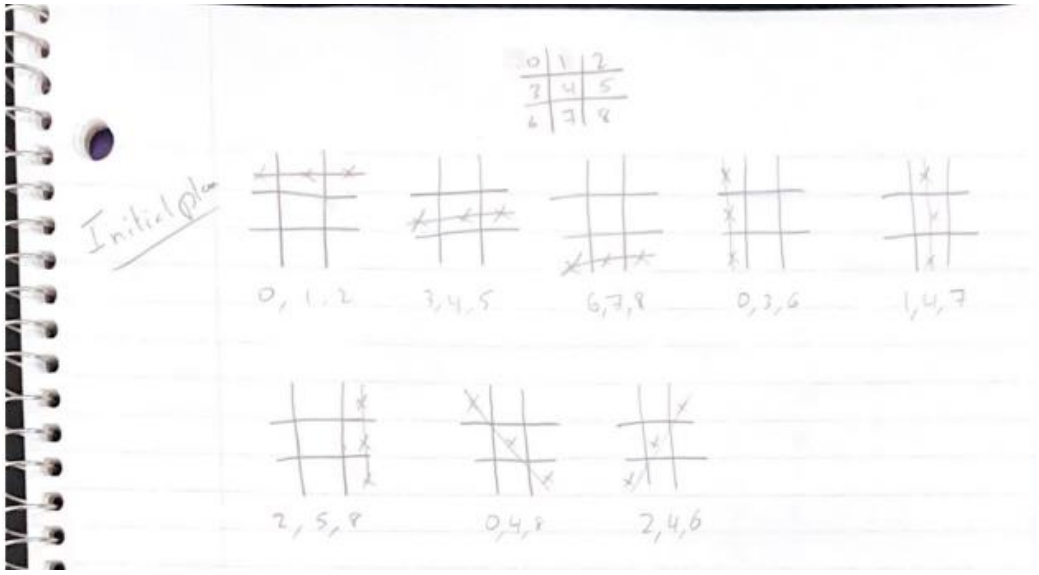
Raphaiel Halim; 100700318
March 3rd, 2022

Introduction

The game chosen to be developed as part of the assignment has been Tic-Tac-Toe GUI based game written in Java. The game is a 2-player game with "X" being the starting symbol, and switching to "O" after each turn, cycling between the two symbols. In addition, there is a menu bar that allows the player to either start a new game or quit at any time. Lastly, a winning screen with which player won or a draw screen shows up depending on if the board is full when no moves are left. To check code coverage, it was done using the built-in coverage tool in eclipse. Finally, all the automated testing was written using the JUnit framework to allow to test nearly every method in the program as well as the winning combinations when running the game.

Reasonings behind particular software process

After deciding to create a tic-tac-toe game in java, I began the research into the possible ways to create one. Initially, I was following the geeksforgeeks guide (https://www.geeksforgeeks.org/tic-tac-toe-game-in-java/), where the game is run in the console, with taking input by typing in which location on the board to place the symbol. How it looked like:

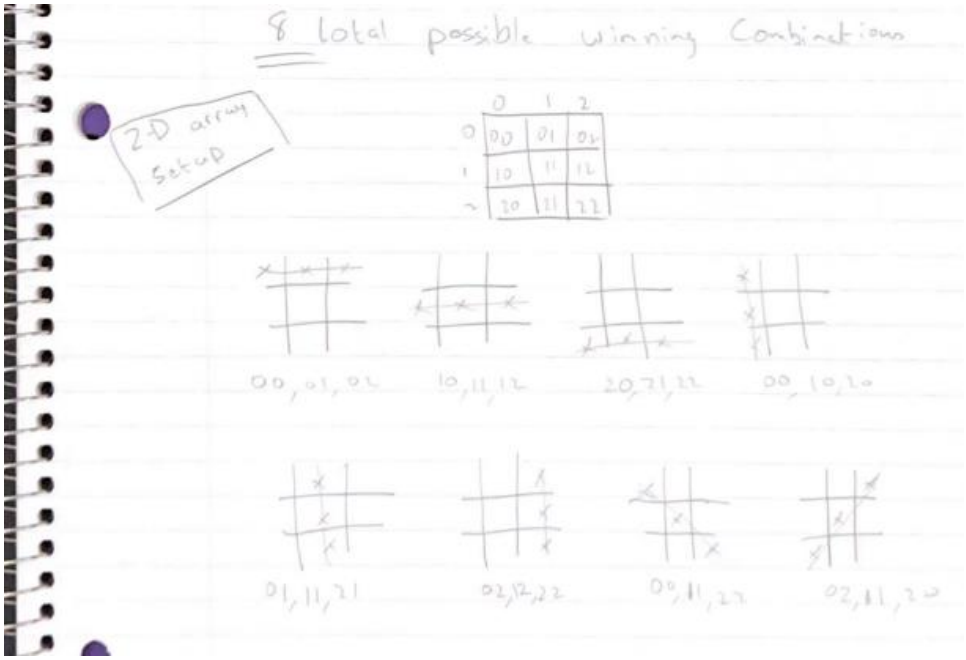


After studying the functionality of the code, I decided to transform the game into a GUI based version. Thus, I began looking into using Swing/JFrame in java. After spending a couple of hours learning it, I began changing how to improve the initial game. I decided to change it the board being based on a 2-D array instead. During the entire process I had used many references to see how to create the game, all of which are listed in the references section at the bottom of the report. The final functionality of the game is that when launched, a window pops up with a 3x3 board. The "X" player is always the starting one. With every turn, the player is switched and the program checks for a winner. In addition, the window has a menu bar with two options. The first being to start a new game, and the second being to quit the game. A breakdown of the methods used are explained below.

Description of functionalities of software and code

- Game written in Java, with the GUI portion being written using Swing and JFrame.
- The code was written by breaking down the functionality into a couple of methods.
- The main methods written for the game were:

- TicTacToeGui extending JFrame: The class containing all the below methods. It initializes all the variables being used in the program as well as the 10 Booleans used for testing purposes (being checked in the Junit testing)
- TicTacToeGui: sets up the functionality of the GUI
- setMenuBar: which was responsible for having the functionality of
    - newGame: which when selected, resets/clears the Tic-tac-toe board
    - quit: which closes the game process
- resetBoard: which contains the functionality of resetting the board, which is done by clearing the 2-D array
- setupBoard: contains the portion responsible for setting up the 2-D array for the game. It checks for when one of the buttons in the 3x3 grid is pressed, after which it checks if a winner is found, switching the current player as well as incrementing the turn counter.
- switchPlayer: switches the current player based on who last placed the symbol on the board
- checkWinner: It checks through the 9 different possibilities (8 possible win scenarios and a draw scenario). If a winner is found, it displays a pop-up with the winner. If the game reaches 9 turns with no winner still detected, a pop-up displaying a draw is reached.



    - Case 1; Top row horizontal win
    - Case 2; Middle row horizontal win
    - Case 3; Bottom row horizontal win
    - Case 4; Left vertical win
    - Case 5; Middle vertical win
    - Case 6; Right vertical win
    - Case 7; Diagonal win top left to bottom right
    - Case 8; Diagonal win top right to bottom left
    - Case 9; When board is full, with no winner found

Challenges faced during test automations

With this being my first project where I need to implement automated testing, I faced lots of challenges in trying to test the game. Since we were taught the basics in class using Junit, I decided to try to use it. I was facing quite several issues trying to get the tests to work. A couple of the methods in the program were initially written as private void methods, and thus I began looking into ways to test them. Leading me to using Maven After spending a couple of hours learning the framework and trying to write the tests in Maven using Mockito without much luck, I decided to go back to just Junit testing. Finally, I had begun understanding further how to better write the testing for it to run properly. The testing is setup by first calling a BeforeEach statement to call the program a new time. After which, a total of 15 test cases were written to test all the functionalities of the game. The test cases are:

- emptyBoardWinner: which checks when a new game is launched if a winner is found
- testResetBoard: which checks that when the board method of resetting the board, the board does get reset
- testTurnCounterReset: checks to make sure that when the resetting of the board is called, the turn counter does go back to 0
- checkNoWinner: checks that after 3 turns (X then O then X) in the same vertical column, that no winner is found. This is done to make sure that the game can recognize that different symbols do not cause a win
- Next, there are testWinCase1 to testWinCase8: which goes through the previously mentioned win cases and test each one out to make sure that all the win cases occur.
- testCase9; which goes through and fills the board, so no winner is possible to make sure a draw is reached
- testSwitchPlayer: checks that after running the switchPlayer method, that the current player does switch to O
- testStartPlayer: checks that the start player is always X

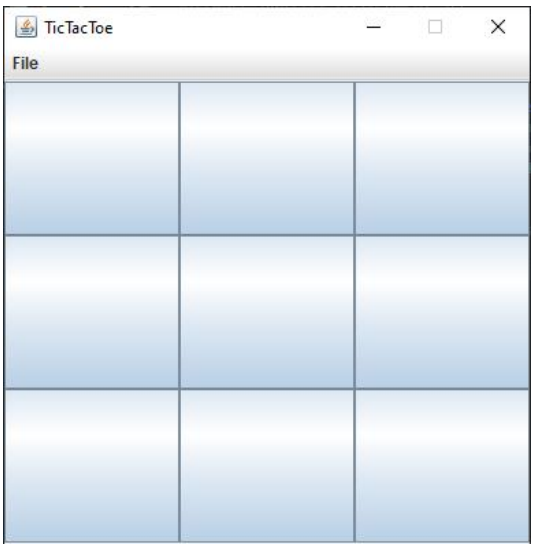
Each of the test case checks to see if a particular Boolean is true. This is done by having the Booleans become true when they are reached when they are reached in the program. For instance, when the testResetBoard test case runs, it checks if the Boolean isReset is false, since it only turns true when the method resetBoard gets called when selecting “New Game” from the menu bar. Similarly, the rest of the test cases work by a comparable way where it checks if a specific Boolean occurs to make sure that the portion it wants to check is reached.

Note regarding the 9 possible solutions

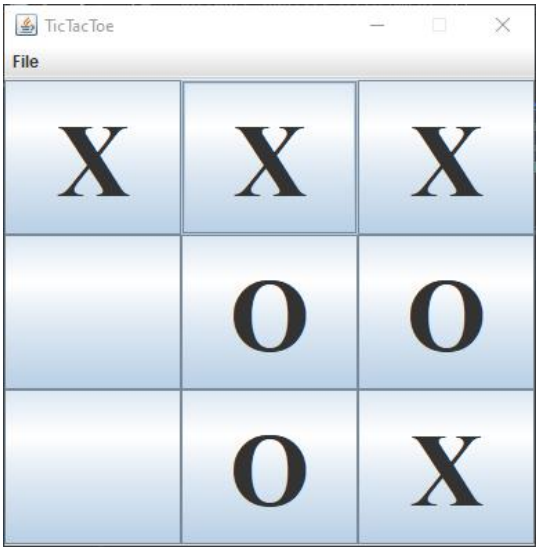
When each of the possible test cases for testing the 8 winning possibilities and draw scenario, a pop up opens with each one to show a visual representation of the game as well as a second popup with the winning/draw message. To continue running the tests, the “OK” needs to be pressed to continue testing. This has been done to allow to check that everything is being done properly.

Documenting detail instructions with snapshots

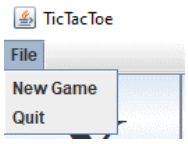
Game GUI new game:



GUI during plays (when boxes are clicked):



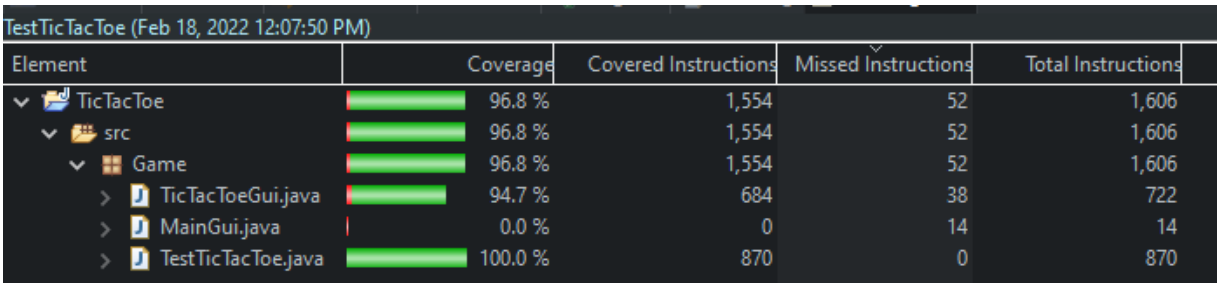
Winning Screen:

Menu bar: 

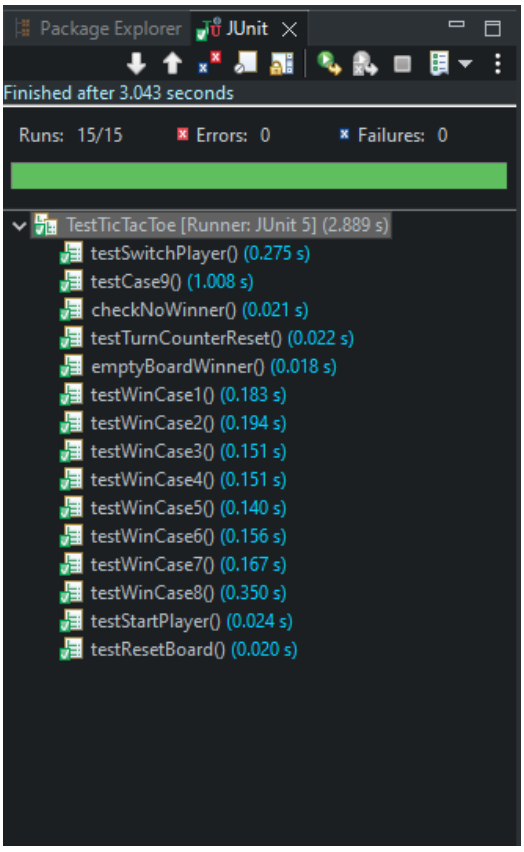
How to use software and test automation framework with sample test run

To run the game, you just need to run the MainGui.java class. It contains the main portion that starts the game (included is a demo video of how to run the game and playing it). To run the tests, you need to run the TestTicTacToe.java class and all 15 test cases should run (included is a demo of running the test cases). Below are the screenshots for the coverage as well as the result of the test cases. Note: that the MainGui.java shows 0.0% coverage due to the coverage being run using the TestTicTacToe class, which calls the TicTacToeGui. The last 5% for the TicTacToeGui not being covered are due to them being the portions of the GUI button listeners, since I was not able to run test cases to simulate the buttons being pressed. Running code coverage on the MainGui class, produces a 100% coverage since all it does is start a new instance of the TicTacToeGui.

Code coverage when running the TestTicTacToe.java:



Test Cases Results:



Instructions to run:

1. Create a java project (Making sure Java 1.8 is included)

2. Create a package called "Game"

3. Drag and drop onto the "Game" package the three classes: MainGui.java, TestTicTacToe.java, TicTacToeGui.java

4. Make sure to add the JUnit library to the java project

5. Now you could run the MainGui.java to try out the game

6. Could also run the TestTicTacToe.java, which goes through and tests all the 15 test-cases

7. In addition, you could check coverage for the classes as well

To add the JUnit Library to the java project:

1. Right click the java project

2. Select "Build Path"

3. Choose "Add Libaries..."

4. Select "JUnit"

5. Choose "JUnit 5"

6. Hit "Finish"

References

References used to code the game:

- https://www.guru99.com/java-swing-gui.html
- https://www.javatpoint.com/java-swing
- https://www.javatpoint.com/java-jframe
- https://www.guru99.com/java-swing-gui.html
- https://www.javatpoint.com/java-jmenuitem-and-jmenu
- The series by: https://www.youtube.com/watch?v=YMeVSoNumAg
- https://www.javatpoint.com/java-jbutton
- https://www.javatpoint.com/java-actionlistener
- https://www.tabnine.com/code/java/methods/javax.swing.JFrame/getContentPane
- https://www.javamex.com/tutorials/threads/invokelater.shtml
- https://www.tutorialspoint.com/what-is-the-importance-of-swingutilities-class-in-java
- https://stackoverflow.com/questions/39945881/java-tic-tac-toe-game-using-2 dimensional-array
- https://stackoverflow.com/questions/23233456/tic-tac-toe-in-java-using-2-d-arrays

References used to code the testing:

- https://www.vogella.com/tutorials/JUnit/article.html
- https://junit.org/junit4/javadoc/latest/org/junit/Assert.html
- https://www.softwaretestinghelp.com/junit-assertions/
- https://howtodoinjava.com/junit5/before-each-annotation-example/
- https://spring.io/guides/gs/maven/
- https://www.tutorialspoint.com/mockito/mockito_junit_integration.htm
- https://www.vogella.com/tutorials/Mockito/article.html
- https://www.javatpoint.com/mockito