Name: Sundarakrishnan Ganesh
LNU ID: sg222wn@student.lnu.se
Github Repo:
https://github.com/sg222wn/Java-Projects

Testing Plan:

This test includes testing the parts of the code that plays the major part in the HangMan Game. Methods to generate a random word, the number of lives left, to hide the words are some that will be JUnit tested. However this test doesn't focus much on testing the GUI (i.e. the JAVAFx) part of the code except some features of the game which will be manually tested from the end user's perspective.

The testing will be done only on a single class (HangmanMain) because it is the only class that has methods and variables that need to be tested. In other words, it is the most vital part of the game as it is the main gameplay part.

The method to generate a random word from the word array is one of the more important parts to keep the player engaged in the game. The game would become boring if the player only had to guess one word throughout his gaming session. Thus it is of utter importance that this method is working fine during run-time. Thus I chose to test this method.

The method to set the number of lives left is another important part of this game. As there are three difficulties in the game, as mentioned in the use cases, the corresponding difficulty levels are based on the number of lives the user is granted. Thus it is vital to test this part of the code also.

The method to generate the dashes according to the word length, is the primary objective in the game. Thus the dashes serves as a medium to hide the words in order for the user to guess the words. Thus it is vital to test whether the dashes are generated or not.

The pause button during gameplay and the select difficulty menu is manually tested as it provides the path to and out of the game.

Time Estimations:

- ***Estimated time for completing the testing plan:*** 6 hours

Actual time taken: 5 hours and 19 mins(2 PM - 7:19 PM approx.)

- ***Estimated time for completing the manual test cases:*** 2 hours
  Actual time taken: 48 mins(12:30 PM - 1:18PM approx.)
- ***Estimated time for completing the manual tests:*** 10 mins
  Actual time taken: 25 mins(2:30PM - 2:55PM)
- ***Estimated time for completing the code for JUnit tests:*** 30 mins
  Actual time taken: 1 hours and 30 mins(5PM - 6:30PM)
- ***Estimated time for completing the JUnit report:*** 1 hour
  Actual time taken: Around 30 mins(7:50- 8:20)

Manual Test Cases:

Test Case - 1 (Pass)

This test case will focus on manually testing the Select Difficulty menu of the program. This is tested because it decides the number of lives allocated regarding your difficulty selection and also the further development of the game.

1. Name: Testing Select Difficulty
2. ID: TC_UI_1
3. Reference: UC 2 Select Difficulty
4. The select difficulty menu is manually tested in this test case to check whether the game sets the correct life number based on the difficulty the user selects.
5. Preconditions for this TestCase: Play Game button in the Main Menu is clicked and the Difficulty Menu is shown.
6. ***Test Steps:***
   a) Click on the "***Noob***" difficulty button.
   (or)
   b) Click on the "***Slightly Average***" difficulty button.
   (or)
   c) Click on the "***Professional***" difficulty button.
   (or)

d) Click on the "***Go Back***" button.

7. ***Expected Results:*** If the ***Noob*** button is clicked, the control goes into the game and the life is set to the highest, which is 12. This means you have 12 tries before you lose. If the ***Slightly Average*** button is clicked then the life is set to 8. If the ***Professional*** button is clicked then the life is set to the lowest which is 6. The number of lives left is displayed on the screen. If the ***Go Back*** button is clicked then the Main Menu is displayed as a new session of the game is created.
8. ***Actual Results:***
   - ☒ Clicking ***Noob*** test(Life set to 12)
   - ☒ Clicking ***Slightly Average*** test(Life set to 9)
   - ☒ Clicking ***Professional*** test(Life set to 6)
   - ☒ Clicking ***Go Back*** test(Displayed the Main Menu)
9. ***Comments:***
   a) Nothing peculiar noted during the test. It worked perfectly.

Test Case - 2 (Pass)

This test will focus on maunally testing the Pause Button while playing the Game. This button is supposed to provide you options to go back to the main menu or resume the game.

1. Name: Testing Pause Game
2. ID. TC_UI_2
3. Reference: UC 4 Pause Game
4. The Pause Game option is manually tested in this test case to check whether the button when clicked displays a two options to either continue the game or exit to main menu.
5. Preconditions for this TestCase: The Game must be running.
6. ***Test Steps:***
   a) Click on the ***Pause Game*** button when the game is being played.
7. ***Expected Results:*** The pause menu is displayed with a prompt saying "Click OK button to go back to the main menu!". Then if the ***OK*** button is clicked then the current windows are closed and the Main Menu is displayed.

8. ***Actual Results:***
   ☒ Click ***Pause Button*** test(Pause menu is displayed)
   ☒ Click ***OK Button*** test(Main menu is displayed, Current Windows are closed)
9. ***Comments:***
   a) One thing I noticed during this test was that the Clicking the Pause Button displayed the Pause Menu perfectly but didn't actually pause the game. So it is a feature that I have yet to implement.
   b) Clicking the OK button took the control back to the Main Menu and ended the current session. Thus the Use case was fulfilled.

Automated-Unit-Tests

The following methods are tested using JUnit tests. These methods are used to set life in the game, a toString method that converts array to string, a method to get a word from the array and store it so the player can guess the word, a method that checks whether the game is over or not, and another method that is used to hide/ replace words with dashes for the user to guess the word. These methods play a vital role in a game such as hangman and so it is of utmost importance to test them automatically. In this case, all the methods have passed the test except the method for hiding the letters using the dashes. Thus that method's return value was changed in the source code to make it work. One of the methods in the source code used JAVAFx components, so the JAVAFx part was commented out for testing purposes and the logic part was tested. Thus all the essential methods have passed the JUnit test. The screenshots for the tests can be seen below.

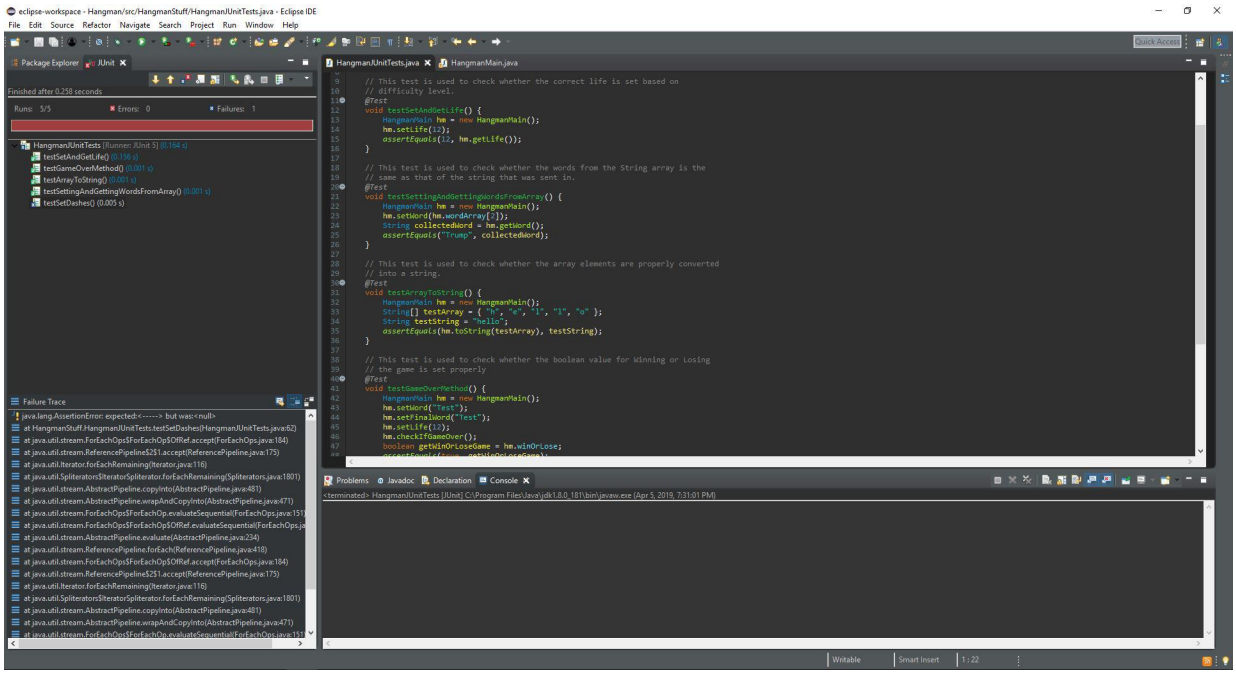

Fig.1(The test for the 5 mentioned methods with one failing the test)

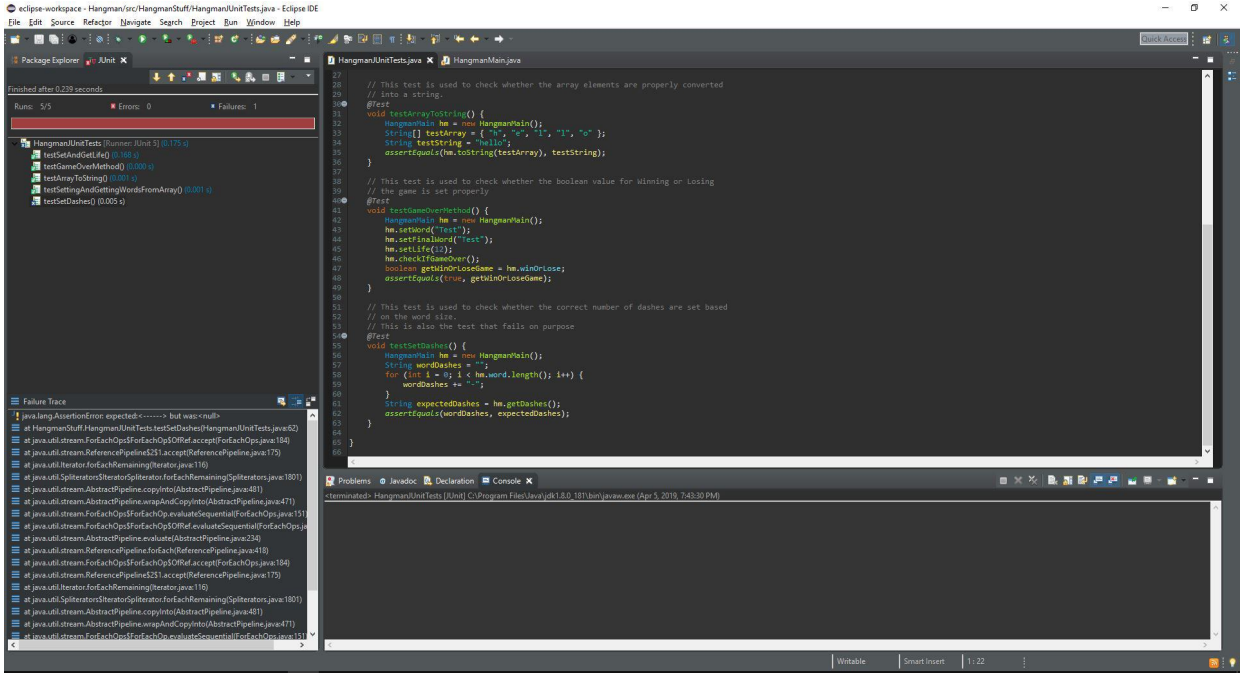

Fig.2(JUnit Test Code for the test that failed)

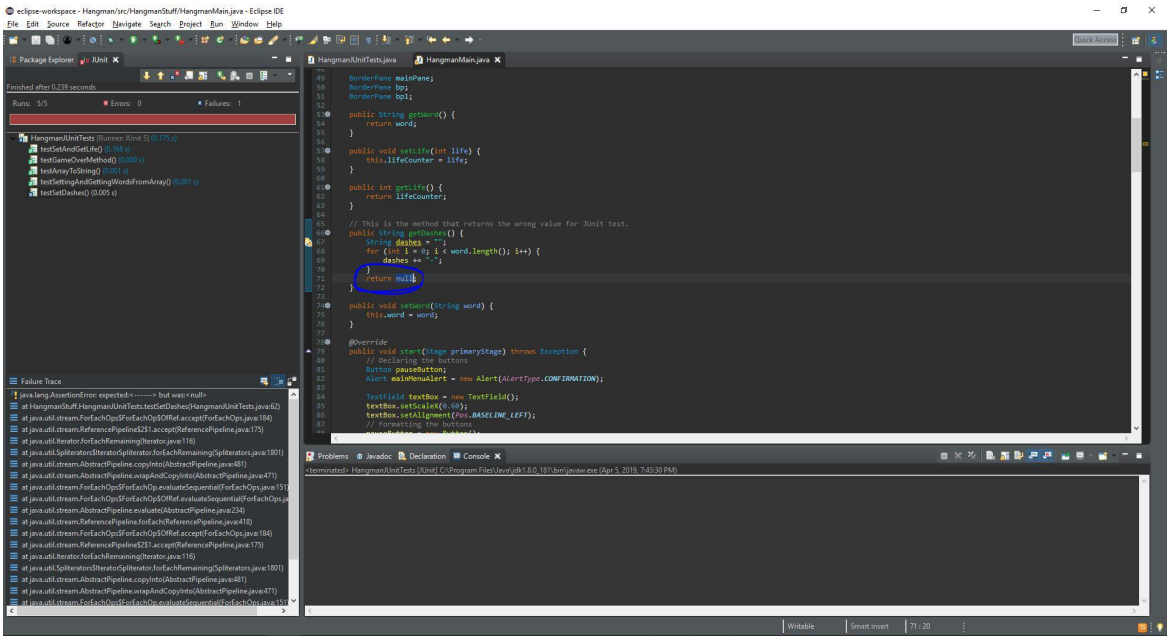

Fig.3(The cause of that test failing)

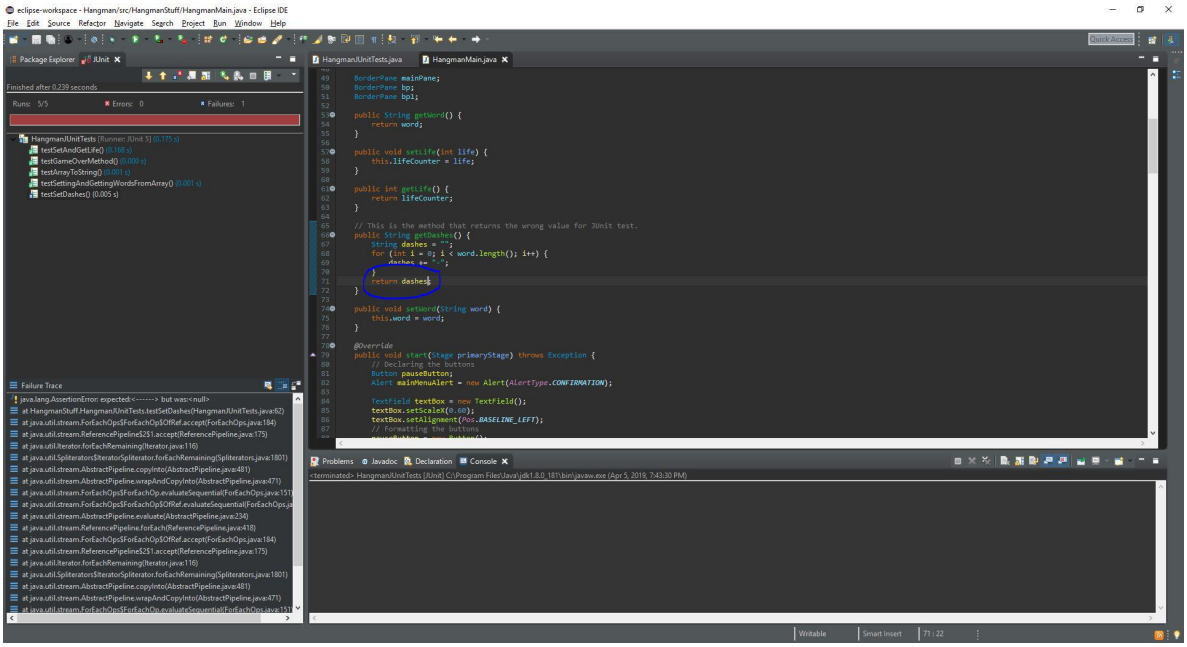

Fig.4(Fixing the bug in the source code to make that test pass)

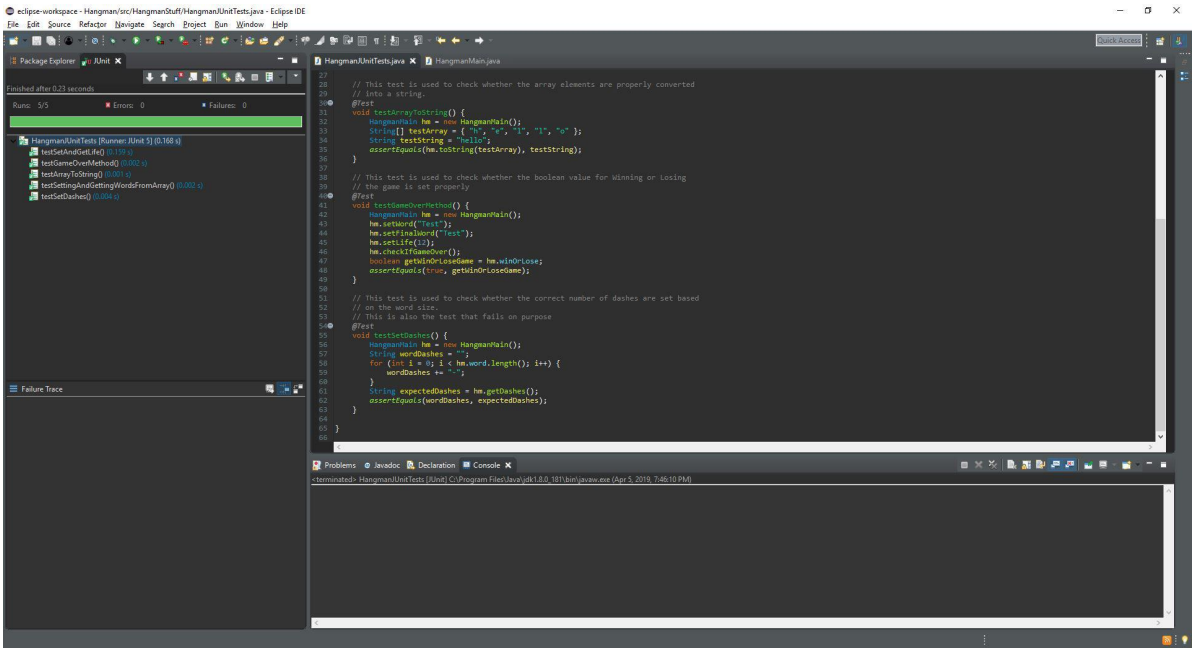

Fig.5(All the tests have passed)

Reflection

By testing this version of the source code, I am now able to point out where the errors are in my code. Testing the essential parts of the source code has made it clear that there were errors that I was later able to fix and make the program run with less glitches or bugs. Now it has occurred to me of how vital a role testing plays in various stages of code development. Manually testing the interface from the client's (end users') perspective have also helped in increasing the user friendliness of the hangman game, by handling exceptions in a way that the layman understands. Estimating the time duration for each process has increased the accuracy of predicting the time in which a process can be done. Thus to sum it up, this assignment has taught me how important testing a product really is.