

Greeter Application

Assignment 3



March 7, 2019

SaNA HAMeed

Sh223nw@student.lnu.se

**The Greeter application**

This application is designed for hangman game. In this application I will briefly describe the use case scenarios of my game. This is the simplest application and help the user how to play this game step by step and we also test all the modules either all modules are working correctly or not. This game have basic idea for guessing words if user guess wrong word the application shows an error if user proceed with correct words then user got score.

**Vision**

Our application is user friendly application. On our platform we provide a user friendly interface in which user enjoy and play game with interactively. Our application also provide some hints which is helpful for the user and user enjoying the game with the winning game. In the game we have some other options which is also helpful for the user.

**Use Cases**

**UC 1:**

Start a new game.

**Precondition:**

1. System support the game configuration
2. The file has been run and game screen appeared.

**Main Scenario:**

1. Go to the new game button and click on it.
2. New game is loaded on system.

**Alternative Scenario:**

Press start new game button and play a game.

**Activity Diagram:**

**UC 2:**

Play a single word game

**Precondition:**

1. Game was played before.

2. Game support to guess a single word.

**Main Scenario:**

1. Play game and try to guess word.
2. You can try for multiple time if you guess wrong

**Alternative Scenario:**

Guess word if you enter wrong word you can enter again a right word

**Activity Diagram:**

**UC 3:**

Select Level

**Precondition:**

1. Required level has been unblocked.

2. Game supports loading level.

**Main Scenario:**

1. Go to the new game and click to the select level.

2. Select level and load the select level.

**Alternative Scenario:**

Click on select level and directly select the level.

**Activity Diagram:**

**UC 4:**

Exit Game

**Precondition:**

A game level is being played.

**Main Scenario:**

1. If user guess wrong words more than 3 time.
2. Game is automatically come in start menu.

**Alternative Scenario:**

Click on exit game.

**Activity Diagram:**

**Test Plan:**

**Objective:**

The objective of testing is to test all the small parts of the game that are free of errors or not.

**2.2 What Will Be Tested and How**

The intention is to test all implemented Use Cases. There will be manual Test Case of the running client application that intend to cover all possible paths in UC1 and UC2. These two use cases have been selected for manual testing because they cover the functionality most essential to the product, meaning selecting to start playing from after the program starts and actually playing the game. All paths in these two use cases should be covered by the tests. Additionally, automated unit tests will be written for the application with the intention to cover as many of the methods in each class as possible. The classes that are currently implemented are Main, Word, Wordlist, Game and Menu. These unit tests will be executed to verify that no obvious errors are present. Due to the small size of the application, the only automated tests that will be created are unit tests. Automated tests for the not yet implemented functionality will be written in the next iteration, alongside their implementation.

**Time Plan:**

|  |  |  |
| --- | --- | --- |
| **Task** | **Estimated** | **Actual** |
| **Manual TC** | 1h 30m | 1h |
| **Unit TC** | 1h | 45m |
| **Running manual tests** | 25m | 5m |
| **Test Inspection** | 2h | 1h 38m |
| **Test report** | 1h | 50m |

**Manual Test cases:**

**1. Manual tests**

Both manual tests were conducted on the Use Case (UC) 2 (play single word game), because it is the best implemented and most complex use case in the project and there are many possible different scenarios to be tested.

For the first manual test, I chose a scenario that tests both UC1 (start game) as well an alternate scenario of UC2 with one of the many ways for the user to win the game (guessing the entire word). Encompassing UC1 in this test is good because, if successful, it may allow the assumption of a successful game start for further tests involving UC2. And testing a win-game scenario is also important and this stage, to find defects in this crucial step of the game.

For the second manual test, I chose a scenario where the user resets the game, because its implementation involves the system asking the user for confirmation, which is appropriate for manual testing.

**Manual test cases**

**TC 2.1 Guessing an entire word correctly**

**Use cases tested**: UC1 – Start game and UC2 – Play a single-word game

**Scenario**: The Player tries to guess the hidden word and succeeds (UC2 alternate scenario)

**Precondition**: The application is executed by specifying the String as an argument to the Hangman constructor in the game’s Main.java class.

**Test steps**

- Start the app

- The system shows the welcome message.

- Press “1” and enter.

- The system shows the new game menu and waits for user input.

- Press “1” and enter.

- The system displays a single game board containing the hidden word, the number of tries left.

- The system shows “Please enter a letter or guess the entire word and waits for user input.

- Write “sanaa” and press enter.

**Expected**

- The system shows the win game board with the text “you won” and “you found

sanaa in 1 tries”, “1 – Return to start menu, 0 – Quit the application”.

- The system shows “Please press a key to select an option” and waits for user input.

**Result**

**Passed**

Failed

Comments: -

**TC 2.2 Resetting an ongoing game**

**Use case tested**: UC2 Play a single-word game

**Scenario**: Resetting an ongoing game.

**Precondition**: A single-word game is running and the system is waiting for the player’s input.

**Test steps:**

- The system shows “Please enter a letter or guess the entire word and waits for user input.

- Write “reset game” and press enter.

- The system shows “Do you really want to end this game and return to the start menu? ("y" = yes, "n" = no):”

- Write “y” and press enter.

**Expected**

- The system shows “resetting the game...”

- The system shows the welcome message “---- Welcome to the world's best Hangman game ever! ----” and the start menu.

**Result**

**Passed successfully**

Failed

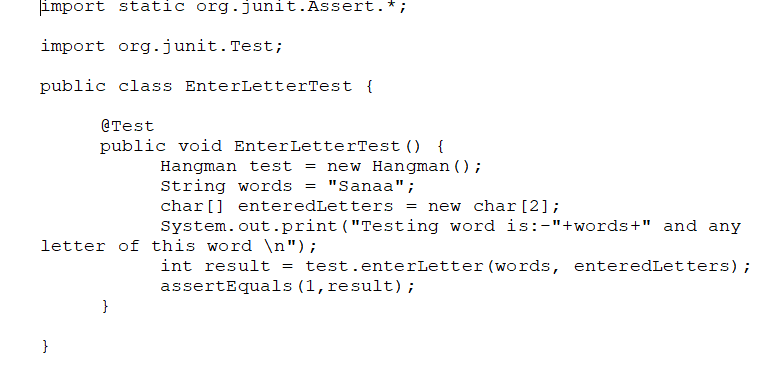
Comments -

**Test Report:**

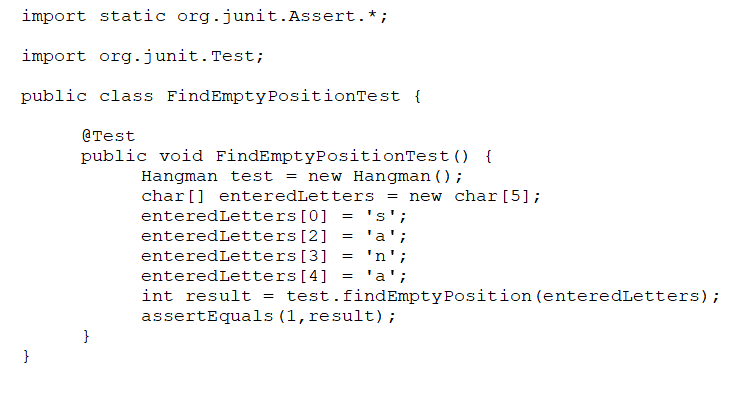
|  |  |  |
| --- | --- | --- |
| **Test** | **UC1** | **UC2** |
| **TC1.1** | **OK** | **OK** |
| **TC1.2** | **OK** | **OK** |
| **Coverage &Success** | **OK** | **OK** |

**Automated unit tests**

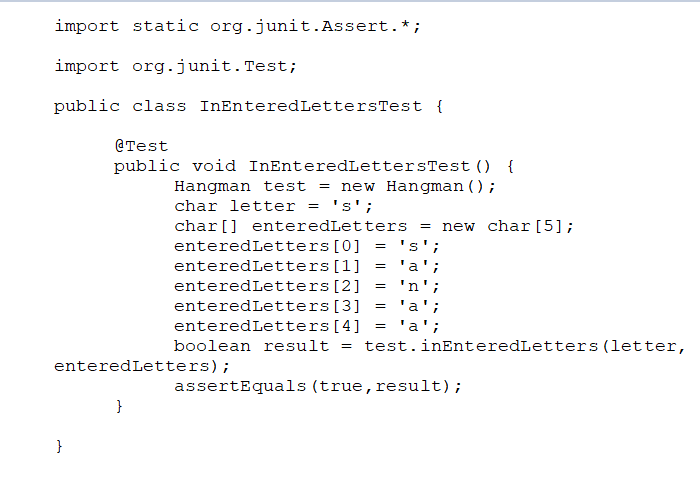
**Testing Enter Letter Test**

****

**Testing empty position test**



Testing InEnter letter test:



**Automated unit test coverage and success**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test** | **Console view** | **Main** | **Greet controller** | **Name DAL** |
| **enter word test** | **0% error** | **0% error** | **0% error** | **100% ok** |
| **Empty position test** | **0% error** | **0% error** | **0% error** | **100% ok** |
| **InEntered letter test** | **0% error** | **0% error** | **0% error** | **100% ok** |

**Reflection:**

Although we spent some time reflecting and discussing non-functional requirements such as ‘testability’ in previous assignments, it has only now become clear what this means. In the previous iteration, I was concerned only about the use cases. Then, it seemed perfectly fine – advantageous, even – to manipulate the game’s attributes (class variables) directly in each method’s body, not using any arguments to pass these values between different methods (because they had class scope). After trying to design automated tests for such methods, I now understand that this isn’t the correct way to do it, because it removes the possibility of assigning custom values to these methods while testing them.

Two good examples of changes to enhance testability were performed on the two methods that were tested with automated unit tests. Neither of these methods received a parameters, because the values used in these methods were assigned to class variables and therefore could be accessed in the methods’ bodies. This removed the possibility to test these methods in isolation using predefined values. I changed their implementations to receive these values as arguments and was then able to perform automated unit testing on these methods in JUnit.

I have no doubts that this insight will change the way I write code in the future, because it gave me a practical example of why it is desirable to pass values between method calls as arguments, independently of the scope of the variables that hold these values.

**Hand in:**

**https://github.com/Sanahameed/sh223nw\_1dv600/tree/master**