Testing plan

## **Objective**

The objective is to test the code that was implemented the last iteration. The testing of this application will be done using both manual testing and unit testing. For the unit testing, we will use JUnit 5 and create our individual test suite. With these unit tests, we will test the classes and their functions in isolation. The manual testing will test the functionality by simply making sure that the application behaves as intended. By manual testing, we will test out how the parts interact with each other and make sure that the game functions in a predictable manner.

## **What to test and how.**

We intend to test the two use cases for this application by manual testing and then test the components in isolation by using unit tests.

## **Time plan**

|  |  |  |
| --- | --- | --- |
| **Task** | **Estimated** | **Actual** |
| Manual TC | 30m | 25m |
| Unit tests | 30m | 20m |
| Running manual tests | 30m | 15m |
| Code inspection | 45m | 30m |
| Test report | 30m | 25m |

## 

## **Manual Test-Cases**

### **TC1.1 Complete a game of Hangman**

**Use case:** Start new game

**Scenario:** Successfully start a new game and correctly guess the word.

The main scenario of UC1 will be tested by running a game of “hangman”.

**Precondition:** None

**Test steps**

* Start the app
* Play a game where we guess the correct word.
* Enter our name if we get a better score than the current, and afterwards we will be given the menu options.

### **Expected**

* The system should print the gallows correctly.
* If we beat the current score, we will be prompted to enter our name for the highscores.

### 

### **TC1.2 Guess an incorrect word**

**Use case**: UC1 Start new game

**Scenario**: Start a new game of hangman and fail to guess the correct words.

**Precondition**: None

#### **Test steps**

* Start the app
* Play a game where we guess the incorrect word 8 times.

### **Expected**

* The system should show the text for failing to guess the correct word, show the whole gallows and show the menu options.

### **TC2.1 Show the high score list**

**Use case**: UC2 Show Highscore

**Scenario**: Use the “show highscores” menu option.

**Precondition**: None

#### **Test steps**

* Start the app
* Press “2” to enter the highscore list.

### **Expected**

* The system should show a sorted list with the current highscores. The application should then return us to the main menu.

## 

## **Test Report**

Test traceability matrix and success

|  |  |  |
| --- | --- | --- |
| **Test** | **UC1** | **UC2** |
| TC1.1 | OK | - |
| TC1.2 | OK | - |
| TC2.1 | - | OK |
| COVERAGE & SUCCESS | OK | OK |

Automated unit test coverage and success

|  |  |  |
| --- | --- | --- |
| **Test** | **HighScore** | **WordList** |
| HighScoreTest | OK | - |
| WordListTest | - | OK |

### **Comment**

Testing concludes that the system behaves as expected.

### **Reflections**

Creating a test plan requires a lot more time than you’d think since it you need to make sure that the entire application is working and that all the functionality has been covered. Thankfully, I the same structure the ”greeter” application used for their test plan, so I didnt have to structure the plan from scratch, but could instead re-use and modify their plan to my application. This saved me a lot of time and also gave me a good clue on where I should put my focus in regards to the testing.