Report

Hangman Project

Author: Osama Zarraa
Semester: HT2018
Area: Software Technology
Course code: 1DV600
Date: 2019-04-18

Contents

1 Revision History

Date	Version	Description	Author
2018-02-08	1.0	Initial Revision	Osama Zarraa
2018-02-08	1.1	Fix some bugs related to filtering the user input	Osama Zarraa
2018-02-10	1.2	Code Refactoring	Osama Zarraa
2018-02-21	3.0	Design and modelling	Osama Zarraa
2018-02-24	4.0	Write Test Use Cases	Osama Zarraa
2018-03-10	4.1	Improve the class diagram and state machine diagram so they can adapt the new timer feature	Osama Zarraa
2018-03-11	4.2	Improve the use case diagrams	Osama Zarraa
2018-03-12	4.3	Implement the new timer feature	Osama Zarraa
2018-03-19	4.4	Run manual testing	Osama Zarraa
2018-03-20	4.5	Write and run Automated unit tests	Osama Zarraa
2018-03-22	5.0	Completed Submission	Osama Zarraa
2019-08-22	5.1	Final submission	Osama Zarraa

2 General Information

Project Summary	
Project Name	Hangman game
Project Id	1DV_10001_HM
Project Manager	Osama Zarraa
Main Client	+13 age user
Key Stakeholders	End users, developers, testers

3 Vision

Hangman project is a text-based and a word-guessing game which consists of a word, players and an image that is drawn based on player's answer that is given character by character. Game starts with loading old files that represents players, scores and old games. After loading, a menu will appear telling the user to choose an option from:

- Registering a player
- Selecting an old player
- Start a game
- Showing players scores
- Exit

After starting a game, a "store" will choose a random word from a list which is loaded previously. Program will print an underscore characters represents the missed word, without showing the real characters, and players can guess the word by giving a character. Each time a player gives a character, program will check if this character is part of the missed word. If character is included in the word the program will replace underscores that matches the given character with the right characters. Otherwise a figure of a hangman will be completed by every wrong answer given by the players. The game has maximum eight or ten wrong answers to complete the hangman figure. If the figure of the hangman is completed the game will be ended and saved in a store for later statistics, else the player who gave the last successful character will be considered as a winner and a log will be saved as part of a point system. The program will also have a time limit which represents that the deadline of the game.

3.1 Reflection

To find out how to write a vision for the product was not that hard because after I read the teacher feedback for about how to write. Online material also helped me on how the vision should be written, and what is the purpose of a vision document in every project plan. In general, I think the vision should be in a separated document not inside the project plan document it does make more sense. Have the vision inside the project plan made some confusion to me since I think it does not have much to do with the project plan.

4 Project Plan

4.1 Introduction

Hangman project is a text-based and a word-guessing game which consists of a word, players and an image that is drawn based on player's answer that is given character by character. This document presents the four iterations which the development of this project goes through.

4.2 Justification

This reason why this game is made is that there is no hangman games in the market that have the same functionality that this game provides. Such as: sitting a timer for each player when they start the game. Also, each player has a history in the system record their scores and name.

4.3 Stakeholders

Stakeholder	**Description**
Software developer	Responsible to develop the requirement and all the features of the game
Software tester	Responsible to test the features of the game
End-user	Players, at age 13+ who have interest puzzle games.
Project owner	The course coordinator.
Parents	Parents of the players who are younger than 18 years old.

4.4 Resources

Resource	**Description**
Man power	Only one, who is responsible for development and testing process.
Literature	
Available time	4 hours per day during this period: From 2019.01.21 to 2019.03.24 Holidays are excluded

4.5 Hard- and Software Requirements

Type	**Description**
Hardware	Computer x64 capable Processor RAM (Random access memory) 1GB
Software	Overleaf & Online LATEX Editor Windows 10 JRE (Java runtime environment) v10 IDE(Intellij) v2019.1 JDK (Java development kit) v11

4.6 Overall Project Schedule

8th February 2019: Project plan.
21st February 2019: Working version of the game with UML.
8th March 2019: Testing.
22nd March 2019: Final report.

4.7 Scope, Constraints and Assumptions

Hangman game is a console application. It is a puzzle game, based on word-guessing game where a player starts the game and start guessing a word by providing characters. A new part of the hangman image will be drawn every time the player fails to guess the right character. Game starts with loading the words and players files. The words file has the words that a player guesses from. Also, the players files contains the players information of (id, name, scores). The features that the game provides are:

- Start game
- Play game, with time limit game
- List words
- Remove a word
- Add word
- List players (with their scores)
- Register player
- Remove player
- Log in player
- Quit

When the player starts the game, the system saves the current time, the players start entering characters, for each wrong character a part of the hangman image shows up and the system will check the time if it exceeded 3 minutes, if the time is exceeded or the player enters 9 wrong characters then the player loses. If the player guesses the word correctly in less 3 minutes and less than 9 wrong guesses then the player wins the game.

4.8 Reflection

Writing a project plan was a bit hard because it is my first time in this course to write such a document like this. I Had to go through many online templates of project plans, so I know what is purpose of it is. Was confused between the different templates online and the template that was given to us in this course. However, it was a good experience for me when I rewrite it more than once to fulfill all the requirements.

5 Iterations

5.1 Iteration 1

Task	Description	Estimated Time(hours)	Deadline
Read documentation	Reading the documentation about how to write the project plan	16	8th February 2019
Prepare working tools	Prepare the tools required for the project	8	8th February 2019
Create the project basic files and classes	Create the first version of the project	2	8th February 2019
Write the project plan	Write the project plan file including the iterations and time log and risks	8	8th February 2019

5.2 Iteration 2

Task	Description	Estimated Time(hours)	Deadline
Read documentation	Reading the documentation about the class diagram and state machine diagram besides reading about use cases and fully dressed use cases	16	21st February 2019
Create Use Case Diagram	Sketching the use case diagram	8	21st February 2019
Write Fully dressed use case	Write a fully dressed use case for the second use case	8	21st February 2019
Create State Machine diagrams	Sketch the state machine diagram to the second use case with and without time limit feature	16	21st February 2019

5.3 Iteration 3

Task	**Description**	**Estimated Time(hours)**	**Deadline**
Read documentation	Reading the documentation about how to make manual and automated testing	16	8th March 2019
Write manual tests	Create manual tests for use cases 1,2,3	8	8th March 2019
Write Automated tests	Create automated tests for the Word class and for the Game class	8	8th March 2019
Write report	Write the final test report with images about the automated tests	8	8th March 2019

5.4 Iteration 4

Task	**Description**	**Estimated Time(hours)**	**Deadline**
Write the plan	Write the plan for the whole project	8	18th April 2019
Model the non-completed features	Create the state machine diagram and use case diagram for the new features	8	18th April 2019
Implement features	Complete the features that are not implemented in the previous iterations	8	18th April 2019
Test the new added features	Testing the new features including the players class	8	18th April 2019

6 Risk analysis

In this project the main three risks are: sickness, hardware failure, software crashing and learning difficulties.

6.1 List of risks

Risk	Impact	Probability	Strategies
Sickness	Not submitting the project at before the deadline	10%	Have a strict plan to finish the tasks before few days before the deadlines
Learning difficulties	Hinder the development process to progress	50%	Try to use slack to ask. Study in studies groups where I can share experience and ask and learn more.
Software crash		10%	Save files multiple times
Hardware damage	Losing the project files	10%	Backup all the files that are related to the assignment

6.2 Reflection

Being prepared for the risks is something mandatory in SDLC (Software development life cycle). It is a way to keep the developer within the time-frame of the project and give him a chance to overcome them without affecting the whole project. Moreover, it gives the developer a future perspective of what would happen and how to make a backup plan for issues that will come.

6.3 Strategies

For sickness issue, I will try to get help from my colleges and I will consider adding more time from the beginning to solve this issue. For hardware failure issue, I will use version control system such as GitHub for preventing losing the project files. For the Software crash, I will make copies of the files that I work on so I can go back to them if needed.

7 Time log

Task	**Actual time(hours)**
Iteration 1	
Reading documentation	15
Prepare the tools	8
Create first version of the project	6
Write a project plan	17
Iteration 2	
Read documentation	16
Create use case diagram	8
Write fully dressed use case	8
Create State Machine diagrams	16
Iteration 3	
Read documentation	12
Write manual tests	8
Write automated tests	7
Iteration 4	
Write the project plan	8
Model the non-completed features	7
Implement features	9
Test the new added features	8

8 Appendix 1

8.1 Use Case 2 (Play game)

ID	1
Title	Play game
Primary Actor	Player
Preconditions	• The game is running
Postconditions	• The main menu is shown
Main Scenario	1. Starts when the user selects to start a game from menu 2. The system picks up a random word from predefined store 3. The system presents the underscores which represents the number of the character of missing word 4. The player enters a character 5. The system validates the entered character 6. The system replaces the underscores with guessed character/s 7. Repeat from step 3 if the whole word is not matched 8. The system shows a message represents that the player won the game 9. Go to main menu
Alternative scenarios	The system failed to pick up a random word • Display error message • Go to main menu
	The player exceeds 3 minutes (allowed playing time) • The system shows a message that represents the player is lost • Go to main menu

Alternative scenarios	The system failed to pick up a random word • Display error message • Go to main menu
	The player exceeds 3 minutes (allowed playing time) • The system shows a message that represents the player is lost • Go to main menu
	The player chooses a special character which is (*) to exit • Go to quit confirmation (UC 3 “According to assignment description”)
	The user enters miss-matched character to the word • The system displays a part of hangman image based on number of failed tries • Repeat from 5.1 if the entered character is not matching and image is not completed • The system displays full hangman image • The system shows a message that represents the player is lost. • Go to main menu
Frequency of Use	A lot
Status	Done
Owner	Osama Zarraa
Priority	1

8.2 Use Case 4 (Login)

ID	UC4
Title	Login
Primary Actor	Player
Preconditions	
Postconditions	• The main menu is shown
Main Scenario	1. Starts when the user selects to log-in from menu 2. The player enters a name 3. The system validates the entered name 4. The system shows a menu message with the player name that the player logged in
Alternative scenarios	The player entered incorrect name • Display error message • Go to players menu
Frequency of Use	Not a lot
Status	Done
Owner	Osama Zarraa
Priority	1

8.3 Use Case 5 (Register Player)

ID	UC5
Title	Register Player
Primary Actor	Player
Preconditions	
Postconditions	• The players menu is shown
Main Scenario	1. Starts when the user selects to register player from menu 2. The user enters a name 3. The system validates the entered name 4. The system returns to players menu
Alternative scenarios	The player entered an empty name • Display error message • Go to players menu
Frequency of Use	Not a lot
Status	Done
Owner	Osama Zarraa
Priority	1

8.4 Class Diagram

The following figure shows a class diagram of HangMan project:

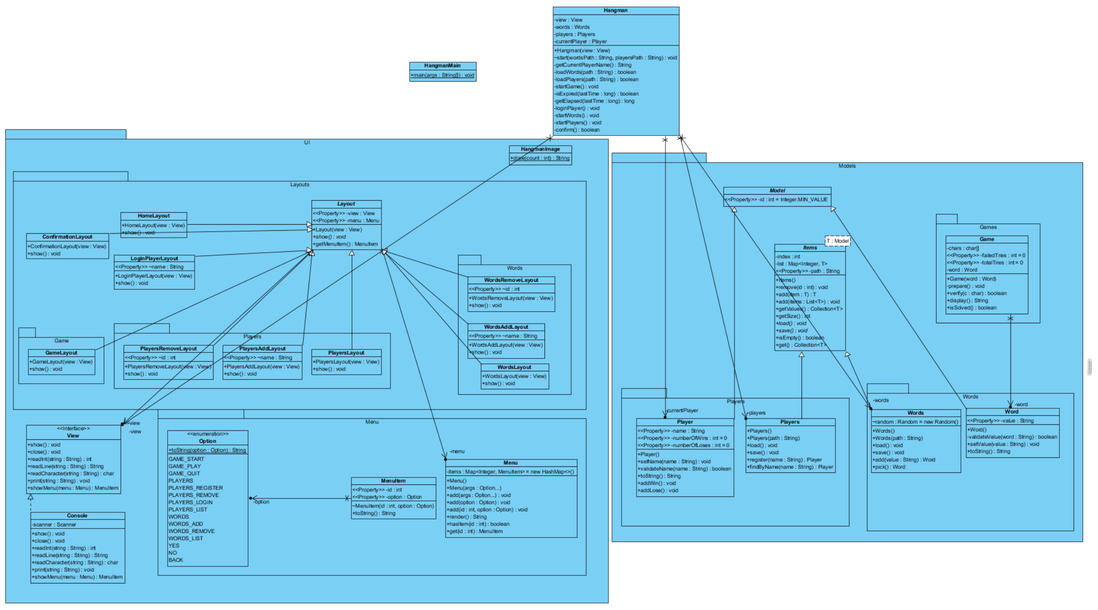


Figure 1: Class Diagram

Figure 1 shows the structure of the system by showing the system's classes in both of the packages: Model and UI packages and the relations in between their classes.

8.5 Use Case Diagram

Figure 2 describes the use case diagram for the game which includes different use cases.

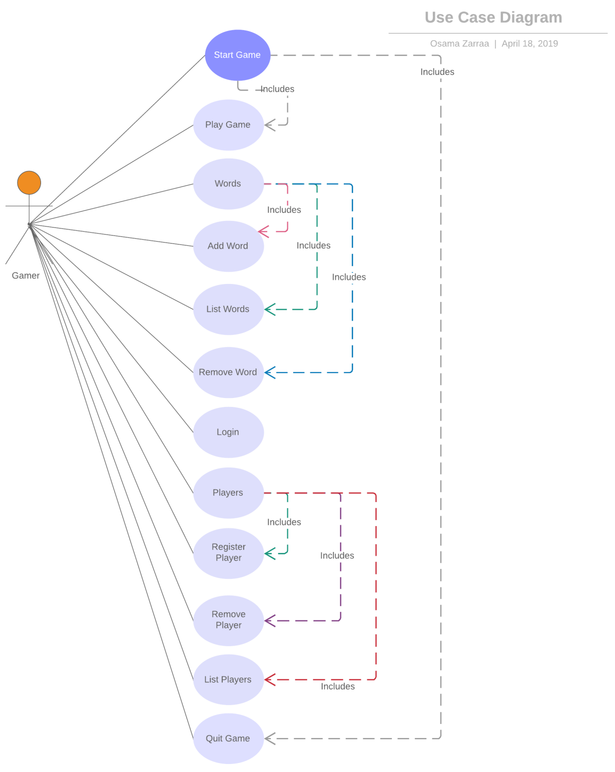


Figure 2: Use Case Diagram

8.6 State Machine Use Case 2

The figure below shows the state machine for the use case 2.

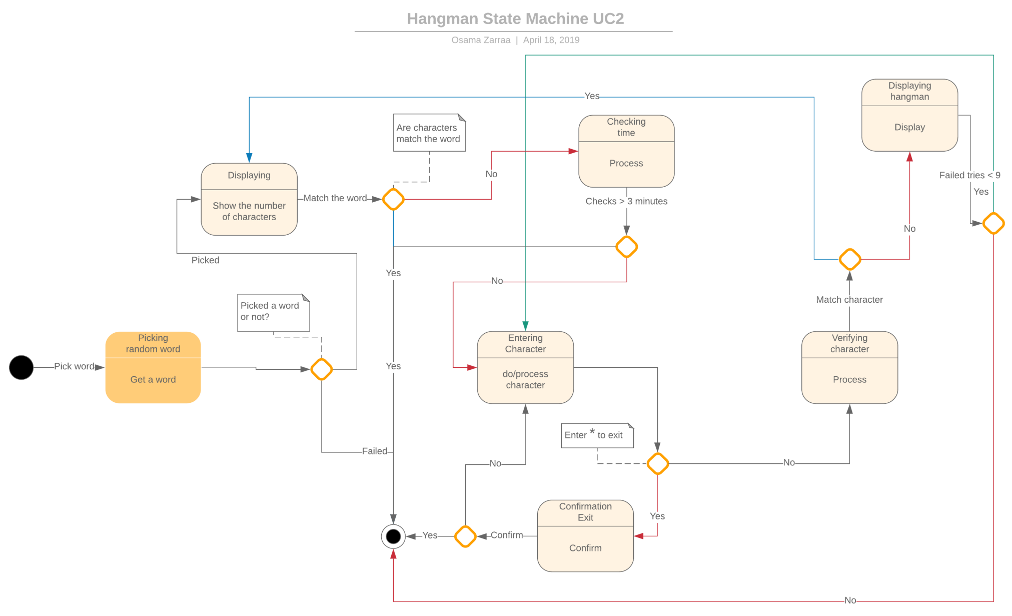


Figure 3: State machine Use Case 2

8.7 Login use case

The figure below shows the log in use case.

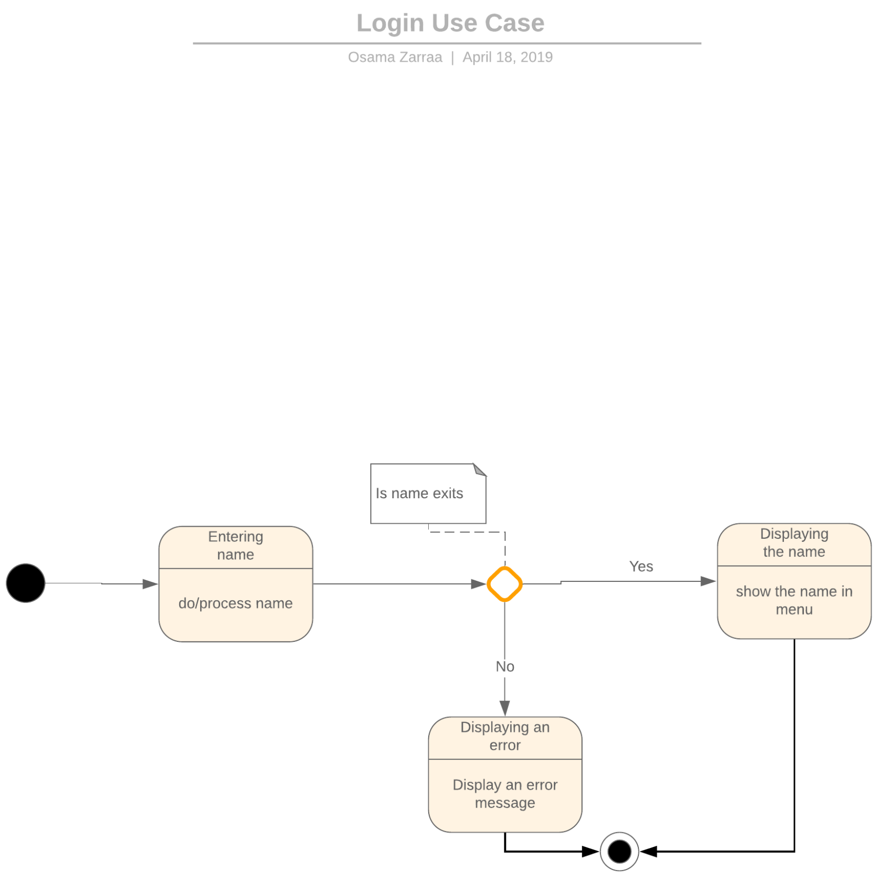


Figure 4: Login Use Case 2

9 Appendix 2

This section describes the test process of the application and starts with the plan for the test and then going through the manual and automated tests and finishes with the test report and reflection about the process.

9.1 Test Plan

9.1.1 Introduction

The Test Plan has been created to test hangman game project. It includes the objectives, what we are going to test and way. Also, it includes how to test and the time plan.

9.1.2 Motivation

The reason for selecting these tests is to check the functionality of the new features (Log in, Register Player, List of Players) are working and give the expected results. It contains 6 manual test cases and testing for 2 classes as automated tests. The manual tests check that use case scenarios are valid. While the automated tests check the code for errors and coverage.

9.1.3 Objectives

The objective is to test part of the code that was implemented the last iteration.

9.1.4 Team Members

Resource Name	Role
Osama Zarraa	Tester

9.1.5 What to test and why

It is intended to test use cases (1,2), Log in use case, Register use case and List of players use case by writing and running dynamic manual test-cases. The reason for testing these use cases is to test the fundamental functionalities of the project.

9.1.6 Scope

To test the project, manual and automated tests are made and saved.

9.1.7 Time plan

Task	Estimated	Actual hours
Manual tests	1h	1h
Unit tests	2h	90m
Running manual tests	15m	8m
Code inspection	30m	30m
Test plan	1h	2h

9.1.8 Risks

Risk	Impact	Effect	Mitigation Plan
Working alone	High	Delays in implementation during the time frame	Have and follow a strict plan
Hardware damage	Low	Loss the project	Upload the project to cloud service
Sickness	Medium	Product did not get delivered on schedule	None

9.2 Manual Tests

9.2.1 TC1

This is the manual test for the use case 1 start game to ensure that game starts without an issue.

ID	TC1
Requirement / Use Case Coverage	UC1
Name	Start game successfully
Precondition	• Java 11 is installed • The words file is existed • The players file is existed
Test Steps	1. Pass path of the file, that contains a list of words which will be used in the game, as an argument to the program. 2. Pass path of the file, that contains a list of players and their scores which will be used in the game, as an argument to the program. 3. Start the application 4. System shows main menu 5. Enter 1 to start the game
Test data	• Path for the existed words file • Path for the existed players file
Expected	The system should show the text "# Game started (Press * if you would like to interrupt the game)"
Actual	As expected
Status	Pass
Date	20190308
Comment	

9.2.2 TC2

This is the manual test for the log-in use case to ensure that player can log-in without an issue.

ID	TC2
Requirement / Use Case Coverage	Login Use Case
Name	Login successfully
Precondition	• Java 11 is installed • The words file is existed • The players file is existed
Test Steps	1. Pass path of the file, that contains a list of words which will be used in the game, as an argument to the program. 2. Pass path of the file, that contains a list of players and their scores which will be used in the game, as an argument to the program. 3. Run the application 4. System shows main menu 5. Enter 2 to log-in in 6. Write a correct player name that exists in the players data
Test data	• Path for the existed words file • Path for the existed players file
Expected	The system should show the text "# What would you like to do (PlayerName)" where PlayerName is the entered name of the player
Actual	As expected
Status	Pass
Date	20190418
Comment	

9.2.3 TC3

This is the manual test for the Log-in use case to check if system handles the player not found.

ID	TC3
Requirement / Use Case Coverage	Login Use Case
Name	Login failed
Precondition	• Java 11 is installed • The words file is existed • The players file is existed
Test Steps	1. Pass path of the file, that contains a list of words which will be used in the game, as an argument to the program. 2. Pass path of the file, that contains a list of players and their scores which will be used in the game, as an argument to the program. 3. Run the application 4. System shows main menu 5. Enter 2 to log-in in 6. Write an incorrect player name that is not existed in the players data
Test data	Path for the non-existed words file
Expected	The system should print an error message like "sorry, invalid name"
Actual	As expected
Status	Pass
Date	20190418
Comment	

9.2.4 TC4

This is the manual test for the use case 2 play game with failing due to the time limit to ensure that game has a time limit scenario.

ID	TC4
Requirement / Use Case Coverage	UC2
Name	Play game and lose due to time limit
Precondition	TC1
Test Steps	User enters one invalid character after 3 minutes from starting the game
Test data	Characters
Expected	The system should show the text “# You lost”
Actual	As expected
Status	Pass
Date	20190418
Comment	

9.2.5 TC5

This is the manual test for the register player use case to ensure that register player scenario.

ID	TC5
Requirement / Use Case Coverage	Register Player Use Case
Name	Register player successfully
Precondition	• Java 11 is installed • The words file is existed • The players file is existed
Test Steps	1. Pass path of the file, that contains a list of words which will be used in the game, as an argument to the program. 2. Pass path of the file, that contains a list of players and their scores which will be used in the game, as an argument to the program. 3. Run the application 4. System shows main menu 5. Enter 3 to enter to players menu 6. Enter 1 to start registering new player 7. Enter non-empty name for the player
Test data	Path for the existed players file
Expected	The program shows the players menu again without any error messages
Actual	As expected
Status	Pass
Date	20190418
Comment	

9.2.6 TC6

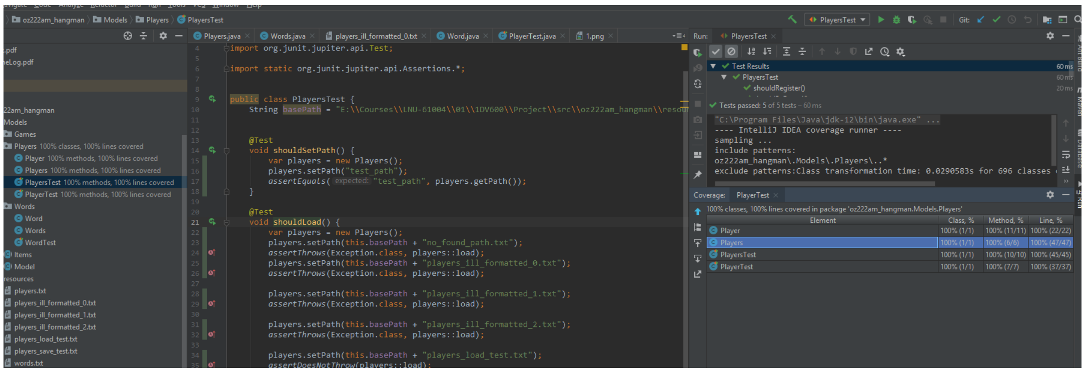
This is the manual test for the display players use case to ensure that game has a display players with scores functionality.

ID	TC6
Requirement / Use Case Coverage	List of Players Use Case
Name	Display players with scores
Precondition	• Java 11 is installed • The words file is existed • The players file is existed
Test Steps	1. Pass path of the file, that contains a list of words which will be used in the game, as an argument to the program. 2. Pass path of the file, that contains a list of players and their scores which will be used in the game, as an argument to the program. 3. Run the application 4. System shows main menu 5. Enter 3 to enter to players menu 6. Enter 3 to show players with scores
Expected	System should show players with their scores
Actual	As expected
Status	Pass
Date	20190418
Comment	

9.3 Automated Test

9.3.1 Player class

The screen-shot below shows “PlayerTest” class with six test methods. As the screen shows, all methods passed the test in the word class. Furthermore, it shows the coverage on the left side. This test is made to verify that new implemented methods (setNumberOfWin, setNumberOfLose, addWin, AddLose) are working.



```
.pdf
eLog.pdf
22am_hangman
Models
  Games
  Players 100% classes, 100% lines covered
    Player 100% methods, 100% lines covered
    Players 100% methods, 100% lines covered
    PlayersTest 100% methods, 100% lines covered
    PlayerTest 100% methods, 100% lines covered
  Words
    Word
    Words
    WordTest
Items
Model
resources
  players.txt
  players_ill_formatted_0.txt
  players_ill_formatted_1.txt
  players_ill_formatted_2.txt
  players_load_test.txt
  players_save_test.txt
  words.txt
```

```java
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;


public class PlayersTest {
    String basePath = "E:\\Courses\\LNU-61004\\01\\1DV600\\Project\\src\\oz222am_hangman\\resou


    @Test
    void shouldSetPath() {
        var players = new Players();
        players.setPath("test_path");
        assertEquals( expected: "test_path", players.getPath());
    }

    @Test
    void shouldLoad() {
        var players = new Players();
        players.setPath(this.basePath + "no_found_path.txt");
        assertThrows(Exception.class, players::load);
        players.setPath(this.basePath + "players_ill_formatted_0.txt");
        assertThrows(Exception.class, players::load);

        players.setPath(this.basePath + "players_ill_formatted_1.txt");
        assertThrows(Exception.class, players::load);

        players.setPath(this.basePath + "players_ill_formatted_2.txt");
        assertThrows(Exception.class, players::load);

        players.setPath(this.basePath + "players_load_test.txt");
        assertDoesNotThrow(players::load);
```

```
Run: PlayersTest
Test Results                60 ms
  PlayersTest               60 ms
    shouldRegister()        20 ms
Tests passed: 5 of 5 tests – 60 ms
"C:\Program Files\Java\jdk-12\bin\java.exe" ...
---- IntelliJ IDEA coverage runner ----
sampling ...
include patterns:
oz222am_hangman\.Models\.Players\..*
exclude patterns:Class transformation time: 0.0290583s for 696 classes
```

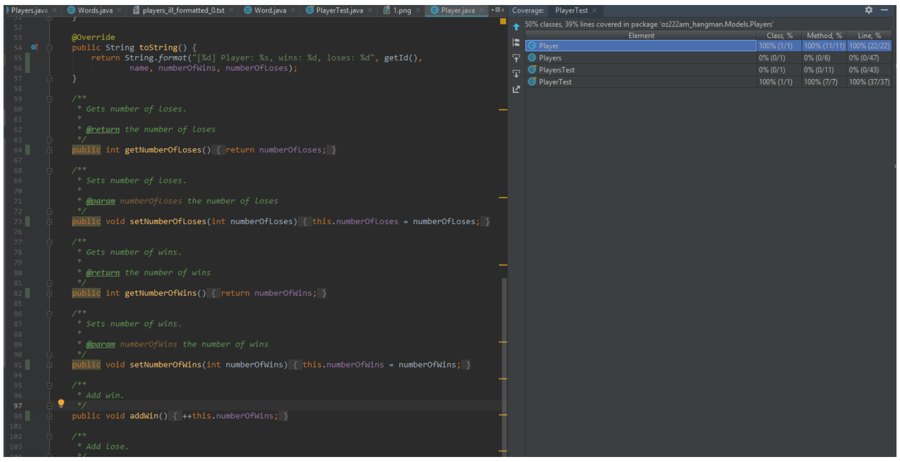
Coverage: PlayerTest

100% classes, 100% lines covered in package 'oz222am_hangman.Models.Players'

Element	Class, %	Method, %	Line, %
Player	100% (1/1)	100% (11/11)	100% (22/22)
Players	100% (1/1)	100% (6/6)	100% (47/47)
PlayersTest	100% (1/1)	100% (10/10)	100% (45/45)
PlayerTest	100% (1/1)	100% (7/7)	100% (37/37)

9.3.2 Players class

The screenshot below shows “PlayersTest” class, it includes five test methods and the code coverage of this class “100%” on the left side of the screen. All the methods have passed.



```java
    @Override
    public String toString() {
        return String.format("[%d] Player: %s, wins: %d, loses: %d", getId(),
                name, numberOfWins, numberOfLoses);
    }

    /**
     * Gets number of loses.
     *
     * @return the number of loses
     */
    public int getNumberOfLoses() { return numberOfLoses; }

    /**
     * Sets number of loses.
     *
     * @param numberOfLoses the number of loses
     */
    public void setNumberOfLoses(int numberOfLoses) { this.numberOfLoses = numberOfLoses; }

    /**
     * Gets number of wins.
     *
     * @return the number of wins
     */
    public int getNumberOfWins() { return numberOfWins; }

    /**
     * Sets number of wins.
     *
     * @param numberOfWins the number of wins
     */
    public void setNumberOfWins(int numberOfWins) { this.numberOfWins = numberOfWins; }

    /**
     * Add win.
     */
    public void addWin() { ++this.numberOfWins; }

    /**
     * Add lose.
```

Coverage: PlayerTest

50% classes, 39% lines covered in package 'oz222am_hangman.Models.Players'

Element	Class, %	Method, %	Line, %
Player	100% (1/1)	100% (11/11)	100% (22/22)
Players	0% (0/1)	0% (0/6)	0% (0/47)
PlayersTest	0% (0/1)	0% (0/11)	0% (0/43)
PlayerTest	100% (1/1)	100% (7/7)	100% (37/37)

This test is made to check that the functionality of the Players class meets the requirements (save, load, register, remove).

9.4 Test Report

Test traceability matrix and success

Test	UC1	UC2	Login UC	Refister Player UC	List of Play-ers UC
TC1	1/OK				
TC2			1/OK		
TC3			1/OK		
TC4		1/OK			
TC5				1/OK	
TC6					1/OK
Coverage & Success	1/OK	1/OK	2/OK	1/OK	1/OK

Automated unit test coverage and success

Test	Player	Players
PlayerTest	100%/OK	
PlayersTest		100%/OK
Coverage & Success	100%/OK	100%/OK

9.5 Reflection

Writing a test plan was not a very hard task since I found many templates on the internet that I can follow. However, the strange thing that I found is that we must submit all the reports (test plan, test report, manual test cases, and personal reflection) in one PDF file. Thus, I had confusion with organizing my final PDF document. I found many templates for the manual and automated test cases. So, I found no difficulties writing them. The estimated time that I sat in my time log was reasonable, well estimated and close to the actual.