HANGMAN DEVELOPMENT PROCESS

Author: Donald Appiah

Institution: Linnaeus University
Course: Software Technology
GitHub:

Table of Contents

1.Revision History

Date	Version	Description	Author
2/02/2019	V1	First Sketch	Donald Appiah
6/02/2019	V2	First Iteration	Donald Appiah
20/03/2019	V3	Revision of Document	Donald Appiah
21/03/2019	V4	Final sketch	Donald Appiah

2. General Information

Project Summary	
Project Name: Legend Hangman	Project ID: L5111_v1.0
Project Manager: Donald Appiah	Main Client: Hangman enthusiasts can know the paper version of the game and want to try a computer version of the game.
Key Stakeholders: • Executives • Project Manager • End-User	
Executive Summary: The project consists of creating the "Hangman" game. Idea behind the game is that the player is going to guess a word by guessing a letter by letter. If a letter given is part of the predefined word, then the letter will be shown at the right position, Else if the player guesses a letter wrong then the game builds the part of a man getting hanged till all the guesses are done. The game will be implemented in a text-based fashion with java code.	

3. Vision

This project consists of creating the "Hangman" game. The basic idea for this game is that the player is going to guess a word by guessing letter by letter. It will be implemented in a text-based fashion with Java code language. If a letter given is inside the predefined word, then the letter will be showed at the right position of the word. If the player guesses a wrong letter, the game is adding a part of a man getting hanged. The player can have 6 letters wrong before the man is hanged and therefore losing the game. (ground pole, head body, left arm, right arm, left leg, right leg).

The game will also let the user a letter or try a word. If they get wrong letter it will be considered the same and they will therefore go to another part of the hanged man. The players will also be able to create an account by entering a username and a password. They will be able to log in and see the amount of times they won and other statistics that could be important for them.

Reflection on writing a vision statement: Writing a vision statement should. emphasis the goal of the project and summarize what everyone involved in the project should know and aim for. I noticed the content, style and idea behind vision statements in different contexts can vary quite a bit. For this project, we were supposed to write a description of the whole system, but in many other examples. If the goal of the vision statement is to establish a common ground between every team member, then I agree that a more objective and descriptive text is more useful. But if the goal is to captivate and capture outside interest, then I understand the need for more concise, appealing and innovative messages.

4. Project Plan

The project plan of the application will first start with planning on what will be needed for the project, like resources, hardware and software requirements how the project should be like, the risk that might pop up while the project is in progress. Next stage will be about how I will model my application how it is going to look like and function and also designing the application with codes and algorithm.

Reflection:

The Project Plan help me to focus more into what could be done and make a more realistic plan. It also helped mapped out my ideas to know exactly what needed to be done and realized what couldn't be done in the given timeframe.

4.1 Introduction

For this project, the "Hangman" game will be created. It will be implemented in a text-based with Java code language. The basic idea for this game is that the player is going to guess a word by guessing letter by letter. If a letter given is inside the predefined word, then the letter will be showed at the right position of the word, else if the player guesses a wrong letter, the game will add a part of a man getting hang. The layer can have 8 letters wrong before the man is hanged and therefore losing the game. (ground pole, head, body, left arm, right arm, left leg and right leg).

4.2 Justification

The application is a project in the course 1DV600 (Software Technology) within the computer science department. The application should be made to meet the requirements of the users.

The game will be an educational tool for both adults and children it will help build how fast they think and how they know their word by guessing them. The game will be a good training for infants starting to learn how to spell words and this will help them have a better future.

4.3 Stakeholders

End user – this includes the individuals going to make use of the project's outcome. This means the developer should meet their requirement by implementing the necessary structured code for better performance and maintainability.

4.4 Resources

Resources for the application will be my computer, books. Javafx jars and IDE software to run my application also a GitHub account will be needed to update my project repository.

Time spent on literature for creating the game are also measured to be part of my resources

Literature	**Activity**	**Time**
Introduction to java. *Y Liang*	Creating classes	1hour 20mins
Introduction to java. *Y Liang*	JavaFx	4hours 15mins

4.5 Hard- and Software Requirements

My requirement will be a functioning computer and
Eclipse IDE for Java Developers
Version: 2018-12 (4.10.0) Build id: 20181214-0600 to run my codes.

To Develop the Game:

PROCESSOR: Intel Core i5 7th Gen

HARDRIVE CAPARCITY: SSD 250GB

RAM: 8GB

OPERATING SYSTEM: macOS Mojave

Software:

Java 8 SDK

Eclipse IDE

To run The Game:

JAVA 8 or higher

MacOS 10 or higher

RAM: Minimum 4 GB

PROCESSOR: Minimum 2,7 GHz Intel Core i5

4.6 Overall Project Schedule

- Jan. 28 – Feb. 6 (Project plan Creation, Vision, Risk Analysis)

- Feb. 7 (Skeleton Code Creation, Time Logs)
- Feb. 8 (Documenting Iteration 1 / Sprint 1 Completion)
- Feb. 10 – 12 (Modeling new Features UML Features, Menu Creation, Duplicate Letter Plan)
- Feb. 13 - 17 Implementing Features
- Feb. 18 – 20 (Documenting Iteration 2 / Sprint 2 Completion)
- Feb. 25 – 28 (Code Inspection, Test Planning)
- Mar. 1 – 3 (JUnit Test execution)
- Mar. 4 – 6 (Documenting Iteration 3 / Sprint 3 Completion)
- Mar. 10 – 13 (Project plan review)
- Mar. 14 – 15 (Adding final features [Display Rules], execute manual Testing)
- Mar. 16 – 22 (Documenting Iteration 4 / Sprint 4 Completion)

4.7 Scope, Constraints and Assumptions

Scope: The project main part is to make sure the main parts of the game hangman functions as planned. The player must be able to guess a single letter of choice and the game will then show the hidden letter. Having different difficulties of the word, example is word with 3-5 letters are easy, while 6-8 are medium and 7-10 or more will be difficult.

The developer is supposed to implement:

- A Graphical Interface
- A place to write the guesses
- The wrong guesses should be showed on the screen
- A hint on how many letters there is on the word should be showed on the screen.
- A graphical representation of the Hangman. The graphical representation should change every time that the player guesses a wrong letter.
- Keep track of the wrong guesses. 7 wrong guesses allowed.
- Text that appears when the player has won or lost.
- A menu where the player can choose to go play or to exit the game.

Constraints:

- The Game runs into an IDE.
- Only 1 player can play.

Assumption: The projects assumes that users are able to perform basic task with their personal computer, the game run a release from jar archive.

Reflection on writing a project plan: Writing a project plan is an essential step when developing a project. The project plan is created at the first part of the project with the goal of structuring the work that has to be done in different activities, assessing the time and effort required for each stage. This project plan is reassessed and updated throughout the entire project, with this project I understand the enormous asset it is in a project, especially if it is a complex project

5. Iterations Plan

5.1 Iteration 1

With this iteration a skeleton code was made on how the game will be like and documentation of the project will be documented, such as the general information of the game, the vision, time estimation and project plan.

Task	Estimation Time	Due Date
Read about Project Planning	2h	Week 6
Write Vision	20min	Week 6
Write Introduction for Project Plan (PP)	10 min	Week 6
Write Justification for PP	15 min	Week 6
Write Stakeholders for PP	20 min	Week 6
Write Resources for PP	5 min	Week 6
Write Hardware and Software for PP	20 min	Week 6
Write Overall Project Schedule for PP	30 min	Week 6
Write Scope, constraints and assumptions for PP	1h	Week 6
Write Iteration 1 Table	20 min	Week 6
Read about Risk Analysis	3h	Week 6
Write Lists of risks	20 min	Week 6
Write Strategies for risks	20 min	Week 6
Create time log	5 min	Week 6
Create skeleton code	5h	Week 6

5.2 Iteration 2

This iteration deals with how the game will be played by using UML diagram to describe it. Case diagram, state machine and a fully dressed diagrams will be implemented.

Task	Estimation Time	Due Date
Create Class Diagram	15 min	Week 8
Write Fully Dressed Use Case	30 min	Week 8
Create Use Case Diagram	15 min	Week 8
Create State Machine	30 min	Week 8
Implement Code	60 min	Week 8

5.3 Iteration 3

This iteration is the third stage of the project and how the game functions and methods that has been implemented will be tested manually for the functions and bottoms and JUnit for the methods using eclipse to run the test. With this test all the bugs in the code should be found and fixed. All the test that will be tested will be documented.

Task	**Estimation Time**	**Due Date**
Create Manual Test Case	1h	Week 10
Write JUnit test	1h30min	Week 10
Running test	40 min	Week 10
Checking the code	30 min	Week 10
Write Report of Testing	30 min	Week 10

5.4 Iteration 4

This is the final iteration of the project, which means everything has been documented, the UML diagrams are in clear view of how the game functions, testing has been made, the vision of the game and all included in the game has been reiterate and the project is now a whole project.

Task	**Estimation Time**	**Due Date**
Update Planning Document	120	Week 12
Write Reflections	30	Week 12
Implement Last Piece of Code	60	Week 12
Create Tests for the Project	30	Week 12
Run Tests	20	Week 12
Write Final Report	90	Week 12

6. Risk Analysis

6.1 List of risks

List the identified risks and specify, as far as possible, the probability of them happening as well as the impact they would have on the project.

Phase	Risk Description	Scale
Iteration #1	Unable to connect to GitHub	moderate
Iteration #1	Unable to complete iteration #1	moderate
Iteration #2	Drawing the UML diagram	low
Iteration #3	Testing error with JUnit	high

6.2 Strategies

- ***Unable to connect to GitHub***
  Solution: Check online YouTube and online websites for tutorials.

- ***Unable to finish coding.***
  Solution: Check on Stack Overflow and YouTube for ideas & implementation.

- ***Unable to draw the UML diagram*.**
  Solution: Fixed it by searching online solutions and visiting websites.

- ***Testing error with JUnit.***
  Solution: Got solution through slack from my supervisor when I requested for help on how to solve it.

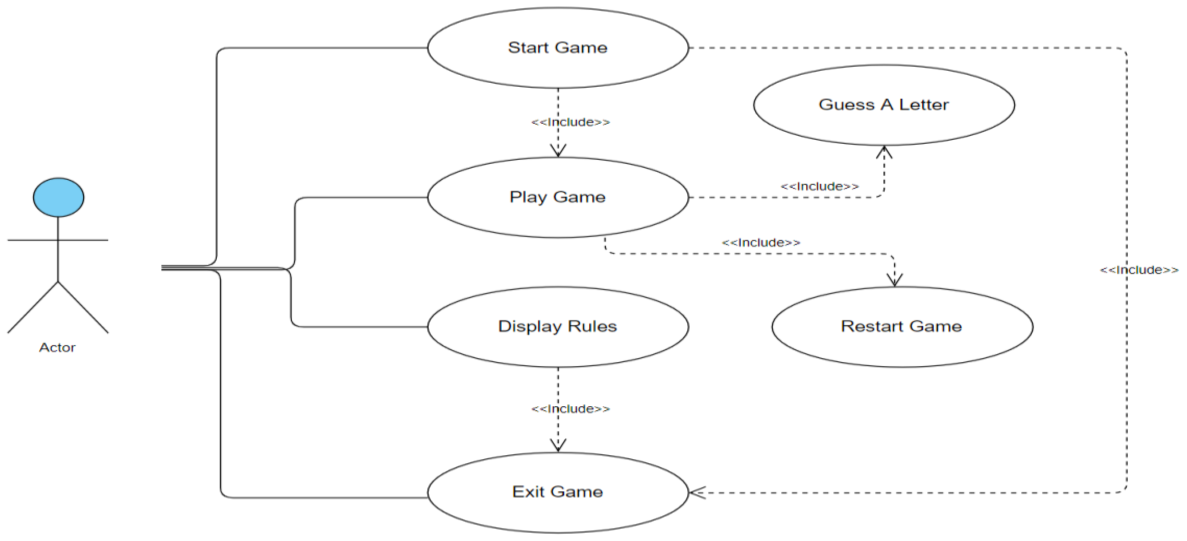
Reflection on risk analysis: When planning a project, it is very necessary to plan on time for the to ignore risks that can happen. This allows the project manager to take certain strategies for handling these different risks in order to minimize or get rid of their impact on the project.

7. Modeling with UML

Modelling is the successor of the project plan process and the precursor to implementation/coding. Undoubtedly, it is the best way to communicate with the team of the project since it is conveyed through diagrams and not through oral communication which is highly subjective. The UML (Unified Modelling Language) used for this project is:

- Use Cases
- State Machine
- Class Diagrams

7.1 Use Case Model



7.1.1 Fully Dressed Use Case

UC 1 Start Game

Precondition: None.

Postcondition: The game is started, and the menu is shown.

1. The user starts the program.

UC 2 Play Game

Precondition: The game is started.

Postcondition: Game is over, and the gamer plays again.

Main Scenario:

1. The user types any button.
2. The system generates a random word.
3. The user guesses a letter.
4. The system checks if the typed letter is contained in the word and updates the system by displaying a message in the screen.

Repeat from step 3 until the player has found the word.

5. The user has now the option to re-play the game or exit.
6. The system executes the command that the user gave from step 5.

Alternative Scenario:

1.1 The user wants to exit the game. (See UC 2 Exit Game)

3.1 Invalid letter input.

   1. the system reduces the remaining guesses if the typed guess is a letter or displays a message if the typed guess is not a letter.

5.1 The player chooses to exit the game (See UC 2 Exit Game)

UC 3 Exit Game

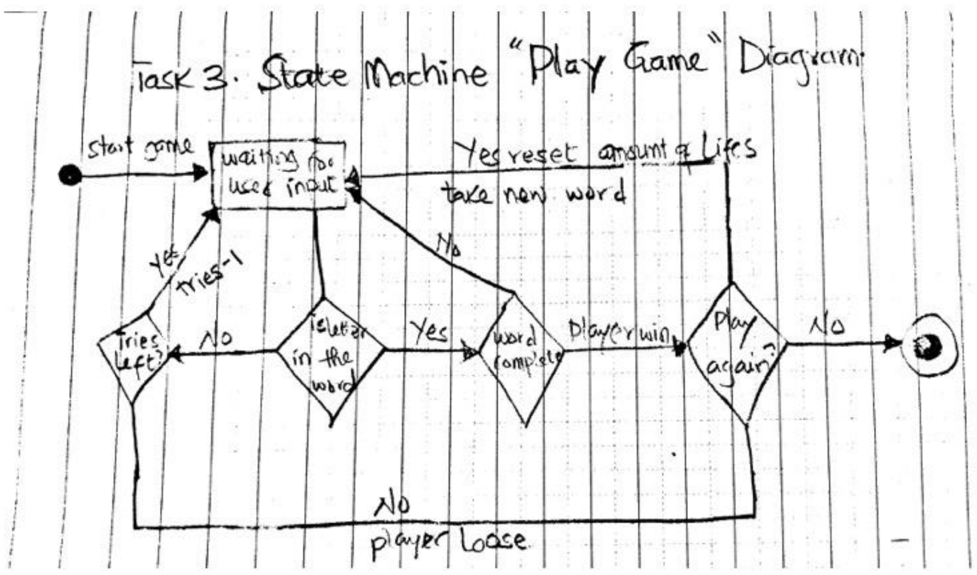
Precondition: Program is launched.

Postcondition: The game is terminated.

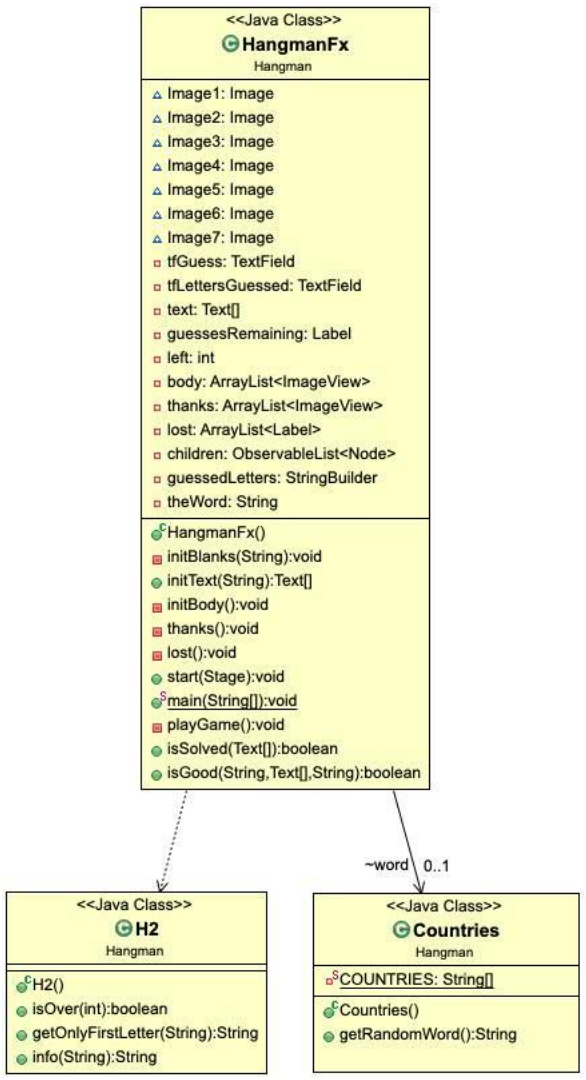
   1. The user types a button to exit the game.

The system displays a “Goodbye” message.

7.2 State Machine



7.3 Class Modelling



8.Testing

8.1Test Plan

Aim:
The aim of this iteration is to test the code that has been implemented in the last iteration focusing on the different cases in the program, finding bugs, testing methods, buttons and any other type of hindrances the code might have. In order to have an easy approach and understanding towards the maintenance of the code in the last iteration.

What to test?

Use-case 2 will be tested through writing and running manual test-cases. It has already been implemented and ready to be tested. And we want to make sure that the game is doing what it is supposed to do. We are going to specifically test the methods getOnlyFirstLeter();,isOver (); and info();. These three methods are important methods that make the application works and therefore the methods have been selected to be tested.

How to test?

A manual testing and automated testing (using Junit) to be able to determine if the application is working according to its requirements.

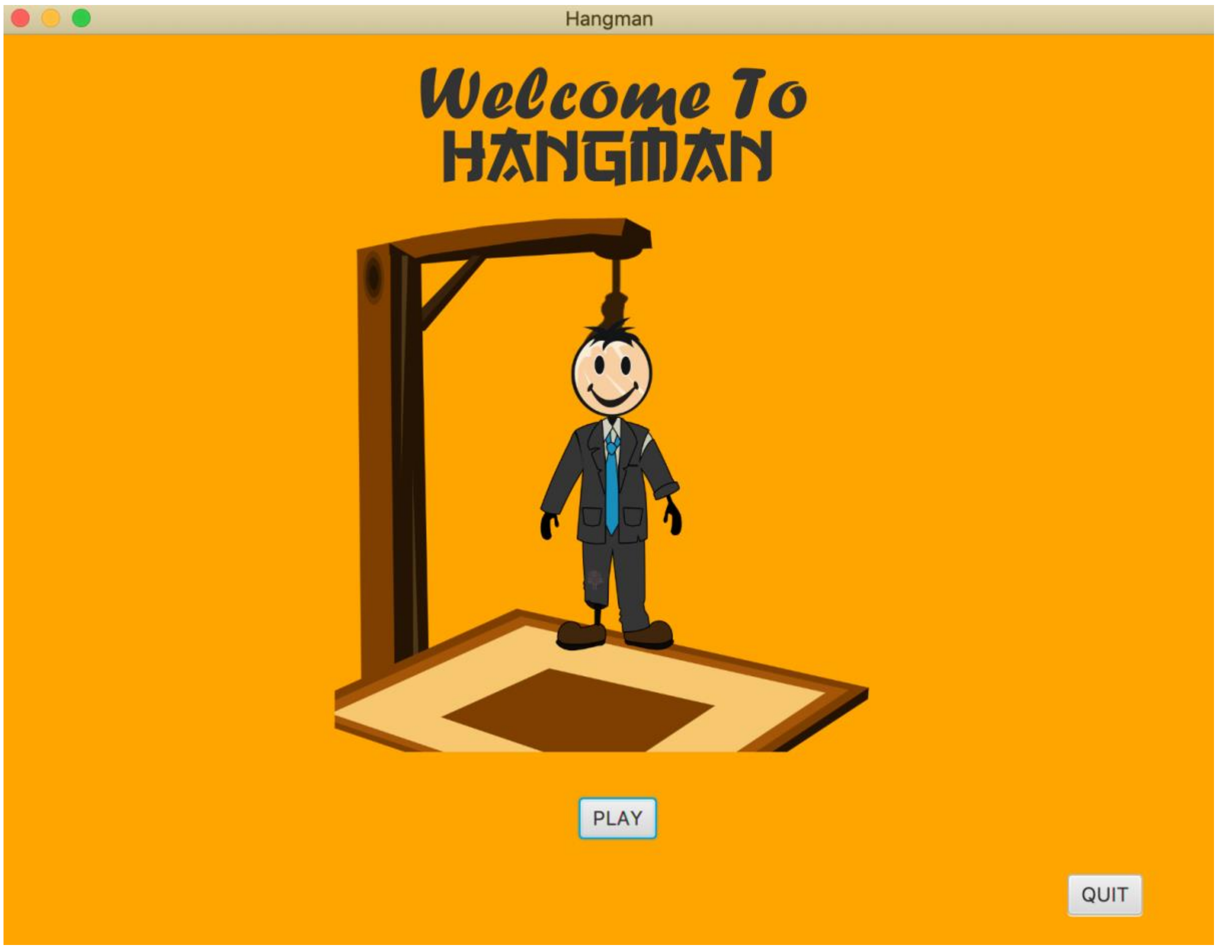
Time Plan:

Task	**Estimated**	**Actual**
Manual Taste Case	1h	30min
Unit Tests	1h 30min	2h
Running Manual Tests	30min	20min
Code Inspection	30min	30min
Test Report	2h30min	1h
Total	6h	4h20min

8.2 Manual Test Cases

TC1.1 Play

Requirement (use-case): UC1 Start Game Scenario: Click on the button “Play”



This is the main scenario of UC1, in which the player clicks on the button “Play” in order to continue to UC2 (game playing).

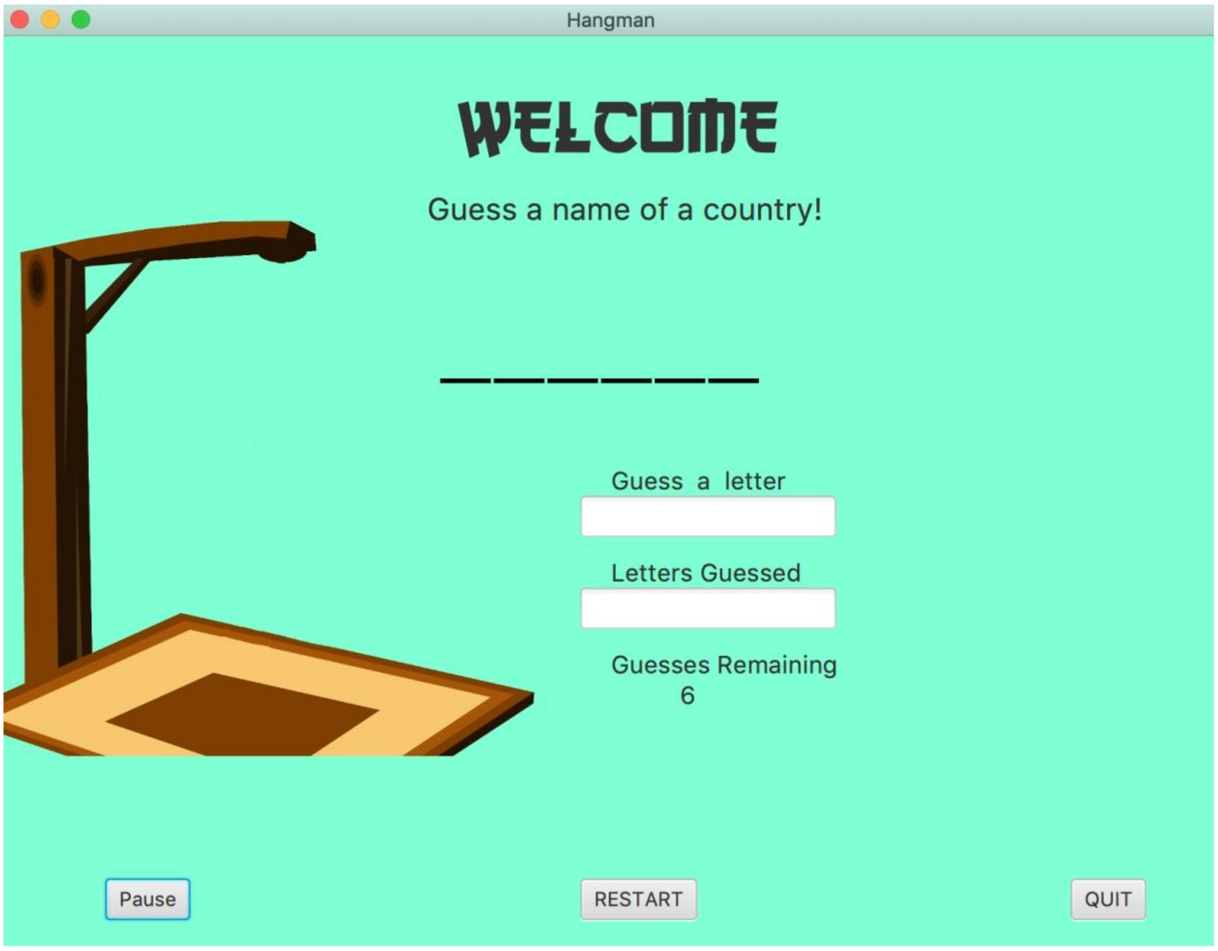
Precondition: Run a release from jar archive.

Test Steps:

- Run a release from jar archive.
- System shows the game menu
- Player clicks on the button “Play”

Expected:

• The system will then redirect the player to the UC2 game playing scenario where the player sees instructions on what to do, example is what categories of words to be guessed.



Results

Did the Test succeed: *YES*.

Comments: ....

TC1.2 Quit

Requirements (use-case): UC1 Start Game

Description/ Scenario: Click on button "Quit". It is an alternate scenario where the user decides to terminate the program and quit the game.

Precondition: Run a release from jar archive.

Test Steps:

- Run a release from jar archive.
- System shows the game menu
- Player clicks on the button “Exit”

Expected:

• The system then terminates the program.

Results

Did the Test succeed*: YES*.

Comments: ....

TC1.3.1 Pause

Requirements (use-case): UC1 Start Game

Description: Click on button “Play” the game is then paused and takes the player back to the main menu UC1 and the player can click on” Play” or decide to terminate the program with a click on the “Quit” button.

Precondition: User decided to click on “Play” button from main menu in UC1.

Test Steps:

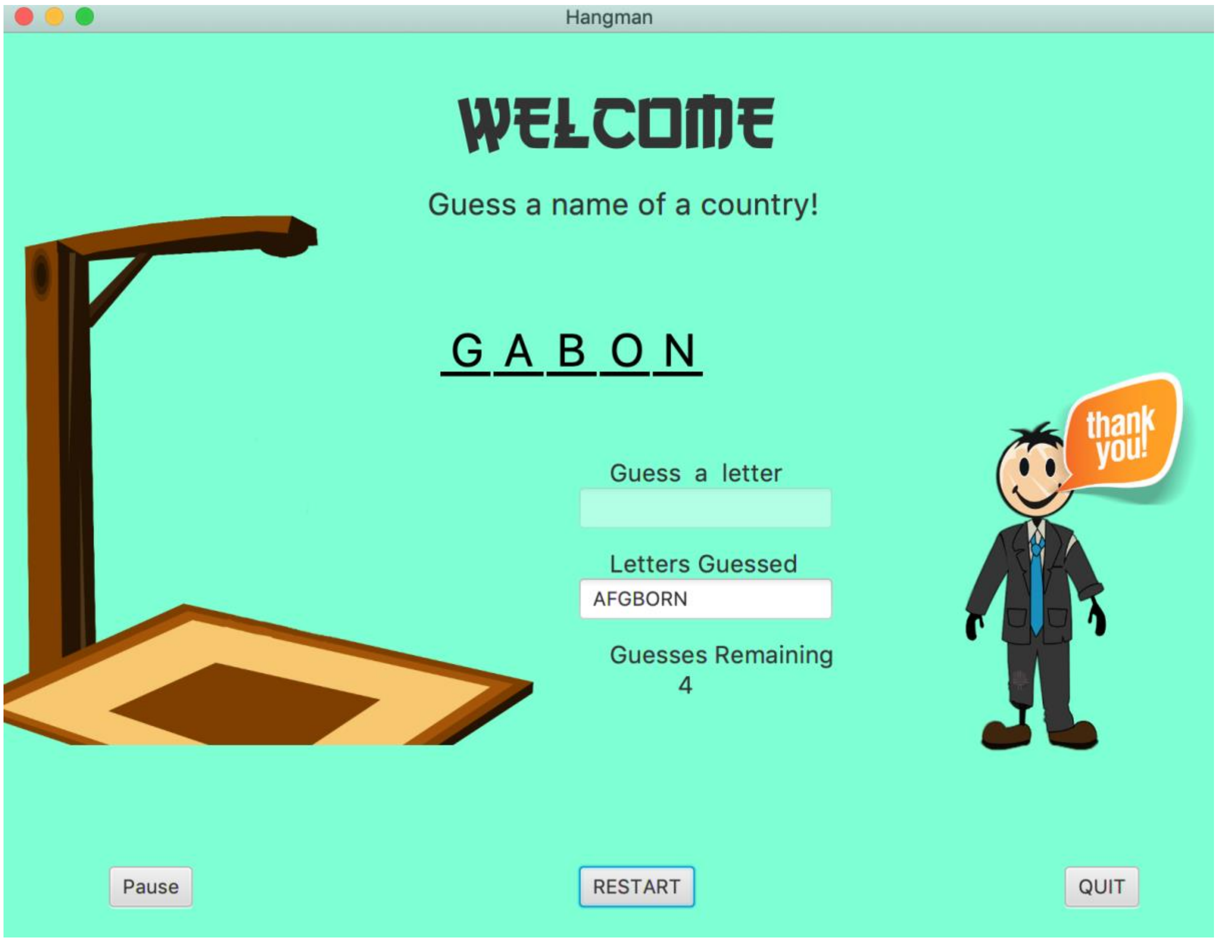
• User clicks on “Pause” button from the UC2.

Expected:

• The system redirects the user to the UC1 main menu.

TC2.1 Win Game
Requirements (use-case): UC2 Play Game

Description: In this test case if the player manages to guess all the right letters of the word without using all guesses remaining and win, Eventually the game will display an image with description thank you!



Precondition: UC1 must have been started and the button played must have been clicked in order to get to the UC2 scenario.

Test Steps:

- The player gets the instructions.
- The player started by inserting letters in such order:
- Inserts: A
- Inserts: F
- Inserts: G
- Inserts: B
- Inserts: O
- Inserts: R
- Inserts: N

Expected:

- The “hangman” gets saved and off the hanging.
- An image is displayed: “Thank you!”

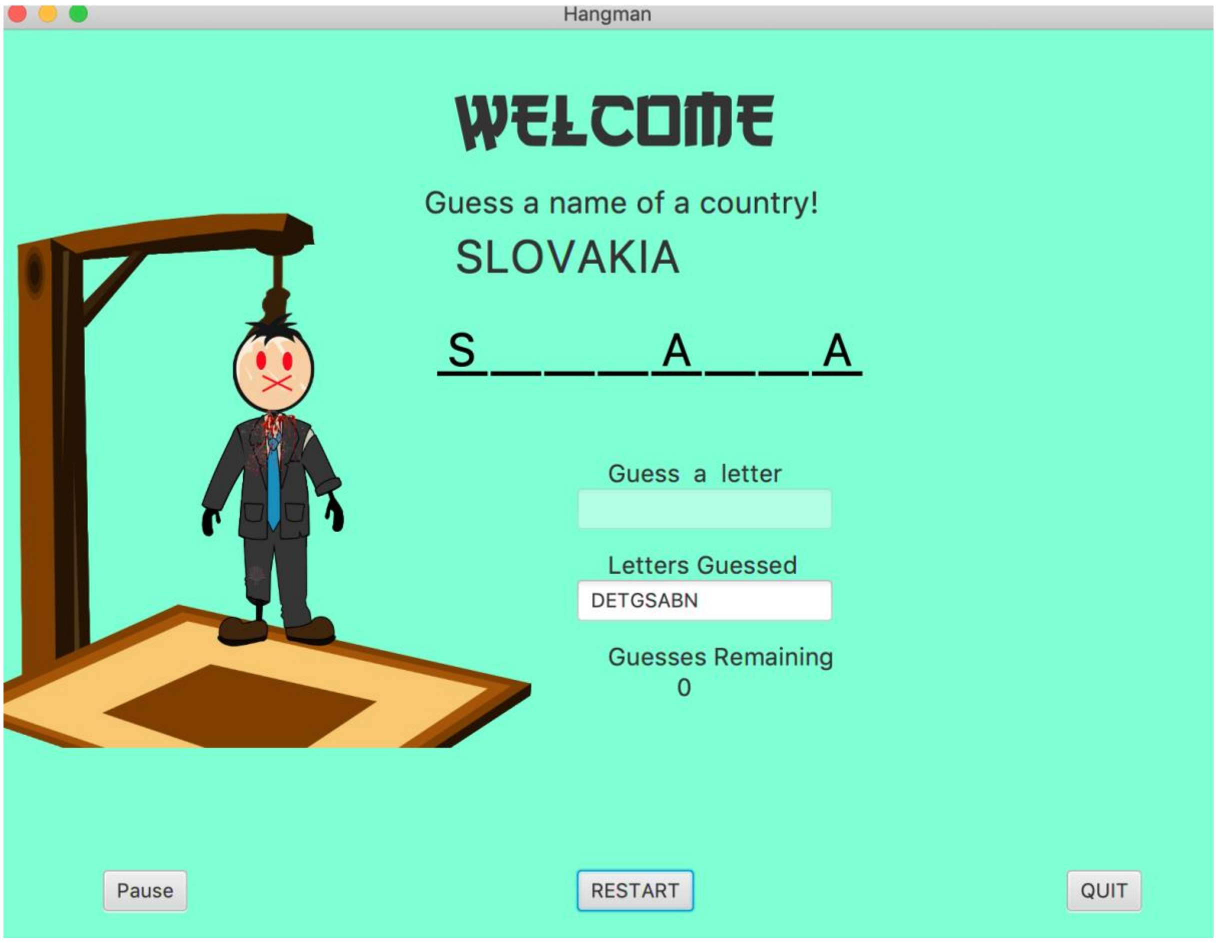
Results

Did the Test succeed*: YES*.

Comments: ....

- **TC2.1 Lose Game**
  Requirements (use-case): UC2 Play Game

Description: In this test case if the player does not guess all the right letters of the word and uses all guesses remaining, the hangman dies and the word which was to be guessed will be displayed.



Guess a name of a country!

SLOVAKIA

S _ _ _ A _ _ A

Guess a letter

Letters Guessed

DETGSABN

Guesses Remaining

0

Precondition: UC1 must have been started and the button played must have been clicked in order to get to the UC2 scenario.

Test Steps:

- The player gets the instructions.
- The player started by inserting letters in such order:

- Inserts: D
- Inserts: E
- Inserts: T
- Inserts: G
- Inserts: S
- Inserts: A
- Inserts: B
- Inserts: N

Expected:

- The “hangman” gets killed.
- The unguessed word is shown to the player.

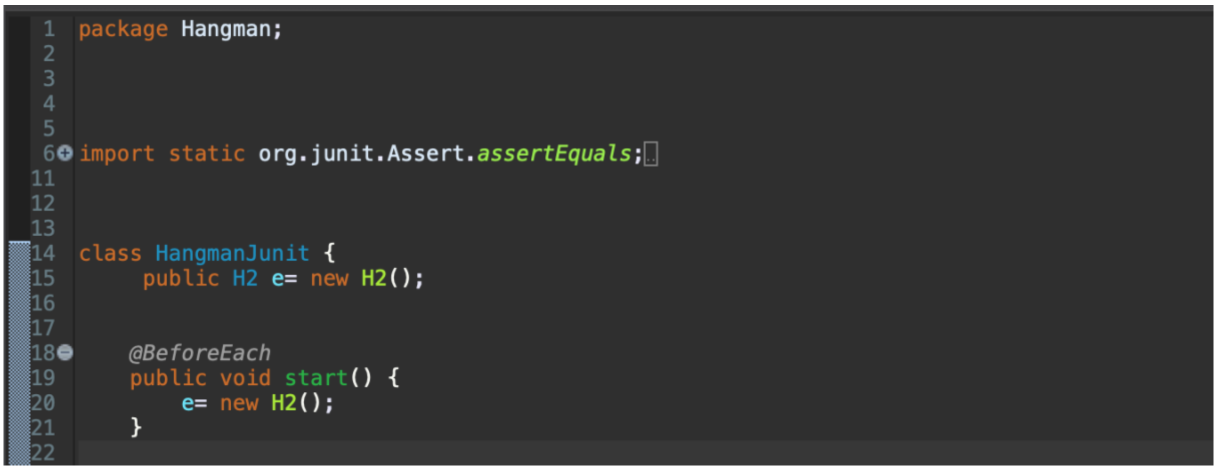
Results

Did the Test succeed*: YES*.

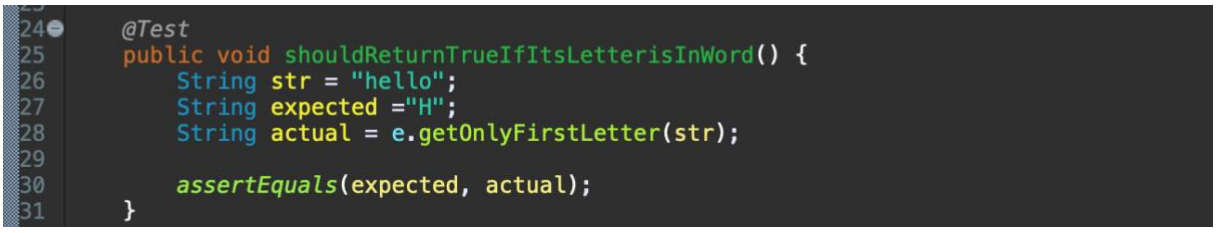
Comments: ....

8.2 Unit Tests

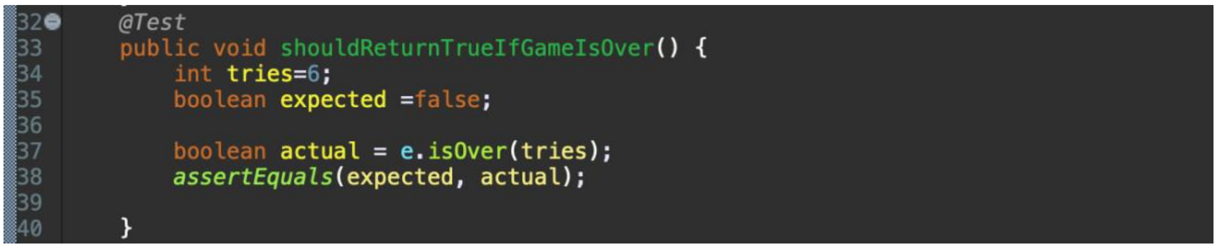
JUnit Test Class:


```
package Hangman;



import static org.junit.Assert.assertEquals;


class HangmanJunit {
     public H2 e= new H2();


    @BeforeEach
    public void start() {
        e= new H2();
    }

```

Automated Unit test 1: getOnlyFirstLetter();

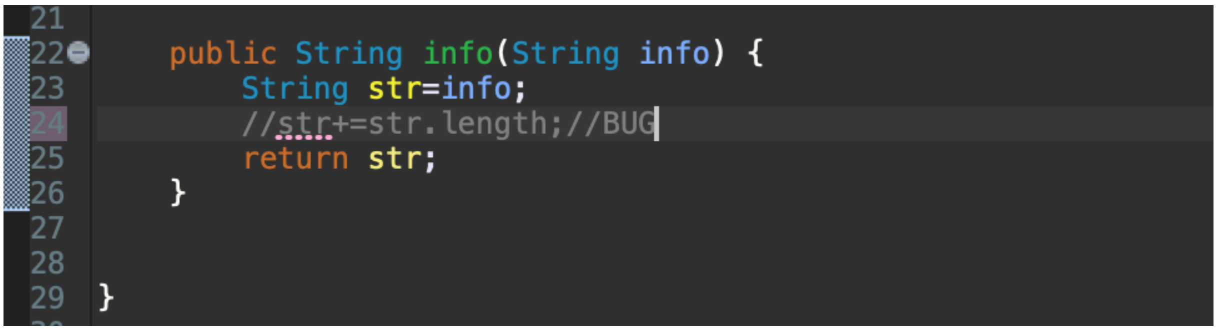

```
    @Test
    public void shouldReturnTrueIfItsLetterisInWord() {
        String str = "hello";
        String expected ="H";
        String actual = e.getOnlyFirstLetter(str);

        assertEquals(expected, actual);
    }
```

Automated Unit test 2: isOver();


```
    @Test
    public void shouldReturnTrueIfGameIsOver() {
        int tries=6;
        boolean expected =false;

        boolean actual = e.isOver(tries);
        assertEquals(expected, actual);

    }
```

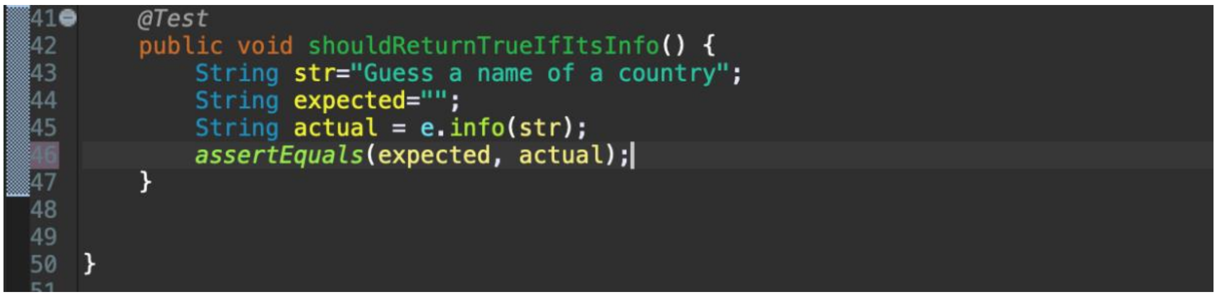
Automated Unit test 2: info()//BUG;

Code:



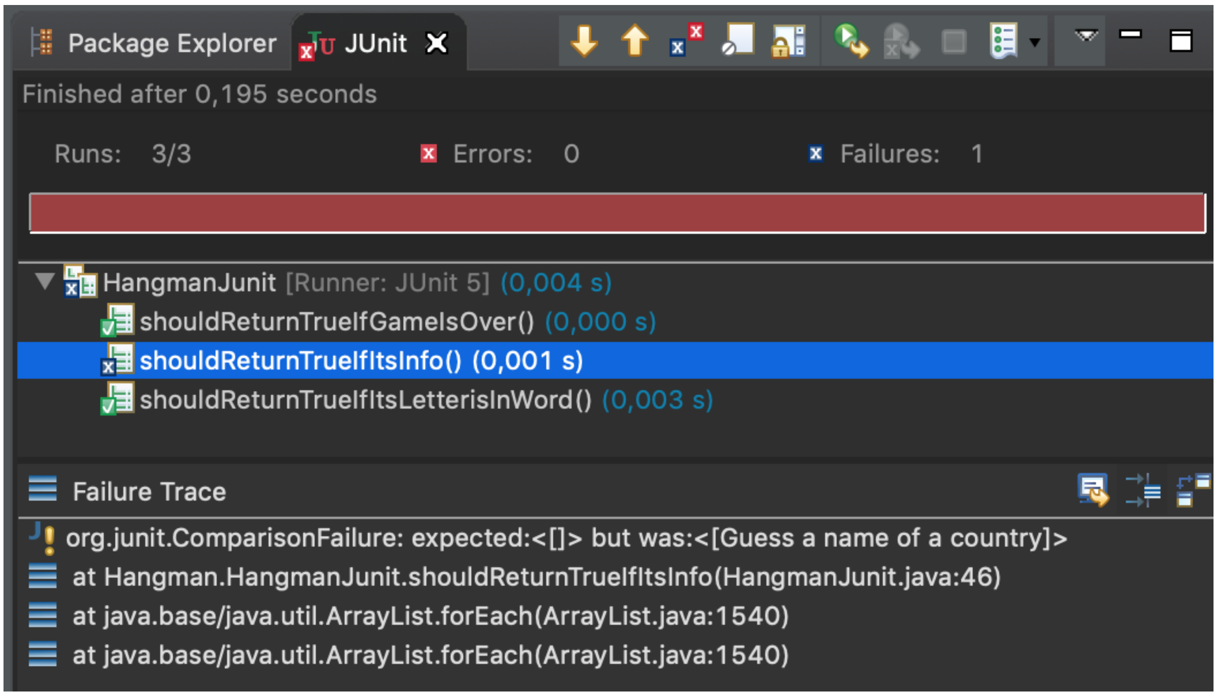
```
    public String info(String info) {
        String str=info;
        //str+=str.length;//BUG
        return str;
    }


}
```

Test:



```
    @Test
    public void shouldReturnTrueIfItsInfo() {
        String str="Guess a name of a country";
        String expected="";
        String actual = e.info(str);
        assertEquals(expected, actual);
    }


}
```

Test Result:



```
Finished after 0,195 seconds

Runs:   3/3          Errors:   0          Failures:   1

HangmanJunit [Runner: JUnit 5] (0,004 s)
    shouldReturnTrueIfGameIsOver() (0,000 s)
    shouldReturnTrueIfItsInfo() (0,001 s)
    shouldReturnTrueIfItsLetterisInWord() (0,003 s)

Failure Trace
org.junit.ComparisonFailure: expected:<[]> but was:<[Guess a name of a country]>
at Hangman.HangmanJunit.shouldReturnTrueIfItsInfo(HangmanJunit.java:46)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1540)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1540)
```

The first two methods were successful. But the test method for “shouldReturnTrueIfItsInfo()” did not work as expected.

Reflection:

Through this Testing iteration I have acquired a new understanding on why this should be done. It won’t only provide the player useful information on how the program should look like but also provides me (the programmer) different ways on the development of software and product. Different testing approach provide different results to be analyzed and certain information. The manual testing helps the programmer and the player to know what is actually supposed to happen in the program, if anything goes not according to the test case then the program must be analyzed once more and looked for the specific problem causing the hindrances. Unit testing on the other hands provides actual information of the skeleton code of the program and helps the developer identify specific hindrances/bug location in the program. Which makes it much easier to approach uncertainties within the software. Being able to fix and correct problems. I will take this into account for the final iteration of this project and for my future projects.

9. Time logs

Assignment 1 – Time Log

Task to Do	Time Estimated (min)	Time Taken (min)
Read about Project Planning	120	130
Write Vision	20	20
Write Introduction for Project Plan (PP)	15	20
Write Justification for PP	15	10
Write Stakeholders for PP	20	15
Write Resources for PP	15	10
Write Hardware and Software for PP	20	10
Write Overall Project Schedule for PP	30	50
Write Scope, constraints and assumptions for PP	40	30
Write Iteration 1 Table	30	30
Read about Risk Analysis	180	160
Write Lists of risks	20	30
Write Strategies for risks	20	20
Create time log	15	20
Create skeleton code	200	500

Assignment 2 – Time Log

Task to Do	Time Estimated (min)	Time Taken (min)
Create Class Diagram	15	20
Write Fully Dressed Use Case	30	40
Create Use Case Diagram	15	15
Create State Machine	30	70
Implement Code	60	90

Assignment 3 – Time Log

Task to Do	Time Estimated (min)	Time Taken (min)
Write Manual Test Case	60	90
Create JUnit test	90	120
Running test	40	20
Checking the code	30	25
Write report of testing	30	60

Assignment 4 – Time Log

Task to Do	Time Estimated (min)	Time Taken (min)
Update Planning Document	120	240
Write Reflections	30	40
Implement Last Piece of Code	60	30
Create Tests for the Project	30	30
Run Tests	20	15
Write Final Report	60	120