SOFTWARE DEVELOPMENT PROJECT TEMPLATE

Xiaohe Zhu

Date 2020-02-02

xz222az@student.lnu.se

Contents

1 | Revision History

Date	Version	Description	Author
2020-2-2	1	Initial version (Iteration 1)	Xiaohe Zhu
2020-2-20	2	Update iteration 2 UML diagrams and document: use case diagram, state machine diagram and class diagram Basic implementation	Xiaohe Zhu
2020-3-5	3	Update iteration 3 Unit test	Xiaohe Zhu
2020-3-20	4	Update the final version of the project Implement extended code + Log in + Create account Update UML corresponding document +Use Case Model + Use Case Diagram +Class Diagram +State Machine Diagram Update test	Xiaohe Zhu

2 | General Information

Project Summary	
Project Name	Project ID
Hangman Game	Hangman
Project Manager	Main Client
Xiaohe Zhu	University
Key Stakeholders	
Developer Project manager The end-user Tester Promoter	
Executive Summary	
This project aims to develop software that can implement the Hangman Game, by using Java language. The basic idea of this game is that the player is going to guess a word, and for every wrong guess the game is building a part of a man getting hanged. The game also contains additional functions like a high score list and a login system. This is a fun game that can also help people learn English.	

3 | Vision

This project aims to create a Hangman game written in the programming language Java. This game is a fun tool that can provide an exciting learning environment for English learners like children and other non-English speakers.

The basic idea of this game is that the player is going to guess a word by suggesting letter after letter. The player is presented with the number of letters in the word but for every wrong guess, the game is building a part of a man getting hanged. The number of wrongs that the player can have is eight. When the player guesses the whole words, the game shows the winning menu. When the player guesses 8 times wrong, the game shows the fail menu. And after that, the game promotes players to play again.

Additionally, to gain a smoother and provide a better experience for players, more functions will be added to the game like a high score list, user registration, persistence, multiplayer, time limit, point systems, the ability to add and remove words.

Learning English by playing the game is an interesting and efficient way since the learner can be more concentrated than traditional learning methods. The function of adding words can provide a customized test and makes it much fun, and the function of high score list can let the users see their improvements.

Reflection on writing vision:
To write the vision part, first I have to learn what the vision should cover. I read the textbook and the general review. I address the aim of this program and the reason we develop this game. The basic requirements can be recognized by the rules of the Hangman game, and I learned it on Wikipedia and played the game several rounds. As the reason for this report is used both the manager and development team, I should write it in nontechnical words. And this part should also cover the benefits and risk for the manager and it because the defined length of this vision part, I just mention the following parts which will address these things.

4 | Project Plan

4.1 Introduction

This project aims to develop software that can implement the Hangman Game, by using Java language. The basic idea of this game is that the player is going to guess a word, and for every wrong guess the game is building a part of a man getting hanged. The number of wrongs that the player can have is eight. When the player guesses the whole words, the game shows the winning menu. When the player guesses 8 times wrong, the game shows the fail menu. And after that, the game promotes players to play again.

The game also contains additional functions like a high score list multiplayer, and a login system. This is a fun game that can also help people learn English.

4.2 Justification

This game is a fun tool that can provide an exciting learning environment for English learners like children and other non-English speakers. Learning English by playing the game is an interesting and efficient way since the learner can be more concentrated than traditional learning methods.

Teachers can also use the game to test the students' English level. The function of adding words can provide a customized test and makes it much fun, and the function of high score list can let the users see their improvements.

4.3 Stakeholders

The end-user:
The teachers use the game as a teaching tool and want it that is easy to use.
The students use the game as a learning tool and want it that is interesting to use.

The project manager wants the game finished on time and correctly.

The developer wants the requirement clear and easy to implement.

4.4 Resources

The total available time is approximately 9 weeks. The literature is Software Engineering by Ian Sommerville. This project also refers to some online courses on Moodle.

4.5 Hard- and Software Requirements

This development language for this project is limited to Java language. Software: IntelliJ IDEA, the JDK 1.8, JRE 1.8.0, node version 10.9.0, npm version 6.2.0.

4.6 Overall Project Schedule

2020-2-3 first iteration and deliver the first version which implements the basic game rules.

2020-2-17 second iteration and deliver the second version which introduces the UML model.

2020-2-24 third iteration and deliver the third version which implements additional functions.

2020-3-2 Fourth iteration and hand the final project.

4.7 Scope, Constraints and Assumptions

Scope: The Hangman game can be run as a console application. The core function is picking words and displaying the lines that represent each letter. The player shall be able to guess letter after letter. If guessed letter exists in the word, the letter shall be showed. If the letter doesn't exist in the word, the system draws one stroke of the hangman picture. The number of wrongs that the player can have is eight. When the player guesses the whole words, the game shows the winning menu. When the player guesses 8 times wrong, the game shows the fail menu. And after that, the game promotes players to play again.

An extra function is including user registration and creating an account before the game start. And the game also includes a multiplayer mode that allows two-player play at the same time. The game also allows the player to add words that can be guessed.

Outside the scope of this game is the limited words database. The database only includes a limited number of words.

The constraints for the project are the time that is limited to 9 weeks. The developer team only contain one person. The developer has limited knowledge about developing and database. This game shall be run as a console application in the terminal environment without any installation and setup. And For that reason the interface is limited.

The assumptions are that the user has a delightful experience. The user can play the game on their personal computer in a terminal environment without any installation and setup.

Reflection on writing project plan:
To write the project plan, I leaned the structure from the textbook and write is followed by the template. At this stage, I write as much as possible for the project plan. My plan mainly focusses on the requirements from the client's descriptions and Wikimedia and my plan basic be designed by following the requirements. And since I am not sure the next courses, the Hard- and Software Requirements may not clear and I may fix this part latter. As an improvement, the Scope, Constraints, and Assumptions may be added more details.

5 | Iterations.

5.1 Iteration 1

The first iteration aims to achieve the basic function.

Start Date 2020-1-22

End Date 2020-2-3

The first step is to recognize the requirements from clients' stories and make these requirements a list. Next is to exactly follow the requirements, get this document up-to-date, and construct the skeleton structure. This structure should include several classes, abstract classes, and interfaces, with fine documentation. The detailed code is not necessary for this stage. For this iteration, filling general information for the project plan, including time log, risk analysis, defined vision, reflection.

The requirements list for this iteration

1. Give a greeting menu and count time.
2. Make a words database
3. Generate random words from a predefined list
4. show the number of letters displayed with equally many underscore signs.
5. Allow players to input the letter.
6. Check the input and display the letter in the right position when the guessing is correct and draw the picture and show it when it is wrong.

5.2 Iteration 2

Start Date 2020-2-3

End Date 2020-2-24

The goal of this iteration is to update UML diagrams and further implement programs.

UML diagrams include:

1. Use Case description
2. Use Case diagram
3. Class Diagram

4. State Machine Diagram

5. Implement Handful code

To build up a more useful Hangman game, this iteration creates three more use cases: login, view high score list, create an account. But due to time limitations, these use cases will be written more detailed, and the corresponding use case diagrams and state machine diagrams will be created in iteration four.

The Hangman game is a console application, and the main interactions are via a scanner object. After this iteration, the basic functions should be finished. The player guess letter by typing into console implemented by scanner object. The system checks the letter and gives the corresponding response: for correct guess system add one point, and for the wrong guess draw one stroke of hangman.

In this iteration, the essential code is finished. The main function is controlled by the GameControl class. The words that need to be guessed are stored in WordDatabase class. The pictures of hangman are stored in the DrawHangman class.

When the game starts, the system shows the main menu and welcome, and generates a random word for WordDatabase. The system displays the lines that represent letters, correct guesses, and wrong guesses, which are controlled by GameControl class. And for restart and quit functions are also implemented in GameControl class.

5.3 Iteration 3

Start Date 2020-2-23

End Date 2020-3-9

The goal of this iteration is to test the existing code. No new features are added in this iteration.

The testing processes are manual tests for use cases and unit tests for code fragment. And due to time limitations, the manual tests are only focused on two use cases: start the game and play game. The unit tests are written by Junit 5, and only test two methods from GameControl class and HighScore class. The test processes and test results are updated to the test report.

5.4 Iteration 4

Start Date 2020-3-9

End Date 2020-3-24

This iteration aims to finish the project. For the Hangman game, this iteration updates current UML diagrams that need adjust for reflecting the actual implementation. The incorrect in state machine diagrams and the class diagram in iteration two need to be corrected.

And for this iteration, additional functions are considered to be added to the system. The additional functions of the high score list is implemented, and the corresponding UML diagrams are updated. Other functions that are implemented in this iteration are login and create an account. Players can choose to log into the system and create an account before they play game (the game can be played without login or create an account). Some of these use cases are defined in iteration two, but there are still things need to be adjusted. The state machine diagram and the class diagram need to be changed in order to reflect the implementation.

The function of viewing a high score list is implemented by creating a new class that stores the top five history scores, and in the GameControl class player can choose to display their score list after they finish one round.

The functions of create account and login are implemented by creating a PlayerDatabase class. This class stores all the players' names and passwords, and when a player chooses to create an account, the system will add their information to this class. When a player chooses to log in, the system will check the name and password that user input, and when these are the correct system will show that you are log in.

The general information also needs to be updated. The justification should be clearer, the different influence of different stockholders should be written. The risk analysis should be updated. The vision, hard and software requirements, and scope and assumption which written in iteration one need to be adjusted. For the testing report, the tests are updated for the new implementation.

Reflection:

In this iteration, I focus on correcting errors made in the previous iteration. The project plan that written in iteration one has a lot of errors and I rewrite this according to the general information. Other main mistakes happen in the UML diagram, most of is in the state machine diagram and class diagram. And those new functions and new implementation also have corresponding documents. And due to time limitations, the plan of changing code to more easily to be test has to be canceled. So I still need to improve the time management.

6 | Risk Analysis

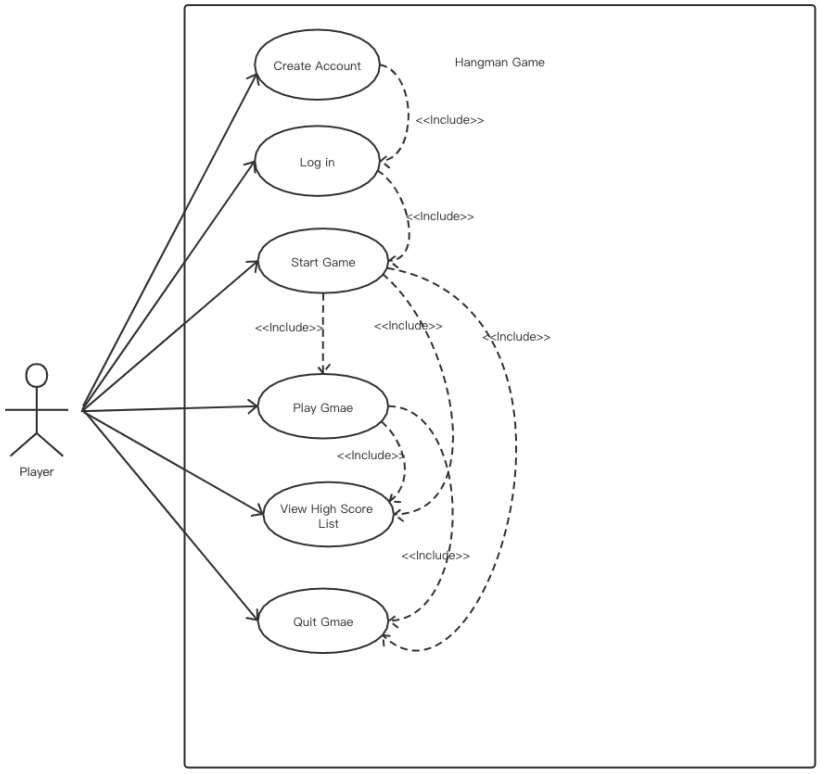
This part is to anticipate the possible risks and set strategies for coping with these risks.

Risk	Probability	Impact	Strategy
Time underestimate	High	Serious	keep attention to the progress of development and change the schedule if necessary
Size underestimate	High	Serious	Carefully plan for each task and keep attention to the current progress.
Hardware crash	Low	Catastrophic	Always save the code on several places, in this case, push into GitLab frequently.
Staff leave since illness	Moderate	Serious	Always starts work early and try to finish before the deadline.
Developer lacks knowledge	High	Tolerate	Read Software Engineering by Ian Sommerville and view the lecture on Moodle
Requirement change	Low	Serious	prepare to change the requirements list and add it to the code.

Reflection on writing risk analysis:
This is a hard part for me since I never did this part before. I read the textbook and just followed the risk list written in the book. The risks analysis should anticipate the future risk during development, and I should not only consider the coding processes and also think about all the processes throughout the whole time. The outside environment should also be covered. At the beginning stage, I just list 4 possible ricks since the project is relatively small and easy. In the future stage, I may continually add new risks to this list.

7 | UML

1. Use Case Diagram



2. Fully Dressed Use Case

UC 1 Start Game

Primary Actors: Player
Precondition: none
Postcondition: The game menu is shown.
Main scenario:

1. It starts when the user wants to begin a session of the hangman game.
2. The system presents the main menu with a title, the option to play, login, view high score list, and quit the game.
3. The Gamer makes the choice to start the game.
4. The system starts the game (see Use Case 2).

Repeat from step 2
Alternative scenarios:

3.1. The Gamer makes the choice to quit the game.
  1. The system quits the game (see Use Case 3)
3.2 Invalid menu choice
  1. The system presents an error message.
  2. Go to step 2
3.3. The gamer choices to log in.
  1. See UC 4.
3.4. The gamer choices to see the high score list.
  1. See UC5.

UC 2 Play Game
Primary Actors: Player

Precondition: The game is started.
Postcondition: The system shows the results and the option to see a high score.
Trigger: Player choices to play.
Main scenario:
1. It starts when the user starts the hangman game.
2. The system shows the number of letters, lines represent letters, and the number of the chances to guess (total number is 9).
3. The system promotes players to guess.
4. The player guesses a letter.
5. The guessed letter is correct. The system shows the guessed letter in the corresponding line and counts one point.
6. Repeat form 3, until the player guesses the whole word.
7. System shows “You win” and the score.
8. System show a high score list (see UC5)
9. The system shows “do you want to restart the game?”
10. The player chooses to play again go to step 2.

Alternative scenarios:
4.1 Player’s guess is not a letter
1. The system show "Please guess a letter".
2. Go to step 3.

4.2 Player’s guess is already guessed before.
1. System show "You have guessed this letter before".
2. Go to step 3.

5.1 The guessed letter is incorrect.
1. The system draws one stroke of hangman, and the number of chances subtract one and display it.

6.1 The Player doesn't have a chance anymore.
1. If the number of chances 0, the system shows "You lose" and go to step 9.

9.1. The player doesn't want to play again
1. The system quit the game, see UC 3.

UC Quit Game
Primary Actors: Player
Precondition: The game is running.
Postcondition: The game is terminated
Main scenario:
1. It starts when the user wants to quit the game.
2. The system prompts for confirmation.
3. The user confirms.
4. The system terminates.

Alternative scenarios:
3.1. The user does not confirm
1. The system returns to its previous state

UC4 Log in
Primary Actors: Player
Precondition: The main menu is shown.
Postcondition: The player logs in to the system and the main menu is shown (UC1).
Main scenario:
1. It starts when a player wants to log in when the main menu is shown (UC1).
2. The system presents the login menu with two options: "Log in" and "Create an account".

3. Player choice to create an account.
4. The system promotes player to enter the user name and password.
5. The player enters the user name and password.
6. System checks user name and password are legal or not (the user name must not be used).
7. The system creates a new account and show the message of "the user is successfully registered".
8. Player logs into the system.
9. The system shows the main menu (UC1).

Alternative scenarios:

3. 1. Player choice to log in.
   1. The system promotes player to enter the user name and password.
   2. The player enters the user name and the password.

3.a. If the user name or password is wrong, the system shows "user name or password is wrong" and goes to step 3.1.1 .

3.b. If the user name and password are wrong, go to step 8.

6.1 System checks the user name or password is illegal.
   1. The system shows "user name or password is illegal".
   2. Go to step 4.

UC 5 View Hign Score List
Primary Actors: Player
Precondition: T main menu is shown (UC1) or the player finishes a round of the game (UC2).
Postcondition: The high score list is shown.
Main scenario:

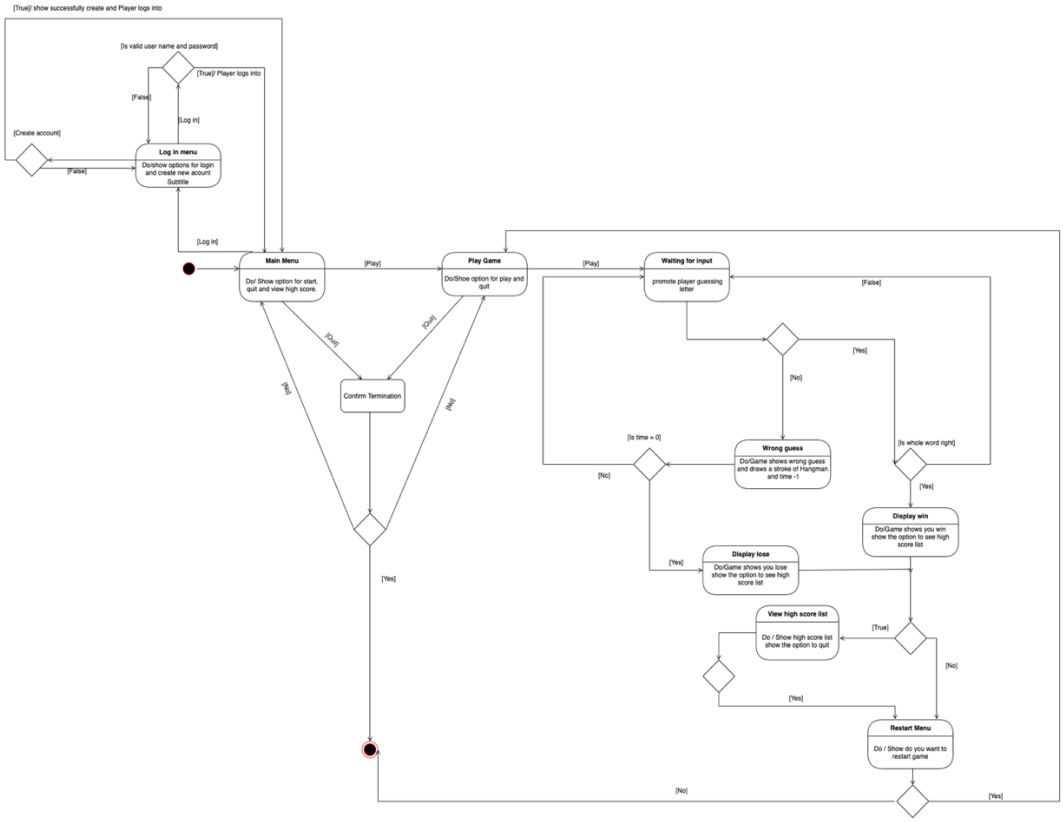
1. Starts when user choices to see the high score list when he or she at the main menu (UC1) or finishes a round of the game (UC2).
2. The system shows “Do you want to see the high score list?”.
3. The player choices yes.
4. The system displays the high score list.
5. The player views the high score list.

Alternative scenarios:

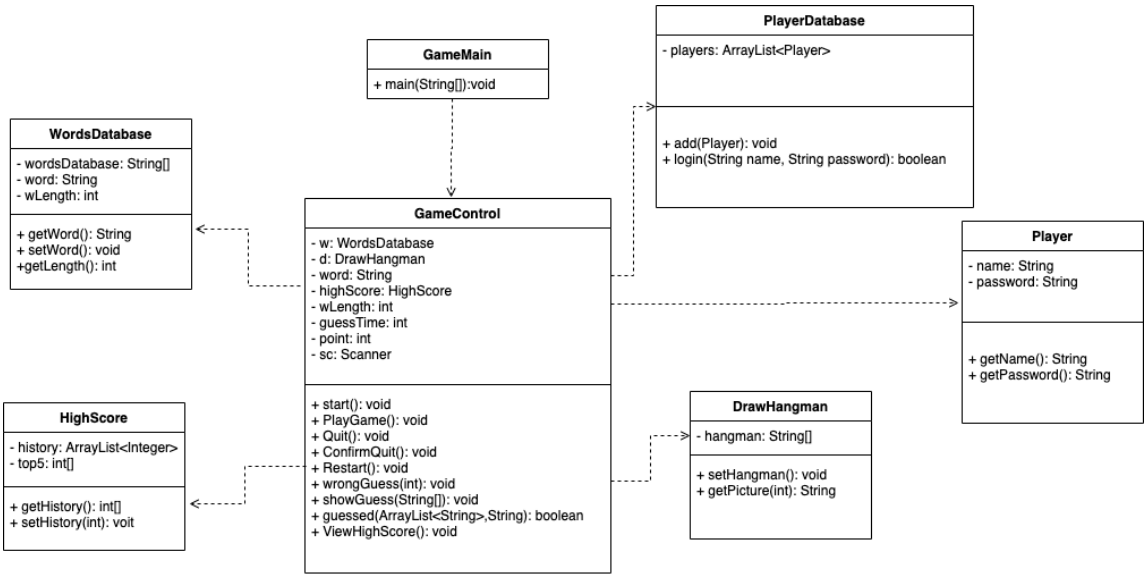
3.2 The player choices no.
   1. The system returns to its previous state.

5.1. The player decides to terminate the list.
   1. The system returns to its previous state.

3. Machine Diagram



4. Class diagram



8 | Test

1. Objective

The objective of this project is to test the code that was implemented for iteration 2 to make sure that the final project is good enough to be used.

2. What to Test and How

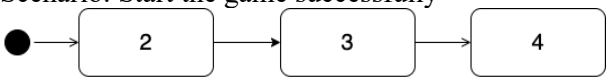
For this iteration, there are only two use cases to be test which is UC1 start game and UC2 play game by writing and running dynamic manual test-case. The main reason we only chose these two use cases to be tested is the time limitation, and these use cases are essential to the project. For other use cases, we choose to write the tests in a later iteration.
Two methods from GameControl class and one unfinished method in HighScore class shall be tested by automated tests.

public boolean guessed(ArrayList<String> guess, String st)
public boolean isLetter(String st)
public int[] getTop5(ArrayList<Integer> history)

There shall be at least two unit tests per method. Due to lack of time, we chose these two methods since they have simple input that can be easily controlled by the tester, and simple output that can be easily asserted by JUnit test.
we only use dynamic tests (i.e. tests which must run through execute a program), not static test. The tests include manual tests and automated tests. The dynamic manual tests are bases on the use cases we defined in iteration 2. And for the automated test, we are using Jnit5.

3. Manual Test-Case

TC1.1 Start game successfully
Use case: UC1 Start Game
Scenario: Start the game successfully

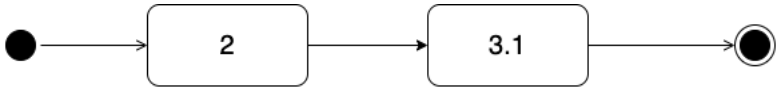


Description: The main scenario of UC1 is tested where a user starts a game successfully (start of UC2).
Precondition: delete all words in WordsDatabase class except wave.
Test steps:
1. Start the game
2. The system shows the main menu "Do you want to play? (Enter play or quit): "
3. User types play and press enter.

Expected:
1. The system should show the game menu. (“Welcome”, “_ _ _ _”, “Guess a letter:”)

Test result:
✓ Pass
☐ Fail
Comment: TC1.1 passed and enter to UC 2 which will test below in TC2.

UC1.2 Quit game
Use case: UC1 Start Game
Scenario: Quit game



Description: The alternative scenario of UC1 is tested where a user quite game (start of UC3).
Precondition: None.
Test steps:
1. Start the game
2. The system shows the main menu "Do you want to play? (Enter play or quit): "
3. User types quit and press enter.

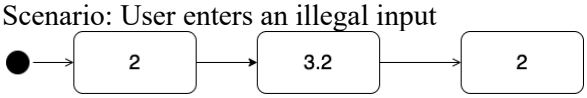
Expected:
1. The system shows "Are you sure to quit? (Enter quit or not)"

Test result:
✓ Pass
☐ Fail

Comment: TC1.2 passed and enter to UC3 quit game which is not tested in this iteration.

TC1.3 Illegal menu choice and promote input again

Use case: UC1 Start Game
Scenario: User enters an illegal input



Description: The alternative scenario of UC1 is tested where a user enters an illegal input.
Precondition: None.
Test steps:
1. Start the game
2. The system shows the main menu "Do you want to play? (Enter play or quit): "
3. Type p and press enter, System shows “invalid input. Do you want to play? (Enter play or quit)”.
4. Type 1 and press enter.

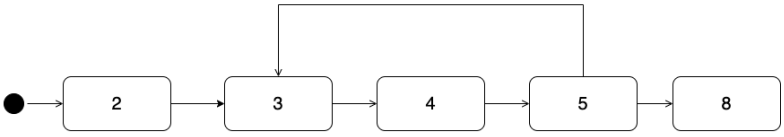
Expected:
1. The system shows "invalid input. Do you want to play? (Enter play or quit)".

```
Invalid input
Do you want to play? (Enter play or quit):
```

Test result:
✓ Pass
☐ Fail
Comment: none.

TC2.1 Guess word successfully without incorrect guess

Use case: UC2 Play Game
Scenario: User guess the entire word without incorrect guess.



Description: The main scenario of UC2 is tested where a user guesses the word successfully.
Precondition: the game is started, and the tester deletes all words in WordsDatabase class except wave.
Test steps:

1. Start the game
2. System shows the main menu guess menu with “_ _ _ _” and “Guess a letter”.
3. Type "w" and press enter, the system shows "Correct guess", "w _ _ _ “,” Guess a letter”.
4. Type “a” and press enter, system shows “Correct guess”, “w a _ _ “,” Guess a letter”.
5. Type "v" and press enter, the system shows "Correct guess", "w a v _ “,” Guess a letter”.
6. Type “e” and press enter.

Expected:

1. system shows “Correct guess”, “w a v e “,” Guess a letter”, “You win and your point is 4”.

Test result:

✔ Pass

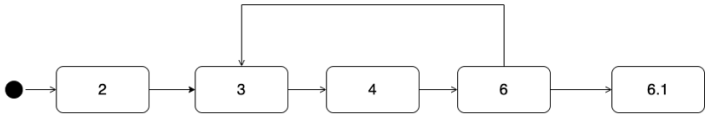
☐ Fail

Comment: TC2.1 passed and enter to UC5 view high score list which is not tested in this iteration.

TC2.1 Guess word incorrectly without correct guess

Use case: UC2 Play Game

Scenario: User guess the word without correct guess.



Description: The alternative scenario of UC2 is tested where a user guesses word without a correct guess.

Precondition: the game is started, and the tester deletes all words in WordsDatabase class except wave.

Test steps:

1. Start the game
2. System shows the main menu guess menu with “_ _ _ _” and “Guess a letter”.
3. Type q and press enter, system shows “Wrong guess and you have 7 times only”, one stroke of hangman, “_ _ _ _”, “Guess a letter.

```
Wrong guess and you have 7 times only
   _______
   |
   |
   |
   |
   |
========


_ _ _ _
Guess a letter:
```

4. Type r and press enter, system shows “Wrong guess and you have 6 times only”, one stroke of hangman, “_ _ _ _”, “Guess a letter.

```
Wrong guess and you have 6 times only
   _______
   |    |
   |
   |
   |
   |
========


_ _ _ _
Guess a letter:
```

5. Type t and press enter, system shows “Wrong guess and you have 5 times only”, one stroke of hangman, “_ _ _ _ _”, “Guess a letter.

```
Wrong guess and you have 5 times only
   _______
   |     |
   |     o
   |
   |
   |
========


_ _ _ _
Guess a letter:
```

6. Type y and press enter, system shows “Wrong guess and you have 4 times only”, one stroke of hangman, “_ _ _ _ _”, “Guess a letter.

```
Wrong guess and you have 4 times only
   _______
   |     |
   |     o
   |     |
   |
   |
========


_ _ _ _
Guess a letter:
```

7. Type u and press enter, system shows “Wrong guess and you have 3 times only”, one stroke of hangman, “_ _ _ _ _”, “Guess a letter.

```
Wrong guess and you have 3 times only
   _______
   |     |
   |     o
   |    /|
   |
   |
========


_ _ _ _
Guess a letter:
```

8. Type g and press enter, system shows “Wrong guess and you have 2 times only”, one stroke of hangman, “_ _ _ _ _”, “Guess a letter.

```
Wrong guess and you have 2 times only
   _______
   |     |
   |     o
   |    /|\
   |
   |
========


_ _ _ _
Guess a letter:
```

9. Type h and press enter, system shows “Wrong guess and you have 1 time only”, one stroke of hangman, “_ _ _ _ _”, “Guess a letter.

```
Wrong guess and you have 1 times only
   _______
   |     |
   |     o
   |    /|\
   |    /
   |
========


_ _ _ _
Guess a letter:
```

10. Type r and press enter.

Expected:

1. system shows “The wrong guess and you have 0 times only”, one stroke of hangman, “_ _ _ _ _”, “You lose”.

```
Wrong guess and you have 0 times only
   _______
   |     |
   |     o
   |    /|\
   |    / \
   |
========


_ _ _ _
You lose
```

Test result:

✔ Pass

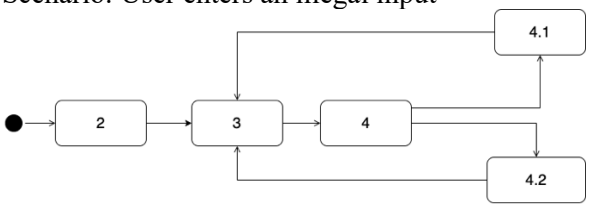
☐ Fail

Comment: TC2.1 passed and enter to UC5 view high score list which is not tested in this iteration

TC2.3 Illegal guess and promote input again

Use case: UC2 Play game

Scenario: User enters an illegal input



Description: The alternative scenario of UC2 is tested where a user enters an illegal input.

Precondition: the game is started, and the tester deletes all words in WordsDatabase class except wave.

Test steps:

1. Start the game
2. System shows the main menu guess menu with "_ _ _ _" and "Guess a letter".
3. Type "w" and press enter, the system shows "Correct guess", "w _ _ _", "Guess a letter".
4. Type "1" and press enter, the system shows "Invalid input", "w _ _ _", "Guess a letter".
5. Type "wave" and press enter, the system shows "Invalid input", "w _ _ _", "Guess a letter".
6. Type "w" and press enter.

Expected:

1. System shows "You have guessed this letter before", "w _ _ _", "Guess a letter".".

Test result:

✔ Pass

☐ Fail

Comment: none.

Test Report

Test	UC1	UC2	UC3	UC4	UC5
TC1.1	1/OK	0	0	0	0
TC1.2	1/OK	0	0	0	0
TC1.3	1/OK	0	0	0	0
TC2.1	0	1/OK	0	0	0
TC2.2	0	1/OK	0	0	0
TC2.3	0	1/OK	0	0	
COVERAGE & SUCCESS	3/OK	3/OK	0	0	0

Comment:

All tests pass, UC3, UC4, and UC5 will be test on the next iteration due to the time limitation.

4. Automated Unit Test

Two methods from GameControl class and one unfinished method in HighScore class shall be tested by automated tests.

```
public boolean guessed(ArrayList<String> guess, String st)
```

test code

```
import ...

class GameControlTest {
    private ArrayList<String> strings = new ArrayList<>();
    GameControl gameControl = new GameControl();
    private ArrayList<Integer> history = new ArrayList<>();
    HighScore highScore = new HighScore();


    @Test
    //when guess the first letter, the arrayList is empty
    void guessed1() {
        boolean actual = gameControl.guessed(strings, st: "GuessOne");
        assertFalse(actual);
    }

    @Test
    //when one guess already be made, the arrayList is not empty, the arrayList contain the guess string
    void guessed2() {
        strings.add("Guessed");
        boolean actual = gameControl.guessed(strings, st: "Guessed");
        assertTrue(actual);
    }

    @Test
    //when one guess already be made, the arrayList is not empty the arrayList dosen't contain the guess string
    void guessed3() {
        boolean actual = gameControl.guessed(strings, st: "Guess");
        assertFalse(actual);
    }
```

```
public boolean isLetter(String st)
test code
```

```
@Test
void isLetter1(){
    String s = "word";
    boolean actual = gameControl.isLetter(s);
    assertFalse(actual);
}
@Test
void isLetter2(){
    String s = "w";
    boolean actual = gameControl.isLetter(s);
    assertTrue(actual);
}
@Test
void isLetter3(){
    String s = "1";
    boolean actual = gameControl.isLetter(s);
    assertFalse(actual);
}
```

```
public int[] getTop5(ArrayList<Integer> history)
test code
```

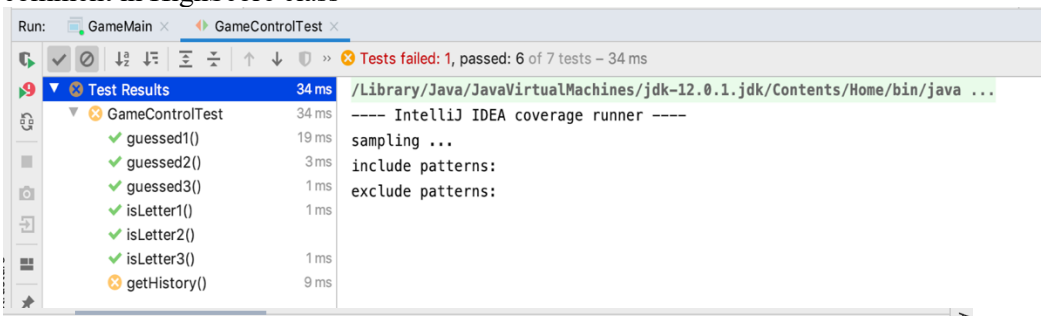
```
    @Test
    //test the method in HighScore class
    void getHistory(){
        history.add(3);
        history.add(1);
        history.add(5);

        int[] top5 = new int[5];
        top5 = highScore.getTop5(history);

        //the top5 should be sorted
        //implement the comment in HighScore class will let the test success
        for (int i = 0;i<4;i++) {
            assertTrue( condition: top5[i] >= top5[i+1]);
        }
    }
}
```

Test result

6 tests pass and one unfinished method fail which will success when removing the comment in HighScore class



83% classes, 45% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
apple			
com			
images			
java			
javax			
jdk			
META-INF	100% (0/0)	100% (0/0)	100% (0/0)
netscape			
org			
sun			
toolbarButtonGraphics			
DrawHangman	100% (1/1)	66% (2/3)	92% (13/14)
GameControl	100% (1/1)	27% (3/11)	16% (19/113)
GameControlTest	100% (1/1)	100% (7/7)	97% (34/35)
GameMain	0% (0/1)	0% (0/1)	0% (0/3)
HighScore	100% (1/1)	50% (2/4)	75% (9/12)
WordsDatabase	100% (1/1)	100% (4/4)	100% (10/10)

My reflection

Due to lack of time, I only create two test cases based on two use cases that defined on iteration 2, and many use cases remain untested now. And for writing manual tests and trying to find every path for each use case, I made a little improvement in the definition of use cases.

The same problem happens to automated unit tests too. I only cover two finished methods and one unfinished method. One reason is lack of time and another main reason is that all of my method depends on other methods or class, and lots of my methods are

void methods that do not have clear input and output, which makes it hard to be tested. To improve the testability, I need to make a lot of changes in my source code.

9 | Time log

Iteration one

Task	Date	Estimate Time	Actual Time	Difference
Reading book	1-27	3 hours	4 hours	1 hour
Project plan	1-31	2 hours	3 hours	1 hour
Time log	1-31	1 hour	45 mins	15 mins
Vision	1-31	30 mins	1 hour	30 mins
Iteration plan	2-1	1 hour	45 mins	15 mins
Risk analysis	2-2	30 mins	30 mins	0
Coding	2-3	1 hour	1 hours	0

Iteration two

Task	Start date	Estimate time	Actual time	Difference
Define the use case and the use case diagram	2.7	2 hours	1.5 hours	0.5 hour
Fully dressed use case	2.8	3 hours	3 hours	
Draw the machine diagram	2.9	1 hour	2 hours	1 hour
Implement the basic version	2.21	4 hours	5 hours	1 hour
Draw the class diagram	2.21	2 hours	1 hour	1 hour
Time log	2.7	30 min	30 min	
Extend version for high grade	2.22	1 hour	1.5 hours	0.5 hour

Iteration there

Task	Start date	Estimate time	Actual time	Difference
View video	3-3	3 h	2.5 h	0.5 h
Time log	3-4	10 min	10 min	0
Test plan	3-4	30 min	45 min	15 min
Manual test	3-4	3 h	2.5 h	0.5 h
Unit test	3-5	2 h	2 h	0
My reflection	3-8	15 min	15 min	0

Iteration four

Task	Start date	Estimate time	Actual time	Difference
Time log	3-15	15 min	15 min	0
Update document	3-15	3 h	3 h	0
Correct previous document	3-17	30 min	1	30 min
Implement new functions	3-18	3 h	4 h	1 h
Update state machine diagram	3-18	2 h	1.5 h	0.5 h
Update class diagram	3-18	1 h	1 h	0
My reflection	3-18	15 min	15 min	0