Test plan

Author: Jinzhe Zhao

Code: https://github.com/jz222cv/1dv600/tree/master/code

Table of content

Introduction

Purpose
In the process of doing a computer software project, the key activity is to design and implement the project plan. In this project plan, creating a game called Hangman, and in the iterations, the game will be improved continuously. The aim of this plan is to design for testing this application. This test plan includes information about the test effort in this iteration of the system. The purpose of the test plan is to make sure that the project can run without system errors.

Project Overview
According to the given requirement, the application is a console game, all the frontends are show in the console. And the structure is MVC, model, controller and viewer. The model contains the game and player models. The controller controls the progress of the game and the viewer show the tips of the game.

The **purpose** of this project is to meet each requirement of the tasks and create a good software application that can achieve the function required such as starting, saving, loading and quitting game. Furthermore, learning the technique that how to design and plan a software project is also a goal of this project.

Audience
The project team performs the tasks that are listed in this document. Project manager plans for the testing activities and tracks the performance of the tests. The project team and project manager are aligned together in order to make sure that the test result is the same as the result are provided by project team.

Test objectives
The objectives in this test are testing basic models: game model and player model. The unit test method to test is using junit. And the manual test is to test by the players playing.

The game model and player model are the basic structures. So the first objective that should be tested are the two classes, because they are the basic structures, and should not occur errors. To test the two classes, I will create two test objects, and set the properties, and compare every property with the result of the methods.

The game model is the model to operate the game that contains the player model and the model to operate the database. It also create the word by random and check if the player guess the correct words.

For the manual tests, I will play the game and input all the cases I may think about. For example, if I input a non-number as a number, it will throw exceptions rather than stop running.

Test table

Test	**Estimated time**	**Actual time**
Game model	0.5 hour	1 hour
Player model	0.5 hour	0.3 hour
Start game test	1 hour	1.5 hours
Guess word test	1 hour	1 hour
Load game test	0.5 hour	2 hours

Test case

1. StartGame

ID: StartGame001
Name: StartGame
Requirement: Play controller
Description: It tests the Play controller. This is a controller that can deal with all the inputs and control the whole progress of the game, it is needed, so it must be tested successfully, or the whole game will not run well. It uses the views to read the input commands and if the input equals the certain command, the Play controller will start the game.
Precondition: The game application must start first and the game has not started.
Steps: Input 1 and press enter. And then input a string as username.
Expected: The controller will start the game and create a word randomly and then instruct the player to start guessing.
Checkbox: Succeed
Comment: When I started to do the test, I met some problems. That is the exception when I input a non-number as the starting command. The starting command is 1, so I must use Scan.nextInt to get it, but if I input a non-number, it will throw exception. Then I check the inputs if it is a number, and then the test succeed.

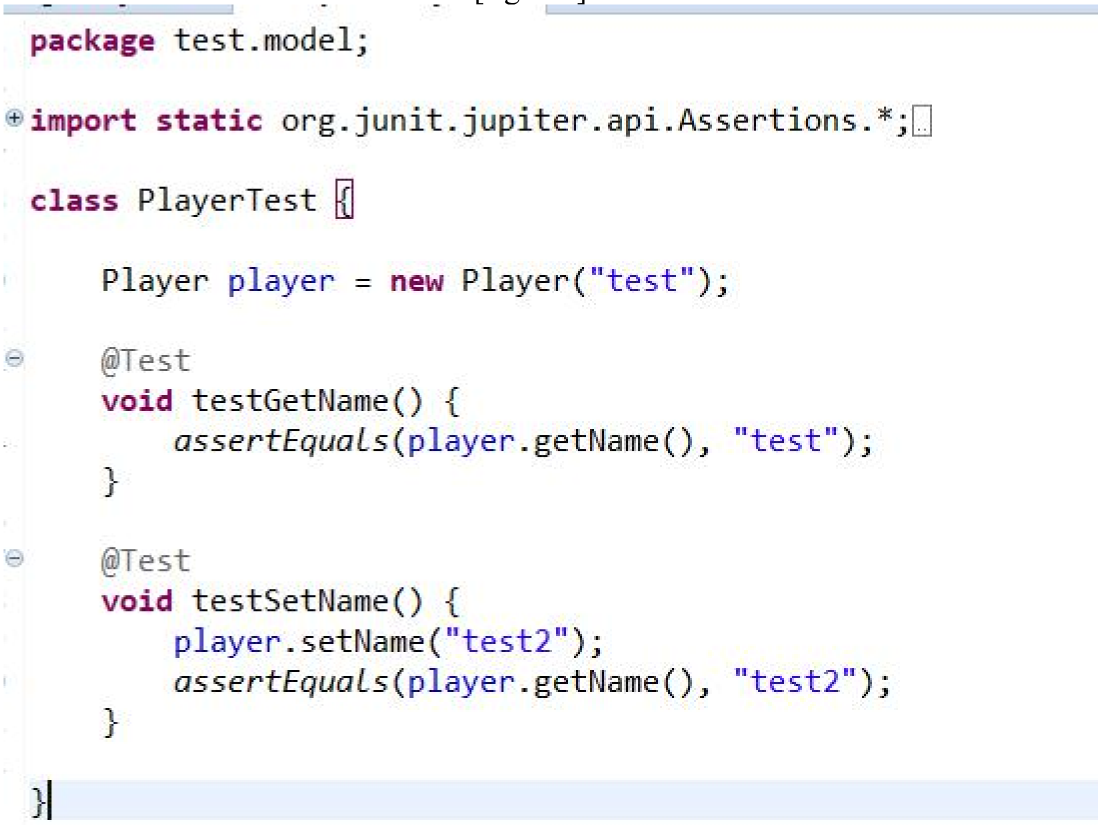
2. LoadGame

ID: LoadGame002
Name: LoadGame
Requirement: Play controller and Game model.
Description: It tests the Play controller. In this test, the Play controller will deal with the input commands and use the Game model to load the game. This is a needed function, so the test must be successful. When the player guesses the word and before go to the next turn, the game will be saved, and if the player plays the game later, they can load the game to continue the record.
Precondition: The game application must start first and the database has the game record.
Steps: Input 2 and press enter.
Expected: The controller will use the Game model to load the game and continue the game.
Checkbox: Succeed
Comment: When I did the test first time, I met the IO exception, and then I check the problem and I found that the database file path was wrong.

Unit test

In unit test, I have tested Game model and Player model. And the others will be tested by manual tests, because in the console application, I cannot test the output by code.

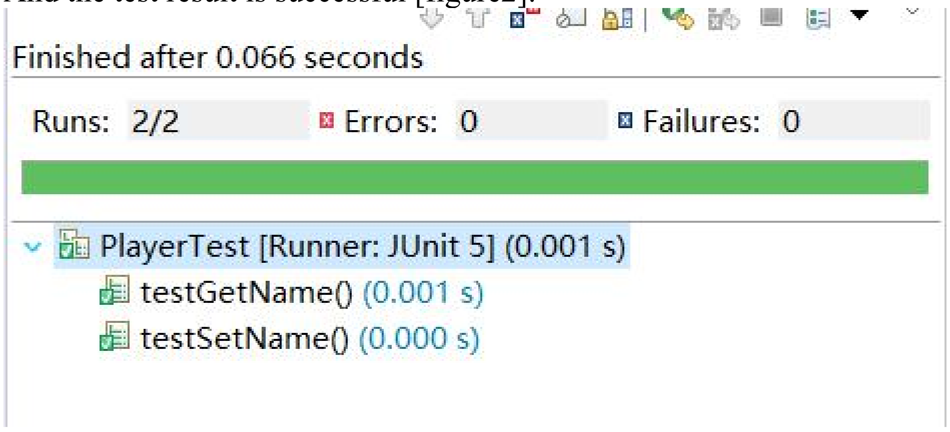
In the Player model test, I tested every method in the class: setName() and getName(). The code is shown in the screenshot [figure1].



```java
package test.model;

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

class PlayerTest {

    Player player = new Player("test");

    @Test
    void testGetName() {
        assertEquals(player.getName(), "test");
    }

    @Test
    void testSetName() {
        player.setName("test2");
        assertEquals(player.getName(), "test2");
    }

}
```

[figure1]

And the test result is successful [figure2].

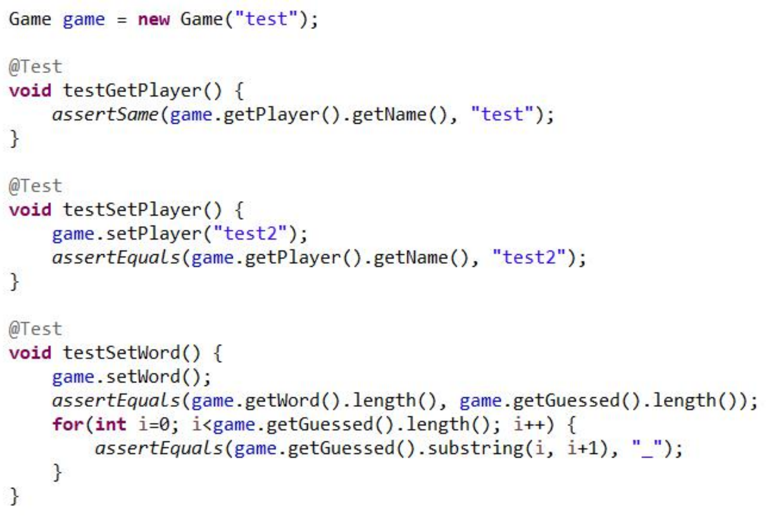


```
Finished after 0.066 seconds

Runs: 2/2    Errors: 0    Failures: 0

PlayerTest [Runner: JUnit 5] (0.001 s)
    testGetName() (0.001 s)
    testSetName() (0.000 s)
```

[figure2]

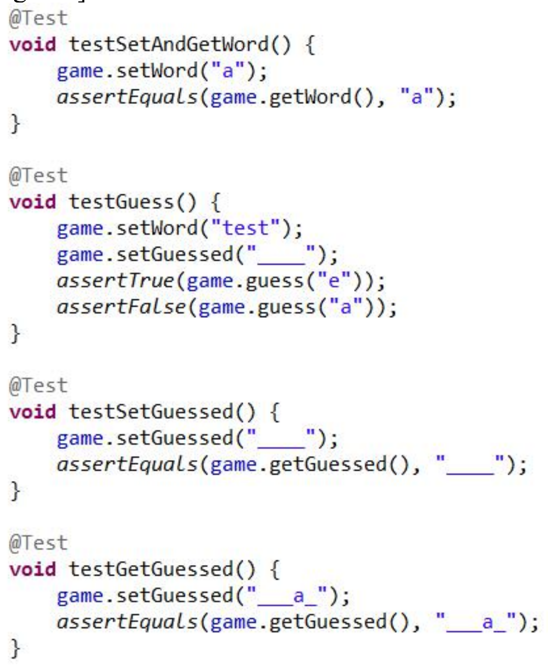
In the Game model test, I tested every method in the class: `getPlayer()`, `setPlayer()`,`setWord()`,`setAndGetWord()`,`guess()`, `setGuessed()`, `getGuessed()`, `checkWin()`, `saveAndDeleteAndLoad()`.

The code is shown in the screenshot [figure3-5].



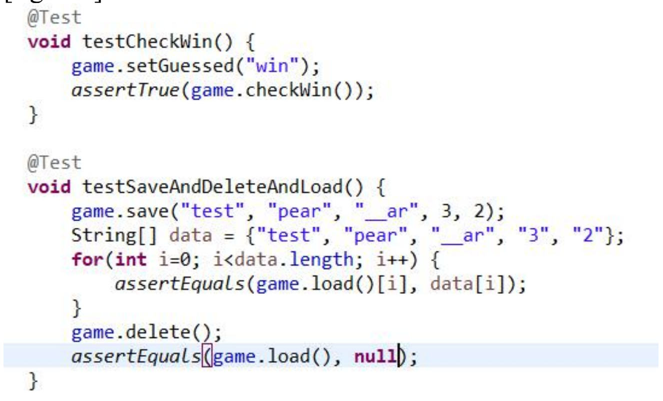
```java
Game game = new Game("test");

@Test
void testGetPlayer() {
    assertSame(game.getPlayer().getName(), "test");
}

@Test
void testSetPlayer() {
    game.setPlayer("test2");
    assertEquals(game.getPlayer().getName(), "test2");
}

@Test
void testSetWord() {
    game.setWord();
    assertEquals(game.getWord().length(), game.getGuessed().length());
    for(int i=0; i<game.getGuessed().length(); i++) {
        assertEquals(game.getGuessed().substring(i, i+1), "_");
    }
}
```

[figure3]



```java
@Test
void testSetAndGetWord() {
    game.setWord("a");
    assertEquals(game.getWord(), "a");
}

@Test
void testGuess() {
    game.setWord("test");
    game.setGuessed("____");
    assertTrue(game.guess("e"));
    assertFalse(game.guess("a"));
}

@Test
void testSetGuessed() {
    game.setGuessed("____");
    assertEquals(game.getGuessed(), "____");
}

@Test
void testGetGuessed() {
    game.setGuessed("___a_");
    assertEquals(game.getGuessed(), "___a_");
}
```

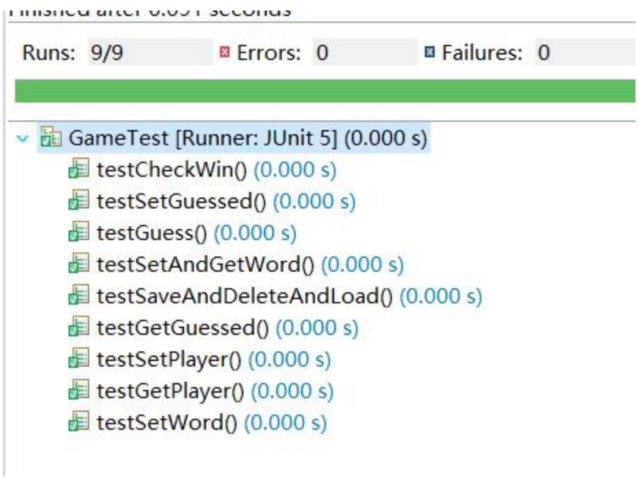
[figure4]



```java
@Test
void testCheckWin() {
    game.setGuessed("win");
    assertTrue(game.checkWin());
}

@Test
void testSaveAndDeleteAndLoad() {
    game.save("test", "pear", "__ar", 3, 2);
    String[] data = {"test", "pear", "__ar", "3", "2"};
    for(int i=0; i<data.length; i++) {
        assertEquals(game.load()[i], data[i]);
    }
    game.delete();
    assertEquals(game.load(), null);
}
```

[figure5]

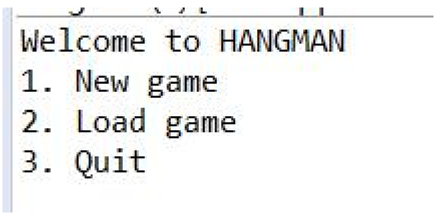
And the test result is successful [figure6].



```
Runs: 9/9    Errors: 0    Failures: 0

GameTest [Runner: JUnit 5] (0.000 s)
    testCheckWin() (0.000 s)
    testSetGuessed() (0.000 s)
    testGuess() (0.000 s)
    testSetAndGetWord() (0.000 s)
    testSaveAndDeleteAndLoad() (0.000 s)
    testGetGuessed() (0.000 s)
    testSetPlayer() (0.000 s)
    testGetPlayer() (0.000 s)
    testSetWord() (0.000 s)
```

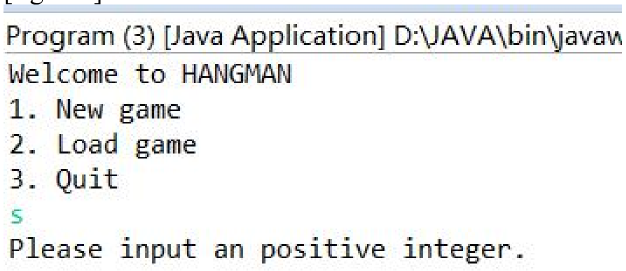
[figure6]

In the starting game test, figure7 is the starting screenshot. All the commands should be number, and if I input a non-number, it will throw exception in the figure8. Then I input 1 and username, the game starts in figure9.



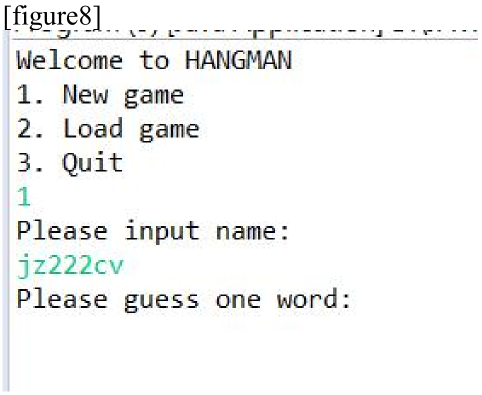
```
Welcome to HANGMAN
1. New game
2. Load game
3. Quit
```

[figure7]



```
Program (3) [Java Application] D:\JAVA\bin\javaw
Welcome to HANGMAN
1. New game
2. Load game
3. Quit
s
Please input an positive integer.
```

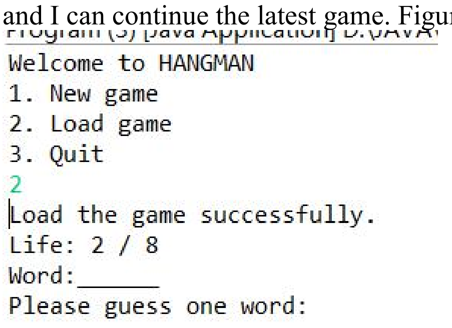
[figure8]



```
Welcome to HANGMAN
1. New game
2. Load game
3. Quit
1
Please input name:
jz222cv
Please guess one word:
```

[figure9]

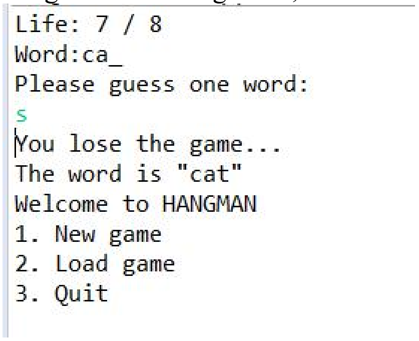
In the load game test, at the starting page, I input 2 as command and the game will load the game and I can continue the latest game. Figure10



```
Welcome to HANGMAN
1. New game
2. Load game
3. Quit
2
Load the game successfully.
Life: 2 / 8
Word:______
Please guess one word:
```

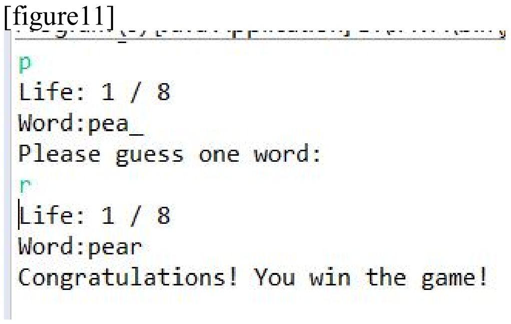
[figure10]

In the guess test, I will input one word, and the game will check if I guess correctly. If I lose all the lives, I will fail the game [figure11]. If I guess out all the words, I will win the game [figure12]. If I guess the wrong word, and I have some lives, I will guess again [figure13].



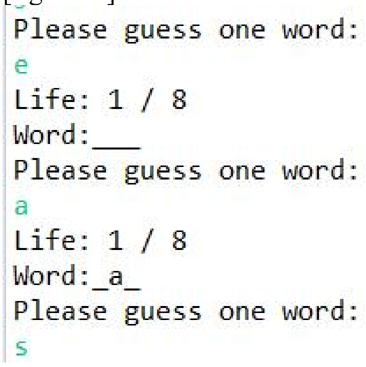
```
Life: 7 / 8
Word:ca_
Please guess one word:
s
You lose the game...
The word is "cat"
Welcome to HANGMAN
1. New game
2. Load game
3. Quit
```

[figure11]



```
p
Life: 1 / 8
Word:pea_
Please guess one word:
r
Life: 1 / 8
Word:pear
Congratulations! You win the game!
```

[figure12]



```
Please guess one word:
e
Life: 1 / 8
Word:___
Please guess one word:
a
Life: 1 / 8
Word:_a_
Please guess one word:
s
```

[figure13]

Reflection

When I started to write the test code, I met several problems. First I used junit5, but I have not used it before, so I googled the junit5 and search some examples, like the example from teachers. And I also met problems in the manual tests, because when I tested the game, it threw lots of exceptions that I have not met before, and then I fixed these bugs, and it runs better than before. So the manual test and unit test are all really necessary.