Lubna snobar

Ls223rv@student.lnu.se

https://github.com/Lubnasnowbar?tab=repositories

Use cases

UC1 checking the inputs

Precondition: the user starts the game.

Postcondition: the user playing by input letters.

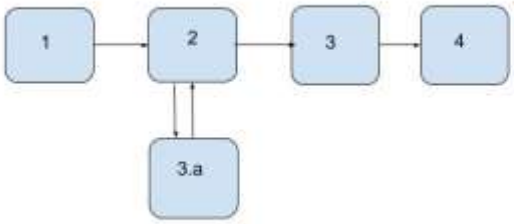
Main scenario

1. the game start when the player chooses “play” choice
2. the system asks to choose a letter.
3. The player presses a random letter.
4. The system shows the result of choosing process.

Alternate scenarios

• 3.1 The player presses a symbol or a number instead of a letter.

• the system shows an error message. And ask the user to make valid input again.



Objective

- The objectives of testing is to test the codes, and to do a dynamic manual testing for use cases and we will expecting the results manually.
- In addition to an automated unit tests that are testing two methods used in the game class
- We will test two methods because the are main part to implement the cod as should.

Time plan

Task	Estimated	Actual
Manual TC	2h	1hour30m
Unit tests	1h	50m

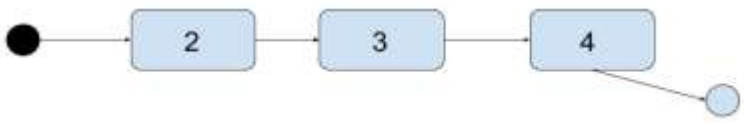
Task	Estimated	Actual
Running manual tests	30m	10m
Code inspection	1h	40m
Test report	1h	30m

Manual test cases:

TC1.1 User made a valid input

UC-1 checking inputs

Scenario: check input successful.



The main scenario of UC-1 is tested if the user pressed a valid letter.

Test steps

1. the system asks to choose a letter.
2. The player presses “g” letter.
3. The system shows the result.

Expected

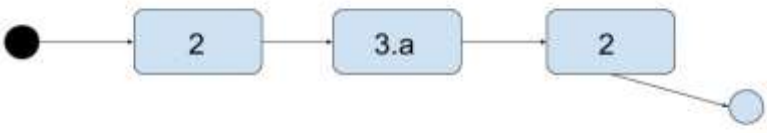
- The system shows if “g” is one of the required words letters.
- The user input another letter to the next round.
- The system continues with asking the user to guess next letter.

TC1.2 user made an invalid input

Use-case: UC1 checking inputs.

Scenario: entering invalid input.

The alternateiv scenario when the user press number or invalid value.



Precondition: the user plays the game.

Test steps

- Start the game.
- System shows a massage” guess the required word”.
- User press “5”.

Expected

- The system shows a massage “Invalid input”.
- System shows " enter a valid letter ".

UC 2 Start Game by enter valid input

Precondition: none.

Postcondition: the game menu is shown.

Main scenario

1. 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 and quit the game.
3. The Gamer makes the choice to start the game.
4. The system starts the game.

Alternative scenarios

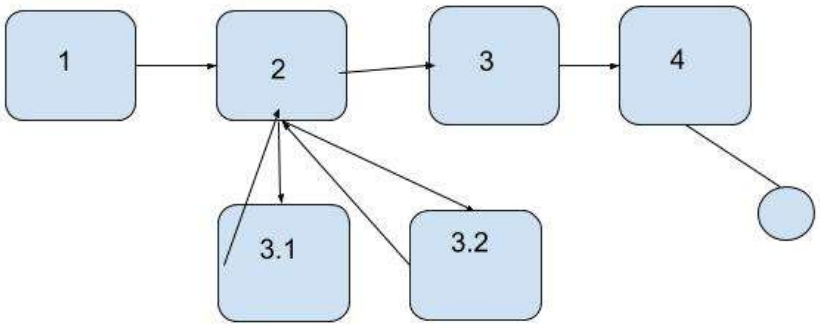
3.1 The Gamer makes the choice to quit the game.

1. The system quits the game.

3.2 Invalid menu choice

1. The system presents an error message.
2. Go to 2.

Activity diagram:

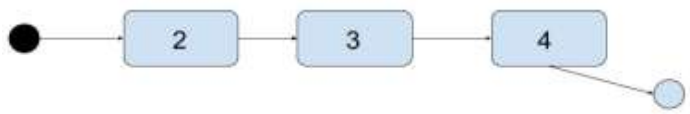


MANUAL TEST-CASES

TC1.2 User start to play the game by entering a valid menu choice.

Use-case: the game started.

Scenario: the user press valid choice to play.



Precondition: the game is running.

Test steps

- Start the game.
- System shows menu
    - 1. Play
    - 2. Results
    - 3. Quit the game

- The player type ”1” and press enter.

Expected

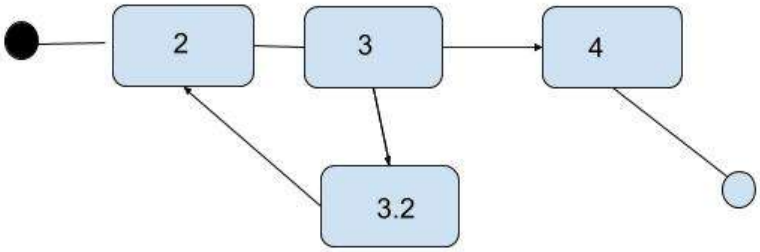
the round finished and the system showed the results.

TC2.2 user wants to quit the game.

Use-case: UC2 Start Game by enter valid input.

Scenario: user input invalid menu choice.

The alternate scenario is the user enters an invalid choice then system will show a message to enter a right choice.



Precondition: user chose to quite the game.

Test steps

- User press “3”
- System shows menu
    - 1. Restart the game.
    - 2. Results
    - 3. Quit the game.
- User input “invalid value” and press enter.

Expected

- System quite the game.
- System shows a massage" choose a valid number ".

Unit Tests

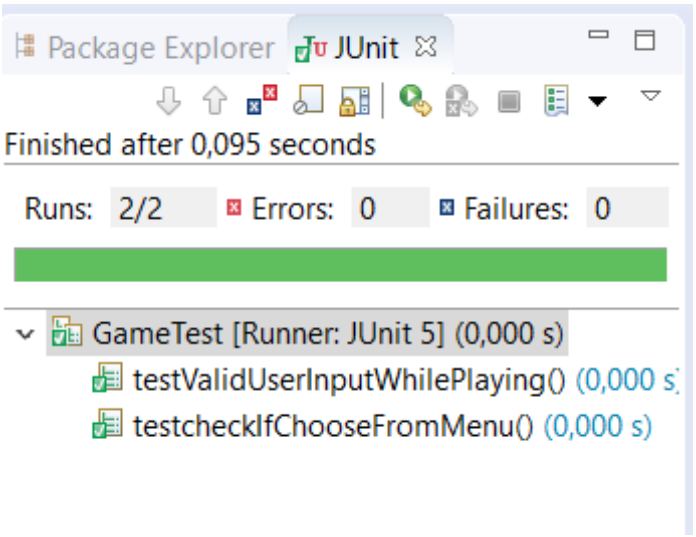
Unit one source code

```
protected boolean validUserInputWhilePlaying(String userInput) {
    if(userInput.length() == 0)
        return false;
    in = userInput.charAt(0);
    boolean isChar = Character.isLetter(in);
    if(userInput.length() == 1 && isChar == true)
        return true;
    else
        return false;
}
```

Unit test 1

```
private String validInputWhilePlaying = "f";
private String inValidInput = "long text with many characters";
private String inValidInputInteger = "3";
private String emptyInput = "";

private String validChoiceFromMenu = "1";
private String validChoiceFromMenu2 = "2";

Game hangMan = new Game();

@Test
void testValidUserInputWhilePlaying() {
    assertEquals(true, hangMan.validUserInputWhilePlaying(validInputWhilePlaying));
    assertEquals(false, hangMan.validUserInputWhilePlaying(inValidInput));
    assertEquals(false, hangMan.validUserInputWhilePlaying(inValidInputInteger));
    assertEquals(false, hangMan.validUserInputWhilePlaying(emptyInput));

}
```

Unit test Result 1



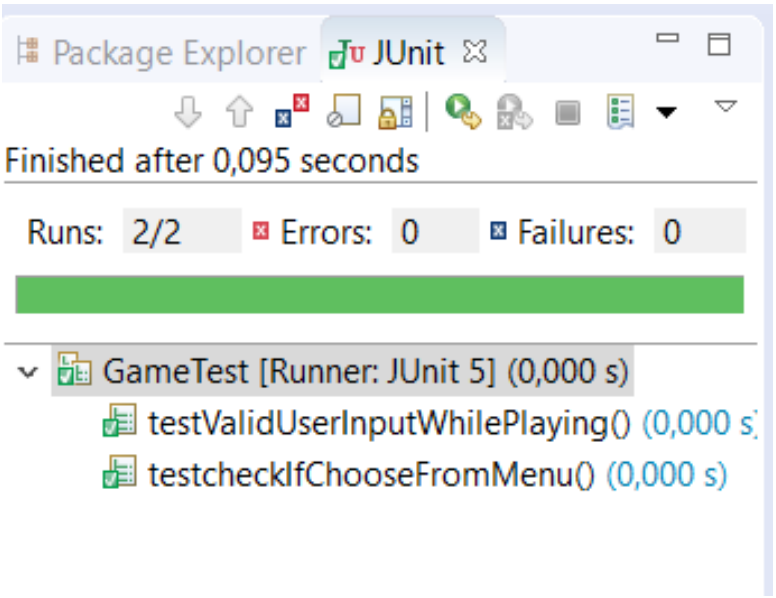
Unit two source code

```
protected boolean checkIfPlayerWantToQuit(String numberChoosen) {
    if(numberChoosen.equals("1")) {
        isStart = true;
        return true;
    }
    else if(numberChoosen.equals("2"))
    {
        isStart = true;
        return true;
    }
    else
        return false;
}
```

Unit test 2

```java
private String validInputWhilePlaying = "f";
private String inValidInput = "long text with many characters";
private String inValidInputInteger = "3";
private String emptyInput = "";

private String validChoiceFromMenu = "1";
private String validChoiceFromMenu2 = "2";

Game hangMan = new Game();

@Test
void testcheckIfChooseFromMenu() {
    assertEquals(true, hangMan.checkIfPlayerWantToQuit(validChoiceFromMenu));
    assertEquals(true, hangMan.checkIfPlayerWantToQuit(validChoiceFromMenu2));
    assertEquals(false, hangMan.checkIfPlayerWantToQuit(inValidInput));
    assertEquals(false, hangMan.checkIfPlayerWantToQuit(inValidInputInteger));
    assertEquals(false, hangMan.checkIfPlayerWantToQuit(emptyInput));

}
```

Unit test Result 2



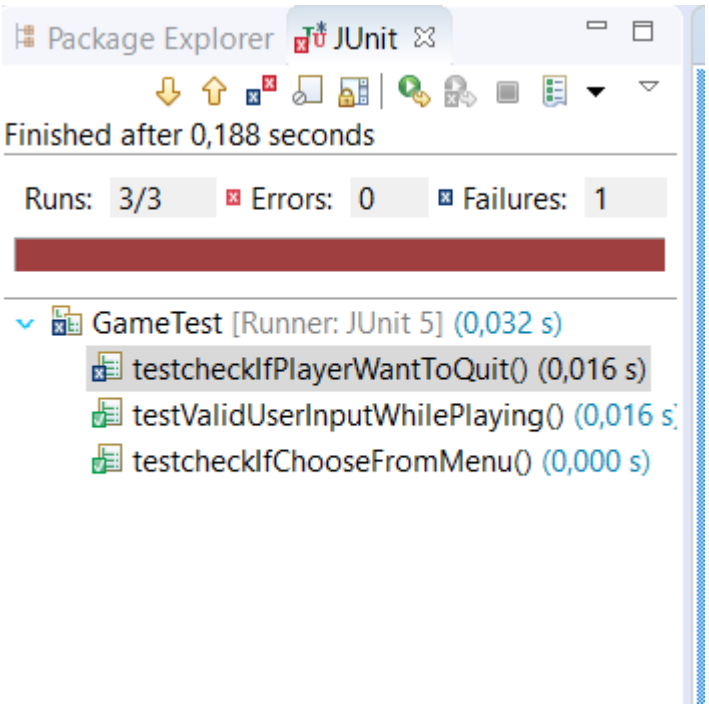
Unit test source code 3

```
protected boolean checkIfPlayerWantToQuit(String numberChoosen) {
    if(numberChoosen.equals("1")) {
        isStart = true;
        return true;
    }
    else if(numberChoosen.equals("2"))
    {
        isStart = true;
        return true;
    }
    else
        return false;
}
```

Unit test 3

```
// the player will press number 2 to quit the game
@Test
void testcheckIfPlayerWantToQuit() {
    assertEquals(true, hangMan.checkIfPlayerWantToQuit("1"));
    assertEquals(false, hangMan.checkIfPlayerWantToQuit("2"));

}
```

Unit test Result 3



Test Report

Test traceability matrix and success

Test	UC1	UC2
TC1.1	1/OK	0
TC1.2	1/OK	0
TC2.1	0	2/Ok
TC2.2	0	2/Ok
COVERAGE & SUCCESS	2/OK	2/Ok

Automated unit test coverage and success

Test	Console View	Game Entity Controller	Main	Game
checkIfPlayerWantToQuit	0	0	0	100%/OK
validUserInputWhilePlaying	0/NA	0/NA	0/NA	100%/OK
testcheckIfPlayerWantToQuit	0	0	0	50%/Fail

Reflection

In this stage of the game I learned a lot.

I improved my skills in coding in addition it was good space to apply the previous information in using classes and methods, and good way to learn how to make a manual testing for the use cases.

It was a good idea to starts our project in software with a simple game like Hang man, actually I didn't find simple at the first especially when I want to find a bug in the code.

The main problem that I suffered from the assignment is the way of writing the questions, in other words I found it so hard to understand what do you really want from the questions though, the solutions was too simple.

In addition, I couldn't match between the information in the book, the lectures and the assignment.