1DV600 - The Hangman Project

Assignment 3 - Testing

Melina Cirverius
Linneaus University

mc222ap@student.lnu.se
https://github.com/MelinaCir/mc222ap-1dv600

2019-03-08

Contents

1 Test Plan

Objective

The objective is to test some of the code for the hangman game that was implemented in the last iteration.

What to test and how

The plan is to test UC1 and UC3 by running dynamic manual test cases. These two use cases were chosen as UC1 is a short but important part of the application as it is the start of the whole hangman game. UC3 on the other hand is the largest use case and also the most important one as it is the one showing how the game is played.

Automated unit tests will also be written for the methods createUnderscores() and evaluateGuess() in the class Hangman. These methods will be tested with at least two tests each. The reason behind choosing these two methods is since they are a crucial part of how the game should be played according to the project plan. The first method makes sure that the word is displayed in the right way for the player to start guessing and the second is a core part of the game as it evaluates the player's guess be comparing letters in the word with the one guessed.

One more unit test will be created for an unfinished method, drawHangman(), that should succeed once that method has been correctly implemented, but will fail at this stage.

Time log

Date finished	Task	Scheduled time	Actual time
2019-02-25	Plan iteration/theme 3	30m	30m
2019-02-25	Create a test plan	1h	45m
2019-03-01	Read chapter 8	2h	3h
2019-02-27	Watch live lecture/workshop on theme 3	2h	2h
2019-03-01	Watch pre-recorded lecture 9	1h 45m	1h 45m
2019-03-05	Watch video on the "Greeter Example"	30m	30m
2019-03-04	Create manual test case for Use case 2	2h	1h
2019-03-04	Create manual test case for Use case 3	2h	1h
2019-03-04	Run manual tests and record the results	2h	30m
2019-03-06	Watch pre-recorded lecture 10	1h 45m	1h 45m
2019-03-06	Watch pre-recorded lecture 10.5	30m	30m
2019-03-06	Watch live lecture/workshop on theme 3	2h	1h 30m
2019-03-07	Create automated unit tests for method createUnderscores()	4h	3h

Date finished	Task	Scheduled time	Actual time
2019-03-07	Create automated unit tests for method evaluateGuess()	4h	2h
2019-03-07	Create automated unit test for unfinished method drawHangman()	2h	1h
2019-03-08	Run automated tests for method (1) and record the results	2h	30m
2019-03-08	Run automated tests for method (2) and record the results	2h	1h
2019-03-08	Run automated tests for method (3) and record the results	2h	30m
2019-03-08	Write reflection	1h	30m
Total time	**Iteration 3**	**35h**	**23h 45m**

This updated planning and time log can also be found in the 1DV600 - Hangman Project Plan.

2 Use Cases to be tested

2.1 Use Case 1 – Start Game

Precondition: none.

Postcondition: The game menu is shown.

Main scenario

1. Starts when the Gamer 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 (see Use Case 2).

Repeat from step 2

Alternative scenarios

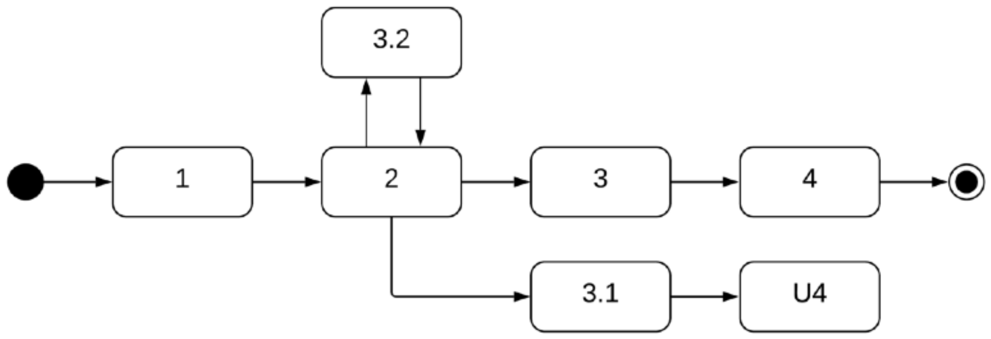
3.1 The Gamer makes the choice to quit the game.

  1. The system quits the game (see Use Case 4)

3.2 Invalid menu choice

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

Activity Diagram



2.2 Use Case 3 – Play Game

Precondition: The difficulty menu is shown.

Postcondition: The game is running.

Main scenario

1. Starts when the Gamer makes the choice of difficulty.

2. The system present a word to guess with the number of letters in the word, and the number of guesses available.
3. The Gamer makes a guess of a letter.
4. The system presents an updated version of the word with the letter, the number of letters left to guess and the number of guesses left.

Repeat from step 3.

5. The Gamer guesses the last letter and so the whole word correctly.
6. The system presents the finished word and the number of guesses used, as well as the option to start a new round.
7. The Gamer makes the choice to quit the game.
8. The system quits the game. (see Use Case 4)

Alternative scenarios

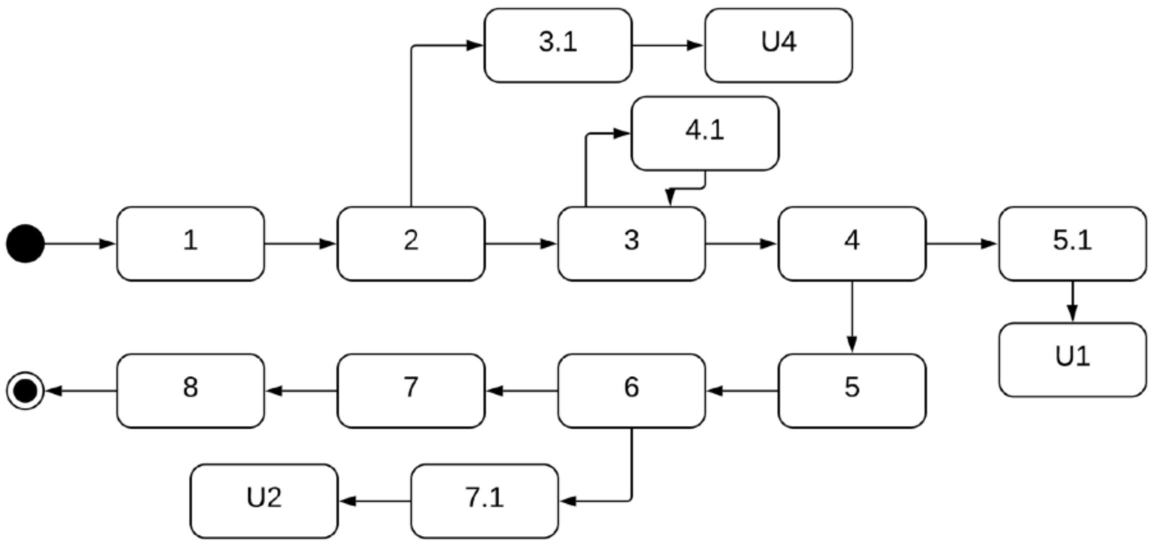
3.1 The Gamer makes the choice to quit the game.
  1. The system quits the game. (see Use Case 4)

4.1 Incorrect guess as the letter is not present in the current word.
  1. The system draws a part of the hangman, and presents the number of guesses left.
  2. Go to 3.

5.1 The Gamer has no guesses left.
  1. The system presents a fully drawn hangman and the correct word.
  2. The Gamer choses to start a new game.
  3. The system shows the game menu. (see use Case 1)

7.1 The Gamer choses to start a new round.
  1. The system shows the difficulty menu. (see Use Case 2)

Activity Diagram

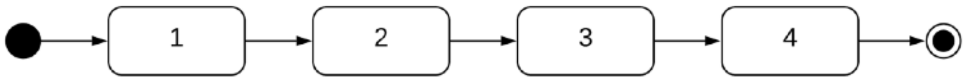


3 Manual Test Cases

TC1.1 Start a game successfully

Use Case: UC1 – Start Game

Scenario: Start a game successfully



The main scenario of UC1 is tested where a user starts a game.

Test steps

- Start the application.
- The system presents “Welcome to Hangman 1.0, Menu, 1. Start Game, 2. Quit Game, Pick a number option:”
- Enter the number “1” and press enter.

Expected

- The system continues to UC3 and shows “Word to guess: _ _ _ _ “

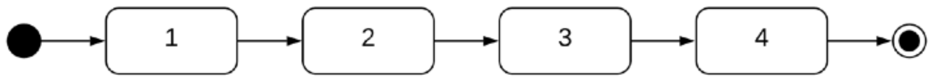
☑ Test successful.

Comments from tester

This test succeeded without any problem. One thing that I noticed was that the test does indeed jump right to UC3, skipping UC2 at this stage as the methods associated with that use case is not implemented correctly yet.

TC3.1 Guess a letter successfully

Use Case: UC3 – Play Game
Scenario: Guess a letter successfully. The first part of UC3 where the system presents a word to guess, the user inputs a letter as a guess and the system presents an updated version of the word.



Precondition: UC1 and UC2 should have been executed successfully, so that the user has chosen to start a game, given a user name and a choice of difficulty. Also a separate instance of the class containing just one word should be used.

Test steps

- Start the application by creating a new instance of the class Hangman with input “LEGO”.
- Execute U1
- The system presents “Word to guess: _ _ _ _ , Number of guesses left: 10”
- Enter the letter “e” and press enter.

Expected

- The system should show “Correct! Word to guess: _ E _ _ , Number of guesses left: 9”
- The system waits for a new input.

☑ Test successful.

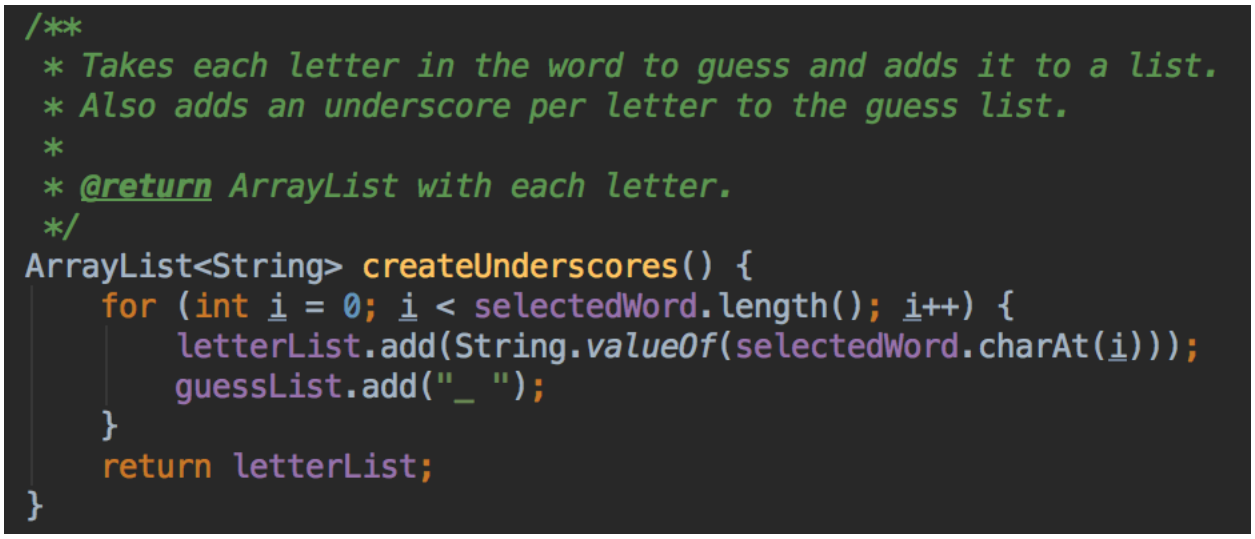
Comments from tester

As the game uses random words generated from a text file, it was necessary to create a separate instance of the class with a pre-determined word for the test.
The result was as expected except “Correct!” is not shown, instead a list of already guessed letters is displayed. In this case, since only one letter is guessed, the list displayed “Guessed letters: E”.

4 Automated Unit Tests

4.1 Method createUnderscores()

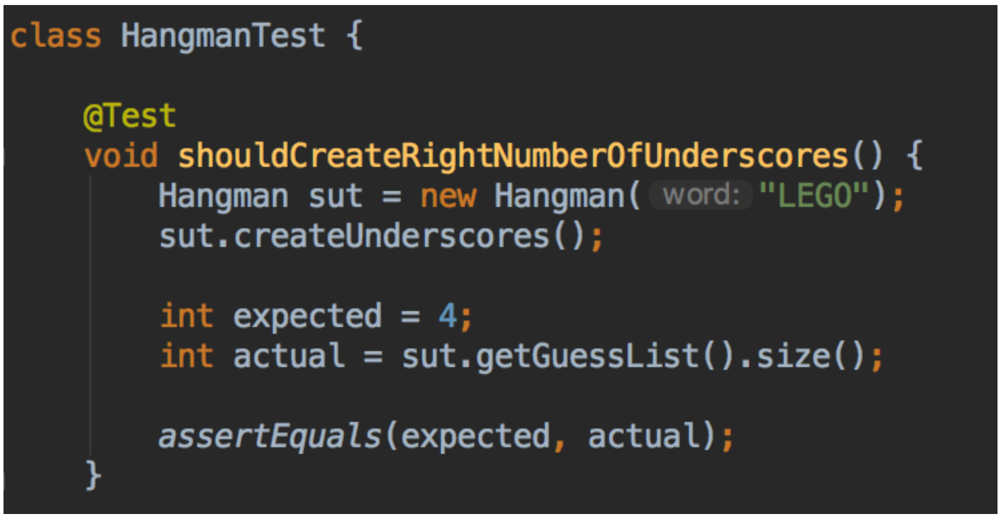
Method to be tested: createUnderscores() in Hangman class



```java
/**
 * Takes each letter in the word to guess and adds it to a list.
 * Also adds an underscore per letter to the guess list.
 *
 * @return ArrayList with each letter.
 */
ArrayList<String> createUnderscores() {
    for (int i = 0; i < selectedWord.length(); i++) {
        letterList.add(String.valueOf(selectedWord.charAt(i)));
        guessList.add("_ ");
    }
    return letterList;
}
```

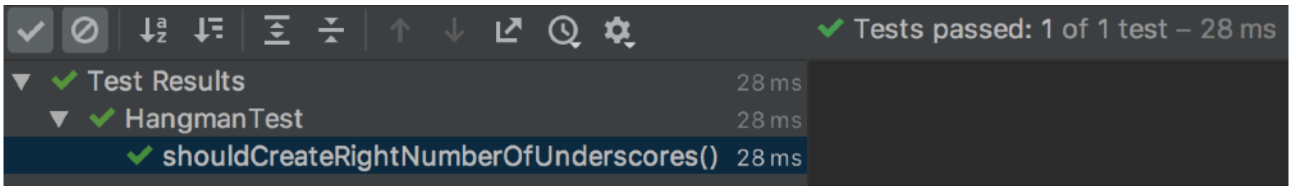
Test 1 – Check number of underscores

Test method shouldCreateRightNumberOfUnderscores() tests if the number of underscores created by the method corresponds to the number of letters in the word to be guessed.



```java
class HangmanTest {

    @Test
    void shouldCreateRightNumberOfUnderscores() {
        Hangman sut = new Hangman( word: "LEGO");
        sut.createUnderscores();

        int expected = 4;
        int actual = sut.getGuessList().size();

        assertEquals(expected, actual);
    }
```

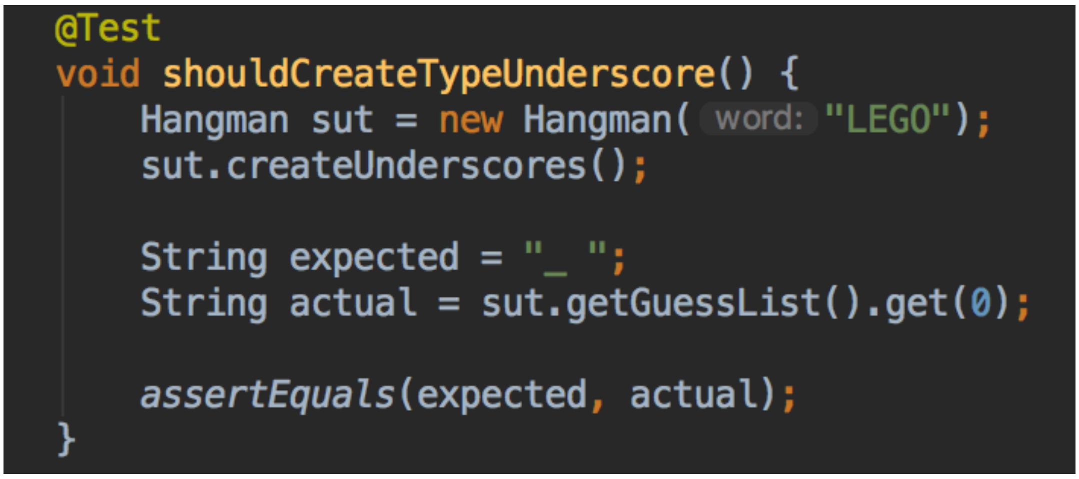
Result:



```
✓ Tests passed: 1 of 1 test – 28 ms
▼ ✓ Test Results                              28 ms
  ▼ ✓ HangmanTest                             28 ms
      ✓ shouldCreateRightNumberOfUnderscores() 28 ms
```

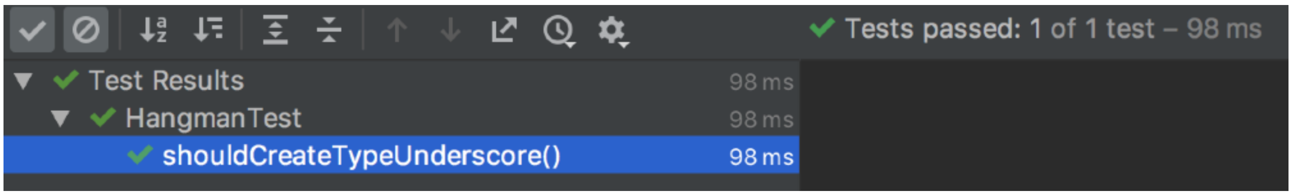
Test 2 – Check value of string created

Test method shouldCreateTypeUnderscore() tests that the method creates the right type/value of the string, so that it is an underscore created per each letter.



```
@Test
void shouldCreateTypeUnderscore() {
    Hangman sut = new Hangman( word: "LEGO");
    sut.createUnderscores();

    String expected = "_ ";
    String actual = sut.getGuessList().get(0);

    assertEquals(expected, actual);
}
```

Result:



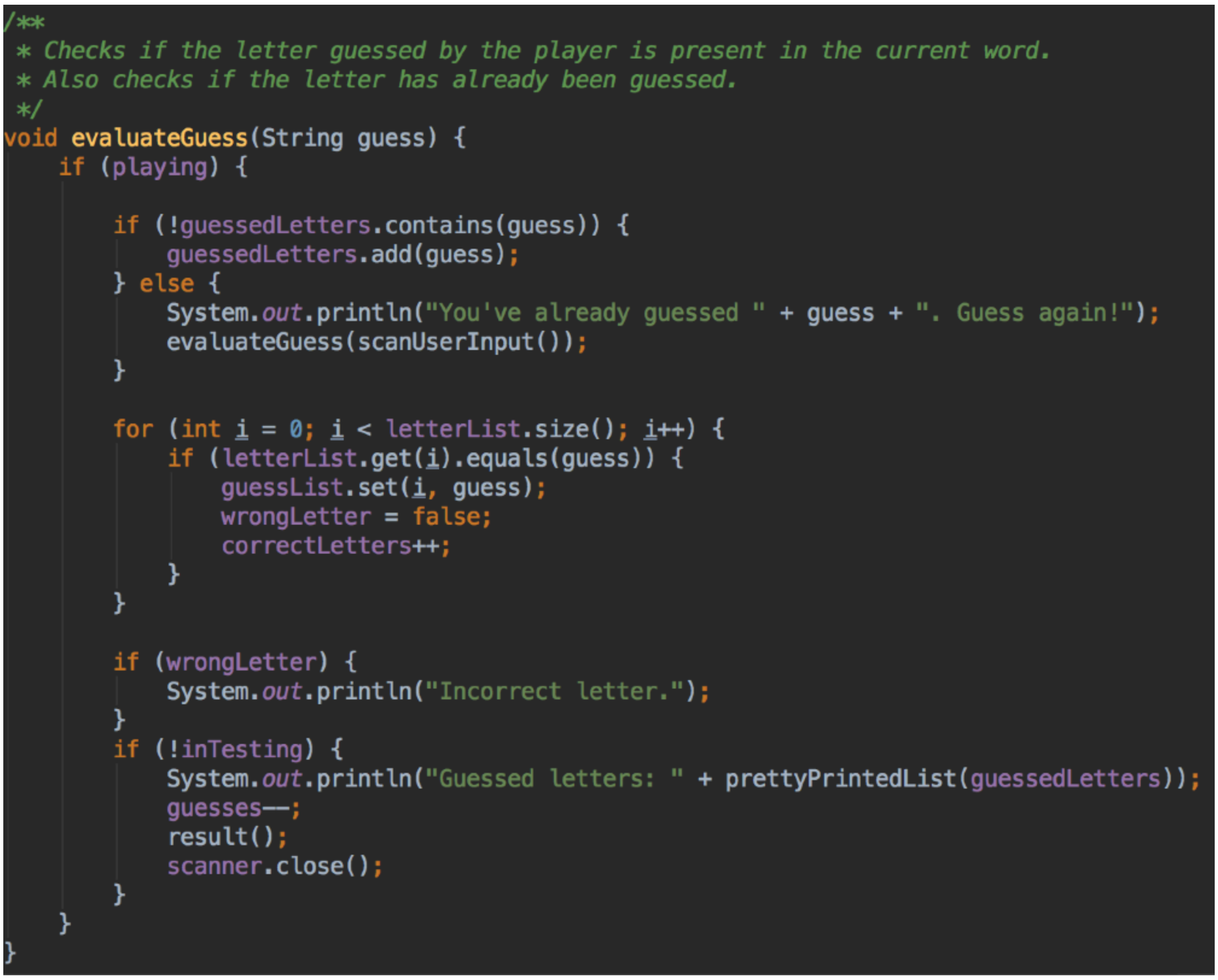
```
✓ Tests passed: 1 of 1 test – 98 ms
▼ ✓ Test Results                          98 ms
  ▼ ✓ HangmanTest                         98 ms
      ✓ shouldCreateTypeUnderscore()      98 ms
```

Comments

These tests proved to be a bit of a challenge to figure out, as how to test this method was not obvious from the start, as well as how to create two different tests for the same method.
The resulting tests created has a great use in making sure this step in the game is handled correctly to that the player knows how many letters are present in the current word. It was also important for this project that the letters are represented with underscores as this is the classic way of playing a hangman game.

4.2 Method evaluateGuess()

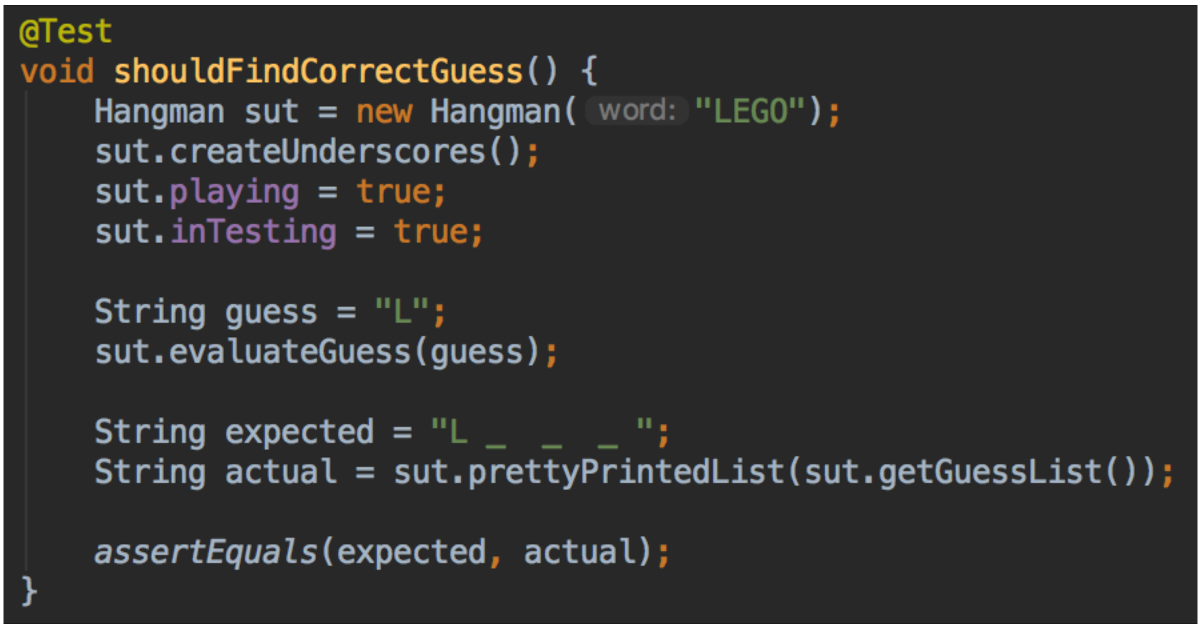
Method to be tested: evaluateGuess(String guess) in Hangman class



```java
/**
 * Checks if the letter guessed by the player is present in the current word.
 * Also checks if the letter has already been guessed.
 */
void evaluateGuess(String guess) {
    if (playing) {

        if (!guessedLetters.contains(guess)) {
            guessedLetters.add(guess);
        } else {
            System.out.println("You've already guessed " + guess + ". Guess again!");
            evaluateGuess(scanUserInput());
        }

        for (int i = 0; i < letterList.size(); i++) {
            if (letterList.get(i).equals(guess)) {
                guessList.set(i, guess);
                wrongLetter = false;
                correctLetters++;
            }
        }

        if (wrongLetter) {
            System.out.println("Incorrect letter.");
        }
        if (!inTesting) {
            System.out.println("Guessed letters: " + prettyPrintedList(guessedLetters));
            guesses--;
            result();
            scanner.close();
        }
    }
}
```

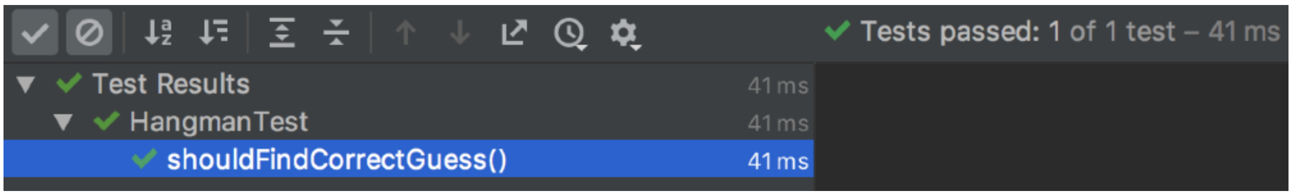
Test 1 – Find correct guess

Test method shouldFindCorrectGuess() tests that when a letter guessed is present in the current word the method finds this to be true, and the list that displays the word to the player updates with the correct letter.



```
@Test
void shouldFindCorrectGuess() {
    Hangman sut = new Hangman( word: "LEGO");
    sut.createUnderscores();
    sut.playing = true;
    sut.inTesting = true;

    String guess = "L";
    sut.evaluateGuess(guess);

    String expected = "L _ _ _ ";
    String actual = sut.prettyPrintedList(sut.getGuessList());

    assertEquals(expected, actual);
}
```

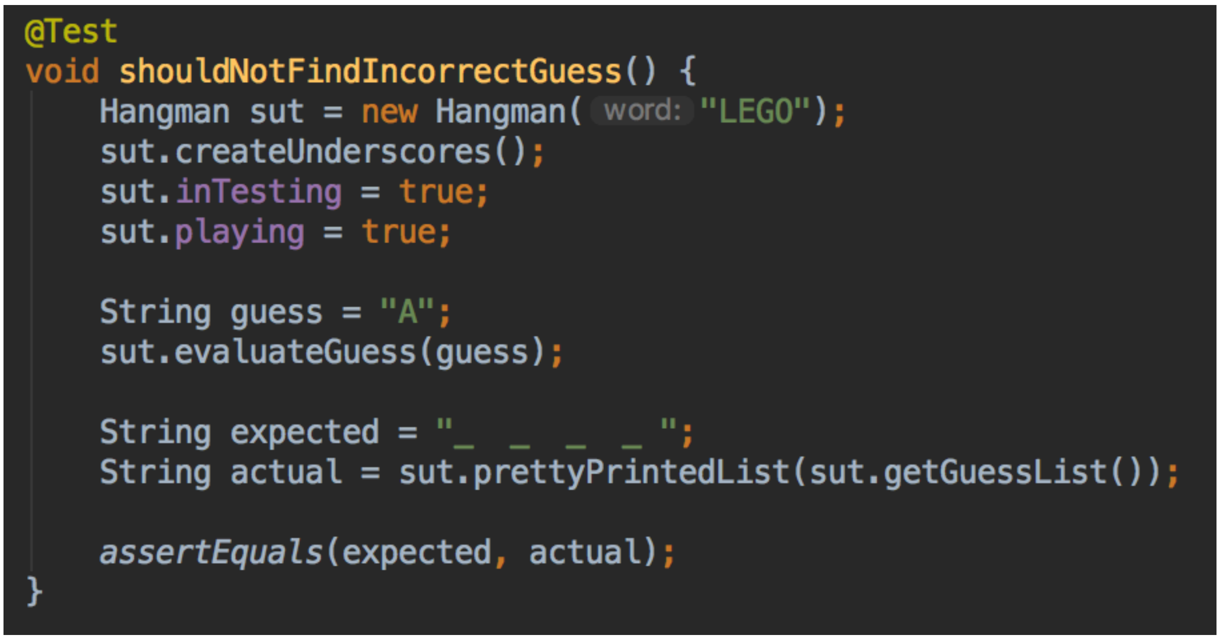
Result:



```
✓ Tests passed: 1 of 1 test – 41 ms
▼ ✓ Test Results                    41 ms
  ▼ ✓ HangmanTest                   41 ms
      ✓ shouldFindCorrectGuess()    41 ms
```

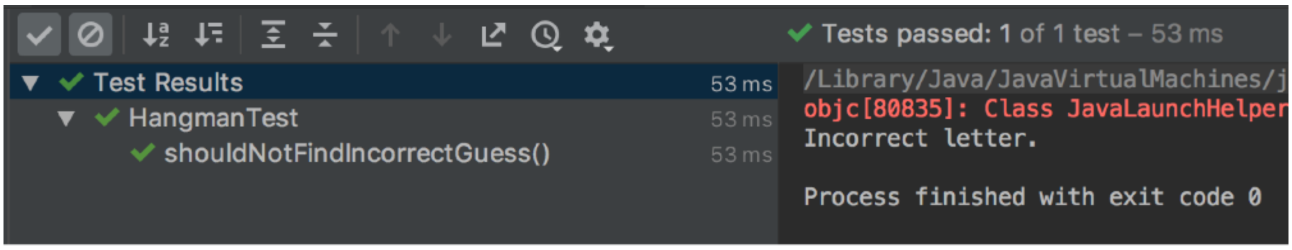
Test 2 – Do not find incorrect guess

Test method shouldNotFindIncorrectGuess() tests that when a letter guessed is not present in the current word the method does not find a correct guess and does not update the word, but instead displays the word in its previous state again for the next guess.



```java
@Test
void shouldNotFindIncorrectGuess() {
    Hangman sut = new Hangman( word: "LEGO");
    sut.createUnderscores();
    sut.inTesting = true;
    sut.playing = true;

    String guess = "A";
    sut.evaluateGuess(guess);

    String expected = "_ _ _ _ ";
    String actual = sut.prettyPrintedList(sut.getGuessList());

    assertEquals(expected, actual);
}
```

Result:



```
✓ Tests passed: 1 of 1 test – 53 ms
▼ ✓ Test Results                        53 ms
  ▼ ✓ HangmanTest                       53 ms
      ✓ shouldNotFindIncorrectGuess()   53 ms
```

```
/Library/Java/JavaVirtualMachines/j
objc[80835]: Class JavaLaunchHelper
Incorrect letter.

Process finished with exit code 0
```

Comments

This method is a core part of the application and the game and as such more tests will be needed to check all the functionality of it. These two first unit tests proved useful as they test both scenarios happening after the player has guessed a letter, when the letter is a correct versus incorrect guess. As the game in its normal state is played by a random word being picked from a text file, the tests had to create a new instance of the Hangman class with a pre-defined word for the tests to work correctly.

4.3 Method drawHangman()

Method to be tested: drawHangman() in Hangman class.

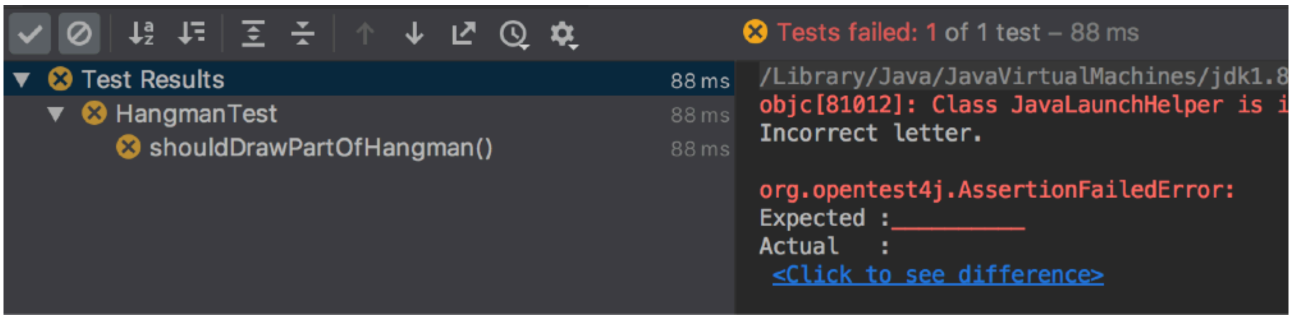
```java
/**
 * Draws next part of the hangman if the guess was incorrect.
 */
String drawHangman() {
    return "";
}
```

Test 1 – Draw part of hangman

Test method shouldDrawFirstPartOfHangman() tests that the first time the player guessed an incorrect letter, the first part of the hangman (the ground) is returned.

```java
@Test
void shouldDrawFirstPartOfHangman() {
    Hangman sut = new Hangman( word: "LEGO");
    sut.createUnderscores();
    sut.inTesting = true;
    sut.playing = true;

    String guess = "A";
    sut.evaluateGuess(guess);

    String expected = "__________";
    String actual = sut.drawHangman();

    assertEquals(expected, actual);
}
```

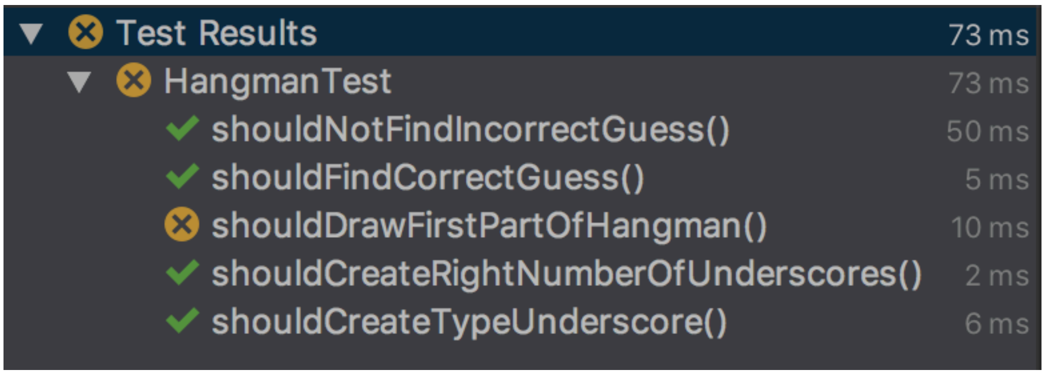
Result:



```
Tests failed: 1 of 1 test – 88 ms

▼ Test Results                          88 ms
  ▼ HangmanTest                         88 ms
      shouldDrawPartOfHangman()         88 ms

/Library/Java/JavaVirtualMachines/jdk1.8
objc[81012]: Class JavaLaunchHelper is i
Incorrect letter.

org.opentest4j.AssertionFailedError:
Expected :__________
Actual   :
 <Click to see difference>
```

5 Reflection

When starting to write the unit tests I found I had to use quite some time to refactor my code for the Hangman game to increase the testability of the code. The code had not been written with testability in mind and a lot more refactoring and reworking of the code will be necessary in iteration four. Testing the code taught me more about how I want the structure in the application to be to create a more functional game. The code in its current state is not written as object oriented as planned and this as well will have to be corrected in the upcoming and final iteration. All in all, testing the code proved to be a learning and important step in the project.



```
▼ ⊗ Test Results                                   73 ms
   ▼ ⊗ HangmanTest                                 73 ms
       ✓ shouldNotFindIncorrectGuess()             50 ms
       ✓ shouldFindCorrectGuess()                   5 ms
       ⊗ shouldDrawFirstPartOfHangman()            10 ms
       ✓ shouldCreateRightNumberOfUnderscores()     2 ms
       ✓ shouldCreateTypeUnderscore()               6 ms
```