HANGMAN GAME TESTING

ADAM ELFSTRÖM
ae223nv@student.lnu.se
github.com/caadel/ae223nv_1dv600
8th March 2019

Contents

1 | Project Introduction

1.1 General information

The Hangman Game is an interactive application that can be run in a terminal/console on any personal computer with Java installed. This application allows users to play a game of "hangman", with the choice of which mode to play with. Depending on which mode the user chooses, rules such as different scoring systems or game length will be applied to the basic game, enhancing the experience.

1.2 Implemented Use Cases

1.2.1 UC1 - Start Game

Precondition: none.
Postcondition: The game is started (See UC2).

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 start playing the game, changing the game mode, view previous scores and quit the game
3. The user chooses to start playing the game
4. The system starts the game (See UC2).

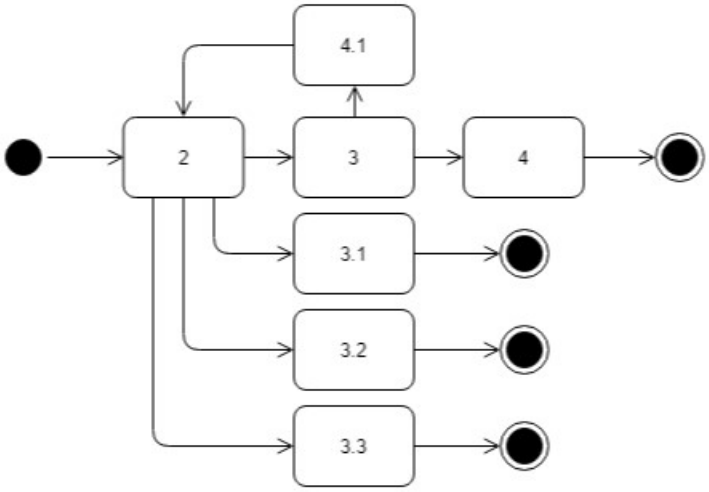
Alternative scenarios

3.1 The user chooses to quit the game
1. The system terminates (See UC3).

3.2 The user chooses to change the game mode
1. The system shows the game mode selection menu (See UC5).

3.3 The user chooses to view previous scores
1. The system shows the previous scores (See UC4).

4.1 Invalid menu choice
1. The system presents an error message
2. Goto 2 in main scenario.



1.2.2 UC2 - Play Game

Preconditions: none.
Postcondition: a victory/loss screen is presented.

Main scenario

1. Starts when the user chooses to play the game
2. The system selects a random word and presents a row of underscores, each representing a letter in the chosen word
3. The user inputs a letter as a guess to what the word contains
4. The guess is correct and all letters have been guessed, so the system presents the victory screen containing a menu with options to play again, save the score (See UC8), go to the main menu or to quit the game (See UC3).

Alternative scenarios

4.1 Correct guess, not all letters have been guessed

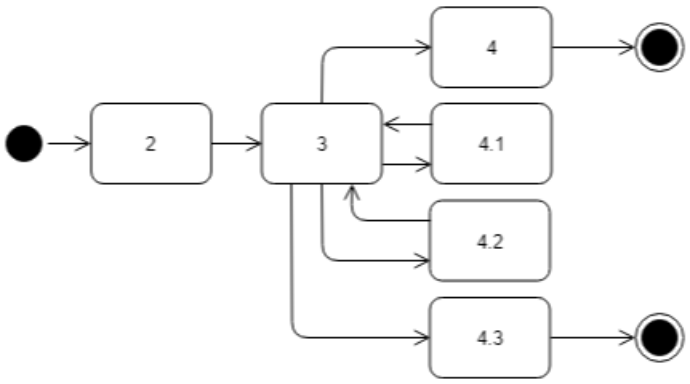
1. The system updates and presents the row of underscores with the correct characters now shown
2. Goto 3 in main scenario.

4.2 Incorrect guess, but it is not the 10th incorrect guess

1. The system adds the guess to and presents presents a list of all incorrect guesses
2. Goto 3 in main scenario.

4.3 Incorrect guess, for the 10th time

1. The system presents the loss screen containing a menu with options to play again, ban the word (See UC7), go back to the main menu or to quit the game (See UC3).



1.2.3 UC3 - Quit Game

Precondition: the game is running.
Postcondition: the game is terminated.

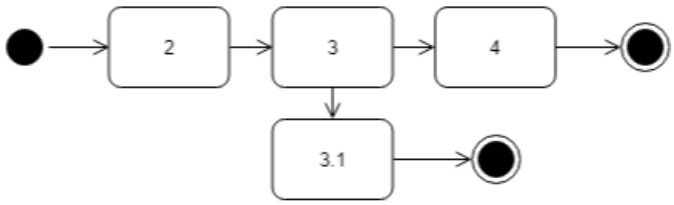
Main scenario

1. 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.



1.2.4 UC5 - Choose game Mode

Precondition: none.
Postcondition: the game mode choice is saved, main menu is presented with the updated game mode.

Main scenario

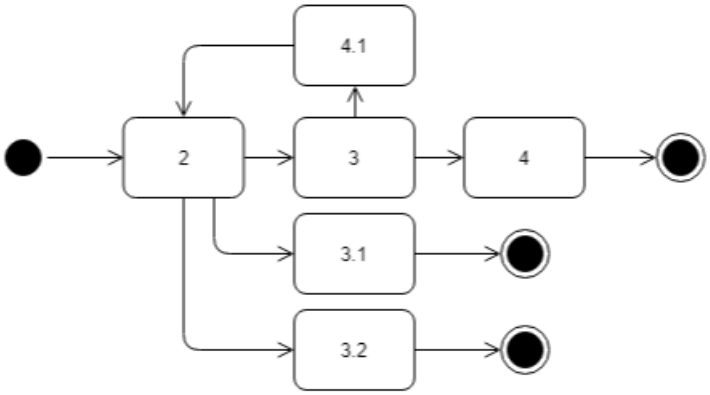
1. Starts when the user wants to choose which game mode to play
2. The system shows all available game modes
3. The user chooses the basic game mode
4. The system saves the game mode choice and presents the main menu.

Alternative scenarios

3.1 The user chooses the game mode where you can specify word length
   1. The system presents the word length selection prompt (See UC5).

3.2 The user chooses the timed or the survival game mode
   1. The system saves the game mode choice and presents the main menu.

4.1 Invalid menu choice
   1. The system presents an error message
   2. Goto 2 in main scenario.



1.2.5 UC6 - Choose Word Length

Precondition: game mode 2 has been selected.
Postcondition: the word length choice is saved, the main menu is presented.

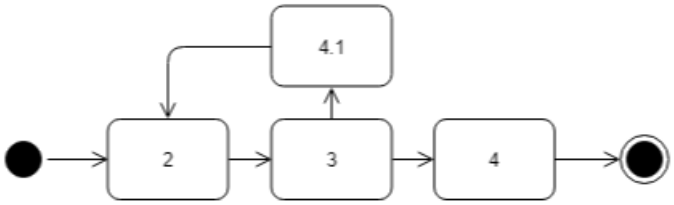
Main scenario

1. Starts when the user wants to choose the length of available words
2. The system prompts the user to specify a length between 2 and 17
3. The user inputs the desired length
4. The system saves the choice and presents the main menu.

Alternative scenarios

4.1 Invalid word length/invalid input

1. The system presents an error message
2. Goto 2 in main scenario.



1.3 Other Use Cases

1.3.1 UC4 - View Scores

Precondition: none.
Postcondition: all previous scores are presented.

Main scenario

1. Starts when the user wants to view the list of previous scores
2. The system presents all previously saved scores and the option to return to the main menu
3. The user chooses to return to the main menu
4. The system shows the main menu.

Repeat from step 2

1.3.2 UC7 - Ban Word

Precondition: the user lost the game.
Postcondition: the last word is removed from the list of available words.

Main scenario

1. Starts when the user wants to ban the last word they were unable to guess
2. The system prompts for confirmation
3. The user confirms
4. The system removes the last word from the list of available words.

Alternative scenarios

3.1 The user does not confirm

1. The system returns to the end of game screen.

1.3.3 UC8 - Save Scores

Precondition: the user won the game.
Postcondition: the score is saved.

Main scenario

1. Starts when the user wants to save their last score
2. The system prompts the user to write their name
3. The user writes their name
4. The system prompts for confirmation
5. The user confirms
6. The system saves the score.

Alternative scenarios

5.1 The user does not confirm

1. The system returns to the end of game screen.

2 | Test Plan

2.1 Objective

The objective of the following tests described in the test plan is to test the implemented code to verify that it works as intended.

2.2 What Will Be Tested And How

The intention is to test all implemented Use Cases (1,2,3,5,6) dynamically. There will be manual Test Case of the running client application that intend to cover all possible paths in UC1 and UC2. These two use cases have been selected for manual testing because they cover the functionality most essential to the product, meaning selecting to start playing from after the program starts and actually playing the game. All paths in these two use cases should be covered by the tests. Additionally, automated unit tests will be written for the application with the intention to cover as many of the methods in each class as possible. The classes that are currently implemented are Main, Word, WordList, Game and Menu. These unit tests will be executed to verify that no obvious errors are present. Due to the small size of the application, the only automated tests that will be created are unit tests. Automated tests for the not yet implemented functionality will be written in the next iteration, alongside their implementation.

2.3 Time Log

The following time log specifies estimated and actual time for each task. Also read Section 8.3 in the Project Plan for tasks completed that are not directly linked to testing.

Task	Estimate	Actual time
Document creation & structuring	30 min	35 min
Basic information & Use Cases	1 h	3 h 30 min
Test Plan	30 min	25 min
Write manual Test Cases	3h	3h 30 min
Draw activity diagrams	20 min	35 min
Run manual tests	20 min	30 min
Update test report	15 min	15 min
Prepare test environment (create fakes/mocks)	1h	1h 45 min
Write and run unit tests	2 h 30 min	2 h 50 min
Write and run failing unit test	10 min	10 min
Run and save result of coverage tool	10 min	20 min
Update test report	30 min	1 h
Write reflection	20 min	20 min

3 | Manual Test Cases

3.1 TC1.1 - Start standard game successfully

Use Case: UC1 - Start Game
Scenario: Start standard game successfully
Description: Main Scenario of UC1 ends with the start of UC2, use is able to start playing the game
Precondition: None
Postcondition: The game is started

Test steps

1. Start the application
2. The system shows the main menu, which consist of the options "1. Play game (words, length)", "2. Change game mode". "3. View scores" and "4. Quit"
3. Press 1 and press enter.

Expected result

1. The system starts the game
2. Output from UC2 - Play Game

Test result

☑ Pass
☐ Fail

Comments

3.2 TC1.2 - Enter game mode selection menu

Use Case: UC1 - Start Game
Scenario: Enter game mode selection menu
Description: Alt Scenario of UC1, user wants to change game mode instead of playing
Precondition: None
Postcondition: The game mode selection menu is shown

Test steps

1. Start the application
2. The system shows the main menu
3. Press 2 and press enter.

Expected result

1. The system shows game mode selection menu.

Test result

☑ Pass
☐ Fail

Comments

3.3 TC1.3 - View saved scores

Use Case: UC1 - Start Game & UC4 - View Scores
Scenario: View saved scores
Description: Alt Scenario of UC1, user wants to view previous scores instead of playing
Precondition: None
Postcondition: The previous scores are shown (See UC4)

Test steps

1. Start the application
2. The system shows the main menu
3. Press 3 and press enter.

Expected result

1. The system shows two lists containing all saved scores from the timed and the survival game mode.

Test result

☐ Pass
☒ Fail

Comments

The expected console output did not match the actual console output. Red message "Scoring system not implemented" was shown instead of a list of scores.

3.4 TC1.4 - Enter game termination menu from main menu

Use Case: UC1 - Start Game
Scenario: Enter game termination menu from main menu
Description: Alt Scenario of UC1, user wants to quit the game instead of playing
Precondition: None
Postcondition: Game termination menu is shown

Test steps

1. Start the application
2. The system shows the main menu
3. Press 4 and press enter.

Expected result

1. The system shows the termination confirmation menu (See UC3, TC3.1 and TC3.2).

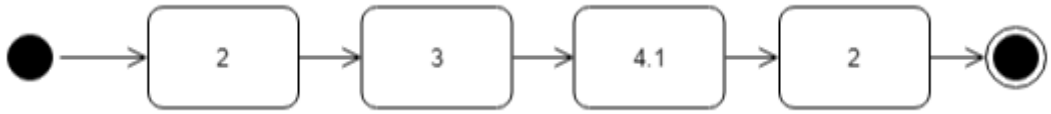
Test result

☑ Pass
☐ Fail

Comments

3.5 TC1.5 - Invalid menu choice force re-prompt

Use Case: UC1 - Start Game
Scenario: Invalid main menu choice forces re-prompt
Description: Alt Scenario of UC1, user enters invalid menu choice and is forced to enter a new menu choice
Precondition: None
Postcondition: The main menu is shown



Test steps

1. Start the application
2. The system shows the main menu
3. Press 5 and press enter.

Expected result

1. The system shows the error message "'5' is an incorrect menu choice"
2. The system removes the error message after a couple of seconds and waits for a new input

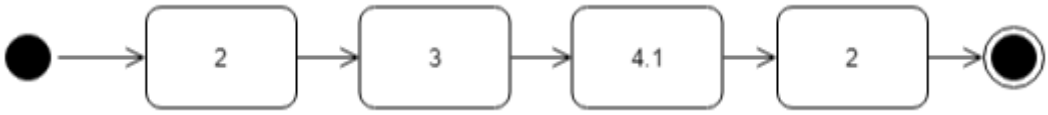
Test result

☑ Pass
☐ Fail

Comments

3.6 TC1.6 - Empty menu choice force re-prompt

Use Case: UC1 - Start Game
Scenario: Empty main menu choice input forces re-prompt
Description: Alt Scenario of UC1, user enters empty menu choice and is forced to enter a new menu choice
Precondition: None
Postcondition: The main menu is shown



Test steps

1. Start the application
2. The system shows the main menu
3. Press enter without having typed anything else.

Expected result

1. The system shows the error message "' ' is an incorrect menu choice"
2. The system removes the error message after a couple of seconds and waits for a new input

Test result

☑ Pass
☐ Fail

Comments

3.7 TC2.1 - Play game successfully guess word with no incorrect guesses

Use Case: UC2 - Play Game
Scenario: Play game successfully and guess the word with no incorrect guesses
Description: Main Scenario and alt scenario 4.1 of UC2, user plays the game and successfully guesses all letters in the chosen word
Precondition: Complete step 1 in TC1.1, then the word "test" must be entered. Steps 2-3 in TC1.1 must then be successfully completed
Postcondition: The end of game screen is shown, containing the words "You win"

Test steps

1. The system shows four rows of text; "Current game mode: One word", "_ _ _ _ _ _", "Incorrect Incorrect guesses left: 10" and "Guess: "
2. Press m and press enter
3. The system shows the same rows of text, but row 2 now reads "m _ _ _ _ _"
4. Press n and press enter
5. The system shows the same rows of text, but row 2 now reads "m _ n n _ _"
6. Press e and press enter
7. The system shows the same rows of text, but row 2 now reads "m _ n n e _"
8. Press a and press enter
9. The system shows the same rows of text, but row 2 now reads "m a n n e _"
10. Press r and press enter

Expected result

1. The system shows 6 rows of text; "- - - - -YOU WIN!- - - - -", "The correct word was manner", "1. Play again with the same settings", "2. Go to main menu", "3. Quit" and "Select 1, 2 or 3:"

Test result

☑ Pass
☐ Fail

Comments

3.8 TC2.2 - Play game successfully guess word with 1 incorrect guess

Use Case: UC2 - Play Game
Scenario: Play game successfully and guess the word with 1 incorrect guesses
Description: Main Scenario, alt scenario 4.1 and alt scenario 4.2 of UC2, user plays the game and successfully guesses all letters in the chosen word but makes a total of 1 incorrect guess
Precondition: Complete step 1 in TC1.1, then the word "test" must be entered. Steps 2-3 in TC1.1 must then be successfully completed. Successfully complete steps 1-9 in TC2.1
Postcondition: The end of game screen is shown, containing the words "You win"

Test steps

1. The system shows four rows of text; "Current game mode: One word", "m a n n e _", "Incorrect guesses left: 10" and "Guess: "
2. Press q and press enter
3. The system shows the same rows of text, but and additional row is shown, reading "Incorrect guesses: q" and row 4 now reads "Incorrect Incorrect guesses left: 9 "
4. Press r and press enter

Expected result

1. The system shows 6 rows of text; "- - - - -YOU WIN!- - - - -", "The correct word was manner", "1. Play again with the same settings", "2. Go to main menu", "3. Quit" and "Select 1, 2 or 3:"

Test result

☑ Pass
☐ Fail

Comments

3.9 TC2.3 - Play game unsuccessfully

Use Case: UC2 - Play Game
Scenario: Play game unsuccessfully
Description: Alt scenario 4.2 and alt scenario 4.3 of UC2, user plays the game and unsuccessfully guesses on the word, guessing wrong a total of 10 times and correct a total of 0 times
Precondition: Complete step 1 in TC1.1, then the word "test" must be entered. Steps 2-3 in TC1.1 must then be successfully completed.
Postcondition: The end of game screen is shown, containing the words "You lose"

Test steps

1. The system shows four rows of text; "Current game mode: One word", "_ _ _ _ _ _", "Incorrect guesses left: 10" and "Guess: "
2. Press q and press enter
3. The system shows the same rows of text, but and additional row is shown, reading "Incorrect guesses: q" and row 4 now reads "Incorrect Incorrect guesses left: 9 "
4. Press y and press enter
5. The system shows the same rows of text, but row 3 now reads "Incorrect guesses: q y" and row 4 now reads "Incorrect Incorrect guesses left: 8"
6. Press w and press enter
7. The system shows the same rows of text, but row 3 now reads "Incorrect guesses: q w y" and row 4 now reads "Incorrect Incorrect guesses left: 7"
8. Press t and press enter
9. The system shows the same rows of text, but row 3 now reads "Incorrect guesses: q t w y" and row 4 now reads "Incorrect Incorrect guesses left: 6"
10. Press s and press enter
11. The system shows the same rows of text, but row 3 now reads "Incorrect guesses: q s t w y" and row 4 now reads "Incorrect Incorrect guesses left: 5"
12. Press c and press enter
13. The system shows the same rows of text, but row 3 now reads "Incorrect guesses: c q s t w y" and row 4 now reads "Incorrect Incorrect guesses left: 4"
14. Press i and press enter
15. The system shows the same rows of text, but row 3 now reads "Incorrect guesses: c i q s t w y" and row 4 now reads "Incorrect Incorrect guesses left: 3"
16. Press h and press enter

17. The system shows the same rows of text, but row 3 now reads "Incorrect guesses: c h i q s t w y" and row 4 now reads "Incorrect Incorrect guesses left: 2"
18. Press k and press enter
19. The system shows the same rows of text, but row 3 now reads "Incorrect guesses: c h i k q s t w y" and row 4 now reads "Incorrect Incorrect guesses left: 1"
20. Press z and press enter

Expected result

1. The system shows 6 rows of text; "- - - - -YOU LOSE!- - - - -", "The correct word was manner", "1. Play again with the same settings", "2. Go to main menu", "3. Quit" and "Select 1, 2 or 3:"

Test result

☑ Pass
☐ Fail

Comments

3.10 TC2.4 - Play game unsuccessfully with one correct guess

Use Case: UC2 - Play Game
Scenario: Play game unsuccessfully with one correct guess
Description: Alt scenario 4.1, 4.2 and 4.3 of UC2, user plays the game and unsuccessfully guesses on the word, guessing wrong a total of 10 times and correct a total of 1 times
Precondition: Complete step 1 in TC1.1, then the word "test" must be entered. Steps 2-3 in TC1.1 must then be successfully completed. Then successfully complete steps 1-19 in TC2.3.
Postcondition: The end of game screen is shown, containing the words "You lose"

Test steps

1. The system shows four rows of text; "Current game mode: One word", "_ _ _ _ _ _", "Incorrect guesses left: 1", "Incorrect guesses: c h i k q s t w y" and "Guess: "
2. Press m and press enter
3. The system shows the same rows of text, but row 2 now reads "m _ _ _ _ _"
4. Press p and press enter

Expected result

1. The system shows 6 rows of text; "- - - - -YOU LOSE!- - - - -", "The correct word was manner", "1. Play again with the same settings", "2. Go to main menu", "3. Quit" and "Select 1, 2 or 3:"

Test result

☑ Pass
☐ Fail

Comments

4 | Automated Unit Tests

4.1 WordTest

The automated tests for the Word class are split up into three images.

```
class WordTest {
    Word sut;

    @Test
    void addCharactersToWordShouldUpdateWordLength() {
        sut = new Word();

        sut.add("Text");
        int expected = 4;
        int actual = sut.length();
        assertEquals(expected, actual);

        sut.add('X');
        expected++;
        actual = sut.length();
        assertEquals(expected, actual);
    }
    @Test
    void addCharactersToWordShouldUpdateStringContent() {
        sut = new Word();

        assertTrue(sut.toString() == "");

        sut.add("Text");
        assertTrue(sut.toString().equals("Text"));

        sut.add('X');
        assertTrue(sut.toString().equals("TextX"));
    }
    @Test
    void emptyWordShouldReturnZeroLength() {
        sut = new Word();
        assertTrue(sut.length() == 0);
    }
    @Test
    void nonEmptyWordShouldNotReturnZeroLength() {
        sut = new Word();
        sut.setWord("Test");
        assertFalse(sut.length() == 0);
    }
```

```
    @Test
    void hiddenWordShouldBeTwiceTheLengthOfWord() {
        sut = new Word("Test");
        assertTrue(sut.getHiddenWord().length() == sut.length()*2);
    }
    @Test
    void hiddenWordShouldCorrectlyConsistOfSpacesAndUnderscores() {
        // Checks that the hidden word is correct ("Test" = "_ _ _ _ ")
        sut = new Word("Test");

        for (int i = 0; i < sut.toString().length(); i++) {
            if (i%2 == 0)
                assertTrue(sut.getHiddenWord().charAt(i) == '_');
            else
                assertTrue(sut.getHiddenWord().charAt(i) == ' ');
        }
    }
    @Test
    void compareWordToWordWithLargerUnicodeShouldReturnFirstWordIsSmaller() {
        sut = new Word("Test");
        assertTrue(sut.compareTo(new Word("Tes")) > 0);
    }
    @Test
    void compareWordToWordWithSmallerUnicodeShouldReturnFirstWordIsLarger() {
        sut = new Word("Test");
        assertTrue(sut.compareTo(new Word("Testt")) < 0);
    }
    @Test
    void compareWordToWordWithSameUnicodeShouldReturnEqual() {
        sut = new Word("Test");
        assertTrue(sut.compareTo(new Word("Test")) == 0);
    }
    @Test
    void shouldReturnTrueIfWordContainsCharacter() {
        sut = new Word("Test");
        assertTrue(sut.contains('T'));
    }
    @Test
    void shouldReturnFalseIfWordDoesNotContainCharacter() {
        sut = new Word("Test");
        assertFalse(sut.contains('F'));
    }
```

```
    @Test
    void shouldShowCharacterInHiddenWordIfWordContainsCharacter() {
        sut = new Word("Test");
        sut.updateHiddenWordIfContains('e');

        assertTrue(sut.getHiddenWord().toString().equals("_ e _ _ "));

        sut = new Word("Test-case");
        sut.updateHiddenWordIfContains('e');

        assertTrue(sut.getHiddenWord().toString().equals("_ e _ _ - _ _ _ e "));
    }

}
```

4.2 WordListTest

```
class WordListTest {
    WordListMock sut;

    @Test
    void shouldThrowExceptionOnEmptyList() {
        sut = new WordListMock();

        assertThrows(NullPointerException.class, () -> sut.chooseRandomWord());
        assertThrows(NullPointerException.class, () -> sut.chooseRandomWordOfLength(2));
    }
    @Test
    void shouldReturnWordOnFilledList() {
        sut = new WordListMock();
        sut.fillWithNouns();

        assertTrue(sut.chooseRandomWord().compareTo(new Word("ox")) == 0);
    }
    @Test
    void shouldReturnWordOfCorrectSizeIfCalledOnFilledList() {
        sut = new WordListMock();
        sut.fillWithNouns();

        assertEquals(sut.chooseRandomWordOfLength(2).length(), 2);
        assertEquals(sut.chooseRandomWordOfLength(8).length(), 8);
        assertEquals(sut.chooseRandomWordOfLength(17).length(), 17);
    }
    @Test
    void shouldThrowExceptionIfCalledWithIncorrectWordSize() {
        sut = new WordListMock();
        sut.fillWithNouns();

        assertThrows(IllegalArgumentException.class, () -> sut.chooseRandomWordOfLength(0));
        assertThrows(IllegalArgumentException.class, () -> sut.chooseRandomWordOfLength(20));
    }
    @Test //Failing test
    void shouldReturnTrueIfWordListContainsWord() {
        sut = new WordListMock();
        sut.fillWithNouns();
        Word wordToSearchFor = new Word("ox");

        assertTrue(sut.contains(wordToSearchFor));
    }
}
```

4.3 PrintFormatterTest

```
class PrintFormatterTest {

    PrintFormatter sut;
    @Test
    void shouldReturnColoredStringForCorrectInput() {
        sut = new PrintFormatter();

        assertEquals(sut.color("Test", "white"),"\033[0mTest\033[0m");
        assertEquals(sut.color("Test", "red"),"\033[31mTest\033[0m");
        assertEquals(sut.color("Test", "green"),"\033[32mTest\033[0m");
        assertEquals(sut.color("Test", "yellow"),"\033[33mTest\033[0m");
        assertEquals(sut.color("Test", "blue"),"\033[34mTest\033[0m");
        assertEquals(sut.color("Test", "purple"),"\033[35mTest\033[0m");
        assertEquals(sut.color("Test", "cyan"),"\033[36mTest\033[0m");
    }
    @Test
    void shouldThrowExceptionForWrongInput() {
        sut = new PrintFormatter();

        assertThrows(IllegalArgumentException.class, () ->  sut.color("Test", "black"));
    }

}
```

5 | Test Report

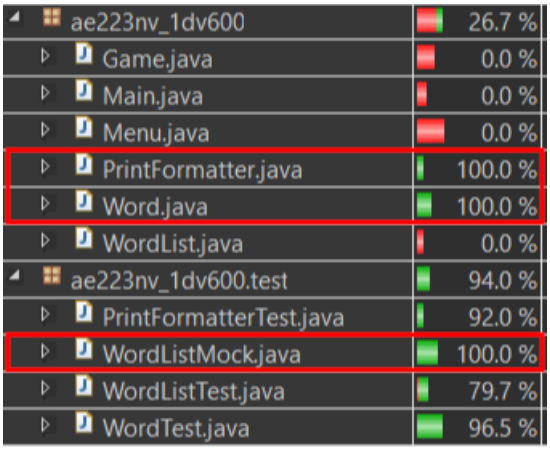
5.1 Manual Testing

Test	UC1	UC2
TC1.1	1/OK	0
TC1.2	1/OK	0
TC1.3	1/FAIL	0
TC1.4	1/OK	0
TC1.5	1/OK	0
TC1.6	1/OK	0
TC2.1	0	1/OK
TC2.2	0	1/OK
TC2.3	0	1/OK
TC2.4	0	1/OK
COVERAGE & SUCCESS	6/FAIL	4/OK

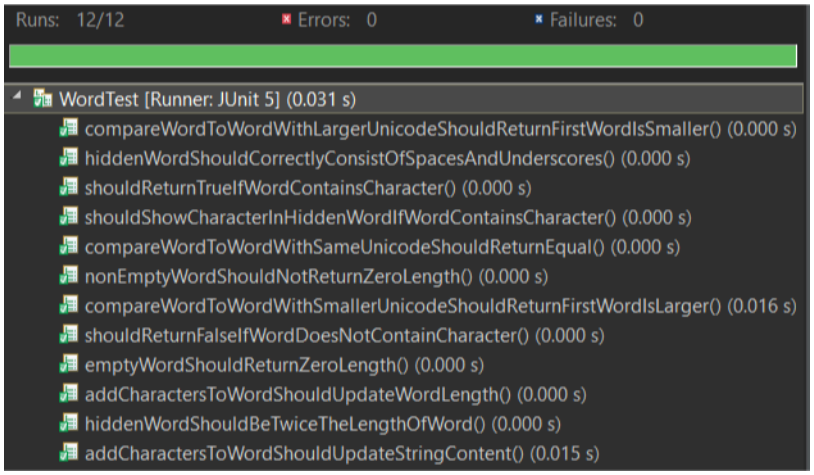
5.2 Automated Unit Testing

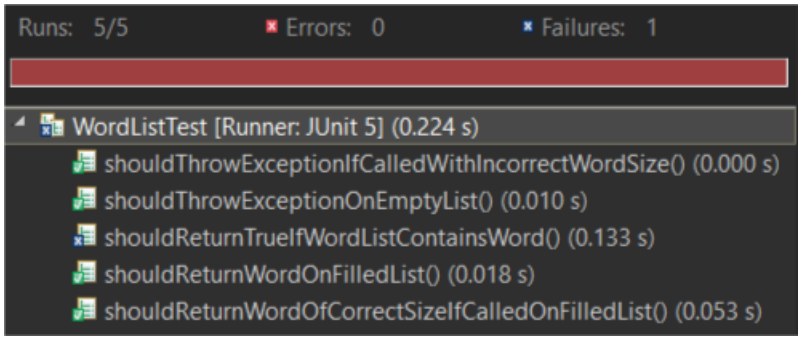
Test	PrintFormatter	Word	WordListMock
WordTest	0%/NA	100%/OK	0%/NA
WordListTest	0%/NA	27.6%/OK	100%/FAIL
PrintFormatterTest	100%/OK	0%/NA	0%/NA
COVERAGE & SUCCESS	100%/OK	100%/OK	100%/FAIL

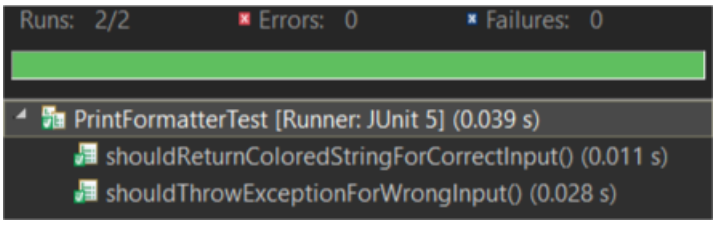
The image below shows test coverage result from JUnit 5, run in Eclipse IDE. The classes highlighted in red were tested with unit tests.



5.2.1 Unit test results







5.3 Comments

TC1.3 gave erroneous results because the functionality tested has not yet been implemented correctly. Manual tests covered UC1, but result was over all a failure because of TC1.3. Manual test covered UC2 with satisfactory results. Automated testing gave 100% coverage for tested classes, but result was not satisfactory since only 3 out of 6 classes were tested.

6 | Reflection

Many things were originally difficult to test due to bad structure, so I had to re-write a large chunk of code and move some features around in the menus. This also meant re-writing use cases to fit the new code. The result was that one single use case no longer included a chain of 3 use cases, meaning that manual test cases became easier to write. The automated tests did not improve as much because I did not have time, or sufficient knowledge, two re-write everything that was not testable (user input, console output, random word selection) as dependency injections. I therefore gave up on the idea of testing everything, and only tested the 3 easiest classes. For one of them, I created a copy of the class to be tested and modified it to be testable. This copy/mock was unit tested instead of the original class.