



TEST DOCUMENTATION

Testing



MARCH 9, 2020

AMINA HAMZA
Linneaus University

Table of Contents

Objective	2
What to test and how.....	2
Manual test-cases	2
TC1 Start Game	3
Use case: UC1.....	3
TC2 Play single player game	3
TC2.1 Play new game, earn two points	4
TC2.2 Return to previous game.....	4
TC3 Play multiplayer version	5
Use case: UC3	5
TC3.1 Player 1 wins	5
TC3.2 Player 2 wins	6
TC3.3 Change word after first input.....	6
TC4 Play game	7
TC4.1 Play game and win	7
TC4.2 Play game and loose.....	8
TC4.3 Enter same character several times during game	9
TC4.4 Enter invalid input	10
TC4.5 Return to menu	10
TC5 Remove word from noun-list.....	11
TC5.1 Remove word present in list	11
TC5.2 Initiate removing of word but do not confirm.	12
TC5.3 Remove word not present in list.....	13
TC6 Quit game	13
TC6.1 From main menu	13
TC6.2 From single player game	14
TC6.3 From multiplayer game	14
TC6.4 From remove word	15
Test Report for Manual Test case	15
Automated Test Cases	15
TC1 Test Game.java	16
TC1.1 getWord	16
TC1.2 setWord.....	16
TC1.3 guessLetter	17
TC1.4 gameSucceded.....	18
TC1.5 gameLost	19
TC1.6 underscoresToString.....	19
TC1.7 checkWord.....	20
Test report for TC 1	21
TC2 Test GuessedLetters.java	22
TC2.1 toString	22
TC2.2 addLetter	23
TC2.3 LetterAlreadyGuessed	24
Test report for TC2.....	25
TC3 Test Words.java	25

TC3.1 getRandomWord.....	25
TC3.2 indexOfWord	26
TC3.3 removeWord,addWord	26
Test report for TC3	27
TC4 Test MultitplayerGame.java	28
TC4.1 checkWord.....	28
Test report for TC4	29
Automated test report	29
Reflections	29
Time report	29

Objective

The objective is to test all the code that was implemented. The testing will be limited to functionality and not equality.

What to test and how

There will only be dynamic tests since static tests are not a requirement at this stage. The use cases one to four will be tested using manual testing. There should be a test plan for each of these use cases, but these test plans should also go through different paths and not only the main scenario of the use case. This will be done manually since it mainly is input and output on the screen which is hard for a computer to evaluate.

As many methods as possible in the implementation of these use cases will be tested using JUnit. All methods will not be able to be tested using JUnit since many require user-input or only print text in the console without returning anything. The methods which are not tested with JUnit are hopefully covered by the manual testing.

Use case five was excluded due to time constraints. This was the least prioritized use case since you can always turn the game off by simply closing the environment it is run in, this should however be tested when there is time.

Checkboxes, test matrix and comments from testers are included in the end of manual testing section.

Manual Test Cases

Note: Before any test cases the code in Words.java must be changed so the txt-file test.txt is used which only contains the word "Test".

TC1 Start Game

Use case: UC1

TC1 should see that the program is able to start and that the correct menu choices are shown. Also, that at least one menu choice works (this will indirectly be tested in further test cases) and that an invalid input result in an error message and that the user is able to enter a new menu choice.

TC1.1 Start game and initiate new game

1. Start the system
2. System should show **"1. Play new game", "2. Return to previous game", "3. Play multiplayer version", "4. Remove word from predefined list", "5. Terminate program"** and **"Choose one of the above"**.
3. Enter **"1"** and press enter.
4. System should show **"Initiating a new game will erase any previous game."**,

"Are you sure you want to initiate a new game?", "1. Yes", "2. No".

TC1.2 Try invalid input

See that an invalid input result in an error message and that user is able to enter a new menu choice.

Test steps

1. Start the system
2. System should show **"1. Play new game", "2. Return to previous game", "3. Play multiplayer version", "4. Terminate program"** and **"Choose one of the above"**.
3. Enter **"6"** and press enter.
4. System should show: **"Invalid input, enter a menu choice:"**
5. Enter **"X"** and press enter
6. System should show: **"Invalid input, enter a number:"**
7. Enter **"?"** and press enter
8. System should show: **"Invalid input, enter a number:"**

TC2 Play single player game

TC2 should see that menu choices **"1. Play new game"** and **"2. Return to previous game"** works. That the user is asked to confirm initiating a new game before previous game is erased. If the player win the game he should earn a point and get to guess on another word. See that player is able to return to a previous game after returning to menu.

Precondition: Before TC2.2 is performed TC4 should be performed.

TC2.1 Play new game, earn two points.

TC2.1 test that player is able to play a new game but is asked to confirm to erase eventual earlier game.

Test steps

1. Start system and choose menu choice 1.
2. System should show **"Initiating a new game will erase any previous game."**,
"Are you sure you want to initiate a new game?", "1. Yes", "2. No"
3. Enter **"1"** and press enter.
4. A game of hangman where no letters are guessed, and no parts of the hangman figure should be displayed, perform **TC4.1**
5. System should show **"Your highscore is: 1"** and **"Enter any character to continue"**.
6. Enter **"x"** and press enter
7. A game of hangman where no letters are guessed, and no parts of the
hangman figure should be displayed, perform **TC4.1**
8. Enter **"x"** and press enter.
9. System should show **"Your highscore is: 2"** and **"Enter any character to continue"**.
10. Enter **"x"** and press enter.
11. A game of hangman where no letters are guessed, and no parts of the
hangman figure should be displayed, perform **TC4.2**
12. System should show **"Your highscore was: 2"** and **"Enter any character to continue"**.
13. Enter **"x"** and press enter.
14. System should return to main menu.

TC2.2 Return to previous game

TC2.2 test that player is able to return to a previous game. Precondition: TC2.1 and TC4 must have been tested first. **Test steps**

1. Perform step **1-10** in **TC2.1** to earn 2 points, perform step **1-4** in **TC4.1** to guess a letter and then perform **TC4.5** to return to menu.
2. Enter **"2"** and press enter.

3. System should show **"Press 1 to return to the Menu", "Press 2 to terminate program", "T _ _ t", empty lines, " _____", "Guessed letters:", "Enter menu choice or guess a letter:"**
4. Enter **"e"** and press enter.
5. Enter **"s"** and press enter.
6. System should show **a stick man raising his arms, "Yay, the man survived! Correct word: Test", "Your highscore is: 3", "Enter any character to continue"**.

TC3 Play multiplayer version

Use case: UC3

TC3 should see that player is able to play a multiplayer version and that it works as intended.

TC4 must be tested before the last part of TC3 can be performed.

TC3.1 Player 1 wins

Use case: UC3

Player 2 should fail to guess the correct word and loses the game.

Precondition: TC4 must be tested for the four last steps in this test case to be able to be performed.

Test steps

1. Start system and choose menu choice 3.
2. System should show **"Multiplayer version mean that one player enters a word and the other player get to guess the letters of the word. If player 2 manage to guess the word that means he won, however if he does not player 1 win!", "Player 1 enter a word:"**
3. Enter the word **"test"**.
4. System should show **"Player 1 have now entered a word!", "Press 1 to play game with this word", "Press 2 to change the entered word"**.
5. Enter **"1"** and press enter.
6. System should show **"Player 2 should now guess.", "Enter any character to start game"**
7. Enter **"x"** and press enter.
8. System should display a hangman game (UC 4).
9. Guess the letters **"q", "w", "r", "y", "u", "i", "o", "p" and "z"** so the game is lost.
10. System should show **"Player 1 won the game! Congratulations!", "Enter any character to continue"**.
11. Enter **"x"** and press enter.

12. System should return to main menu.

TC3.2 Player 2 wins

Use case: UC3

Player 2 guess all the letters in the word correct at first try and win the game.

Precondition: TC4 must be tested for the four last steps in this test case to be able to be performed.

Test steps

1. Start system and choose menu choice 3.
2. System should show **"Multiplayer version mean that one player enters a word and the other player get to guess the letters of the word. If player 2 manage to guess the word that means he won, however if he does not player 1 win!", "Player 1 enter a word:"**
3. Enter the word **"test"**.
4. System should show **"Player 1 have now entered a word!", "Press 1 to play game with this word", "Press 2 to change the entered word"**.
5. Enter **"1"** and press enter.
6. System should show **"Player 2 should now guess.", "Enter any character to start game"**
7. Enter **"x"** and press enter.
8. System should display a hangman game (UC 4).
9. Guess the letters **"t", "e" and "s"**.
10. System should show **"Player 2 won the game! Congratulations!", "Enter any character to continue"**.
11. Enter **"x"** and press enter.
12. System should return to main menu.

TC3.3 Change word after first input

Use case: UC3

Player 1 should be able to change the word that was entered before player 2 get to guess.

Test steps

1. Start system and choose menu choice 3.
2. System should show **"Multiplayer version mean that one player enters a word and the other player get to guess the letters of the word. If player 2 manage to guess the word that means he won, however if he does not player 1 win!", "Player 1 enter a word:"**

3. Enter the word **"test"**.
4. System should show **"Player 1 have now entered a word!", "Press 1 to play game with this word", "Press 2 to change the entered word"**.
5. Enter **"2"** and press enter.
6. System should show **"Player 1 enter a word:"**.

TC4 Play game

Use case: UC4

TC4 tests the actual game of hangman. That the player is able to guess letters and that the program displays correctly where in the word it is placed or, if not part of the word, adds the letter to guessed letters and draw new part of hangman. This testcase should also see that guesses are not case sensitive. In single player game the word is always "Test" during testing.

Precondition: A game must be able to be initiated.

TC4.1 Play game and win

Use case: UC4

Player guess all the letters in the word correct at first try and win the game.

Precondition: A game must be able to be initiated.

Test steps

1. Start system and choose menu choice 1 and confirm creating new game.
2. System should show:
 - - **"Press 1 to return to the Menu"**
 - - **"Press 2 to terminate program"**
 - - Four separated underscores (one for each letter in the word),
 - - Five empty lines where the hangman figure is going to be drawn and a line with underscores representing the ground
 - - **"Guessed letters: "**
 - - **"Enter a menu choice or guess a letter:"**
3. Enter the letter **"t"** and press enter.
4. System should show the same as before, but the first underscore should be changed to a **"T"** and the fourth to a **"t"**.
5. Enter the letter **"S"** and press enter.
6. The third underscore should be changed to a **"s"**
7. Enter the letter **"e"** and press enter.
8. System should show a stickman raising his arms under a pole and the text **"Yay, the man survived! Correct word: Test"**
9. System should return to previous state.

TC4.2 Play game and loose

Use case: UC4

Player guess wrong letter 9 times so that the entire hangman is drawn and the game is lost.

Precondition: A game must be able to be initiated.

Test steps

1. Start system and choose menu choice 1 and confirm creating new game.
2. System should show:
 - - **"Press 1 to return to the Menu"**
 - - **"Press 2 to terminate program"**
 - - Four separated underscores (one for each letter in the word)
- - Five empty lines where the hangman figure is going to be drawn and a line with underscores representing the ground
- - **"Guessed letters: "**
- - **"Enter a menu choice or guess a letter:"**
3. Enter the letter **"q"** and press enter.
4. System should add **"q"** after **"Guessed letters: "** and add the vertical pole in the

hangman figure:
5. Enter the letter **"w"** and press enter.
6. System should add **"w"** after **"Guessed letters: "** and add the horizontal line in the hangman figure.
7. Enter the letter **"r"** and press enter.
8. System should add **"r"** after **"Guessed letters: "** and add the rope in the

hangman figure
9. Enter the letter **"y"** and press enter.
10. System should add **"y"** after **"Guessed letters: "** and add the head in the hangman figure.
11. Enter the letter **"u"** and press enter.
12. System should add **"u"** after **"Guessed letters: "** and add the body in the

hangman figure.
13. Enter the letter **"i"** and press enter.
14. System should add **"i"** after **"Guessed letters: "** and add the left arm in the hangman figure.
15. Enter the letter **"o"** and press enter.
16. System should add **"o"** after **"Guessed letters: "** and add the right arm in the hangman figure.

17. Enter the letter **"p"** and press enter.
18. System should add **"p"** after **"Guessed letters: "** and add the left leg in the
hangman figure.
19. Enter the letter **"a"** and press enter.
20. System should show the complete hangman figure and the text **"Oh no, he died!
Correct word: Test"**
21. System should return to previous state.

TC4.3 Enter same character several times during game

Use case: UC4

Player guesses the same character more than once, game should only react the first time.

Precondition: A game must be able to be initiated.

Test steps

1. Start system and choose menu choice 1 and confirm creating new game.
2. System should show:
 - - **"Press 1 to return to the Menu"**
 - - **"Press 2 to terminate program"**
 - - Four separated underscores (one for each letter in the word)
 - - Five empty lines where the hangman figure is going to be drawn and a line with underscores representing the ground
 - - **"Guessed letters: "**
 - - **"Enter a menu choice or guess a letter:"**
3. Enter the letter **"q"** and press enter.
4. System should add **"q"** after **"Guessed letters: "** and add the vertical pole in the
hangman figure:
 5. Enter the letter **"q"** and press enter.
 6. Nothing should happen.
 7. Enter the letter **"w"** and press enter.
 8. System should add **"w"** after **"Guessed letters: "** and add the horizontal line in the
hangman figure.
 9. Enter the letter **"w"** and press enter.
 10. Nothing should happen.
 11. Enter the letter **"s"** and press enter.
 12. The third underscore should be changed to a **"s"**
 13. Enter the letter **"s"** and press enter.
 14. Nothing should happen.
 15. Enter: **"z", "x", "c", "v", "b", "n", "m"**.

16. The game should be lost.

TC4.4 Enter invalid input

Use case: UC4

Player enters invalid input during game. Precondition: A game must be able to be initiated.

Test steps

1. Start system and choose menu choice 1 and confirm creating new game.
2. System should show:
 - - **"Press 1 to return to the Menu"**
 - - **"Press 2 to terminate program"**
 - - Five empty lines where the hangman figure is going to be drawn and a line

with underscores representing the ground
 - - Four separated underscores (one for each letter in the word),
 - - **"Guessed letters: "**
 - - **"Enter a menu choice or guess a letter:"**
3. Enter the text **"qq"** and press enter.
4. System should show **"Invalid input, enter a letter or a menu choice:"** and waits for new input.
5. Enter the text **"3"** and press enter.
6. System should show **"Invalid input, enter a letter or a menu choice:"** and

waits for new input.
7. Enter the text **"+"** and press enter.
8. System should show **"Invalid input, enter a letter or a menu choice:"** and waits for new input.
9. Press enter, without any input.
10. System should show **"Invalid input, enter a letter or a menu choice:"** and waits for new input.
11. Enter the letter **"q"** and press enter.
12. System should add **"q"** after **"Guessed letters: "** and add the vertical pole in the

hangman figure.

TC4.5 Return to menu

Use case: UC4

Player chooses to return to menu during game. Precondition: A game must be able to be initiated.

Test steps

1. Start system and choose menu choice 1 and confirm creating new game.
2. System should show:
 - - **"Press 1 to return to the Menu"**
 - - **"Press 2 to terminate program"**
 - - Four separated underscores (one for each letter in the word),
 - - Five empty lines where the hangman figure is going to be drawn and a line

with underscores representing the ground

- - **"Guessed letters: "**
 - - **"Enter a menu choice or guess a letter:"**
3. Enter the text **"1"** and press enter.
 4. System should show: **"Are you sure you want to go back to the menu?"**,

"Your game will be saved until a new game is initiated.", "1. Yes", "2. No"
 5. Enter **"2"** and press enter.
 6. System should return to the game.
 7. Enter the text **"1"** and press enter.
 8. System should show: **"Are you sure you want to go back to the menu?", "Your game will be saved until a new game is initiated.", "1. Yes", "2. No"**
 9. Enter **"1"** and press enter.
 10. System should return to the main menu.

TC5 Remove word from noun-list

Use case: UC5

TC5 should see that player is able to remove words from the predefined list.

TC5.1 Remove word present in list

Use case: UC5

Player chooses to remove a word in main menu and removes the word "Test"

Precondition: Tester must have access to scr folder with txt-file. File must contain the word "Test".

Test steps

1. Start system and choose "4. Remove word from predefined list".
2. System should show **"Press 1 to return to menu", "Press 2 to terminate program"** and **"Enter the word to be removed or a menu choice"**.

3. Enter **"Test"** and press enter.
4. System should show: **"Are you sure you want to remove the word: Test", "1. Yes" and "2. No"**.
5. Enter **"1"** and press enter.
6. System should show: **"The word Test is now removed" and "Enter any character to continue:"**.
7. Enter **"x"** and press enter.
8. System should show **"Press 1 to return to menu", "Press 2 to terminate program" and "Enter the word to be removed or a menu choice"**.
9. Enter **"Test"** and press enter.
10. System should show **"Word Test is not part of the list!" and "Enter any character to continue"**.
11. Enter **"x"** and press enter.
12. System should show **"Press 1 to return to menu", "Press 2 to terminate program" and "Enter the word to be removed or a menu choice"**.
13. Enter **"1"** and press enter.
14. System should return to main menu.
15. Open **test.txt** and see that the file is empty, add the word **"Test"** for upcoming tests to work.

TC5.2 Initiate removing of word but do not confirm.

Use case: UC5

Player chooses to remove a word in main menu and enter a word that is present in the list but when asked to confirm press no.

Precondition: Tester must have access to scr folder with txt-file.

Test steps

1. Start system and choose **"4. Remove word from predefined list"**.
2. System should show **"Press 1 to return to menu", "Press 2 to terminate program" and "Enter the word to be removed or a menu choice"**.
3. Enter **"Test"** and press enter.
4. System should show: **"Are you sure you want to remove the word: Test", "1. Yes" and "2. No"**.

5. Enter "2" and press enter.
6. The system should show: **"The word Test was not removed"** and **"Enter any character to continue: "**.
7. Enter "x" and press enter.
8. System should show **"Press 1 to return to menu", "Press 2 to terminate program"** and **"Enter the word to be removed or a menu choice"**.
9. Enter **"Test"** and press enter.
10. System should show: **"Are you sure you want to remove the word: Test", "1. Yes"** and **"2. No"**.
11. Open **test.txt** and see that the word **"Test"** is left.

TC5.3 Remove word not present in list

Use case: UC5

Player try to remove a word not present in the list Precondition:

Test steps

1. Start system and choose "4. Remove word from predefined list".
2. System should show **"Press 1 to return to menu", "Press 2 to terminate program"** and **"Enter the word to be removed or a menu choice"**.
3. Enter **"Example"** and press enter.
4. System should show **"Word Test is not part of the list!"** and **"Enter any character to continue"**.
5. Enter "x" and press enter.
6. System should show **"Press 1 to return to menu", "Press 2 to terminate program"** and **"Enter the word to be removed or a menu choice"**.

TC6 Quit game

Use case: UC6

TC6 should see that player is able to terminate program from different states of the game

TC6.1 From main menu

Use case: UC6

Player chooses to quit game from main menu.

Precondition: System should be running and main menu is shown **Test steps**

1. Enter "5" and press enter.

2. System should show: **"Terminating the program means all previous games will be lost"**, **"Are you sure you want to terminate the program?"**, **"1. Yes"** and **"2. No"**.
3. Enter **"2"** and press enter.
4. System should return to main menu.
5. Enter **"5"** and press enter.
6. System should show: **"Terminating the program means all previous games will be lost"**, **"Are you sure you want to terminate the program?"**, **"1. Yes"** and **"2. No"**.
7. Enter **"1"** and press enter.
8. System should be terminated.

TC6.2 From single player game

Use case: UC6

Player chooses to quit game from a single player game.

Precondition: TC2 must have been tested.

Test steps

1. Start system and choose **"2. return to previous game"**.
2. System should show a single player game of hangman.
3. Enter **"2"** and press enter.
4. System should show: **"Terminating the program means all previous games will be lost"**, **"Are you sure you want to terminate the program?"**, **"1. Yes"** and **"2. No"**.
5. Enter **"1"** and press enter.
6. System should be terminated.

TC6.3 From multiplayer game

Use case: UC6

Player chooses to quit game from a multiplayer game. Precondition: TC3 must have been tested.

Test steps

1. Start system and choose **"3. Play multiplayer version"**.
2. Perform step **1-7** in **TC3.1**.
3. Enter **"2"** and press enter.
4. System should show: **"Terminating the program means all previous games will be lost"**, **"Are you sure you want to terminate the program?"**, **"1. Yes"** and **"2. No"**.
5. Enter **"1"** and press enter.
6. System should be terminated.

TC6.4 From remove word

Use case: UC6

Player chooses to quit game after going to the menu choice "remove word"

Precondition: TC5 must have been tested.

Test steps

1. Start system and choose "4. Remove word from predefined list".
2. Enter "2" and press enter.
3. System should show: **"Terminating the program means all previous games will be lost"**, **"Are you sure you want to terminate the program?"**, **"1. Yes"** and **"2. No"**.
4. Enter "1" and press enter.
5. System should be terminated.

Test Report for Manual Test case

Test	UC1	UC2	UC3	UC4	UC5	UC6
TC1.1	1/OK	0	0	0	0	0
TC1.2	1/OK	0	0	0	0	0
TC2.1	1/OK	1/OK	0	1/OK	0	0
TC2.2	1/OK	1/OK	0	1/OK	0	0
TC3.1	1/OK	0	1/OK	1/OK	0	0
TC3.2	1/OK	0	1/OK	1/OK	0	0
TC3.3	1/OK	0	1/OK	0	0	0
TC4.1	1/OK	1/OK	0	1/OK	0	0
TC4.2	1/OK	1/OK	0	1/OK	0	0
TC4.3	1/OK	1/OK	0	1/OK	0	0
TC4.4	1/OK	1/OK	0	1/OK	0	0
TC4.5	1/OK	1/OK	0	1/OK	0	0
TC5.1	1/OK	0	0	0	1/OK	0
TC5.2	1/OK	0	0	0	1/OK	0
TC5.3	1/OK	0	0	0	1/OK	0
TC6.1	1/OK	0	0	0	0	1/OK
TC6.2	1/OK	1/OK	0	1/OK	0	1/OK
TC6.3	1/OK	0	1/OK	1/OK	0	1/OK
TC6.4	1/OK	0	0	0	1/OK	1/OK
COVERAGE & SUCCESS	0/OK	8/OK	4/OK	11/OK	4/OK	4/OK

Automated Test Cases

This part of the document shows the results of the automated test-cases. Before each test a brief description of the method is given and a figure of the test is appended.

Before any tests is performed the code in Words.java must be changed to this:


```

public class Words {

    //private String searchCode = "/Users/mummy/Desktop/ah224uq_1dv600/Hangman/src/nounlist 2.txt"; // Real noun listComment out during testing
    private String searchCode = "/Users/mummy/Desktop/ah224uq_1dv600/Hangman/src/test.txt"; // Use during testing fake noun list1
}

```

TC1 Test Game.java

Each testcase in TC1 tests a method from the class Game. Before each sub-testcase in TC1 a new instance of Game is created.

```

14 public class TestGame {
15
16     private Game sut;
17
18     @Before
19     public void setUp() { sut = new Game(); }
20
21
22

```

TC1.1 getWord

During testing the TestGetWord should always return the word "Test".

```

// word is always "Test" in current textfile
@Test
public void testGetWord() {
    String expected = "Test";

    String actual = sut.getWord();

    assertEquals(expected, actual);
}

```

Figure 1 Check that the correct word during testing is returned

TC1.2 setWord

During testing TestGetWord should accept words that consists of letters and dashes, invalid word should cause the method to throw an exception.

```

// setWord(String newWord)
@Test
public void testSetWord() {
    String expected = "Expected";
    sut.setWord(expected);

    String actual = sut.getWord();

    assertEquals(expected, actual);
}

```

Figure 2 Test a word that only consists of letters

```

@Test
public void testSetWordWithDash() {
    String expected = "In-Law";
    sut.setWord(expected);

    String actual = sut.getWord();

    assertEquals(expected, actual);
}

```

Figure 3 Test a word that consists of letters and a dash

```

@Test(expected = IllegalArgumentException.class)
public void testSetWordInvalidWord() {
    String invalidWord = "#";

    sut.setWord(invalidWord); // this should throw an IllegalArgumentException
}

```

Figure 4 Test a word that consists of an unaccepted sign

TC1.3 guessLetter

The guessLetter method replaces the underscores when the player guesses the correct letter. If the player guesses the wrong letter methods which is part of other classes, the test will be performed elsewhere.

```

@Test
public void testRightGuess() {
    sut.guessLetter( input: "t");
    String expected = " T _ _ t";

    String actual = sut.underscoresToString();

    assertEquals(expected, actual);

    sut.guessLetter( input: "e");
    expected = " T e _ t";

    actual = sut.underscoresToString();

    assertEquals(expected, actual);
}

```

Figure 5 Testing correct guesses

```
@Test
public void testWrongGuess() {
    sut.guessLetter( input: "y");
    sut.guessLetter( input: "v");
    sut.guessLetter( input: "r");
    sut.guessLetter( input: "w");
    sut.guessLetter( input: "u");
    sut.guessLetter( input: "j");
    sut.guessLetter( input: "o");
    sut.guessLetter( input: "k");
    sut.guessLetter( input: "b");
    String expected = " _ _ _ _ ";

    String actual = sut.underscoresToString();

    assertEquals(expected, actual);
}
```

Figure 6 Testing incorrect guesses which should not affect the underscores

TC1.4 gameSucceeded

This method should return true if the player has guessed all the letters in the word.

```
@Test
public void testGameSucceededFalse() {
    boolean expected = false;
    sut.guessLetter( input: "t");
    sut.guessLetter( input: "r");
    sut.guessLetter( input: "y");

    boolean actual = sut.gameSucceeded();

    assertEquals(expected, actual);
}
```

Figure 7 Test for when the method should return false

```
@Test
public void testGameSucceededTrue() {
    boolean expected = true;
    sut.guessLetter( input: "t");
    sut.guessLetter( input: "e");
    sut.guessLetter( input: "s");

    boolean actual = sut.gameSucceeded();

    assertEquals(expected, actual);
}
```

Figure 8 Test for when the method should return true

TC1.5 gameLost

This method should return true if player is out of guesses and have not guessed the correct word.

Figure 9 Test for when method should return false

```
@Test
public void testGameLostTrue() {
    boolean expected = true;
    sut.guessLetter( input: "k");
    sut.guessLetter( input: "g");
    sut.guessLetter( input: "c");
    sut.guessLetter( input: "v");
    sut.guessLetter( input: "y");
    sut.guessLetter( input: "n");
    sut.guessLetter( input: "w");
    sut.guessLetter( input: "q");
    sut.guessLetter( input: "d");// nine different guesses = out of guesses

    boolean actual = sut.gameLost();

    assertEquals(expected, actual);
}

// underscoresToString()
```

Figure 10 Test for when method should return true

TC1.6 underscoresToString

This method should return a string with as many underscores as there are letters in the word. If the word consists of dashes, then the underscores should be replaced by a dash in the same position.

```

@Test
public void testUnderscoresToString() {
    String expected = " _ _ _ _";

    String actual = sut.underscoresToString();

    assertEquals(expected, actual);
}

```

Figure 11 Test for a word that only consist of letter

```

@Test
public void testUnderScoresToStringWithDash() {
    sut.setWord("In-Law");
    String expected = " _ _ - _ _ _ |";

    String actual = sut.underscoresToString();

    assertEquals(expected, actual);
}

```

Figure 12 Test for a word that consist of letters and a dash

TC1.7 checkWord

This method is designed to return true if word is valid otherwise false.

```

@Test
public void testCheckWordValidWord() {
    boolean expected = true;

    boolean actual = sut.checkWord("Example");

    assertEquals(expected, actual);
}

```

Figure 13 Testing when method should return true

```

@Test
public void testCheckWordValidWordWithDash() {
    boolean expected = true;

    boolean actual = sut.checkWord("In-Law");

    assertEquals(expected, actual);
}

```

Figure 14 Testing that method return true when word contains a dash

```

@Test
public void testCheckWordInvalidWord() {
    boolean expected = false;

    boolean actual = sut.checkWord("*,*");

    assertEquals(expected, actual);
}

```

Figure 15 Testing for when method should return false

Test report for TC 1

The screenshot displays an IDE window with a test report for TC 1. The report shows 15 tests in total, with 14 passed and 1 failed. The failed test is 'testUnderScoresTostringWithDash', which failed due to an assertion error. The error message indicates that the expected result was '<_ _ _ _ _>' but the actual result was '<_ _ _ _ _>'. The IDE also shows the source code for the test method, which is a JUnit test for the 'testCheckWordInvalidWord' method.

Test Name	Duration	Status
TestGame (HangmanGame)	98 ms	Failed
testUnderscoresTostring	20 ms	Passed
testRightGuess	12 ms	Passed
testGameSucceededTrue	1 ms	Passed
testCheckWordInvalidWord	1 ms	Passed
testGameLostfalse	2 ms	Passed
testSetWordInvalidWord	3 ms	Passed
testGameLostTrue	2 ms	Passed
testUnderScoresTostringWithDash	31 ms	Failed
testCheckWordValidWord	7 ms	Passed
testWrongGuess	3 ms	Passed
testGetWord	8 ms	Passed
testSetWordWithDash	1 ms	Passed
testGameSucceededFalse	3 ms	Passed
testCheckWordValidWordWithDash	2 ms	Passed
testSetWord	2 ms	Passed

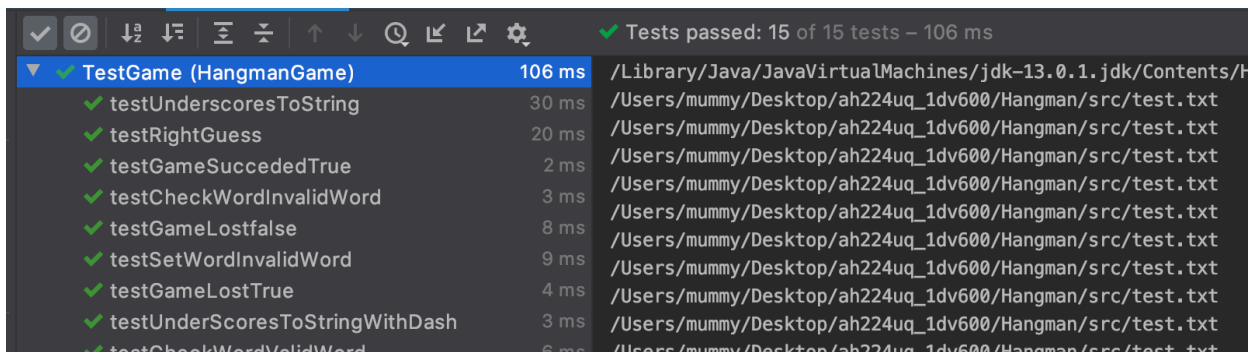
Tests failed: 1, passed: 14 of 15 tests - 98 ms

org.opentest4j.AssertionFailedError: expected: <_ _ _ _ _> but was: <_ _ _ _ _>
Expected : <_ _ _ _ _>
Actual : <_ _ _ _ _>
<Click to see difference>

<5 internal calls>
at HangmanGame.TestGame.testUnderScoresTostringWithDash(TestGame.java:189) <20 internal calls>
at com.intellij.rt.junit.IdeaTestRunner\$Repeater.startRunnerWithArgs(IdeaTestRunner.java:33)
at com.intellij.rt.junit.JUnit4TestRunner.prepareStreamsAndStart(JUnit4TestRunner.java:230)
at com.intellij.rt.junit.JUnit4TestRunner.main(JUnit4TestRunner.java:58)

Process finished with exit code 255

Figure 16 Shows the test can fail



Test Name	Duration	Path
TestGame (HangmanGame)	106 ms	/Library/Java/JavaVirtualMachines/jdk-13.0.1.jdk/Contents/H
testUnderscoresToString	30 ms	/Users/mummy/Desktop/ah224uq_1dv600/Hangman/src/test.txt
testRightGuess	20 ms	/Users/mummy/Desktop/ah224uq_1dv600/Hangman/src/test.txt
testGameSucceededTrue	2 ms	/Users/mummy/Desktop/ah224uq_1dv600/Hangman/src/test.txt
testCheckWordInvalidWord	3 ms	/Users/mummy/Desktop/ah224uq_1dv600/Hangman/src/test.txt
testGameLostfalse	8 ms	/Users/mummy/Desktop/ah224uq_1dv600/Hangman/src/test.txt
testSetWordInvalidWord	9 ms	/Users/mummy/Desktop/ah224uq_1dv600/Hangman/src/test.txt
testGameLostTrue	4 ms	/Users/mummy/Desktop/ah224uq_1dv600/Hangman/src/test.txt
testUnderScoresToStringWithDash	3 ms	/Users/mummy/Desktop/ah224uq_1dv600/Hangman/src/test.txt
testCheckWordValidWord	6 ms	/Users/mummy/Desktop/ah224uq_1dv600/Hangman/src/test.txt

Figure 17 Shows updated figure for all passed test

Comments

The focus here was to show that a test can pass or fail. The code that was not tested contained mostly input from a user or output to the console. Which is hard to test with JUnit, hopefully the manual testing has covered it. Since Game.java depend on other classes a lot of code from other classes was also run

TC2 Test GuessedLetters.java

Each testcase in TC2 test a method from the class GuessedLetters. Before each sub- testcase in TC2 a new instance of GuessedLetters is created.

```
public class TestGuessedLetters {  
  
    GuessedLetters sut;  
  
    @BeforeEach  
    public void setup() { sut = new GuessedLetters(); }  
}
```

TC2.1 toString

The toString method returns a String of all the letters that is currently guessed.

```

@Test
public void testToString() {
    sut.addLetter('a');
    String expected = "a ";

    String actual = sut.toString();

    assertEquals(expected, actual);

    sut.addLetter('b');
    sut.addLetter('x');
    expected = "a b x ";

    actual = sut.toString();

    assertEquals(expected, actual);
}

```

Figure 18 Testing that the format of the String is correct

TC2.2 addLetter

This method should add the parameter letter if the letter is not already added.

```

@Test
public void testAddLetter() {
    sut.addLetter('a');
    sut.addLetter('b');
    String expected = "a b ";

    String actual = sut.toString();

    assertEquals(expected, actual);
}

```

Figure 19 Testing that adding two different letters works correctly


```

@Test
public void testAddLetterLetterAlreadyAdded() {
    sut.addLetter('a');
    sut.addLetter('a');
    sut.addLetter('c');
    sut.addLetter('c');
    String expected = "a c ";

    String actual = sut.toString();

    assertEquals(expected, actual);
}

```

Figure 20 Testing that adding the same letter twice should not result in doublets

TC2.3 LetterAlreadyGuessed

This method should determine if the parameter letter is already a guessed letter.

```

@Test
public void testLetterAlreadyGuessedFalse() {
    boolean expected = false;
    sut.addLetter('s');
    sut.addLetter('q');

    boolean actual1 = sut.letterAlreadyGuessed('t');
    boolean actual2 = sut.letterAlreadyGuessed('a');
    boolean actual3 = sut.letterAlreadyGuessed('r');
    boolean actual4 = sut.letterAlreadyGuessed('x');
    boolean actual5 = sut.letterAlreadyGuessed('o');

    assertEquals(expected, actual1);
    assertEquals(expected, actual2);
    assertEquals(expected, actual3);
    assertEquals(expected, actual4);
    assertEquals(expected, actual5);
}

```

Figure 21 Testing that not yet guessed letters make the method return false

```

@Test
public void testLetterAlreadyGuessedTrue() {
    sut.addLetter('a');
    sut.addLetter('x');
    sut.addLetter('o');
    boolean expected = true;

    boolean actual1 = sut.letterAlreadyGuessed('a');
    boolean actual2 = sut.letterAlreadyGuessed('o');
    boolean actual3 = sut.letterAlreadyGuessed('x');

    assertEquals(expected, actual1);
    assertEquals(expected, actual2);
    assertEquals(expected, actual3);
}

```

Figure 22 Testing that already guessed letters make the method return true

Test report for TC2

Main × TestGuessedLetters ×	
✓ Tests passed: 5 of 5 tests – 33 ms	
✓ Test Results	33 ms
✓ TestGuessedLetters	33 ms
✓ testToString()	25 ms
✓ testLetterAlreadyGuessedTrue()	2 ms
✓ testAddLetter()	1 ms
✓ testLetterAlreadyGuessedFalse()	1 ms
✓ testAddLetterLetterAlreadyAdded()	5 ms

Figure 23 All test passed

Comments

The main focus was to test `GuessedLetters.java`. All the code in `GuessedLetters` was executed and 0% of other classes, that is because `GuessedLetters` do not depend on any other classes.

TC3 Test Words.java

Each testcase in TC3 test a method from the class `Words`.

TC3.1 getRandomWord

This method returns a random word from a text-file. During testing this list only consist of one word which is "Test".

```

@Test
public void testGetRandomWord() {
    String expected = "Test";

    String actual = sut.getRandomWord();

    assertEquals(expected, actual);
}

```

Figure 24 Testing that the word returned is "Test"

TC3.2 indexOfWord

This method should return the index of the word in the noun-list if present, otherwise it should return -1.

```

@Test
public void testIndexOfWordWhenWordPresent() {
    int expected = 0;

    int actual = sut.indexOfWord("Test");

    assertEquals(expected, actual);
}

```

Figure 25 Test when word is present in list

```

@Test
public void testIndexOfWordWhenWordNotPresent() {
    int expected = -1;

    int actual = sut.indexOfWord("NotPresentWord");

    assertEquals(expected, actual);
}

```

Figure 26 Test when word is not present in list

TC3.3 removeWord, addWord

removeWord should remove a word from the noun-list and it should also be removed after turning the game off. AddWord is simply used to return the word to the list for other tests to work.

```

@Test
public void testRemoveWord() {
    int expected = -1;
    sut.removeWord("Test");

    int actual = sut.indexOfWord("Test");

    assertEquals(expected, actual);

    sut = new Words();
    actual = sut.indexOfWord("Test");
    assertEquals(expected, actual);

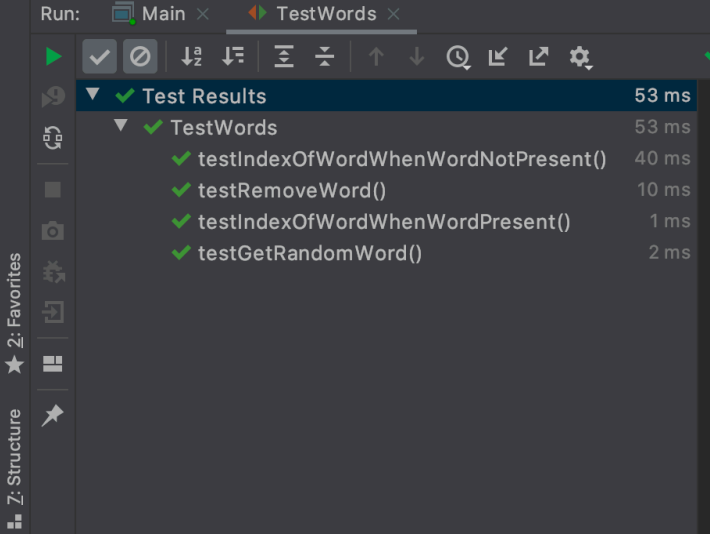
    sut.addWord("Test");
    expected = 0;
    actual = sut.indexOfWord("Test");
    assertEquals(expected, actual);

    sut = new Words();
    actual = sut.indexOfWord("Test");
    assertEquals(expected, actual);
}

```

Figure 27 Testing of removeWord

Test report for TC3



Test Results	Time
TestWords	53 ms
testIndexOfWordWhenWordNotPresent()	40 ms
testRemoveWord()	10 ms
testIndexOfWordWhenWordPresent()	1 ms
testGetRandomWord()	2 ms

Figure 28 The tests succeeded

Comments

List should probably be tested with a list of different words as well. 84,5% of the class to be tested was executed.

TC4 Test MultitplayerGame.java

Each testcase in TC4 test a method from the class MultiplayerGame.

TC4.1 checkWord

This method should check that the parameter word only contains letters and dashes.

```
@Test
public void testCheckWordValidWord() {
    MultiplayerGame sut = new MultiplayerGame();
    boolean expected = true;

    boolean actual = sut.checkWord("Example");

    assertEquals(expected, actual);
}
```

Figure 29 Testing a word which only consist of letters

```
@Test
public void testCheckWordValidWordWithDash() {
    MultiplayerGame sut = new MultiplayerGame();
    boolean expected = true;

    boolean actual = sut.checkWord("Grand-parent");

    assertEquals(expected, actual);
}
```

Figure 30 Testing a word which consist of letters and a dash

```
@Test
public void testCheckWordInvalidWord() {
    MultiplayerGame sut = new MultiplayerGame();
    boolean expected = false;

    boolean actual = sut.checkWord("*,*");

    assertEquals(expected, actual);
}
```

Figure 31 Testing an invalid word

Test report for TC4

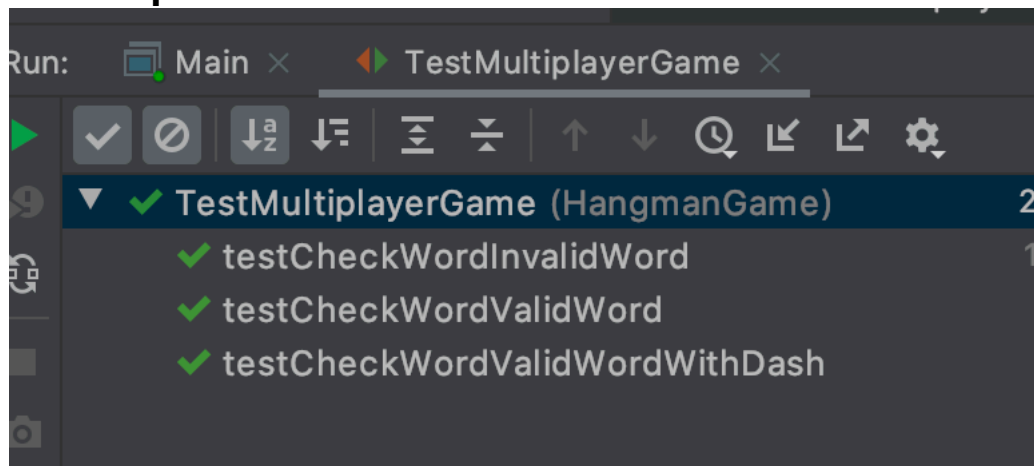


Figure 32 No tests failed

Comments

A fairly low percentage of MultiplayerGame was executed (27,2%). Most of the code in this class involved input from a player and output to the console which is hard to test using JUnit.

Automated test report

The test cases exercise about 60% of the code in the implementation. Big parts of the game involve interaction between the user and the system which is hard to simulate in automated testing and must be tested using manual testing. Two tests in the test cases failed but the cause was detected and fairly simple to correct.

Reflections

This assignment has helped expand my knowledge in java. My knowledge in the concept of inheritance has been tested and I have learnt new ways in developing code. Though the assignment was challenging in the beginning it has been an exciting way to learn testing. I have been very careful with testing my program, therefore there has been few issues detected during the process. Perhaps the input of an independent tester could have pointed out something I must have missed. My inability to make an overall automated test is because there is a lot of interaction with the player, which is generally difficult to test. Overall, this assignment has enhanced my knowledge in programming and was quite interesting.

Time report

Task	Estimated Time	Actual Time
Manual TC	2hr	3hr
Unit TC	2hr	5hr
Running automated Test	1hr	10 min
Code Inspection	2hr	1hr
Test Report	3hr	6hr