Johan Dahlberg
jd222qd@student.lnu.se
https://github.com/jd222qd/jd222qd_dv600

The Hangman Application

Vision

The goal of this project is to implement the game “Hangman” in a text based fashion, using Java as the coding language. The basics of this game is that a random word will be generated from a list of words, which the player then has to guess by suggesting one letter at a time. The words will be imported from a text file.
The amount of guesses (parts used to hang the man) a player has will depend on the difficulty that they choose. There will be 3 difficulties, Easy, Medium & Hard.
Easy - 13 guesses/parts (ground, vertical pole, horizontal pole, rope, head, body, left arm, right arm, right leg, left leg, eyes, nose, mouth)
Medium - 10 guesses/parts (ground, vertical pole, horizontal pole, rope, head, body, left arm, right arm, right leg, left leg)
Hard - 7 guesses/parts (rope, head, body, left arm, right arm, right leg, left leg), the structure (ground, vertical pole, horizontal pole) will already be built on this difficulty.

When starting the program, the player will be greeted by a ‘Main Menu’, with the options to:

1) Start Game
2) Check High scores
3) Quit Game

Pressing ‘1’ will take the player to a new ‘Choose difficulty’ menu where they can choose difficulty and start the game, or choose to go back to the main menu:

1) Easy
2) Medium
3) Hard
4) Back to Main Menu

Pressing ‘2’ in the main menu will bring up a list of high scores with the “player name” and their score. The top 10 best scores will be shown.

Pressing ‘3’ will quit the program.

Once a game has been started, a random word will be chosen and will be matched by the same amount of underscores. For example:
Dog → _ _ _

For each incorrect guess the player makes, an image will appear, with the use of Unicode, which represents the hanging of the man. The letter used to guess will also show up on the

screen under “letters used” so the player can easily keep track of the guessed letters. If the player runs out of guesses, they lose gets redirected back to the main menu.
If the player manages to guess the word, they have beat the game, and will be given a score. They then have the option to save their score using ‘save score’ or ‘go back to main menu’. Choosing to save the score will make the player enter a “player name”. If the score is good enough to make it into the top 10, they can view it the main menu by choosing ‘Check Highscores’. If the player chooses not to save their score, it will not be added to the high score list, even if it would’ve been in the top 10.

Use cases

This application contains 6 use-cases which are described in more detail below.

UC 1 Start Application

Precondition: None.

Postcondition: The application is launched.

Main scenario

1. The application starts.
2. The system presents the main menu with the option to play, check high scores, and quit the application.
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

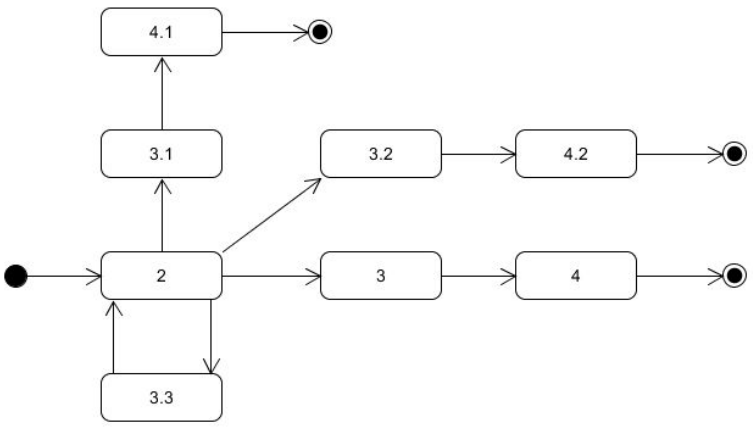
3.1 The Gamer makes the choice to quit the application.

4.1 The system quits the game (see Use Case 3)

3.2 The Gamer makes the choice to check high scores.

4.2 The system displays the high scores (see Use Case 4)

3.3 The Gamer enters an invalid input. The system presents an error message. Goto step 2 in Main Scenario.



UC 2 Launch Game

Precondition: None.

Postcondition: A game of hangman is initiated.

Main scenario

1. Goto Use Case 5.

Alternative scenarios

None.

UC 3 Quit Application

Precondition: The application is running.

Postcondition: The application is terminated.

Main scenario

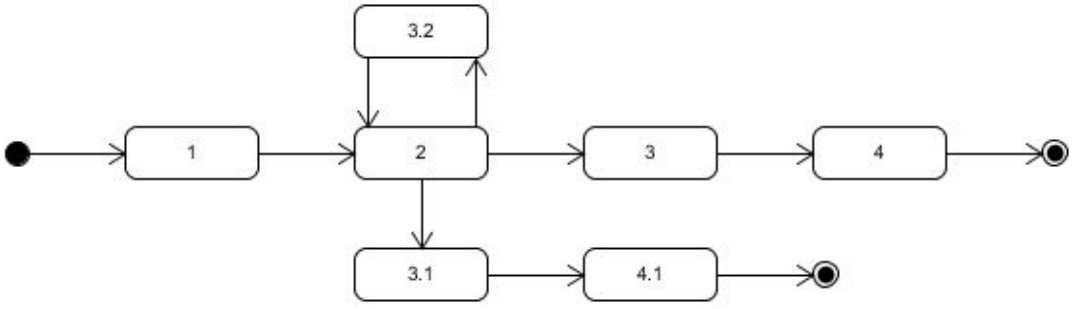
1. Starts when the Gamer chooses to quit the application.
2. The system prompts for confirmation.
3. The Gamer confirms.
4. The system terminates.

Alternative scenarios

3.1. The Gamer does not confirm

4.1 The system returns to its previous state (UC 1 / UC 6)

3.2 The Gamer enters an invalid input. Goto step 2 in Main Scenario.



UC 4 Check high scores

Precondition: None.

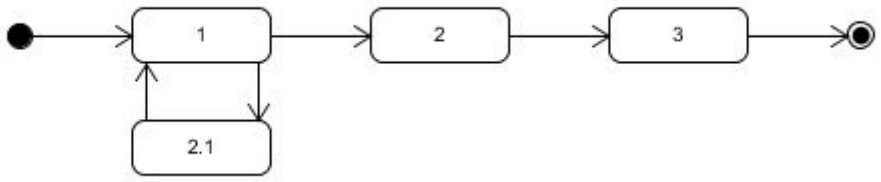
Postcondition: The high scores are displayed.

Main scenario

1. The system prints the top 10 high scores and presents the option to return to Main Menu.
2. The Gamer chooses to go back.
3. Goto Use Case 1, step 2.

Alternative scenarios

2.1 The Gamer enters an invalid input. Goto step 1 in Main Scenario.



UC 5 Choose Difficulty

Precondition: None.

Postcondition: A difficulty is chosen for the game.

Main scenario

1. The system displays the difficulty options (Easy, Medium, Hard) and the option to return to Main Menu.
2. The Gamer chooses the ‘Easy’ difficulty.
3. The system launches the game on ‘Easy’ difficulty.
4. Goto Use Case 6.

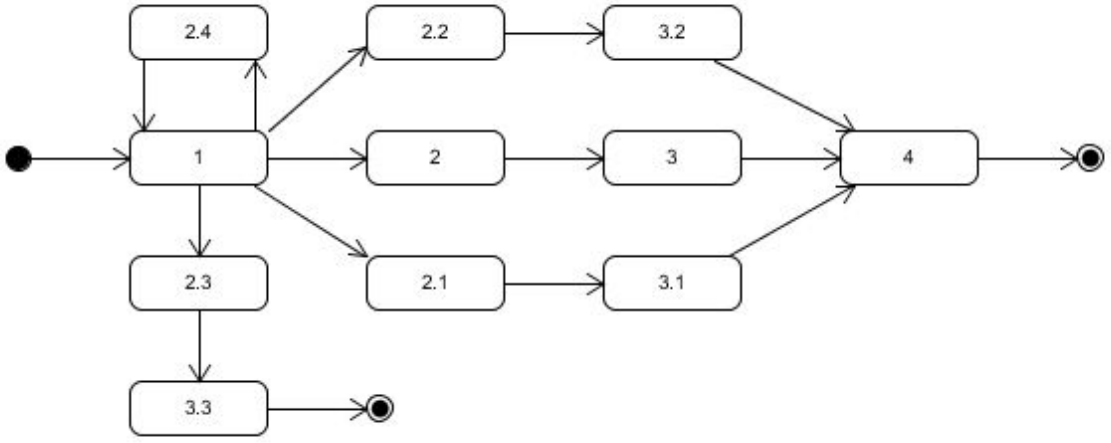
Alternative scenarios

2.1 The Gamer chooses ‘Medium’ difficulty.
3.1 The system launches the game on ‘Medium’ difficulty.
4 Goto Use Case 6.

2.2 The Gamer chooses ‘Hard’ difficulty.
3.2 The system launches the game on ‘Hard’ difficulty.
4 Goto Use Case 6.

2.3 The Gamer chooses to Return to Main Menu.
3.3 Goto Use Case 1, step 2.

2.4 The Gamer enters an invalid input. Goto step 1 in Main Scenario.



UC 6 Guess Letter

Precondition: A game of hangman is currently running.

Postcondition:

Main scenario

1. The system randomly picks a word (based on difficulty) from a pre-defined list and presents the word, using underscores.
2. The system displays the option to guess a letter, go back to Main Menu, or Quit the application.
3. The Gamer makes a correct guess.
4. The system displays a ‘correct’ message and displays the position(s) of the guessed letter, within the word.Goto step 2.

Alternative scenarios

3.1 The Gamer makes an incorrect guess.

4.1 The system displays a ‘incorrect’ message and displays the hangman figure with an additional part. Goto step 2 in Main Scenario.

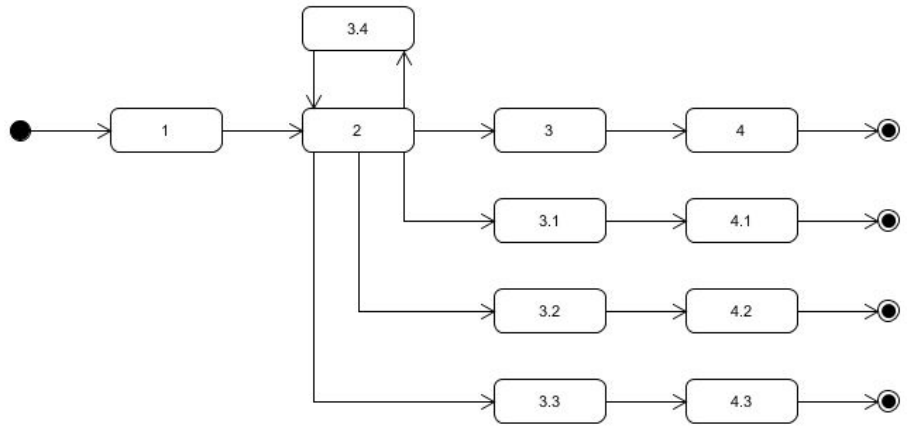
3.2 The Gamer chooses to go back to Main Menu.

4.2 Goto Use Case 1, step 2.

3.3 The Gamer chooses to Quit the application.

4.3 Goto Use Case 3.

3.4 The Gamer enters an invalid input. Goto step 2 in Main Scenario.



Test Plan

Objective

The objective is to test all the current implemented code from the previous iterations.

What to test and how

I intend to test all Use Cases (UC1- UC6) by running dynamic manual test-cases. I will also write automated unit tests for all methods in the HangManLogic.java class & the ReadWords.java class. Each system under testing - method will have at least two tests.

Time plan

Task	Estimated	Actual
Manual test-cases	3h	4h
Running manual test cases	1h	1h
Unit tests	4h	5h
Test report	1h	1h

Manual Test-Cases

TC1.1 Game start successful

Use case: UC1 Start Application

Scenario: Game start successful

The main scenario of UC1 is tested where the gamer chooses to start a game.

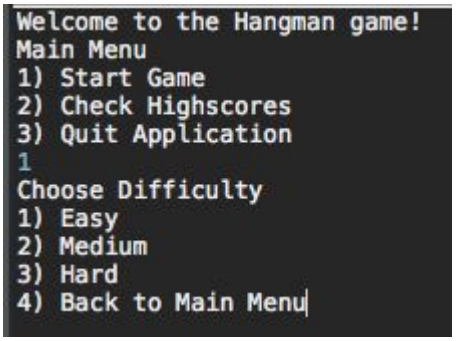
Precondition: None

Test steps

- Start the application
- The system presents the main menu with the option to start game, check high scores, and quit the application.
- Enter the key “1” and press enter.
- The system “starts” the game (see Use Case 2).

Expected

- The system should “start a game” by bringing the user to the difficulty menu (see Use Case 2).



TC1.2 Quit application confirmation successful

Use case: UC1 Start Application

Scenario: Quit application confirmation successful

Alternative scenario of UC1 is tested where the gamer chooses to quit the application.

Precondition: None

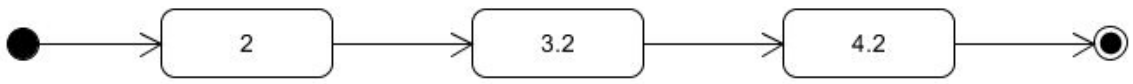
Test steps

- Start the application
- The system presents the main menu with the option to start game, check high scores, and quit the application.
- Enter the key “3” and press enter.
- The system prompts the user to confirm termination (see Use Case 3).

Expected

- The system should prompt the user to confirm termination by bringing the user to the next menu (see Use Case 3).

```
Welcome to the Hangman game!
Main Menu
1) Start Game
2) Check Highscores
3) Quit Application
3
Are you sure you want to quit the application?
1) Yes
2) No
```



TC1.3 Check highscores successful

Use case: UC1 Start Application

Scenario: Check highscores successful

Alternative scenario of UC1 is tested where the gamer chooses to check the highscores.

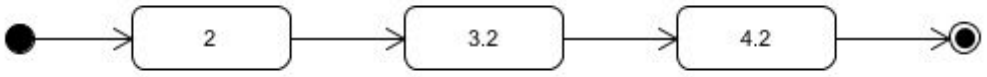
Precondition: None

Test steps

- Start the application
- The system presents the main menu with the option to start game, check high scores, and quit the application.
- Enter the key “2” and press enter.
- The system starts the game (see Use Case 4).

Expected

- The system should print the highscores (see Use Case 4).



TC1.4 Return on invalid input successful

Use case: UC1 Start Application

Scenario: Return on invalid input successful

Alternative scenario of UC1 is tested where the gamer enters an invalid input and gets redirected back to step 2.

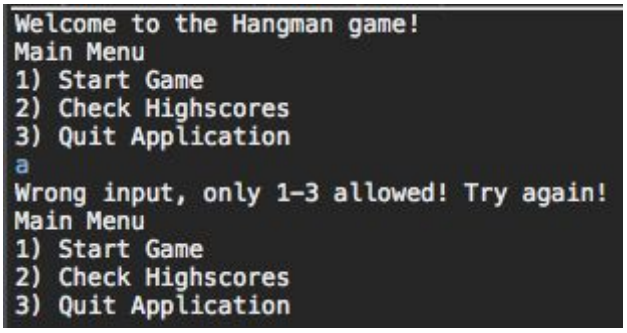
Precondition: None

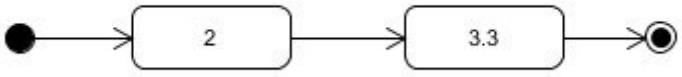
Test steps

- Start the application
- The system presents the main menu with the option to start game, check high scores, and quit the application.
- Enter the “a” key and press enter.
- The system prints an error message and redirects the gamer back to step 2.

Expected

- The system should print an error message and redirect the gamer back to step 2.





TC2.1 Game launch successful

Use case: UC2 Launch Game

Scenario: Game launch successful

The main scenario of UC2 is tested where a game is launched.

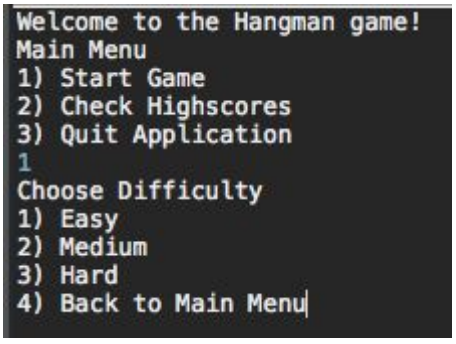
Precondition: The gamer chose to start a game in UC1.

Test steps

- None.

Expected

- The system should launch a game by displaying the difficulty menu (see Use Case 5).



TC3.1 Application termination confirmed successful

Use case: UC3 Quit Application

Scenario: Application termination confirmed successful

Main scenario of UC3 is tested where the gamer chooses to confirm the termination of the application.

Precondition: UC1 / UC6 was successful.

Test steps

- The system prompts the gamer for confirmation regarding the termination.
- Enter the key “1” and press enter.
- The application terminates.

Expected

- The system should terminate successfully.

TC3.2 Return to previous state successful

Use case: UC3 Quit Application

Scenario: Return to previous state successful

Alternative scenario of UC3 is tested where the gamer chooses to not confirm the termination of the application.

Precondition: The application is running & the gamer chose to quit the application.

Test steps

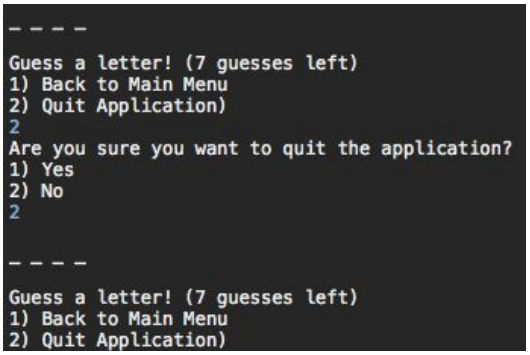
- The system prompts the user for confirmation regarding the termination.
- Enter the key “2” and press enter.
- The system returns to its previous state (Main Menu (UC1) / current game (UC6)).

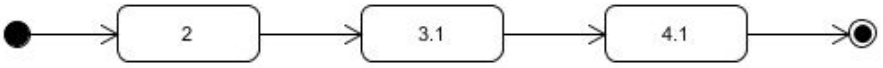
Expected

- The system should return to its previous state successfully.

Previous state = Main Menu (UC1).

Previous state = current game (UC6).




2
3.1
4.1

TC3.3 Return on invalid input successful

Use case: UC3 Quit Application

Scenario: Return on invalid input successful

Alternative scenario of UC3 is tested where the gamer enters an invalid input and gets redirected back to step 2.

Precondition: The application is running & the gamer chose to quit the application.

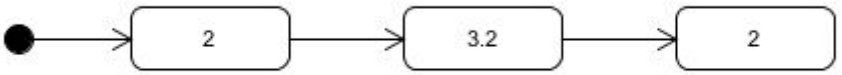
Test steps

- The system prompts the user for confirmation regarding the termination.
- Enter the key “a” and press enter.
- The system prints an error message and redirects the gamer back to step 2.

Expected

- The system should print an error message and redirect the gamer back to step 2.

```
Welcome to the Hangman game!
Main Menu
1) Start Game
2) Check Highscores
3) Quit Application
3
Are you sure you want to quit the application?
1) Yes
2) No
a
Wrong input, only 1-2 allowed! Try again!
Are you sure you want to quit the application?
1) Yes
2) No
```



TC4.1 Highscores printed successful

Use case: UC4 Check highscores

Scenario: Highscores printed successful

Main scenario of UC4 is tested where the gamer chooses to check the highscores and return to Main Menu.

Precondition: The gamer chose to check the highscores.

Test steps

- The system prints the highscores and gives the option to return to Main Menu.
- Enter the key “1” and press enter.
- The application returns to the Main Menu. (see Use Case 1)

Expected

- The system should print the highscores and return to the Main Menu (see Use Case 1)

TC4.2 Return on invalid input successful

Use case: UC4 Check highscores

Scenario: Return on invalid input successful

Alternative scenario of UC4 is tested where the gamer enters an invalid input and gets redirected back to step 1.

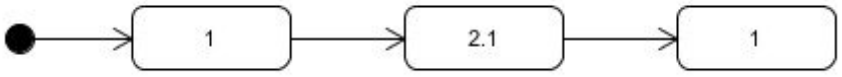
Precondition: The application is running & the gamer chose to quit the application.

Test steps

- The system prompts the user for confirmation regarding the termination.
- Enter the key “a” and press enter.
- The system prints an error message and redirects the gamer back to step 1.

Expected

- The system should print an error message and redirect the gamer back to step 1.



TC5.1 A round of hangman launched on “easy” difficulty successful

Use case: UC5 Choose Difficulty

Scenario: A round of hangman launched on “easy” difficulty successful

Main scenario of UC5 is tested where the gamer chooses to start a round of the hangman game on “easy” difficulty.

Precondition: The gamer successfully launched a game (UC2).

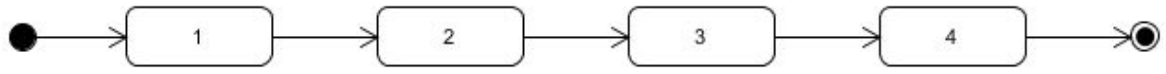
Test steps

- The system presents the difficulty menu
- Enter the key “1” and press enter.
- The system starts a round of the hangman game on “easy” difficulty. (see Use Case 6)

Expected

- The system should start a round of the hangman game on “easy” difficulty. (see Use Case 6)

```
Choose Difficulty
1) Easy
2) Medium
3) Hard
4) Back to Main Menu
1

_ _ _ _

Guess a letter! (13 guesses left)
1) Back to Main Menu
2) Quit Application)
```



TC5.2 A round of hangman launched on “medium” difficulty successful

Use case: UC5 Choose Difficulty

Scenario: A round of hangman launched on “medium” difficulty successful

Alternative scenario of UC5 is tested where the gamer chooses to start a round of the hangman game on “medium” difficulty.

Precondition: The gamer successfully launched a game (UC2).

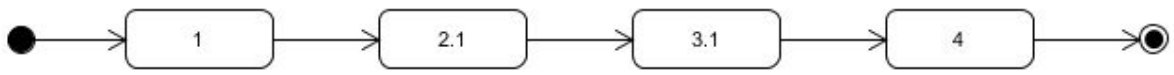
Test steps

- The system presents the difficulty menu
- Enter the key “2” and press enter.
- The system starts a round of the hangman game on “medium” difficulty. (see Use Case 6)

Expected

- The system should start a round of the hangman game on “medium” difficulty. (see Use Case 6)

```
Choose Difficulty
1) Easy
2) Medium
3) Hard
4) Back to Main Menu
2

_ _ _ _

Guess a letter! (10 guesses left)
1) Back to Main Menu
2) Quit Application)
```



TC5.3 A round of hangman launched on “hard” difficulty successful

Use case: UC5 Choose Difficulty

Scenario: A round of hangman launched on “hard” difficulty successful

Alternative scenario of UC5 is tested where the gamer chooses to start a round of the hangman game on “hard” difficulty.

Precondition: The gamer successfully launched a game (UC2).

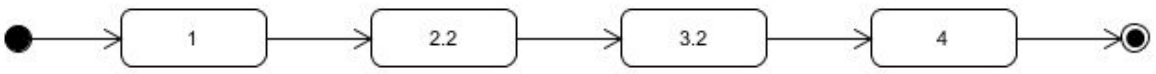
Test steps

- The system presents the difficulty menu
- Enter the key “3” and press enter.
- The system starts a round of the hangman game on “hard” difficulty. (see Use Case 6)

Expected

- The system should start a round of the hangman game on “hard” difficulty. (see Use Case 6)

```
Choose Difficulty
1) Easy
2) Medium
3) Hard
4) Back to Main Menu
3

_ _ _ _

Guess a letter! (7 guesses left)
1) Back to Main Menu
2) Quit Application)
```



TC5.4 Return to Main Menu successful

Use case: UC5 Choose Difficulty

Scenario: Return to Main Menu successful

Alternative scenario of UC5 is tested where the gamer chooses to return to the main menu (UC 1).

Precondition: The gamer successfully launched a game (UC2).

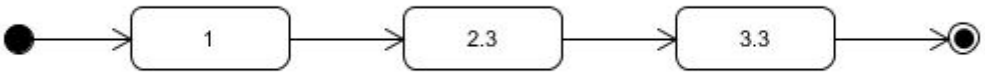
Test steps

- The system presents the difficulty menu
- Enter the key “4” and press enter.
- The system returns to the Main Menu (see Use Case 1).

Expected

- The system should return to the Main Menu (see Use Case 1).

```
Choose Difficulty
1) Easy
2) Medium
3) Hard
4) Back to Main Menu
4
Main Menu
1) Start Game
2) Check Highscores
3) Quit Application
```



TC5.5 Return on invalid input successful

Use case: UC5 Choose Difficulty

Scenario: Return on invalid input successful

Alternative scenario of UC5 is tested where the gamer enters an invalid input and gets redirected back to step 1.

Precondition: The gamer successfully launched a game (UC2).

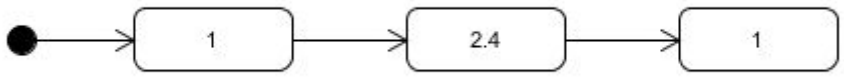
Test steps

- The system displays the option to guess a letter, go back to Main Menu, or Quit the application.
- Enter the key “a” and press enter.
- The system prints an error message and redirects the gamer back to step 1.

Expected

- The system should print an error message and redirect the gamer back to step 1.

```
Choose Difficulty
1) Easy
2) Medium
3) Hard
4) Back to Main Menu
a
Wrong input, only 1-4 allowed! Try again!
Choose Difficulty
1) Easy
2) Medium
3) Hard
4) Back to Main Menu
```



TC6.1 Correct letter guess successful

Use case: UC6 Guess Letter

Scenario: Correct letter guess successful

Main scenario of UC6 is tested where the gamer enters a correct letter and the system displays the position(s) of the guessed letter, within the word.

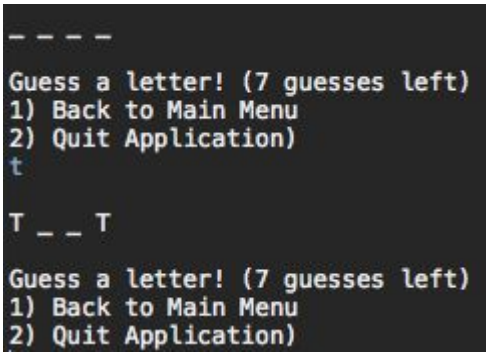
Precondition: The gamer successfully chose a difficulty (UC5).

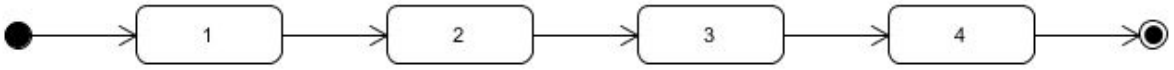
Test steps

- The system displays the option to guess a letter, go back to Main Menu, or Quit the application.
- Enter the key “t” and press enter.
- The system displays the position(s) of the guessed letter, within the word.

Expected

- The system should display the position(s) of the guessed letter, within the word.





TC6.2 Incorrect letter guess successful

Use case: UC6 Guess Letter

Scenario: Incorrect letter guess successful

Alternative scenario of UC6 is tested where the gamer enters an incorrect letter and the system displays the hangman figure with an additional part.

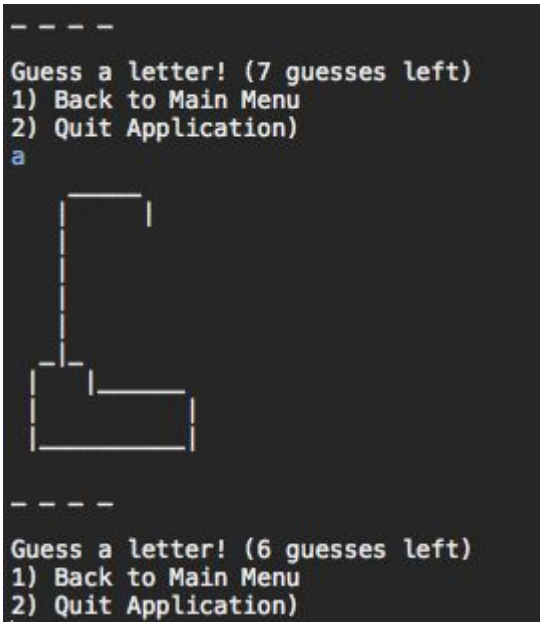
Precondition: The gamer successfully chose a difficulty (UC5).

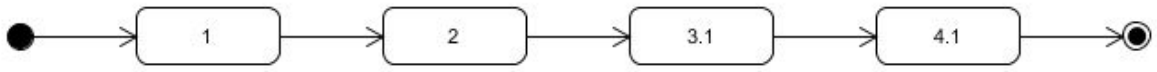
Test steps

- The system displays the option to guess a letter, go back to Main Menu, or Quit the application.
- Enter the key “a” and press enter.
- The system displays the hangman figure with an additional part.

Expected

- The system should display the hangman figure with an additional part.





TC6.3 Return to Main Menu successful

Use case: UC6 Guess Letter

Scenario: Return to Main Menu successful

Alternative scenario of UC6 is tested where the gamer chooses to return to the Main Menu (UC 1).

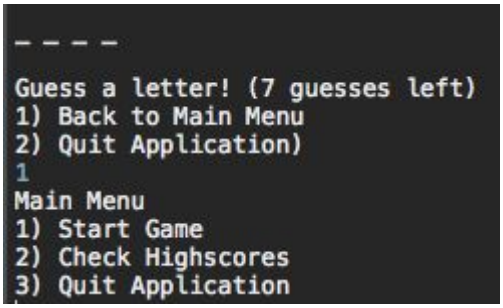
Precondition: The gamer successfully chose a difficulty (UC5).

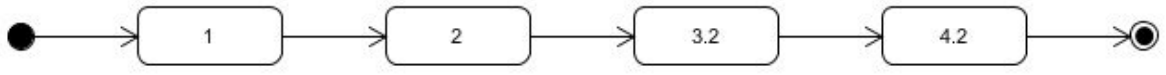
Test steps

- The system displays the option to guess a letter, go back to Main Menu, or Quit the application.
- Enter the key “1” and press enter.
- The system returns the gamer to the Main Menu. (see Use Case 1).

Expected

- The system should return the gamer to the Main Menu.(see Use Case 1).





TC6.4 Quit application successful

Use case: UC6 Guess Letter

Scenario: Quit application successful

Alternative scenario of UC6 is tested where the gamer chooses to quit the application (UC 3).

Precondition: The gamer successfully chose a difficulty (UC5).

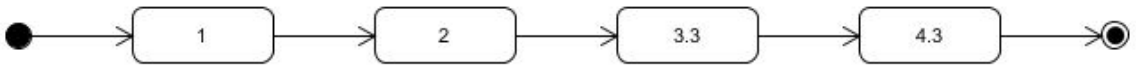
Test steps

- The system displays the option to guess a letter, go back to Main Menu, or Quit the application.
- Enter the key “2” and press enter.
- The system displays the confirm termination menu. (see Use Case 3)

Expected

- The system should display the confirm termination menu. (see Use Case 3)

```
_ _ _ _

Guess a letter! (7 guesses left)
1) Back to Main Menu
2) Quit Application)
2
Are you sure you want to quit the application?
1) Yes
2) No
```



TC6.5 Return on invalid input successful

Use case: UC6 Guess Letter

Scenario: Return on invalid input successful

Alternative scenario of UC6 is tested where the gamer enters an invalid input and gets redirected back to step 2.

Precondition: The gamer successfully chose a difficulty (UC5).

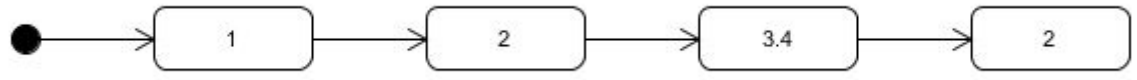
Test steps

- The system displays the option to guess a letter, go back to Main Menu, or Quit the application.
- Enter the key “3” and press enter.
- The system prints an error message and redirects the gamer back to step 2.

Expected

- The system prints an error message and redirects the gamer back to step 2.

```
- - - -

Guess a letter! (7 guesses left)
1) Back to Main Menu
2) Quit Application)
3
Wrong input, only a-z & 1-2 allowed! Try again!

- - - -

Guess a letter! (7 guesses left)
1) Back to Main Menu
2) Quit Application)
```



Test Report

Manual tests

Manual test traceability matrix and sucess:

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
TC1.3	1/ FAIL	0	0	0	0	0
TC1.4	1/OK	0	0	0	0	0
TC2.1	0	1/OK	0	0	0	0
TC3.1	0	0	1/OK	0	0	0
TC3.2	0	0	1/OK	0	0	0
TC3.3	0	0	1/OK	0	0	0
TC4.1	0	0	0	1/ FAIL	0	0
TC4.2	0	0	0	1/ FAIL	0	0
TC5.1	0	0	0	0	1/OK	0
TC5.2	0	0	0	0	1/OK	0
TC5.3	0	0	0	0	1/OK	0
TC5.4	0	0	0	0	1/OK	0

TC5.5	**0**	**0**	**0**	**0**	**1/OK**	**0**
TC6.1	**0**	**0**	**0**	**0**	**0**	**1/OK**
TC6.2	**0**	**0**	**0**	**0**	**0**	**1/OK**
TC6.3	**0**	**0**	**0**	**0**	**0**	**1/OK**
TC6.4	**0**	**0**	**0**	**0**	**0**	**1/OK**
TC6.5	**0**	**0**	**0**	**0**	**0**	**1/OK**
Coverage & Success	**3/4 OK**	**1/1 OK**	**3/3 OK**	**0/2 OK**	**5/5 OK**	**5/5 OK**

Comments

The random word was set to always be “TEST” during these test cases, to keep consistency.

Everything related to Use Case 4 fails because those features have not yet been implemented. This includes TC1.3, TC4.1, TC4.2.

Automated Unit Tests

ATC 1 - DrawHangmanTest (DrawHangman.java)

```java
class DrawHangmanTest {

private static int test_count = 0;

    DrawHangman test;

    /* Is executed before every test method (not test case).*/
    @BeforeEach
    public void setUp() {
        test_count++;
        System.out.println("Test "+ test_count);
    }


    @Test
    void shouldDrawAllHangmanStagesOnEasy() {

        DrawHangman sut = new DrawHangman();
        for (int i = 0; i < 12; i++) {
        assertEquals(sut.printHangMan("Easy", i), i);
        assertTrue(sut.printHangMan("Easy", i) < i+1);
        }
    }
    @Test
    void shouldDrawAllHangmanStagesOnMedium() {
        DrawHangman sut = new DrawHangman();
        for (int i = 0; i < 9; i++) {
        assertEquals(sut.printHangMan("Medium", i), i);
        assertTrue(sut.printHangMan("Medium", i) < i+1);
        }
    }

    @Test
    void shouldDrawAllHangmanStagesOnHard() {
        DrawHangman sut = new DrawHangman();
        for (int i = 0; i < 6; i++) {
        assertEquals(sut.printHangMan("Hard", i), i);
        assertTrue(sut.printHangMan("Hard", i) < i+1);
        }
    }

}

```

ATC 2 - HangmanGameTest (HangmanGame.java)

```
10  class HangmanGameTest {
11
12  private static int test_count = 0;
13
14      /* Is executed before every test method (not test case).*/
15      @BeforeEach
16      public void setUp() {
17          test_count++;
18          System.out.println("Test "+ test_count);
19      }
20
21      @Test
22      void shouldReturnZeroIfGameHasNotBeenLaunched() {
23          HangmanGame sut = new HangmanGame();
24          assertEquals(0, sut.guessesLeft());
25      }
26
27
28
29  }
```

ATC 3 - HangmanMenusTest (HangmanMenus.java)

```
12  class HangmanMenusTest {
13
14      private static int test_count = 0;
15
16      /* Is executed before every test method (not test case).*/
17      @BeforeEach
18      public void setUp() {
19          test_count++;
20          System.out.println("Test "+ test_count);
21      }
22
23      @Test
24      void test() throws InterruptedException{
25          HangmanMenus sut = new HangmanMenus();
26          System.setIn(new ByteArrayInputStream("1".getBytes()));
27          assertTrue(sut.confirmTermination());
28      }
29  }
```

ATC 4 - HighScoresTest (HighScores.java)

```
class HighScoresTest {

    @Test
    void shouldNotReturnNull() {

        HighScores sut = new HighScores();
        assertTrue(sut.getHighScores() != null);
    }

}
```

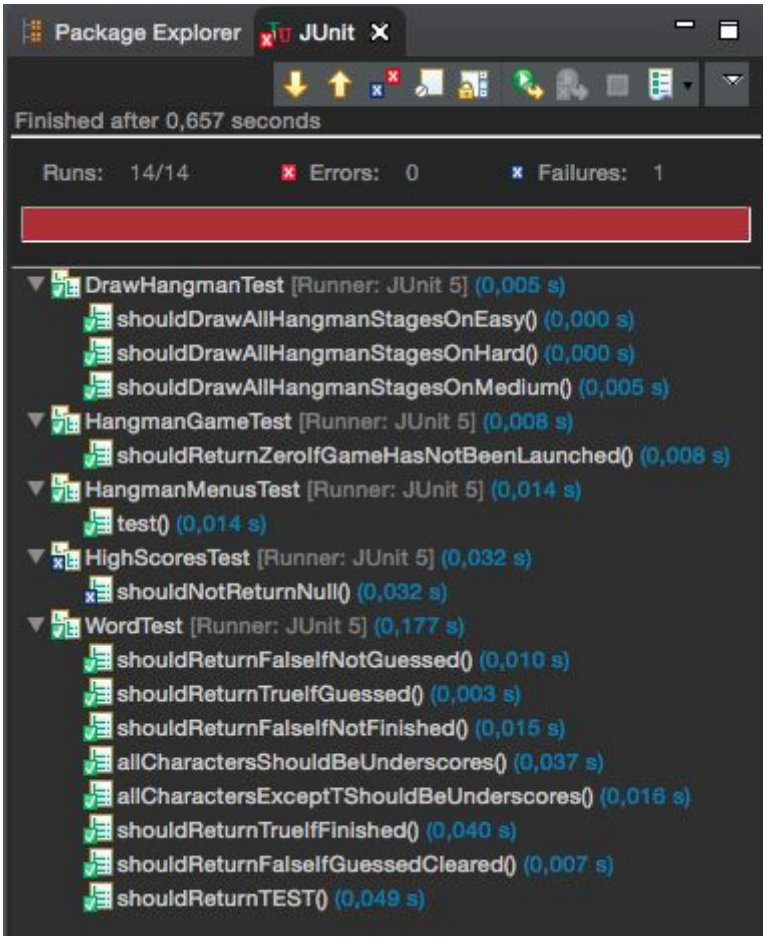
ATC 5 - WordTest (Word.java)

```java
class WordTest {

private static int test_count = 0;

    /* Is executed before every test method (not test case).*/
    @BeforeEach
    public void setUp() {
        test_count++;
        System.out.println("Test "+ test_count);
    }

    @Test
    void shouldReturnFalseIfNotGuessed() {
        Word sut = new Word();
        assertFalse(sut.contains("a"));
    }

    @Test
    void shouldReturnTrueIfGuessed(){
        Word sut = new Word();
        sut.add("a");
        assertTrue(sut.contains("a"));
    }

    @Test
    void shouldReturnFalseIfGuessedCleared(){
        Word sut = new Word();
        sut.add("a");
        sut.clear();
        assertFalse(sut.contains("a"));
    }

    @Test
    void allCharactersShouldBeUnderscores(){ //_ _ _ _
        Word sut = new Word();
        sut.setupWord();
        sut.printUnderscores();
        for (int i = 0; i < sut.getWordLetters().length; i++) {
            String actual = sut.getWordLetters()[i];
            String expected ="_";
            assertEquals(actual, expected);
        }
    }
```

```
53
54      @Test
55      void allCharactersExceptTShouldBeUnderscores(){ //T _ _ T
56          Word sut = new Word();
57          sut.setupWord();
58          sut.updateWordLetters("T");
59              String actual = sut.getWordLetters()[0];
60              String expected = "T";
61              assertEquals(actual, expected);
62              actual = sut.getWordLetters()[1];
63              expected = "_";
64              assertEquals(actual, expected);
65              actual = sut.getWordLetters()[2];
66              expected = "_";
67              assertEquals(actual, expected);
68              actual = sut.getWordLetters()[3];
69              expected = "T";
70              assertEquals(actual, expected);
71      }
72
73      @Test
74      void shouldReturnTrueIfFinished(){ //T E S T
75          Word sut = new Word();
76          sut.setupWord();
77          sut.updateWordLetters("T");
78          sut.updateWordLetters("E");
79          sut.updateWordLetters("S");
80
81          assertTrue(sut.checkIfWordFinished());
82      }
83
84      @Test
85      void shouldReturnFalseIfNotFinished(){ //T E S T
86          Word sut = new Word();
87          sut.setupWord();
88          assertFalse(sut.checkIfWordFinished());
89      }
90
91      @Test
92      void shouldReturnTEST(){ //T E S T
93          Word sut = new Word();
94          sut.setupWord();
95          assertEquals(sut.getWord(), "TEST");
96      }
```

Automated unit test coverage and success

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
jd222qd_dv600	63,9 %	796	449	1 245
src	63,9 %	796	449	1 245
hangman_project	50,0 %	442	442	884
HangmanGame.java	7,0 %	18	239	257
HangmanMenus.java	17,5 %	34	160	194
DrawHangman.java	93,3 %	237	17	254
HangmanMain.java	0,0 %	0	10	10
HighScores.java	50,0 %	10	10	20
ReadWords.java	86,0 %	37	6	43
Word.java	100,0 %	106	0	106
Automated_tests	98,1 %	354	7	361
HighScoresTest.java	73,3 %	11	4	15
DrawHangmanTest.java	97,3 %	107	3	110
HangmanGameTest.java	100,0 %	29	0	29
HangmanMenusTest.java	100,0 %	34	0	34
WordTest.java	100,0 %	173	0	173



Comments

I was unable to test some methods due to the way that they were structured. For example void methods which need some of mock to be tested properly. Also, some of the method in HangmanMenus.java automatically calls other methods when given certain inputs. To test these methods, they either need to be refactored or you need to use some sort of mock tool.

The test for the method getHighscores() in the class HighScores.java failed because the method does not have a correct code implementation yet.

All other tests were successful.

Reflection

Writing and performing tests on the code turned out to be more challenging than expected. I had to refactor parts of it to even make it testable in the first place, and some other parts I just wasn't able to refactor with the time I had. The manual test cases were fairly simple to perform because the application worked as expected and with no errors. However, because of the way that it was written, I had trouble performing the automated unit tests. This is something to think about in the future, i.e write code that is easily testable. All in all it was a valuable iteration for me personally, because I learned a lot about how the code should be structured and written for testability.