TESTING

Name: Alva Fandrey
Email: af222ug@student.lnu.se

GitHub-repo: https://github.com/afandrey/af222ug_1dv600

| Contents

1 | Test Plan

1.1 Overview

The purpose of this document is to gather all necessary information to plan and control the test effort during this iteration. The document will describe the approach chosen to test the software and which features will be tested also which features that will not be tested and for what reasons. The goal with the testing for this iteration is to have a somewhat playable version of the Hangman game.

1.2 Features to be tested

1.2.1 Manual Test Cases

During this iteration we have chosen to perform manual testing on all use cases.

* Use Case 1 – Start Game
* Use Case 2 – Choose a level
* Use Case 3 – Play Game
* Use Case 4 – Quit Game
* Use Case 5 – View high score list

1.2.2 Unit Tests

During this iteration we have chosen to perform unit testing on these methods:

* QuitGame()
* generateRandomWord()
* getWordList()

The reason for choosing to test these methods is because their functionality is crucial for having a nice user experience when using the application as well as crucial for having the software working properly.

1.3 Test Approach

During this iteration we will write and run dynamic manual test-cases to test the scenarios described in the chosen use cases. We will also write automated unit tests for the methods setName and checkGuess in the class Game. Each sut-method will have two tests each.

1.4 Time Plan

Task	Estimated Time	Actual Time
Plan for Iteration 4	1h	1h
Implementation – add more/new words with level categories	2h	1h 45min
Implementation – fix: word cannot contain same letter	2.5h	3h
Implementation – dog breeds with “two words”	2h	3h

Update Vision and Project Plan, replaced admin with levels	3h	2h
Update Use Case Diagram	1h	2.5h
Update Use Cases	3h	2h 45min
Update State Machine	3h	3.5h
Update Class Diagram	2h	1.5h
Implementation – add levels	6h	3h
Implementation – add high score functionality	3h	6.5h
Implementation – draw hangman	3h	4h
Update Test Plan	4h	2h
Update/create and execute Manual Test Cases	5h	7h
Update Automated Unit-tests	4h	2h
Implementation – fix appearance in console	1h	2h

2 | Manual Test Cases

Test-Case 1.1: Start game successfully

Use Case: UC1 – Start Game

Scenario: The user gets asked to choose a level for the game.

Preconditions: none.

Test steps:

- Start the application
- The system shows: “Enter your name:”
- Type in the name “Alva” and press enter
- The system presents the text “Welcome, Alva” and the menu options.
- Type the number “1” and press enter

Expected output: The text “Choose a level 1-3 or type in r for a random level:” is shown in the console.

☒Success ☐Fail

Comments:

Test-Case 1.2: Invalid menu choice

Use Case: UC1 – Start Game

Scenario: The user makes an invalid menu choice and the system presents an error message.

Preconditions: none.

Test steps:

- Start the application
- The system shows: “Enter your name:”
- Enter the name “Alva” and press enter
- The system presents the text “Welcome, Alva” and the menu options.
- Type the number “9” and press enter

Expected output: The text “Wrong input” is shown in the console along with the menu options.

☒Success ☐Fail

Comments:

Test-Case 2.1: Choose correct level

Use Case: UC2 – Choose a level

Scenario: The user choses a level and the game start successfully.

Preconditions: Test-Case 1.1, the game is started.

Test steps:

- The system asks the user to choose a level
- Type in the number “1”

Expected output: The game was successfully started in the console.

☒Success ☐Fail

Comments:

Test-Case 2.2: Choose test level

Use Case: UC2 – Choose a level

Scenario: The user choses the test word and the game start successfully.

Preconditions: Test-Case 1.1, the game is started.

Test steps:

- The system asks the user to choose a level
- Type in “test”

Expected output: The text “The test word is: dog” is shown in the console and the game started successfully.

☒Success ☐Fail

Comments:

Test-Case 2.3: Choose incorrect level

Use Case: UC2 – Choose a level

Scenario: The user chose non existing level and the system returns to its previous state.

Preconditions: Test-Case 1.1, the game is started.

Test steps:

- The system asks the user to choose a level
- Type in the number “9”

Expected output: The game returns to its previous state, presenting the menu options again.

☒Success ☐Fail

Comments:

Test-Case 3.1: Winning the game

Use Case: UC3 – Play Game

Scenarios: The user inputs correct letters until the word is completed.

Preconditions: Test-Case 2.2, the game is started with the test word “dog”.

Test steps:

- The system presents a randomly chosen word with underscore signs
- Type in the letter “d” and press enter
- The system exchanges the underscore for the correct letters
- Type in the letter “o” and press enter
- The system exchanges the underscore for the correct letters
- Type in the letter “g” and press enter

Expected output: The text “Congratulations, you won!” is shown in the console along with the menu options.

☒Success ☐Fail

Comments:

Test-Case 3.2: Losing the game

Use Case: UC3 – Play Game

Scenarios: The user inputs faulty letters and have no more remaining tries.

Preconditions: Test-Case 2.2, the game is started with the test word “dog”.

Test steps:

- The system presents a randomly chosen word with underscore signs
- Type in any letter, except for “d”, “o” or “g”, and press enter
- The system decreases remaining number of tries and draws a part of a man getting hanged
- Continue typing in letters, except for “d”, “o” or “g”, one at a time. Do this until remaining number of tries is zero

Expected output: The text “Sorry, you lost the game!” and a fully hanged man is shown in the console along with the menu options.

☒Success ☐Fail

Comments: Will success as long as we do not use the letters “d”, “o” or “g” when guessing.

Test-Case 4.1: Confirm quit game

Use Case: UC4 – Quit Game

Scenario: The game ends successfully.

Preconditions: Test-Case 1.1, the game is started.

Test steps:

- The system presents the menu options
- Enter the number “0”
- The system asks “Are you sure you want to quit the game? y/n”
- Enter “y”

Expected output: The text “Goodbye!” is shown in the console and the game was ended successfully.

☒Success ☐Fail

Comments:

Test-Case 4.2: Quit game without confirming

Use Case: UC4 – Quit Game

Scenario: The user does not confirm and the system returns to its previous state.

Preconditions: Test-Case 1.1, the game is started.

Test steps:

- The system presents the menu options
- Enter the number “0”
- The system asks “Are you sure you want to quit the game? y/n”
- Press enter

Expected output: The system returns to its previous state, showing the menu options again.

☒Success ☐Fail

Comments:

Test-Case 5.1: View high scores

Use Case: UC5 – View high score list

Scenario: The high score list is presented in the console.

Preconditions: Test-Case 1.1, the game is started.

Test steps:

- The system presents the menu options
- Enter the number “2”

Expected output: The high score list is shown in the console and then the menu options are displayed again.

☐Success ☒Fail

Comments: The high score list is presented but then the application turns off, the system does not present the menu options again.

3 | Automated Unit Tests

3.1 Automated Unit Tests Code

```
let assert = require('chai').assert;
let QuitGame = require('../src/QuitGame.js');
let WordGenerator = require('../src/WordGenerator.js');
let Menu = require('../src/Menu.js');

describe('Hangman Game', function () {
    describe('QuitGame', function () {
        it('should be a string', function () {
            assert.isString(QuitGame());
        });

        it('should return Goodbye! message', function () {
            assert.equal(QuitGame(), 'Goodbye!');
        });
    });

    describe('generateRandomWord', function () {
        let testArray = ['buddy'];
        let testRandom = WordGenerator.generateRandomWord(testArray);

        it('should be a string', function () {
            assert.isString(testRandom);
        });

        it('should return random word "buddy"', function () {
            assert.equal(testRandom, 'buddy');
        });
    });

    describe('getWordList', function () {
        it('should return wordList array', function () {
            assert.isArray(Menu.getWordList());
        });
    });
});
```

3.2 Test Result Screenshot

```
$ npm test

> af222ug_1dv600@1.0.0 test C:\Users\alvaf\WP\af222ug_1dv600
> mocha

buddy


  Hangman Game
    QuitGame
      √ should be a string
      √ should return Goodbye! message
    generateRandomWord
      √ should be a string
      √ should return random word "buddy"
    getWordList
      √ should return wordList array


  5 passing (15ms)
```