TESTING

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

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

| Contents

```
'use strict';

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

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

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

    describe('generateRandomWord', function () {
        let testArray = ['buddy'];
        let testRandom = WordGenerator.generateRandomWord(testArray);
        it('should return random word "buddy"', function () {
            assert.equal(testRandom, 'buddy');
        });

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

    describe('Highscore', function () {
        it('should return highscore list array', function () {
            assert.isArray(Highscore());
        })
    })
});
```

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

When performing manual testing we have chosen to test these use cases:

* Use Case 1 – Start Game
* Use Case 2 – Play Game

The reason for choosing these two use cases for this iteration is because they are important to have been fully tested before releasing the finished software.

1.2.2 Unit Tests

When performing unit testing we have chosen to test these methods:

* QuitGame()
* generateRandomWord()
*Highscore()

The reason for choosing to test these two methods is because their functionality is crucial for making sure the entire software is 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 Assignment 3	0.5h	0.5h
Learn about JavaScript Test Framework Mocha	3h	4.5h
Create Test Plan	2h	2.5h
Create and execute Manual Test Cases	3h	4h
Create and execute Automated Unit Tests	7h	6.5h
Restructure code for testing	3h	5h

2 | Manual Test Cases

Test-Case 1.1: Start game successfully

Use Case: UC1 – Start Game

Scenario: The game is started successfully.

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 successfully started game is shown in the console.

☒Success ☐Fail

Comments:

Test-Case 1.2: Admin starts game

Use Case: UC1 – Start Game

Scenario: Admin is logged in and presented with an alternate menu with options to view all words, add new word and delete word.

Preconditions: none.

Test steps:

- Start the application
- The system shows: “Enter your name:”
- Enter the name “Admin” and press enter

Expected output: The text “Welcome, Admin” and the alternative menu options are shown in the console.

☐Success ☒Fail

Comments: Output is “TODO: Implement Admin menuOptions” along with ordinary menu options.

Test-Case 2.1: Winning the game

Use Case: UC2 – Play Game

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

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

Test steps:

- The system presents a randomly chosen word with underscore signs
- Type in a letter and press enter
- The system exchanges the underscore for the correct letters
- Continue typing in letters, one at a time, until word is completed

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

☒Success ☐Fail

Comments: The Test-Case could fail based on what the randomly picked word is and what letters the tester chooses to guess, which could lead to the game being lost instead.

Test-Case 2.2: Losing the game

Use Case: UC2 – Play Game

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

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

Test steps:

- The system presents a randomly chosen word with underscore signs
- Type in a letter and press enter
- The system decreases remaining number of tries
- Continue typing in letters, one at a time, until remaining number of tries is zero

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

☒Success ☐Fail

Comments: The Test-Case could fail based on what the randomly picked word is and what letters the tester chooses to guess, which could lead to the game being won instead.

3 | Automated Unit Tests

3.1 Automated Unit Tests Code

```
'use strict';

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

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

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

    describe('generateRandomWord', function () {
        let testArray = ['buddy'];
        let testRandom = WordGenerator.generateRandomWord(testArray);
        it('should return random word "buddy"', function () {
            assert.equal(testRandom, 'buddy');
        });

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

    describe('Highscore', function () {
        it('should return highscore list array', function () {
            assert.isArray(Highscore());
        })
    })
});
```

3.2 Test Result Screenshots

```
  Hangman Game
    QuitGame
      √ should return Goodbye! message
      √ should be a string
    generateRandomWord
      √ should return random word "buddy"
      √ should be a string
    Highscore
      1) should return highscore list array


  4 passing (19ms)
  1 failing

  1) Hangman Game
       Highscore
         should return highscore list array:
     AssertionError: expected 'Highscore list' to be an array
      at Context.<anonymous> (test\Test.js:33:20)



npm ERR! Test failed.  See above for more details.
```

```
  Hangman Game
    QuitGame
      √ should return Goodbye! message
      √ should be a string
    generateRandomWord
      √ should return random word "buddy"
      √ should be a string
    Highscore
      √ should return highscore list array


  5 passing (18ms)
```

4 | Reflection

I started out with looking into which JavaScript Test Framework I should use and chose “Mocha” and tried to learn as much as possible about that Framework since I have not written any automated unit tests before. I chose to not implement any additional features during this iteration; instead I restructured a whole lot of code because I did not like how most of it looked and worked. Also I did not have that many functions that returned something and from what I could understand the functions needed to return something for me to be able to test them and therefor I needed to refactor some code to make it easier to create unit tests. The additional features will instead be implemented during next iteration.

I thought it was very hard to create the unit tests and therefor they are not very advanced. This was the first time I learned about unit testing and creating them and also using mocha so I think my unit tests work fine for those reasons.

The manual test cases took a little bit longer to execute because I wanted to try multiple times to see if the test case always succeeded or if it could fail. If the test case could fail sometimes I chose to write this as a comment to that test and mark it as a success. I was unsure how to present the results if the test case was executed multiple times.

I did not present my results as in the “greeter” example because that not how I interpreted the assignment instructions.