Software Development Project - 1DV600 - Test documentation

Claes Weyde
e-mail: cw222av@student.lnu.se
github: https://github.com/Cosbus/cw222av_1dv600_

March 8, 2019

Contents

1 Test Plan

1.1 Objectives

The objective of this testing phase is to snuff out as many problems with the application as possible. The testing is aimed at making sure the critical system functions are operational. As this is primarily a game, a large focus is put on the gaming part to make sure that it is working properly. However, since the words used in the game are of great importance to minimize repetitiveness quite a large focus is put on the functionality regarding adding, deleting and updating the words.

1.1.1 What to test?

As was described above the main testing will be performed on the gaming part of the application. Manual test will be performed to determine that all the critical functionality works with regards to game play, specifically UC2 - Play game (UC 1 is concerned with the main menu and is not tested here). The main scenario as well as some alternative scenarios will be tested.

- Choosing a game difficulty.
- Providing a letter successfully and unsuccessfully.
- Quitting the game.
- Finishing a word.

Some attention will also be put on managing words, i.e. UC 5, 6 and 7. All of the manual dynamic tests are to be performed after the code for the use cases have been implemented. For a full breakdown of the manual testing please see section 4.

Automated unit tests will also be written to test some of the methods provided in the different objects of the application.

1.2 To-do

- Study course material regarding testing
- Manual TC - planning

- Manual TC - perform
- Automated unit test - planning (including learning frameworks etc.)
- Automated unit test - perform
- Test report

1.3 Time estimates and Time log

Task	Estimate [hrs]	Actual time
Study course material regarding testing	5	3
Manual TC - planning	2	2.5
Manual TC - perform	0.5	0.5
Automated unit test - planning	4	3.5
Automated unit test - perform	4	2.5
Test report	2	2.5
Total time	17.5	14.5

2 Manual test cases

2.1 Manual tests of Play Game main scenario

2.1.1 TC 2.2 - Successfully providing a correct letter

- Name: Successfully providing a correct letter.
- Test case ref: TC 2.2
- Use case ref: UC 2
- Description: The test case tests how the system responds when the user provides a letter which is correct, and the word is not completely solved.

Input

1. Precondition: The user has started the game.
2. Test case TC 2.1. choose "play game" from main menu.
3. Input a correct letter.
4. Press enter.

Expected

- The system prints "Correct!" on the screen.
- The system prints the number of letters remaining on the screen.
- The system prints the number of tries remaining on the screen.
- The system prompts the user to enter a new letter by printing "Please provide a letter:" on the screen.

Test successful ☐
Test not successful ☐

Comments

2.1.2 TC 2.3 - Providing an incorrect letter

- Name: Providing an incorrect letter.
- Test case ref: TC 2.3
- Use case ref: UC 2
- Description: The test case tests how the system responds when the user provides a letter which is incorrect, and the user has more tries.

Input

1. Precondition: The user has started the game.
2. Test case TC 2.1. choose "play game" from main menu.
3. Input an incorrect letter.
4. Press enter.

Expected

- The system prints "Unfortunately wrong." on the screen.
- The system prints the number of letters remaining on the screen.
- The system prints the number of tries remaining on the screen.
- The system prompts the user to enter a new letter by printing "Please provide a letter:" on the screen.

Test successful ☐
Test not successful ☐

Comments No problems detected.

2.1.3 TC 2.6 - Returning from play to main menu

- Name: Returning from play to main menu.
- Test case ref: TC 2.6
- Use case ref: UC 2
- Description: The test case tests how the system responds when the user has finished a word (either by finding the word or having exhausted his/hers amount of tries) and chooses to return to the main menu.

Input

1. Precondition: The user has finished a word and the system has provided the user with the choice of returning to main menu, playing a new word or quitting the game altogether.
2. Finish a word.
3. Input "m" to return to main menu.
4. Press enter.

Expected

- The system shows the main menu on the screen.

Test successful ☐
Test not successful ☐

Comments No problems encountered.

2.1.4 TC 2.a* - Quitting the game while playing

- Name: Quitting the game while playing.
- Test case ref: TC 2.a*
- Use case ref: UC 2
- Description: The test case tests how the system responds when the user enters the code for quitting the game while playing.

Input

1. Precondition: The user is playing the game.
2. Input "q-t" and press enter.
3. The system asks the user if he/she really wants to quit.
4. Input "y" and press enter.

Expected

- The system quits the application entirely.

Test successful ☐
Test not successful ☐

Comments No problems encountered.

2.1.5 TC 5.3 - Inputting a new word

- Name: Inputting a new word.
- Test case ref: TC 5.3
- Use case ref: UC 5
- Description: The test case tests how the system responds when the user enters a new word to save to the word list.

Input

1. Precondition: The user has moved to the manage words menu and chosen to add a new word. The system prompts the user to input the word by printing "Input a word:" on the screen.
2. Input a word and press enter.

Expected

- The system prints "Make a choice: (use arrow keys)".
- The system prints the three difficulty levels "easy", "moderate" and "hard" which the user can move between.

Test successful ☐
Test not successful ☐

Comments Works as expected.

2.1.6 TC 5.5 - Choosing a difficulty for an input word

- Name: Choosing a difficulty for an input word.
- Test case ref: TC 5.5
- Use case ref: UC 5
- Description: The test case tests how the system responds when the user enters a difficulty for an input word.

Input

1. Precondition: The user has input a word and pressed enter.
2. TC 5.3. Choose a difficulty presented by the system.
3. Press enter.

Expected

- The system prints "Successfully saved" in green text.
- The system prints "Add another word? [y/n]" and waits for the user to respond.

Test successful ☐
Test not successful ☐

Comments No problems found.

2.1.7 TC 6.2 - Choosing to remove a word

- Name: Choosing to remove a word.
- Test case ref: TC 6.2

- Use case ref: UC 6
- Description: The test case tests how the system responds when the user chooses to remove a word.

Input

1. Precondition: The user has moved to the manage words menu and the system has printed the menu on the screen.
2. Choose to remove a word by moving the cursor to the "delete word" choice using the arrow keys.
3. Press enter.

Expected

- The system prints "Remove word. What difficulty are you interested in" on the screen.
- The system prints "Make a choice: (use arrow keys)" on the screen.
- The system prints the choices "easy", "moderate" and "hard" on the screen and waits for the user choice.

Test successful ☐
Test not successful ☐

Comments Works, the word has been removed.

2.1.8 TC 6.3.1 - Choosing a difficulty for the word to remove

- Name: Choosing a difficulty for the word to remove.
- Test case ref: TC 6.3.1
- Use case ref: UC 6
- Description: The test case tests how the system responds when the user chooses a difficulty for the word to be removed.

Input

1. Precondition: The user has chosen to remove a word (TC 6.2).
2. Choose a difficulty and press enter.

Expected

- The system prints all the words on file for the given difficulty on the screen and waits for the user choice. The user can move between the words using the arrow keys to choose.

Test successful ☐
Test not successful ☐

Comments Works.

2.1.9 TC 6.3.2 - Choosing a word to remove from the list

- Name: Choosing a word to remove from the list.
- Test case ref: TC 6.3.2
- Use case ref: UC 6
- Description: The test case tests how the system responds when the user chooses a word from the list to remove.

Input

1. Precondition: The user has chosen a difficulty for the word to remove (TC 6.3.1).
2. Choose a word from the list by using the arrow keys.
3. Press enter.

Expected

- The system prints "Are you sure you want to remove "[thechoosen-word]" [y/n]?" and awaits the users answer.

Test successful ☐
Test not successful ☐

Comments No problems.

2.1.10 TC 6.3.2a - Confirming removal of a chosen word

- Name: Confirming removal of a chosen word.
- Test case ref: TC 6.3.2a
- Use case ref: UC 6
- Description: The test case tests how the system responds when the user confirms removal of the chosen word.

Input

1. Precondition: The user has chosen a word to remove from the list (TC 6.3.2).
2. Write "y".
3. Press enter.

Expected

- The system prints "[word] successfully removed!" in green text.
- The system prints "Delete another word? [y/n]" and awaits the user response.

Test successful ☐
Test not successful ☐

Comments No problems.

2.1.11 TC 6.3.2b - Not confirming to remove a chosen word

- Name: Not confirming to remove a chosen word.
- Test case ref: TC 6.3.2b
- Use case ref: UC 6
- Description: The test case tests how the system responds when the user chooses to not confirm to remove a word chosen for removal.

Input

1. Precondition: The user has chosen a word to remove from the list (TC 6.3.2).
2. Write "n".
3. Press enter.

Expected

- The system prints "Maybe a wise choice?".
- The system prints "Do you want to remove another word [y/n]? and awaits the users answer.

Test successful ☐
Test not successful ☐

Comments Not successful. The system moves back to the choice of choosing a difficulty to remove. When continuing the system goes haywire after a while. It would seem that the system emits new events while not removing old ones. This must be taken care of.

2.1.12 TC 7.3 - Choosing a word to update

- Name: Choosing a word to update.
- Test case ref: TC 7.3
- Use case ref: UC 7
- Description: The test case tests how the system responds when the user chooses a word to update.

Input

1. Precondition: The user has chosen to update a word from the manage word menu and chosen which difficulty the word should belong to (TC 7.2).
2. Choose a word by using the arrow keys.

3. Press enter.

Expected

- The system prints "What word do you want to update "[word]" to?".
- The system awaits the user choice.

Test successful ☐
Test not successful ☐

Comments Works.

2.1.13 TC 7.5 - Choosing a word to update to

- Name: Choosing a word to update to.
- Test case ref: TC 7.5
- Use case ref: UC 7
- Description: The test case tests how the system responds when the user inputs a word to update to.

Input

1. Precondition: The user has chosen a word to update from the list of words (TC 7.2).
2. Write the word to update to.
3. Press enter.

Expected

- The system prints "Are you sure you want to update "[oldword]" to "[newword]" [y/n]?" and awaits the user answer.

Test successful ☐
Test not successful ☐

Comments Works.

2.1.14 TC 7.6 - Answering with wrong letter to [y/n] question

- Name: Answering with wrong letter to [y/n] question.
- Test case ref: TC 7.6
- Use case ref: UC 7
- Description: The test case tests how the system responds when the user answers with an un-allowed letter to a yes/no question when updating words.

Input

1. Precondition: The user has entered a word to update to (TC 7.5).
2. Write any letter or number which differs from "y" or "n".
3. Press enter.

Expected

- The system prints "Please answer "y" or "n"!".
- The system prints "Are you sure you want to update "[oldword]" to "[newword]" [y/n]?" and awaits the user answer.

Test successful ☐
Test not successful ☐

Comments Works.

3 Automated tests

The unit test cover a class called authenticator to authenticate if the user can manage words, a class called FileHandler which handles the saving and loading of words for a given game, and a class called Word which handles a single word in the wordlist. The frameword Mocha together with Chai and Chai-as-promised (for testing asynchronous methods) was used for performing automated testing in Node.js.

3.1 Test runs

A screenshot of the output from the automated tests can be found in figure 3.1. As can be seen from figure 3.1, the method getSumAndQ in the Authenticator class fails for a certain scenario. Figure 3.1 show a new test-run performed after having fixed the bug in the code. As can be seen, all test now succeed.

```
Authenticator:determineNumbers
  for any valid number
    √ should return an array of length 2
  for any valid ceiling number
    √ should return two numbers below or equal to the ceiling number

Authenticator:getSumAndQ
  without arguments
    √ should return 0
  with only one argument
    √ should return that argument
  for any two numbers
    1) should return the sum of the numbers

FileHandler:loadWordList
  with "difficulty" set to "easy"
    √ should return an array with length > 1
  with "difficulty" set to "moderate"
    √ should return an array with length > 1
  with "difficulty" set to "hard"
    √ should return an array with length > 1
  with "difficulty" set to nothing
    √ should throw an error

Word:hasLetter
  when called for the first time with a letter comprised in the word
    √ should return true
  when called a second time with the same letter which was previously comprised in the word
    √ should return false
  when called with a letter not comprised in the word
    √ should return false


11 passing (61ms)
1 failing

1) Authenticator:getSumAndQ
     for any two numbers
       should return the sum of the numbers:

    AssertionError: expected 2 to equal 4
```

Figure 1: Automated test run with a bug in the code.

```
Authenticator:determineNumbers
  for any valid number
    √ should return an array of length 2
  for any valid ceiling number
    √ should return two numbers below or equal to the ceiling number

Authenticator:getSumAndQ
  without arguments
    √ should return 0
  with only one argument
    √ should return that argument
  for any two numbers
    √ should return the sum of the numbers

FileHandler:loadWordList
  with "difficulty" set to "easy"
    √ should return an array with length > 1
  with "difficulty" set to "moderate"
    √ should return an array with length > 1
  with "difficulty" set to "hard"
    √ should return an array with length > 1
  with "difficulty" set to nothing
    √ should throw an error

Word:hasLetter
  when called for the first time with a letter comprised in the word
    √ should return true
  when called a second time with the same letter which was previously comprised in the word
    √ should return false
  when called with a letter not comprised in the word
    √ should return false


12 passing (28ms)
```

Figure 2: Automated test run with the bug removed.

3.2 The test code

The test code can be found in the figures below. Three files were used, one for each class tested (due to space constraints there are two screenshots for each test file).

```
/**
 * Test module for Authenticator
 *
 * @module test/testAuthenticator.js
 * @author Claes Weyde
 * @version 1.0.0
 */

const chai = require('chai')
const expect = chai.expect
const chaiAsPromised = require('chai-as-promised')
chai.use(chaiAsPromised)

const Authenticator = require('../src/lib/Authenticator.js')
const auth = new Authenticator()

/**
 * Testing the determineNumbers method of Authenticator
 */
describe('Authenticator:determineNumbers', function () {
  context('for any valid number', function () {
    it('should return an array of length 2', function () {
      expect(auth.determineNumbers(4)).to.be.a('array').and.to.have.lengthOf(2)
    })
  })

  context('for any valid ceiling number', function () {
    it('should return two numbers below or equal to the ceiling number', function () {
      expect(auth.determineNumbers(15)[0]).to.be.below(16)
      expect(auth.determineNumbers(15)[1]).to.be.below(16)
    })
  })
})
```

Figure 3: Test code for Authenticator 1 of 2

```
/**
 * Testing the getSumAndQ method of Authenticator
 */
describe('Authenticator:getSumAndQ', function () {
  context('without arguments', function () {
    it('should return 0', function () {
      expect(auth.getSumAndQ().theSum).to.equal(0)
    })
  })

  context('with only one argument', function () {
    it('should return that argument', function () {
      expect(auth.getSumAndQ(5).theSum).to.equal(5)
    })
  })

  context('for any two numbers', function () {
    it('should return the sum of the numbers', function () {
      expect(auth.getSumAndQ(2, 2).theSum).to.equal(4)
    })
  })
})
```

Figure 4: Test code for Authenticator 2 of 2

```
/**
 * Test module for FileHandler
 *
 * @module test/testFileHandler.js
 * @author Claes Weyde
 * @version 1.0.0
 */

const chai = require('chai')
const expect = chai.expect
const chaiAsPromised = require('chai-as-promised')
chai.use(chaiAsPromised)

const Filehandler = require('../src/lib/FileHandler.js')
const fh = new Filehandler()

/**
 * Testing the loadwordlist function of filehandler
 */
describe('FileHandler:loadWordList', function () {
  context('with "difficulty" set to "easy"', function () {
    it('should return an array with length > 1', function () {
      return fh.loadWordList('easy')
        .then(function (array) {
          expect(array).to.be.a('array').and.to.have.lengthOf.above(1)
        })
    })
  })
```

Figure 5: Test code for FileHandler 1 of 2

```
  context('with "difficulty" set to "moderate"', function () {
    it('should return an array with length > 1', function () {
      return fh.loadWordList('moderate')
        .then(function (array) {
          expect(array).to.be.a('array').and.to.have.lengthOf.above(1)
        })
    })
  })

  context('with "difficulty" set to "hard"', function () {
    it('should return an array with length > 1', function () {
      return fh.loadWordList('hard')
        .then(function (array) {
          expect(array).to.be.a('array').and.to.have.lengthOf.above(1)
        })
    })
  })

  context('with "difficulty" set to nothing', function () {
    it('should throw an error', async () => {
      await expect(fh.loadWordList('')).to.be.rejected
    })
  })
})
```

Figure 6: Test code for FileHandler 2 of 2

```
/**
 * Test module for Word
 *
 * @module test/testWord.js
 * @author Claes Weyde
 * @version 1.0.0
 */

const chai = require('chai')
const expect = chai.expect
const assert = chai.assert      module "chai-as-promised"
const chaiAsPromised = require('chai-as-promised')
chai.use(chaiAsPromised)

const Word = require('../src/lib/Word.js')
const word = new Word()

/**
 * Setting up the word object
 */
word.setWord('testing')

/**
 * Testing the hasLetter method of Word
 */
describe('Word:hasLetter', function () {
  context('when called for the first time with a letter comprised in the word', function () {
    it('should return true', function () {
      let actual = word.hasLetter('t')
      assert.isTrue(actual)
    })
  })
```

Figure 7: Test code for Word 1 of 2

```
  context('when called a second time with the same letter which was previously comprised in the word', function () {
    it('should return false', function () {
      let actual = word.hasLetter('t')
      assert.isFalse(actual)
    })
  })

  context('when called with a letter not comprised in the word', function () {
    it('should return false', function () {
      let actual = word.hasLetter('q')
      assert.isFalse(actual)
    })
  })
})
```

Figure 8: Test code for Word 2 of 2

4 Test report for tests

4.1 Manual test matrix

Test	UC 2	UC 4	UC 5	UC 6	UC7
TC 2.2	1	0	0	0	0
TC 2.3	1	0	0	0	0
TC 2.6	1	0	0	0	0
TC 2.a*	1	0	0	0	0
TC 5.3	0	0	1	0	0
TC 5.5	0	0	1	0	0
TC 6.2	0	0	0	1	0
TC 6.3.1	0	0	0	1	0
TC 6.3.2	0	0	0	1	0
TC 6.3.2a	0	0	0	1	0
TC 6.3.2b	0	0	0	1/Not OK	0
TC 7.3	0	0	0	0	1
TC 7.5	0	0	0	0	1
TC 7.6	0	0	0	0	1
Coverage and success	4/OK	0	2/OK	3/4 OK	3/OK

4.2 Automated test matrix

Test	Word	Funcs	FileHandler	Authenticator
testWord	3	0	0	0
testFileHandler	0	0	4	0
testAthenticator	0	0	0	5
Coverage and success	3/OK	0	4/OK	4/OK

Test	Game	WordList	HighScores	Player
testWord	0	0	0	0
testFileHandler	0	0	0	0
testAthenticator	0	0	0	0
Coverage and success	0	0	0	0

5 Reflection

Having performed these tests I believe that testing is not only crucial for finding faults within the code, but also that testing at an early phase will actually make the code better. I believe this is especially true for the unit tests since these made me realize and think more about my actual method implementations, specifically with regards to my inputs and the outputs received. Thus, the code would not only be better because less bugs are found, but also because the code is logical with a good separation of concerns between different components. Continuing forward my goal will be to test early if possible.