Name: Viktor Ödman
Email: vo222dq@student.lnu.se
Github: https://gitlab.lnu.se/1dv600/student/vo222dq/assignment-3

TEST PLAN

Objective

The objective is to test some of the code and requirements that was implemented in the previous iteration and also to test some of the code that will be implemented in this iteration.

What to test and how

We are going to test Use Case 3 (Quit Game) and Use Case 2 (Play Game) by running dynamic test-cases. Use Case 2 will not be tested until the functionality to test that use-case is implemented. We will also write automated unit test for the methods ”getFilesInDir” and ”getJsonFile” in the FileReader Class.The method getFilesInDir will have two tests and getJsonFile will have three tests. We are also going to create a unit test for the method ”checkWordComplete” in the GameLogic class.

Rationale

The reason for choosing to test Use Case 3 is to see if it is possible to exit the application. The reason for choosing Use Case 2 is to test if it was possible to play through the game the way that the Use Case described it. The reason for choosing to write unit tests for ”getFilesInDir” and ”getJsonFile” is to see if the error handling for those methods works as intended. Since the file path for the word lists are currently ”hard coded” in the program, this will be a good way to find bugs if the file path is changed.

Time Plan

Task	Estimated Time	Actual
Writing Manual Test Cases	2 h	3 h
Running the manual tests	30 min	10 min
Writing Unit tests	4 h	6 h
Writing a reflection	1 h	1 h

MANUAL TEST-CASES

TC 3.1 To quit the game

Link to UC3

Description: The test checks if the player can successfully exit the application. This is tested to check if the player can successfully exit the application from the main menu and to check if the exit confirmation works.

Use case: UC3 Quit Game

Scenario: To quit the game

The main scenario of UC3 is tested when the player successfully quits the game.

Precondition: The game should be started

Test steps

- The system shows the menu with a selected menu item. (The selected item has a blue color and a greater-than sign to the left of the text)
- By using the arrow keys on the keyboard, select the menu item with the text ”Quit Game” and press the enter key.
- The system displays the text ”Are you sure you want to quit” and asks the player to confirm by entering the letter y or n.
- By using the keyboard, enter the letter y and press enter.

Expected

- The system should now be exited

TC 2.1 Winning the game

<u>Link to UC2</u>

Description: The test checks if the player can successfully play the game and win. This is tested to check that the output of the game changes as intended and that the input works.

Use case: UC2 Play Game

Scenario: Playing the game and winning

The main scenario of UC2 is tested when the player successfully wins a game of hangman.

Precondition: While being in the directory /assignment-3/Terminal-Hangman, the game should be started by entering ”npm start test”.

Test steps

- The system displays the text ”Guessed Letters:”, a drawing of the ”gallows pole”, the text ”Placeholders: __” and asks the user to enter a letter.
- By using the keyboard, enter the letter t the press enter.
- The system displays the text ”Guessed Letters: t”, a drawing of the ”gallows pole”, the text ”Placeholders: t_” and asks the user to enter a letter.
- By using the keyboard, enter the letter e the press enter.

Expected

- The system presents the texts ”YOU WIN”, ”Wrong Guesses: 0” and ”

Correct Word: te”
- The system asks the player to play again.

Test Report

Test traceability matrix and success

TEST	UC2	UC3
TC3.1	0	1/OK
TC2.1	1/OK	0
COVERAGE & SUCCESS	1/OK	1/OK

Comment

Both test cases where successful.

UNIT TESTS

Link to the test code

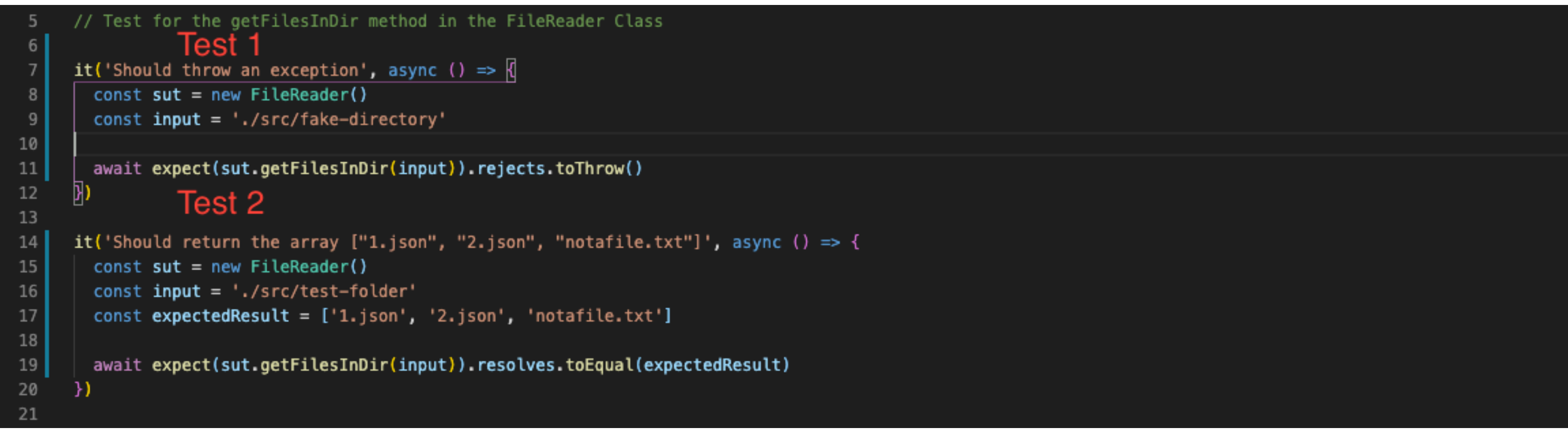
The methods that will be tested ”getFilesInDir” and ”getJsonFile” in the FileReader class. The getFilesInDir method will have two tests and the getJsonFile will have three tests.

Unit tests for getFilesInDir

Link to getFilesInDir

The function takes a file path to a directory as an argument and returns the files in that directory as an array. If the directory does not exist, the function should throw an exception.

Test 1 will test if the function throws an exception if the directory does not exist.
Test 2 will test if the function returns the files in a specific directory.



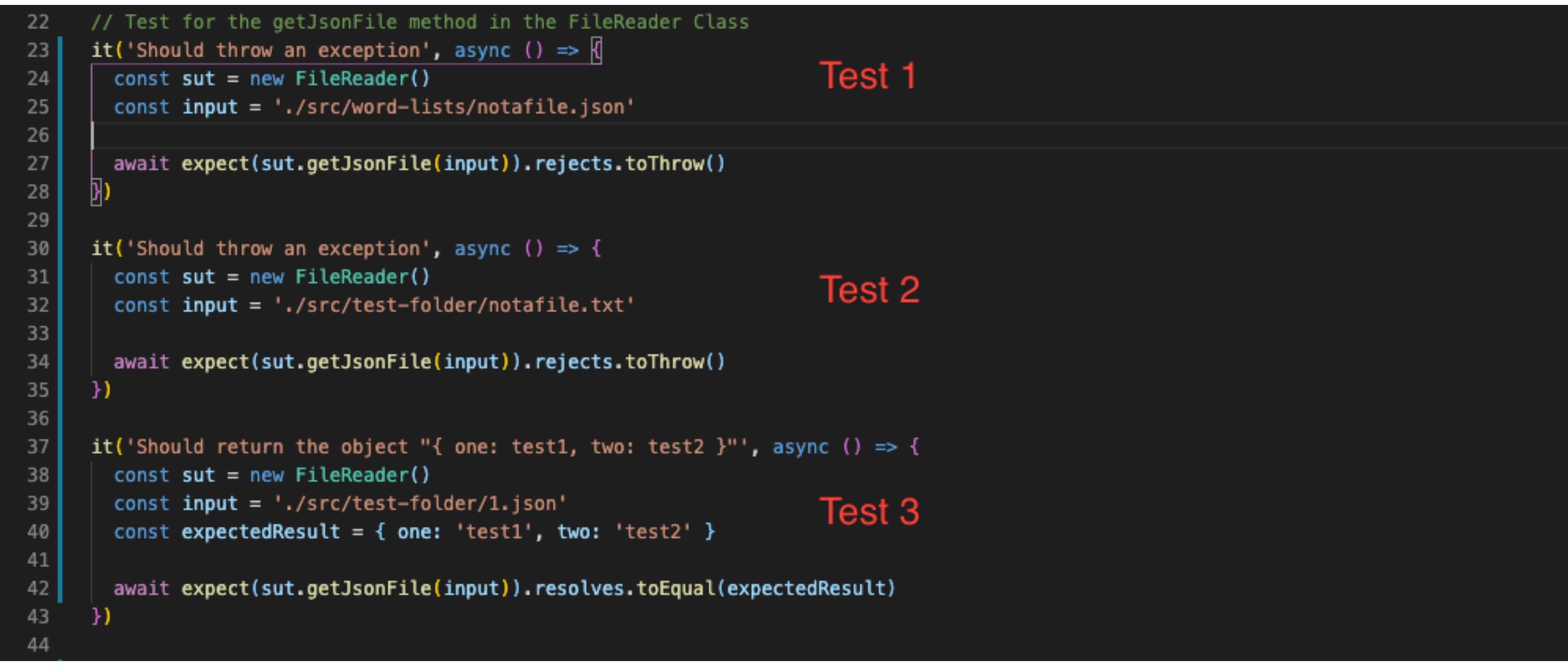
```
// Test for the getFilesInDir method in the FileReader Class

it('Should throw an exception', async () => {
  const sut = new FileReader()
  const input = './src/fake-directory'

  await expect(sut.getFilesInDir(input)).rejects.toThrow()
})

it('Should return the array ["1.json", "2.json", "notafile.txt"]', async () => {
  const sut = new FileReader()
  const input = './src/test-folder'
  const expectedResult = ['1.json', '2.json', 'notafile.txt']

  await expect(sut.getFilesInDir(input)).resolves.toEqual(expectedResult)
})
```

Unit test for getJsonFile

Link to getJsonFile

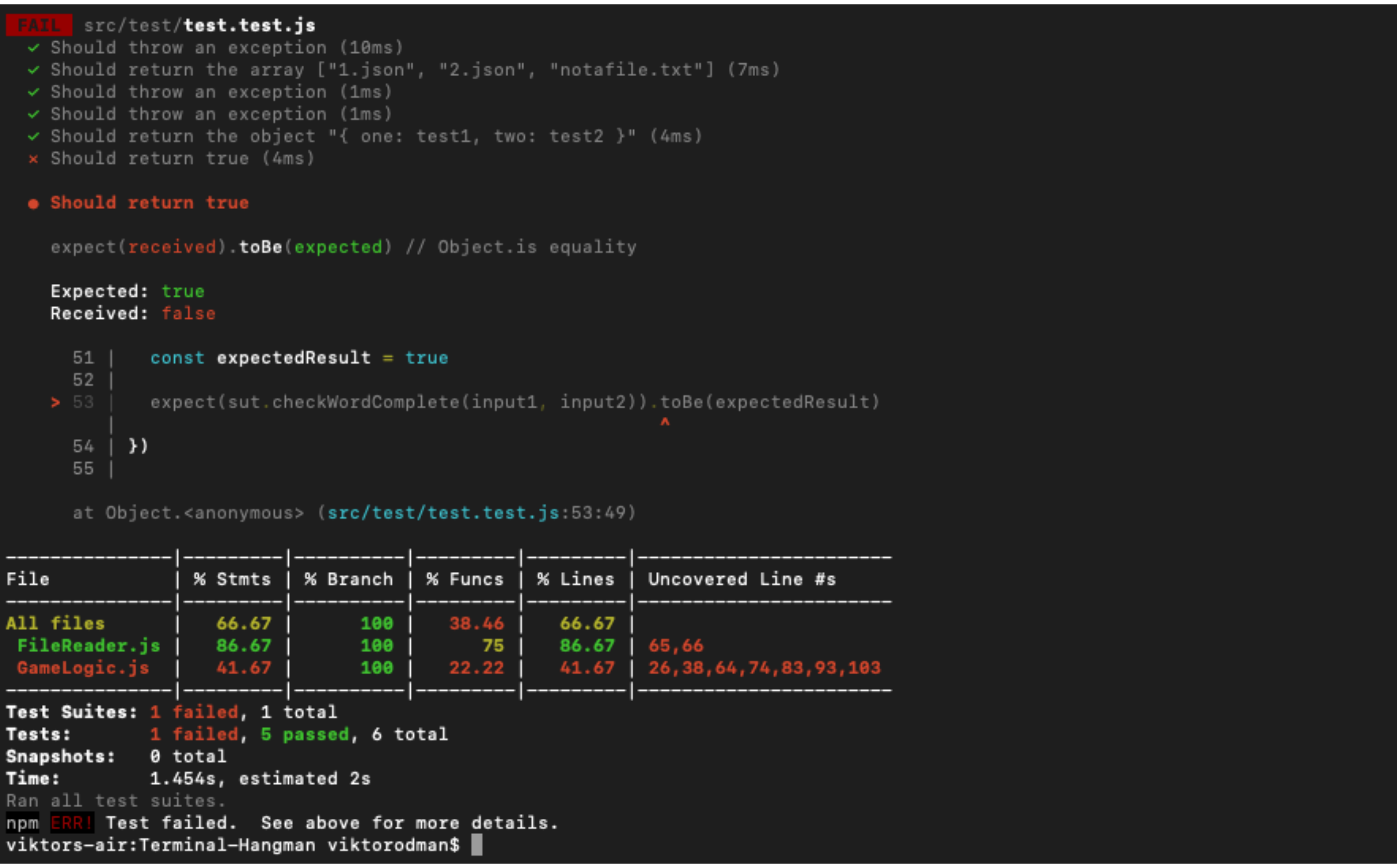
The function takes a file path to a specific json file as an argument and returns the json file converted to a javascript object. If the passed file path does not exist, the function should throw an exception. If the passed file is not an json file, the function should throw an exception.

Test 1 will test if the function throws an exception if the passed file does not exist.
Test 2 will test if the function throws an exception if the passed file is not a json file.
Test 3 will test if the function returns a specific javascript object when passed an existing json file.



```
// Test for the getJsonFile method in the FileReader Class
it('Should throw an exception', async () => {
  const sut = new FileReader()
  const input = './src/word-lists/notafile.json'

  await expect(sut.getJsonFile(input)).rejects.toThrow()
})

it('Should throw an exception', async () => {
  const sut = new FileReader()
  const input = './src/test-folder/notafile.txt'

  await expect(sut.getJsonFile(input)).rejects.toThrow()
})

it('Should return the object "{ one: test1, two: test2 }"', async () => {
  const sut = new FileReader()
  const input = './src/test-folder/1.json'
  const expectedResult = { one: 'test1', two: 'test2' }

  await expect(sut.getJsonFile(input)).resolves.toEqual(expectedResult)
})
```

Test result



```
 FAIL  src/test/test.test.js
  ✓ Should throw an exception (10ms)
  ✓ Should return the array ["1.json", "2.json", "notafile.txt"] (7ms)
  ✓ Should throw an exception (1ms)
  ✓ Should throw an exception (1ms)
  ✓ Should return the object "{ one: test1, two: test2 }" (4ms)
  × Should return true (4ms)

  ● Should return true

    expect(received).toBe(expected) // Object.is equality

    Expected: true
    Received: false

      51 |     const expectedResult = true
      52 |
    > 53 |     expect(sut.checkWordComplete(input1, input2)).toBe(expectedResult)
         |                                                    ^
      54 |   })
      55 |

      at Object.<anonymous> (src/test/test.test.js:53:49)

---------------|----------|----------|----------|----------|-------------------|
File           |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
---------------|----------|----------|----------|----------|-------------------|
All files      |    66.67 |      100 |    38.46 |    66.67 |                   |
 FileReader.js |    86.67 |      100 |       75 |    86.67 | 65,66             |
 GameLogic.js  |    41.67 |      100 |    22.22 |    41.67 | 26,38,64,74,83,93,103 |
---------------|----------|----------|----------|----------|-------------------|
Test Suites: 1 failed, 1 total
Tests:       1 failed, 5 passed, 6 total
Snapshots:   0 total
Time:        1.454s, estimated 2s
Ran all test suites.
npm ERR! Test failed.  See above for more details.
viktors-air:Terminal-Hangman viktorodman$ 
```

CLASS	TESTED METHOD	TEST RESULT
FileReader	getFilesInDir	2/2 Passed
FileReader	getJsonFile	3/3 Passed
GameLogic	checkWordComplete	0/1 Passed

REFLECTION

This was an interesting assignment. I had to refactor a lot of code to make it more testable. It took some time to understand to be very descriptive when writing the manual test cases. Writing a test ”flag” for the game really helped me shorten the test case for UC2.
Learning how to use the tool ”jest” for the automatic unit tests was pretty easy when trying to test simpler functions. I had some struggle when I was testing asynchronous functions and trying to make the function throw exceptions. I can see how creating test for future implementation could be useful.