HANGMAN SOFTWARE PROJECT

Iteration 3 – Test documentation

NAME: Niall Thurrat

lnu email address: nt222fc@student.lnu.se

github repo: https://github.com/niall-thurrat/nt222fc_1dv600.git
github latest release: https://github.com/niall-thurrat/nt222fc_1dv600/releases/tag/v1.2

Contents

1 Test Plan

1.1 What are the objectives of the testing in this iteration?

The objectives of the testing during this iteration are to test the code which has been developed during the first and second iterations of the project, to ensure that 1) the program functions as its expected to do (in terms of its functional requirements), and 2) to identify defects which may be present in the code.

1.2 What to test? Include a short rationale

I intend to do manual test cases for both UC 1 (Start Game) and UC 2 (Play Game). I have chosen UC 1 as it is the first use case and faults in this part of the program should be identified first, as they may affect the functionality of further use cases. I have chosen to test UC 2 (Play Game) as it is the primary function of the application and the main reason the user will be using the app.

I intend to do 2 automated unit tests on the imageGenerator function as it has simple input and output and testing will give me confidence that images will be returned properly when called in the playGame function. I will do 2 automated unit tests on the wordUpdater function as this has a slightly more complicated input parameter object and the tests will be useful in future iterations to ensure that problems created by code changes will be identified. I will also create a mock function on the imageGenerator module and a failing test to show the test can detect that the code does not work.

1.3 How this testing is going to be done?

The tests will be dynamic, i.e. the application will be executed during testing. I shall use manual testing and automated unit tests.

1.4 Time plan

TASK	ESTIMATED (hrs:mins)	ACTUAL (hrs:mins)
Manual TC 1 – 2 planning	2	as estimated
Running manual TC 1 - 2	0:20	0:05
Unit Test 1 – 4 planning and coding	4	8*
Running and documenting unit test 1 - 4	0:20	as estimated
Reflections	1	0:50

*** The large gap between estimated and actual time taken for unit testing is because I had difficulty learning how to use mocha and chai (particularly with thrown exceptions) and needed to complete a few tutorials**

2 Manual Test Cases

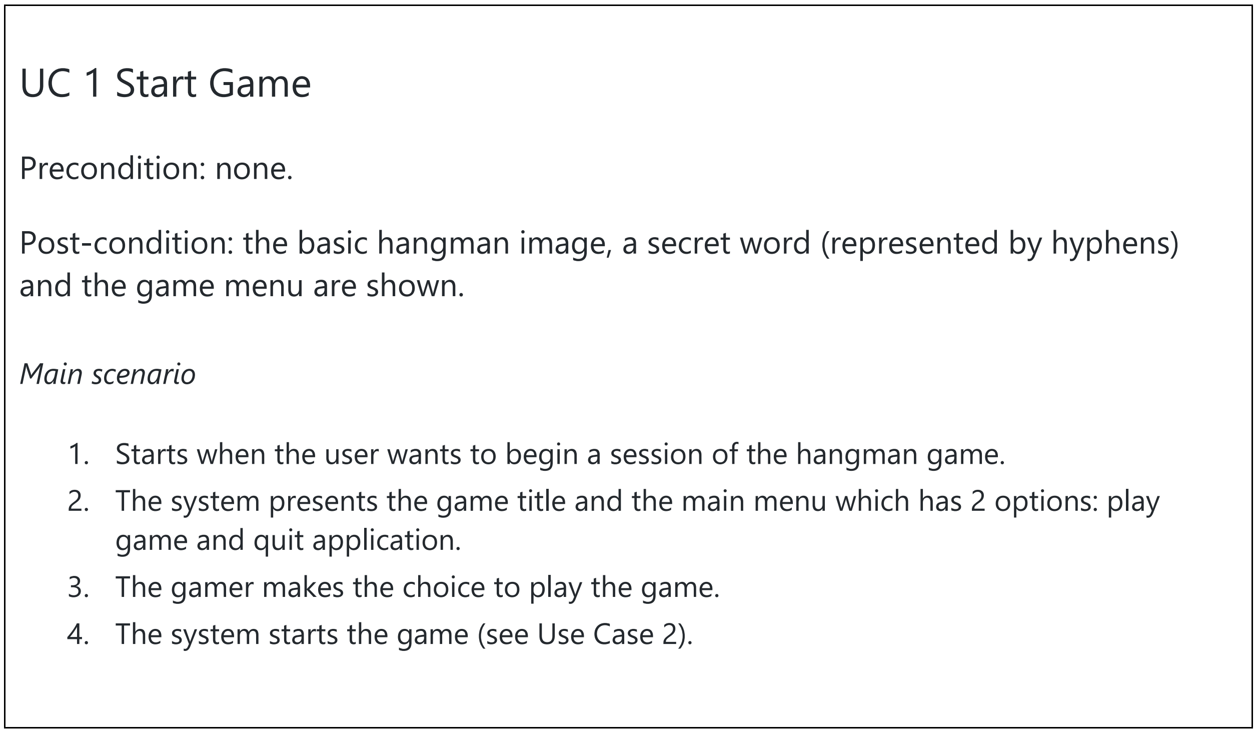
2.1 Start Game Successful Test Case

2.1.1 Name and id of the test-case

Name: Start Game Successful
ID: TC1.1

2.1.2 Reference to what requirement (use-case) that is tested

Use Case 1 – Start Game (main scenario) is tested



UC 1 Start Game

Precondition: none.

Post-condition: the basic hangman image, a secret word (represented by hyphens) and the game menu are shown.

Main scenario

1. Starts when the user wants to begin a session of the hangman game.
2. The system presents the game title and the main menu which has 2 options: play game and quit application.
3. The gamer makes the choice to play the game.
4. The system starts the game (see Use Case 2).

2.1.3 Short description of what is tested and why

The main scenario from the Start Game use case will be tested here to ensure that the game can be started as it should from the main menu. This is the starting point of the application and will therefore be the starting point of my testing as it is important to ensure this functions as it should before moving on to testing other scenarios or other uses cases which rely on this part of the application to function properly.

2.1.4 Preconditions that need to be fulfilled before this test is can be started

All application files as well as dependencies from npm must be installed to prepare the application environment (in an IDE) and the app must be started using the command 'npm start' in the terminal before the test can be completed. No other test cases need to be ran before this one.

2.1.5 Test-steps including test-data.

- Press '1' to select Play Game in main menu

2.1.6 A description of the expected result.

The system shows the base hangman image, a secret word (represented by hyphens) and the game menu which has 3 options: guess a letter, quit game and quit application.

2.1.7 Checkboxes if the test did succeed or fail.

☑ The system shows the base hangman image, a secret word (represented by hyphens) and the game menu which has 3 options: Guess a letter, Quit Game and Quit Application.

2.1.8 A space for comments by the tester.

Nothing noteworthy. Test passed as expected.

2.2 Play Game Test Case

2.2.1 Name and id of the test-case.

Name: Play Game Test Case
ID: TC2.1

2.2.2 Reference to what requirement (use-case) that is tested.

Use Case 2 – Play Game (main scenario) is tested

UC 2 Play Game

Precondition: a random secret word has been selected.
Post-condition: the gamer has been notified if they have won or lost the game.

Main scenario:

1. Starts when the gamer wants to play a game of hangman.
2. System presents a base hangman image, secret word (represented by hyphens) and game menu.
3. Gamer guesses a letter.
4. System updates the secret word where letter is found.

Gamer repeats steps 3-4 until word is complete.

5. System generates a winner message then returns to main menu.

2.2.3 Short description of what is tested and why.

The main scenario from the Play Game use case will be tested here to ensure that when the correct letters are entered the game will be won as expected. This will be tested as it is important to ensure that the game can be won by the player as expected.

2.2.4 Preconditions that need to be fulfilled before this test is can be started.

TC1.1 should be passed before conducting this test case. UC1 (main scenario) must be completed to start the game and there should be a comment at the top of the game which reveals the 6 letter secret word for the purposes of this test, e.g. 'SECRET WORD REVEALED FOR DEV TESTING: retain'.

2.2.5 Test-steps including test-data.

1. The game menu will give you 3 options: [1] Guess a letter, [2] Quit game and [3] Quit application. Type 1 on your keyboard to select the first option.
2. The question 'What letter do you want to guess?' will appear. For test purposes you will see the secret word answer revealed at the top of the game screen. On your keyboard, type the first letter which is currently represented by a hyphen in the six letter secret word, then press 'Enter'.
3. The system should replace the hyphen(s) in the 'SECRET WORD' which represent the letter you correctly entered, e.g. if you enter the letter 'r' and only the first letter of the secret word (which is initially represented as '- - - - - -') is an 'r', then you should see 'SECRET WORD: r - - - - -' on the screen. If there are other instances of your entered letter in the revealed word, these should also be replaced in the secret word (check this against the revealed word).
4. *Repeat step 1-3 until you have entered all letters to complete the secret word.*

2.2.6 A description of the expected result.

- The system will show the message 'YOU WIN!!' for 3 seconds on the screen.
- The application will then present the main menu which has 2 options: [1] play game and [2] quit application.

2.2.7 Checkboxes if the test did succeed or fail.

☑ The system will show the message 'YOU WIN!!' for 3 seconds on the screen.

☑ The application will then present the main menu which has 2 options: [1] play game and [2] quit application.

2.2.8 A space for comments by the tester.

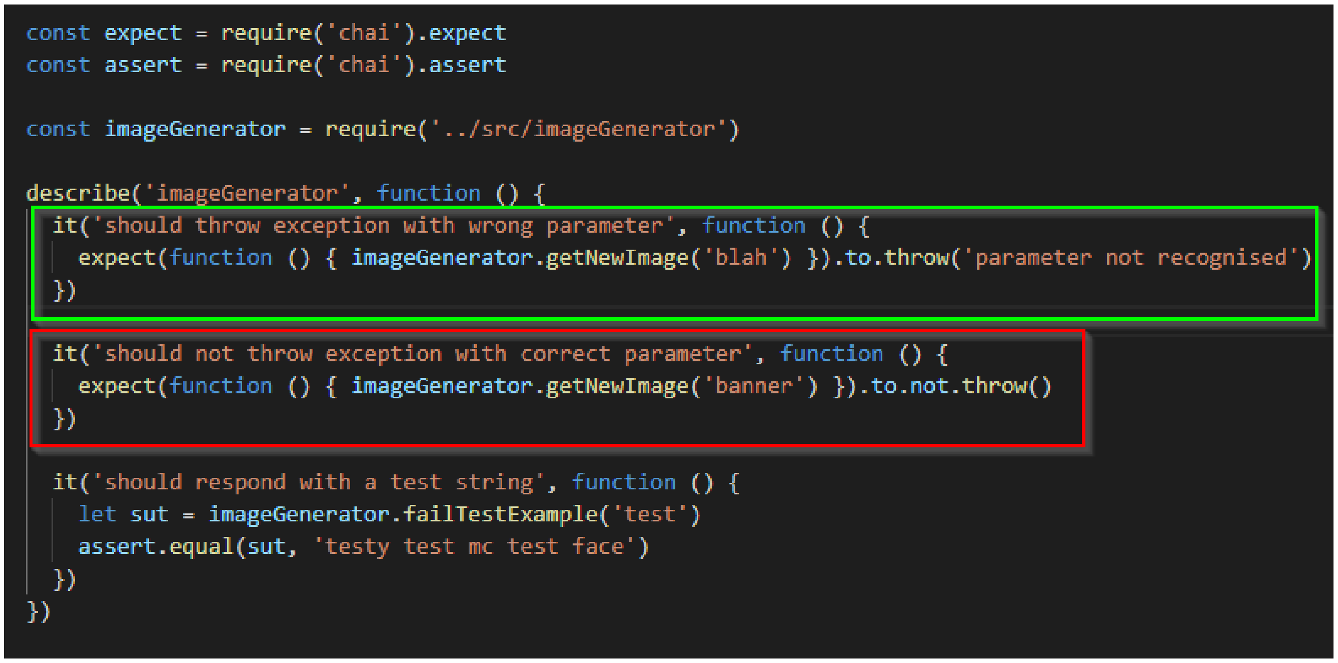
Nothing noteworthy. Test passed as expected.

3 Automated Unit Tests

3.1 TEST CODE

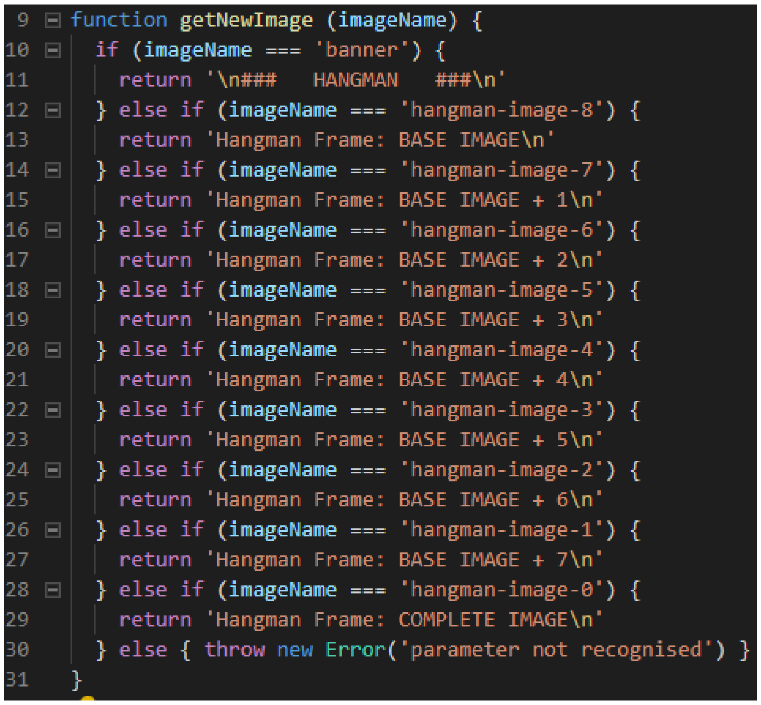
3.1.1 Method 1 – automated unit tests

TEST CODE



```
const expect = require('chai').expect
const assert = require('chai').assert

const imageGenerator = require('../src/imageGenerator')

describe('imageGenerator', function () {
  it('should throw exception with wrong parameter', function () {
    expect(function () { imageGenerator.getNewImage('blah') }).to.throw('parameter not recognised')
  })

  it('should not throw exception with correct parameter', function () {
    expect(function () { imageGenerator.getNewImage('banner') }).to.not.throw()
  })

  it('should respond with a test string', function () {
    let sut = imageGenerator.failTestExample('test')
    assert.equal(sut, 'testy test mc test face')
  })
})
```

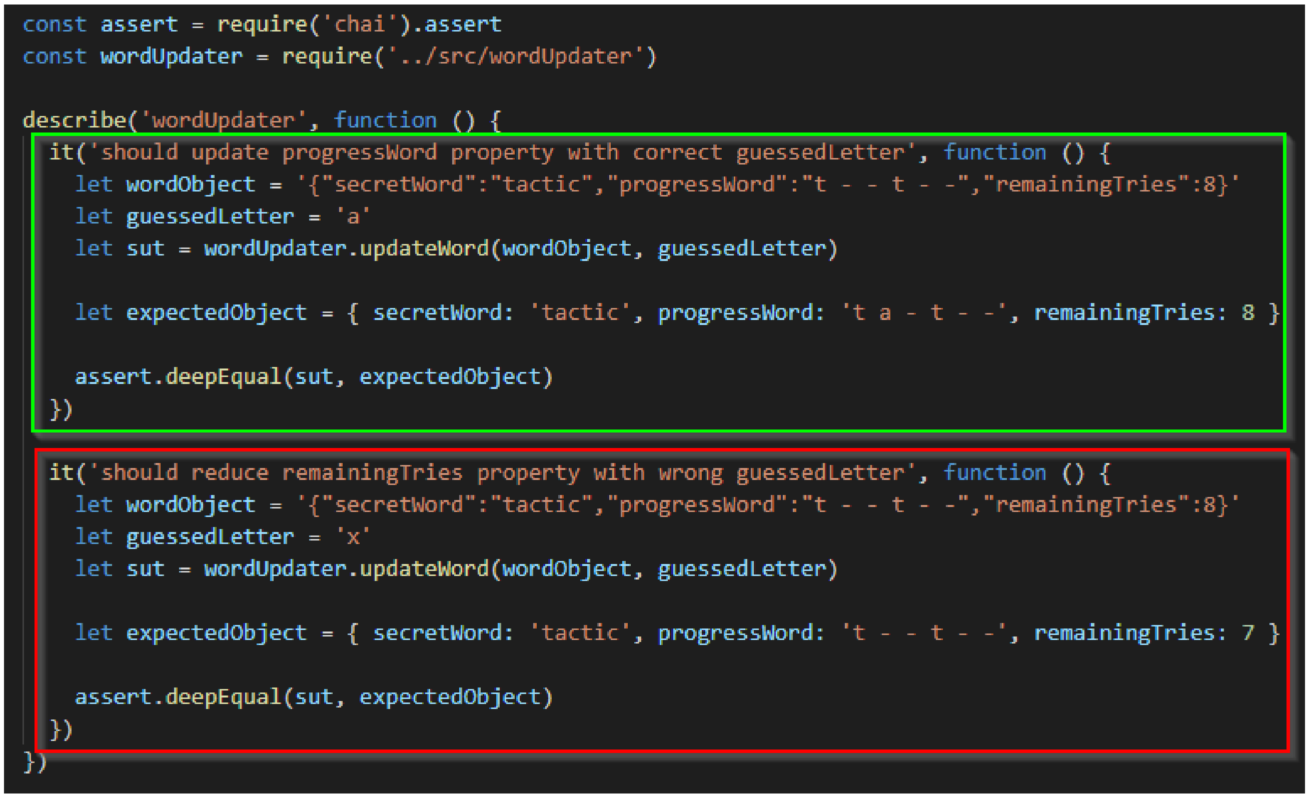
TESTED CODE



```
function getNewImage (imageName) {
  if (imageName === 'banner') {
    return '\n###    HANGMAN    ###\n'
  } else if (imageName === 'hangman-image-8') {
    return 'Hangman Frame: BASE IMAGE\n'
  } else if (imageName === 'hangman-image-7') {
    return 'Hangman Frame: BASE IMAGE + 1\n'
  } else if (imageName === 'hangman-image-6') {
    return 'Hangman Frame: BASE IMAGE + 2\n'
  } else if (imageName === 'hangman-image-5') {
    return 'Hangman Frame: BASE IMAGE + 3\n'
  } else if (imageName === 'hangman-image-4') {
    return 'Hangman Frame: BASE IMAGE + 4\n'
  } else if (imageName === 'hangman-image-3') {
    return 'Hangman Frame: BASE IMAGE + 5\n'
  } else if (imageName === 'hangman-image-2') {
    return 'Hangman Frame: BASE IMAGE + 6\n'
  } else if (imageName === 'hangman-image-1') {
    return 'Hangman Frame: BASE IMAGE + 7\n'
  } else if (imageName === 'hangman-image-0') {
    return 'Hangman Frame: COMPLETE IMAGE\n'
  } else { throw new Error('parameter not recognised') }
}
```

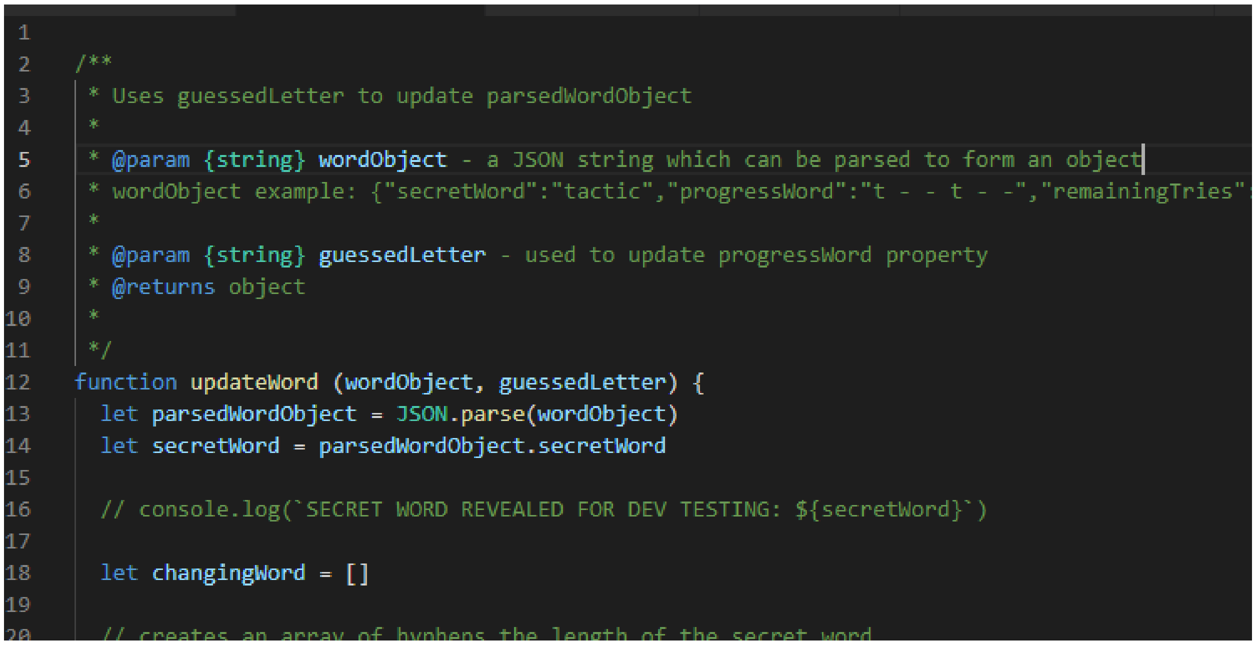
3.1.2 Method 2 – automated unit tests

TEST CODE



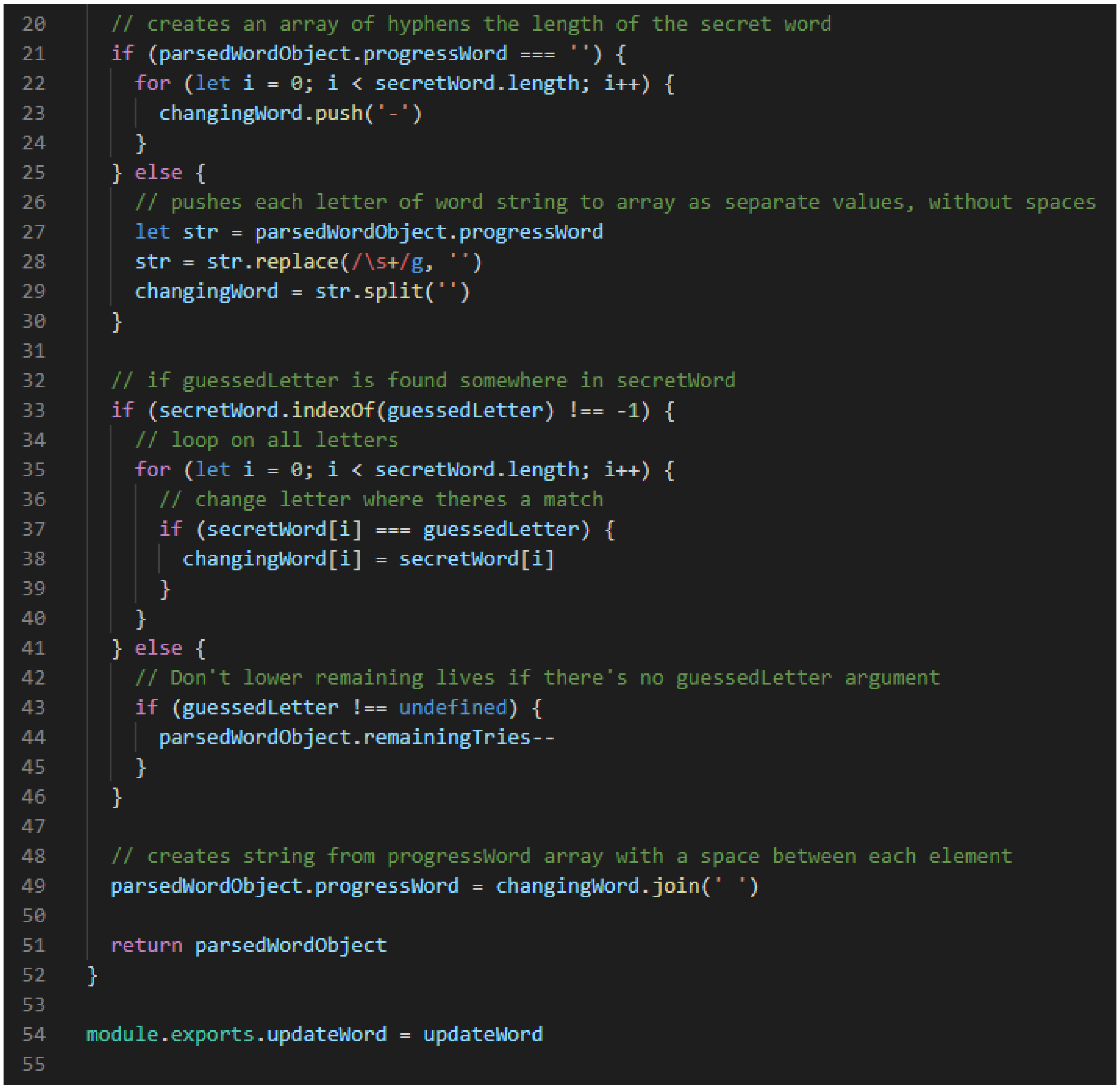
```
const assert = require('chai').assert
const wordUpdater = require('../src/wordUpdater')

describe('wordUpdater', function () {
  it('should update progressWord property with correct guessedLetter', function () {
    let wordObject = '{"secretWord":"tactic","progressWord":"t - - t - -","remainingTries":8}'
    let guessedLetter = 'a'
    let sut = wordUpdater.updateWord(wordObject, guessedLetter)

    let expectedObject = { secretWord: 'tactic', progressWord: 't a - t - -', remainingTries: 8 }

    assert.deepEqual(sut, expectedObject)
  })

  it('should reduce remainingTries property with wrong guessedLetter', function () {
    let wordObject = '{"secretWord":"tactic","progressWord":"t - - t - -","remainingTries":8}'
    let guessedLetter = 'x'
    let sut = wordUpdater.updateWord(wordObject, guessedLetter)

    let expectedObject = { secretWord: 'tactic', progressWord: 't - - t - -', remainingTries: 7 }

    assert.deepEqual(sut, expectedObject)
  })
})
```

TESTED CODE



```

/**
 * Uses guessedLetter to update parsedWordObject
 *
 * @param {string} wordObject - a JSON string which can be parsed to form an object
 * wordObject example: {"secretWord":"tactic","progressWord":"t - - t - -","remainingTries"
 *
 * @param {string} guessedLetter - used to update progressWord property
 * @returns object
 *
 */
function updateWord (wordObject, guessedLetter) {
  let parsedWordObject = JSON.parse(wordObject)
  let secretWord = parsedWordObject.secretWord

  // console.log(`SECRET WORD REVEALED FOR DEV TESTING: ${secretWord}`)

  let changingWord = []

  // creates an array of hyphens the length of the secret word
```

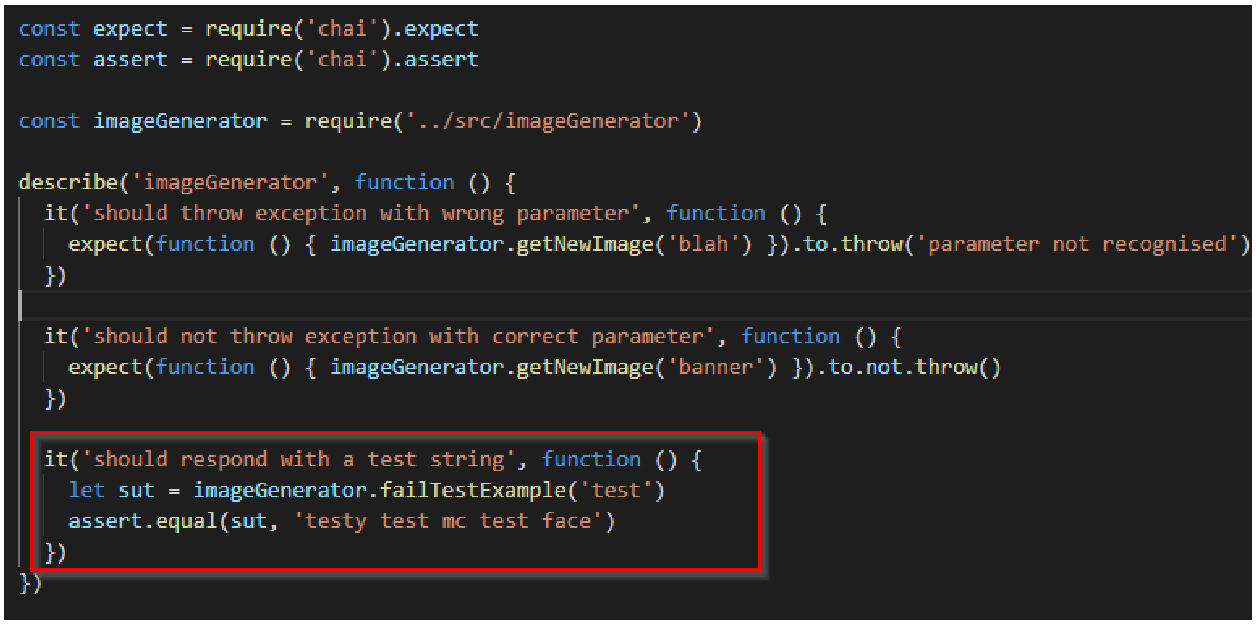
continues below…….



```
  // creates an array of hyphens the length of the secret word
  if (parsedWordObject.progressWord === '') {
    for (let i = 0; i < secretWord.length; i++) {
      changingWord.push('-')
    }
  } else {
    // pushes each letter of word string to array as separate values, without spaces
    let str = parsedWordObject.progressWord
    str = str.replace(/\s+/g, '')
    changingWord = str.split('')
  }

  // if guessedLetter is found somewhere in secretWord
  if (secretWord.indexOf(guessedLetter) !== -1) {
    // loop on all letters
    for (let i = 0; i < secretWord.length; i++) {
      // change letter where theres a match
      if (secretWord[i] === guessedLetter) {
        changingWord[i] = secretWord[i]
      }
    }
  } else {
    // Don't lower remaining lives if there's no guessedLetter argument
    if (guessedLetter !== undefined) {
      parsedWordObject.remainingTries--
    }
  }

  // creates string from progressWord array with a space between each element
  parsedWordObject.progressWord = changingWord.join(' ')

  return parsedWordObject
}

module.exports.updateWord = updateWord
```

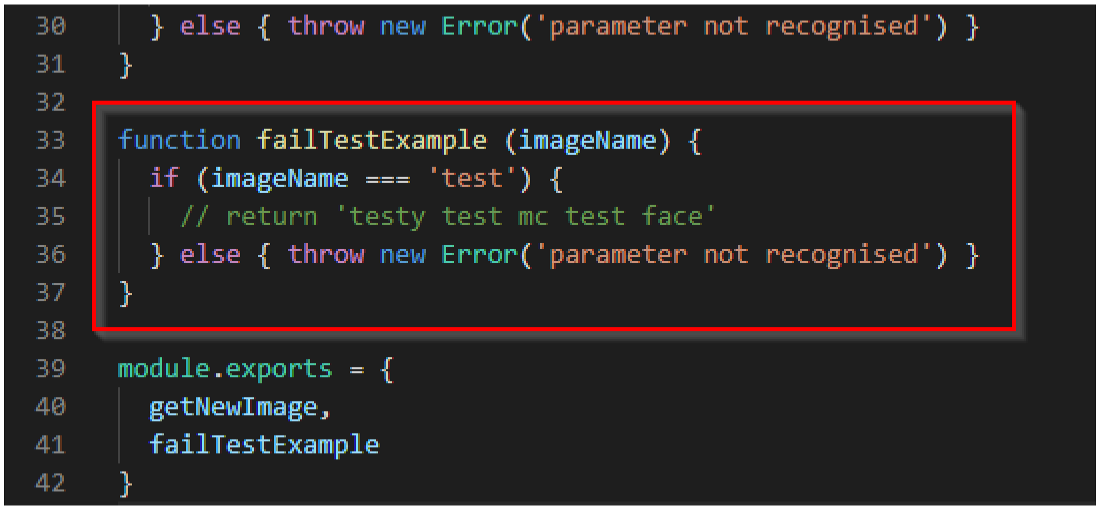
3.1.3 Failing test - automated unit test 3

TEST CODE



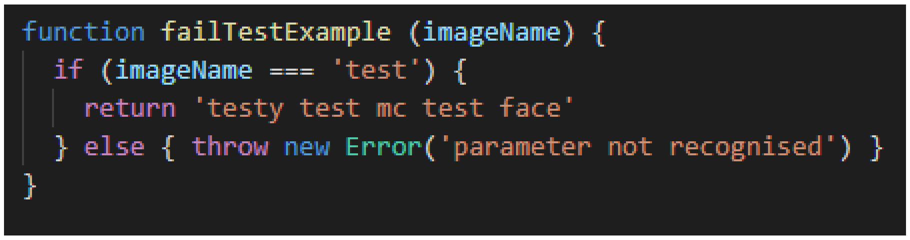
```
const expect = require('chai').expect
const assert = require('chai').assert

const imageGenerator = require('../src/imageGenerator')

describe('imageGenerator', function () {
  it('should throw exception with wrong parameter', function () {
    expect(function () { imageGenerator.getNewImage('blah') }).to.throw('parameter not recognised')
  })

  it('should not throw exception with correct parameter', function () {
    expect(function () { imageGenerator.getNewImage('banner') }).to.not.throw()
  })

  it('should respond with a test string', function () {
    let sut = imageGenerator.failTestExample('test')
    assert.equal(sut, 'testy test mc test face')
  })
})
```

TESTED CODE (FAILING)



```
  } else { throw new Error('parameter not recognised') }
}

function failTestExample (imageName) {
  if (imageName === 'test') {
    // return 'testy test mc test face'
  } else { throw new Error('parameter not recognised') }
}

module.exports = {
  getNewImage,
  failTestExample
}
```

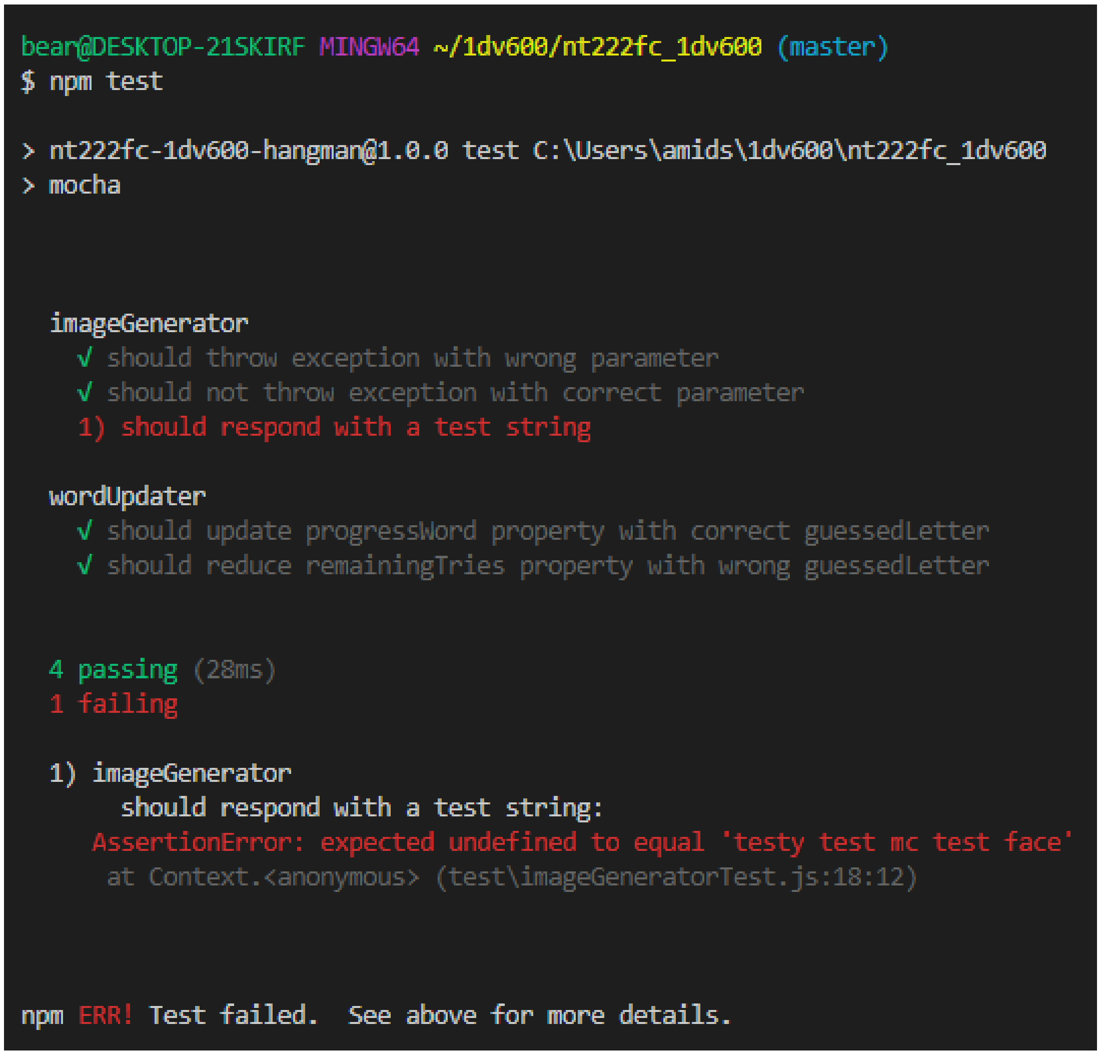
TESTED CODE (CORRECTED)



```
function failTestExample (imageName) {
  if (imageName === 'test') {
    return 'testy test mc test face'
  } else { throw new Error('parameter not recognised') }
}
```

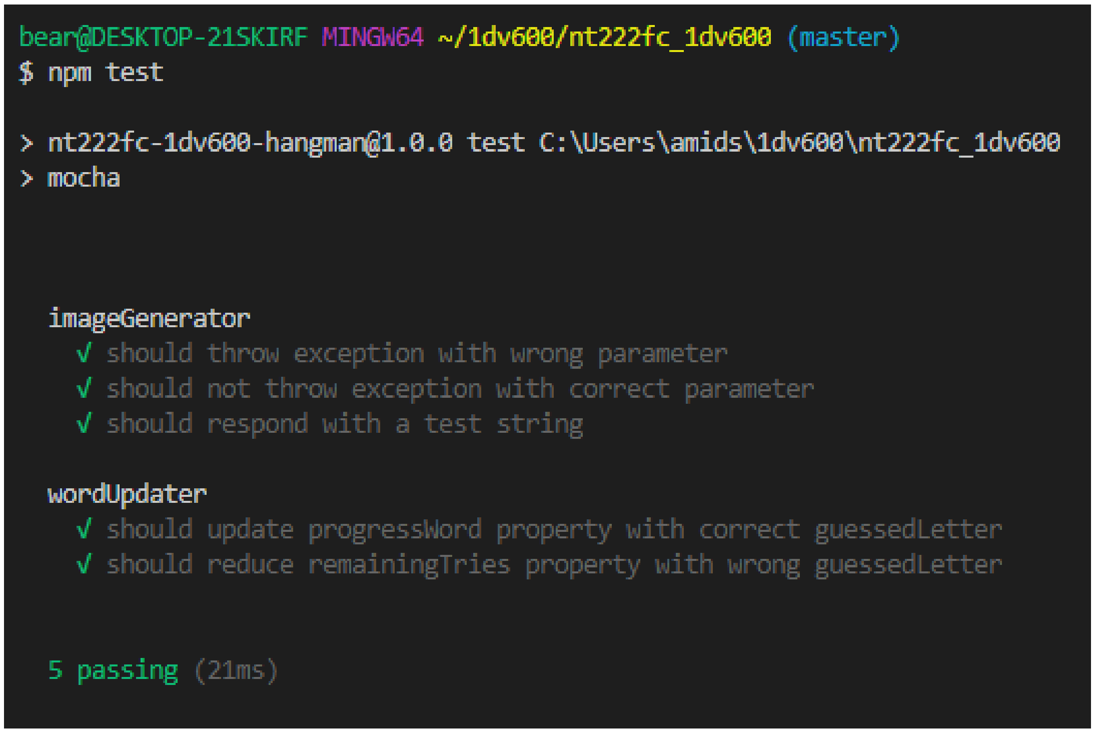
3.2 TEST RESULTS

3.2.1 Regular test results (with failing test)



```
bear@DESKTOP-21SKIRF MINGW64 ~/1dv600/nt222fc_1dv600 (master)
$ npm test

> nt222fc-1dv600-hangman@1.0.0 test C:\Users\amids\1dv600\nt222fc_1dv600
> mocha



  imageGenerator
    √ should throw exception with wrong parameter
    √ should not throw exception with correct parameter
    1) should respond with a test string

  wordUpdater
    √ should update progressWord property with correct guessedLetter
    √ should reduce remainingTries property with wrong guessedLetter


  4 passing (28ms)
  1 failing

  1) imageGenerator
       should respond with a test string:
     AssertionError: expected undefined to equal 'testy test mc test face'
      at Context.<anonymous> (test\imageGeneratorTest.js:18:12)



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

3.2.2 Test results with corrected fail test



```
bear@DESKTOP-21SKIRF MINGW64 ~/1dv600/nt222fc_1dv600 (master)
$ npm test

> nt222fc-1dv600-hangman@1.0.0 test C:\Users\amids\1dv600\nt222fc_1dv600
> mocha



  imageGenerator
    √ should throw exception with wrong parameter
    √ should not throw exception with correct parameter
    √ should respond with a test string

  wordUpdater
    √ should update progressWord property with correct guessedLetter
    √ should reduce remainingTries property with wrong guessedLetter


  5 passing (21ms)
```

4 Reflection

I had some difficulty learning how to use Mocha and Chai to test my application. Chai became necessary for me to test exceptions being throw in relation to what parameters where used in the test, and I was really confused for a while when using assert.equals to compare 2 input and output objects which failed, even though I was sure they were the same (apparently javascript doesn’t consider objects equal due to where they’re stored). That aside, I’ve found mocha and chai to be really useful and full of a wide range of fairly intuitive tests. I now realise that the code in my main playGame function is difficult to test because it is very large and also works with a lot of console.logs instead of returned values, and should therefore be broken up into more testable and understandable functions/modules.