Assignment 3

Testing

Thimmy Stenlund
ts222ub@student.lnu.se
https://github.com/Mrrshaw85/ts222ub_1dv600

Test Plan

Objective

The object is to test the code that was implement during assignment 2, the core game play, but also get a glimpse of possible future testing of the “Hang The Man” extra feature mode that will be implemented in iteration 4.

What to test and how

Our intent is to test Use Case 2 (“Play Game”), Use Case 3 (“Quit Game”) and Use Case 4 (“Play Hang The Man”) by writing and running dynamic manual test-cases, to begin with. We will also examine the code for Use Case 2 and for the unimplemented Hang The Man feature mode. We will review the code as well, to see where and how we could create unit tests to span all methods eventually.
IMPORTANT: The Test.js file content will be included in the end of this document.

(Screenshot included from the Use Model and Use Cases documentation, from Assignment 2)

UC 2 Play Game

Precondition: The game is running.

Postcondition: The standard game mode is active

Main scenario

1. Starts when the user chooses the “Play Standard Game” option in the menu.
2. A random word is selected from an array, which is not shown to the user.
3. For each letter in the word, an underline is displayed.
4. The user can guess letters, if it’s correct – the user can continue. If its wrong, the counter will increase by one.
5. If the user guesses all letters before the counter reaches 8, the user will see a “You Win!” being logged to the console, together with the time it took, and an option to go back to the menu. (see Use Case 1)
6. If the counter goes to 8, i.e. the user guesses wrong letter 8 times, the user will see a “You lose!” being logged to the console, with an option to go back to the menu. (see Use Case 1)

Alternative scenario

2.1 The Gamer wants to leave the game and go back to the main menu. (see Use Case 1)

UC 4 Start Hang The Man Game

Precondition: The game is running.

Postcondition: The game menu is logged

Main scenario

1. Starts when the user chooses the “Play Standard Game” option in the menu.
2. A random word is selected from an array, which is not shown to the user.
3. For each letter in the word, an underline is displayed.
4. The user can guess letters, if it’s correct – the user can continue. If it’s wrong, the counter will increase by one.
5. If the user guesses all letters before the counter reaches 8, the user will see a “You Win! The Man is Hanged” being logged to the console, together with the time it took, and an option to go back to the menu. (see Use Case 1)
6. If the counter goes to 8, i.e. the user guesses wrong letter 8 times, the user will see a “You lose! The Man is Free” being logged to the console, with an option to go back to the menu. (see Use Case 1)

Note: The difference between the game mode “Standard” and “Hang The Man” is purely graphical. In standard, you are aiming for not hanging the man. In “Hang The Man” you are struggling to actually hang the man.

UC 3 Quit Game

Precondition: The game is running.

Postcondition: The game is terminated.

Main scenario

1. Starts when the user wants to quit the game.
2. The system prompts for confirmation.
3. The user confirms.
4. The system terminates.

Alternative scenarios

3.1. The user does not confirm

1. The system returns to its previous state

How the testing will be done

UC2 , UC3 and UC4 will be dynamically tested through Unit Tests, covering different sections of the UC's across several tests and there will also be manual tests done on at least two of the User Cases.

Time plan

Task	Estimated	Actual
Manual TC 1.1	30m	20m
Manual TC 1.2	30m	15m
Unit Test 1.1	30m	10m
Unit Test 1.2	30m	30m
Unit Test 2.1	30m	45m
Unit Test 2.2	30m	30m
Unit Test 2.3	30m	30m
Running Manual Tests	15m	15m
Running Unit Tests	30m	60m

Comment on Time Plan: Mostly accurate. Time exceeding estimating was mostly due to refactorizing the whole game to have it fit a somewhat decent Test Strategy and Test Execution. Hope to get more tuned in to actual versus estimated.

Manual Test Cases

TC1.1 The player will choose an allowed, not previously used letter that is found in the word.

The Hangman application will ask the user for a letter. In this TC the user will select a non-reused letter that can be found in the hidden word itself.

Precondition: The game must be running. The hidden word must be pre-determined ("FOX").

UC: 2

Test Steps:
(* Start the application) - Precondition
(* Play a standard Game) - Precondition
* The pre-defined word will be "FOX"
* The user will try to guess the letter "O" by pressing the "O" button and follow that up with Enter key.
* The application will display the letter "O" in the centre of the word and let the user guess again.

☒ Test Succeeded

Comments: No irregularities detected. Test Steps was followed and the Test itself returned the expected result.

Screenshot snippet of the Letter handling in the StandardGame.js and the console.

```
26      let currLetter = readlineSync.question('Guess a letter (type "Quit" to quit, "Main" to go back to Main Menu): ')
27      currLetter = currLetter.toLowerCase()
28      notFound = true
29      if (currLetter.length !== 1 && currLetter !== 'quit' && currLetter !== 'main') {
30        console.log('Please enter a single letter! (type "Quit" to quit, "Main" to go back to Main Menu)')
31      } else if (currLetter === 'quit') {
32        quit = Quit()
33      } else if (currLetter === 'main') {
34        console.log('')
35        clearGameState()
36        return true
37      } else {
38        for (let j = 0; j < word.length; j++) {
39          if (word[j] === currLetter && answerArr[j] !== currLetter) {
40            answerArr[j] = currLetter
41            remainingLetters--
42            notFound = false
43          }
44        }
45        if (notFound && currLetter !== 'quit' && currLetter !== 'main') {
46          guessCount++
47          console.log(`Letter ${currLetter} not found.`)
48          console.log(`You have ${(maxGuesses - guessCount)} guesses left!`)
49        }
50      }
```

```
> ts222ub_1dv600@1.0.0 start C:\Users\Kong\Documents\Högskola\Webbprogrammering LNU\1dv600\ts222ub_1dv600
> node app.js

HangTheMan v1.0
--------MENU-------
1: Play Standard Game
2: Play HangTheMan
3: Quit
------------------
Option: 1

_ _ _
Guess a letter (type "Quit" to quit, "Main" to go back to Main Menu): o
_ o _
Guess a letter (type "Quit" to quit, "Main" to go back to Main Menu):
```

TC2.1 The user decides to quit the application in the menu

The menu will ask for an option that the user will choose. The user will select 3 for Quit and confirm that he wants to quit the application.

UC: 3

Precondition: The application must be running. The user must be situated in the main menu.

Test Steps:
(* Start the application) - Precondition
* The user will select “3”
* The application will display a question “Are you sure you want to quit? (Yes or No): “
* The user will type “Yes” and press the Enter key.
* The application will stop running.

☒ Test Succeeded

Comments: No irregularities detected. Test Steps was followed as per above definition and the Test itself returned the expected result.

Screenshot of Quit.js and the Manual Testing Session.

```
'use standard'

const readlineSync = require('readline-sync')

function quitPrompt () {
  let yesOrNo = readlineSync.question('Are you sure you want to quit? (Yes or No): ')
  yesOrNo = yesOrNo.toUpperCase()
  if (yesOrNo === 'YES') {
    return true
  } else if (yesOrNo === 'NO') {
    console.log('')
    return false
  } else {
    quitPrompt()
  }
}

module.exports = quitPrompt
```

```
> ts222ub_1dv600@1.0.0 start C:\Users\Kong\Documents\Högskola\Webbprogrammering LNU\1dv600\ts222ub_1dv600
> node app.js

HangTheMan v1.0
--------MENU-------
1: Play Standard Game
2: Play HangTheMan
3: Quit
------------------
Option: 3
Are you sure you want to quit? (Yes or No): Yes
PS C:\Users\Kong\Documents\Högskola\Webbprogrammering LNU\1dv600\ts222ub_1dv600>
```

Unit Tests (Automatic Tests)

Unit Test 1.1: For each letter in a "random" word – create a underscore that will be presented to the User

StandardGame (UC2) Testing - #createUnderscoreArr: Create Empty Array with _ for each char

When the User opt to play a standard game, the application will display a number of underscores (_) connected to the number of characters in the random selected word.

UC: 2 (Play Standard Game)

Precondition: The application must be running. A preselected word will be used - "Stormcloud".

Test Steps:
(* Start the application) – Precondition
* The user selects "1" (Play Game)
* The Game will display a total of 10 underscores, each representing a character of the "random" (preselected in the Test) word

☒ Test Succeeded

Comments: No irregularities detected. Test Steps was followed as per above definition and the Test itself returned the expected result.

Screenshot of the createUnderscoreArr function from the StandardGame.js module.

```
function createUnderscoreArr (w) {
  let aArr = []
  for (let i = 0; i < w.length; i++) {
    aArr[i] = '_'
  }
  return aArr
}
```

```
> ts222ub_1dv600@1.0.0 test C:\Users\Kong\Documents\Högskola\Webbprogrammering LNU\1dv600\ts222ub_1dv600
> mocha



  StandardGame (UC2) Testing
    #createUnderscoreArr: Create Empty Array with _ for each char
      √ should return ('_','_','_','_','_','_','_','_','_','_')
```

Unit Test 1.2: When there is time to play a new game or when game is lost, the game need to clear the values and return true (so that the program might now that the game has been cleared)

StandardGame (UC2) Testing - #clearGameState: Clear the values and return true

When the user either returns to main menu, quits the game or wins – the game will clear the game state (as in – put the counter of moves back to zero and return true

UC: 2 (Play Standard Game)

Precondition: The application must be running. We assume that the user chose to go to Main.

Test Steps:
(* Start the application) – Precondition
* The User opts to go back to main

☒ Test Succeeded

Comments: No irregularities detected. Test Steps was followed as per above definition and the Test itself returned the expected result.

Screenshot of the clearGameState function and the test case:

```
function clearGameState () {
  answerArr = []
  guessCount = 0
  return true
}
```

```
  describe('#clearGameState: Clear the values and return true ', function () {
    it('should return true ', function () {
      let shouldBeTrue = false
      shouldBeTrue = stdGame.clearGameState()
      assert.deepEqual(shouldBeTrue, true)
    })
  })
})
```

Unit Test 2.1

Play Hang the Man (UC4) Testing '#manGoesFree: console.logs that the man goes free and returns false

If the user either runs out of guesses or if the (not implemented) checkFinalAnswer function ends with a faulty answer to the last question, this function is run.

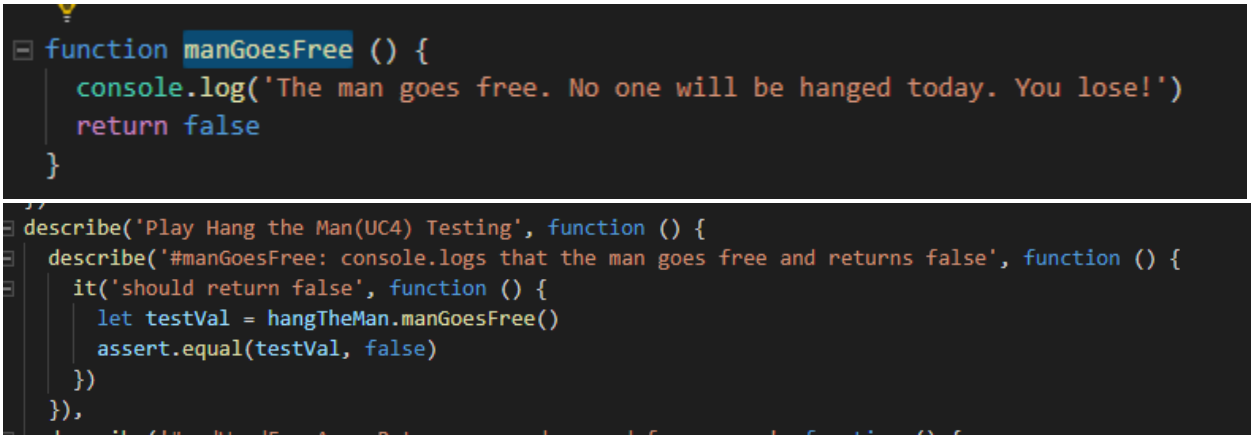
UC: 4 (Play Hang the Man)

Precondition: The game mode “Hang the Man” must be running.
Test Steps:
(* Start the application) – Precondition
(* Start Hang the Man) - Precondition
* The user runs out of guesses
* The manGoesFree function is activated, send a console.log and returns false.

☒ Test Succeeded

Comments: No irregularities detected. Test Steps was followed as per above definition and the Test itself returned the expected result.

Screenshot of the manGoesFree function and the test case:



```
function manGoesFree () {
  console.log('The man goes free. No one will be hanged today. You lose!')
  return false
}
```

```
describe('Play Hang the Man(UC4) Testing', function () {
  describe('#manGoesFree: console.logs that the man goes free and returns false', function () {
    it('should return false', function () {
      let testVal = hangTheMan.manGoesFree()
      assert.equal(testVal, false)
    })
  }),
```

Unit Test 2.2

Play Hang the Man (UC4) Testing '#rndWordFromArr: Returns a random word from an array of words

When the User starts Hang the Man, there is a random word that is selected from an array from a function outside of the normal “Play Game” function.

UC: 4 (Play Hang the Man)

Precondition: The application is started.

Test Steps:
(* Start the application) – Precondition
* Start Hang the Man
* The function selects a random word from an array (predetermined to only contain the word ‘fox’)
* The function returns the word

☒ Test Succeeded

Comments: No irregularities detected. Test Steps was followed as per above definition and the Test itself returned the expected result.

Screenshot of the rndWordFromArr function and the test case:

```
function rndWordFromArr (wArr) {
  return wArr[Math.floor(Math.random() * wArr.length)]
}
```

```
30    describe('#rndWordFromArr: Returns a random word from array', function () {
31      it('should return "fox"', function () {
32        let wArr = ['fox']
33        let testRndW = hangTheMan.rndWordFromArr(wArr)
34        assert.equal(testRndW, 'fox')
35      })
36    }),
```

Unit Test 2.3

Play Hang the Man (UC4) Testing '#checkFinalAnswer: Should return true if the input to the function is the string “true”

When the User manage to find the hidden word in Hang the Man, there is a question that needs to be answered. For the purpose of the test (and that it’s currently unimplemented/not finished), the test will fail as the function is set to only return false.

UC: 4 (Play Hang the Man)

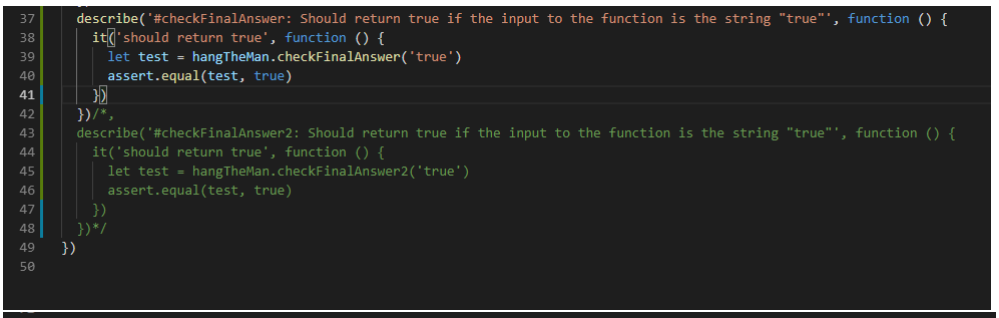
Precondition: The application is started. Hang the Man game is running.

Test Steps:
(* Start the application) – Precondition
(* Start Hang the Man) – Precondition
* The user finds out the correct word
* A final question is shown that the user has to answer true or false.
* If the user answers “true”, the function should return the Boolean true.

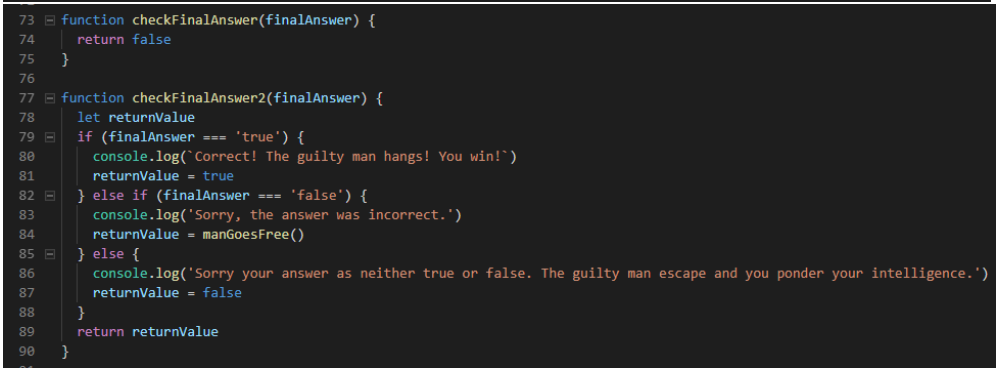
☐ Test Succeeded

Comments: Test fails. The function is not yet fully implemented. A beta version is shown below in screenshots (called checkFinalAnswer2) that works. Might be tweaked for iteration 4. The passing test has been commented out and the failing test is left in the test file.

Screenshot of the checkFinalAnswer function and the test case:



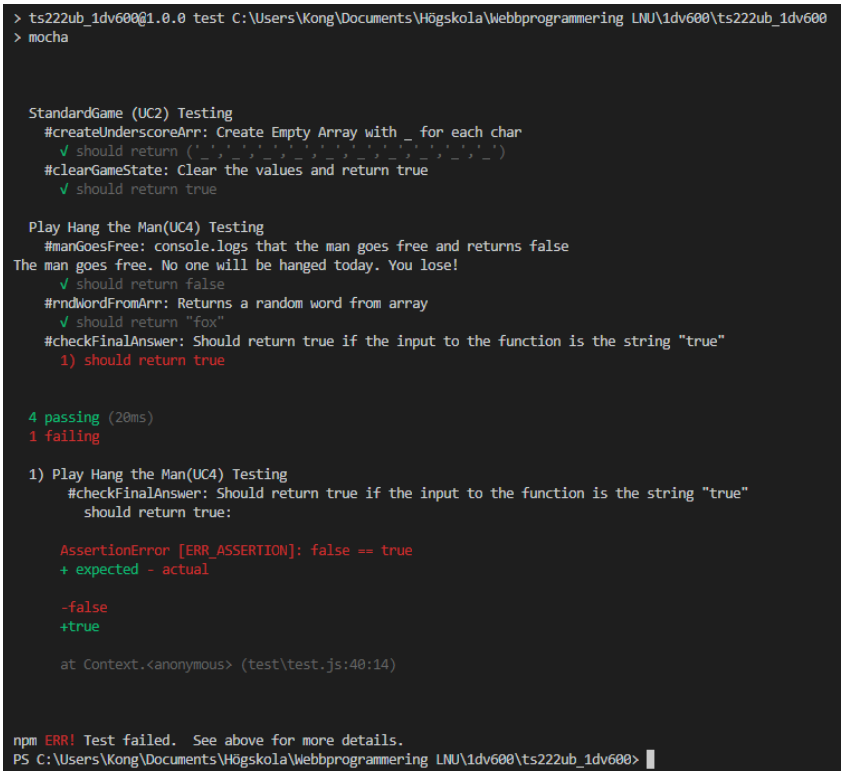
```
describe('#checkFinalAnswer: Should return true if the input to the function is the string "true"', function () {
  it('should return true', function () {
    let test = hangTheMan.checkFinalAnswer('true')
    assert.equal(test, true)
  })
})/*,
describe('#checkFinalAnswer2: Should return true if the input to the function is the string "true"', function () {
  it('should return true', function () {
    let test = hangTheMan.checkFinalAnswer2('true')
    assert.equal(test, true)
  })
})*/
})
```



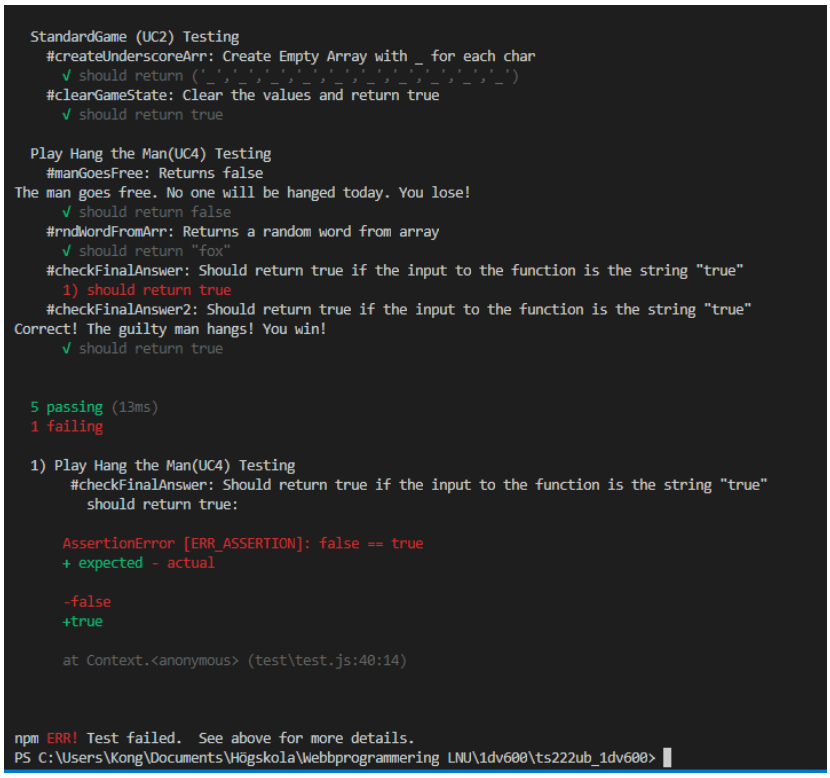
```
function checkFinalAnswer(finalAnswer) {
  return false
}

function checkFinalAnswer2(finalAnswer) {
  let returnValue
  if (finalAnswer === 'true') {
    console.log(`Correct! The guilty man hangs! You win!`)
    returnValue = true
  } else if (finalAnswer === 'false') {
    console.log('Sorry, the answer was incorrect.')
    returnValue = manGoesFree()
  } else {
    console.log('Sorry your answer as neither true or false. The guilty man escape and you ponder your intelligence.')
    returnValue = false
  }
  return returnValue
}
```

The whole test suite (excluding the passing checkFinalAnswer2 test)



The whole test suite (including the checkFinalAnswer2 test)



Reflection:

I really enjoy the Testing side of coding. There is some things in this assignment that makes it a bit hard to do good tests on. If the first assignment (assignment 1) had more structure – as in demands on how the game should look like, what it should contain, that it should be structured and divided (heavily) – I think the end result would be better. I do like working with Mocha (I have not used Mocha at work, we use mainly Katalon for our UI testing, which is my “speciality”, that we are hooking up to a CI/TA pipeline. I will surely work more with Mocha on my own little JavaScript projects from now on (for instance at our upcoming “Own Project”). Looking forward for iteration 4 and tie it all together.