

Assignment 4

Gabriel Björlin

What will be done?

This is the last and final assignment for this course which means that the project has to be finished. I added the last functionalities in the last assignment which means that I had to come up with one more functionality. I figured it would be nice to be able to pick difficulty. This means that the words that will be picked are longer which makes them harder to guess.

It was recommended to have this document in the first assignment file. But I think it will be too messy then so I made a separate one instead since it was an option. This means that some needed information is in that document, and some of the other files.

Since most of the planning was done upfront it will not be a lot of planning done here, but you can find some necessary planning information below.

This is such a simple functionality so the information might seem rather vague.

Risk analysis

The risks for this last functionality in the hangman project is not very large. There is not much that can happen since everything has a backup. If I for some reason can not make this functionality work the game would still work fine.

List of risks

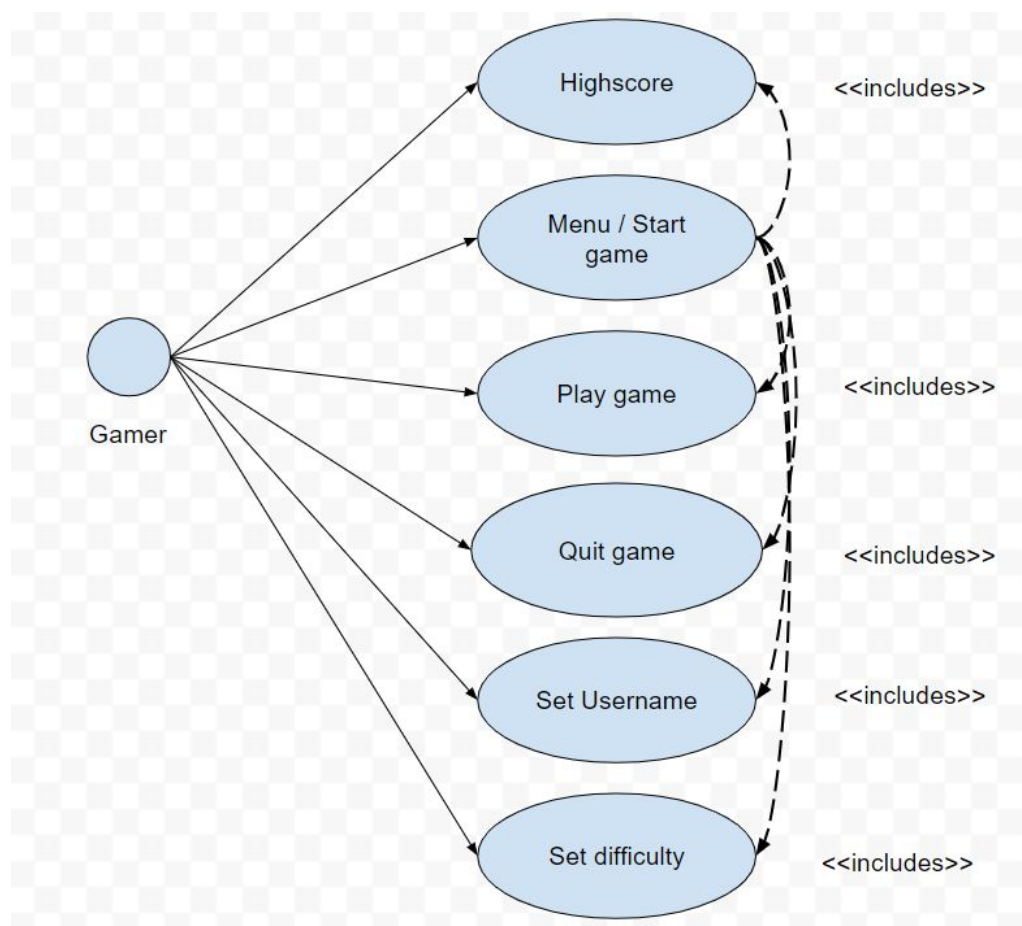
- I can not make the function work.
- Problem with my computer when handing in
- out of time
- Getting sick

Strategy

My plan is to use contingency plan just like I did for the entire project. This means that I will take care of the problem if it occurs.

Since the risk of problems is fairly low I will not think about this part too much. It is only one function and it should not be too hard to implement.

Note: Most relevant information about this iteration will be found on the first plan (iteration 1).



UC 6 Difficulty

Precondition: The game menu is shown

postcondition: The difficulty screen is shown.

Main scenario

1. Starts when gamer wants to change difficulty.
2. The system asks the gamer to set a difficulty.
3. The gamer types in the difficulty.
4. The system shows the new difficulty.
5. The system puts the gamer back to main menu.

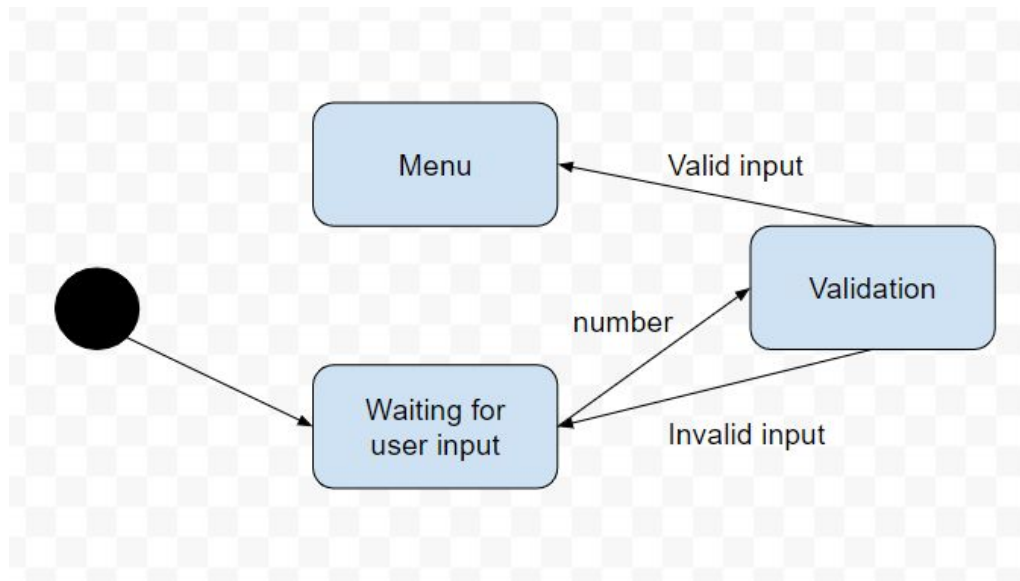
Alternative scenario

- 3.1. The gamer makes an invalid input. And point 2 repeats.

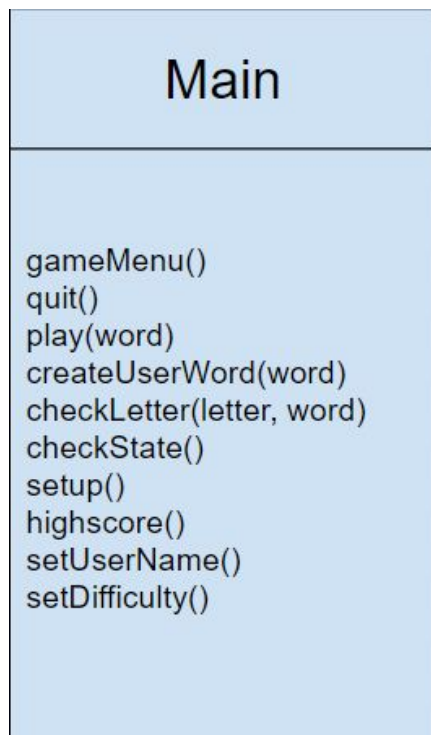
Note: The other use cases can be found on assignment 2 (on github).

State machine

State machine for set difficulty



Class diagram



Test plan

- What are the objectives of the testing in this iteration?
 - To make sure that the new feature is working as expected, since the prior tests tested the whole game (pretty much) it should be enough to test only the new function.

- What to test?

I am going to test the new function which is to set a difficulty. I am going to make sure that the function is shifting to a new set of words. I will have one manual test case to test that new words will be generated. I will also use a unit test that will test if the parameter difficulty is changing.

- Manual test will see if different kind of words will be generated.
- The first unit test will test if the internal parameter "difficulty" is being changed
- The last unit test will test if the length of words in each difficulty have different lengths. difficulty 1 should have 4 - 5 in length, difficulty 2 should have 6 - 7 in length and difficulty 3 should have 10 in length

Manual Test Case

TC1.1

Use case: Difficulty (uc 6)

Scenario: Gamer set the difficulty

This is going to test the set difficulty function, the gamer is supposed to set a difficulty depending on how hard he wants the word to be.

In order to perform this test you need to be inside this function.

You will be directed here by typing "difficulty" when you are in the main menu (main menu will be displayed when you start the application)

```
-----  
-----  "play"   to start game  -----  
-----  "quit"   to to quit   -----  
-----  "name"   to set username -----  
-----  "highscore" to see hihscore -----  
-----  "difficulty" to set difficulty -----  
-----  
-----  HANGMAN MENU  -----  
-----  
Input:
```

Preconditions:

- cd into the src, (for me: cd gb222hq_1dv600/src) with your cmd
- Type in "node mainManualTest.js" in cmd.
- Type in "difficulty" to get into this test.

The test steps that need to be performed is:

1. Type in "2" and press then enter
2. Type in "play" and press enter (will run the game)
3. Type in the letter "s" and then press enter
4. Then repeat the previous step but with "p", "y", "i", "n", "g"
5. when you press enter on "g", the text "You won!" will be displayed and you will get back to the main menu.

We are also going to test another difficulty in order to get a valid test. This test will be performed right after the previous steps.

1. Type in "difficulty"
2. Type in "3" and then press enter
3. Type in "play" and press enter (will run the game)
4. Type in the letter "a" and then press enter
5. Then repeat the previous step but with "l", "g", "o", "r", "i", "t", "h", "m", "s"
6. when you press enter on "s", the test "You Won!" will be displayed and you will get back to the main menu.

Here you can see pictures how it should look like:

1.

```
Input: difficulty
-----
"1", "2" or "3" to set difficulty
"1" is default
-----
input:
```

2.

```
-----
"play" to start game
"quit" to to quit
"name" to set username
"highscore" to see hihscore
"difficulty" to set difficulty
-----
HANGMAN MENU
-----
Input: play|
```

3.

```
-----
your username: Guest123
life: 7/7
"menu" to get back to menu
Your wrong letters:
Your word: [ '_', '_', '_', '_', '_' ]
-----
input: s

-----
your username: Guest123
life: 7/7
"menu" to get back to menu
Your wrong letters:
Your word: [ 's', '_', '_', '_' ]
-----
input: p

-----
your username: Guest123
life: 7/7
"menu" to get back to menu
Your wrong letters:
Your word: [ 's', 'p', '_', '_' ]
-----
input: y

-----
your username: Guest123
life: 7/7
"menu" to get back to menu
Your wrong letters:
Your word: [ 's', 'p', 'y', '_', '_' ]
-----
input: |
```


4.

```
input: g
You won!

-----
-----  "play"  to start game  -----
-----  "quit"  to to quit  -----
-----  "name"  to set username  -----
-----  "highscore"  to see hihscore
-----  "difficulty"  to set difficulty
-----
-----  HANGMAN MENU  -----
-----
Input:
```

X	First word correct
X	Second word correct
X	Won when completed the word
	Comments:

Unit tests

First unit test:

The first test will test if the difficulty parameter will be changed when we run the function to set difficulty.

```
set Difficulty
  ✓ Should change difficult value to 1
  ✓ should change difficulty value to 2
  ✓ should change difficulty value to 3
```

The code for this test:

```
describe('set Difficulty', () => {
  // should change value of difficulty to 1
  it ('Should change difficult value to 1', () => {
    let difficulty = app.readDifficulty('1')
    assert.equal(difficulty, 1)
  })
  // should change value of difficulty to 2
  it ('should change difficulty value to 2', () => {
    let difficulty = app.readDifficulty('2')
    assert.equal(difficulty, 2)
  })
  // should change value of difficulty to 3
  it ('should change difficulty value to 3', () => {
    let difficulty = app.readDifficulty('3')
    assert.equal(difficulty, 3)
  })
})
```

Second unit test:

The second and last test will test if the different difficulties give the right length of words.

```
set Difficulty
  ✓ difficulty 1 should have length 4 or 5
  ✓ difficulty 2 should have length 6 or 7
  ✓ difficulty 3 should have length 10
```

The code for this test:

```
describe('set Difficulty', () => {
  // the length of difficulty 1 should be 4 or 5
  it('difficulty 1 should have length 4 or 5', () => {
    let word = app.setup(1)
    if (word.length === 4) {
      assert.equal(word.length, 4)
    } else {
      assert.equal(word.length, 5)
    }
  })
  // the length of difficulty 2 should be 6 or 7
  it('difficulty 2 should have length 6 or 7', () => {
    let word = app.setup(2)
    if (word.length === 6) {
      assert.equal(word.length, 6)
    } else {
      assert.equal(word.length, 7)
    }
  })
  // the length of difficulty 3 should be 10
  it('difficulty 3 should have length 10', () => {
    let word = app.setup(3)
    assert.equal(word.length, 10)
  })
})
```

Reflection

In this iteration I only tested one function (set difficulty). This is not an important function but can make the game more fun. All the tests did succeed and the manual test seems to be working. The manual test could be good enough for testing this function and the unit tests was not really needed since they are testing the same basic thing. This could also be said about Manula test, that it was not needed. Overall I think it was a good test that is making sure the right words will be generated.

Time log

This time log is not very precise on each task but it would be very time consuming to report and clock each minor subtask. That's why I made it pretty raw. If any of the tasks would have taken a very long time it could have been good to report that task

Task	Expected	Actual
Planning and writing	90 min	60 min
Uml diagrams	45 min	30 min
Documentation for uml	90 min	60 min
Coding	45 min	75 min
Writing tests	60 min	45 min
Documentation for tests	90 min	45 min
Other	60 min	90 min

Overall it seems like my estimations were a bit high. I think this was due to the knowledge I now have. I knew I would complete this much faster this time than the last time but I was a bit quicker than I thought.

The only ones who were faster were "other" and coding. The coding part was a bit more complex than I thought in order to not create bugs etc. But it was not a major problem. Other is things around the project like reading and making sure everything is in order.

More documentation

Downbelow you can find the documentation for the other parts of the project.

The first iteration (planning) can be found on the first iteration on this course. I will not implement that document in this file since I need to update and make changes in that one and I will not be able to finish that in time. However all the other tasks will be downbelow.

Note: some information might be missing due to the changes above. (the changes is only about difficulty)

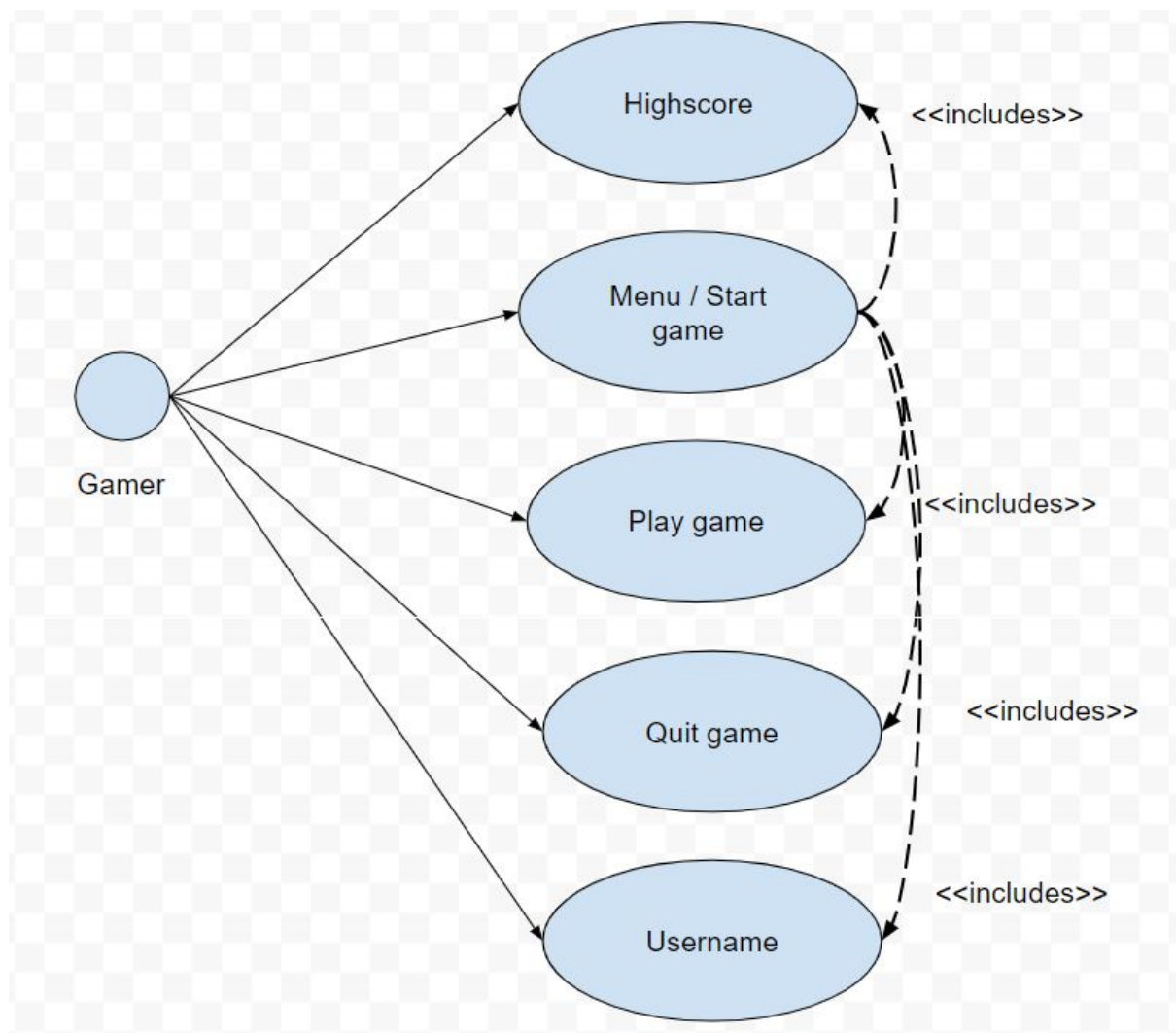
Timelog:

expected:

- reading instructions: 45 min.
- Making timelog: 10 min.
- Case diagram: 1h 30 min.
- State Machine Diagram for "Play Game": 1h
- Implementation: 12h
- Class diagram: 1h

Actual:

- reading instructions: 60 min
- making timelog: 7 min
- Case diagram, mostly confusion: 1h 20 min
- State Machine Diagram for "Play Game": 1h 10 min
- Implementation: 9h
- Class diagram: 30 min
- Things around like research etc depending on how it counts: 3 - 20h



UC 1 Start Game

Precondition: none.

Postcondition: the game menu is shown.

Main scenario

1. Starts when the user wants to begin a session of the hangman game.
2. The system presents the main menu with a title, the option to play and quit the game. Aswell change username and see highscore
3. The Gamer makes the choice to start the game.
4. The system starts the game (see Use Case 2).

Repeat from step 2

Alternative scenarios

3.1 The Gamer makes the choice to quit the game.

1. The system quits the game (see Use Case 2)

4.1 Invalid menu choice

1. The system presents an error message.
2. Goto 2

UC 2 Play Game

Precondition: The game menu is shown.

Postcondition: The game is running.

Main scenario

1. The gamer will see information about the word, tries and wrong letters.
2. The gamer gives an input (letter as main scenario)
3. The system gives feedback, spelling out wrong or adds the letter to the word.
4. This process repeats

Alternative scen

2.1 The gamer puts in menu and will be directed back to main menu.

2.2 Invalid input and a error is displayed

4.1 The gamers wins and gets directed back to main menu

4.2 The gamer loses and gets directed back to main menu

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 scen

- 3.1. The user does not confirm
1. The system returns to its previous state

UC 4 Highscore

Precondition: The game menu is shown.

Postcondition: The highscore is shown.

Main scenario

1. Start when the gamer wants to see the highscore.
2. The system shows the top 5 best scores.
3. The gamer types in Menu and gets back to the main menu

UC 5 Username

Precondition: The game menu is shown.

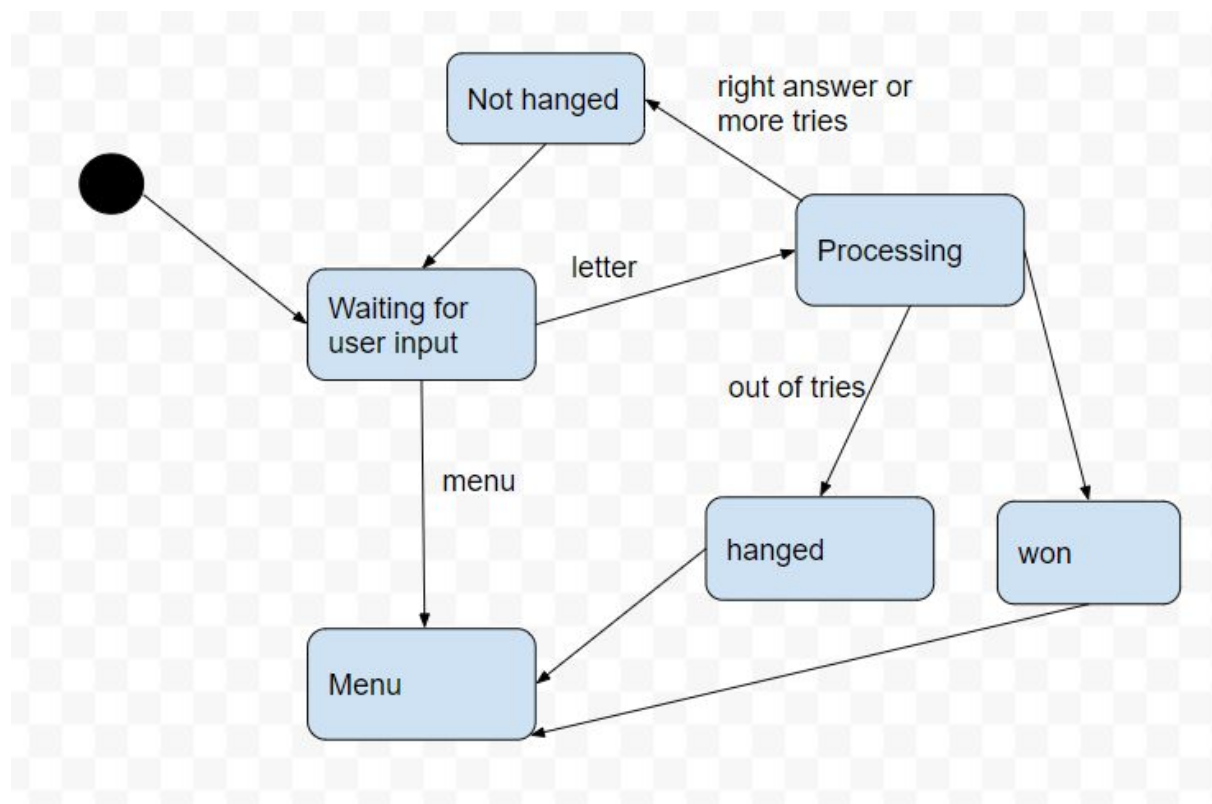
Postcondition: The username menu is displayed.

Main scenario

1. Start when the gamer wants to set or change username.
2. The System ask the gamer to set a username.
3. The gamer types in the username.
4. The Systems displays the new username and the gamer gets directed back to the main menu.

Alternative scen

1. The gamer type in "menu" and gets back to the main menu without changing name.



Class diagram

Not sure if this is the best way to make it but since I only have one class currently I believe this is the way to make it.

Time plan

Estimated:

install testing tools and reading documentation: 1h 30 min

writing test plan: 1h

Manual test 1: 1h

manual test 2: 1h

unit test 1: 1.5h

unit test 2: 1.5h

unit test 3: 1.5h

Actual:

Install testing tools and reading documentation: 2h

writing test plan: 45 min

manual test 1: 3h (misunderstood and made it very complectade and then redid.)

manual test 2: 30 min

unit test 1: 1.5h

unit test 2: 1h

unit test 3: 1h

Other: 3h

Test plan

- What are the objectives of the testing in this iteration?
 - To make sure the application is working as expected and that there is no bugs etc. We are going to make manual testing and automated unit testing.
- What to test? Include a short rationale for why you choose to test these objects and not others. (Look in Task 2, 3 for this...)
 - In the manual test I am going to test the main function (play game). I picked this since it is the most important function in this application. Without this it is not a playable game
 - The second manual test will be the quit function. I picked this since this is also a pretty important part. We need to be able to quit the game without shutting down the entire console.
 - The first unit test is going to be the visual word. This is a function that will create a copy of the word with blank likes to show to length and picked letters. This is aswell a major function in the game to give the gamer feedback and information.
 - The second unit test will be the state test. This function is making up the state of the game. Has the player won, lost or keep guessing. This is a part of the main game to make it work which make it very a important function.
 - The last unit test is the not implemented function. This function will show the highscore. This is the only function that we are testing that is not important to make the game work. However this is a function that makes the game more fun and competitive. That's why I also picked this one, and it is not implemented yet.

All these tests is important except for the high score, that's why I chose to test these methods.

- How this testing is going to be done, what should be dynamically or statically tested, what testing techniques are going to be used. (Look in Task 2, 3 for this...)
 - I am going to test the manual tests dynamically, and the unit tests will be statically tested

- I am going to use mocha and chai with the unit testing. This is a framework for javascript testing that is very popular and reliable.
- Make a time plan for this including time-estimations and measure and fill in actual time taken when you execute your plan.
 - See above (time plan at page 2)

Manual Test Case

TC1.1

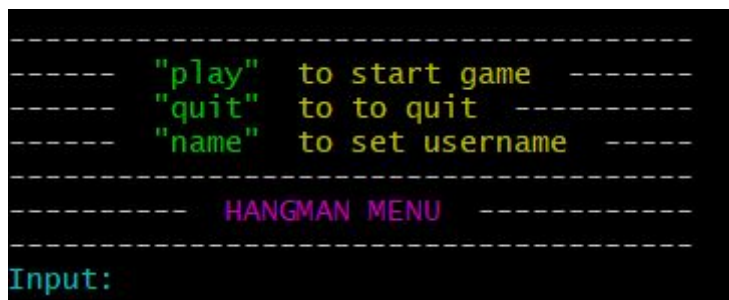
use case: Play the game (UC 2)

Scenario: Gamer plays the game of hangman

This is going to test the main scenario of the game were the gamer puts in a letter and hope to get it right. He wins or loses depending on if got the word right in time or not.

I am testing this since this is the most important part of the project, since this is a function that you have to use in order to take part of the game.

In order to perform this test you have to be inside this function. You get directed here by typing in "play" when you are in the main menu (main menu will be displayed when you start the application)



```
-----  
----- "play"  to start game -----  
----- "quit"  to to quit -----  
----- "name"  to set username -----  
-----  
----- HANGMAN MENU -----  
-----  
Input:
```

Preconditions:

- cd into the src, (for me: cd gb222hq_1dv600/src) with your cmd
- Type in "node mainManualTest.js" in cmd.
- Type in play to get into this test.

The test steps that needs to be performed is:

1. Type in the letter "m" then press enter.
2. Then repeat the previous step but with "o", "u" "s" "e"
3. when you press enter on "e", the text "You won!" will be displayed and you will get back to the main menu.

Here you

can see pictures how it should look like:

1.

```
-----
your username: Guest123 -----
life: 7/7 -----
"menu" to get back to menu ---
Your wrong letters:
Your word: [ '_ ', '_ ', '_ ', '_ ' ]
-----
input:
```

2.

```
-----
your username: Guest123 -----
life: 7/7 -----
"menu" to get back to menu ---
Your wrong letters:
Your word: [ 'm', '_ ', '_ ', '_ ', '_ ' ]
-----
input: o

-----
your username: Guest123 -----
life: 7/7 -----
"menu" to get back to menu ---
Your wrong letters:
Your word: [ 'm', 'o', '_ ', '_ ', '_ ' ]
-----
input: u

-----
your username: Guest123 -----
life: 7/7 -----
"menu" to get back to menu ---
Your wrong letters:
Your word: [ 'm', 'o', 'u', '_ ', '_ ' ]
-----
input: |
```

3.

```
input: e

You won!

-----
"play" to start game -----
"quit" to to quit -----
"name" to set username -----
-----
HANGMAN MENU -----
-----
Input: |
```

X	Letter added to word
X	Nothing unexpected
X	Won when completed the word
	Comments:

TC1.2

use case: Quit the application (UC 3)

Scenario: Gamer quits the application

This is going to test the quit function. The reason I am testing this function is because it is important to be able to quit the game, without terminating the console.

Preconditions:

- cd into the src, (for me: cd gb222hq_1dv600/src) with your cmd
- Type in "node mainManualTest.js" in cmd.

The test steps that needs to be performed is:

1. Type in "quit"
2. Type in "yes"

Expected result:

When you type in quit a box with yes --- no will show up.

when yes is performed the application will shut down and you will get back to where you were before you wrote "node mainMaunalTes.js".

You can also see pictures below how it should look like.

1.

```
gabri@LAPTOP-BG70C84H MINGW64 ~/gb222hq_1dv600/src (master)
$ node mainManualTest.js
Welcome to the hangman project

-----
-----  "play"  to start game  -----
-----  "quit"  to to quit   -----
-----  "name"  to set username -----
-----
-----  HANGMAN MENU  -----
-----
Input: quit|
```

2.

```
-----
-----  "yes"  or  "no"  -----
-----
input: yes
gabri@LAPTOP-BG70C84H MINGW64 ~/gb222hq_1dv600/src (master)
$ |
```

X	Quit keyword
X	Nothing unexpected
X	Yes quits application
	Comments:

Unit tests

unit test for visual word:

Test run for the visual word. The purpose of this function is to create an array of the randomly picked word and show the length of it and at what place your letters are.

```
Create visual word (createUserWord)
  ✓ Should be an array
  ✓ each spot should have _, test for last pos
  ✓ each spot should have _, test for first spot
```

The code for this test:

```
describe('Create visual word (createUserWord)', () => {
  wordArr = app.createUserWord('maybe')
  // testing that the visual word is an array, later each spot will be filled with one letter
  it('Should be an array', () => {
    assert.typeOf(wordArr, 'array')
  })

  // evry postion is the array should have _ that later will be replace with a letter, test for last position
  it('each spot should have _, test for last pos', () => {
    assert.equal(wordArr[wordArr.length - 1], '_')
  })

  // same as above but with first position
  it('each spot should have _, test for first spot', () => {
    assert.equal(wordArr[0], '_')
  })
})
```

Unit test for state:

Test run for state. This function is testing if the player has lost, won or should keep going. This is a picture of the original test. Underneath you can find a modified picture that does not show the interface and just the test.

```

state test

You got hanged

-----
-----  "play"  to start game  -----
-----  "quit"  to to quit   -----
-----  "name"  to set username -----
-----
-----  HANGMAN MENU  -----
-----
Input:    ↓ no tries left (lost)
          ↓ tries left, not complete word (keep going)

You won!

-----
-----  "play"  to start game  -----
-----  "quit"  to to quit   -----
-----  "name"  to set username -----
-----
-----  HANGMAN MENU  -----
-----
Input:    ↓ tries left, complete word (won)

```

The same test but modified picture to make the test more clear.

```

state test

↓ no tries left (lost)
↓ tries left, not complete word (keep going)
↓ tries left, complete word (won)

```

The code for state test:

```

describe('state test', () => {
  it('no tries left (lost)', () => {
    life = 0
    word = ['_', '_', '_']
    // testing for out of tries, player lose!
    assert.equal(app.checkState(life, word), true)
  })
  // testing for tries left but not yet a complete word
  it('tries left, not complete word (keep going)', () => {
    life = 3
    assert.equal(app.checkState(life, word), false)
  })

  // Testing for comple word, player won!
  it('tries left, complete word (won)', () => {
    word = ['w', 'o', 'w']
    assert.equal(app.checkState(life, word), true)
  })
})

```

Unit test for getHighScore:

This test is rather vague but it specifies how it should be structured but how the information is going to be displayed is to be decided at a later stage. The purpose of this function is to get the information about the highest scores.

```
6 passing (33ms)
2 failing

1) Get highscore
   should return an array with highscore:
AssertionError: expected undefined to be an array
at Context.it (test.js:45:16)

2) Get highscore
   should return top 5 scores:
TypeError: Cannot read property 'length' of undefined
at Context.it (test.js:49:47)
```

The code for the gethighscore:

```
describe('Get highscore', () => {
  it ('should return an array with highscore', () => {
    assert.typeOf(app.getHighscore(), 'array')
  })
  // should return an array with length 5, where every value in the array is information about the score
  it ('should return top 5 scores', () => {
    let lengthofScore = app.getHighscore().length
    assert.equal(lengthofScore, 5)
  })
})
```

Here is the entire test run:

```
Create visual word (createUserWord)
  ✓ Should be an array
  ✓ each spot should have _, test for last pos
  ✓ each spot should have _, test for first spot
```

```
state test
```

```
You got hanged
```

```
-----
-----  "play"  to start game  -----
-----  "quit"  to to quit   -----
-----  "name"  to set username -----
-----  "highscore" to see hihscore
-----
```

```
-----  HANGMAN MENU  -----
-----
```

```
Input:      ✓ no tries left (lost)
            ✓ tries left, not complete word (keep going)
```

```
You won!
```

```
-----
-----  "play"  to start game  -----
-----  "quit"  to to quit   -----
-----  "name"  to set username -----
-----  "highscore" to see hihscore
-----
```

```
-----  HANGMAN MENU  -----
-----
```

```
Input:      ✓ tries left, complete word (won)
```

```
Get highscore
```

- 1) should return an array with highscore
- 2) should return top 5 scores

```
6 passing (33ms)
2 failing
```

```
1) Get highscore
```

```
  should return an array with highscore:
  AssertionError: expected undefined to be an array
    at Context.it (test.js:45:16)
```

```
2) Get highscore
```

```
  should return top 5 scores:
  TypeError: Cannot read property 'length' of undefined
    at Context.it (test.js:49:47)
```

Reflection

The application seems to be working as expected. There is only one bug in the entire application that I have found so far and that is the high score function which put you into an dead end. This will be fixed in the next iteration.

Overall I think the tests are good and testing what needs to be tested. The high score test might need to be more precise to make it a solid test.

The fact that the unit test is executing the function with `console.log` etc makes the test a bit more confusing but I could not find a solution for this in mocha and chai.