

# The Hangman project

Project plan.

---

Author: Jonas Nilsson

Organisation: LNU

Date: 2020-02-03

# Contents

<b>1</b>	<b><u>Revision History</u></b>	<b>2</b>
<b>2</b>	<b><u>General Information</u></b>	<b>3</b>
<b>3</b>	<b><u>Vision</u></b>	<b>4</b>
<b>4</b>	<b><u>Project Plan</u></b>	<b>5</b>
4.1	Introduction . . . . .	5
4.2	Justification . . . . .	6
4.3	Stakeholders . . . . .	6
4.4	Resources . . . . .	6
4.5	Hard- and Software Requirements . . . . .	7
4.6	Overall Project Schedule . . . . .	7
4.7	Scope, Constraints and Assumptions . . . . .	7
<b>5</b>	<b><u>Iterations</u></b>	<b>9</b>
5.1	Iteration 1 . . . . .	9
5.2	Iteration 2 . . . . .	10
5.3	Iteration 3 . . . . .	11
5.4	Iteration 4 . . . . .	11
<b>6</b>	<b><u>Risk Analysis</u></b>	<b>13</b>
6.1	List of risks . . . . .	13
6.2	Strategies . . . . .	14
<b>7</b>	<b><u>Use case diagram</u></b>	<b>16</b>
<b>8</b>	<b><u>Fully dressed use cases</u></b>	<b>17</b>
<b>9</b>	<b><u>State machine diagrams</u></b>	<b>20</b>
<b>10</b>	<b><u>Class diagrams</u></b>	<b>21</b>
<b>11</b>	<b><u>Test</u></b>	<b>22</b>
<b>12</b>	<b><u>Time log</u></b>	<b>30</b>
<b>12</b>	<b><u>Handing in</u></b>	<b>31</b>

## 1 | Revision History

Date	Version	Description	Author
2020-02-03	1	First iteration: The first step where we start up the the documentation.	Jonas Nilsson
2020-02-24	2	Second iteration: We create Use case, state machine and class diagrams	Jonas Nilsson
2020-03-09	3	Third iteration: We test our code to make sure it works and no bugs	Jonas Nilsson
2020-03-20	4	Fourth iteration : Updating some instructions and general information. <ul style="list-style-type: none"><li>- Updating time logs for iteration 4</li><li>- Completing the product</li><li>- Updating Use cases</li><li>- Updating state machine</li><li>- Updating Class diagram</li></ul>	Jonas Nilsson

## 2 | General Information

### Project Summary

**Project Name**

Hangman

**Project ID**

JN\_Hangman\_Game

**Project Manager**

Jonas Nilsson

**Main Client**

Linnéuniversitet

**Key Stakeholders**

Project Manager

Tester and developer

User

**Executive Summary:**

Hangman is a game that most of you have probably played when you were kids. It's a game where the objective is to figure out a hidden word by guessing letters that are in the word you think it is. The purpose is to create a text based game with low requirements yet look interesting enough for people to play it. It will also serve as educational for children who are studying geography to learn the capital cities in europe. The player has to get the word in an limited amount of guesses, if said player manages to do that they win.

### 3 | Vision

The purpose of this game is to not let our old fun games get all dusty and forgotten. So we will reignite the passion for the game called Hangman and to not only have fun with it but to learn from it.

The games purpose is to have a hidden word and only showing with blank lines how long the word is. It is the players role to try to guess what word it is, if the player guessing a letter that the word doesn't have the game will draw out a line bit by bit which represents parts of the man that will be hanged.

To be able to win this game you have to get the correct word before all lines of the hanged man has been drawn.

Reflection:

Writing a vision about the product you are going to make is an easy way to plan ahead and make the way to the final product a straighter line. Instead of having to come up everything like, design etc while working on product, we have a goal from start and work to achieve that. Which I think relieves a huge amount of work regarding misunderstandings and mistake during the project.

## 4 | Project Plan

### 4.1 Introduction

Hangman will be a game that will be played in the console, where the player will guess letter by letter to get the correct capital city.

When the application is started the game will welcome the player and present a menu. As the game is started it will present to the player an line of ‘\_’ which will represent the word, so to say if capital to guess is tirana, which is six letters. The game will present a line looking like this: ‘\_ \_ \_ \_ \_ \_’.

In the cases a player guesses the correct letter it will appear in the the underline line replace that underline. Continuing on last example, if player guesses the letter ‘t’ the line will now look like this: ‘t \_ \_ \_ \_ \_’. In the cases a player guesses a letter that isn’t part of the capital city, the picture of the man that is going the be hanged will be presented. The picture will be completely drawn after 9 incorrect guesses which means Game over.

#### Game play:

To start the application you write, npm start in the console.

When application is started player will be presented with, menu prompting the player to choose between starting game or quitting game, player navigate by using up and down arrows, then confirm with enter. When game is started and the ‘\_ \_ \_ \_ \_ \_’ lin is shown, the game prompts player to guess a letter, player uses keyboard and types the letter ‘t’ then presses enter to guess the word.

If player wants to quit game mid session, writing ‘.exit’ and then pressing enter will make the game show : do you really want to quit? and player will pick between yes and no, using up and down arrows and then enter.

When game is won or lost game will prompt what the player is interested in doing next, ‘play again?’ or ‘quit game’ player navigates on the choices with the up and down arrows, if player picks play again a new session of the game is started. If player picks quit game the game will ask for confirmation if player rly wants to quit.

We will be working with this project by splitting up the workload in 4 iterations where we will complete different stages towards the goal of completing the application. The durations of this project will be for 9 weeks and the deadline is on Friday the 20th in March

## 4.2 Justification

Hangman is made in the purpose of giving the students some variation in the way of learning and studying. By giving them this variation of having fun while playing and at the same time learning we think will improve the kids learning abilities in for the future.

## 4.3 Stakeholder

Jonas Nilsson: Project Manager, programmer and tester

Programmer creates well structured and easy maintainable code for the hangman game

Tester has the role of testing the product during the creation progress to see if it is anything that doesn't work as it should or if some changes should be made to make it easier to understand or more interesting.

Project Manager has the role of scheduling the development of the project, estimating how much progress the product should have each day/week for it to get completed within the deadline. He also controls so the product is following the requirements of the product and also keeps the costs of the product the planned amount.

User will be the ones who will be playing the game, students who wants to learn the capitals in Europe.

## 4.4 Resources

Time frame for project: 7 to 8 weeks.

Lectures: live lectures and pre recorded ones on mymoodle.

The pre recorded lectures is a total of 17 hours to watch and learn about important techniques that will be useful for this project.

Literature: Software Engineering 10th Edition by Ian Sommerville.

There will be a lot of chapter to read, but the ones that will help us the most for this project is gonna be: 2, 3, 4, 5, 6, 7, 8, 15, 20, 22, 23

## 4.5 Hard- and Software Requirements

### Hardware requirements:

A working computer

### Software requirements:

Node.js v12.9.1

IDE: Visual studio code

Lucidchart.com will be used to create diagrams such as Use Case diagram

## 4.6 Overall Project Schedule

First iteration deadline 2020-02-03

Second iteration deadline 2020-02-24

Third iteration deadline 2020-03-09

Fourth iteration deadline 2020-03-20

## 4.7 Scope, Constraints and Assumptions

Scope: Our console application Hangman is constructed on the Node.js platform. The basic usage of this application is that it should be able to randomize a capital city from the given list we have. After generating a word it will display said word as a line of \_ representing each letter from that word. The player should be able to guess letter by letter to get the word. In the case of the letter existing in the given word it will replace the \_ at the correct place, if it's wrong the hangman character will be started to be drawn out. In the case where the full picture of the hangman is drawn the player loses and its game over. When the player loses he will be presented a choice to either play again or quit the game. If he want to play again the game will generate a new word. If the player instead guesses the whole word correct he will win. After winning he will be asked if he want to play again or quit, same as what happens when losing.

For the application to pick random capital cities in Europe we have a file containing them.

Our project will not include any multiplayer options in the current version, might be a feature in future versions of the game.



Constraints: No experience creating multiplayer options for games and also first time creating a project plan to follow through the course of a creation of an application so might not be optimized to the best efficiency.

One other constraint is that this project only has 1 developer which is also doing the work of the project manager, which limits the project in the cases of experience, knowledge and other aspects that can be needed for this project.

Assumptions: The user have basic knowledge about using a computer and can understand English.

### **Reflection:**

For me it was quite hard coming up with the project plan and how it was supposed to be like. The main problem for me was to figure out a reasonable justification for the project so it could get a feeling of a real project.

## 5 | Iterations

Our project is split up into four iterations during the development process, where each iteration is connected to the next one, hence it is very important to do them one by one to have an easier and clear view of how the progress of the product is going. If you choose to not do any iterations it is a very high risk that the final product is different from what the developer had in mind.

### 5.1 Iteration 1 (deadline: 2020-02-03)

In this projects first iteration our goal is to create a plan covering things as:

Planned activities	Estimated Time
Writing General information	30 min
Writing Vision	1 h
Writing project plan subcategories	1 h 30 min
Writing Iterations	2 h
Writing Time log	20 min
Writing skeleton code	30 min

## 5.2 Iteration 2 (deadline: 2020-02-24)

The second iterations is where we start thinking about the modeling. We will be using the UML language to create the project.

Planned activities	Estimated Time	Time taken
1. Create time log and estimations of work progress.	30 min	45 min
2. Read Software Engineering( chapters: 4,5,6,7,15,20)	8 h	14h
3. Watch pre recorded lectures on the course	7 h	7 h
4. Create Use Case Diagram	3.5 h	3 h
5. Create Fully Dressed Use Case for play game	4 h	4.5 h
6. Create State Machine Diagram	4 h	4. 2 h
7. Implement the basic menu flow of the game	4.5 h	4 h
8. Create Class Diagram	4 h	3.5 h
9. Make updates in project plans along the way	2 h	3 h

### 5.3 Iteration 3 (deadline: 2020-03-09)

In the third step it is time for us to start doing the testing. Because of that it is a good idea to put up a testing plan, so we know what kinda tests we should be doing. We need to create some test cases for our use case and have these test be executed. All of our test should cover as much as possible of the code, so we can inspect the whole code and also do automatic tests on it.

After we have done all of our testing we will follow that up with a report of how the test went.

Planned activities	Estimated Time
Manual Test Cases	4 h
Unit Tests	6 h
Running manual tests	1 h
Code inspection	2.5 h
Test report	1 h

### 5.4 Iteration 4 (deadline: 2020-03-20)

This is the final iteration for our project. In this iteration we have a finished game which we will present with documentation and also having it thoroughly tested.

Planned activities	Estimated Time
Update Project plan	2 h
Update Use cases	2h
Update State machine diagram	1 h
Complete the application	5 h
Update Class diagram	1 h
Adding extra feature (highscore)	2 h

**Reflection iteration 4**

This is the final iteration of the project so in this step, I completed the code to make the game be fully playable, completing the code for the game was the most time consuming part of this but could have been prevented if I did get further on it in earlier steps than I did. Updating the use cases, state machine diagram and class diagram went decently fast since I used the ones from earlier iteration with just some new added stuff to fit to how the final product looks like. By updating the project plan and putting it all together as a whole made it feel like it actually was getting into the state of completion, and made me feel good. I added the extra feature of highscores for this iteration, it was quite hard to get it to work but using json to save and add them made it easier after reading the documentation.

## 6 | Risk Analysis

### 6.1 List of risks

Risk	Probability	Impact
Time estimation risks: <ul style="list-style-type: none"> <li>- Underestimate time to code the product.</li> <li>- Underestimate amount to collect data</li> </ul>	High	Serious
Personal risks: <ul style="list-style-type: none"> <li>- Getting infected by some virus and getting sick (coronavirus is a very dangerous virus going around right now)</li> <li>- Not having experience working according to a planned schedule like this</li> <li>- Forgetting to save</li> </ul>	Low   Low   Low	Catastrophic   Moderate   Serious
Technology risks: <ul style="list-style-type: none"> <li>- Hardware that is being used for projects cease to function</li> </ul>	Low	Serious/Moderate
Requirement risks: <ul style="list-style-type: none"> <li>- Releases of new requirements that causes major change to the product</li> </ul>	Low	Serious

## 6.2 Strategies

### Time estimation risks:

- Underestimate time to code:  
In order to prevent us from getting behind our schedule is to have proper estimation of how long it will take to write the code. This is quite hard to estimate on bigger projects at least because it always differ from person to person. Therefore we plan it in such a way so we have room for some delay which is not optimal to be but in case of increasing morale on workers when you are ahead of schedule.
- Underestimating the amount of data to collect and how much time the learning of each will take is also a risk that is easy to happen when you don't have enough experience. We follow the same tactic here and schedule for it to take longer than we actually think it will take.

### Personal risks:

- Getting sick causes a major problem as it makes our energy levels go down and decreases our effectivity, if we are even feeling healthy enough to be working, this causes us to get behind on schedule a lot. It would be extremely catastrophic if we were to caught coronavirus as that would make it even harder for us to get our work done in time.  
In order to prevent this we encourage our workers have a good hygiene wear proper clothing to not be cold and to always use hand sanitizer. If person still get sick we have so we can work from home at laptop and we also want to have good documentation so other persons can understand the code incase help is needed.
- Experience is something you have to build up, so doing something for the first time will always be a risk you have to take. As long as you are striving to improve this is something you can overcome in time and build up your experience.
- We are all human beings and noone is perfect, so mistakes are bound to happen. Mistakes to happen is never a thing you want to happen but it's

important that you learn from them. Forgetting to save is mistake you most likely will have done but in order for us to not make that mistake we try to fix in the habit of saving every 10 minutes of work so we don't lose our progress.

#### Technological risks:

- Hardware seizing to function, is a risk that can cause a great deal of delay on a project, losing all of our data and work. In order to reduce the seriousness of this risk we work with gitlab/github and push up our code there so we have it saved both there and on our hardware. So it's not the end of the world if our hardware brakes.

#### Requirement risks:

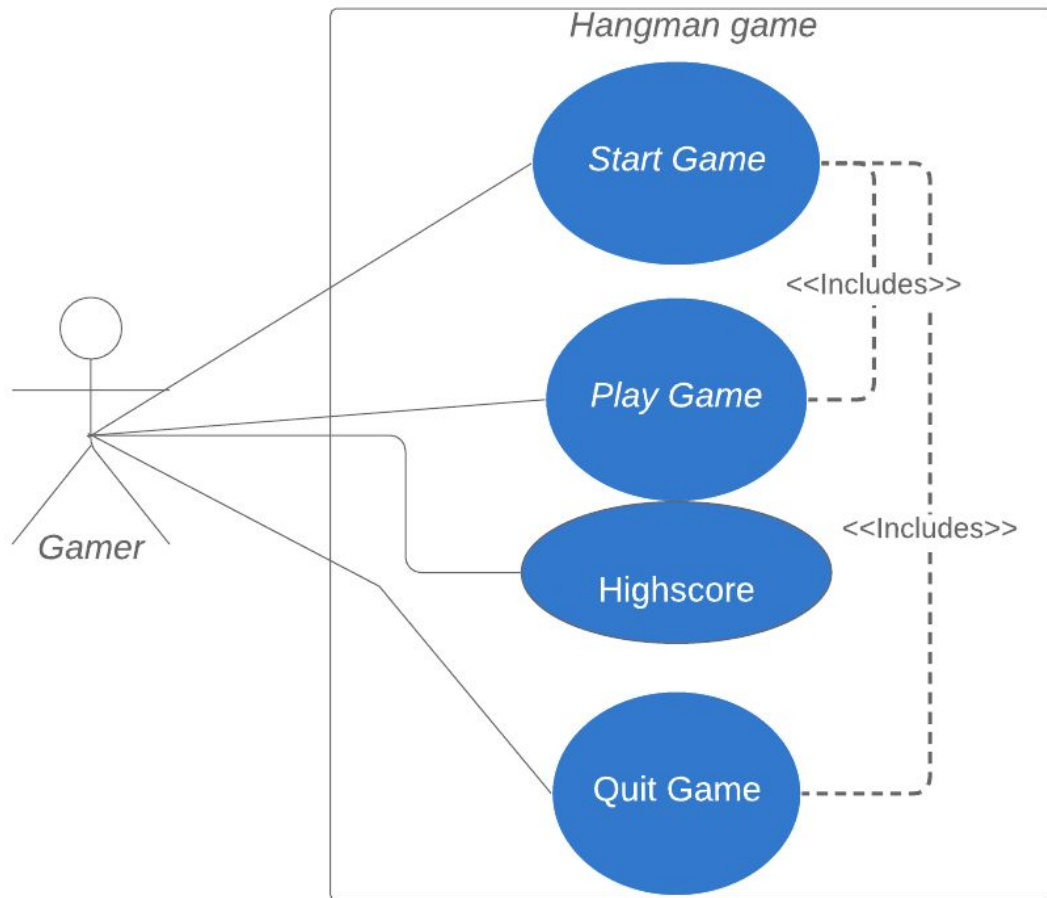
- Releases of new requirements can cause a lot of changes being made for the product, depending on when the new requirements are being released this can cause a great inconvenience forcing us to go backwards to add or fix some parts that wasn't needed before. In order to prevent this from being a big inconvenience is to make to code easy manageable to that it's easy to add changes.

#### Reflection:

It is an important task to take into account that risks are bound to happen while working on a project. Which is why we have to think about what we can do to prevent them and also what course of action do we take if any of these risks happens. Thinking about the risks that can happen during a project opened my eyes and made my realise how important it is to be aware and also thinking ahead for the best possible solution.



## 7 | Use case diagram



## 8 | Fully dressed use cases

### 2.1 UC 1 Start Game

Precondition: None.

Postcondition: the game menu is shown.

#### Main scenario

1. Starts when a player wants to play a session of hangman.
2. The system presents the main menu with a welcome message and the asks user the enter a username.
3. Player enters username by typing 'Jonas' with the keyboard and presses enter.
4. Game menu is shown giving the player the options of Play game, Highscores, Quit game.
5. Player navigate with up and down arrows to the Play game and presses enter, the system starts up the game (See Use Case 2).

#### Alternative scenario

5.1 The player decides to quit the game.

1. The system quits the game(see Use Case 3)

5.1 Invalid menu choice

1. The system presents an error message
2. Go back to menu, step 2

## 2.2 UC 2 Play Game

Precondition: The game is started.

Postcondition: The game shows a line of \_ showing the word to guess.

### Main scenario

1. Starts when player has decided to start the game.
2. A string of \_ is presented representing the word to be guessed.
3. The player guesses a letter.
4. The system shows that letter at the correct place of the word.
5. Repeats until player guess the last correct letter.
6. System presents a Winning message showing that the player won.
7. An option to play again or quit game is shown.

### Alternative scenario

3.1 The player guesses the wrong letter.

1. System draws a part of the picture of the man that is about the get hanged
2. Go to step 3

6.1 The player guesses the last guess wrong(the hangman picture is complete)

1. The system announces that the game is over.
2. Go to step 7

## 2.3 UC 3 Highscore

Precondition: Game is running

Postcondition: Highscore list is shown.

### Main scenario

1. Starts when the player wants to see the highscore list.
2. List of usernames of users and the amount of guesses will be shown.

Looking like this:

- 1 Jonas with the score of: 5
- 2 Peter with the score of: 6

## 2.4 UC 4 Quit Game

Precondition: Game is running.

Postcondition: Game is terminated.

### Main scenario

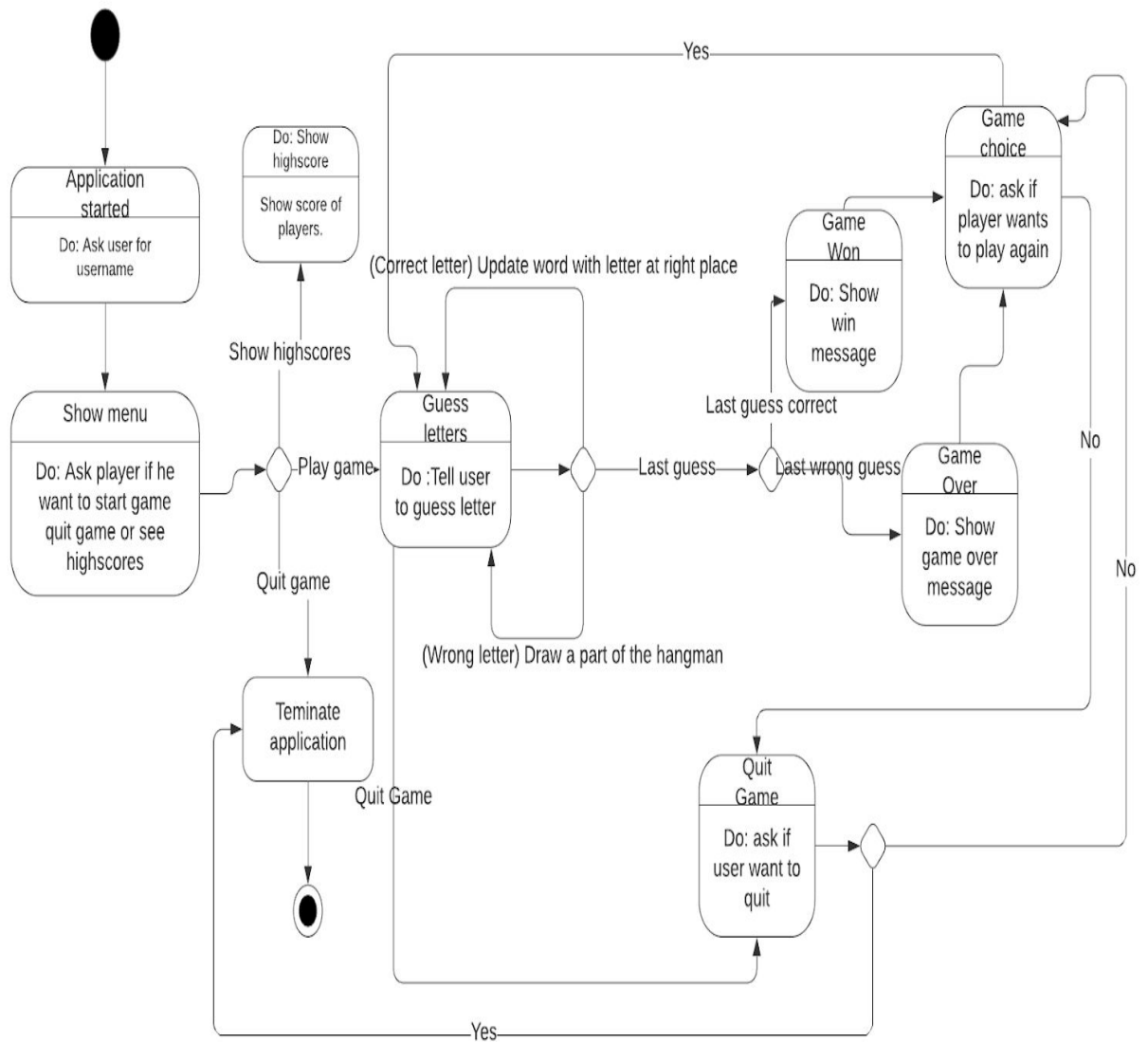
1. Starts when the player wants to quit the game.
2. The system asks for confirmation.
3. Player confirms.
4. The system terminates the session.

### Alternative scenario

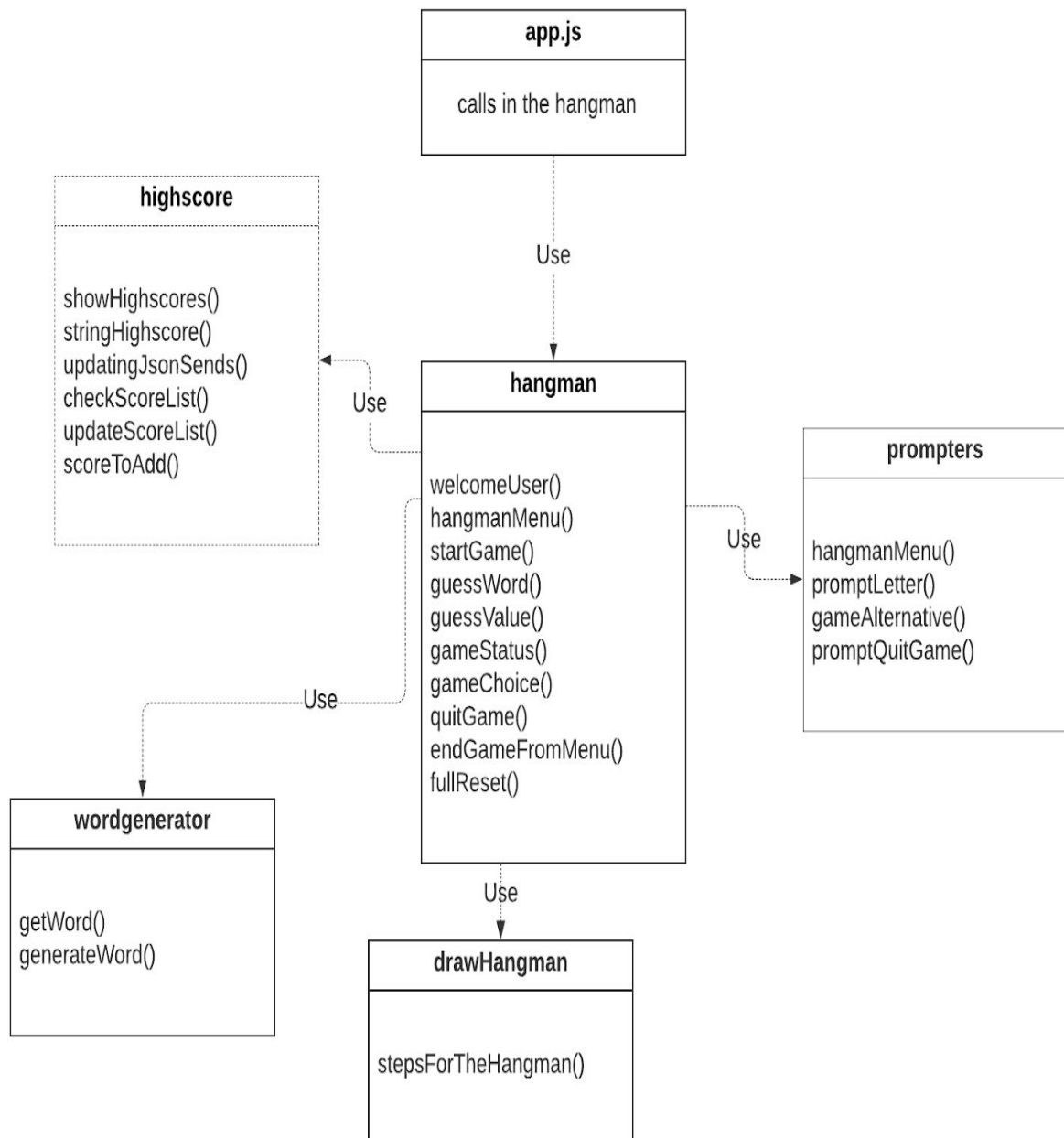
3.1 The player does not confirm.

1. The system returns to previous stage.

## 9 | State machine diagrams



## 10 | Class diagrams



# 11 | Test

## Objective

The objective of this is to test our code that has been implemented in the iterations we have done.

## What to test and how we test

The purpose of the tests will be to find bad written code and errors as early as possible and improve the overall quality of the code.

I will be testing my UC1 and UC2 in the method of writing test cases manually.

Automated test is something that will also be used for the implemented code. The framework I've chosen to work with will be "jest" since it works with node.js and it's simple to use. Where we will test the code by using the logic of the developer.

## Time Plane

Task	Estimate	Actual
Manual Test Cases	4 h	4 h
Unit Tests	6 h	6.5 h
Running manual tests	1 h	1 h
Code inspection	2.5 h	3 h
Test report	1 h	1 h

## Manual Test Cases

### TC 1.1 Start Game

Use Case: UC 1 Start Game

Scenario: The game menu is shown.

Is started when the application is started

Precondition: None

1. Start app by entering "npm start" in the terminal and press enter.
2. Game menu shows up welcoming the user and asking if you want to Play Game or Quit Game.

Expected:

Welcome to hangman, please have a fun and relaxed time playing the game!

? Welcome to Hangman! Hope you have a fun time!

> Play game

Quit game

### TC 2.1 Play Game

Use Case: UC 2 Play Game

Scenario: A random word shown as \_ to represent the words length.

Is started if user decides to Play Game

For Testing purposes we want to get the word tirana. For us to be able to test this, uncomment the code in module “wordgenerator.js” on line 31. to set “this.capital = ‘tirana’.

Precondition: User in the previous step from the menu choose to start game by navigating with up and down arrows to “Play Game” and then pressing enter.

Expected:

The base of the hangman picture before errors starts to apply, this will always show and after wrong guesses it will start to build up more.

Getting a Line of \_ in the likes of “\_ \_ \_ \_ \_” representing our word.

A text saying ‘Guess the European capital! If you want to exit the game write “.exit” ‘ giving the user information on what to guess on and also information on how to quit the game.

The representation will look similar to this:

```

|
|
|
|
|
|
|_ _ _ _ _

```

-----  
 Guess the European capital! If you want to exit the game write ".exit"

The word is tirana.



### TC 3.1 Highscore

Use Case: UC 3 Highscore

We generate up a list that will show us users that have played the game before.

We want to test so that this work so our user can use this to compete and trying to improve their guessings for both learning and for competitive progression. By making a highscore list we believe it will make the game more fun as our user will strive to get that number 1 spot.

Expected:

Will represent a list of usernames and say what score they got.

Looking like this:

1 Jonas with the score of: 5

2 Peter with the score of: 6

## Test Report Manual Test Cases

Test	UC 1	UC2	UC 3	UC 4
TC 1.1	1/OK			
TC 2.1		1/OK		
TC 3.1			1/OK	
Coverage & Success	1/OK	1/OK		

## Unit Test Results

The unit tests are created in Visual Studio Code and the framework I have used is called jest.

To run the tests write “npm test” in the terminal and press enter

### Test result:

```
Test Suites: 2 failed, 2 passed, 4 total
Tests:      2 failed, 8 passed, 10 total
Snapshots:  0 total
Time:       3.713s
Ran all test suites.
npm ERR! Test failed.  See above for more details.
```

**wordgenerator.test.js:**

```

tests > wordgenerator.test.js > ...
3
4  * @author Jonas Nilsson
5  * @version 1.0.0
6  */
7
8  const { WordGenerator } = require('../src/wordgenerator')
9  const wordTest = new WordGenerator()
10
11  // Tests method getWord
12  test('Should return word "tirana"', () => {
13      let input = 0
14      let expected = 'tirana'
15      let real = wordTest.getWord(input)
16      expect(real).toBe(expected)
17  })
18  test('Should return word "sarajevo"', () => {
19      let input = 6
20      let expected = 'sarajevo'
21      let real = wordTest.getWord(input)
22      expect(real).toBe(expected)
23  })
24
25  // Tests method generateWord
26  test('Should return an array of 6 underscores', () => {
27      wordTest.lineArray = []
28      wordTest.capital = 'tirana'
29
30      let expected = [ '_', '_', '_', '_', '_', '_' ]
31      let real = wordTest.generateWord()
32      expect(real).toEqual(expected)
33  })
34  test('Should return an array of 8 underscores', () => {
35      wordTest.lineArray = []
36      wordTest.capital = 'sarajevo'
37
38      let expected = [ '_', '_', '_', '_', '_', '_', '_' ]
39      let real = wordTest.generateWord()
40      expect(real).toEqual(expected)
41  })
42

```

Link to the test-code source:

<https://gitlab.lnu.se/1dv600/student/jn223ck/assignment-4/blob/master/tests/wordgenerator.test.js>

**hangman.test.js:**

```


tests > hangman.test.js > ...
4  ~ @author Jonas Nilsson
5  * @version 1.0.0
6  */
7  'use strict'
8  const { Hangman } = require('../src/hangman')
9  const hangmanTest = new Hangman()
10
11  let outputData = ''
12  const storeLog = inputs => (outputData += inputs)
13
14  test('hangmanTest.guessValue should contain "a"', () => {
15    let input = 'a'
16    hangmanTest.wordToGuess = 'tirana'
17    hangmanTest.guessValue(input)
18    let expected = 'a'
19    let actual = hangmanTest.lettersGuessed
20    expect(actual).toContain(expected)
21  })
22
23  test('We should get a console log telling us "You have already guessed: "a"', () => {
24    console['log'] = jest.fn(storeLog)
25    hangmanTest.countErrors = 0
26
27    let input = 'a'
28    hangmanTest.guessValue(input)
29
30    let expected = (`You have already guessed: "a"`)
31    expect(outputData).toBe(expected)
32  })
33  test('We should get our underline array to look like [ _ _ _ a _ a ]', () => {
34    hangmanTest.lettersGuessed = []
35    hangmanTest.wordToGuess = 'tirana'
36    hangmanTest.gameWord.lineArray = [ '_', '_', '_', '_', '_', '_' ]
37
38    let input = 'a'
39    hangmanTest.guessValue(input)
40    let expected = [ '_', '_', '_', 'a', '_', 'a' ]
41    let actual = hangmanTest.gameWord.lineArray
42    expect(actual).toEqual(expected)
43  })

```

Link to the test-code source:

<https://gitlab.lnu.se/1dv600/student/jn223ck/assignment-4/blob/master/tests/hangman.test.js>

highscore.test.js:

```
tests >  highscore.test.js > ...
1  /**
2   * Module for testing highscores
3   * @author Sobaze Nilsson
4   * @version 1.0.0
5   */
6  'use strict'
7
8  const { Highscore } = require('../src/highscore')
9
10 const testScore = new Highscore()
11
12 test('Should return the array sorted by points, lowest first [{name: Sobaze, points: 5}, {name: Logan, points: 6}, {name:Per, points: 10} ]', async () => {
13   let taking = [{ 'name': 'Per', 'points': 10},{ 'name': 'Sobaze', 'points': 5},{ 'name': 'Logan', 'points': 6 }]
14   let expected = [{ 'name': 'Sobaze', 'points': 5}, { 'name': 'Logan', 'points': 6}, { 'name': 'Per', 'points': 10}]
15
16   let actual = testScore.showHighscores(taking)
17
18   expect(actual).toEqual(expected)
19 })
20
21 test('Should return "[1 Jonas with the score of: 5, 2 Erik with the score of: 6]"', async () => {
22   let taking = [ { 'name': 'Jonas', 'points': 5}, { 'name': 'Erik', 'points': 6}]
23
24   let expected = ['1 Jonas with the score of: 5', `2 Erik with the score of: 6`,]
25   let actual = testScore.stringHighscore(taking)
26   expect(actual).toEqual(expected)
27 })
28
```

Link to the test-code source:

<https://gitlab.lnu.se/1dv600/student/jn223ck/assignment-4/blob/master/tests/highscore.test.js>

failing.test.js:

```
tests > failing.test.js > ...
1  /**
2   * Module for a failing test
3   *
4   * @author Jonas Nilsson
5   * @version 1.0.0
6   */
7
8
9  const { Hangman } = require('../src/hangman')
10
11  const failTest = new Hangman()
12
13  test('Should return true if an even number', () => {
14    const evenNumber = failTest.failingTest(3)
15    expect(evenNumber).toEqual(true)
16  })
17
```

Link to the test-code source:

<https://gitlab.lnu.se/1dv600/student/jn223ck/assignment-4/blob/master/tests/failing.test.js>

### Motivation:

- 1) First I constructed test files for my wordgenerator module, my reasoning behind this was that I wanted to make sure I could generate out the correct word I wanted from the list of the json file they came from, also wanted to make sure the underline array representing the words came out as it was expected to so I also tested this.
- 2) Secondly I tested my hangman module. here I wanted to test out if they value of a guess went through correctly and that it also tracked that the letter had already been guessed which the first two test does control. I also wanted to make sure in the testing that if a letter is in the word to be guessed that the underline array actually adds it at the correct places.

- 3) For my third test I wanted to test my newly added feature which is highscore, the main points I wanted to make sure worked, was if the sorting of the array worked, sadly I didn't manage to make the testing for that work, but I tried it myself and it worked. Second test I did was to check if it writes out the leaderboard of the highscore list correctly, this one succeeded and worked.
- 4) For the third test which is my failing test, as we needed to have a test that was supposed to fail. It is a very simple one just to learn and see how to make a test fail.

## 12 | Time log

We compare our estimates that we set during our planning and time it took to implement. To improve see what went wrong to improve for future.

### Iteration 1

Planned activities	Estimated Time	Time taken
Writing General information	30 min	35 min
Writing Vision	1 h	1 h 20 min
Writing project plan subcategories	1 h 30 min	1 h
Writing Iterations	2 h	2 h
Writing Time log	20 min	25 min
Writing skeleton code	30 min	30 min

### Iteration 2

Planned activities	Estimated Time	Time taken
1. Create time log and estimations of work progress.	30 min	45 min
2. Read Software Engineering( chapters: 4,5,6,7,15,20)	8 h	14h
3. Watch pre recorded lectures on the course	7 h	7 h
4. Create Use Case Diagram	3.5 h	3 h
5. Create Fully Dressed Use Case for play game	4 h	4.5 h
6. Create State Machine Diagram	4 h	4. 2 h
7. Implement the basic menu flow of the game	4.5 h	4 h
8. Create Class Diagram	4 h	3.5 h
9. Make updates in project plans along the way	2 h	3 h

## Iteration 3

<b>Planned activities</b>	<b>Estimated Time</b>	<b>Time taken</b>
Manual Test Cases	4 h	4 h
Unit Tests	6 h	6.5 h
Running manual tests	1 h	1 h
Code inspection	2.5 h	3 h
Test report	1 h	1 h

## Iteration 4

<b>Planned activities</b>	<b>Estimated Time</b>	<b>Time taken</b>
Update Project plan	2 h	3 h
Update Use cases	2h	1 h 30 min
Update State machine diagram	1 h	1 h
Complete the application	5 h	6 h
Adding new feature (highscore)	2 h	3 h

## 13 | Handing in

app.js

hangman.js

wordgenerator.js

highscore.js

prompters.js

drawHangman.js