SOFTWARE DEVELOPMENT PROJECT - HANGMAN

Mattias Ruljeff
Rulles Hack

Project name	Hangman
Revision	7
Project ID	1
Project manager	Mattias Ruljeff
Main Client	High School‘s with programming orientation
Key stakeholders	Developer
Executive Summary	For teachers teaching programming and students who wants to learn programming in an easy and fun way

Content

1. Revision History

Date	Revision	Description	Author
2020-02-03	1	• Added this document.	MR
2020-02-13	2	• Added User Cases, iteration 2 and Check-state diagram.	MR
2020-02-17	3	• Changed the vision. • Added reflections on the vision, risks and the project plan.	MR
2020-02-21	4	• Changed the Check-state diagram, planned time for hangman-game changed to 10 hours. • Rewritten UC case 1-2-3. Added class diagram.	MR
2020-03-02	5	• Added test plan, test cases and test report.	MR
2020-03-05	6	• Added screenshots on unit tests and the results of the tests.	MR
2020-03-20	7	• Added time-logs for all iterations. • Added highscore in unit test. • Changed screenshots on Unit Tests and removed failed test. • Changed risk strategies.	MR

Full name of Author	
Initials	**Author name**
MR	Mattias Ruljeff

2. Vision

The game will be targeted to High Schools and their students, for understanding programming structures and how they can develop games in JavaScript using an IDE.

This game target is to get students interested in programming, and games are often a good way to get students engaged and excited.

This will be a good template for teachers for teaching. The teachers can inspire the students to make their custom features in the game and how to, in an easy way, implement them.

In the Hangman-game, a user should guess a given word, letter by letter. It shall be a game that the user thinks is worth spending time with, being easy to get started playing with via the menus, that shall be minimal in choices that the user can make. The scoreboard at the end of the game will give the user motivation to play the game repeatedly.

3. Project plan

Introduction

A game of hangman. Played by a user in the computer console window.
The user shall be able to insert a user-name.
The animation-steps of the hangman-game are:

1. The ground and vertical pole.
2. The horizontal pole.
3. The supporting pole for the horizontal pole.
4. The rope.
5. The head.
6. The torso.
7. The arms.
8. The legs.

The user will be given eight chances to guess the letters in the word.
When the user loses, a scoreboard will be presented and the user will be asked if he/she wants to play again.

Justification

- Hangman game will be a good teachers-tool to learn students how to program, use the console window, use Gitlab to save project-files and download project-files.

Stakeholders

- Development team, high Schools, teachers and students.

Resources

- Sommerville software engineering.
- Videos in 1DV600 course-page.
- An IDE of choice (preferable Visual Studio Code), Gitlab and Gitbash.

Hard- and Software Requirements

- A computer.
- An IDE, GitLab and Gitbash.

Overall Project Schedule

- 6 (Complete assignment 1)
- 9 (Complete assignment 2)
- 11 (Complete assignment 3)
- 12 (Submit complete project)

Scope, Constraints, and Assumptions

Scope:

- The game will initially not be connected to any database.
- The game will be playable from the start. The teacher or the students will be able to modify and add functionality to the game.

Constraints:

- The game will only be playable in the terminal window, and will not be a stand-alone app.
- The user must have all the files containing the game and an IDE of choice.

Assumptions:

The project will assume that the user knows how to use Visual Studio Code (or any similar IDE) to start the game via the console window. The user knows how to install Node.js latest version and the packages that are included in the game.

4. Iterations

Included parts:

- Read Software engineering.
- Create Hangman-game.
- Create documentation.
- Plan iterations.

Iteration 1

- Create documentation.
- Start to build the Hangman-game. Add all necessary files for the project.
- Read chapters 2, 3, 22 and 23.
- Watch video-lectures 2 and 3.

Iteration 2

- User case diagrams.
- User cases.
- Class diagram.
- Check-state diagram.
- Introduce more functionality to the game.
- Read chapters 4, 5, 6, 7, 15 and 20.
- Watch lectures 4, 5, 6, 7 and 8.

Iteration 3

- Game-testing.
- Test plan.
- Test Cases, manual and unit tests.
- Coding Unit tests.
- Hangman unicode-array.
- Read chapter 8.
- Watch lectures 9, 10 and 10.5.

Iteration 4

- Complete the game:
  - Add lines in the console where there shall be letters.
  - Create a check for the word “quit” to exit the game while guessing letters in the word.
  - Create the scoreboard.
- Fix issues from earlier iterations.
  - Time-log assignment 2
  - User Case 2 – Add trigger, add more steps in UC2 main scenario.
  - Fix State Machine. Change state-names and change action-symbols.
  - Rewrite Class Diagram.
- Complete the report.

5. Risk Analysis

List of risks

Risk	Affect	Description	Probability	Effects
Hardware failure	Project	If the computer/hard-drive crashes	Low	Catastrophic
Sickness	Project and product	If I get sick and can't go to school	Moderate	Serious
Size underestimate	Project and product	If the added features are too complicated/hard to integrate	Moderate	Serious
Requirement change	Project and product	If the product specifications change	Moderate	Tolerable

Strategies

Risk	Strategy to minimize impact of risk
Hardware/software	Save all the files to Gitlab. Backup-computer available
Sickness	Work from home in case of sickness
Size underestimate	Skip extra functions
Requirement change	Build modules that can easily be modified

6. Time Log

Tasks	Planned time (hours)	Actual time (hours)	Starting date	Latest update	End date
Iteration 1					
Create document	4	5	2020-02-28 09:00	2020-02-03 13:00	**2020-02-03 13:00**
Start creating Hangman Game	5	3	2020-02-28 09:00	2020-02-03 13:00	**2020-02-03 13:00**
Education and research	12	16	2020-02-28 09:00	2020-02-03 13:00	**2020-02-03 13:00**
Iteration 2					
Use Case diagram	1	1	2020-02-10 09:00	2020-02-24 16:00	**2020-02-24 16:00**
User Cases	1	1	2020-02-10 09:00	2020-02-24 16:00	**2020-02-24 16:00**
Class Diagram	1	0,5	2020-02-10 09:00	2020-02-24 16:00	**2020-02-24 16:00**
Check-state Diagram	1	1	2020-02-10 09:00	2020-02-24 16:00	**2020-02-24 16:00**
Adding more functions to Hangman game	3	2	2020-02-10 09:00	2020-02-24 16:00	**2020-02-24 16:00**
Education and research	16	18	2020-02-10 09:00	2020-02-24 16:00	**2020-02-24 16:00**

Tasks	Planned time (hours)	Actual time (hours)	Starting date	Latest update	End date
Iteration 3					
Project plan	10	9	2020-02-03 11:00	2020-02-17 14:30	**2020-02-17 14:30**
Hangman-game	10	4	2020-01-31 15.30	2020-02-21 14:00	**2020-02-21 14:00**
Writing tests	10	7	2020-03-02 09:00	2020-03-05 20:46	**2020-03-05 20:46**
Testing	5	3	2020-03-04 09:00	2020-03-05 20:46	**2020-03-18 14:00**
Iteration 4					
Complete Hangman Game	2	4	2020-03-18 15:00	2020-03-20 15:30	**2020-03-20 15:30**
Fix Issues from earlier Iterations	4	4	2020-03-18 09:00	2020-03-18 14:00	**2020-03-18 14:00**
Complete the report	3	7	2020-03-18 09:00	2020-03-20 22:00	**2020-03-20 22:00**
Total time	113	85,5			

10

7. User Cases

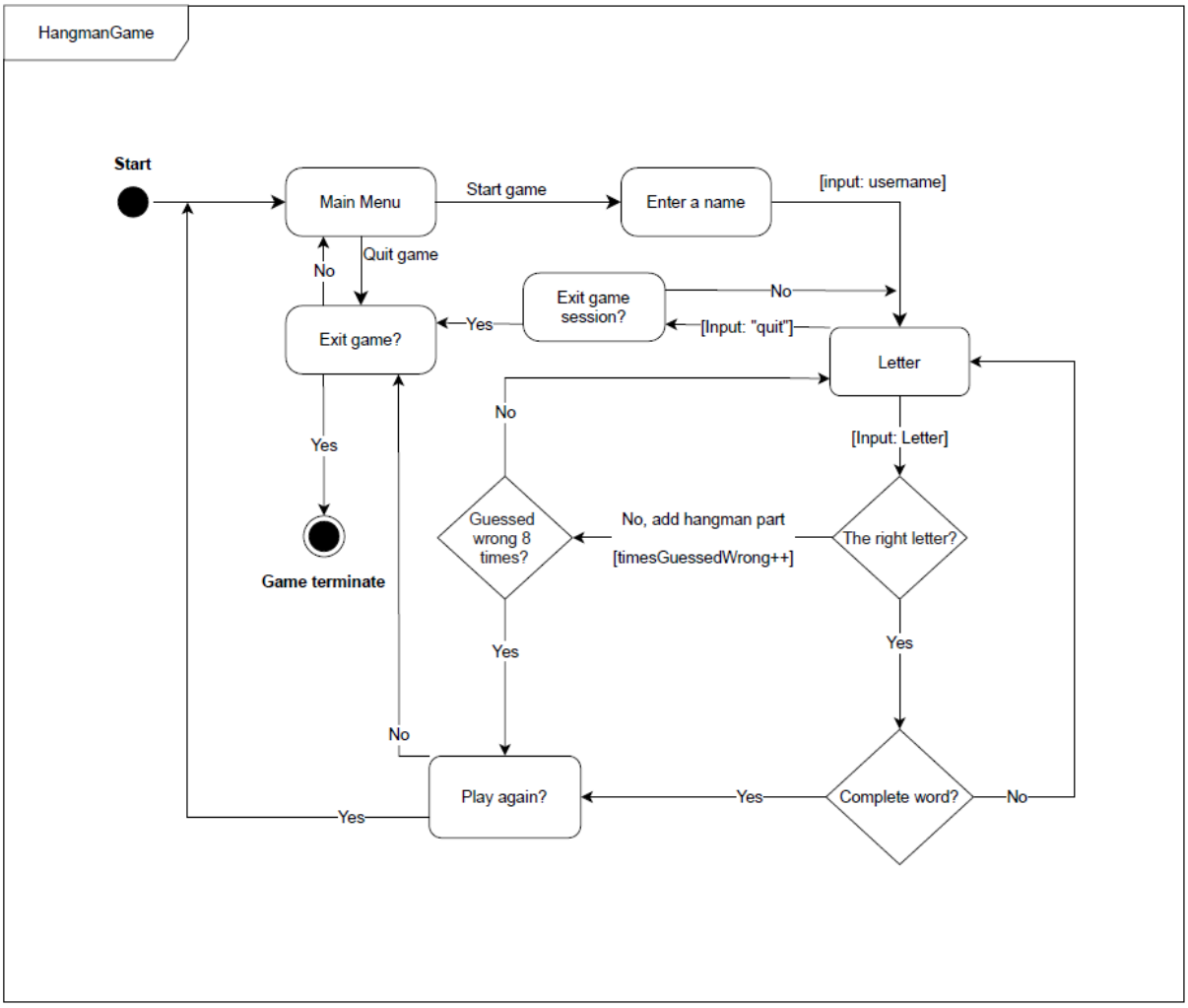
UC 1 Start Game	
Preconditions:	None
Post conditions	The game menu is shown
Main Scenario	
1. Starts when the User starts a session of Hangman-game. 2. The Systems presents the main menu, presenting a welcome message, the options to play or to quit the game. 3. The User starts the game.	
Alternative scenarios	
2.1 The User quits the game. 2.2 The System prompts if the User wants to quit the game. (See UC 3)	

UC 2 Play the game	
Preconditions:	• User has chosen to play the game. • The game must be started.
Post conditions	The UC 1 or UC 3 is triggered
Main Scenario	
1. The User enters a name. 2. The System displays the word-lines and prompts the User to enter a letter. 3. The User enters a letter. 4. Repeat steps 2-3 until the User has guessed the whole word. 5. The System prompts if the User wants to play again. (return to UC 1)	
Alternative scenarios	
1.1 The user doesn't enter a name, the system prompts the user to "Enter a name with more than 1 letter". 2.1.A The system detects that the letter is wrong, creates parts of the hanged man and presents it to the user. 2.1.B The user guessed wrong 8 times, loses the game. (See UC 3) 2.1.C The system prompts if the user wants to play again. 2.2.A The system detects that the letter is wrong, creates parts of the hanged man and presents it to the user. 2.2.B The user enters "quit". 2.2.C The System prompts if the User wants to quit to the main menu. 2.2.D The User confirms (see UC 3)	

Continues on the next page.

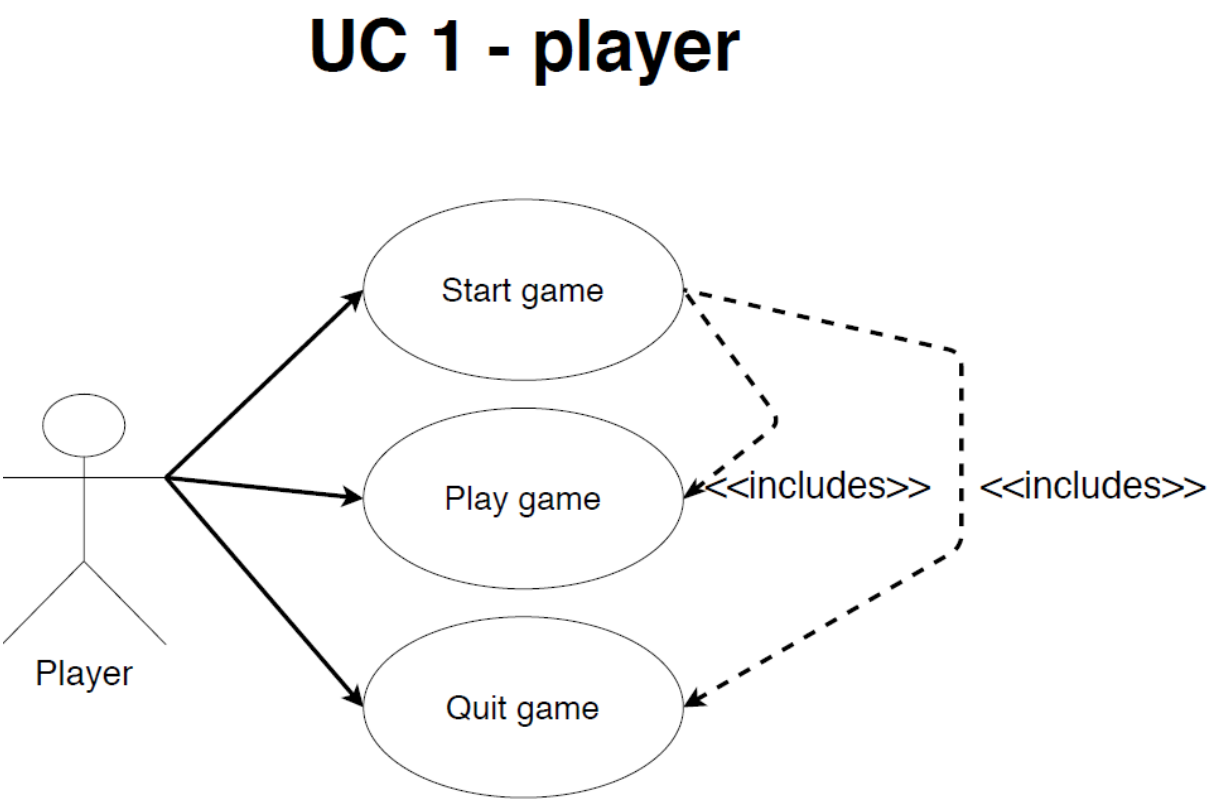
UC 3 Quit the game	
Preconditions:	The game session is running
Post conditions	The game session is terminated
Main Scenario	
1. Starts when the User wants to quit the game, win or lose. 2. The System prompts if the user wants to quit the game. 3. The User confirms. 4. The System terminates.	
Alternative scenarios	
3.1 The User does not confirm. 3.2 The System returns to UC 1 start game	

Check State Diagram



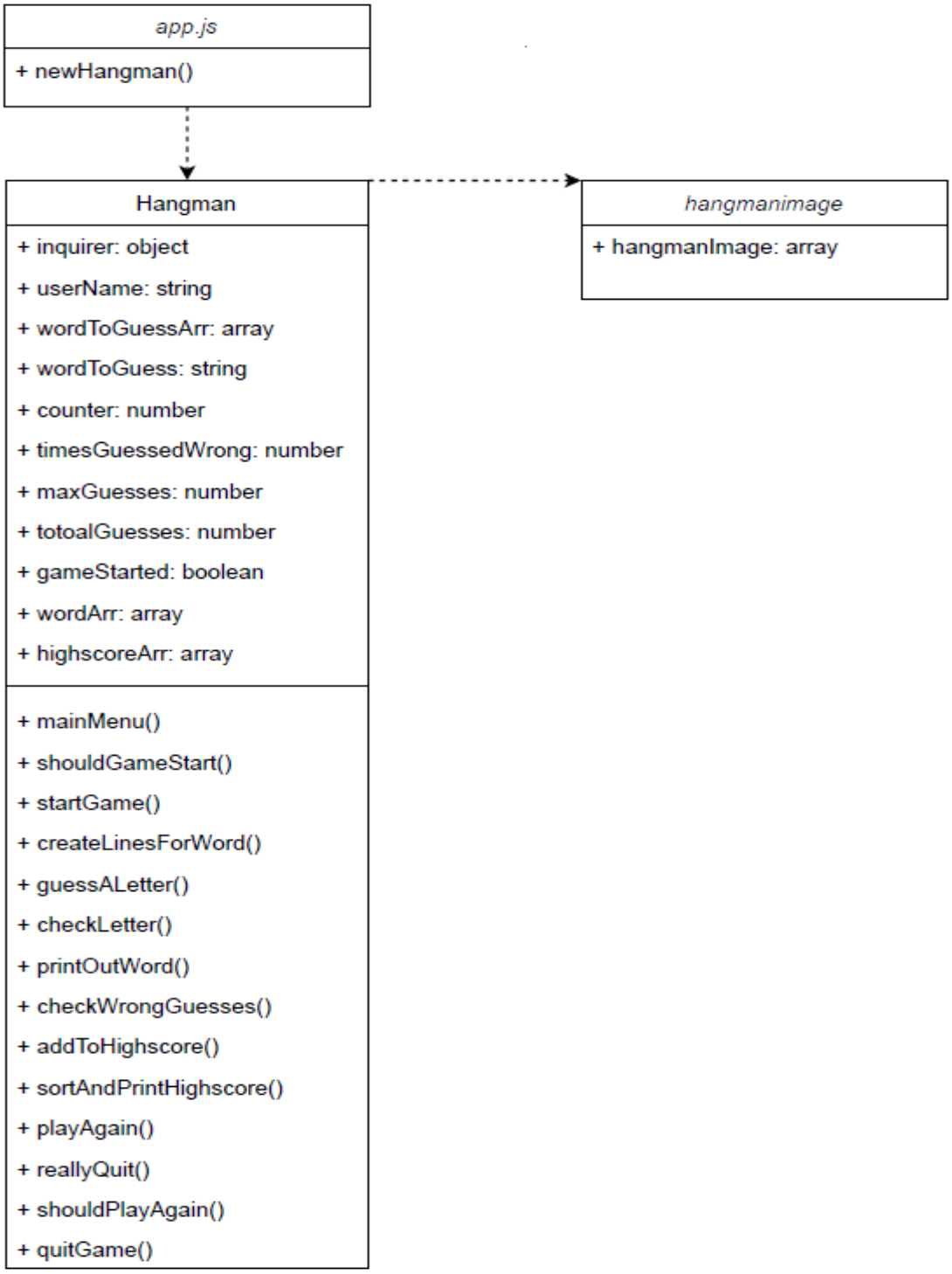
This diagram has been chosen to show the programmers and the final customer the flow and logic of the game. It is an easy way for everyone, not only people with programming experience, to show how the logic in the game works.

User Case Diagram



This diagram is an easy way to show witch User cases are used and how they are connected (includes) to each other.

Class Diagram



This is mostly for the programmers. It shows witch parts of the game belongs to which class/file. Gives a quick overview of the methods and the variables.

8. Test plan for Manual Test Cases and Unit Tests

The methods that will be tested in the manual test is UC2 and the Unit Test will contain “shouldGameStart”, “checkLetter”, “checkWrongGuesses” and “highscore”. All these methods are critical for the game functionality.

Manual Tests:

- The test TC2.1 test the systems for the ability to retrieve username input. It’s chosen because of the high score-implementation, so that a name can be added to the highscore. It will be a manual static test.
- Test case TC2.2 test the error handler if a user does not provide a username and presses enter. The System shall prompt the User to enter a valid username. It will be a manual static test.
- Manual inspection of the code to find bugs.

Unit tests:

- **shouldGameStart** – Tests if the input from the user is the right input to start the game.
- **checkLetter** – Tests if the letter that the user inputs when guessing the letter in the word. Checks if it is the correct letter according to the letter in the word at a given index.
- **checkWrongGuesses** – Checks how many times the user has guessed the wrong letter in the word. If the guesses are exceeding the maximum guesses allowed, the game should prompt that the user has lost the game.
- **highscore** – Checks if the User has guessed more than 2 times and adds the User to the highscore-list.

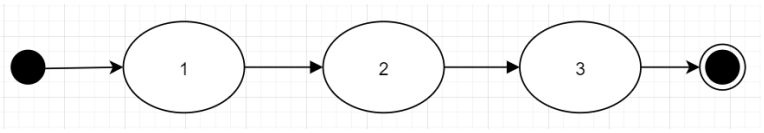
Task	Planned time	Actual time
Manual test Case	1 h	1 h
Writing unit tests	1 h	4 h
Unit testing	30 min	20 min
Running manual tests	1 h	10 min
Code inspection	1 h	40 min
Test report	1 h	30 min

9. Manual Test Cases

TC2.1 Test if username can be entered:

Use case: UC2 play the game.

Scenario: The system will prompt the entered username. The user will enter the name “Johan”. (The user uses the keyboard to navigate)



The main scenario of UC2 is tested if the user can enter a username.

Preconditions:

- The game must be started.
- The user must have chosen to play the game.

Test steps

1. The System prompts the user, “Enter your name”.
2. The user enters the name “Johan” and press enter to confirm the username.
3. The system prints:
   “Welcome Johan”
   “The word is a northern country Europe country”.
   “_ _ _ _” (one “_” for each letter in the word).
   The system prompts: “Guess a letter. To quit, write “quit” and press enter”.

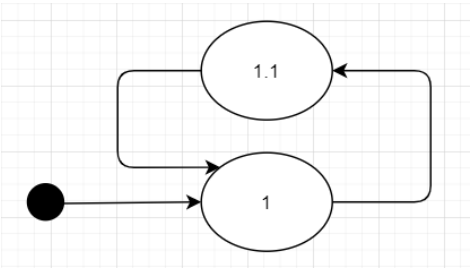
Expected results

- The user is prompted to enter a letter or to write “quit” to quit the game.

TC2.2 Test error-handling if no username is provided:

Use case: UC2 play the game.

Scenario: The System prompts the User to “Enter your name”. The user shall press enter. The system prompts an error message. (The user uses the keyboard to navigate)



The main scenario of UC2 is tested if the systems prompts an error if the user does not enter a username and presses enter.

Preconditions:

- The game must be started.
- The user must have chosen to play the game.

Test steps

1. The System prompts the user “Enter tour name”.
2. The User leaves the input field empty and presses enter.
3. The System prints “Enter a username with at least 1 letter” and returns to step 1.

Expected results

- The system returns to step 1.

10. Unit Tests

```
describe('shouldGameStart', function () {
  let enteredInput = 'Play game'

  it('Should return true if enteredInput is "Play game"', function () {
    assert.isTrue(sut.shouldGameStart(enteredInput))
  })
  it('Should return false if enteredInput is not "Play game"', function () {
    enteredInput = 'Quit'
    assert.isFalse(sut.shouldGameStart(enteredInput))
  })
})

describe('checkLetter', function () {
  let enteredLetter = 'a'
  const givenWordToCheckWith = 'a'

  it('Should return true if enteredLetter is the same as givenLetterToCheckWith', function () {
    assert.isTrue(sut.checkLetter(enteredLetter, givenWordToCheckWith))
  })
  it('Should return false if enteredLetter is not the same as givenLetterToCheckWith', function () {
    enteredLetter = 'b'
    assert.isFalse(sut.checkLetter(enteredLetter, givenWordToCheckWith))
  })
})

describe('checkWrongGuesses', function () {
  let numberOfWrongGuesses = 7
  const maximumNumberOfWrongGuesses = 8

  it('Should return true if numberOfWrongGuesses is a smaller number than maximumNumberOfWrongGuesses', function () {
    assert.isTrue(sut.checkWrongGuesses(numberOfWrongGuesses, maximumNumberOfWrongGuesses))
  })
  it('Should return false if numberOfWrongGuesses is a larger number than maxGuesses', function () {
    numberOfWrongGuesses = 9
    assert.isFalse(sut.checkWrongGuesses(numberOfWrongGuesses, maximumNumberOfWrongGuesses))
  })
})

describe('highscore', function () {
  const enteredName = 'Mats'
  const numberOfTries = 3
  const highscoreArr = []

  it('Should return true if the user has guessed more than 2 times and is added to the highscoreArr.', function () {
    assert.isTrue(sut.addToHighscore(enteredName, numberOfTries, highscoreArr))
    assert.isNotEmpty(highscoreArr)
  })
})
```

11. Unit Test results

```
  shouldGameStart
    √ Should return true if enteredInput is "Play game"
    √ Should return false if enteredInput is not "Play game"

  checkLetter
    √ Should return true if enteredLetter is the same as givenLetterToCheckWith
    √ Should return false if enteredLetter is not the same as givenLetterToCheckWith

  checkWrongGuesses
        |
        |
        |
        |
        |
        |
_______|
7 times remaining to guess
    √ Should return true if numberOfWrongGuesses is a smaller number than maximumNumberOfWrongGuesses
    √ Should return false if numberOfWrongGuesses is a larger number than maxGuesses

  highscore
####### Scoreboard #######
#1 Player: Mats Score: 3
##########################
    √ Should return true if the player has guessed more than 2 times and is added to the highscoreArr.


  7 passing (33ms)
```

12. Test Report

Manual Test Cases	UC2
TC2.1	1/OK
TC2.2	1/OK
COVERAGE AND SUCCESS	2/OK

Unit Test	UC2
shouldGameStart	1/OK
checkLetter	1/OK
checkWrongGuesses	1/OK
highscore	1/OK
COVERAGE AND SUCCESS	4/OK

13. Reflections

Reflections on the Vision

The vision is hard to come up with when having a project that contains such a small game.

To motivate the team that develops the game, and to have a good vision is necessary, also as a selling point to the customer!

The vision should reflect the goal of the development, the things that will make the game special, to make the development teamwork in the same direction for the same goal! That is what I wanted to accomplish with my vision.

The task also included a special feature. I chose to give the game a scoreboard so that the user wants to come back and beat the best player. That is my way of giving the user a “goal” to the game, be the best player.

Reflections on the project plan

The project plan was easier to write than the Vision because the project plan is more how the game is meant to be structured so that the developers of the game and the stakeholders in the project plan knows how the final product shall work. It was harder to find good ways of writing the Scope, Constraints, and Assumptions.

Reflections on the risk analysis

The risks in this project are not many. This project is a small project and the hardware and software are not extensive. The changing of requirements is a risk that is more likely to happen, but that may be because the development team comes up with additional functions to make the game more playable and interesting.

Risk in a project is the hardest thing to plan. It is always hard to know how to avoid risks, but if a project has risk management plans it is less likely that the risk will make much damage to the project and take too long time. It is also easier to re-plan if something happens and the project have a plan-B.

Reflections on Test

I found the Unit test to be the hardest part to write. The first tests I wrote were too simple and only tested if the return was an Array or String. After asking how to make useful tests and how to write better code that could be tested, I wrote better tests that tested easy input and easy output.

The manual test cases were easier to find methods to test because the tester can interact via the console and use the input to control and test the application.

Reflection on final hand-in

In this iteration, the focus has been on fixing the feedback from earlier iterations and completing the Hangman Game.

The completion of the game was not hard, but it took a while to make the final implementations.

The hard part was understanding how to complete the final project and include all other iterations in a good way.