SOFTWAREDEVELOPMENT HANGMAN

Anna Malmgren
Linnéuniversitetet, Software Technology

22/03/2019

Contents

1 | Revision History

Date	Version	Description	Author
08/02/2019	v1.0	Initial version of the documentation.	Anna Malmgren
14/02/2019	v1.1	Updating tasks for iteration 2 and the time log.	Anna Malmgren
8/03/2019	v1.2	Updating tasks for iteration 3 and the time log	Anna Malmgren
21/03/2019	v1.3	1. Rewriting general information: ✓ Improving main client ✓ Improving executive summary. 2. Rewriting vision 3. Rewriting project plan: ✓ Improving jusification ✓ Adding how stakeholders influence the project. ✓ Improving scope, constraints and assumptions. ✓ Iterations as tables instead of lists and adding description to every task. 4. Rewriting risk analyse to better suit the project. 5. Updating iteration 4 and time log. 6. Adding modelling diagrams. 7. Adding test plan.	Anna Malmgren

2 | General Information

Project Summary	
Project Name	Project ID
Hangman	am223rj _1DV600
Project Manager	Main Client
Anna Malmgren	Elementary school teachers – using the game for testing knowledge in students. Elementary school students – playing the game.
Key Stakeholders	
The end users / main client, project manager, developer , organization owning the project.	
Executive Summary	
Hangman is an application for making learning fun. It should motivate elementary school students to want to learn by providing a fun way of testing their knowledge. The player is presented with a representation of a word to be guessed and has a pre-decided number of guesses to guess the word. If the player guesses the word before he/she is out of guesses the player wins and possibly gets in the high-score list. Otherwise the player loses.	

3 | Vision

Hangman is a platform developed for fun learning and creates an exciting learning environment where only your fantasy sets the limits. The goal is to motivate elementary school students to study by providing a fun game to check their knowledge. Adding words makes it easy to test different subjects or themes and high score lists adds motivation to learn and improve.
This should make it easy for teachers to create different tests and also freeing time for the teachers by the test correcting itself.
The basic idea of Hangman is that the player is going to guess a word by suggesting letter after letter. The player is presented with the number of letters in the word represented by underscore signs and a clue. For every wrong guess the game is building a part of a man getting hanged. The number of wrongs that the player can have is depending on how many parts the hanged man consists of. If the man gets hanged, the player loses. If the player guesses the word before the man gets hanged, the player wins and if the scores is good enough gets a place in the high score list.

Reflections:

The hardest part of writing the vison for me was trying make the vision motivating and inspiring for others. I am more a matter of fact person so what's inspiring to me may not be so inspiring to most others. I'm still not sure if this vision is motivating enough but I tried to find a balance between keeping the projects direction clear but still have some abstract parts so one can form new ideas within the path of the project.

4 | Project Plan

Reflections:
To me the Project plan had many difficult parts. The first was to try to understand what I was supposed to write under the different titles. I hope I got at least most parts ok. Next issue was that I realized that some of my different sections have very similar content which might be wrong, I tried to fix it but didn't entirely succeed. My favorite part of the project plan was dividing the assignments in the iterations to smaller tasks and try to find a good balance in size and number.

4.1 Introduction

Hangman is a console application where the player guesses a word by suggesting letter after letter. The player shall be greeted when starting the game and when beginning her or she is presented with a word from a predefined list of nouns that is randomly picked. The number of letters is displayed with equally many underscore signs and a short clue connected to the word. The number of wrong guesses that the player can have is depending on how many parts the man to be hanged is drawn with. On every wrong guess a part of the hanged man is drawn. The hanging man is build using the available characters on the keyboard. If to many wrong guesses are made the man is hanged and the game is over. If the player guesses right and the letter is included in the word the underscore sign/signs where the letter is located is replaced with the guessed letter. If the player can guess the word the result is evaluated to see if he/she made it into the high score list and the player gets the option to view the high score list.

The project shall be executed in four iterations described in more detail in section five, Iterations. This project goes on for 9 weeks and the final product shall be done at latest on Friday 22 of mars at 23:55.

4.2 Justification

Hangman is made to be an alternative way for elementary school teachers to test their students' knowledge by using a game instead of a regular test. The application should motivate the students to study by providing a fun way for them to show their knowledge and competing to get into the high score list.
It should also be an easy way for the teacher to create tests for the students by just adding words to the application and let the application correct the answers.

4.3 Stakeholders

- Main client/end users:
    - The teachers want an application that is easy to understand and work with.
    - The students want an application that is fun and exciting to play.
- Developer wants the code to be correct, testable, maintainable and readable.
- Project Manager wants the project to stay on time and documentation to be updated and clear.
- Organization owning the project wants the end product to stay on time and meet the expectations and requirements of the main clients.

4.4 Resources

Time: 9 weeks, 20 h per week.
Literature: Software Engineering by Ian Sommerville
Training: Tutoring sessions, pre-recorder lectures and live lectures.

4.5 Hard- and Software Requirements

Software: IDE used is visual studio code, Node version 10.9.0, npm version 6.2.0 and the terminal git bash `2.18.0.windows.1. The OS used during development is Windows 10 Education.`
Hardware: Processor used during development is Intel(R) Core™ i7-8750H.

4.6 Overall Project Schedule

Iteration	Deadline
1	Friday, 8th of February 2019, 23:55
2	Thursday, 21st of February 2019, 12:00
3	Friday, 8th of Mars 2019, 23:55
4	Friday 22nd of Mars 2019, 23:55

4.7 Scope, Constraints and Assumptions

Hangman is developed on the Node.js platform as a console application. The basics of the application is that it shall contain functionality for picking words from a predefined list of nouns. Displaying the words to the player with underscore signs representing every letter in the word. The player shall be able to guess letter after letter. If the letter guessed exists in the word the underscore sign is exchanged for the letter else a part of the hangman is drawn. The hangman is build using the available characters on the keyboard. If every part of the hangman is drawn the game is over and the player should be able to start over. Else if the word is guessed before the hanged man gets fully drawn the game is won.
Extra features to be included are that the system before starting the game should prompt the player for a nickname, the application should also include a high score list shown if the player wins and a timer taking the time it takes to finish the game.
A file with some words and clues about IT/programming is included and it should be possible to add more words to that file.

Outside the scope of this project is connecting the application to a database for saving words and creating a proper log in functionality. That is mainly due to the time constraints and lack of knowledge about creating data bases.

The constraints for the project are the time that is limited to 20 hours a week for nine weeks.
The project only has one developer that also have the role of project manager. That means that the experience, knowledge and skills in this project is limited to one person.
The constraints for the application are that it is a console application created with JavaScript. JavaScript isn't optimal for console applications but was chosen because it is the only language the developer has worked with. Being a console application also limits the user interface design but makes for easier testability.

It's assumed that the user is comfortable with console applications and has access to roughly the same software and hardware that was used in the development for this application and are defined is section 4.5.

5 | Iterations

5.1 Iteration 1

Due date: Friday, 8th of February 2019, 23:55

Task	Description	Time estimate
Add a time log	Estimate time for each task in iteration 1	30 min
Write general information	Project summary for Hangman application: • Project name and ID • Project Manager, main client and key stakeholders • Executive Summary	1 h
Write vison	The vision for Hangman application	1 h
Write project plan	Project plan for Hangman application including: • Introduction • Justification • The stakeholders • Resources • Hard and software requirements • Overall schedule • Scope, constrains and assumptions • Iterations planning.	3 h
Write risk analyze	Identify risks for hangman and come up with strategies for handling the risks.	2 h
Create GitHub repository	Repository for Hangman on GitHub with README.md, package.json and .gitignore file.	20 min
Add npm packages	Add packages standard and snazzy for dev dependencies.	15 min
Update time log	Complete time log for iteration 1 with actual time for the tasks and an analysis.	1 h

5.2 Iteration 2

Due date: Thursday, 21st of February 2019, 12:00

Task	Description	Time estimate
Update iteration 2	Update iteration 2 in project plan.	45 min
Add time log	Estimate time for each task in iteration 2	40 min
Create use case diagram	UML use case diagram for hangman application.	5 h
Create fully dressed use case	Fully dressed use case for “Play Game”	5 h
Create basic state machine	UML State Machine for “Play Game”	6 h
Implement basic hangman functionality.	Write the code for basic version of hangman. Functionality should include • Start game – display main menu. • Get a random word and display it as underscore signs instead of letters together with a clue. • Play game – prompt player for letter, check letter and display result for the user. • Display if the game is won or lost • Quit game.	10 h
Create class diagram	UML class diagram for the implemented code.	6h
Create extended use case	Fully dressed use case of “High Score”	3h
Create extended State Machine	UML State Machine diagram for the whole hangman application.	3h
Update time log	Complete time log for iteration 2 with actual time for the tasks and an analysis.	45 min

5.3 Iteration 3

Due date: Friday, 8th of Mars 2019, 23:55

Task	Description	Time estimate
Update iteration 3	Update iteration 3 in project plan.	1 h
Add time log	Estimate time for each task in iteration 3	30 min
Write test plan	Write a test plan containing: • Objectives • What to test and how • Time plan	4h
Write unit tests	Write unit tests for the code implemented in iteration 2.	8h
Update test plan with unit tests.	Add screenshots of the unit tests to the test plan and screenshots of the result and coverage.	2 h
Write manual test cases	Write manual test cases for UC1 “Start Game”, UC2 “Play Game” and UC4 “Quit Game” in the test plan.	8h
Execute manual test cases	Execute the written manual tests cases	30 min
Write test report	Write report of the result of the manual test cases in the test plan.	30 min
Update time log	Complete time log for iteration 3 with actual time for the tasks and an analysis.	30 min

5.4 Iteration 4

Due date : Friday 22nd of Mars 2019, 00:00

Task	Description	Time estimate
Update Iteration 4	Update iteration 4 in project plan.	1 h
Add time log	Estimate time for each task in iteration 4	30 min
Rewrite general information	Improve main client and executive summary.	30 min
Rewrite vision	Improve the vision by being clearer about who the main client is.	1 h
Rewrite project plan	- Rewrite justification with a clear target for the game. - Add how different stakeholders influence the project. - Add what is out of scope for the production and improve the constrains. - Change layout for iteration form lists to table and add description.	3 h
Rewrite risks and strategies	Update risks and strategies to what feels relevant	1 h
Update use case diagram	Check if the use case diagram is accurate, write documentation if something is not.	2 h
Update use cases	Add new features, prompt for nickname and time taking in play game use case.	2 h
Make a state machine diagram	Create a state machine representing the whole game.	1 h
Implement drawing hangman	Functionality for drawing a piece of the man to be hanged with every wrong answer.	3 h
Implement nickname	System prompts player for nickname before starting the game.	1 h
Implement high score	Functionality that saves high score list for the game. And the ability to display the high score list after won game.	8 h
Implement time taker.	Takes the time it takes to finish the game. Should be able to start and stop time taking.	1 h
Write test plan	Write a test plan containing: • Objectives • What to test and how • Time plan	2 h
Add manual test cases	Add manual test cases to cover all use cases main scenarios and alternative scenarios	3 h
Execute manual test cases	Execute all the manual test cases and document the results	1 h
Add test result to the test plan	Add a table with the test results to the test plan	30 min
Implement unit tests	Add unit test to test the methods in the hangman application.	8 h
Add unit tests and result to the test report	Add the methods and the results to the test report.	2 h

Update time log	Update time log with actual time for each task.	30 min

6 | Risk Analysis

Reflections:
It's quite easy finding risks but judging the probability and impact is according to me more difficult. I'm still not quite sure if I've labeled every risk right. When it came to strategies I also had a problem, it felt ok coming up with strategies but how do I know that the strategies I came up with will work? Should one have back up strategies for the strategies?

6.1 List of risks

Risk	Probability	Impact
The size of the software is underestimated.	High	Tolerable
Developer/ project manager is ill during critical times in the project.	Moderate	Tolerable
The time required to develop the software is underestimated.	High	Serious
Developer / project manager does not have the skills required.	Moderate	Serious
Insufficient documentation	High	Serious
Misunderstanding of requirements that require major rework.	Moderate	Tolerable
The time testing takes is underestimated	High	Serious
Application does not meet the requirements when the deadline has passed.	Moderate	Catastrophic

6.2 Strategies

Risk	Strategy
Size underestimated	Careful planning for each task. Time estimates for every task complemented with the actual time for improving ability to estimate.
Developer / project manager ill.	Developer can work from home.
Time underestimated	Schedule every task, estimate the time and add time to the estimate for eventual problems.
Developer / project manager lacks in skills	Read the literature “Software Engineering”, attend the lectures, sessions and tutoring.
Insufficient documentation	Read “Software Engineering”, attend tutoring and ask questions on slack. Reread and update the documentation every iteration.
Misunderstanding requirements.	Thoroughly read the specified requirements at least two times before starting iteration and before handing in deliverables .
Testing time	Estimate time for testing and add 30% extra time to the estimate.
Does not meet requirements	Thoroughly read the specified requirements and make sure every single one is planned for in the iterations and test the application for every requirement.

Time log

7.1 Time Log

Iteration 1

Task	Time estimate	Actual time
Create time log	30 min	3 h
Write general information	1 h	2 h
Write vision	2 h	3 h
Write project plan	3 h	10 h
Write risk analyze	2 h	5 h
Create GitHub repository	20 min	15 min
Add npm packages	15 min	20 min
Update time log	1 h	45 min

Iteration 2

Task	Time estimate	Actual time
Update iteration 2	45 min	1 h
Create time log	40 min	30 min
Create use case diagram	5 h	5 h
Create fully dressed use case	5 h	5 h
Create basic state Machine diagram	6 h	5 h 45 min
Implement code for basic hangman application	10 h	14 h
Create class diagram	6 h	4 h
Create extended version of the fully dressed use case	3 h	2 h
Create extended version of state machine diagram	3 h	3 h
Update time log	45 min	30 min

Iteration 3

Task	Time estimate	Actual time
Update iteration 3	1 h	1 h
Create time log	30 min	30 min
Write test plan	4 h	4 h
Write unit tests	8 h	11 h
Update test plan with unit tests	2 h	2 h
Write manual test cases	8 h	9 h
Execute manual test cases	30 min	30 min
Write test report	30 min	30 min
Update time log	30 min	30 min

Iteration 4

Task	Time estimate	Actual time
Update Iteration 4	1 h	1 h
Add time log	30 min	20 min
Rewrite general information	30 min	45 min
Rewrite vision	1 h	2 h
Rewrite project plan	3 h	3 h
Rewrite risks and strategies	1 h	45 min
Update use case diagram	2 h	2 h
Update use cases	2 h	1 h 30 min
Make a state machine diagram	1 h	4 h
Implement drawing hangman	3 h	2 h
Implement nickname	1 h	1 h
Implement high score	8 h	7 h
Implement time taker.	1 h	1 h
Write test plan	2 h	1 h
Add manual test cases	3 h	3 h
Execute manual test cases	1 h	1 h
Add test result to the test plan	30 min	15 min
Implement unit tests	8 h	7 h
Add unit test result to the test report	15 m	15 min
Update time log	30 min	30 min

7.2 Analyzes

Iteration 1

For the first iteration the actual time is unfortunately not as accurate as it should be. That is because of bad planning in the beginning, starting on multiple things at the same time and forgetting to clock the reading and learning part. Trying to minimize the damage the overall time for learning is divided and added to the tasks it's affecting. It became clear that the time needed for learning was very underestimated and will be more considered ahead.

Iteration 2

The time log for the second iteration is better than the one for the first iteration. The estimates are often closer to the actual time but there is still room for improvement. The implementation took a bit longer than expected mostly due to a misunderstanding of the requirements. That means more time should be estimated to the implementation parts.

Iteration 3

In this iteration most of the estimates were good. It is still the implementation part (in this case write unit tests) that is the worst estimate. So, more time for implementation still needs improvement.

Iteration 4

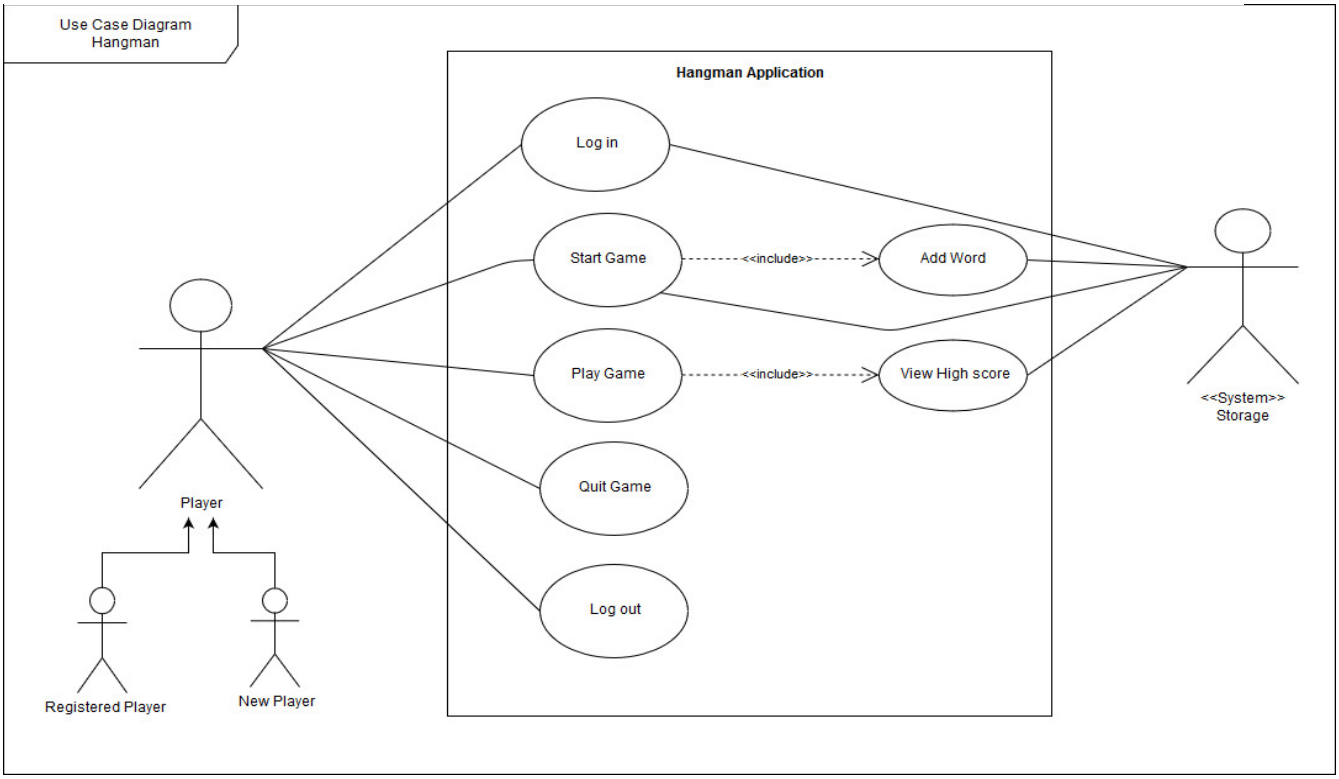
I am quite pleased with my estimates in this iteration. I feel that I've improved a bit since the start of the project. It is the estimate of drawing a state machine that is the worst one. That is because I made the diagram before I got the feedback from iteration 2 and when I got the feedback I realized I had to redo it again plus changing some things in the basic state machine.

1 | Use Case Diagram

The first use case diagram is the first developed for the project. After some consideration the Log in functionality and external storage system had to be removed from this scope due to lack of time and knowledge. The second use case diagram is the one that applies to the scope of this project.

1.1 Use Case Diagram 1 (out of scope)

FIGURE 1 USE CASE DIAGRAM NO LONGER IN THE SCOPE OF THIS PROJECT



1.2 Use Case Diagram 2 (updated version)

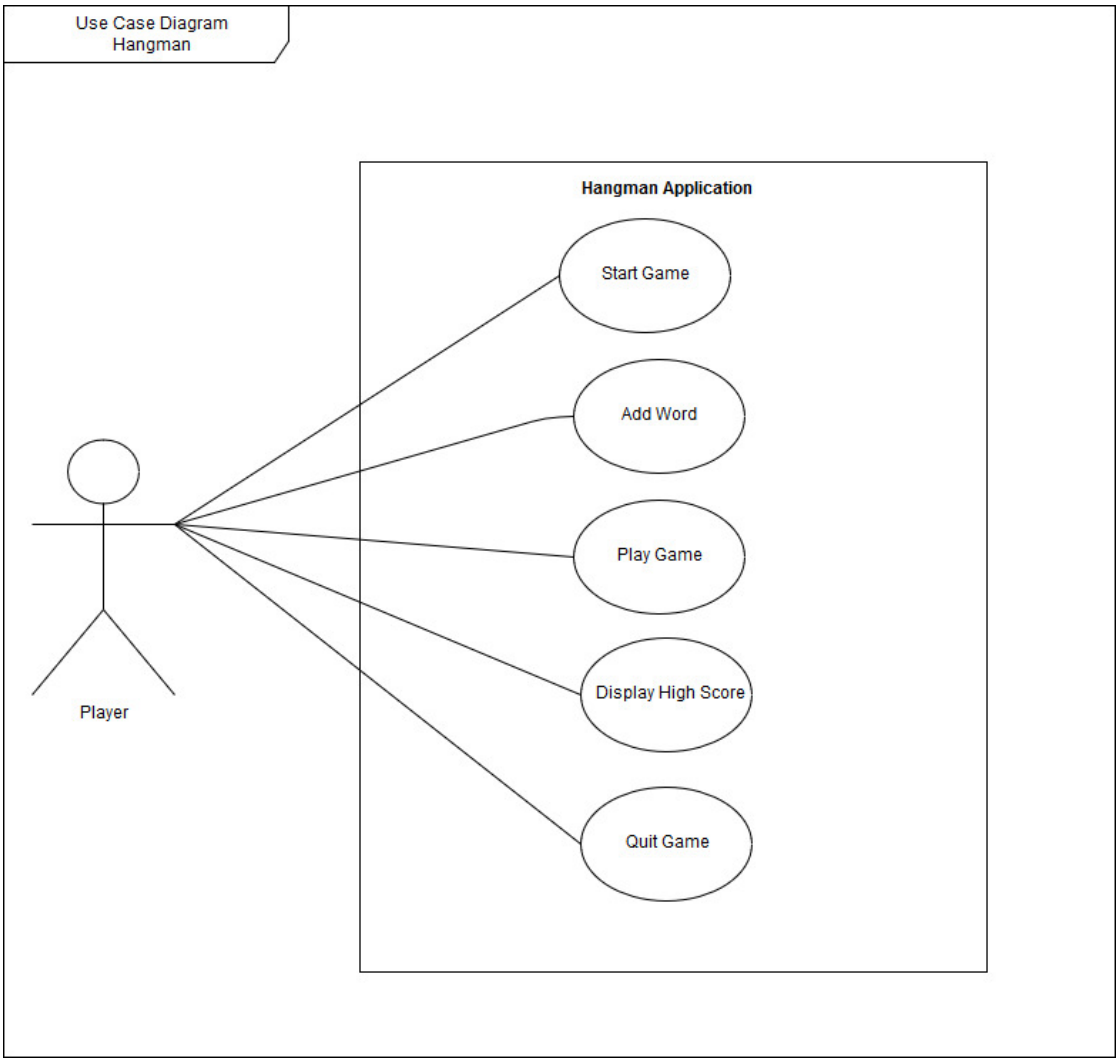


FIGURE 2 UPDATED USE CASE DIAGRAM FOR THIS PROJECT

| Fully dressed Use Cases

2.1 UC 1 Start Game

Precondition: none.

Postcondition: The game menu is shown.

Main scenario

1. Starts when the player wants to begin a session of the hangman game.
2. The system greets the player and prompts for a nickname
3. The player enters a nickname.
4. The system presents the main menu with the option to start game, quit or add word to the game.
5. The player makes the choice to start the game.
6. The system starts the game (see Use Case 3).

Alternative scenarios

3.1 The player doesn't enter a nickname.

1. The system displays an error message. Go to step 2.

5.1 The player makes the choice to quit the game.

1. The system quits the game (see Use Case 5)

5.2 The player makes the choice to add word to the game.

1. see Use Case 2 Add Word.

2.2 UC 2 Add Word

Precondition: The main menu is displayed.

Postcondition: A word is added, and the main menu is displayed.

Main scenario

1. Starts when the user wants to add a word to the game.
2. The system prompts the player for a word.
3. The player enters a word.
4. The system prompts the player for a clue.
5. The player enters a clue.
6. The system displays a success message for saving the word and displays the main menu, UC 1 step 4.

2.3 UC 3 Play Game

Precondition: The game is started.

Postcondition: The game presents option to see high score list (see Use Case 4).

Main scenario

1. Starts when the player has started the game.
2. The system presents a representation of the word to be guessed and prompts the player for a letter.
3. The player guesses the right letter.
4. The system presents the letter at the right position/positions.
5. The system prompts the player for a letter.
6. The player guesses the right last letter.
7. The system declares a win for the player.
8. The system presents option to see high score list (see Use Case 4).

Alternative scenarios

3.1 The player guesses the wrong letter.

1. The system draws a part of the picture of the hanged man.
2. Go to step 5.

6.1 The player guesses the last guess wrong

1. The system declares game over and the hanged man is fully drawn.
2. Go to Use Case 1 step 4.

2.4 UC 4 High Score

Precondition: The player has played a round of hangman (Use Case 3 main scenario)

Postcondition: The system presents the high score list.

Main scenario

1. Starts when the player has played a round of hangman (Use Case 3)
2. The system prompts the user if the high score list should be displayed.
3. The player accepts.
4. The system displays the list of high scores.
5. The system presents the main menu (see Use Case 1 step 4)

Alternative scenarios

3.1 The user denies

1. Go to step 5.

2.5 UC 5 Quite Game

Precondition: The game is running.

Postcondition: The game is terminated.

Main scenario

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

Alternative scenarios

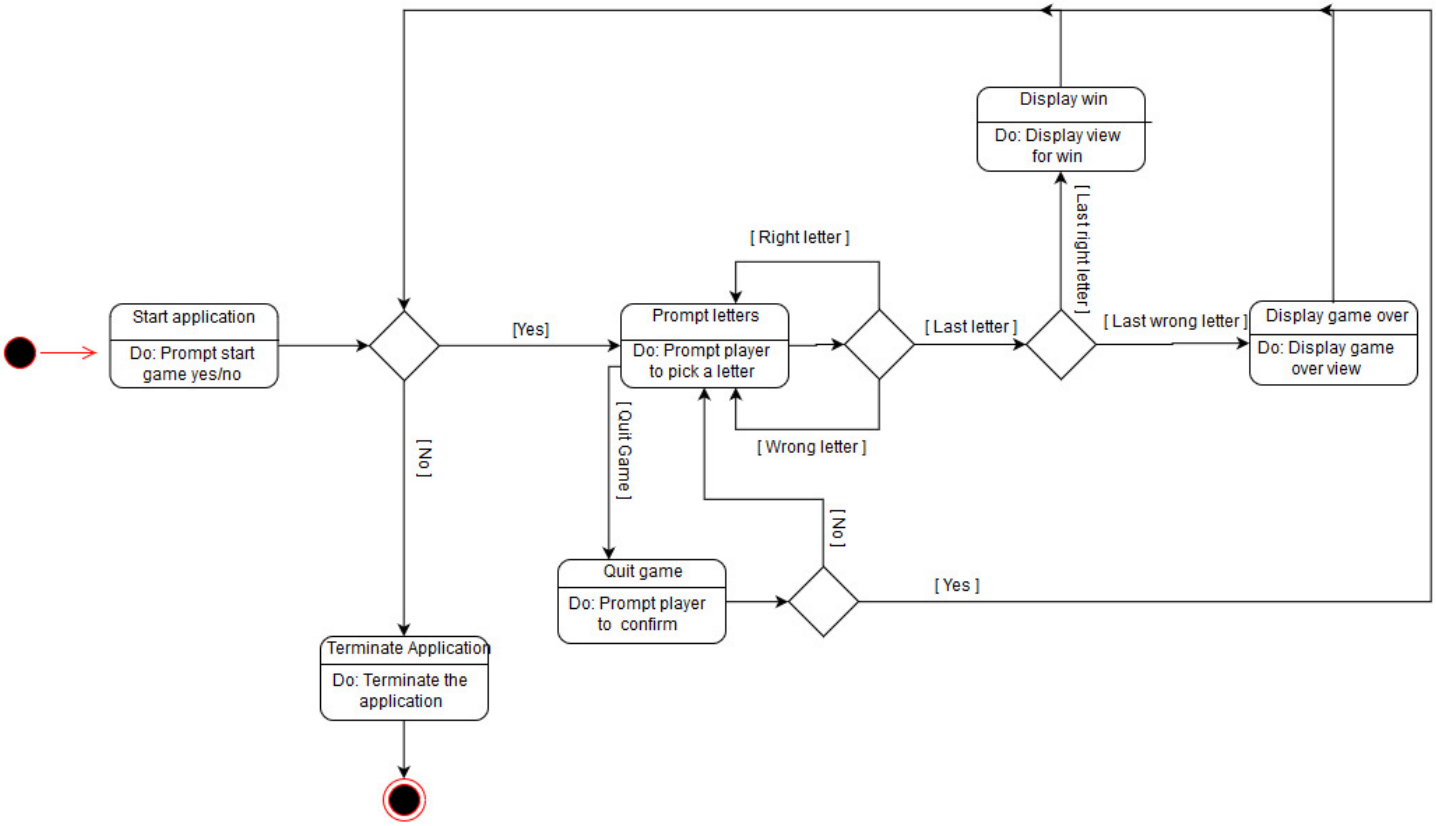
3.1. The player does not confirm

1. The system returns to its previous state

3 | State Machine Diagram

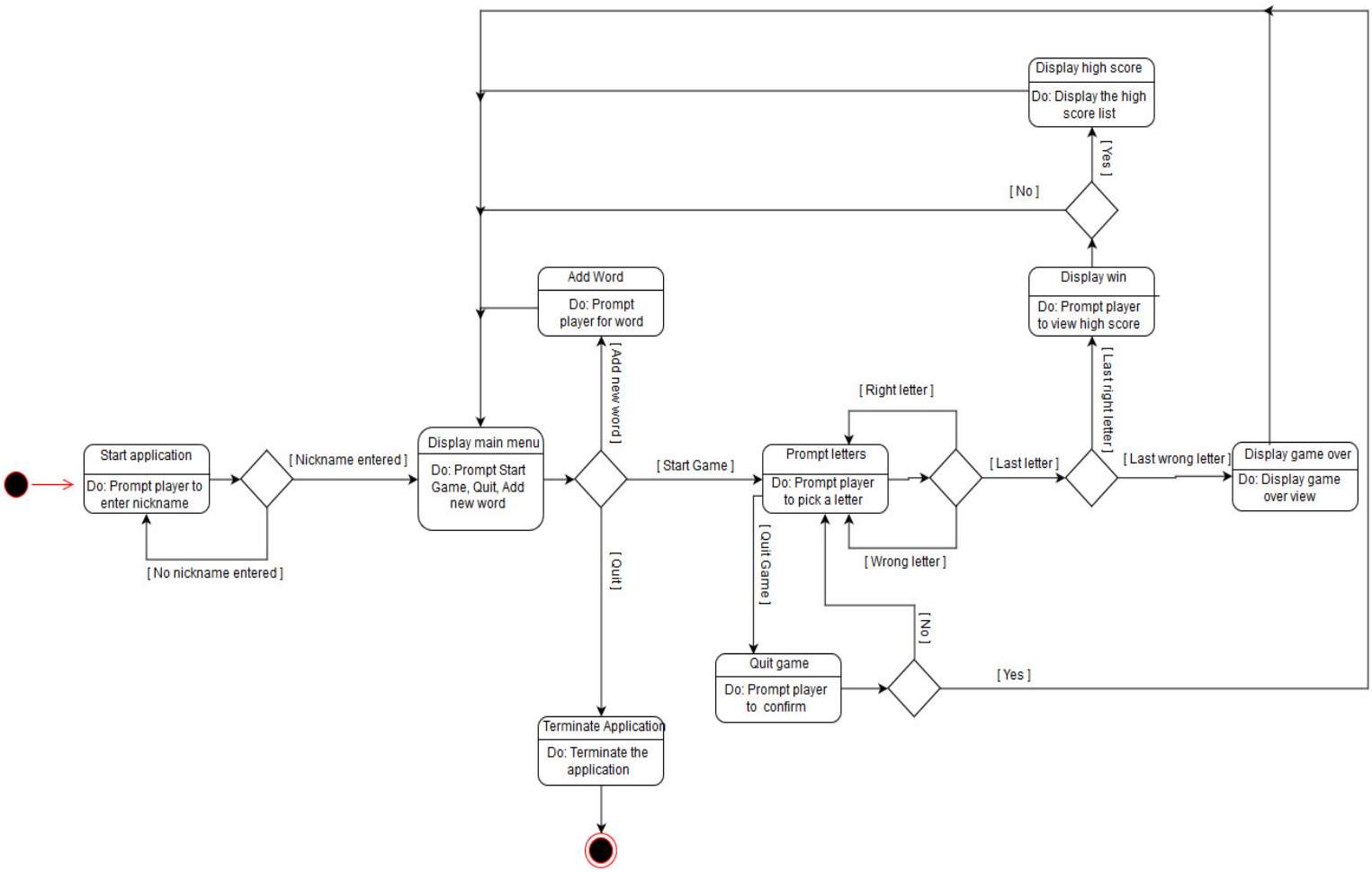
3.1 Basic State Machine

Basic functionality of hangman application.

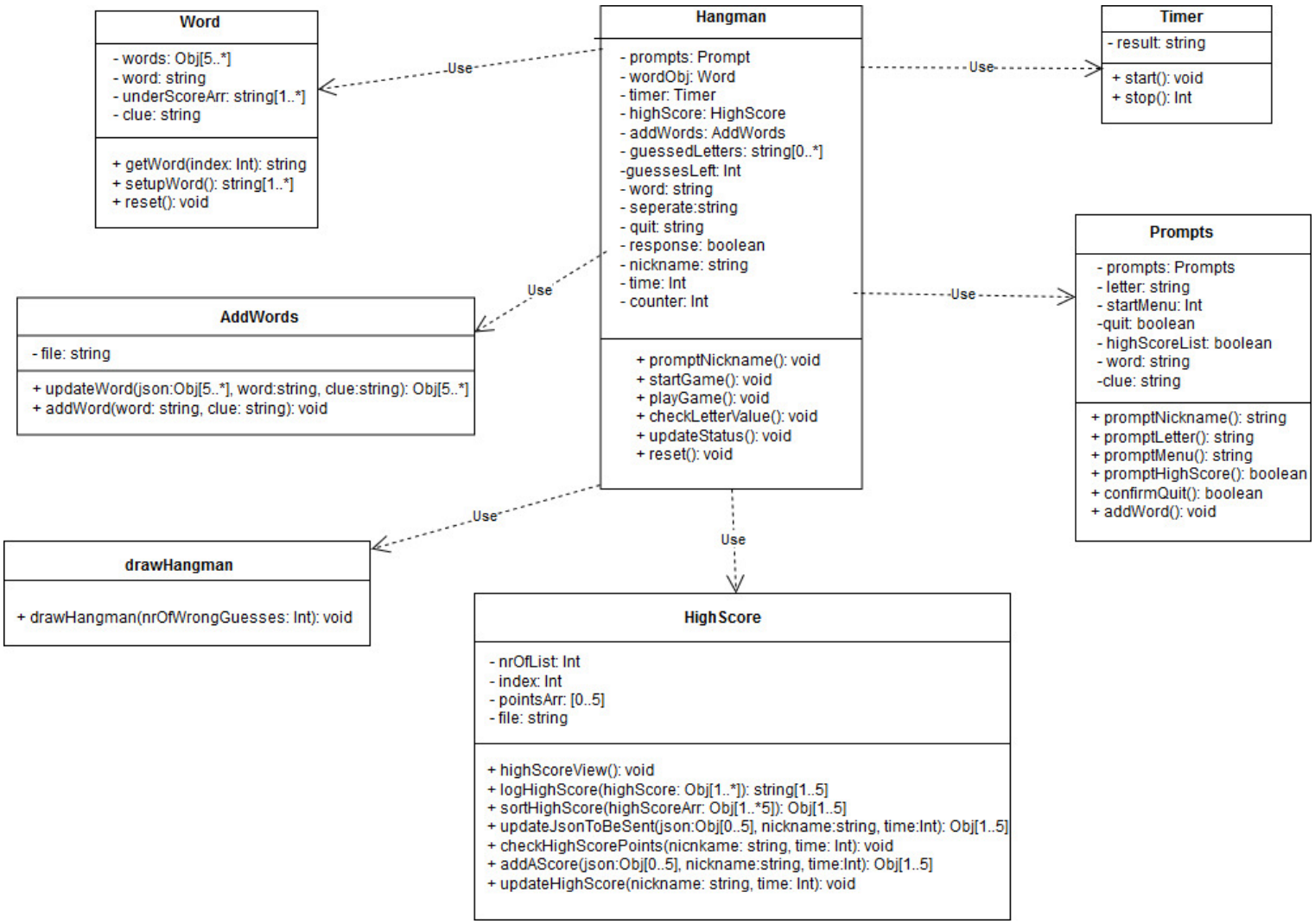


3.2Extended State Machine

Complete hangman application.



4 | Class Diagram



Test Plan

Objective

The objective is to test the code that has been implemented in all the iterations.

What to test and how

The code shall go through a static code inspection so that errors and bad written code has a chance to be found as early as possible and the code quality gets improved.
All fully dressed use cases, UC 1 Start Game, UC2 Add Word, UC3 Play Game, UC4 High Score and UC5 Quit Game, shall be tested by writing and running dynamic manual test cases. There shall be at least on test case per scenario in the use case, main scenario and alternative scenarios, this to cover the requirements of the application.
Automated unit tests for the implemented code shall be written. The testing framework used is "jest", it was chosen because it works with node.js and because of its simplicity. The code to be tested should mainly be logic written by the developer and not third-party modules. There shall be at least two unit tests per method tested. The automated unit tests shall add confidence to the code written by the developer and together with the manual test cases it should give a good enough coverage.

Time plan

Task	Estimate	Actual
Manual test cases	3 h	3 h
Unit Tests	8 h	7 h
Running manual test cases	1 h	1 h
Static code inspection	2 h	3 h
Test Report	15 min	15 min

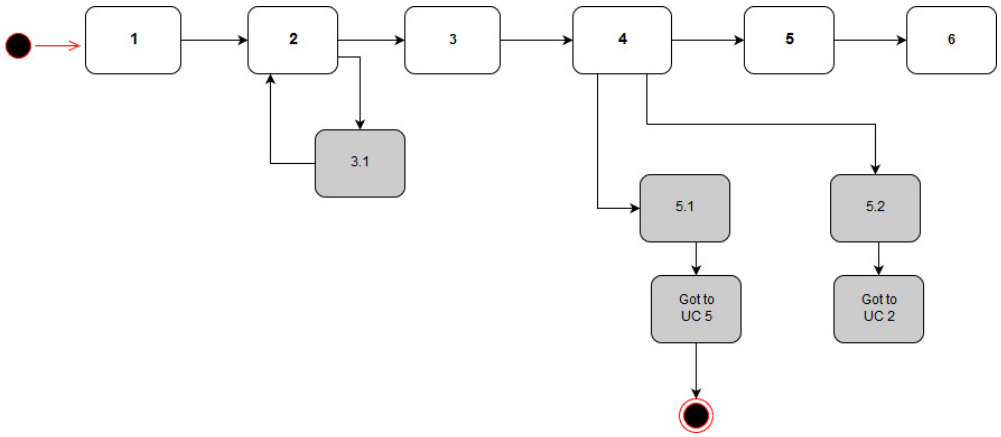
Manual Test Cases

The results of the manual test cases are to be found under the heading “Test Report Manual Test Cases”. The greyed-out steps are the steps in the use case not covered in the test case.

TC 1.1 Start game successfully

Use Case: UC 1 Start Game

Scenario: The main scenario of UC 1 where a user starts the game successfully.



Precondition: None

Test steps:

1. Start app by entering “npm run testEnv” in the terminal and press enter.
2. The system prompts “Enter your nickname”
3. Enter “Lisa” and press enter.
4. The system prompts “What do you want to do?” with options “Start Game”, “Quit” and “Add a word to the game”.
5. “Start Game” is highlighted and marked with “>”. Press Enter.

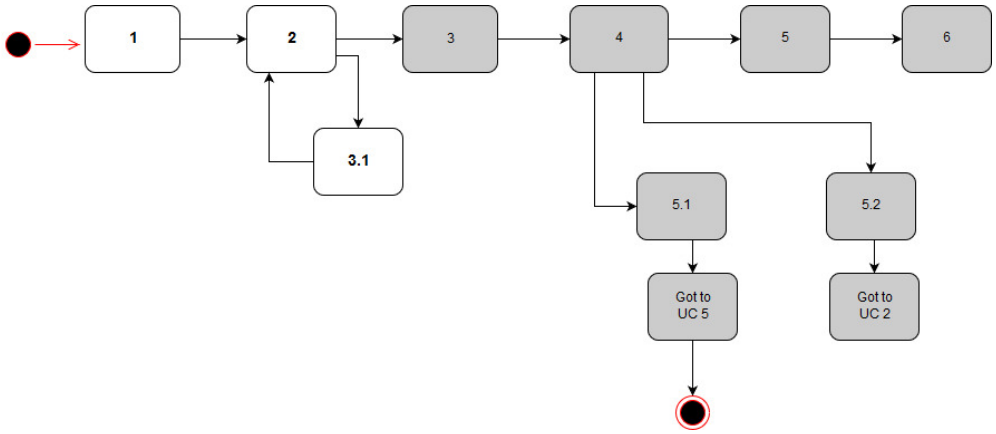
Expected: System displays text :
“Welcome Lisa”
“Your clue: A programming language”
“_ _ _ _ _”

The system prompts: “Enter a letter”

TC 1.2 Don’t enter a nickname.

Use Case: UC 1 Start Game

Scenario: Alternative scenario of UC 1 where user doesn’t enter a nickname.



Precondition: Do TC 1.1 step 1 and 2.

Test steps:

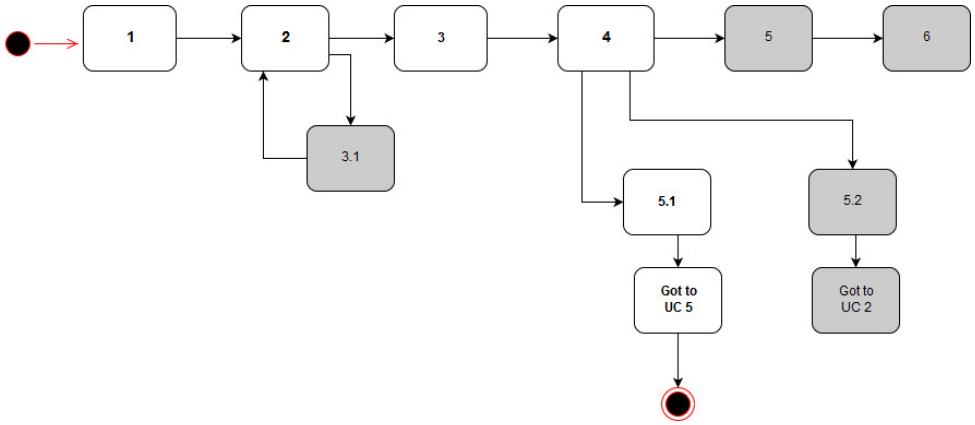
1. Press enter.

Expected: System displays text : “You need to enter a nickname.”
The system prompts: “Enter your nickname”

TC 1.3 Don’t start game

Use Case: UC 1 Start game

Scenario: Alternative scenario of UC 1 where user chooses not to start the game.



Precondition: Do TC 1.1 step 1, 2 and 3.

Test steps:

1. Press page down arrow on keyboard once.
2. System highlights “Quit” and marks the option with “>”
3. Press enter

Expected: System displays text:
“Welcome Lisa”
“Shutting down app... OK”
The system shuts down the app.

TC 2.1 Add Word

Use Case: UC 2 Add Word.

Scenario: Main scenario of UC 2 add word where the user adds a word to the game.

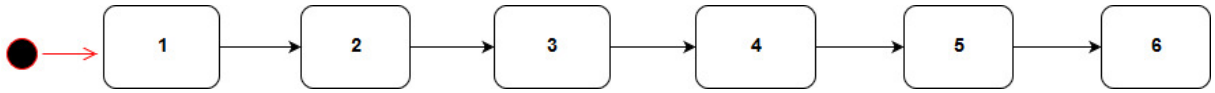


FIGURE 3 THE FLOW IN UC 2

Precondition: First open the folder “data” then open the folder “testFiles” and click on the file “testword.json”. Control that “testword.json” only contain: “[{“word”: “java”, “clue”: “A programming language”}]” and save (ctrl s). If “testword.json” contains anything else, delete it. Do TC 1.1 step 1, 2, 3 and 4.

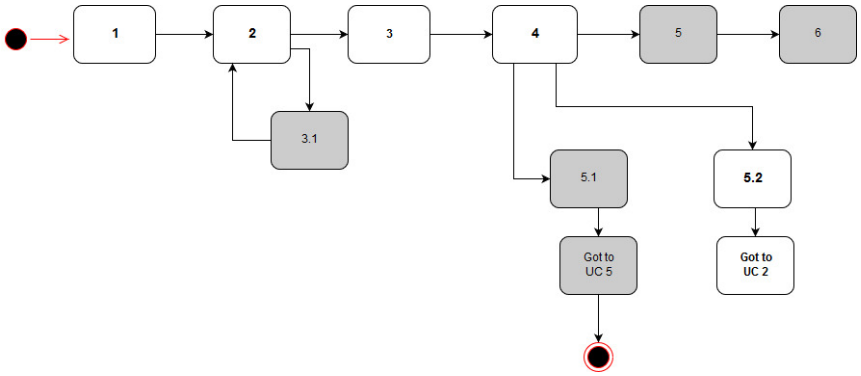


FIGURE 4 THE FLOW IN IN THE PRECONDITION (UC 1)

Test steps:

1. Press page down arrow on keyboard twice.
2. System highlights “Add a word to the game” and marks the option with “>”
3. Press enter.
4. System prompts “What word do you want to add?”
5. Type word “Wolf” and press enter.
6. System prompts “Add a clue for that word”
7. Type “Howling animal” and press enter.

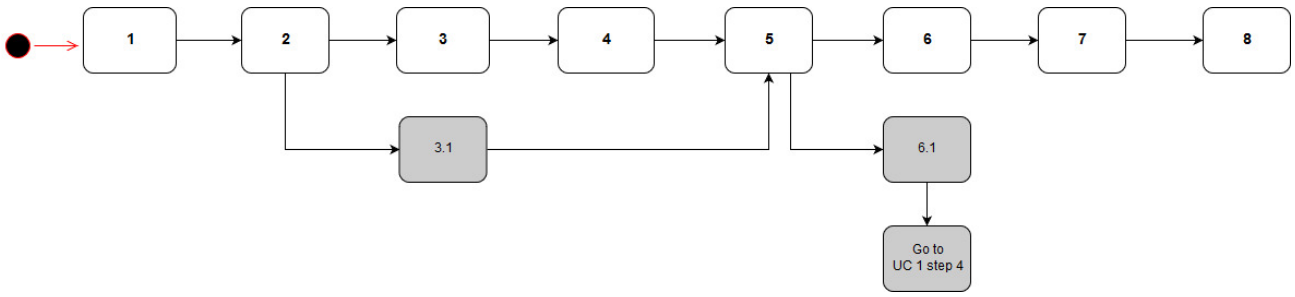
Expected: System prompts “What do you want to do?” with options:
“Start Game”,
“Quit”,
“Add word to the game”
System displays text : “Word added successfully.”

Open folder “data” then open folder “testFiles” and click on the file “testword.json”. “testword.json” contains “[{“word”: “java”, “clue”: “A programming language”}, {“word”: “Wolf”, “clue”: “Howling animal”}]”.

TC 3.1 Play Game successfully

Use Case: UC 3 Play Game

Scenario: The main scenario of UC 3 where user guesses the right letters until the word is guessed and the game is won.



Precondition: Open the folder data then open the folder testFiles and click on the file testHighscore.json. Remove the fifth object “,{“name”: “Lisa”, “points”: (contains a number)}” if it exists and save (ctrl s). Do all steps in TC 1.1, the system prompts “Enter a letter”.

Test steps:

1. Enter the letter ”j” and press enter.
2. System prompts “Enter a letter”
3. Enter the letter “a” and press enter
4. System prompts “Enter a letter”
5. Enter the letter “v”

Expected:

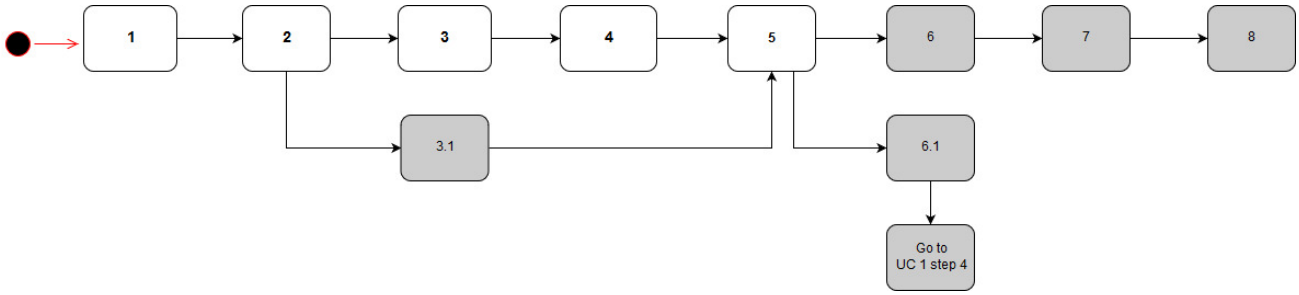
System displays texts:
“ _ _ _ _ _ _ _ _ _ _ _ _ _ _ ”
“Clue: A programming language”
“j a v a”
“Guesses left: 9”
“Guessed Letters: j, a, v”
“To terminate write .exit and press enter”
“You guessed the word “java”, congratulations you win!”
“**********”

The system prompts “Do you want to view the high score?” (Y/n).

TC 3.2 Play Game guess right letter

Use Case: UC 3 Play Game

Scenario: A piece of the main scenario where the user guesses one right letter.



Precondition: Do all steps in TC 1.1, the system prompts “Enter a letter”

Test steps:

1. Enter the letter “a” and press enter.

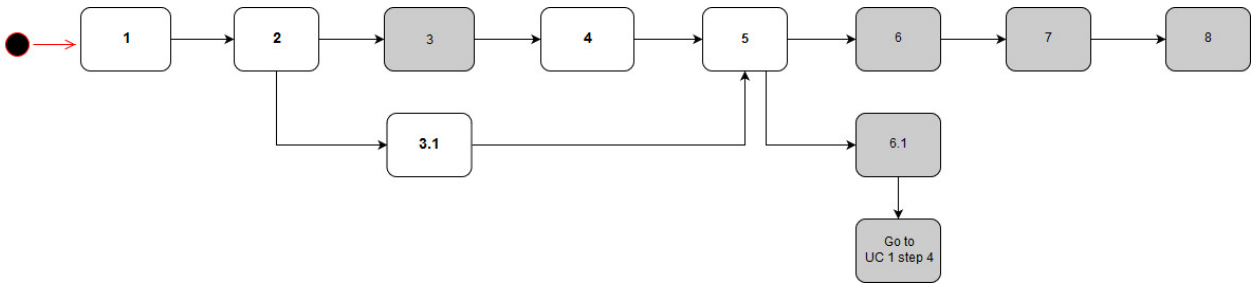
Expected: The system displays texts:
“_____”
“Clue: A programming language”
“_ a _ a”
“Guesses left: 9”
“Guessed letters: a”
“To terminate write .exit and press enter”.

System prompts: “Enter a letter”

TC 3.3 Play Game guess wrong letter

Use Case: UC 3 Play Game

Scenario: An alternative scenario of UC 3 where user guesses a wrong letter.



Precondition: Do all steps in TC 1.1, the system prompts “Enter a letter”.

Test steps:

1. Enter the letter “k” and press enter.

Expected: The system displays text:

“ |
|
|
|
|
|
|_____ “

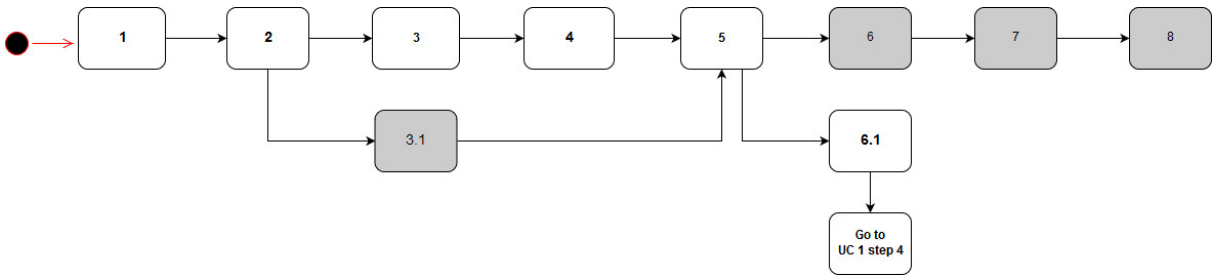
“Clue: A programming language”
“ _ _ _ _ ”
“Guesses left: 8”
“Guessed letters: k”
“To terminate write .exit and press enter”

System prompts: “Enter a letter”

TC 3.4 Play Game guess wrong last letter

Use Case: UC 3 Play Game

Scenario: An alternative scenario of UC 3 where user uses up all the guesses and loses.



Precondition: Do all steps in TC 1.1 and TC 2.3 the system prompts “Enter a letter”

Test steps:

1. Enter the letter “h” and press enter.
2. System prompts “Enter a letter”
3. Enter the letter “i” and press enter.
4. System prompts “Enter a letter”
5. Enter the letter “p” and press enter.
6. System prompts “Enter a letter”
7. Enter the letter “q” and press enter.
8. System prompts “Enter a letter”
9. Enter the letter “w” and press enter.
10. System prompts “Enter a letter”
11. Enter the letter “e” and press enter.
12. System prompts “Enter a letter”
13. Enter the letter “u” and press enter.
14. System prompts “Enter a letter”
15. Enter the letter “o” and press enter.

Se expected on next page.

Expected: The system displays texts:
“ |+---------+
| |
| O
| \|/
| / \
|
|_____”

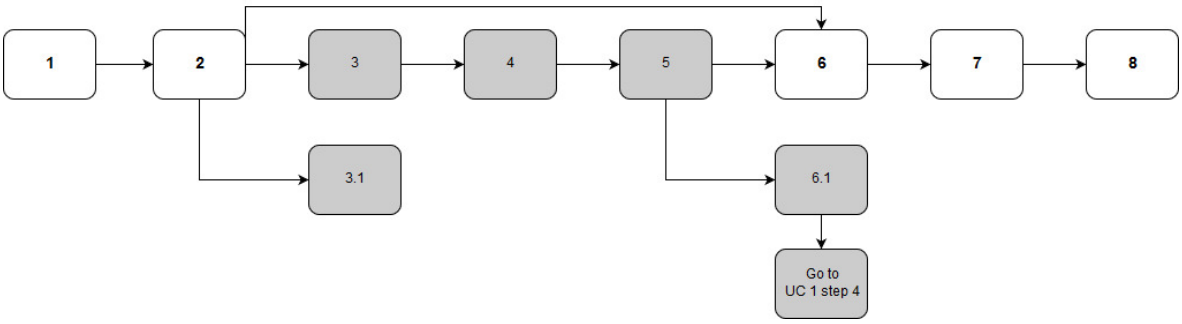
“Clue: A programming language”
“ ”
“Guesses left: 0”
“Guessed letters: k, h, i, p, q, w, e, u, o”
“To terminate write .exit and press enter”
“Game Over”
“**********”

System prompts: “What do you want to do?” with options:
“Start Game”,
“Quit”,
“Add word to the game”

TC 3.5 Play Game guess the word right

Use Case: UC 3 Play Game

Scenario: A piece of the main scenario where the user guesses the right word.



Precondition: Open the folder data then open the folder testFiles and click on the file testHighscore.json. Remove the fifth object “,{“name”: “Lisa”, “points”: (contains a number)}” if it exists and save (ctrl s). Do all steps in TC 1.1, the system prompts “Enter a letter”.

Test steps:

1. Enter “java” and press enter.

Expected: The system displays texts:
“_____”
“Clue: A programming language”
“java”
“Guesses left: 9”
“Guessed Letters: java”

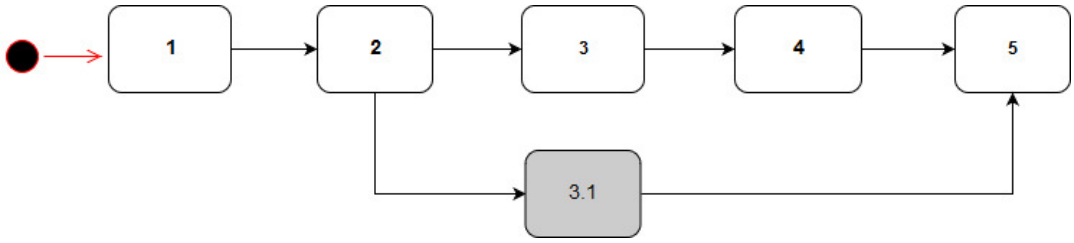
“You guessed the word “java”, congratulations you win!”
“**********”

The system prompts “Do you want to view the high score?” (Y/n).

TC 4.1 Display High Score

Use Case: UC 4 High Score

Scenario: The main scenario of UC 4 where user confirms display high score.



Precondition: Open the folder data then open the folder testFiles and click on the file testHighscore.json. Remove the fifth object “,{“name”: “Lisa”, “points”: (contains a number)}” if it exists and save (ctrl s). Have a timer ready. Do all steps in TC 1.1, when completed step 5 start the timer. Do all steps in TC 3.1, when completed step 5 stop the timer.

Test steps:

1. The system prompts “Do you want to view the high score?” (Y/n).
2. Enter Y

Expected: System prompts: “What do you want to do?” with options:
“Start Game”,
“Quit”,
“Add word to the game”

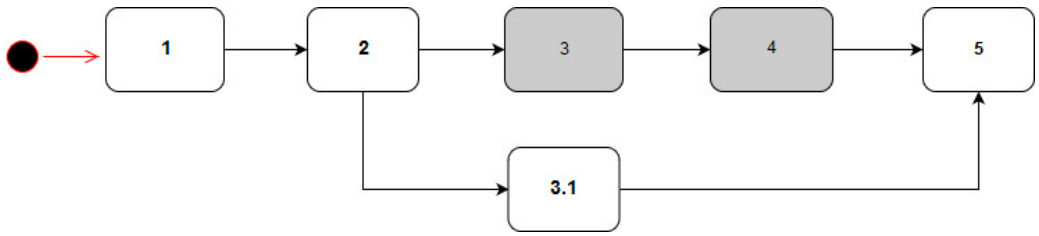
System displays high score in text:
“
1| Lisa with result (should show roughly the time gotten on the timer used in precondition)
2| Niklas with result 30.2
3| Anna with result 35
4| Elin with result 38.2
5| Markus with result 41.2
“

Note that the order of the high scores in the list can change depending on the time gotten in the precondition. The list should be sorted descending.

TC 4.2 Don’t display High Score

Use Case: UC 4 High Score

Scenario: An alternative scenario where the user choses not to display the high score.



Precondition: Open the folder data then open the folder testFiles and click on the file testHighscore.json. Remove the fifth object “,{“name”: “Lisa”, “points”: (contains a number)}” if it exists and save (ctrl s). Do all steps in TC 1.1 and then do all steps in TC 3.1.

Test steps:

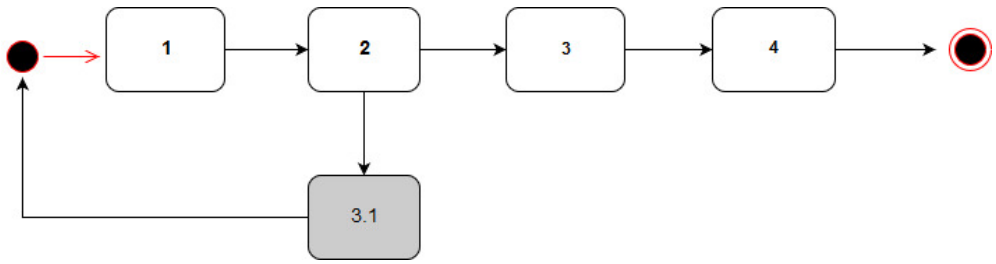
1. The system prompts “Do you want to view the high score?” (Y/n).
2. Enter n

Expected: System prompts: “What do you want to do?” with options:
“Start Game”,
“Quit”,
“Add word to the game”

TC 5.1 Quit game successfully

Use Case: UC 5 Quit Game.

Scenario: The main scenario for UC 5where the user quits the game successfully.



Precondition: Do all steps in TC 1.1, the system prompts “Enter a letter”.

Test steps:

1. Enter “.exit” and press enter
2. System prompts “Are you sure you want to quit?” (y/N)
3. Enter y

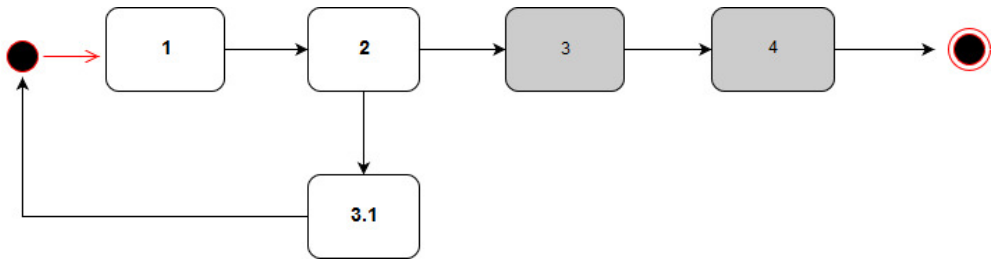
Expected: The system displays text:
“Closing down game… OK”
“**********”

System prompts: “What do you want to do?” with options:
“Start Game”,
“Quit”,
“Add word to the game”

TC 5.2 Do not quit game

Use Case: UC 5 Quit Game.

Scenario: An alternative scenario of UC 5 where the user chooses not to quit the game.



Precondition: : Do all steps in TC 1.1. Do step 1 in TC 5.1.

Test steps:

1. System prompts “Are you sure you want to quite?” (y/N)
2. Enter n

Expected: The system displays texts:
“ _____ ”

“Clue: A programming language”
“ _ _ _ _ ”
“Guesses left: 9”
“Guessed letters: ” and “To terminate write .exit and press enter”

System prompts : “Enter a letter”

Test Report Manual Test Cases

Test	UC 1	UC 2	UC 3	UC 4	UC 5	Comments
TC 1.1	1/OK					
TC 1.2	1/OK					
TC 1.3	1/OK					
TC 2.1	1/OK	1/OK				
TC 3.1			1/OK			
TC 3.2			1/OK			
TC 3.3			1/OK			
TC 3.4			1/OK			
TC 3.5			1/OK			
TC 4.1				1/OK		
TC 4.2				1/OK		
TC 5.1					1/OK	
TC 5.2					1/OK	
Coverage/Success	4/OK	1/OK	5/OK	2/OK	2/OK	

Unit Tests Results

The unit tests are developed in Visual Studio Code using the Jest testing framework. To run the tests write “npm test” in the terminal and press enter. To get coverage run the command “npm test -- --coverage” .

Test result

```
Test Suites: 5 passed, 5 total
Tests:       37 passed, 37 total
Snapshots:   0 total
Time:        2.952s
Ran all test suites.
```

Coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	61.57	45	60	61.51	
AddWords.js	50	60	40	50	... 57,58,59,60,62
Hangman.js	67.01	50	85.71	67.01	... 42,143,145,156
HighScore.js	56.67	27.78	52.94	56.9	... 66,167,168,169
Prompts.js	38.46	100	42.86	38.46	... 15,127,128,129
Timer.js	28.57	0	50	28.57	... 42,45,46,47,50
Word.js	100	100	100	100	
drawHangman.js	100	100	100	100	

Reflection

I have noticed that writing testable code can be quit challenging sometimes. When I’m writing the code for the first time my focus use to be to solve the problem. That can sometimes result in code that is quite hard to test, especially if one isn’t that comfortable with using mocks. I think that the result gives good enough coverage though. The unit tests, tests much of the logic I have written and the manual test cases tests the requirements.

This project has made me interested in testing TTD because I feel that it would be a good way to learn writing testable code. I also have realised that I need to read more about how to write mock functions.