SOFTWARE DEVELOPMENT PROJECT

LNU 1dv600 Programvaruteknik

FINAL HANGMAN PROJECT DOCUMENTATION

NAME: Niall Thurrat

Date Started: 23 Jan 2019

DOCUMENT VERSION: v1.3 (Iteration 4)

Contents

1 PLANNING

1.1 Revision History

Various sections of this document have been changed or removed over time, but are recorded in past versions (i.e. Project Plan v1.0, v1.1 and v1.2) which are stored in the separate iteration folders in the project repository in the documents folder.

Date	Version	Description	Author
7/2/19	1.0	~~Basic structural code of hangman game~~ Initial Project plan fully completed with following exceptions: sections 5.2-5.4 (Iteration 2-4) to be developed in later revisions, sections 7.2-7.3 (Timelogs) to be completed in future revisions	Niall Thurrat
21/2/19	1.1	Project Plan has had following changes: - Section 1: revision history amended - Section 3: System Vision amended - Section 4.1.5 Hard- and Software Requirements has had 2 unnecessary items removed - Section 5.2 Iteration 2 has been developed - Section 7.2 Time log Iteration 2 has been added	Niall Thurrat
19/3/19	1.2	Project Plan has had following revisions: - Section 5.3 Iteration 3 has been developed. This includes moving implementation of additional features to Iteration 4 - Section 5.4 Iteration 4 has been developed. This includes deletion of additional features which will not be included in the project, and addition of additional features from Iteration 3 (namely adding usernames and the high score board) - Section 7.3 Time log Iteration 3 has been added	Niall Thurrat
8/4/19	1.3	Project Plan has had following revisions: - Cover page: title added. - Contents – changed to reflect new document scope, including testing, modelling, game instructions and reflections. Note that this document has now been developed to document the project as a whole. The previous version which was a Project Plan, is now	Niall Thurrat

		contained in Section 1, with all other sections after being added to the doc to complete the project. - Section 1 (Revision History): into paragraph added to explain where past versions of the doc can be found - Section 2 (General Info: Executive Summary): added modelling and testing docs in iteration 4 as deliverables - Section 3.1 (System Vision): game difficulty settings and whole word guessing removed (due to time constraints) - Section 4.1.4 (Resources): Modelling and testing tools added - Section 4.1.5 (System Requirements): removed visual studio code as a specific IDE requirement - Section 4.1.6 (Overall Project Schedule): added testing and modelling docs to iteration 4 as deliverables, and added 2 new deadlines for attempt 2 for iterations 3 and 4. - Section 4.1.7 (Scope, Constraints and Assumptions): the scope of having game difficulty settings as well as random words being obtained from the internet have been removed (due to time constraints), assumptions paragraph updated to include knowledge of using IDE and github - Section 5.4 (iteration 4 Plan): dates/deadlines changed – as I am submitting this project for the Attempt 2 deadline, the iteration deadlines have been changed to reflect this - Section 8 (Reflections iteration 4/whole project): added	

1.2 General Information

Project Summary	
Project Name	Project ID
Hangman	#1
Project Manager	Main Client
Niall Thurrat	LNU teachers
Key Stakeholders	
LNU 1dv600 Software Engineering teachers Project Manager – Niall Thurrat	
Executive Summary	
The game "Hangman" will be developed as a console application using JavaScript and displayed in a text based fashion in a computer console for a single user to play, and released on Github. The project will have four iterations and the application will therefore be released 4 times incrementally. Deliverables will include the four releases of the application, as well as this Project Plan document which will be revised and developed at the end of each iteration, and additional documentation regarding modelling and testing during the second and third iteration respectfully, as well as additional modelling and testing docs during the final iteration which will document the final application as a whole.	

1.3 Vision

1.3.1 System Vision

SUMMARY

Hangman is a word guessing game where the user must guess letters of a predefined word (nouns from the English dictionary will be used) with a view to finally guessing the whole word. With each wrong guess a piece will be added to a picture of a hanging man which will be constructed using common keyboard characters. If all the pieces which complete the hangman are added to the picture, the game is lost. The player will win the game if they complete the word by guessing all the letters before the whole hangman is complete.

FUNCTIONALITY

The game will be developed in a text based fashion and presented on a console terminal for a single user to play. When the user starts the application they will see a main menu which will give them the option to play the game, see the high score board, read the game instructions or quit the application. When the gamer starts a new game, two things will happen before the game begins: 1) the user will be asked to enter a username, and 2) a secret noun will be randomly selected. Once the username is entered, the terminal will present a row of hyphens, each representing a letter of the randomly chosen secret noun, as well as a game menu where the user can either choose to guess a letter, quite the game or quit the application. A help message will always be displayed to provide guidance to the player about, for example, the rules or how many wrong tries they have remaining.

If the player correctly guesses a letter which exists in the word, that letter is revealed (multiple times if appropriate) and the hangman picture remains the same. If a guessed letter is not a part of the secret word, a piece of the hangman will be added and the letter will be added to a list of 'Wrong Letters'. Likewise, a wrongly guessed word will see the hangman develop and will show in a 'Wrong Words' list. The application will make use of data persistence to enable progression through the game and to store the highest game scores. The high score chart which is accessible from the main menu will show the top 5 results.

1.3.2 Reflections on Creating a Vision Document

I've structured my system vision document with a *Summary* section to help stakeholders understand the system at a glance, as well as a *Functionality* section to give further detail which will be used as guidance (by myself, the development team) to create a more detailed breakdown of necessary project tasks to be completed in each application iteration (see chapter 5 regarding project iterations).

I consider vision documents to be an important method to allow stakeholders to understand an application better as well as to help project workers gain a common understanding about what it is they must create. Vision documents are an effective way to outline key functionality which can later be used to identify project tasks, without being as time consuming as a full specification requirements document.

1.4 Project Plan

1.4.1.1 Introduction

The primary objective of this project is to create a software application called “Hangman”, based on the classic game of the same name. Hangman is a word guessing game where the user must guess letters of a predefined secret word with a view to finally guessing the whole word. This Hangman application will be developed using JavaScript and displayed in a text based fashion on a console terminal for a single user to play.

1.4.1.2 Justification

The primary purpose of developing this application is to demonstrate my understanding and ability to utilize and document a software engineering process as a requirement for my Software Engineering course (1DV600) at Linnaeus University. A secondary reason for the creation of the application is to develop my knowledge of the core elements of the course and to further develop my programming skills and experience in relation to that knowledge.

1.4.1.3 Stakeholders

- System end-users: The people who will use the finished system (The Software Engineering teachers at Linnaeus University, Niall Thurrat, the public)
- Project manager: Coordinates who does what and when during the project and makes sure that this plan is followed, that tasks are completed on time, and that risks and delays are managed (Niall Thurrat)
- System architect: Responsible for the structuring and design of the system (Niall Thurrat)
- Client engineer – responsible for implementing the plan and coding the application (Niall Thurrat)
- Tester: responsible for ensuring the system functions as it should without bugs (Niall Thurrat)

1.4.1.4 Resources

People (support and advice)	- Teachers (MyMoodle, slack, tutorials, QA) - Other students (slack)
Info, guidance, project brief and tutorials	Instructions and advice provided on course’s MyMoodle page, internet/google
Literature	Software Engineering (10th Edition) by Ian Sommerville
Version control and project repository	Github
Project task timer	Toggl timer app tool
Development tools and environment (inc. app dependencies)	Node.js, npm, visual studio code
Word Processing	MS word
Modelling (UML tool)	Enterprise Architect
Testing	Mocha and Chai npm packages

1.4.1.5 Hard- and Software Requirements

Hardware	- Computer with internet
Software	- Node.js and npm - IDE

1.4.1.6 Overall Project Schedule

DATE	**DELIVERABLE**
23 January 2019	- Project begins
8 February 2019	- Iteration 1 of Hangman app - Project Plan document
21 February 2019	- Iteration 2 of Hangman app - Project Plan document (iteration 2) - UML modelling documentation
~~8 March 2019~~	~~- Iteration 3 of Hangman app – ATTEMPT 1~~ ~~- Project Plan document (iteration 3)~~ ~~- Test documentation~~
~~22 March 2019~~	~~- Iteration 4 of Hangman app (final project completion) – ATTEMPT 1~~ ~~- Project Plan document (iteration 4 and whole project)~~ ~~- modelling document for whole project~~ ~~- testing document for whole project~~
5 April 2019	- Iteration 3 of Hangman app – ATTEMPT 2 - Project Plan document (iteration 3) - Test documentation
19 April 2019	- Iteration 4 of Hangman app (final project completion) – ATTEMPT 1 - Project Plan document (iteration 4 and whole project) - modelling document for whole project - testing document for whole project

1.4.1.7 Scope, Constraints and Assumptions

The scope of the project is to create an application which can be played in a computer console and will therefore not have a GUI. The files will be sourced in a github repository and available to any member of the public to play on their home computer. The game will be completely generated by common keyboard characters and thus will not contain any actual image files. Furthermore sound will not be used. The game will include a high-score chart and username entry.

A major constraint will be time, given I do not have much more than 20 hours a week to devote to the project and it appears that more than half of this time will be dedicated to reading and tutorials. My lack of knowledge and experience in writing terminal applications is also a constraint.

I assume that end-users will have sufficient knowledge of how to use an IDE, how to clone/download the game from github, how to use a computer console, and how to install node.js and the npm dependencies, as well as start the application.

1.4.2 Reflection on Creating a Project Plan

The Project Plan provides project stakeholders with an essential overview of the purpose of the project, interested parties, resources involved and the most important dates during the duration of the project. Creating the document has helped me give further consideration to the resources and time that is available to me, and has helped me clarify in simple terms the purpose for this project.

1.5 Iteration Plan

1.5.1 Iteration 1 Plan

DATE (start - due date)	TIME ESTIMATE (HRS)	TASK: Sub-Task	RESOURCES REQUIRED
WEEK 4 (21/1 - 27/1) – 10 EFFORT HRS			
23/1	1.75	LEARNING: Lecture - intro	MyMoodle
23/1 – 27/1	8.25 ->	LEARNING: reading literature (ch 2, 3, 22, 23)	book
WEEK 5 (28/1 – 3/2) – 20 EFFORT HRS			
28/1 – 30/1	<- 3.75 (12 hrs total)	LEARNING: reading literature (ch 2, 3, 22, 23)	book
28/1 – 30/1	3.5	LEARNING: pre-recorded lecture vids (x 2)	MyMoodle
30/1	1.75	LEARNING: QA - processes	YouTube, teacher
30/1 ->	7.5 ->	PLANNING: Start Project Plan document	MS Word, book, MyMoodle
31/1 – 1/2	0.5	EXAM: Theme 1 online exam	MyMoodle
2/2 – 3/2	3	LEARNING: Have a think and google about console applications and how mine might function and be structured	Internet, google, past program exercises
WEEK 6 (4/2 – 10/2) – 20 EFFORT HRS			
<- 4/2	<- 5 (12.5 hrs total)	PLANNING: continue and finish 1st draft of Project Plan doc	MS Word, book, MyMoodle
4/2 – 6/2	9	LEARNING: reading literature (ch 4, 5, 20)	book
6/2	1.75	LEARNING: Tutoring – modelling	teachers, MyMoodle
4/2	3	PLANNING/DESIGN: create a flow chart to predict as much basic and additional game functionality as possible from start to finish of application to help me populate this Iteration Plan with project tasks.	MS Word
4/2 – 8/2	1	IMPLEMENTATION: create development environment using IDE and node.js. Create probable modules and structure project directory, export and import as necessary and create basic functions/objects in each which print test messages to console.	vsc, node,
4/2 – 8/2	0.25	IMPLEMENTATION: Do release on Github/MyMoodle	github,MyMoodle

ITERATION 1 DEADLINE: 23:55 8/2

1.5.2 Iteration 2 Plan

DATE (start - due date)	TIME ESTIMATE (HRS)	TASK: Sub-Task	RESOURCES REQUIRED
WEEK 7 (11/2 – 17/2) – 20 EFFORT HRS			
11/2 – 13/2	1	LEARNING: reading literature (ch 7.1)	book
11/2 – 13/2	3	LEARNING: UML practical tutorials including identifying and downloading appropriate software	MyMoodle, UML tools
13/2	1.75	LEARNING: Tutoring session – UML	teachers, MyMoodle, youtube
13/2	0.25	Reading assignment 2 brief	Github wiki
13/2	0.5	PLANNING: develop Iteration 2 tasks and time predictions in Project Plan document	MS Word, MyMoodle
13/2 – 17/2	2	MODELLING: create Use Case diagram of system as described in the project Vision with UML	UML tool, github wiki, book
13/2 – 17/2	2	MODELLING: create fully dressed use case for basic "Play Game"	MS Word, github wiki, book
13/2 – 17/2	2	MODELLING: create State Machine Diagram for basic "Play Game"	UML tool, github wiki, book
13/2 – 17/2	0.5	IMPLEMENTATION: design a basic game banner image and 8 placeholders of basic hangman image (prints to console), return placeholders from imageGenerator.js	vsc, node.js, google
13/2 – 17/2	2	IMPLEMENTATION: create a skeleton menu which is printed to console from Hangman.js (write from scratch or use npm package/find re-useable code)	vsc, node.js, npm, google
13/2 – 17/2	0.25	IMPLEMENTATION: add ‘play new game’ option to start menu and configure so that it uses imageGenerator.js to produce banner and hangman	vsc, node.js
13/2 – 17/2	0.5	IMPLEMENTATION: develop wordGenerator.js module which returns a randomly selected noun from a hardcoded list of 10 words. Develop wordBoard.js module to call wordGenerator.js module for a new word and return	vsc, node.js

13/2 – 17/2	0.5	IMPLEMENTATION: develop messageGenerator.js module to return a message from a list of predefined game messages (welcome, unlucky, good guess, game over, etc)	vsc, node.js
13/2 – 17/2	0.25	IMPLEMENTATION: develop Hangman.js to log new game screen to console (inc. banner, hangman base pic, wordBoard, Message, Menu)	vsc, node.js
13/2 – 17/2	3.5 ->	IMPLEMENTATION: develop Hangman.js/wordBoard.js logic to deal with guessed letters and reprint new game state to console	vsc, node.js
WEEK 8 (18/2 – 24/2) – 30 EFFORT HRS			
18/2-20/2	10	LEARNING: reading literature (ch 6, 7, 15)	book
18/2-20/2	5	LEARNING: pre-recorded lecture vids (x 3)	MyMoodle
20/2	1.75	LEARNING: Tutoring – Design	teachers, MyMoodle
21-22/2	1.5	EXAM: Theme 2 online exam including topic revision	MyMoodle
20/2 – 21/2	<- 2 (5.5 hrs total)	IMPLEMENTATION: finish developing game logic to deal with guessed letters	vsc, node.js
20/2 – 21/2	2	IMPLEMENTATION: use local storage to save game progress	vsc, local storage
20/2 – 21/2	0.5	IMPLEMENTATION: create 'game over' and 'you win!' images (placeholders) and develop Hangman/wordBoard to print to console when game lost or won	vsc, node.js
20/2 – 21/2	0.75	IMPLEMENTATION: add 'quit game' item to menu and develop logic to reset game ('game over' or 'game quit') in Hangman.js	vsc, node.js
20/2 – 21/2	1	IMPLEMENTATION: add 'quit application' to start menu and develop logic to exit (terminate) app in Hangman.js	vsc, node.js
20/2 – 21/2	2	MODELLING: Create a class diagram from implementation	UML tool, github wiki, book
21/2	0.5	RELEASE: Do release on Github/MyMoodle	github, MyMoodle
ITERATION 2 DEADLINE: 12:00 21/2			

1.5.3 Iteration 3 Plan

DATE (start - due date)	TIME ESTIMATE (HRS)	TASK: Sub-Task	RESOURCES REQUIRED
WEEK 9 (25/2 – 3/3) – 20 EFFORT HRS			

25/2	2	PLANNING: revise Project Plan document, add further detail to Iteration 3 Plan and do time estimates	MS word
25/2 – 27/2	3	LEARNING: reading literature (ch 8)	book
25/2 – 27/2	6	LEARNING: pre-recorded lecture vids (x 3 + greeter)	MyMoodle
27/2	2	LEARNING: Tutoring – Test plan (online lecture)	teachers, MyMoodle
27/2 – 3/3	1	TESTING: test planning – reading the brief for assignment 3	MyMoodle, MS Word
27/2 – 3/3	3	TESTING: test planning – complete test plan in Test.pdf	MyMoodle, MS Word
27/2 – 3/3	1	TESTING: complete plan documentation for 1st Manual Test Case	MyMoodle, MS Word
27/2 – 3/3	2	TESTING: write 1st Manual Test Case	MyMoodle, MS Word
WEEK 10 (4/3 - 10/3) – 20 EFFORT HRS			
4/3 – 6/3	1	TESTING: complete plan documentation for 2nd Manual Test Case	MyMoodle, MS Word
4/3 – 6/3	1.5	TESTING: write 2nd Manual Test Case	MyMoodle, MS Word
4/3 – 6/3	5	TESTING: research which JavaScript testing framework to use and learn how to use it	MyMoodle, Google
6/3	2	LEARNING: Tutoring - Test (online lecture)	teachers, MyMoodle
6/3 – 8/3	2	TESTING: write and run automated unit test 1 for Method 1, document results	testing framework, VSC, MS Word
6/3 – 8/3	2	TESTING: write and run automated unit test 2 for Method 1, document results	testing framework, VSC, MS Word
6/3 – 8/3	2	TESTING: write and run automated unit test 1 for Method 2, document results	testing framework, VSC, MS Word
6/3 – 8/3	2	TESTING: write and run automated unit test 2 for Method 2, document results	testing framework, VSC, MS Word
6/3 – 8/3	1	TESTING: write, run and report a failed test case	testing framework, VSC, MS Word

6/3 – 8/3	1	TESTING: write reflections and finalise Test.pdf	MS Word
8/3	0.5	IMPLEMENTATION: Do release on Github/MyMoodle, check all documentation in order	github, MyMoodle
ITERATION 3 DEADLINE: 23:55 8/3 **ITERATION 3 DEADLINE: 23:55 5/4 – ATTEMPT 2**			

1.5.4 Iteration 4 Plan

DATE (start - due date)	TIME ESTIMATE (HRS)	TASK: Sub-Task	RESOURCES REQUIRED
ITERATION 4 - WEEK 1 (8/4 – 14/4) – 22.75 EFFORT HRS			
8/4 – 9/4	1.5	LEARNING/PLANNING: Read assignment brief and read notes on slack	MyMoodle, slack
8/4 – 9/4	1.5	LEARNING/PLANNING: watch final recorded online lecture/QA session (with Tobias Andersson Gidlund recorded 13th March)	teachers, youtube
8/4 – 9/4	3	PLANNING: complete Iteration 4 in *Final Hangman Project Documentation* re final additional features and project completion as a whole, including time estimations and new deadlines for handing in assignment in Take 2	MS word
8/4 – 9/4	0.25	PLANNING: add Iteration 4 Time Log to project plan	MS Word
8/4 – 9/4	1	PLANNING: revise *Iteration 4 Project Plan-Final Project Doc* (e.g. project vision), note all changes in Revision History	MS word
8/4 – 9/4	0.75	PLANNING: Add *Iteration 4/Whole Project Reflections* section (inc. Planning, Modelling, Implementation and Testing sub-sections) and complete *Planning* reflections	Ms word
8/4 – 9/4	0.5	PLANNING: Change structure of document – will be Final Hangman Project Documentation with sections Planning, Modelling, Testing, Game Instructions, Reflections	MS Word
8/4 – 9/4	0.25	DOCUMENTATION: Create file structure in Documentation folder in repository to clearly separate all docs for each iteration	vsc, node.js and npm, github
10/4 – 11/4	2	MODELLING: Find and learn to use a new UML modelling tool (my last was a free month trial)	google
10/4 – 11/4	0.5	MODELLING: create Use Case diagram of system as described in the project Vision	UML tool, book, mymoodle

10/4 – 11/4	0.75	MODELLING: create use case for 'Start Game' (include Game Instruction, and High Score Board).	MS Word, book, mymoodle
10/4 – 11/4	0.25	MODELLING: create use case for 'Quit Game'	MS Word, book, mymoodle
10/4 – 11/4	0.25	MODELLING: create use case for 'Quit Application'	MS Word, book, mymoodle
10/4 – 11/4	1	MODELLING: create use case for 'Play Game' (include user name and high score board)	MS Word, book, mymoodle
10/4 – 11/4	2	MODELLING: create State Machine Diagram for 'Play Game' (include user name and high score board)	UML tool, book
10/4 – 11/4	0.5	DOCUMENTATION: add all models to *Final Hangman Project Documentation*	github, MS Word
10/4 – 11/4	0.75	DOCUMENTATION: add *Modelling* reflections to *Final Hangman Project Documentation*	Ms word
12/4 - 15/4	3	IMPLEMENTATION: improve game appearance including banner image, hangman images and games messages	vsc, node.js and npm
12/4 - 15/4	2	IMPLEMENTATION: Divide functions in Hangman class into more testable functions	vsc, node.js and npm
12/4 - 15/4	0.25	IMPLEMENTATION: Develop/increase secret word list in wordGenerator	vsc, node.js and npm
12/4 - 15/4	0.75	IMPLEMENTATION: Add 'wrong letter list' to game display and sessionObject so player knows previous bad guesses	vsc, node.js and npm
ITERATION 4 - WEEK 2 (15/4 – 24/3) – 23.5 EFFORT HRS			
12/4 - 15/4	2	IMPLEMENTATION: add 'enter username' logic to ask gamer for name when they start game, develop logic to store username in sessionObject	vsc, node.js and npm
12/4 - 15/4	5	IMPLEMENTATION: add high score board module and functionality – should be option in main menu to view current high scores and scores must store in local-storage	vsc, node.js and npm
12/4 - 15/4	2	IMPLEMENTATION: add messages which update after each user guess	vsc, node.js and npm
12/4 - 15/4	1.5	IMPLEMENTATION: add *Game Instructions* to main menu, write code for user to access this and write the instructions	vsc, node.js and npm
10/4 – 11/4	0.5	MODELLING: create class diagram for game classes/modules	UML tool, book, mymoodle

16/4 – 17/4	0.5	TESTING: include relevant previous testing sections into *Final Hangman Project Documentation*	MS Word
16/4 – 17/4	1	TESTING: test planning – update test plan in *Final Hangman Project Documentation*	MS Word
16/4 – 17/4	0.25	TESTING: remove failed test case from application and *Final Hangman Project Documentation*	VSC, MS Word
16/4 – 17/4	0.5	PLANNING: update Iteration 4 Plan in *Final Hangman Project Documentation: Planning* regarding unit tests (these have not been decided before implementation)	MS word
16/4 – 17/4	0.5	TESTING: document and conduct Start Game Successful Manual Test Case	VSC, MS Word
16/4 – 17/4	1	TESTING: write and conduct Play Winning Game with High Score - Manual Test Case	VSC, MS Word
16/4 – 17/4	0.5	TESTING: write and conduct Play Winning Game with No High Score - Manual Test Case	VSC, MS Word
16/4 – 17/4	0.25	TESTING: imageGeneratorTest – test 1. Update and run AUT 1.1 (automated unit test for Method 1: Test 1)	Mocha, VSC, MS Word
16/4 – 17/4	0.25	TESTING: imageGeneratorTest – test 2. Update and run AUT 1.2 (automated unit test for Method 1: Test 2)	Mocha, VSC, MS Word
16/4 – 17/4	0.25	TESTING: wordUpdaterTest test 1. Update and run AUT 2.1 (automated unit test for Method 2: Test 1)	Mocha, VSC, MS Word
16/4 – 17/4	0.25	TESTING: wordUpdaterTest – test 2. Update and run AUT 2.2 (automated unit test for Method 2: Test 2)	Mocha, VSC, MS Word
16/4 – 17/4	1	TESTING: highScoreBoardTest.displayBoard – test 1. Write and run AUT 3.1 (automated unit test for Method 3: Test 1)	Mocha, VSC, MS Word
16/4 – 17/4	1	TESTING: highScoreBoardTest.displayBoard – test 2. Write and run AUT 3.2 (automated unit test for Method 3: Test 2)	Mocha, VSC, MS Word
16/4 – 17/4	1	TESTING: highScoreBoardTest.updateHighScores – test 1. Update and run AUT 4.1 (automated unit test for Method 4: Test 1)	Mocha, VSC, MS Word
16/4 – 17/4	1	TESTING: highScoreBoardTest.updateHighScores – test 2. Update and run AUT 4.2 (automated unit test for Method 4: Test 1)	Mocha, VSC, MS Word
18/4	1	DOCUMENTATION: Enter testing reflections into *Iteration 4 Project Plan-Final Project Doc*	MS Word
18/4	1	DOCUMENTATION: include complete instructions on how the game is started, played and tested in *Final Hangman Project Documentation*	MS Word

18/4	0.75	DOCUMENTATION: Finalise *Final Hangman Project Documentation* and add to *Documentation: Iteration-4* folder in repository	MS Word, github, vsc
18/4	0.5	DEPLOYMENT: do release on Github/MyMoodle, check all documentation in order	github, MyMoodle
ITERATION 4 DEADLINE (ATTEMPT 2): 19/4 – 00:00			

1.6 Risk Analysis

1.6.1 List of Risks

RISK (AFFECTS): DESCRIPTION	PROBABILITY	EFFECTS
Loss of project personnel – permanent (project): I leave the project before it finishes	Low	Catastrophic
Loss of project personnel – temporary (project): I am ill at critical times in the project, or cannot commit as much time due to unforeseen circumstances	Moderate	Serious
Programming skills inadequate (project and product): Project developer does not have necessary skills required to develop planned system	High	Serious
Size underestimate (project and product): The size of the application is underestimated	Moderate	Serious
Time underestimate (project and product): The time required to develop the software is underestimated	High	Serious
Requirements changes (project): A larger number of changes to the system requirements than anticipated	Low	Tolerable
Software tool underperformance (product): Software tools that support the project do not perform as anticipated	Low	Tolerable

1.6.2 Strategies

RISK	STRATEGY (type)
Loss of project personnel – permanent	In the event that I cannot complete the project (and am still alive) I will contact the course responsible and program admin re decisions taken, save any project documentation for possible future use, and inquire about

	possible opportunities to begin the project again in a future course. (contingency plan)
Loss of project personnel – temporary	- Avoid contracting illness during project duration (e.g. eating healthily, taking supplements to ensure adequate vitamin and nutrient intake, using alcohol gel hand sanitizer before eating and constantly washing hands, avoiding unnecessarily leaving apartment). (avoidance strategy) - Plan to have essential project requirements and important extra features completed by the third iteration and use the fourth iteration to add desirable (non-essential) extra features. This way tasks which cannot be completed during iteration 1-3 can be moved to the final iteration at the expense of unnecessary feature completion if time is lost due to unforeseen circumstances. (minimization strategy) - Ask my employer for time off to devote additional hours to working on the project and thus increase the effort (person-months) initially assigned to the project. (minimization strategy)
Programming skills inadequate	Allow extra time for learning when estimating project tasks in the Iteration Plan, especially for aspects of coding which I am unsure about (minimization strategy)
Size underestimate	- Plan to have essential project requirements and important extra features completed by the third iteration and use the fourth iteration to add desirable (non-essential) extra features. This way tasks which cannot be completed during iteration 1-3 can be moved to the final iteration at the expense of unnecessary feature completion if time is lost due to the application being larger than anticipated. (minimization strategy) - Ask my employer for time off to devote additional hours to working on the project and thus increase the effort (person-months) initially assigned to the project. (minimization strategy)
Time underestimate	- Plan to have essential project requirements and important extra features completed by the third iteration and use the fourth iteration to add desirable (non-essential) extra features. This way tasks which cannot be completed during iteration 1-3 can be moved to the final iteration at the expense of unnecessary feature completion if time is lost due to the application taking longer to develop than anticipated. (minimization strategy) - Ask my employer for time off to devote additional hours to working on the project and thus increase the effort (person-months) initially assigned to the project. (minimization strategy)
Requirements changes	- This is a low probability as the essential application requirements are known from the start of the project and will not change. It is possible that desirable additional features may change the requirements, or that I have misunderstood the project requirements in the brief to a certain degree. Any effect should be minimized by dropping non-essential features from the final iteration to accommodate for extra time and resources necessary

	to cope with changes. I may ask my employer for time off also, to devote more hours to the project (minimization strategy) - Project instructions should be triple-checked to ensure understanding of system requirements before designing and implementing system. (avoidance strategy)
Software tool underperformance	- Try to use software which I am familiar with (when possible) to complete planned tasks in the project. (avoidance strategy) - Use slack and other communication channels to discuss software tool underperformance and seek suggestions for alternatives from other student and teachers (minimization)

1.6.3 Reflections on Risk Analysis

As this project is quite small with an eight week duration (approx. one person-month of effort) and one project team-member, there is not a large list of moderate-high risks to consider and, as a consequence, all but one risk have consequences for the project. The risks highlight that due to having a one-man team, the effects of a number risks which might delay the project are categorised as serious.

As a minimizing strategy to address the effects of four risks in the list, I have decided to plan to have all essential tasks regarding system requirements (as well as important non-essential features) complete by the third iteration, allowing desirable extra features to be implemented on the final iteration. This allows flexibility to move essential tasks to the final iteration, if for some reason they cannot be completed on schedule. Writing this document also made me realise that a viable risk minimization strategy is to increase the project effort (person-months) to ensure timely completion of the project (i.e. ask for time off work).

1.7 Time Log

1.7.1 Iteration 1 Time Log

DATE (start - due date)	TIME ESTIMATE (HRS)	TASK / Sub-Task	TIME TAKEN (HRS)
WEEK 4 (21/1 - 27/1) – 10 EFFORT HRS			
23/1	1.75	LEARNING: Lecture - intro	as estimated
23/1 – 27/1	8.25 ->	LEARNING: reading literature (ch 2, 3, 22, 23)	total below
WEEK 5 (28/1 – 3/2) – 20 EFFORT HRS			
28/1 – 30/1	<- 3.75 (12 hrs total)	LEARNING: reading literature (ch 2, 3, 22, 23)	13
28/1 – 30/1	3.5	LEARNING: pre-recorded lecture vids (x 2)	as estimated
30/1	1.75	LEARNING: QA - processes	1.5
30/1 ->	7.5 ->	PLANNING: Start Project Plan document	total below
31/1 – 1/2	0.5	EXAM: Theme 1 online exam	as estimated
2/2 – 3/2	3	LEARNING: Have a think and google about console applications and how mine might function and be structured	2
WEEK 6 (4/2 – 10/2) – 20 EFFORT HRS			
<- 4/2	<- 5 (12.5 hrs total)	PLANNING: continue and finish 1st draft of Project Plan doc	14.75
4/2 – 6/2	9	LEARNING: reading literature (ch 4, 5, 20)	10
6/2	1.75	LEARNING: Tutoring – modelling	as estimated
4/2	3	PLANNING/DESIGN: create a flow chart to predict as much basic and additional game functionality as possible from start to finish of application to help me populate this Iteration Plan with project tasks.	as estimated
4/2 – 8/2	1	IMPLEMENTATION: create development environment using IDE and node.js. Create probable modules and structure project directory, export and import as necessary and create basic functions/objects in each which print test messages to console.	1.5
4/2 – 8/2	0.25	IMPLEMENTATION: Do release on Github/MyMoodle	as estimated
ITERATION 1 DEADLINE: 23:55 8/2			

1.7.2 Iteration 2 Time Log

DATE (start - due date)	TIME ESTIMATE (HRS)	TASK / Sub-Task	TIME TAKEN (HRS)
WEEK 7 (11/2 – 17/2) – 20 EFFORT HRS			
11/2 – 13/2	1	LEARNING: reading literature (ch 7.1)	1.25
11/2 – 13/2	3	LEARNING: UML practical tutorials including identifying and downloading appropriate software	2.5
13/2	1.75	LEARNING: Tutoring session – UML	2.25
13/2	0.25	Reading assignment 2 brief	0.75
13/2	0.5	PLANNING: develop Iteration 2 tasks and time predictions in Project Plan document	0.75
13/2 – 17/2	2	MODELLING: create Use Case diagram of system as described in the project Vision with UML	3.25*
13/2 – 17/2	2	MODELLING: create fully dressed use case for basic "Play Game"	4.5*
13/2 – 17/2	2	MODELLING: create State Machine Diagram for basic "Play Game"	2.5
13/2 – 17/2	0.5	IMPLEMENTATION: design a basic game banner image and 8 placeholders of basic hangman image (prints to console), return placeholders from imageGenerator.js	0.25
13/2 – 17/2	2	IMPLEMENTATION: create a skeleton menu which is printed to console from Hangman.js (write from scratch or use npm package/find re-useable code)	8**
13/2 – 17/2	0.25	IMPLEMENTATION: add 'play new game' option to start menu and configure so that it uses imageGenerator.js to produce banner and hangman	as estimated
13/2 – 17/2	0.5	IMPLEMENTATION: develop wordGenerator.js module which returns a randomly selected noun from a hardcoded list of 10 words. Develop wordBoard.js module to call wordGenerator.js module for a new word and return	0.25
13/2 – 17/2	0.5	IMPLEMENTATION: develop messageGenerator.js module to return a message from a list of predefined game messages (welcome, unlucky, good guess, game over, etc)	0.5
13/2 – 17/2	0.25	IMPLEMENTATION: develop Hangman.js to log new game screen to console (inc. banner, hangman base pic, wordBoard, Message, Menu)	as estimated
13/2 – 17/2	3.5 ->	IMPLEMENTATION: develop Hangman.js/wordBoard.js logic to deal with guessed letters and reprint new game state to console	total below

WEEK 8 (18/2 – 24/2) – 30 EFFORT HRS			
18/2-20/2	10	LEARNING: reading literature (ch 6, 7, 15)	7
18/2-20/2	5	LEARNING: pre-recorded lecture vids (x 3)	as estimated
20/2	1.75	LEARNING: Tutoring – Design	2
21-22/2	1.5	EXAM: Theme 2 online exam including topic revision	1
20/2 – 21/2	<- 2 (5.5 hrs total)	IMPLEMENTATION: finish developing game logic to deal with guessed letters	7.25
20/2 – 21/2	2	IMPLEMENTATION: use local storage to save game progress	3
20/2 – 21/2	0.5	IMPLEMENTATION: create 'game over' and 'you win!' images (placeholders) and develop Hangman/wordBoard to print to console when game lost or won	1
20/2 – 21/2	0.75	IMPLEMENTATION: add 'quit game' item to menu and develop logic to reset game ('game over' or 'game quit') in Hangman.js	0.25
20/2 – 21/2	1	IMPLEMENTATION: add 'quit application' to start menu and develop logic to exit (terminate) app in Hangman.js	0.25
20/2 – 21/2	2	MODELLING: Create a class diagram from "Play Game" use case after implementation	1.5
21/2	0.5	RELEASE: Do release on Github/MyMoodle	as estimated
ITERATION 2 DEADLINE: 12:00 21/2			

Notes on big differences between estimate and actual times for iteration 2:

* the modelling of the use case diagram and the fully dressed use case took much longer than expected as a significant amount of time was spent learning how to do these, as well as having to redraw them both due to not reading the instructions properly

** finding the right npm package to use and learning how to implement it took 4 times longer than anticipated. This was always a risk due to my lack of experience and knowledge with node.js, npm packages and coding command line interfaces.

1.7.3 Iteration 3 Time Log

DATE (start - due date)	TIME ESTIMATE (HRS)	TASK / Sub-Task	TIME TAKEN (HRS)
WEEK 9 (25/2 – 3/3) – 20 EFFORT HRS			

25/2	2	PLANNING: revise Project Plan document, add further detail to Iteration 3 Plan and do time estimates	2.5
25/2 – 27/2	3	LEARNING: reading literature (ch 8)	2.75
25/2 – 27/2	6	LEARNING: pre-recorded lecture vids (x 3 + greeter)	7
27/2	2	LEARNING: Tutoring – Test plan (online lecture)	3*
27/2 – 3/3	1	TESTING: test planning – reading the brief for assignment 3	as estimated
27/2 – 3/3	3	TESTING: test planning – complete test plan in Test.pdf	3.5
27/2 – 3/3	1	TESTING: complete plan documentation for 1st Manual Test Case	1.5
27/2 – 3/3	2	TESTING: write 1st Manual Test Case	1
WEEK 10 (4/3 - 10/3) – 20 EFFORT HRS			
4/3 – 6/3	1	TESTING: complete plan documentation for 2nd Manual Test Case	0.5
4/3 – 6/3	1.5	TESTING: write 2nd Manual Test Case	0.45
4/3 – 6/3	5	TESTING: research which JavaScript testing framework to use and learn how to use it	5.45
6/3	2	LEARNING: Tutoring – Test (online lecture)	3*
6/3 – 8/3	2	TESTING: write and run automated unit test 1 for Method 1, document results	1
6/3 – 8/3	2	TESTING: write and run automated unit test 2 for Method 1, document results	0.5
6/3 – 8/3	2	TESTING: write and run automated unit test 1 for Method 2, document results	as estimated
6/3 – 8/3	2	TESTING: write and run automated unit test 2 for Method 2, document results	0.25
6/3 – 8/3	1	TESTING: write, run and report a failed test case	0.5
6/3 – 8/3	1	TESTING: write reflections and finalise Test.pdf	as estimated
8/3	0.5	IMPLEMENTATION: Do release on Github/MyMoodle, check all documentation in order	as estimated
ITERATION 3 DEADLINE: 23:55 8/3			

Notes on big differences between estimate and actual times for iteration 3:

* took me a little longer to watch online tutorials as they were done in Swedish – good practice for me but I do need to do a lot of rewind and replay!

1.7.4 Iteration 4 Time Log

DATE (start - due date)	TIME ESTIMATE (HRS)	TASK: Sub-Task	RESOURCES REQUIORED	TIME TAKEN (HRS)
ITERATION 4 - WEEK 1 (8/4 – 14/4) – 22.75 EFFORT HRS				
8/4 – 9/4	1.5	LEARNING/PLANNING: Read assignment brief and read notes on slack	MyMoodle, slack	2
8/4 – 9/4	1.5	LEARNING/PLANNING: watch final recorded online lecture/QA session (with Tobias Andersson Gidlund recorded 13th March)	teachers, youtube	1
8/4 – 9/4	3	PLANNING: complete Iteration 4 in *Final Hangman Project Documentation* re final additional features and project completion as a whole, including time estimations and new deadlines for handing in assignment in Take 2	MS word	3.5
8/4 – 9/4	0.25	PLANNING: add Iteration 4 Time Log to project plan	MS Word	as estimated
8/4 – 9/4	1	PLANNING: revise *Iteration 4 Project Plan-Final Project Doc* (e.g. project vision), note all changes in Revision History	MS word	0.75
8/4 – 9/4	0.75	PLANNING: Add *Iteration 4/Whole Project Reflections* section (inc. Planning, Modelling, Implementation and Testing sub-sections) and complete *Planning* reflections	Ms word	0.75
8/4 – 9/4	0.5	PLANNING: Change structure of document – will be Final Hangman Project Documentation with sections Planning, Modelling, Testing, Game Instructions, Reflections	MS Word	1
8/4 – 9/4	0.25	DOCUMENTATION: Create file structure in Documentation folder in repository to clearly separate all docs for each iteration	vsc, node.js and npm, github	as estimated
10/4 – 11/4	2	MODELLING: Find and learn to use a new UML modelling tool (my last was a free month trial)	google	0.75*

10/4 – 11/4	0.5	MODELLING: create Use Case diagram of system as described in the project Vision	UML tool, book, mymoodle	as estimated
10/4 – 11/4	0.75	MODELLING: create use case for 'Start Game' (include Game Instruction, and High Score Board).	MS Word, book, mymoodle	0.5
10/4 – 11/4	0.25	MODELLING: create use case for 'Quit Game'	MS Word, book, mymoodle	as estimated
10/4 – 11/4	0.25	MODELLING: create use case for 'Quit Application'	MS Word, book, mymoodle	as estimated
10/4 – 11/4	1	MODELLING: create use case for 'Play Game' (include user name and high score board)	MS Word, book, mymoodle	0.75
10/4 – 11/4	2	MODELLING: create State Machine Diagram for 'Play Game' (include user name and high score board)	UML tool, book	1.5
10/4 – 11/4	0.5	DOCUMENTATION: add all models to *Final Hangman Project Documentation*	github, MS Word	as estimated
10/4 – 11/4	0.75	DOCUMENTATION: add *Modelling* reflections to *Final Hangman Project Documentation*	Ms word	1.25
12/4 - 15/4	3	IMPLEMENTATION: improve game appearance including banner image, hangman images and games messages	vsc, node.js and npm	1**
12/4 - 15/4	2	IMPLEMENTATION: Divide functions in Hangman class into more testable functions	vsc, node.js and npm	3.25***
12/4 - 15/4	0.25	IMPLEMENTATION: Develop/increase secret word list in wordGenerator	vsc, node.js and npm	as estimated
12/4 - 15/4	0.75	IMPLEMENTATION: Add 'wrong letter list' to game display and sessionObject so player knows previous bad guesses	vsc, node.js and npm	0.25
ITERATION 4 - WEEK 2 (15/4 – 24/3) – 23.5 EFFORT HRS				
12/4 - 15/4	2	IMPLEMENTATION: add 'enter username' logic to ask gamer for name when they start game, develop logic to store username in sessionObject	vsc, node.js and npm	1.25
12/4 - 15/4	5	IMPLEMENTATION: add high score board module and functionality – should be option in main menu to view current high	vsc, node.js and npm	9.75****

		scores and scores must store in local-storage		
12/4 - 15/4	2	IMPLEMENTATION: add messages which update after each user guess	vsc, node.js and npm	0.5*****
12/4 - 15/4	1.5	IMPLEMENTATION: add *Game Instructions* to main menu, write code for user to access this and write the instructions	vsc, node.js and npm	1
10/4 – 11/4	0.5	MODELLING: create class diagram for game classes/modules	UML tool, book, mymoodle	as estimated
16/4 – 17/4	0.5	TESTING: include relevant previous testing sections into *Final Hangman Project Documentation*	MS Word	0.75
16/4 – 17/4	1	TESTING: test planning – update test plan in *Final Hangman Project Documentation*	MS Word	1.5
16/4 – 17/4	0.25	TESTING: remove failed test case from application and *Final Hangman Project Documentation*	VSC, MS Word	as estimated
16/4 – 17/4	0.5	PLANNING: update Iteration 4 Plan in *Final Hangman Project Documentation: Planning* regarding unit tests (these have not been decided before implementation)	MS word	0.75
TEST PLAN: TIME PLAN (corresponds to section 3.1.4)				
16/4 – 17/4	0.5	TESTING: document and conduct Start Game Successful Manual Test Case	VSC, MS Word	0.25
16/4 – 17/4	1	TESTING: write and conduct Play Winning Game with High Score - Manual Test Case	VSC, MS Word	0.75
16/4 – 17/4	0.5	TESTING: write and conduct Play Winning Game with No High Score - Manual Test Case	VSC, MS Word	as estimated
16/4 – 17/4	0.25	TESTING: imageGeneratorTest – test 1. Update and run AUT 1.1 (automated unit test for Method 1: Test 1)	Mocha, VSC, MS Word	0 (not required)
16/4 – 17/4	0.25	TESTING: imageGeneratorTest – test 2. Update and run AUT 1.2 (automated unit test for Method 1: Test 2)	Mocha, VSC, MS Word	as estimated
16/4 – 17/4	0.25	TESTING: wordUpdaterTest test 1. Update and run AUT 2.1 (automated unit test for Method 2: Test 1)	Mocha, VSC, MS Word	as estimated

16/4 – 17/4	0.25	TESTING: wordUpdaterTest – test 2. Update and run AUT 2.2 (automated unit test for Method 2: Test 2)	Mocha, VSC, MS Word	as estimated
16/4 – 17/4	1	TESTING: highScoreBoardTest.displayBoard – test 1. Write and run AUT 3.1 (automated unit test for Method 3: Test 1)	Mocha, VSC, MS Word	as estimated
16/4 – 17/4	1	TESTING: highScoreBoardTest.displayBoard – test 2. Write and run AUT 3.2 (automated unit test for Method 3: Test 2)	Mocha, VSC, MS Word	as estimated
16/4 – 17/4	1	TESTING: highScoreBoardTest.updateHighScores – test 1. Update and run AUT 4.1 (automated unit test for Method 4: Test 1)	Mocha, VSC, MS Word	0.75
16/4 – 17/4	1	TESTING: highScoreBoardTest.updateHighScores – test 2. Update and run AUT 4.2 (automated unit test for Method 4: Test 1)	Mocha, VSC, MS Word	0.5
18/4	1	DOCUMENTATION: Enter testing reflections into *Iteration 4 Project Plan-Final Project Doc*	MS Word	1.25
18/4	1	DOCUMENTATION: include complete instructions on how the game is started, played and tested in *Final Hangman Project Documentation*	MS Word	0.5
18/4	0.75	DOCUMENTATION: Finalise *Final Hangman Project Documentation* and add to *Documentation: Iteration-4* folder in repository	MS Word, github, vsc	0.5
18/4	0.5	DEPLOYMENT: do release on Github/MyMoodle, check all documentation in order	github, MyMoodle	
ITERATION 4 DEADLINE (ATTEMPT 2): 19/4 – 00:00				

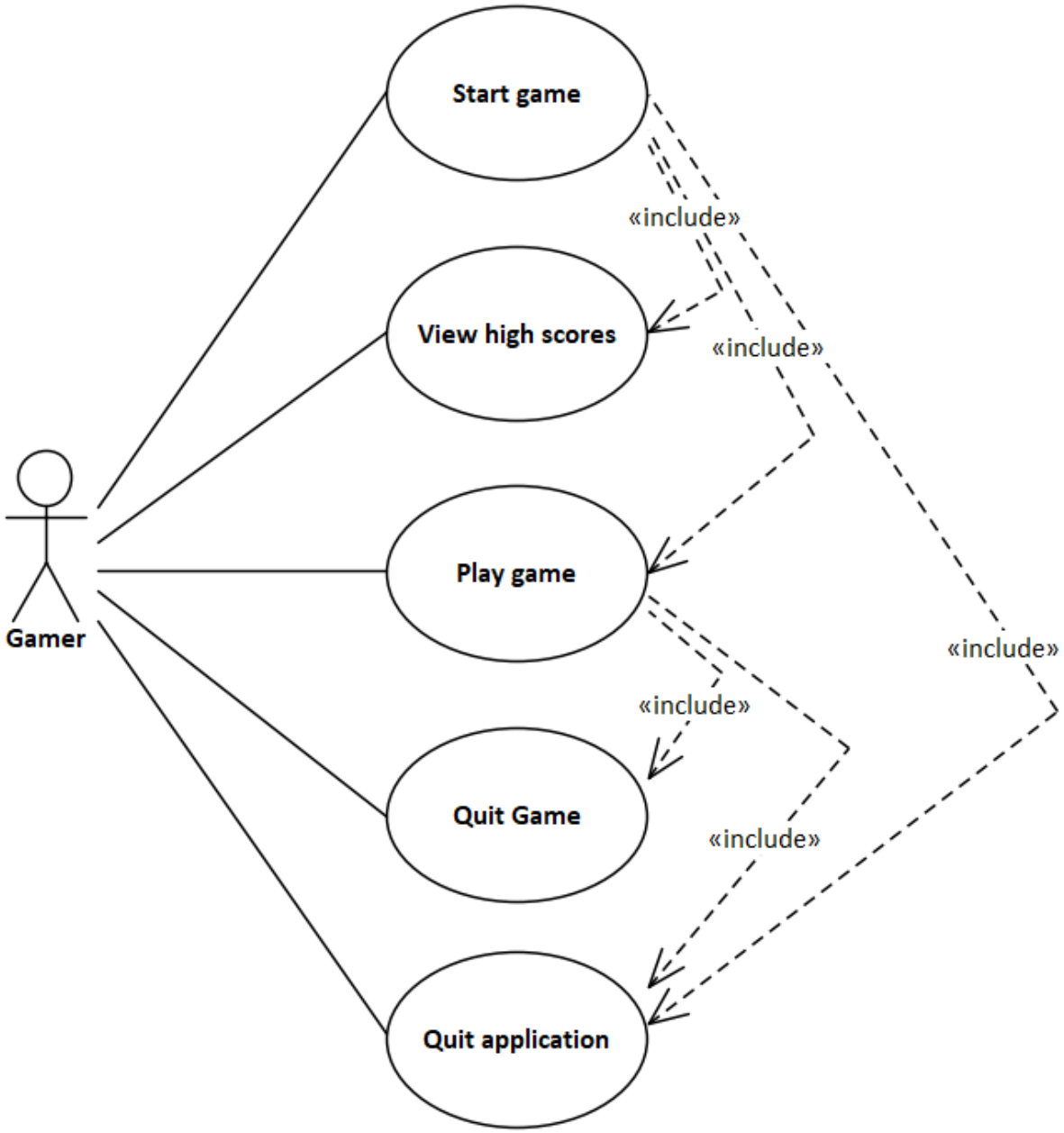
EXPLANATIONS ON LARGE GAPS BETWEEN EXPECTED AND ACTUAL TIME TAKEN:

* I expected it to take longer to identify a new UML tool and learn to use it but am still able to use my trial version of Enterprise Architect with acceptable restrictions

** It took me longer than expected to make my app a little nicer and add a banner and the hangman images as I was able to use CFonts npm package for the banner and the hangman frame was much easier to construct than I'd anticipated.
*** Took a lot longer than expected to divide code into more readable and testable functions. Still not happy but time constraints mean I must move on. Difficulties with having dependencies control menu items and most functions result in console logging.
**** The new feature took a lot longer than anticipated. I stumbled on some really odd behaviour where I couldn't compare 2 arrays. Very frustrating. Had to rewrite highScoreBoard module and Hangman.gameEnds method. Also took longer than expected to store, sort and compare arrays of high score results.
*****I forgot that this functionality was partially developed already

2 MODELLING

2.1 Hangman Use Case Diagram



2.2 Use Case 1 – Start Game

Precondition: none.

Post-condition: the base hangman image, a secret word (represented by hyphens) and the game menu are shown.

Main scenario

1. Starts when the gamer wants to begin a session of the hangman game.
2. The system presents the game title and main menu, with the options to play game, view high score board, view game instructions and quit application.
3. The Gamer makes the choice to start the game.
4. The system starts the game (see Use Case 2).

Alternative scenarios

3.1 The Gamer makes the choice to quit the application (see Use Case 4)

3.2 The Gamer makes the choice to view the high score board.

1. The system presents the high score board.

3.3 The Gamer makes the choice to view the game instructions.

1. The system presents the game instructions.

2.3 Use Case 2 – Play Game

Precondition: A random secret word has been selected. The high-score board is empty.

Post-condition: The gamer has been notified if they have won or lost the game. The high score board is updated if the players score is good enough.

Main scenario

1. Starts when the gamer wants to play a game of hangman.
2. System asks the gamer to enter a username.
3. Gamer enters an appropriate username.
4. System presents a base hangman image, secret word (represented by hyphens) and game menu.
5. Gamer guesses a letter.
6. System updates the secret word where letter is found.

Gamer repeats steps 5-6 until word is complete.

7. System updates the high score board, generates a winner message with high score notification, then returns to main menu.

Alternative scenarios

3.1 Gamer enters a username which is not appropriate.

1. System displays error message and prompts gamer for new username
2. Go to 3.

5.1 Gamer makes the choice to quit the current game (see Use Case 3).

5.2 Gamer makes the choice to terminate the application (see Use Case 4).

5.3 Gamer guesses no letter or more than 1 letter.

1. System generates an error message.
2. Go to 3.

6.1 System updates the hangman, the ‘Wrong Letter List’ and the number of remaining guesses.

1. Go to 3.

6.2 System completes the hangman, generates a loser message and then returns to main menu.

2.4 Use Case 3 – Quit Game

Precondition: The gamer has selected to quit the game in the game menu.

Post-condition: The main menu is displayed.

Main scenario

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

Alternative scenarios

3.1. The user does not confirm

1. The system returns to its previous state

2.5 Use Case 4 – Quit Application

Precondition: The app is running.

Post-condition: The app is terminated.

Main scenario

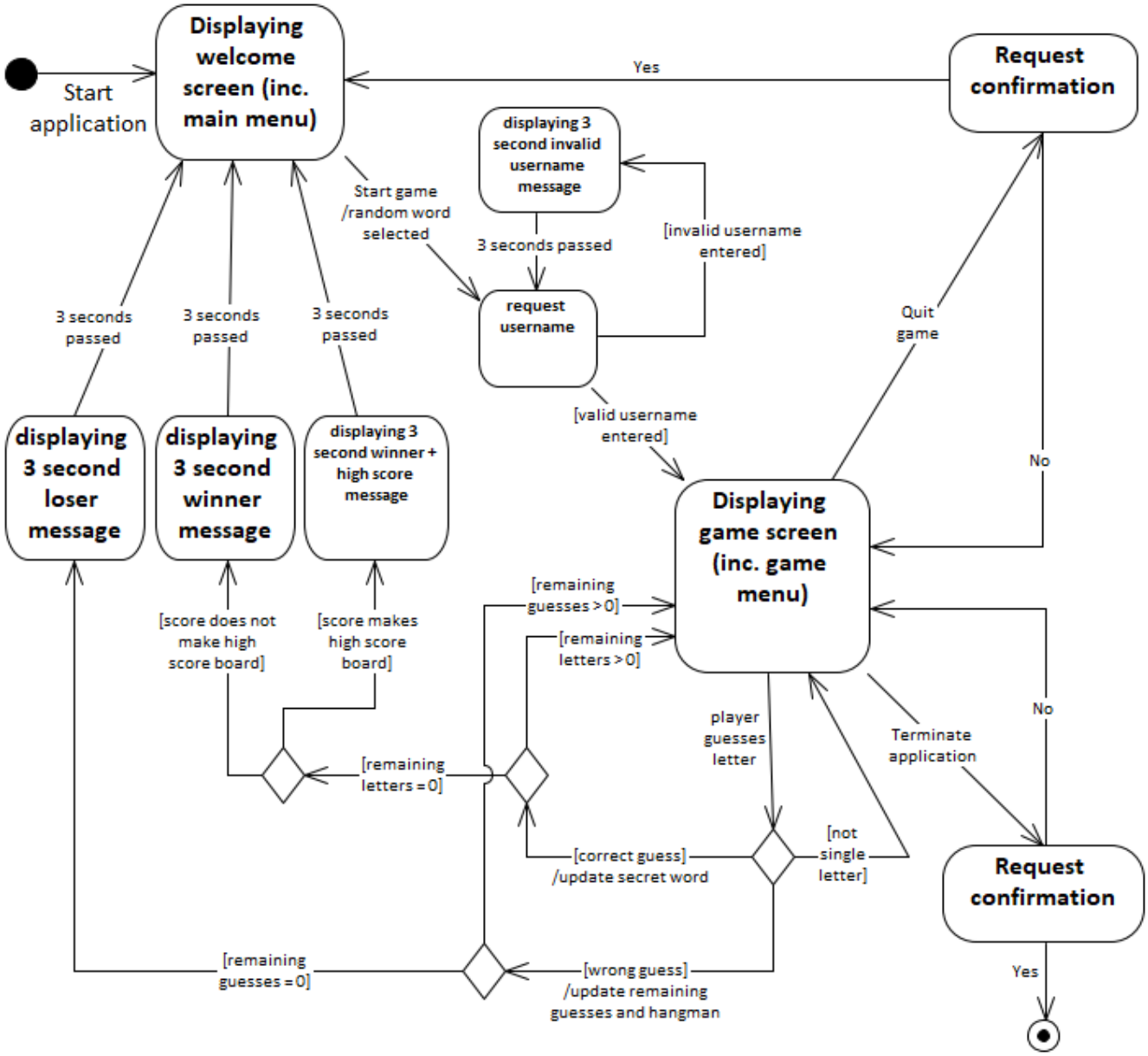
1. Starts when the user wants to quit the app.
2. The system prompts for confirmation.
3. The user confirms.
4. The system terminates.

Alternative scenarios

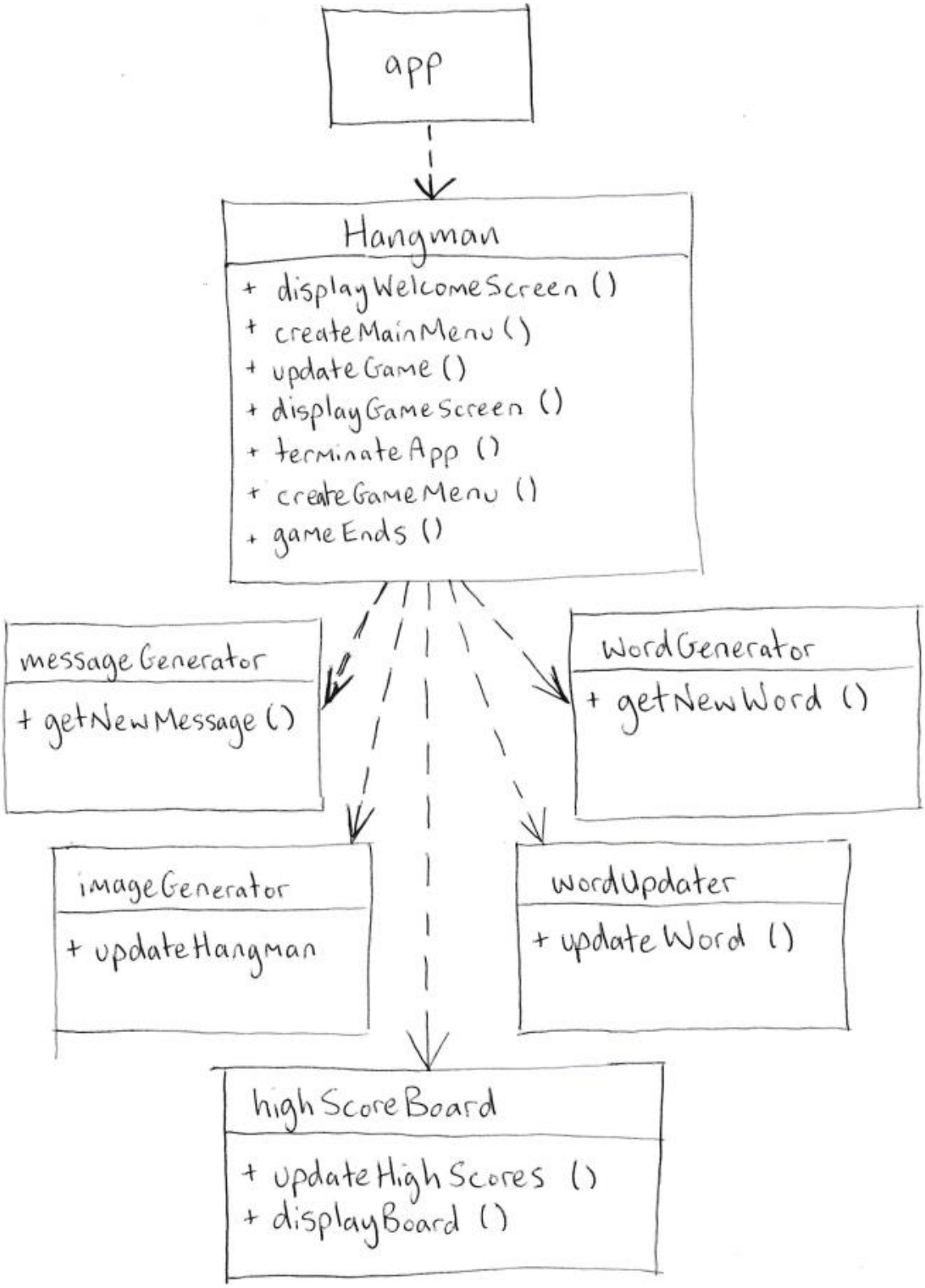
3.1. The user does not confirm

2. The system returns to its previous state

2.6 Play Game State Machine



2.7 Hangman Class Diagram



3 TESTING

3.1 Test Plan

3.1.1 What are the objectives of the testing in this iteration?

The objectives of the testing during this iteration are to update the tests from iteration 3 to function for the refactored code in iteration 4, as well as test the newly added high-score board feature, to ensure that 1) the program functions as its expected to do (in terms of its functional requirements), and 2) to identify defects which may be present in the code.

3.1.2 What to test? Include a short rationale

I intend to update the test case for UC 1 (Start Game) and will create two separate test cases from the UC 2 (Play Game) which test winning the game with a score with gets onto the high-score board and winning the game with a score which does not. I have chosen UC 1 as it is the first use case and faults in this part of the program should be identified first, as they may affect the functionality of further use cases. I have chosen to test UC 2 (Play Game) as it is the primary function of the application and the main reason the user will be using the app. I have chosen to test both making the high-score board and not to ensure the new feature functions properly.

I intend to update the 2 automated unit tests from iteration 3 on the imageGenerator module. These test are important because they will give me confidence that the hangman images will be returned properly when called in the updateGame function. I will update the 2 automated unit tests from iteration 3 on the wordUpdater function also, and chose to test this because it has a slightly more complicated input parameter object and the tests will be useful in future iterations to ensure that problems created by code changes will be identified. I intend to write 2 automated tests for each function in the new high-score board feature to ensure that they return the correct values and will alert in future if the code is changed and results in the function failing.

3.1.3 How this testing is going to be done?

The tests will be dynamic, i.e. the application will be executed during testing. I shall use manual testing and automated unit tests.

3.1.4 Time plan

See Iteration 4 Plan above (section 1.5.4).

3.2 Manual Test Cases

3.2.1 TC 1.1 - Start Game Successful

3.2.1.1 Name and id of the test-case

Name: Start Game Successful
ID: TC1.1

3.2.1.2 Reference to what requirement (use-case) that is tested

Use Case 1 (see section 2.2 Use Case 1 – Start Game)

3.2.1.3 Short description of what is tested and why

The main scenario from the Start Game use case will be tested here to ensure that the game can be started as it should from the main menu. This is the starting point of the application and will therefore be the starting point of my testing as it is important to ensure this functions as it should before moving on to testing other scenarios or other uses cases which rely on this part of the application to function properly.

3.2.1.4 Preconditions that need to be fulfilled before this test is can be started

All application files (including dependencies from npm) must be downloaded/installed to prepare the application environment (in an IDE) and the app must be started using the command 'npm start' in the terminal before the test can be completed. No other test cases need to be ran before this one.

3.2.1.5 Test-steps including test-data.

- Press '1' to select Play Game in main menu

3.2.1.6 A description of the expected result.

The system shows the base hangman image, a secret word (represented by hyphens) and the game menu which has 3 options: Guess a letter, Quit game and Quit application.

3.2.1.7 Checkboxes if the test did succeed or fail.

☑ The system shows the base hangman image, a secret word (represented by hyphens) and the game menu which has 3 options: Guess a letter, Quit Game and Quit Application.

3.2.1.8 A space for comments by the tester.

Nothing noteworthy. Test passed as expected.

3.2.2 TC 2.1.1 - Play Winning Game and Achieve High-score

3.2.2.1 Name and id of the test-case.

Name: Play Winning Game and Achieve High-score
ID: TC 2.1.1

3.2.2.2 Reference to what requirement (use-case) that is tested.

Use Case 2 – Play Game (main scenario) is tested (see section 2.3)

3.2.2.3 Short description of what is tested and why.

The main scenario from the Play Game use case will be tested here to ensure that when the correct letters are entered the game will be won as expected and final score will be sufficient to be automatically entered onto the high-score board. This will be tested as it is important to ensure that the game can be won by the player as expected, as well as to test that the high-score board is populated with game scores which are good enough.

3.2.2.4 Preconditions that need to be fulfilled before this test is can be started.

TC1.1 should be passed before conducting this test case. UC1 (main scenario) must be completed to start the game and there should be a comment at the top of the game which reveals the 6 letter secret word for the purposes of this test, e.g. 'SECRET WORD REVEALED FOR DEV TESTING: retain'. The storedHighScores file in the localstorage folder should contain the following string:

[{"username":"neo","highScore":7},{"username":"jim","highScore":6},{"username":"dave","highScore":5},{"username":"bob","highScore":4},{"username":"filip","highScore":3}]

3.2.2.5 Test-steps including test-data.

1. The system will ask you to enter your username
2. Type 'tester' and then the *Enter* key.
3. The game screen will open and shows the game menu which will give you 3 options: [1] Guess a letter, [2] Quit game and [3] Quit application. Type 1 on your keyboard to select the first option.
4. The question 'What letter do you want to guess?' will appear. For test purposes you will see the secret word answer revealed at the top of the game screen. On your keyboard, type the first letter which is currently represented by a hyphen in the six letter secret word, then press *Enter*.
5. The system should replace the hyphen(s) in the 'SECRET WORD' which represent the letter you correctly entered, e.g. if you enter the letter 'r' and only the first letter of the secret word (which is initially represented as '- - - - - -') is an 'r', then you should see 'SECRET WORD: r - - - - -' on the screen. If there are other instances of your entered letter in the revealed word, these should also be replaced in the secret word (check this against the revealed word).
6. *Repeat step 3-5 until you have entered all letters to complete the secret word.*

7. The system should display a 3 second message notifying you that you have won and that your score is good enough to be entered on the high-score board.
8. The system will display the main menu screen where you will have 4 options [1] Play game, [2] View high-score board [3] Read game instructions, and [4] Quit application.
9. Type 2 on your keyboard to ensure you can see 'tester' in first position with a score of 8 in the high-score board.

3.2.2.6 A description of the expected result.

- The system will show the message 'YOU WIN!!! CONGRATULATIONS YOU'VE MADE THE HIGH-SCORE BOARD!!!' for 3 seconds on the screen (after step 6 is complete)
- The tester is returned to the main menu after winning the game
- The high-score board will show 'tester' in first position with a score of 8.

3.2.2.7 Checkboxes if the test did succeed or fail.

☑ The system will show the message 'YOU WIN!!! CONGRATULATIONS YOU'VE MADE THE HIGH-SCORE BOARD!!!' for 3 seconds on the screen (after step 6 is complete)

☑ The tester is returned to the main menu after winning the game

☑ The high-score board will show 'tester' in first position with a score of 8.

3.2.2.8 A space for comments by the tester.

Nothing noteworthy. Test passed as expected.

3.2.3 TC 2.1.2 - Play Winning Game and don't Achieve High-score

3.2.2.1 Name and id of the test-case.

Name: Play Winning Game and don't Achieve High-score
ID: TC 2.1.2

3.2.2.2 Reference to what requirement (use-case) that is tested.

Use Case 2 – Play Game (main scenario) is tested (see section 2.3)

3.2.2.3 Short description of what is tested and why.

The main scenario from the Play Game use case will be tested here to ensure that when the correct letters are entered the game will be won as expected and that a final score that is not sufficient to be entered onto the high-score board is not. This will be tested as it is important to ensure that the game can be won by the player as expected but does not lead to the high-score board being populated with the game score.

This test case also tests that incorrect letters (which do not result in the game being lost) are handled appropriately.

3.2.2.4 Preconditions that need to be fulfilled before this test is can be started.

TC1.1 should be passed before conducting this test case. UC1 (main scenario) must be completed to start the game and there should be a comment at the top of the game which reveals the 6 letter secret word for the purposes of this test, e.g. 'SECRET WORD REVEALED FOR DEV TESTING: retain'. The storedHighScores file in the localstorage folder should contain the following string:

[{"username":"neo","highScore":7},{"username":"jim","highScore":6},{"username":"dave","highScore":5},{"username":"bob","highScore":4},{"username":"filip","highScore":3}]

3.2.2.5 Test-steps including test-data.

1. The system will ask you to enter your username
2. Type 'tester' and then the *Enter* key.
3. The game screen will open and shows the game menu which will give you 3 options: [1] Guess a letter, [2] Quit game and [3] Quit application. Type 1 on your keyboard to select the first option.
4. The question 'What letter do you want to guess?' will appear. For test purposes you will see the secret word answer revealed at the top of the game screen. On your keyboard, type a letter which does not appear in the secret word, then press *Enter*.
5. The system should not replace any hyphen(s) in the 'SECRET WORD', the 'REMAINING TRIES' should be reduced by 1, the 'WRONG LETTER LIST' should be updated with the letter you entered, and the hangman image should have had a piece of the hanging man added.
6. *Repeat step 3-5 6 times so that there are 2 'REMAINING TRIES' left.*
7. Type 1 on your keyboard to select option [1] in the game menu (Guess a letter).
8. The question 'What letter do you want to guess?' will appear. Type the first letter which is currently represented by a hyphen in the six letter secret word, then press *Enter*.
9. The system should replace the hyphen(s) in the 'SECRET WORD' which represent the letter you correctly entered, e.g. if you enter the letter 'r' and only the first letter of the secret word (which is initially represented as '- - - - - -') is an 'r', then you should see 'SECRET WORD: r - - - - -' on the screen. If there are other instances of your entered letter in the revealed word, these should also be replaced in the secret word (check this against the revealed word).
10. *Repeat step 7-9 until you have entered all letters to complete the secret word.*
11. The system should display a 3 second message notifying you that you have won.
12. The system will display the main menu screen where you will have 4 options [1] Play game, [2] View high-score board [3] Read game instructions, and [4] Quit application.
13. Type 2 on your keyboard to ensure you cannot see 'tester' in the high-score board.

3.2.2.6 A description of the expected result.

- The system will show the message 'YOU WIN!!!' for 3 seconds on the screen (after step 10 is complete)
- The tester is returned to the main menu after winning the game.

- The high-score board does not show the player ‘tester’ with a score of 2.

3.2.2.7 Checkboxes if the test did succeed or fail.

☑ The system will show the message ‘YOU WIN!!!’ for 3 seconds on the screen (after step 10 is complete)

☑ The tester is returned to the main menu after winning the game

☑ The high-score board does not show the player ‘tester’ with a score of 2.

3.2.2.8 A space for comments by the tester.

Nothing noteworthy. Test passed as expected.

3.3 Automated Unit Tests

Below I only list the automated unit tests in the Hangman project – see Testing Reflections (section 5.3) for further information on reasoning for each tests inclusion and reflections on implementing the tests.

3.3.1 AUT 1.1

imageGeneratorTest – should throw exception with wrong parameter
Tests the updateHangman function in the imageGenerator module.

3.3.2 AUT 1.2

imageGeneratorTest – should not throw exception with correct parameter
Tests the updateHangman function in the imageGenerator module.

3.3.3 AUT 2.1

wordUpdaterTest – should update progressWord property with correct guessedLetter
Tests the updateWord function in the wordUpdater module

3.3.4 AUT 2.2

wordUpdaterTest – should update remainingTries and wrongLetterList with wrong guessedLetter
Tests the updateWord function in the wordUpdater module

3.3.5 AUT 3.1

highScoreBoardTest - should insert scoresForBoard into board object
Tests the displayBoard function in the highScoreBoard module

3.3.6 AUT 3.2

highScoreBoardTest - should return board object if scoresForBoard empty
Tests the displayBoard function in the highScoreBoard module

3.3.7 AUT 4.1

highScoreBoardTest - should insert gameScore into high-score array
Tests the updateHighScores function in the highScoreBoard module

3.3.8 AUT 4.2

highScoreBoardTest - should not insert gameScore into high-score array
Tests the updateHighScores function in the highScoreBoard module

4 GAME INSTRUCTIONS

4.1 How to start the application

To play this game you must:

- Have installed:
    node.js
    npm
    IDE

- go to https://github.com/niall-thurrat/nt222fc_1dv600 and use the 'Clone or download' button to either get the URL to clone the repo or download the application files to your computer
- open the application folder in your favourite IDE (I recommend visual studio code!)
- open a terminal
- run 'npm install' to get the app dependencies
- make sure your terminal screen is opened up pretty big to see the whole app and run 'npm start' to open the main menu screen
- enjoy!

4.2 How to play the game

HOW TO PLAY HANGMAN

Hangman is a word guessing game where you must guess letters of a predefined word (nouns from the English dictionary) with a view to finally guessing all the letters in the word. With each correct guess your letter will be added into the 'SECRET WORD' which starts of as a bunch of dashes. With each wrong guess a piece will be added to the hanging man which starts off as a simple base, and your 'REMAINING TRIES' will be reduced. If all the pieces which complete the hangman are added to the picture, you lose the game. You win the game if you complete the word by guessing all the letters. Simple!

REMEMBER

- You can only add one letter at a time. If you add a number you'll lose a life for sheer stupidity. If you add 0 or more than 1 letter you'll not lose a life, though you probably should. If you add the same letter wrongly again, you'll lose a life again.

- The ‘WRONG LETTER LIST’ reminds you which letters you’ve guessed

THE HIGH-SCORE BOARD

Your final game score is equivalent to the amount of ‘REMAINING TRIES’ you have left when you complete the word. The winning message when you compete the word will let you know when you’ve got onto the score board. You can always check the high score board from the main menu to see if you’ve made it.

GOOD LUCK!!

4.3 How to test the game

When you have the application open in your IDE, simply run ‘npm test’ in your terminal to run the automated tests.

5 REFLECTIONS (iteration 4/whole project)

5.1 Planning reflections

Having nearly completed this Final Hangman Project Documentation, I can now say that I can really appreciate its value. For such a small application the documentation has seemed at times to be more planning than necessary, but I understand that its main purpose is to help me learn and develop my ability to plan and structure a software project. In doing so, my main reflections are:

- Having a System Vision really helped me focus on what I wanted to achieve at an early stage
- In this final iteration I've benefited from being able to read over past versions of the Project Plan to understand my original vision. Looking at the progression through iterations demonstrates how my thinking about the project has changed as I've had to adapt due to limited resources.
- I have a new appreciation for how planning documentation can help to gauge how much can be fit into a development iteration and have developed strategies for adapting the documentation and the plan when things don't go according to plan (e.g. when I planned too many features and had to prioritise tasks and scale the application down to meet deadlines)
- Completing the iteration plans (section 5) became much easier in the end and really helped me develop my ability to divide an iteration into tasks and to gauge how much time they would take
- It is not a good idea to have a one man development team as losing that person due to, for example, personal reasons means the whole project grinds to a halt and deadlines will be missed

5.2 Modelling reflections

Below I give my reasons for including each model in this Hangman project (found in section 2, Modelling). Due to the small size of the application, I feel it has been sufficient to only use 1 diagram from each different perspective. However I have chosen to create diagrams of the system from 3 different perspectives to allow future developers on the project to gain a rich understanding of it, including: an external/contextual perspective (Use Case Diagram), a behavioural perspective (State Diagram of Play Game Use Case) and a structural perspective (Class Diagram). Furthermore I have developed 3 Use Cases which have acted as specification requirements and a basis for me to implement the application code.

Note that the inclusion of the extra feature 'High Score Board' has not resulted in extra modelling as, although it enhances the gaming experience, it does not add enough complexity to the application to merit a separate model. Instead, it and other extra functionality such as entering a username, game messages and the 'Wrong Letter List', have all been captured by the amended 'Use Case 2 – Play Game' and 'Play Game State Machine' models.

5.1 Hangman Use Case Diagram

I have decided to use a Use Case Diagram for my project as it helps to give an overview of the Use Cases and Actors that will be involved in the system. This was important as it helped me visualise the different ways the system will be used and by whom. I found it difficult to choose which items to include in this diagram, as I tried to only include uses which are things the gamer wants to do. For this reason I have not included 'View Game Instructions' in the diagram. I have however included both 'Quit Game' and 'Quit Application' as I felt it was necessary to make a distinction between these two use cases in this overview diagram, even if they are not necessarily something the user 'wants to do'.

5.2 Use Case 1 – Start Game

I felt it was important to include this use as a fully dressed use case in the project because it is the starting point of the application and is built upon by the main Play Game use case. This is also used to aid manual testing. This use case has functioned as a system requirements doc and been used by the development team to implement the system.

5.3 Use Case 2 – Play Game

I have included this use case as I consider it the main use case in the project because it details the most significant interaction that will be made between the Actor and System during normal use of the application. It focuses on what I consider to be the gamer's main goal when he/she uses the application, i.e. what the gamer really wants to do when they start the application. This use case has functioned as a system requirements doc and been used by the development team to implement the system.

5.4 Use Case 3 – Quit Game

I have included this use case as it is fundamental to the operation of the game and builds on Use Case 2. This use case has functioned as a system requirements doc and been used to implement the system.

5.5 Use Case 4 – Quit Application

I have included this use case as it is fundamental to the operation of the game and builds on Use Case 1 and 2, therefore applying the DRY rule and removing a bit of text from both of these use cases. I also think it is useful to make a further distinction between quit game and quit application use cases.

5.6 Play Game State Machine

This behavioural model is an important diagram which shows how the system reacts to all user input when the game is being played. I've found it really useful to help me think of the system as a set of states. I consider this abstract to be the most important diagram perspective of the system for future developers on this project to understand how the system functions.

5.7 Hangman Class Diagram

This structural model has been included to give future developers on the project an overview of the structure of the different classes/modules in the application code. Creating this diagram helped me to clarify logical grouping of code in my application.

5.3 Testing reflections

5.3.1 Iteration 3 testing reflections

I had some difficulty learning how to use Mocha and Chai to test my application. Chai became necessary for me to test exceptions being throw in relation to what parameters where used in the test, and I was really confused for a while when using assert.equals to compare 2 input and output objects which failed, even though I was sure they were the same (apparently javascript doesn't consider objects equal due to where they're stored). That aside, I've found mocha and chai to be really useful and full of a wide range of fairly intuitive tests. I now realise that the code in my main playGame function is difficult to test because it is very large and also works with a lot of console.logs instead of returned values, and should therefore be broken up into more testable and understandable functions/modules.

5.3.1 Iteration 4 testing reflections

For the final iteration I decided to use my Play Game use case for 2 different manual test cases to test 2 different scenarios where the high-score board is and is not updated after the game has been won. I feel a losing game test case would be beneficial but have not had time to write and conduct this dues to time constraints. I have chosen to use these manual tests as I believe they test the primary functionality of the game and also because I have not been able to write automatic unit tests for a lot of the functions in the Hangman class which simply return side effects (log to the console) and do not have simple input and output.

I have also decided to update the four previous automated unit tests, as well as write 4 four more which are dedicated to the new high-score board feature. I believe the automated tests cover the most essential functionality in the code which is easier/possible to test (without further refactoring) but I feel that more tests would be useful and did not write them due to time constraints. For example the wordGenerator function in particular should have automated tests written. Another reason that I have found functions difficult to test is that I have used a lot of third party npm packages to build the app, for example readlineSync which outputs the menu items.

Bearing in mind that I realised during iteration 3 that my code was difficult to test I made sure that the new high-score board feature is more testable by ensuring that the code returns the scores to a function in the Hangman class so that score-board items can be tested. I think that modelling and implementing code with testing in mind has definitely encouraged me to write cleaner code with smaller, simpler functions which have a clearer purpose.