SOFTWARE DEVELOPMENT PROJECT – Hangman game

José Duarte de Sousa Barroca
Student e-mail address: jd222qf@student.lnu.se
GitHub 1DV600 repository: https://github.com/JBarroca/jd222qf_1dv600

2019-03-22

TABLE OF CONTENTS

1 – Revision history

Date	Version	Description	Author
2019-02-08	0.1	Initial version (iteration 1)	José Barroca
2019-03-19	0.2	Adding deliverables from iterations 2 and 3 before starting to compose final project documentation for iteration 4	José Barroca
2019-03-22	1	Finalised version after completing iteration 4	José Barroca

2 – General Information

Project Summary	
Project Name	Project ID
Hangman	JBarrocaHangman2019
Project Manager	Main Client
José de Sousa Barroca	Linnéuniversitetet
Key Stakeholders	
Project manager – José Barroca Project planning – José Barroca Project development, validation, evolution – José Barroca Client – Teachers and teaching assistants of the course 1DV600 – Software technology of Linnaeus University, Sweden	
Executive Summary	
This project consists of implementing a game of Hangman in the console using the programming language Java. In this single-player game, the player will be able to compete in single games or 5-round games and will be challenged by a score system that rewards guessing harder words in fewer tries and will be motivated by persistent high score tables.	

3 – Vision

This project's goal is to create a console-based Hangman game, written in the programming language Java. When entering the application, the user can select (using the keyboard, as for all further interactions) either to start a game, view the high scores table or quit the application. If the player decides to start a game, he/she may choose to play a single word or a 5-word game. When a new game starts, a random word is chosen from a text file containing words in the English language and is presented to the player with each of its letters replaced by hyphen characters. At the beginning of each game, the hangman drawing will be empty, and the player will have a total of 10 chances to guess the letters included in the word. With each correct guess, the guessed letter is revealed in its position in the word, whilst the others remain replaced by hyphen characters. With each incorrect guess, a new section of the hangman drawing is revealed, and the number of remaining guesses is decremented. There are 10 different sections of the drawing (corresponding to the 10 tries): ground, vertical pole, horizontal pole, rope, head, body, right arm, left arm, right leg, left leg. At any time during the game, the player may guess the entire word by writing it in the console. If the word is correctly guessed, the game is won. After 10 wrong guesses, the drawing is complete and the player loses, revealing the word, the score and the option to either quit the application or start a new game. The score is calculated based on the player's guesses (right guesses increase score, wrong guesses decrease it), an extra bonus for guessing the correct word before finding all the letters and on a word-specific difficulty index based on the relative frequency of each of the word's letters in the English language. The score of a 5-word game will consist of the sum of the scores for each of the 5 words. When the player obtains a high-score (defined as one of the 5 highest scores in the record), the player is requested a username, which will be saved together with the high score. A different high scores table will be kept for single games and 5-word games.

Reflection on writing a vision statement: Writing a vision statement should encompass the goal of the project and summarize what everyone involved in the project should know and aim for. Since I have never had any contact with project management/planning and vision statements before, I felt the need to search for examples of vision statements. I noticed the content, style and idea behind vision statements in different contexts can vary quite a bit. For this project, we were supposed to write a description of the whole system, but in many other examples vision statements are short, captivating messages which strive to engage possible clients/users/investors in the product, and not so much descriptions of the product itself. If the goal of the vision statement is to establish a common ground between every team member, then I agree that a more objective and descriptive text is more useful. But if the goal is to captivate and capture outside interest, then I understand the need for more concise, appealing and innovative messages.

4 – Project Plan

4.1 Introduction

This application is an implementation of the “hangman” game in the console, written using the Java programming language. In this game, the player has a certain number of attempts to guess a random word by guessing its letters, one at a time. For each failed guess, a portion of a drawing of a hanging man is produced and, if the drawing is completed after a certain number of attempts without the word being guessed, the player loses the game.

4.2 Justification

This application is a part of a project in the course 1DV600 (Software Technology) of the Software Development and Operations program from Linnaeus University, aimed at providing a first practical experience with structured project planning.

4.3 Stakeholders

I will be responsible for project planning and management (including activity planning, risk management and scheduling) as well as for the project’s development (including specification, development, validation and evolution). Being an educational project, I consider the faculty’s teachers who will be grading this project as ‘clients’, since the request for this project came from their behalf and they were the ones who defined the project’s initial constraints and goals (just as the client usually defines at the start of a software project). This project lacks a specific end user, but, given that the end product is a simple game of Hangman, the end user will be everyone interested in playing it, which includes both myself and the ‘clients’ (teachers). Since there aren’t any further managerial or financial ramifications to the project, there are no further stakeholders.

4.4 Resources

Coding will be done using the IDE Microsoft Visual Studio Code and Java as a programming language (requiring, therefore, an installation of JDK as well as several Visual Studio Code Extensions to support development in Java). Git version control will be done using Visual Studio Code’s integrated terminal to update the project’s repository in GitHub. An internet browser (Google Chrome, to be more specific) will be used for other miscellaneous tasks, such as communication with the project’s “clients” and handling of project deliverables via the course’s myMoodle platform. The project plan will be edited using Microsoft Office Word.
Regarding hardware, other than my own personal computer (either a laptop running Linux or an iMac running Mac OS X), no other hardware devices will be used.

4.5 Hard- and Software Requirements

To use this product (a text-based game which runs in the console), there are no software requirements other than an operating system with a JRE allowing for the execution of java programs. No specific hardware is required either other than a personal computer with the previously mentioned software installed.

4.6 Overall project schedule

- 2019-02-08 Project plan, GitHub repository, skeleton code
- 2019-02-21 Requirements, UML behavior- and structure modelling
- 2019-03-08 Testing
- 2019-03-22 Complete project

4.7 Scope, constraints and assumptions

Scope: As mentioned in the project vision, the game will include a main menu, two different game modes (single game and 5-round game) and separate high score tables for each game mode, with data persistency between different uses using text files. This game is to be played locally by only one player and will not have any kind of multiplayer functionality. This game will be implemented as a console application and, as such, will not have a graphical user interface besides the text and symbols allowed in the console. The game is a purely text-based application and will not include any type of media files such as images, videos or audio files.

Constraints to the project: Since this is not a commercial project, its realization is not dependent on any budget and there are no financial constraints to consider. Time, however, will be an important constraint, since I have an infant son which requires my care during the day, so I depend mostly on late afternoons and evenings to develop this project (as well as remaining university work). A related constraint is my lack of knowledge – I'm currently learning everything about project management for the first time, so I need to devote a lot more time than probably necessary to planning this project, while also requiring some time for the development tasks due to my inexperience with programming Java.

Assumptions: This project assumes the end user is able to perform basic tasks with their personal computer (such as turning it on and using its operative system) and has sufficient knowledge with a terminal application to be able to follow the instructions specified on the README.md file to execute the program and interact with it through key commands.

Reflection on writing a project plan: Writing a project plan is an essential step when developing a project. The project plan is created at the beginning of a project with the goal of structuring the work that has to be done in different activities, assessing the time and effort required for each, assigning them to team members and anticipating risks and strategies to deal with them. This project plan is reassessed and updated throughout the entire project. This is my first contact with elaborating a project plan, but I can already understand the enormous asset it is in a project, especially if it is a complex project involving many team members and many different tasks. I was able to reflect on my project in a much more structured and objective way thanks to the planning that this document requires, and, now that this first version of the document is created, I feel I have a solid foundation to organize all the work that lays ahead.

5 – Iterations

5.1 Iteration 1

Timeframe: 2019-01-23 to 2019-02-08

Planned activities and estimated time:

- Writing project plan
    - Vision (1 hour)
    - Project plan subcategories (2 hours)
    - Iteration planning (1 hour)
    - Risk analysis (1 hour)
    - Time log (30 minutes)
- Creating project GitHub repository, sharing it with teachers (15 minutes)
- Writing and pushing skeleton code (15 minutes)
- Pushing project plan to the GitHub repository as a documentation file (<15 minutes)
- Doing a release in GitHub (<15 minutes)

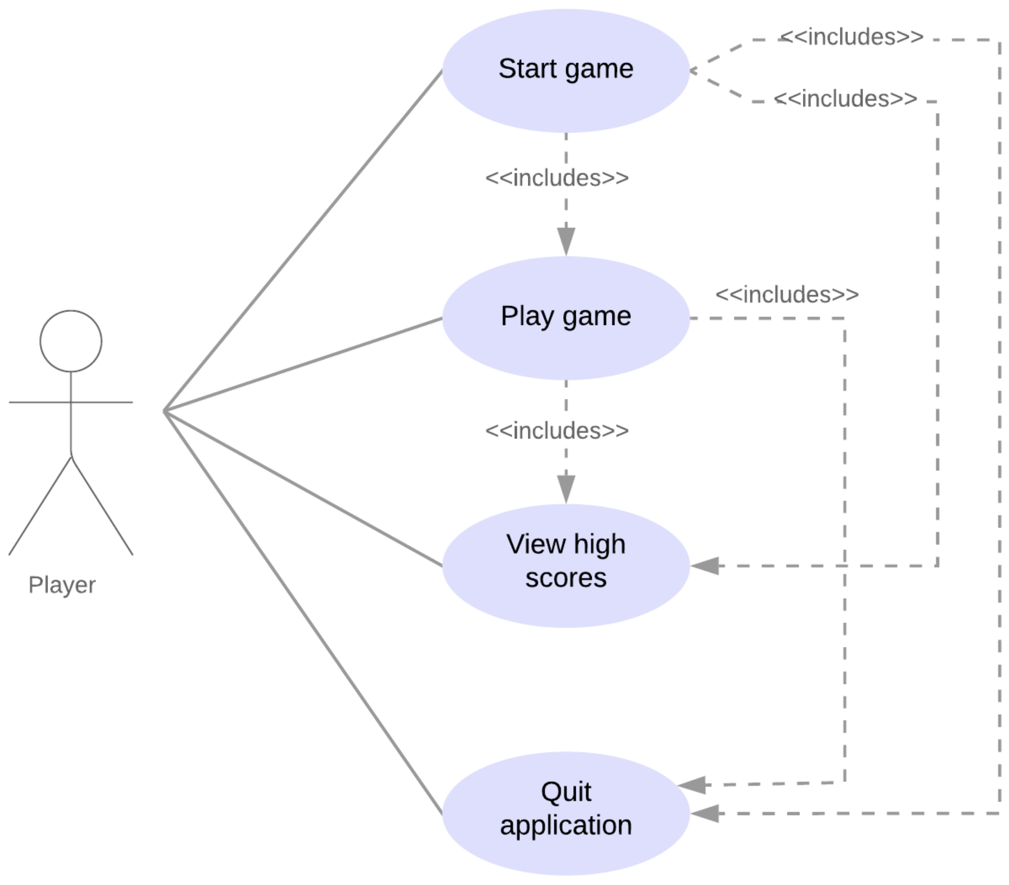
5.2 Iteration 2 – Modelling and implementation

Timeframe: 2019-02-09 to 2019-02-21

Planned activities and estimated time:

- Reading theme 2 book chapters and miscellaneous study material (12 hours)
- Watching recorded Q&A lectures (2 hours)
- Sketching use case diagram (30 minutes)
- Adapting Use Cases and writing fully dressed UC2 (1 hour 30 minutes)
- Building “play game” state chart (2 hours)
- Implementing the game core functionality (6 hours)
- Building class diagram (30 minutes)

Use Case diagram



UC 1 Start Game

Precondition: none.
Postcondition: the start menu is shown.

Main scenario:
1. Starts when the Player wants to begin a session of the hangman game.
2. The system presents the main menu with a title, the option to play a new game, view the high score tables and quit the game.
3. The Player makes the choice to start the game.
4. The system starts a new game (see Use Case 2).

Repeat from step 2

Alternative scenarios:

3.1 The Player makes the choice to view the high scores.
   1. The system presents the high scores menu (see Use Case 3)

3.2 The Player makes the choice to quit the game.
   1. The system quits the game (see Use Case 4)

4.1 Invalid menu choice
   1. The system presents an error message.
   2. Goto 2

UC 2 Play game

Precondition: the start menu is shown.
Postcondition: a new game is running.

Main scenario:
1. Starts when the Player wants to play a new game.
2. The system presents the new game menu, with a title, the option to play a single-word game, to play a 5-word game, or to go back to the main menu.
3. The Player makes the choice to start a single-word game.
4. The system starts a new single-word game, selects a random word to be guessed, displays it, and waits for player input.
5. The Player enters a letter which is contained in the word
6. The system checks the input, updates the displayed word revealing the guessed letter(s) and asks the Player for a new input

Repeat from step 5 until the player has no more tries left or the word has been guessed

Alternative scenarios:

3.1 The Player makes the choice to start a 5-word game.
   1. The system starts a new single-word game
   2. Repeat from step 5
   3. After each single game is finished, the system starts a new single-word game until a total of 5 single-word games has been played

3.2 The Player makes the choice to go back to the start menu.
1. The system returns to the start menu (see Use Case 1).

5.1 The Player enters a letter which is not a part of the word
1. The system decreases the number of tries by 1, updates the hangman drawing and asks for new user input
2. Goto 5

5.2 The Player enters an invalid character
1. The system exhibit a message and asks the Player for new input
2. Goto 5

5.3 The Player enters the word “quitgame”
1. The system asks the Player for confirmation
2. The Player confirms
3. The system terminates (see Use Case 4)

5.4 The Player enters the word “resetgame”
1. The system asks the Player for confirmation
2. The Player confirms
3. The system returns to the start menu (see Use Case 1)

5.5 The Player tries to guess the hidden word and fails
1. The system reduces the number of tries remaining, updates the hangman drawing and asks the Player for a new input
2. Goto 5

5.6 The Player tries to guess the hidden word and succeeds
1. The system displays a message informing that the Player has won and the options to go back to the start menu (see Use Case 1) or quit the application (see Use Case 4)

6.1 The Player guesses every letter in the hidden word and obtains a new high score
1. The system displays a message informing that the user has won and obtained a new high score, and asks the Player to provide a name
2. The Player enters a name
3. The system displays the options to go back to the start menu (see Use Case 1), view the high scores (see Use Case 3) or quit the application (see Use Case 4)

6.2 The Player guesses every letter in the hidden word but does not obtain a new high score
1. The system displays a message informing that the Player has won, displays the options to go back to the start menu (see Use Case 1), view the high scores (see Use Case 3) or quit the application (see Use Case 4)

6.3 The Player runs out of tries
1. The system displays a message informing that the user has lost and the options to go back to the start menu (see Use Case 1) or quit the application (see Use Case 4)

UC3 View high scores

Precondition: the start menu is shown.
Postcondition: a high scores table is shown.

Main scenario:

1. Starts when the Player wants to view the current high scores.
2. The system presents the high scores menu, with a title, the option to view high scores for single-word games, view high scores for 5-word games, or to go back to the main menu.
3. The Player makes the choice to view high scores for single-word games.
4. The system presents a table containing the top 5 high scores for single-word games and the options to return to the start menu, return to the high scores menu, or quit the application.

Alternative scenarios:

3.1 The Player makes the choice to view the high scores for 5-word games.
   1. The system presents a table containing the top 5 high scores for 5-word games and the options to return to the start menu, return to the high scores menu, or quit the application

3.2 The Player makes the choice to go back to the start menu.
   1. The system returns to the start menu (see Use Case 1).

3.3 The Player enters an invalid input
   1. The system presents an error message
   2. Goto 2

UC4 Quit game

Precondition: the game is running.
Postcondition: the application is terminated.

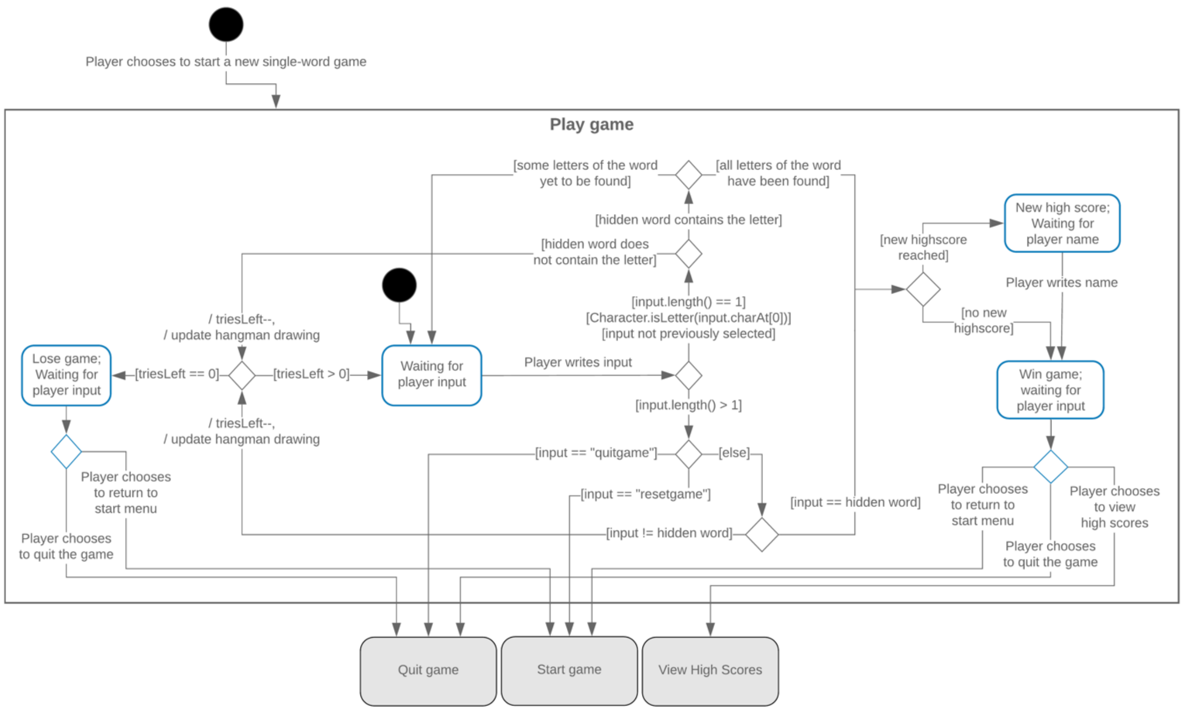
Main scenario:

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

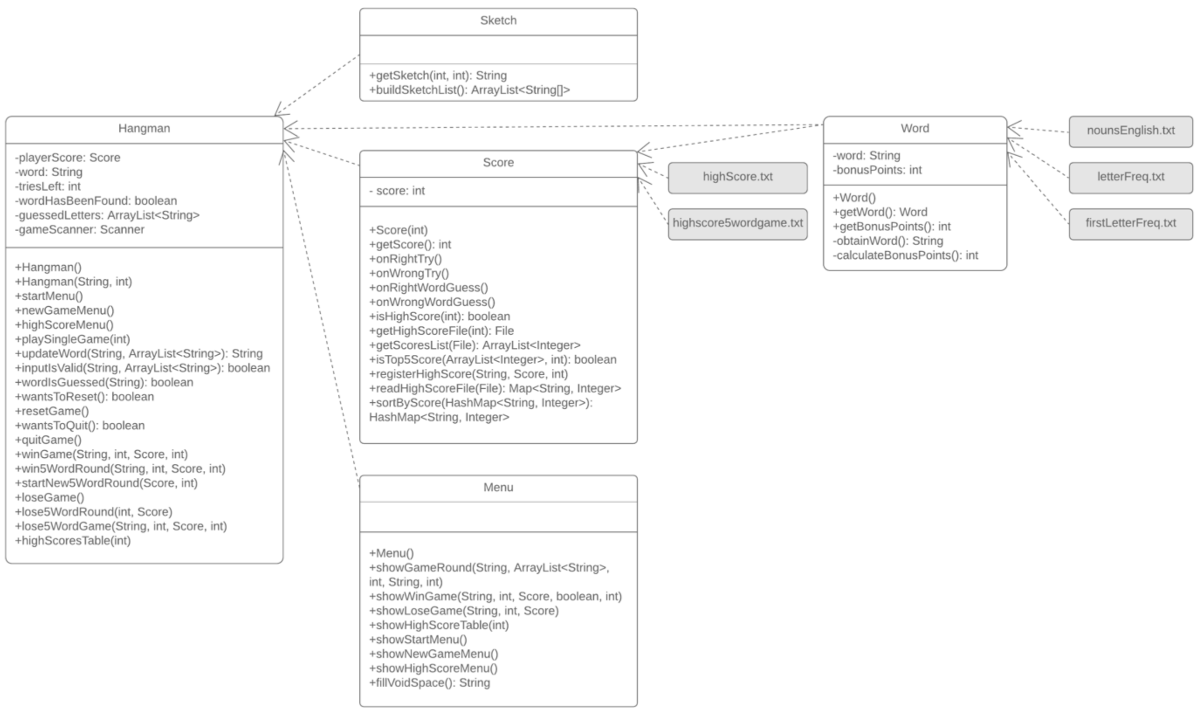
Alternative scenarios:

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

Use Case 2 (Play game) state chart


Player chooses to start a new single-word game
Play game
[some letters of the word yet to be found]
[all letters of the word have been found]
[hidden word contains the letter]
[hidden word does not contain the letter]
[input.length() == 1] [Character.isLetter(input.charAt[0])] [input not previously selected]
/ triesLeft--, / update hangman drawing
Lose game; Waiting for player input
[triesLeft == 0]
[triesLeft > 0]
Waiting for player input
Player writes input
[input.length() > 1]
/ triesLeft--, / update hangman drawing
[input == "quitgame"]
[else]
[input == "resetgame"]
[input != hidden word]
[input == hidden word]
Player chooses to return to start menu
Player chooses to quit the game
New high score; Waiting for player name
[new highscore reached]
Player writes name
[no new highscore]
Win game; waiting for player input
Player chooses to return to start menu
Player chooses to view high scores
Player chooses to quit the game
Quit game
Start game
View High Scores

Class diagram


Sketch
+getSketch(int, int): String
+buildSketchList(): ArrayList<String[]>
Hangman
-playerScore: Score
-word: String
-triesLeft: int
-wordHasBeenFound: boolean
-guessedLetters: ArrayList<String>
-gameScanner: Scanner
+Hangman()
+Hangman(String, int)
+startMenu()
+newGameMenu()
+highScoreMenu()
+playSingleGame(int)
+updateWord(String, ArrayList<String>): String
+inputIsValid(String, ArrayList<String>): boolean
+wordIsGuessed(String): boolean
+wantsToReset(): boolean
+resetGame()
+wantsToQuit(): boolean
+quitGame()
+winGame(String, int, Score, int)
+win5WordRound(String, int, Score, int)
+startNew5WordRound(Score, int)
+loseGame()
+lose5WordRound(int, Score)
+lose5WordGame(String, int, Score, int)
+highScoresTable(int)
Score
- score: int
+Score(int)
+getScore(): int
+onRightTry()
+onWrongTry()
+onRightWordGuess()
+onWrongWordGuess()
+isHighScore(int): boolean
+getHighScoreFile(int): File
+getScoresList(File): ArrayList<Integer>
+isTop5Score(ArrayList<Integer>, int): boolean
+registerHighScore(String, Score, int)
+readHighScoreFile(File): Map<String, Integer>
+sortByScore(HashMap<String, Integer>): HashMap<String, Integer>
highScore.txt
highscore5wordgame.txt
Word
-word: String
-bonusPoints: int
+Word()
+getWord(): Word
+getBonusPoints(): int
-obtainWord(): String
-calculateBonusPoints(): int
nounsEnglish.txt
letterFreq.txt
firstLetterFreq.txt
Menu
+Menu()
+showGameRound(String, ArrayList<String>, int, String, int)
+showWinGame(String, int, Score, boolean, int)
+showLoseGame(String, int, Score)
+showHighScoreTable(int)
+showStartMenu()
+showNewGameMenu()
+showHighScoreMenu()
+fillVoidSpace(): String

5.3 Iteration 3 - Testing

Timeframe: 2019-02-22 to 2019-03-08

Planned activities and estimated time:

- Writing test plan introduction and reflection (1 hour)
- Planning and writing manual test cases (2 hours)
- Running manual tests (10 minutes)
- Writing manual tests' results (30 minutes)
- Automated unit tests (1 hour)

Test objects

Manual tests

Manual tests were used extensively because many methods and functionalities either depended on user interaction or had outputs consisting of the printing of specific information to the console. Several use cased-based tests were used, covering all of the project's use cases, testing every type of interaction and input confirmation from the user.

Automated tests

Two methods from the Hangman class were adequate for automated testing because they had simple inputs and outputs:

- public boolean inputIsValid(String input, ArrayList<String> guessedLetters)
- public String updateWord (String gameWord, ArrayList<String> guessedLetters)

Unit testing also revealed to be adequate during testing after implementing the Score class and respective functionality during Iteration 4 (please refer to Section 5.4.3, item 2.5.1 – Writing and running automated unit tests).

Testing techniques

For this testing process, only dynamic testing techniques were used (i.e. techniques whereby the system is tested through its execution) and no static test were used. Unit testing was performed using automated test cases written and executed in JUnit 5. Again, in the context of development testing, these were conducted to find faults in the implementation of these methods. System testing was performed using use case-based testing and manual tests. The goal with system testing during development is to test the interaction between the system's different components and to test the system's emergent behaviours. Given the multiplicity of different interactions and the many resulting outcomes, it is often impractical to use automated tests for system testing.

Manual test cases

TC 2.1 Winning a game by guessing an entire word correctly and obtaining a high score

Use cases tested: UC1 – Start game and UC2 – Play game
Scenario: The Player starts the application, starts a new game, wins it by guessing the hidden word at once (UC2 alternate scenario 5.6) and obtains a new high score

Precondition:
- The application is executed by specifying the String “starlight” as an argument to the Hangman constructor in the game’s Main.java class.
- The file resources/highscore.txt contains either less than five scores, or at least one among the highest five that is lower than 331 points

Test steps
- Start the app
- The system shows the welcome message “---- Welcome to the world's best Hangman game ever! ----” as well as the start menu and waits for user input.
- Press “1” and enter.
- The system shows the new game menu and waits for user input.
- Press “1” and enter.
- The system displays a single game board containing no sections of the hanging man sketch, the hidden word, the number of tries left (10) and the previously entered letters (none).
- The system shows “Please enter a letter or guess the entire word (or write 'resetgame' to return to start menu or 'quitgame' to quit the program):” and waits for user input.
- Write “starlight” and press enter.
- The system displays the win game board, still with no sections of the hanging man sketch, showing the messages “YOU WON!!!!”, “You found STARLIGHT in 1 tries”, “Your score: 331” and “NEW HIGHSCORE!”. The system then displays “Please enter your nickname to register your highscore: “
- Write ‘abc’ and press enter

Expected
- The system displays the messages “highscore successfully registered as abc!” and “Please press a key to select an option: 1 – Return to Start menu 2 – Return to High scores menu 0 – Quit the application”.

Result

Succeeded	☒
Failed	☐
Comments:	-

TC 2.2 Winning a game guessing the word letter by letter, without obtaining a new high score

Use case tested: UC2 – Play game
Scenario: The Player plays a single-word game from start to finish, guessing the hidden word, but not reaching enough points to obtain a new high score.

Preconditions:
- The application is executed by specifying the String “starlight” as an argument to the Hangman constructor in the game’s Main.java class
- The file resources/highscore.txt has at least five entries, all greater than 331 points
- A single-word game is running, waiting for Player input.

Test steps
- The system shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘s’ and press enter
- The system does not update the hangman sketch, updates the hidden word, shows the message “ -- :) Nice guess! -- “, and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘t’ and press enter
- The system does not update the hangman sketch, updates the hidden word, shows the message “ -- :) Nice guess! -- “, and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘a’ and press enter
- The system does not update the hangman sketch, updates the hidden word, shows the message “ -- :) Nice guess! -- “, and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘r’ and press enter
- The system does not update the hangman sketch, updates the hidden word, shows the message “ -- :) Nice guess! -- “, and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘l’ and press enter
- The system does not update the hangman sketch, updates the hidden word, shows the message “ -- :) Nice guess! -- “, and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘i' and press enter
- The system does not update the hangman sketch, updates the hidden word, shows the message “ -- :) Nice guess! -- “, and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘g’ and press enter
- The system does not update the hangman sketch, updates the hidden word, shows the message “ -- :) Nice guess! -- “, and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘h’ and press enter

Expected
- The system doesn’t update the hangman sketch and displays the messages “YOU WON!!!!”, “You found STARLIGHT in 1 tries”, “Your score: 331”, “Please press a key to select an option: 1 – Return to Start menu 2 – View High scores 0 – Quit the application:“

Result

Succeeded	☒
Failed	☐
Comments:	-

TC 2.3 Losing a single-word game

Use case tested: UC2 – Play game
Scenario: The Player plays a single-word game from start to finish and loses by running out of tries (UC2 alternate scenario 6.3)

Preconditions:
- The application is executed by specifying the String “bee” as an argument to the Hangman constructor in the game’s Main.java class
- A single-word game is running, waiting for Player input.

Test steps
- The system shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘a and press enter
- The system updates the hidden word, shows the message “ -- :(Nope, no a’s in this word! -- “, updates the number of tries left to 9, updates the hangman sketch and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘c’ and press enter
- The system updates the hidden word, shows the message “ -- :(Nope, no c’s in this word! -- “, updates the number of tries left to 8, updates the hangman sketch and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘d’ and press enter
- The system updates the hidden word, shows the message “ -- :(Nope, no d’s in this word! -- “, updates the number of tries left to 7, updates the hangman sketch and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘f’ and press enter
- The system updates the hidden word, shows the message “ -- :(Nope, no f’s in this word! -- “, updates the number of tries left to 6, updates the hangman sketch and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘g’ and press enter
- The system updates the hidden word, shows the message “ -- :(Nope, no g’s in this word! -- “, updates the number of tries left to 5, updates the hangman sketch and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘h’ and press enter
- The system updates the hidden word, shows the message “ -- :(Nope, no h’s in this word! -- “, updates the number of tries left to 4, updates the hangman sketch and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘i’ and press enter
- The system updates the hidden word, shows the message “ -- :(Nope, no i’s in this word! -- “, updates the number of tries left to 3, updates the hangman sketch and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘j’ and press enter
- The system updates the hidden word, shows the message “ -- :(Nope, no j’s in this word! -- “, updates the number of tries left to 2, updates the hangman sketch and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “

- Write ‘k’ and press enter
- The system updates the hidden word, shows the message “ -- :(Nope, no k’s in this word! -- “, updates the number of tries left to 1, updates the hangman sketch and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘l’ and press enter

Expected
- The system displays the full hangman sketch and the messages “YOU LOSE :(” and “Word was BEE”. The system then displays “Please press a key to select an option: 1 – Return to Start menu 0 – Quit the application: “.

Result

Succeeded	☒
Failed	☐
Comments:	-

TC 2.4 Resetting an ongoing game
Use cases tested: UC2 – Play game, UC1 – Start game
Scenario: Resetting an ongoing game (main scenario, alternate scenario 5.4, where the Player enters the command ‘resetgame’ during an ongoing game)

Precondition: A single-word game is running, and the system is waiting for the Player’s input.

Test steps
- The system shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘resetgame’ and press enter.
- The system shows “Do you really want to end this game and return to the start menu? (“y” = yes, “n” = no): “
- Write ‘y’ and press enter.

Expected
- The system shows “resetting the game…”
- The system shows the welcome message “--- Welcome to the world’s best Hangman game ever! ---” and the start menu.

Result

Succeeded	☒
Failed	☐
Comments:	-

TC 2.5 Winning a round of a 5-word game and proceeding to the next round

Use cases tested: UC1 – Start game and UC2 – Play Game
Scenario: The Player starts a 5-word game, plays and wins the first round and choses to proceed to the 2nd round.

Preconditions:
- The application is executed by specifying the String "bee" as an argument to the Hangman constructor in the game's Main.java class
- The start menu is shown

Test steps
- The system displays the start menu and the message "please press a key to select an option: "
- Write '1' and press enter
- The system displays the new game menu and the message "please press a key to select an option: "
- Write '2' and press enter
- The system displays the game board for the 1st round of 5, with no sections of the hanging man sketch and displaying the heading "GAME 1 OF 5", the hidden word and the messages "let's play!", "Number of tries left: 10" and "Please enter a letter or guess the entire word (or write 'resetgame' to return to start menu or 'quitgame' to quit the program):"
- Write 'bee' and press enter
- The system does not update the hangman sketch and displays the messages "YOU WON!!!!", "You found BEE in 1 tries" and "Your score: 239". The system does not show any message concerning a new high score. The system displays "Please press a key to select an option: 1 – Proceed to round 2 of 5 2 - Leave the current 5-word game and return to the Start menu 0 – Quit the application: ".
- Write '1' and press enter

Expected
- The system displays the game board for the 2nd round of 5 with the heading "GAME 2 OF 5", a different hidden word and the messages "let's play!", "Number of tries left: 10" and "Please enter a letter or guess the entire word (or write 'resetgame' to return to start menu or 'quitgame' to quit the program): "

Result

Succeeded	☒
Failed	☐
Comments:	-

TC 2.6 Losing a round of a 5-word game and proceeding to the next round

Use case tested: UC1 – Start game and UC2 – Play game

Scenario: The Player starts a 5-word game, plays and loses the first round and choses to proceed to the 2nd round.

Preconditions:

- The application is executed by specifying the String “bee” as an argument to the Hangman constructor in the game’s Main.java class
- The start menu is shown

Test steps

- The system displays the start menu and the message “please press a key to select an option: ”
- Write ‘1’ and press enter
- The system displays the new game menu and the message “please press a key to select an option: “
- Write ‘2’ and press enter
- The system displays the game board for the 1st round of 5 with the heading “GAME 1 OF 5”, the hidden word and the messages “let’s play!”, “Number of tries left: 10” and “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘a and press enter
- The system updates the hidden word, shows the message “ -- :(Nope, no a’s in this word! -- “, updates the number of tries left to 9, updates the hangman sketch and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘c’ and press enter
- The system updates the hidden word, shows the message “ -- :(Nope, no c’s in this word! -- “, updates the number of tries left to 8, updates the hangman sketch and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘d’ and press enter
- The system updates the hidden word, shows the message “ -- :(Nope, no d’s in this word! -- “, updates the number of tries left to 7, updates the hangman sketch and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘f’ and press enter
- The system updates the hidden word, shows the message “ -- :(Nope, no f’s in this word! -- “, updates the number of tries left to 6, updates the hangman sketch and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘g’ and press enter
- The system updates the hidden word, shows the message “ -- :(Nope, no g’s in this word! -- “, updates the number of tries left to 5, updates the hangman sketch and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘h’ and press enter
- The system updates the hidden word, shows the message “ -- :(Nope, no h’s in this word! -- “, updates the number of tries left to 4, updates the hangman sketch and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘i’ and press enter

- The system updates the hidden word, shows the message “ -- :(Nope, no i’s in this word! -- “, updates the number of tries left to 3, updates the hangman sketch and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘j’ and press enter
- The system updates the hidden word, shows the message “ -- :(Nope, no j’s in this word! -- “, updates the number of tries left to 2, updates the hangman sketch and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘k’ and press enter
- The system updates the hidden word, shows the message “ -- :(Nope, no k’s in this word! -- “, updates the number of tries left to 1, updates the hangman sketch and shows “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “
- Write ‘l’ and press enter
- The system displays the lose game board, with the completed hangman sketch, the messages “YOU LOSE :(”, “Word was BEE” and “Current score: 0”. The system displays “Please press a key to select an option: 1 – Proceed to round 2 of 5 2 - Leave the current 5-word game and return to the Start menu 0 – Quit the application: “.
- Write ‘1’ and press enter

Expected
- The system displays the game board for the 2nd round of 5 with the heading “GAME 2 OF 5”, a different hidden word and the messages “let’s play!”, “Number of tries left: 10” and “Please enter a letter or guess the entire word (or write ‘resetgame’ to return to start menu or ‘quitgame’ to quit the program): “

Result

Succeeded	☒
Failed	☐
Comments:	-

TC 3.1 Viewing high score tables and quitting the game
Use cases tested: UC3 – View high scores, UC4 – Quit Game
Scenario: Viewing both high score tables (for single games and for 5-word games) and quitting the application

Precondition: The start menu is shown

Test steps
- The system displays the start menu and the message “please press a key to select an option”
- Write ‘2’ and press enter
- The system displays the high scores menu and the message “please press a key to select an option”
- Write ‘1’ and press enter
- The system displays the single game high scores table and the message “Please press a key to select an option: 1 – Return to Start menu 2 – Return to High scores menu 0 – Quit the application”.
- Write ‘2’ and press enter

- The system displays the high scores menu and the message “please press a key to select an option”
- Write ‘2’ and press enter
- The system displays the 5-word game high scores table and the message “Please press a key to select an option: 1 – Return to Start menu 2 – Return to High scores menu 0 – Quit the application”.
- Write ‘0’ and press enter
- The system shows “Do you really want to quit the application? (“y” = yes, “n” = no): “
- Write ‘y’ and press enter

Expected
- The system shows “Bye bye!” and terminates

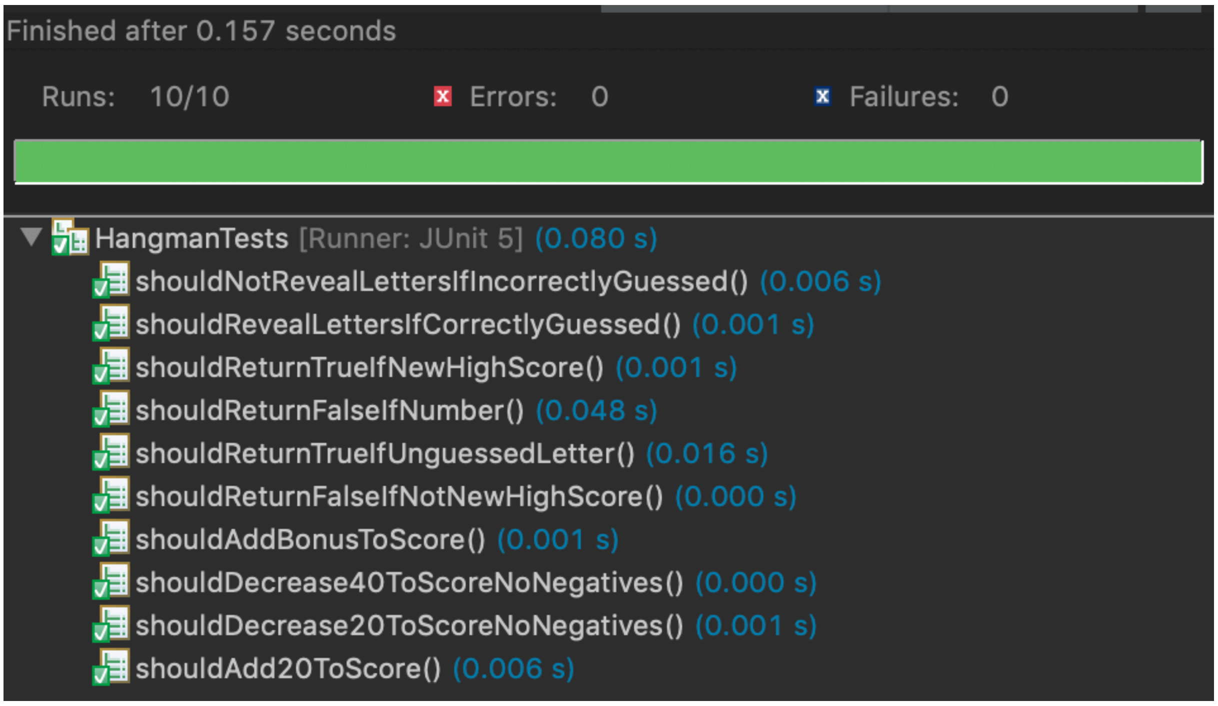
Result

Succeeded	☒
Failed	☐
Comments:	-

Manual tests report

Manual tests	UC1	UC2	UC3	UC4
TC2.1	1/OK	1/OK	0	0
TC2.2	0	1/OK	0	0
TC2.3	0	1/OK	0	0
TC2.4	1/OK	1/OK	0	0
TC2.5	1/OK	1/OK	0	0
TC2.6	1/OK	1/OK	0	0
TC3.1	0	0	1/OK	1/OK
COVERAGE & SUCCESS	4/OK	6/OK	1/OK	1/OK

Automated Unit Tests (execution)



```
Finished after 0.157 seconds

Runs:   10/10          Errors:   0          Failures:   0

HangmanTests [Runner: JUnit 5] (0.080 s)
    shouldNotRevealLettersIfIncorrectlyGuessed() (0.006 s)
    shouldRevealLettersIfCorrectlyGuessed() (0.001 s)
    shouldReturnTrueIfNewHighScore() (0.001 s)
    shouldReturnFalseIfNumber() (0.048 s)
    shouldReturnTrueIfUnguessedLetter() (0.016 s)
    shouldReturnFalseIfNotNewHighScore() (0.000 s)
    shouldAddBonusToScore() (0.001 s)
    shouldDecrease40ToScoreNoNegatives() (0.000 s)
    shouldDecrease20ToScoreNoNegatives() (0.001 s)
    shouldAdd20ToScore() (0.006 s)
```

5.4 Iteration 4 – Iterative implementation of remaining features

Timeframe: 2019-03-09 to 2019-03-22

The deliverable of this final iteration was the finished product, a complete and playable version of the game. I chose to plan different sub-iterations, since several key product features (as specified in the original vision statement and scope) were still to be implemented: score functionality, Use Case 3 (view high scores), 5-word games and the hanging man sketch. I also intended to move the methods responsible for building and displaying the game menus into a class of their own, to improve the code's organization and readability.

The following sections enumerate the tasks defined for each sub-iteration, together with a short description/motivation for each and their originally estimated time.

5.4.1 – Sub-iteration 4.1: Refactoring code and documentation according to previous feedback

The first step before starting the sub-iterations concerning the implementation of new feature was to refactor the previous code and documentation according to feedback from previous assignments, so that errors from the previous iterations could be fixed before producing this iteration's deliverables.

1.1- Reviewing the project plan

- No relevant changes seemed to be needed according to feedback. Since the goal was to implement all the missing features in this iteration, the original vision and scope remained in effect and did not need reviewing.
- Estimated time: 15 minutes

1.2- Importing deliverables from iteration 2 and iteration 3 into the project plan

- Importing, trimming text to content relevant to iteration 4 and re-formatting
- Estimated time: 2 hours

1.3- Correcting the Use Case Diagram after feedback

- Removing unnecessary distinction between single-word games and 5-word games from the diagram
- Estimated time: 15 minutes

1.4- Correcting the Use Case 2 State Diagram

- Although no changes were needed according to feedback, I noticed that I had included in the diagram pseudo-states that are actually part of UC 1 – start game – and UC 4 – quit game – which, therefore, had to be removed
- Estimated time: 30 minutes

1.5 – Updating time log and writing reflections

- Estimated time: 15 minutes

No correction to testing was needed in this sub-iteration since I had not received any objective feedback on the existing testing procedures themselves.

5.4.2 – Sub-iteration 4.2: Implementing game score, high scores and Use Case 3 (View High Scores)

I chose to start the implementation of new features by implementing all the different functionalities involving score. I chose to begin with this feature because, besides requiring a lot of new code, the implementation of the different score functionalities would depend considerably on modifying existing methods in the Hangman class, and it would be better to do this as soon as possible, before the existing code becomes even more extensive. In contrast, the remaining planned tasks for iteration 4 (reorganizing the game's menus, implementing 5-word games and the hanging man sketches) would probably depend less on already existing code, so these could be done later in the iteration.

2.1- Planning the different tasks in the sub-iteration

- Estimated time: 1 hour

2.2 – Coding

- **2.2.1 – Basic functionality**
  - Creating Score class and building basic methods (for adding and subtracting points with right or wrong guesses)
  - Estimated time: 45 minutes
- **2.2.2 – Modifying Hangman class methods to modify score during a game**
  - Estimated time: 1 hour
- **2.2.3 – Implementing High Score functionality**
  - Building methods for creating, reading and writing high scores in external text files
  - Estimated time: 2 hours
- **2.2.4 – Modifying Hangman class methods to use high score functionality**
  - Estimated time: 1 hour
- **2.2.5 – Displaying High Score (implementing Use Case 3)**
  - Writing methods to build and display the high score tables by reading the external high score text files
  - Estimated time: 1 hour
- **2.2.6 – Modifying Hangman class methods to display high score tables**
  - Estimated time: 30 minutes

2.3 – Updating class diagram after implementing the Score class

- Estimated time: 45 minutes

2.4 – Updating Use Case 2 state diagram after implementing the Score class

- The introduction of high score functionalities meant new interactions with the player when he/she won a game, requiring an update to the play game state diagram
- Estimated time: 45 minutes

2.5 – Updating tests after implementing the Score class

- **2.5.1 – Writing and running automated unit tests**
  - Some of the methods in the Score class are suitable for automated unit testing because they have simple inputs and outputs and don't require complex user interaction. Examples are methods such as onRightTry or onWrongWordGuess, which merely add or subtract to the current score, or the method isTop5Score, which returns a Boolean value representing if a score is at least the 5th highest on record. For these methods, automated unit tests were written in JUnit5.
  - Estimated time: 1 hour

- **2.5.2 – Writing and executing manual tests**
  - The methods in the Score class involving I/O operations with external files were not suitable for automated testing because they required user interaction and their testing was more dependent on specific scenarios.
  - I opted therefore to adjust the existing test case TC 2.1, which ran through the main scenario in the play game use case, to include obtaining a new high score. Additionally, two new test cases would be written: TC 2.2, to test winning a game without obtaining a new high score and TC 3.1, to test Use Case 3 (view high scores) and displaying both high score tables.
  - Estimated time: 1 hour
- **2.5.3 – Updating manual test report**
  - Estimated time: <15 minutes

2.6 – Updating time log and writing reflections
- Estimated time: 30 minutes

5.4.3 – Sub-iteration 4.3: Implementing the Menu class

The Menu class would consist simply of a number of static methods returning the String objects representing each menu. This class would therefore act merely as a 'container' for these methods. Although no functional requirement depended on the implementation of a separate Menu class, and even though this class would not bring any added functionality, I still considered important to create it, in order to:

- Improve the manageability and readability of the Hangman class (and the entire project)
- Improve the separation of functions between different classes, leaving the Hangman class essentially to handle the interaction with the user and the flow of the game.
- Facilitate the later implementation of the hanging man sketches by uniformizing the dimensions of every menu's container box for the hanging man sketch

3.1 – Planning the different tasks in this sub-iteration
- Estimated time: 1 hour

3.2 – Creating the Menu class and its methods
- This included moving the methods previously in the Hangman class to the Menu class and also adjusting and uniformizing the dimensions of the different game boards to later accommodate the hanging man sketches.
- Estimated time: 1 hour

3.3 – Adapting the code in the Hangman class to use the Menu class methods
- Estimated time: 15 minutes

3.4 – Updating class diagram after implementing Menu class
- Estimated time: 30 minutes

The methods in the Menu class aren't adequate for automated testing because they have complex String objects as outputs and depend on user interaction. Instead, they could be tested using existing manual tests, since by now they encompassed every Use Case and covered every existing menu in the application. Therefore, no new tests were needed.

3.5 – Updating time log and writing reflections
- Estimated time: 15 minutes

5.4.4 – Sub-iteration 4.4: Implementing 5-word games

A 5-word game is simply a sequence of 5 individual games during which the player's score accumulates (regardless of winning or losing each individual game) and, after the 5th game is over, may generate a new 5-word game high score. By this stage, the score and high score functionalities had already been implemented and tested, so it seemed adequate to implement 5-word games.

4.1 – Planning the different tasks in this sub-iteration
- Estimated time: 1 hour

4.2 – Coding
- **4.2.1 – Modifying the Hangman class to support 5-word games**
  - Estimated time: 45 minutes
- **4.2.2 – Modifying methods in the Menu class to adjust the display information during 5-word games**
  - Estimated time: 30 minutes
- **4.2.3 – Adapting high score functionality of Score class to 5-word games**
  - Estimated time: 1 hour

4.3 – Updating class diagram
- Estimated time: 30 minutes

4.4 – Testing
- **4.4.1 – Writing and executing manual test cases for 5-word games**
  - The added functionality regarding 5-word games concerned especially the transition between different game "rounds" independently of winning or losing each round. This kind of interaction is, once again, more suitable to manual test cases.
  - Two manual test cases were therefore planned, one (TC 2.5) playing and winning round 1 of 5 and proceeding to round 2 of 5, and another (TC 2.6) playing and losing round 1 of 5 and proceeding to round 2 of 5.
  - Estimated time: 30 minutes
- **4.4.2 – Updating manual tests report**
  - Estimated time: <15 minutes

4.5 – Updating time log and writing reflections
- Estimated time: 15 minutes

5.4.5 – Sub-iteration 4.5: Implementing the hanging man sketches

The final feature implemented in iteration 4 were the sketches of the hanging man to be displayed in the game board and updated after every failed guess.

5.1 – Planning the different tasks in this sub-iteration

- Estimated time: 30 minutes

5.2 – Coding

- The different game boards had been uniformized in sub-iteration 4.3, allowing the same hangman sketch to be displayed in different menus. This allowed each sketch to consist of an array of Strings, each representing a row of the sketch to be appended to each row of the menu. Because the choice of sketch is dependent on the number of failed guesses, the number of remaining tries could be used as input to return the appropriate array of Strings.
- **5.2.1 – Creating the class Sketch and building the different "sketches"**
  - Estimated time: 30 minutes
- **5.2.2 – Implementing the required functionality to obtain the appropriate sketch**
  - Estimated time: 30 minutes

5.3 – Update class diagram after implementing the Sketch class

- Estimated time: 30 minutes

5.4 – Updating manual test cases to explicitly test the hanging man sketches and re-running the tests

- Once again, similarly to the also text-based Menu class, the Sketch class is not suited for automated testing due to the output of String objects and the dependency on user interaction. It is, however, easily integrated in the many manual tests already covering the Use Case 2 (play game).
- Estimated time: 45 minutes

5.5 – Updating time log and writing reflections

- Estimated time: 15 minutes

6 – Risk analysis

6.1 List of risks

Risk	Probability	Impact
<u>Estimation risks:</u> - Underestimate the time required to develop the software - Underestimate the degree of difficulty associated to specific features of the project's implementation	High Moderate	Serious Tolerable
<u>People risks:</u> - I become ill and unable to work on the project - I don't have enough knowledge to develop the project according to planned specifications	Low Low	Catastrophical Serious
<u>Requirement risks:</u> - Changes to requirements that require major design rework are proposed	Moderate	Serious
<u>Technology risks:</u> - I lose access to my project files - The hardware I use to develop the project stops functioning/becomes inaccessible	Low Low	Catastrophical Serious
<u>Tools risks:</u> - Microsoft Visual Studio Code IDE's Java Extensions don't work as expected	Low	Tolerable

6.2 Strategies

Risk	Strategy
Estimation risks	Anticipate which requirements are non-essential and can be sacrificed in case time becomes critical.
People risks	No avoidance or minimization possible against staff illness in this case (only me). If encountered, be prepared to discuss with the course's staff the possibility of an extended deadline for a deliverable or the need for a re-take of the project.
Requirement risks	Discuss the changed requirements in detail with the client (course's staff) in order to re-plan and resume development as soon as possible.
Technology risks	Using a light, open source IDE, version control and an online GitHub repository minimizes the dependency on a specific computer (in case I lose access to my home computers, it's easy to install the needed tools in another unit to carry on with the project). Using cloud-based tools such as Microsoft Office Online and OneDrive allows me to also work in the project's plan independently of any specific hardware unit and without needing to physically backup my files.
Tools risks	If any problem occurs due to the use of Microsoft Visual Studio Code with Java Language extensions, I can easily change the IDE to Eclipse and use Git version control via a separate terminal application.

Reflection on risk analysis: When planning a project, it is very important to plan ahead for the many different types of risks that can happen. This allows the project manager to anticipate strategies for handling these different risks in order to minimize their impact on the project. Although some categories can be defined for the different types of risks, this is an aspect of project planning that is not entirely objective, and it's easy to understand how much this process depends on the project manager's experience when you have no experience at all and try to plan this step. However, trying to apply risk management principles to this particular project, which is very different than the examples one usually reads about (large software developers with multiple teams, etc.) definitely makes it more understandable.

7 – Time Log

Task	Estimated time (hours:minutes)	Real time (hours:minutes)

Iteration 1 tasks		
1- Project plan – Vision	01:00	01:15
2- Project plan – Section 4 (sub-categories)	02:00	04:30
3- Project plan – Iteration planning	01:00	00:45
4- Project plan – Risk analysis	01:00	01:30
5- Project plan – Time log	00:30	00:30
6- Creating and sharing GitHub repository	00:15	00:15
7- Writing and pushing skeleton code	00:15	00:15
8- Pushing project plan into GitHub repository	<00:15	<00:15
9- Doing a release in GitHub	<00:15	<00:15

Iteration 2 tasks		
1- Reading book chapters and study material	12:00	**20:30**
2- Watching recorded lectures	02:00	**03:45**
3- Sketching Use Case Diagram	00:30	**01:45**
4- Adapting Use Cases, writing fully dressed UC2	01:30	01:45
5- Building UC2 State Chart	02:00	**04:15**
6- Implementing game	06:00	**15:45**
7- Building Class Diagram	00:30	00:45

Reflection about time discrepancies for iteration 2:

- **1** – It took much longer than expected to read and study this iteration's learning material, because I hadn't anticipated how much I would need to complement the course's book with online tutorials and other resources.

- **1** and **2** – For both the study material and the recorded lectures, I failed to correctly anticipate how much time it would take for me to write notes when studying.

- **3** and **5** – It took more time than expected to draw both the use case diagram and the UC 2 state chart, because I needed to search examples and information about the use of various UML components in such diagrams.

- **6** – I failed to divide this task in more detailed components, and that prevented me to analyse where exactly in the implementation I have lost more time. I can however say that, globally, I had underestimated the implementation's difficulty, particularly the amount of time I required to figure out how to properly implement user input confirmation. I also ran into problems using Visual Studio Code with the Java programming language and had to change IDEs to Eclipse (which I had already anticipated in my risk analysis during iteration 1).

Iteration 3 tasks		
1- Writing test plan introduction and reflection	01:00	**01:45**
2- Planning and writing manual test cases	02:00	02:30
3- Running manual tests	<00:15	<00:15
4- Writing manual test results	00:30	00:15
5- Planning, writing and running automated unit tests	01:00	**04:30**

Reflection about time discrepancies for iteration 3:

- **1** – Writing the test also took a bit longer than anticipated due to time mostly spent on writing an adequate reflection and organizing my thoughts on the Test Objects and Testing Techniques sections.

- **5** – The most relevant difference is how much longer it took to perform automated unit testing. This difference was due to the changes I had to make in several methods and constructors to make the code testable, namely by introducing parameters to pass data between methods).

Iteration 4 tasks

Sub-iteration 1:

1.1- Reviewing the project plan	00:15	00:15
1.2- Importing previous deliverables into project plan	02:00	**03:30**
1.3- Correcting the Use Case Diagram	00:15	<00:15
1.4- Correcting the Use Case 2 State Diagram	00:30	00:30
1.5- Updating Time Log	00:15	00:15

Sub-iteration 2:

2.1- Planning tasks for the sub-iteration	01:00	**02:30**
2.2.1- Coding: Basic functionality	00:45	**01:30**
2.2.2- Coding: Modifying Hangman methods to modify the score during a game	01:00	**02:00**
2.2.3- Coding: High score functionality	02:00	**02:30**
2.2.4- Coding: Modifying Hangman methods to use high score functionality	01:00	01:00
2.2.5- Coding: Displaying high score tables (UC 3)	01:00	01:30
2.2.6- Coding: Modifying Hangman methods to show high score tables	00:30	00:30
2.3- Updating Class Diagram	00:45	00:45
2.4- Updating UC2 State Diagram	00:45	01:00
2.5.1- Testing: Automated Unit Tests	01:00	01:15
2.5.2- Testing: Manual Tests	01:00	01:45
2.5.3- Testing: Updating manual tests report	<00:15	<00:15
2.6- Updating time log, writing reflections	00:30	00:30

Reflection about time discrepancies:

- The earlier stages of this sub-iteration (**2.1 to 2.2.3**) took longer than anticipated because I had not expected the difficulty of planning an implementation on top of already existing code. I had to constantly rethink my pseudocode in this initial stage to find the most efficient way to utilize already existing methods in the Hangman class to pass the player's score among themselves.
- **2.2.3, 2.2.4** – A particular problem I had not thought about when working on the implementation of high score functionality was exception handling (IOException and FileNotFoundException), because of the dependency on exterior text files. I had to spend some time planning and writing try/catch blocks to handle these exceptions appropriately so that they wouldn't be thrown by methods in the Hangman class.

Sub-iteration 3:

3.1- Planning tasks for the sub-iteration	01:00	01:30
3.2- Creating the Menu class and its methods	01:00	01:15
3.3- Adapting Hangman class to use Menu class	00:15	00:15
3.4- Updating class diagram	00:30	00:15
3.5- Updating time log	00:15	00:15

Sub-iteration 4:

4.1- Planning tasks for the sub-iteration	01:00	00:45
4.2.1- Coding: Modifying Hangman class to support 5-word games	00:45	**01:45**
4.2.2- Coding: Modifying methods in the Menu class to adjust the display information during 5-word games	00:30	00:30
4.2.3- Coding: Adapting high score functionality of Score class to 5-word games	01:00	00:30
4.3- Updating class diagram	00:30	00:15
4.4.1- Testing: Writing and executing manual test cases for 5-word games	00:30	00:30
4.4.2- Testing: Updating manual tests report	<00:15	<00:15
4.5- Updating time log and writing reflections	00:15	00:15

Reflection about time discrepancies:

- **4.2.1-** It took longer than anticipated to adapt the code in Hangman class to 5-word games, because initially I had thought to do it using a global class variable to indicate if the game in progress was a single-word or a 5-word game. However, after feedback from iteration 3 concerning the preferable use of local variables and method parameters, I spent some time trying to find a way to implement 5-word games without resorting to class variables, and eventually succeeded.

Sub-iteration 5:

5.1- Planning tasks for the sub-iteration	00:30	00:30
5.2.1- Coding: creating Sketch class and the sketches	00:30	00:30
5.2.2- Coding: Implementing functionality	00:30	00:30
5.3- Updating class diagram	00:30	00:15
5.4- Updating and re-running manual tests	00:45	00:30
5.5- Updating time log	00:15	<00:15