SOFTWARE DEVELOPMENT
Hangman Game

Name: Johan Dahlberg
Email: jd222qd@student.lnu.se
Date: 18-03-2019

Github repository link:
https://github.com/jd222qd/jd222qd_dv600

1 | Revision History

Date	Version	Description	Author
08-02-2019	1	Implemented skeleton code and wrote a document including general information, vision, project plan, risk analysis & time log.	Johan Dahlberg
22-02-2019	2	Implemented enough features to make the game playable (navigate between menus, play game, quit game etc.) Write Use Cases. Also modelled the implemented features in UML by creating a use case diagram, ‘play game’ state machine, extended state machine and a class diagram.	Johan Dahlberg
08-03-2019	3	Implemented the visual drawing of the hangman feature & heavily refactored the code into more classes. A test plan was written, as well as manual test-cases and automated unit tests. The tests were performed on the code and activity diagrams were created. Finally a test report was written	Johan Dahlberg
22-03-2019	Final	Implemented the final highscore feature which completed the project. Wrote and performed manual test-cases and automated unit tests for the highscore feature. Updated use case diagram, state machine diagrams, class diagram and made activity diagrams for the new feature. Finally, revisited parts of previous iterations, made updates if necessary and finished the whole project.	Johan Dahlberg

2 | General Information

Project Summary	
Project Name	Project ID
Hangman Game	HANGMAN_GAME
Project Manager	Main Client
Johan Dahlberg	Anyone
Key Stakeholders	
Programmer Tester Product Manager End user	
Executive Summary	
The purpose of this project is to create a Hangman game for the 1DV600 - Software Technology course, with the goal of learning more about the software development process. The game will be played in the console of eclipse by 1 player at a time. The player will be able to choose a difficulty and then play then game by guessing 1 letter at a time. If a game is won, the player will be given a score and the option to save that score. The top 10 highest scores will be tracked and viewable in a highscore list. It will also be possible to navigate between menus and quit the application.	

3 | Vision

The game is being developed with the purpose of learning more about different software development approaches and how to plan bigger projects in the future.

The goal of this project is to implement the game “Hangman” in a text based fashion, with basic functionality such as being able to navigate menus, guess letters of a randomly chosen word and quit the application. Additional features such as having a highscore list and being able to choose difficulty is also a goal that will give the game more depth and complexity. Since the game is being developed with the purpose of learning more about software development, it will only be playable in the console of Eclipse (perhaps other IDE’s also).

The finished product should look something like this:

Main Menu
Start Game
Check Highscores
Quit Application

Choose Difficulty
Easy
Medium
Hard

```
  _____
  |    |
  |    o
  |   / |
  |    |
  |
 _|_
|   |_____
|         |
|_________|
```

_ E W S P A P E _

Guess a letter! (3 guesses left)
Back to Main Menu
Quit Application

Reflection:

Writing a vision for the project was pretty simple due to the clear instructions of what needs to be achieved in the project. The benefit of creating a vision document like this is that everyone involved in the project gets a clearer understanding of everything that needs to be done, and what the goals are. That makes it easier to split the project into different steps and facilitate the project planning process to the assign everyone their tasks. It is important to get everyone involved in the project on the same course, which is the purpose of this vision document.

4 | Project Plan

4.1 Introduction

The goal of the project is to complete the Hangman game within the given timeframe (around 7-8 weeks).

4.2 Justification

It's a project that will be created with the purpose of learning more about software development and how to plan bigger projects in the future.

4.3 Stakeholders

Programmer - The one who writes the program and produces the source code. The programmer is responsible for maintaining the code and keeping it well-structured for simplified maintainability and potential refactoring.

Tester - Tests the code to make sure it works and that the program performs in the intended way. Also tries to find any potential bugs/issues so that they can be corrected.

Product Manager - Responsible for planning, estimating, and scheduling project development and assigning people to tasks. They supervise the work to ensure that it is carried out to the required standards, and they monitor progress to check that the development is on time and within budget.

End user - The person who will use the finished product and make sure that it fulfills the requirements they made at the beginning och the project's development process.

4.4 Resources

The resources needed for the project are:

The time to complete the project, which is about 7-8 weeks and will be spent mostly on writing the code for the project, writing documentation and attending lectures in the 1DV600 course. A computer will also be needed to run the software used to develop the product. The software used to create the product will be *Eclipse IDE for Java Developers (Version 2018.09 (4.9.0))* & *Java JDK 11.0.1.* The language used when writing the code will be Java. The literature that will be used is *Big Java Late Objects,* by Cay Hortstmann & *Software Engineering (10th edition),* by Ian Sommerville. Additional information from lectures in the 1DV600 course and resources from the internet, such as https://stackoverflow.com , will also be used.

4.5 Hard- and Software Requirements

To create the program you need:

Hardware requirements:
Computer

Software requirements:
Eclipse IDE for Java Developers (Version 2018.09 (4.9.0))
Java JDK 11.0.1 (java development kit)

To use the program you need:

Hardware requirements:
Computer

Software requirements:
Eclipse IDE for Java Developers (Version 2018.09 (4.9.0))
Java JDK 11.0.1 (java development kit)

4.6 Overall Project Schedule

Iteration 1 - (08-02-2019)
Implement skeleton code and write a document containing general information about the project, a vision about the project, a project plan and a risk analysis.

Iteration 2 - (22-02-2019)
Implement enough features to make the game playable (navigate between menus, play game, quit game etc.) Model the features in UML and create a Use Case Diagram as well as an Extended Use Case Model. Also make a 'play game' State Machine, an Extended State Machine and a Class Diagram.

Iteration 3 - (08-03-2019)
Implement the visual drawing of the hangman feature. Write a test plane, manual test-cases and automated unit tests. Perform the tests on the code and create activity diagrams. Also write a test report.

Iteration 4 - (21-03-2019)
Implement the highscore feature as well as write and perform test-cases + automated unit tests for the feature. Revisit previous iterations and update Class Diagram and Use Cases with the highscore feature. This will finish the project.

4.7 Scope, Constraints and Assumptions

Scope: This game is a text based Hangman game with 3 different difficulties, “Easy”, “Medium” & “Hard”. Depending on the chosen difficulty, the player will have a set amount of guesses. 13 for “Easy”, 10 for “Medium” and only 7 for “Hard”. When a round of the hangman game is started, a random word will be chosen by the system, for the user to guess. The word will be represented using underscores. At any point during the round, the player has the option to guess a letter, return back to the main menu, or quit the application. Whenever the player makes a correct guess, the guessed letter will appear on the correct position(s) within the word. If the player makes an incorrect guess, 1 guess will be removed and the system will print 1 extra part of the hangman figure, using unicode characters. If the player runs out of guesses, the game is lost. The player will then be returned back to the main menu. If the player manages to guess the word before running out of guesses, the game is won. A score will then be calculated based on the chosen difficulty and the amount of guesses left. The player then has the option to save this score.If the player chooses to save it, a name must be entered. The score will then be put on the highscore list if it’s good enough to make it into the top 10. The highscore list can be accessed via the Main Menu and will display the 10 highest scores.

Out of scope: The game will not include multiplayer mode because the developer is unexperienced in making online applications. Multiplayer could be implemented locally, but a decision was made not to. The reason for this is because the game is run in the console of Eclipse, which logs all user input. This means that if Player 1 picks a word by entering it into the console, Player 2 would be able to scroll up and see what the picked word is. This defeats the purpose of the game and will therefore not be included. A GUI won’t be included in the game either because of the developer’s lack of knowledge within the field.

Constraints: The allocated time for the project is the biggest constraint. The more time that is available, the more features can be implemented. Another constraint is that the game is only able to be played through the console of Eclipse (and possibly other IDE’s). The functional requirements for the project such as starting a game of hangman, viewing highscores, choosing difficulty, guessing a letter & quitting the application are also constraints. Another constraint might be potential lack of knowledge in a specific field, for example within GUI. Non-functional requirements are non-existent in this project.

Assumptions: I made the assumption that the user will be able to read and understand English on a basic level, because the game and the documentation about the game, is written in English. No additional assumptions were made due to the clear requirements.

Reflection:

The purpose of writing a project plan is to show the way in which the project will be finished. It is important because it makes it clearer for the people working on it, what needs to be done at a specific time. The project plan also makes it possible to measure progress towards the plan and then alter the plan if necessary. Writing this project plan was fairly simple because the deadlines for each iteration, and what needs to be done in each one, was already given to us.

5 | Iterations

5.1 Iteration 1

Time available: 2 weeks **Due date: 08-02-2019**

In this iteration the project will be initiated and planning will be the biggest factor. A lot of time will be spent on learning about software processes, planning and managing projects. This will be done by attending Lecture 2 & 3 in the 1DV600 course, and also by reading chapters 2-3 + 22-23 in *Software Engineering (10th edition)* by Ian Sommerville.
It is then time to start documenting the project by creating a document with the following contents:

- Project summary
- Vision
- Project plan (including introduction, justification, stakeholders, resources, hard- and software requirements, overall project schedule, scope, constraints, assumptions and a reflection)
- Risk analysis (including a list of risks, strategies for dealing with risks and a reflection)
- Time log

Finally, some skeleton code will be implemented for the project. This will be the least time consuming part because no actual features need to be included.

To quickly summarize, the 3 parts on this iteration is:

- Learning,
- Documenting
- Coding

Out of these, learning and documenting will make up the majority of the iteration, with coding only being a very small part.

As previously stated, a lot of time will be allocated to learning about everything, since this is the first iteration. When documenting the project, writing the Project plan and the Risk analysis involve many steps than the other tasks and will therefore be allocated more time. Writing a project summary, vision and time log does not include that many steps and will therefore not be allocated as much time.

Below is a table of all the tasks to be done as well as the estimated time it will take to perform each trask.

Task	Estimated Time
Read chapter 2 in *Software Engineering (10th edition)* by Ian Sommerville.	1h
Read chapter 3 in *Software Engineering (10th edition)* by Ian Sommerville.	1h
Learn about Software Processes (Lecture 2).	2h
Read chapter 22 in *Software Engineering (10th edition)* by Ian Sommerville.	1h
Read chapter 23 in *Software Engineering (10th edition)* by Ian Sommerville.	1h
Learn about Planning and Managing projects (Lecture 3).	2h
Write a project summary.	30min
Write a vision for the project.	1h
Write a reflection about the vision.	30min
Write an introduction.	10min
Write a justification.	15min
Write about stakeholders.	10min
Write about resources for the project.	30min
Write about hard- and software requirements	15min
Write the project schedule.	30min
Write about the project scope.	30min
Write about the project constraints.	30min
Write about project assumptions.	15min
Write a reflection about the project plan.	30min
Write the list of risks for the project.	15min
Write about strategies concerning the risks.	30min
Write a reflection about the risk analysis.	30min
Write skeleton code for the game.	30min
Write revision history for the iteration.	10min
Write time log for the iteration.	30min

5.2 Iteration 2

Time available: 2 weeks **Due date: 22-02-2019**

In this iteration the main focus will be on modelling features using UML, and then implementing these features to make the game playable. To do this, we have to first learn about Software modeling, architecture & the transformation from design to implementation, as well as how UML works. This will be done by attending Lecture 4,5,6,7 & 8 in the 1DV600 course, as well as reading chapter 4-7, 15 & 20 in *Software Engineering (10th edition)* by Ian Sommerville.

UML documents to be created:

- Use case diagram
- Extended use case model
- 'Play game' state machine diagram
- Extended state machine diagram
- Class diagram

Sufficient features will be implemented to make the game playable, i.e navigate between menus, play game, quit game, guess a letter. When playing the game, the amount of guesses will be displayed and a correct guess will display the guessed letter at the correct position in the word. Additional features may be added, such as: Difficulties, High scores, & Display hangman pictures.

The iteration can be divided into 4 parts:

- Learning
- Modelling
- Documenting
- Coding

A lot of time will be allocated to learning because this it will be important in order to actually being able to perform some of these tasks, such as creating UML diagrams. On the topic of UML, the most time consuming part will most likely be creating different Use Cases and a Use Case Diagram. This is because the amount of Use Cases that need to be written. Creating the state machine diagrams will probably take a bit longer than the class diagram because the class diagram is based on the existing code, which is easy to interpret.
The time for implementing the code shouldn't be too high considering the use cases will be written before the code is implemented and the requirements of what is to be done should be clear at that point.

Below is a table of all the tasks to be done as well as the estimated time it will take to perform each trask.

Task	Estimated Time
Read chapter 4 in *Software Engineering (10th edition)* by Ian Sommerville.	1h
Read chapter 5 in *Software Engineering (10th edition)* by Ian Sommerville.	1h
Read chapter 20 in *Software Engineering (10th edition)* by Ian Sommerville.	1h
Learn about Systems and Software Modeling Lecture 4).	2h
Read chapter 7 in *Software Engineering (10th edition)* by Ian Sommerville.	1h
Learn about Modeling with UML (Lecture 5).	2h
Read chapter 6 in *Software Engineering (10th edition)* by Ian Sommerville.	1h
Learn about Software Architecture (Lecture 6).	2h
Read chapter 15 in *Software Engineering (10th edition)* by Ian Sommerville.	1h
Learn about Software Design (Lecture 7).	2h
Learn about Transformation From Software Design to Implementation (Lecture 8).	2h
Create Use Case Diagram	30min
Create Use Case 1 - Start Application	15min
Create Use Case 2 - Launch Game	15min
Create Use Case 3 - Quit Application	15min
Create Use Case 4 - Check Highscores	15min
Create Use Case 5 - Choose Difficulty	15min
Create Use Case 6 - Guess Letter	15min
Create ‘Play Game’ State Machine.	30min
Create Extended State Machine.	45min
Implement Use Case 1 - Start Application	1h
Implement Use Case 2 - Launch Game	1h
Implement Use Case 3 - Quit Application	1h
Implement Use Case 5 - Choose Difficulty	1h
Implement Use Case 6 - Guess Letter	2h
Create Class Diagram	30min

Write revision history for the iteration.	10min
Write time log for the iteration.	30min

5.3 Iteration 3

Time available: 2 weeks **Due date: 08-03-2019**

In this iteration, the main focus is to plan, perform and document tests on the project. However, addition features to the project may also be added. To learn about software testing techniques Lecture 9,10 & 10.5 will be attended and chapter 8 in *Software Engineering (10th edition)* by Ian Sommerville will be read.

The visual drawing of the hangman feature will be implemented. A test plan will be written , containing what to test and how to test it. Manual test-cases will be written and performed on all 6 Use Cases, as well as Automated Unit Tests for most of the classes. Activity diagrams will also be made to complement the manual test-cases. Finally, a test report will be written for the manual test-cases and the automated unit tests, as well as a reflection about testing in general.

A decent amount of time will be allocated to learning all of the materials concerning testing in order to being able to create and execute well written tests. Not a lot of time will be will be allocated to describing what to test and how, instead the focus will be on creating the Manual Test-Cases as well as the Automated Unit tests. This is because of the sheer quantity of tests that need to be created and the run. Writing the reports about the Manual Test-Cases and the Automated Unit tests should be fairly easy once the actual tests are made. The allocated time for the reports will therefore not be that much.

Task	**Estimated Time**
Read chapter 8 in *Software Engineering (10th edition)* by Ian Sommerville.	1h
Learn about Software Testing (Lecture 9).	2h
Learn about Testing Techniques (Lecture 10).	2h
Learn about Testing (Lecture 10.5).	2h
Implement visual drawing of hangman feature	1h
Write what to test and how.	30min
Create and perform Manual Test-Case on Use Case 1	30min
Create and perform Manual Test-Case on Use Case 2	30min

Create and perform Manual Test-Case on Use Case 3	30min
Create and perform Manual Test-Case on Use Case 4	30min
Create and perform Manual Test-Case on Use Case 5	30min
Create and perform Manual Test-Case on Use Case 6	30min
Create activity diagrams for all Manual Test-Cases.	2h
Write test report for Manual Test-Cases	45 min
Create Automatic Unit test for HangmanGame.java	45min
Create Automatic Unit test for HangmanMenus.java	45min
Create Automatic Unit test for Word.java	45min
Create Automatic Unit test for DrawHangman.java	45min
Create automated unit test coverage and success report.	10min
Write reflection about testing.	30min
Write revision history for the iteration.	10min
Write time log for the iteration.	30min

5.4 Iteration 4

Time available: 2 weeks **Due date: 08-03-2019**

In this iteration, the project will be completed. No learning will be done this time. The focus will be to implement the remaining feature (highscores) to finish the project. Use Cases will be written for this new feature and the Use Case diagram will be updated, as well as the Class Diagram. New Manual Test-Cases for these Use Cases also need to be created and Automated Unit tests for the HighScore class. Activity Diagrams will in turn be written for the Manual Test-Cases and the test reports will be updated.

A good amount of time will be allocated to implement the new feature and then write and update any needed documentation regarding the feature. For example Diagrams, Use Cases and Tests. Once everything has been updated, the project will be finished.

Task	Estimated Time
Implement highscore feature.	2h
Write Use Case 7 - Save highscore	15min
Write Use Case 8 - Enter Gamer Name	15min
Create and perform Manual Test-Case on Use Case 7	30min
Create and perform Manual Test-Case on Use Case 8	30min
Create Automatic Unit test for HighScores.java	30min
Create Activity Diagram for Test-Cases on UC 7-8.	20min
Update Use Case Diagram.	5min
Update Class Diagram.	5min
Update Manual Test-Case report	10min
Update Automated Unit test Code Coverage and Success report.	5min
Write revision history for the iteration.	10min
Write time log for the iteration.	30min

6 | Risk Analysis

6.1 List of risks

Risk	Probability	Impact
The time required to develop the software is underestimated.	Low	Catastrophic
Loss of code progress due to an accident of some kind .	Low	Serious
The size of the software is underestimated.	Moderate	Serious
New requirements.	Low	Serious
Lack of knowledge.	Moderate	Tolerable
Key staff are ill at critical times in the project.	Moderate	Insignificant

6.2 Strategies

The time required to develop the software is underestimated - Make sure to use all of the time available and update the project plan if necessary. Have at least 1-2 weeks extra to work with in case anything goes wrong.

Loss of code progress due to an accident of some kind - Make sure to always save all your work at least every hour, or whenever a big change has been made. Also make backups and put them in a safe place, such as dropbox, google drive etc.

The size of the software is underestimated - Divide the project into subtasks and estimate how much time each subtask will take. Once the project has been going on for a while, you can compare your progress to the estimation and then update the plan if needed.

New requirements - Structure the project in a way to make it easier to adapt it to new requirements. Have a clear understanding of what kind of requirements are needed to avoid redoing things.

Lack of knowledge - Try to get an understanding of what needs to be done and start reading into those specific subjects as early into the project planning as possible.

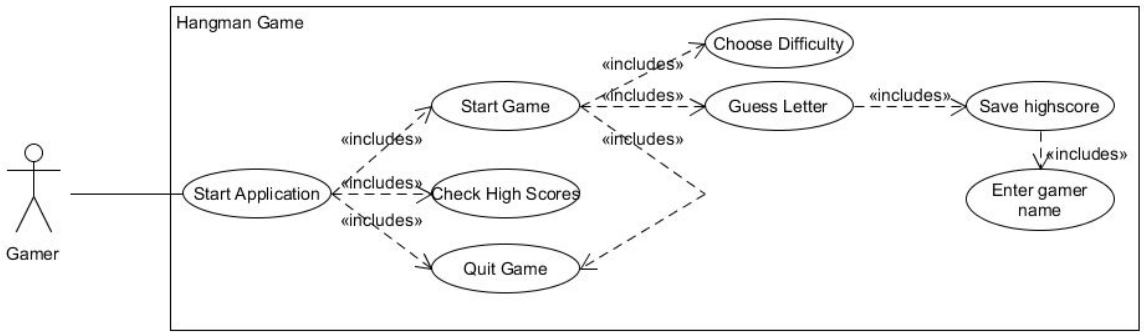
Key staff are ill at critical times in the project - Have your work available on a laptop so you can work from home if needed.

Reflection:

The purpose of writing a risk analysis is to easier understand the potential risks that are involved when working on a project. This makes it easier to plan for those risks and avoid them if possible. If one of the listed risks where to occur, there will be a fitting strategy on how to deal with that specific situation, which can save both time and money, as well as the project itself. It’s not always easy to write a risk analysis because there may risks that you forget to add, or can’t think of at the time.

7 | Use cases

Extended Use Case Model + Diagram



UC 1 Start Application

Precondition: None.

Postcondition: The application is launched.

Main scenario

1. The application starts.
2. The system presents the main menu with the option to play, check high scores, and quit the application.
3. The Gamer makes the choice to start the game.
4. The system starts the game (see Use Case 2).

Repeat from step 2

Alternative scenarios

3.1 The Gamer makes the choice to quit the application.

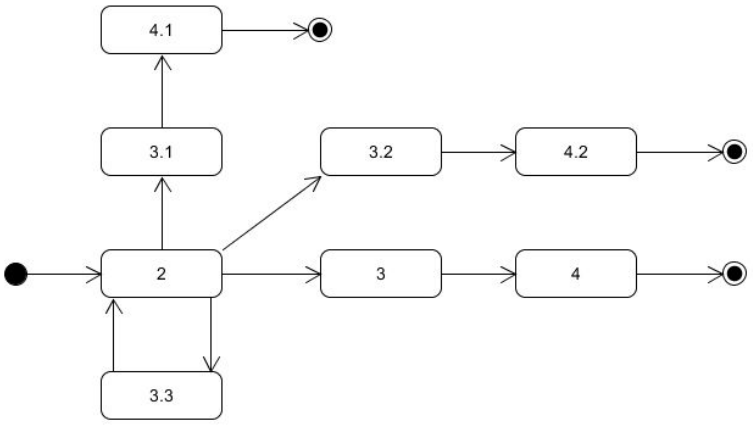
4.1 The system quits the game (see Use Case 3)

3.2 The Gamer makes the choice to check high scores.

4.2 The system displays the high scores (see Use Case 4)

3.3 The Gamer enters an invalid input. The system presents an error message. Goto step 2 in Main Scenario.

Activity Diagram


4.1
3.1
3.2
4.2
2
3
4
3.3

UC 2 Launch Game

Precondition: None.

Postcondition: A game of hangman is initiated.

Main scenario

1. Goto Use Case 5.

Alternative scenarios

None.

Activity Diagram

UC 3 Quit Application

Precondition: The application is running.

Postcondition: The application is terminated.

Main scenario

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

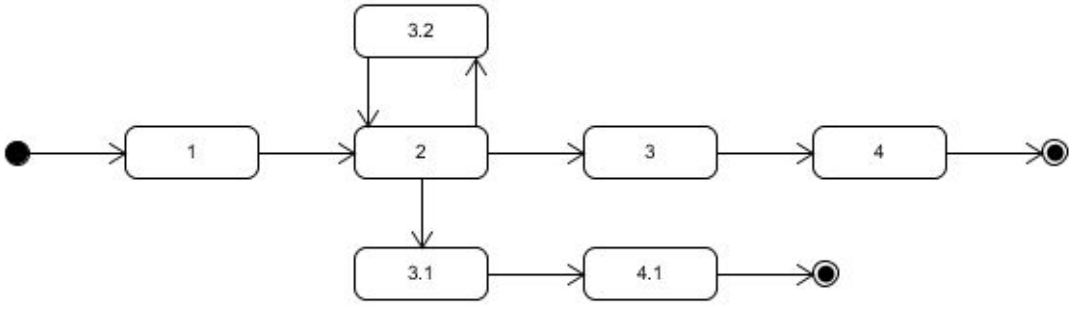
Alternative scenarios

3.1. The Gamer does not confirm

4.1 The system returns to its previous state (UC 1 / UC 6)

3.2 The Gamer enters an invalid input. Goto step 2 in Main Scenario.

Activity Diagram



UC 4 Check high scores

Precondition: None.

Postcondition: The high scores are displayed.

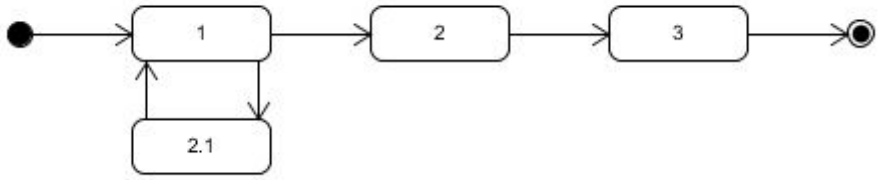
Main scenario

1. The system prints the top 10 high scores and presents the option to return to Main Menu.
2. The Gamer chooses to go back.
3. Goto Use Case 1, step 2.

Alternative scenarios

2.1 The Gamer enters an invalid input. Goto step 1 in Main Scenario.

Activity Diagram



UC 5 Choose Difficulty

Precondition: None.

Postcondition: A difficulty is chosen for the game.

Main scenario

1. The system displays the difficulty options (Easy, Medium, Hard) and the option to return to Main Menu.
2. The Gamer chooses the ‘Easy’ difficulty.
3. The system launches the game on ‘Easy’ difficulty.
4. Goto Use Case 6.

Alternative scenarios

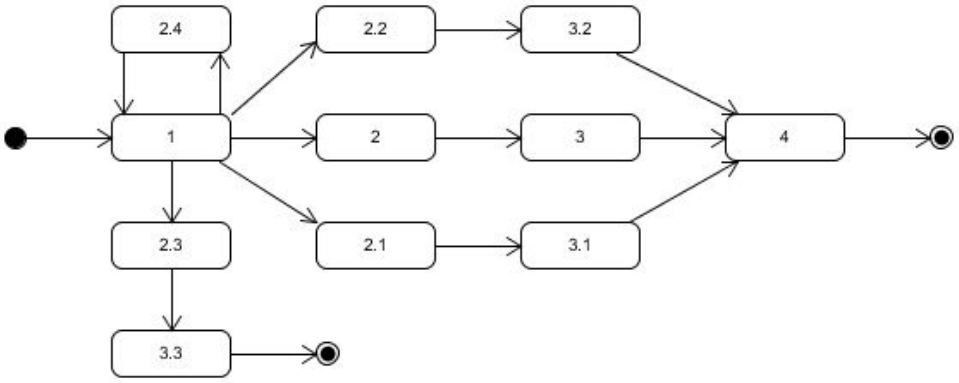
2.1 The Gamer chooses ‘Medium’ difficulty.
    3.1 The system launches the game on ‘Medium’ difficulty.
    4 Goto Use Case 6.

2.2 The Gamer chooses ‘Hard’ difficulty.
    3.2 The system launches the game on ‘Hard’ difficulty.
    4 Goto Use Case 6.

2.3 The Gamer chooses to Return to Main Menu.
    3.3 Goto Use Case 1, step 2.

2.4 The Gamer enters an invalid input. Goto step 1 in Main Scenario.

Activity Diagram



UC 6 Guess Letter

Precondition: A game of hangman is currently running.

Postcondition:

Main scenario

1. The system randomly picks a word (based on difficulty) from a pre-defined list and presents the word, using underscores.
2. The system displays the option to guess a letter, go back to Main Menu, or Quit the application.
3. The Gamer makes a correct guess.
4. The system displays a ‘correct’ message and displays the position(s) of the guessed letter, within the word.Goto step 2.

Alternative scenarios

3.1 The Gamer makes an incorrect guess.

4.1 The system displays a ‘incorrect’ message and displays the hangman figure with an additional part. Goto step 2 in Main Scenario.

3.2 The Gamer chooses to go back to Main Menu.

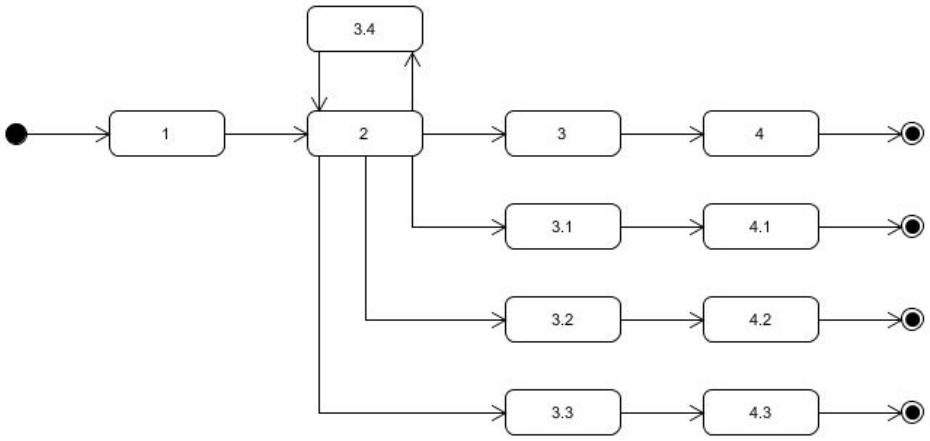
4.2 Goto Use Case 1, step 2.

3.3 The Gamer chooses to Quit the application.

4.3 Goto Use Case 3.

3.4 The Gamer enters an invalid input. Goto step 2 in Main Scenario.

Activity Diagram



UC 7 Save highscore

Precondition: The gamer has won a game of hangman.

Postcondition: The gamer is asked to enter his gamer name or will be returned back to the main menu.

Main scenario

1. The system asks the gamer if he/she wants to save the score.
2. The gamer chooses to save the score.
3. Goto use case 8.

Alternative scenarios

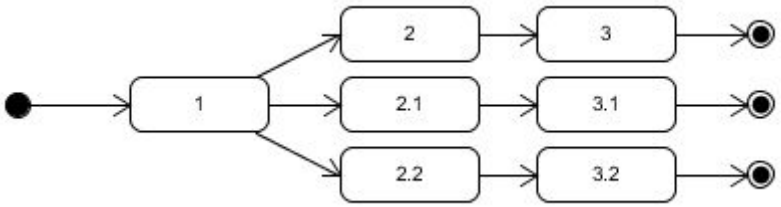
2.1. The gamer chooses not to save the score

3.1 The gamer is returned to the main menu (goto Use Case 1, step 2)

2.2 The gamer enters an invalid input.

3.2 Goto step 1 in Main Scenario.

Activity Diagram



UC 8 Enter gamer name

Precondition: The gamer has chosen to save his/her score.

Postcondition: The gamer's score is added to the highscore list (if the score is high enough).

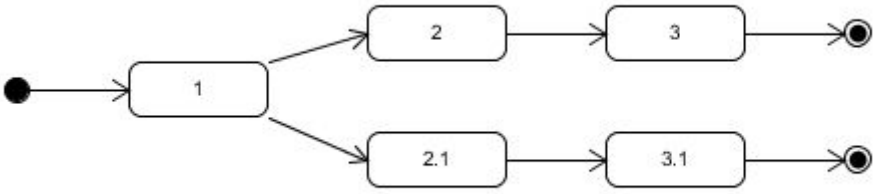
Main scenario

1. The system prompts the gamer to enter a name.
2. The gamer enters a correct name.
3. Goto use case 1, step 2.

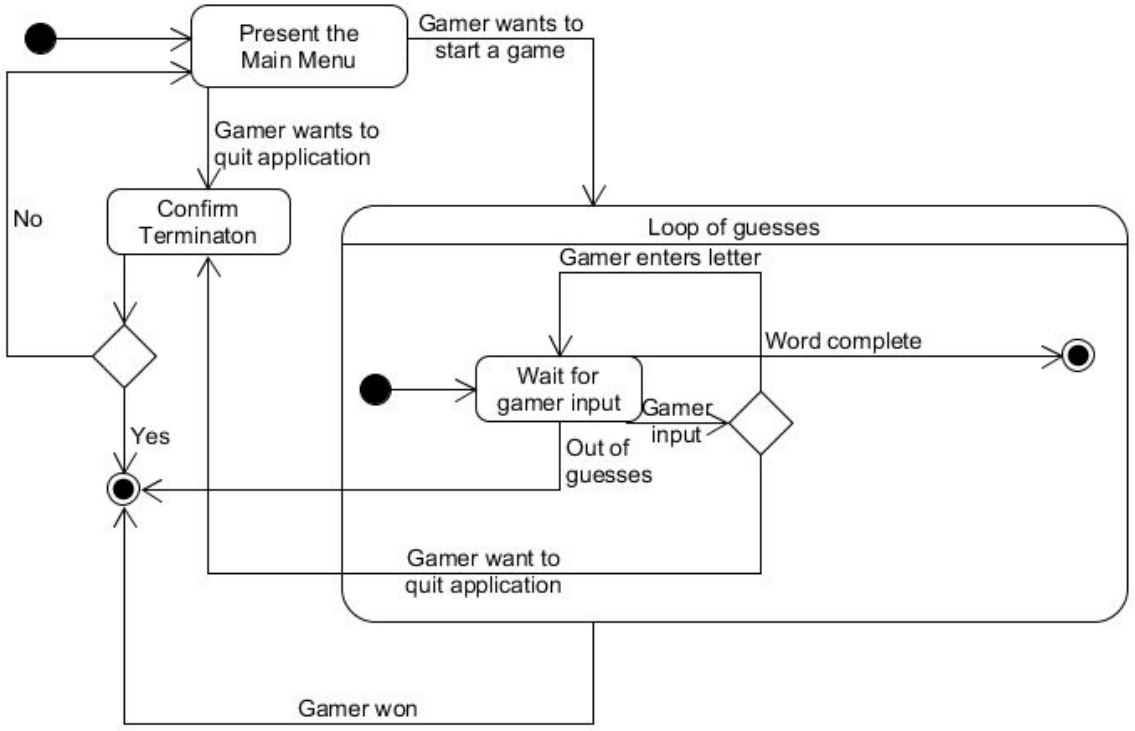
Alternative scenarios

2.1. The gamer enters an incorrect name
    3.1 Goto step 1 in main scenario.

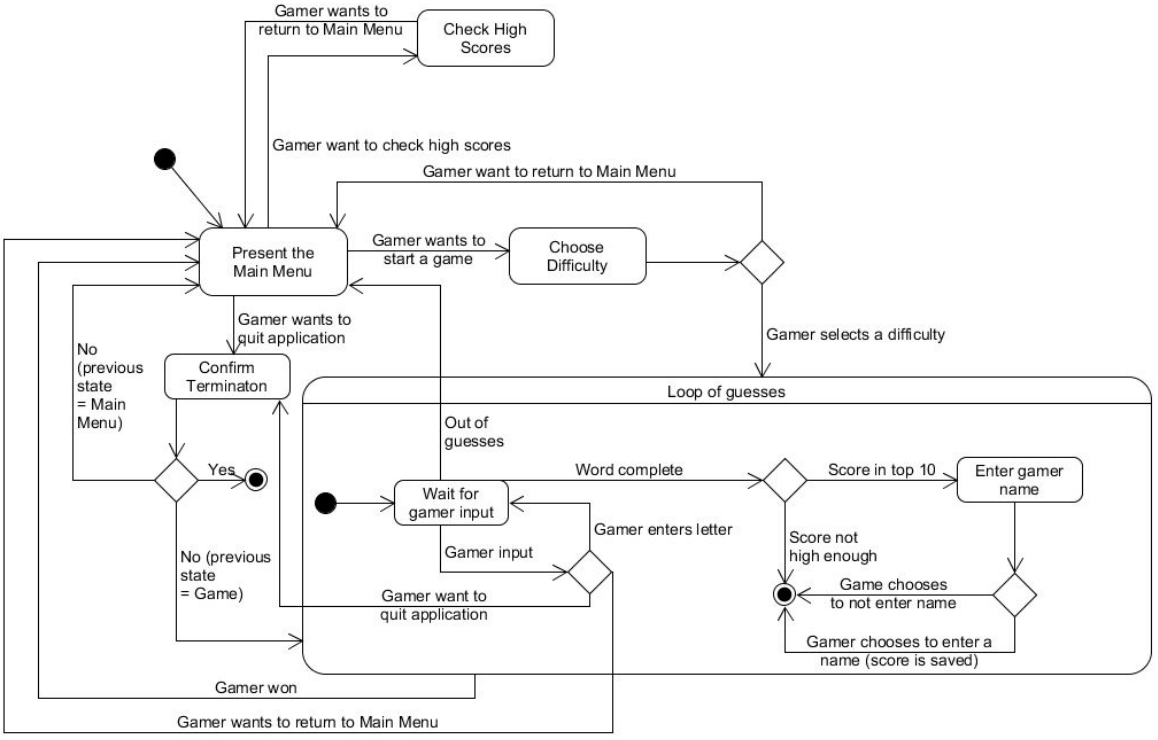
Activity Diagram



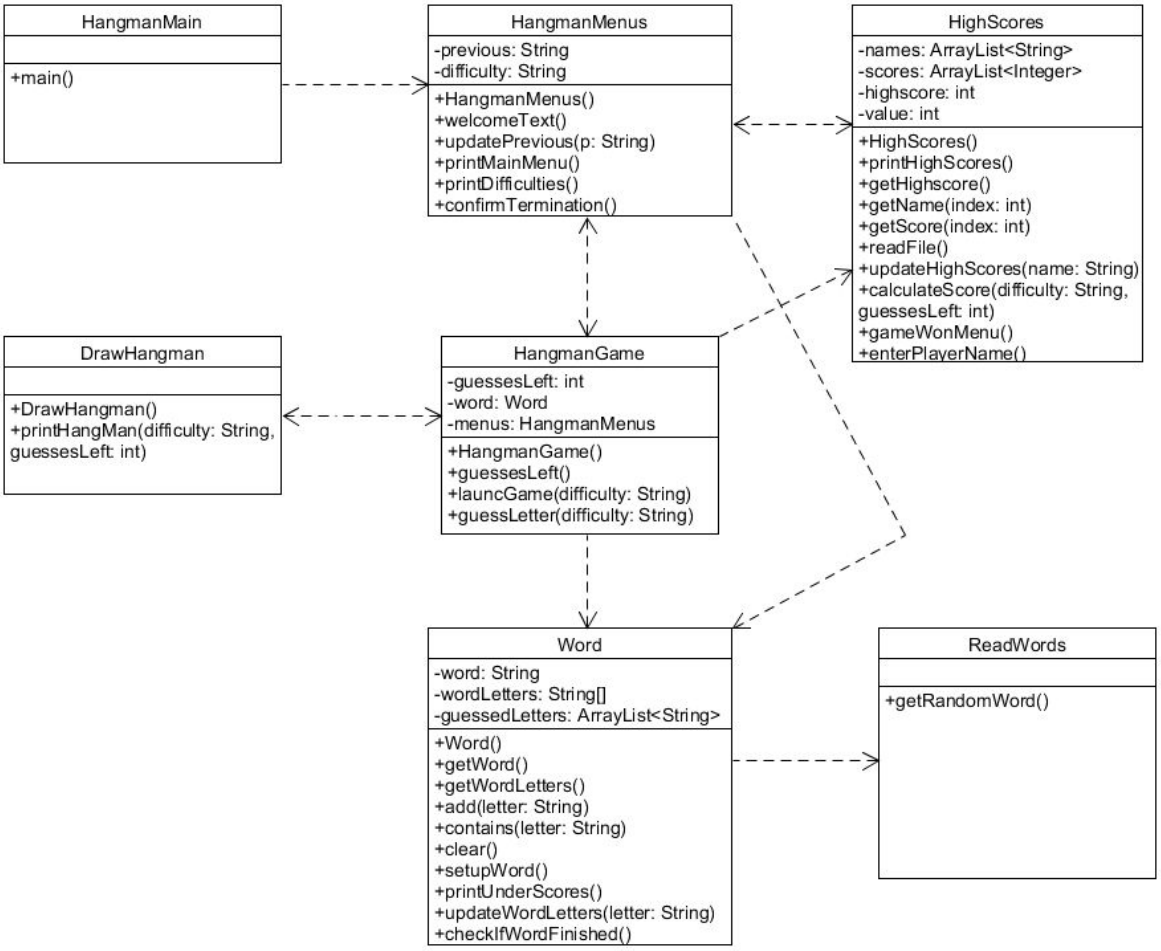
State Machine diagram(play game)



State Machine diagram (extended)



Class diagram



8 | Testing

Test Plan

Objective

The objective is to test all the current implemented code for the finished project and to make sure it works as intended, removing any potential bugs.

What to test and how

I intend to test all Use Cases (UC1- UC8) by running dynamic manual test-cases. I will also write automated unit tests for as many methods as possible in the different classes. I will not be able to test everything with the unit tests due to time constraints. However, what is not covered in the automated unit tests will be covered in the manual test-cases.This means that even if the code-coverage appears low for the unit tests, majority of the code has actually been covered in the manual test-cases.

NOTE: To perform the automated tests, line 37 needs to be commented away in the ReadWords.java file. Also remove the comment infront of line 38.

To clarify, it **should** look like this when testing:

```
//return word; //comment this line away to test
return "TEST"; //remove the comment on this line to test
```

Not like this:

```
return word; //comment this line away to test
//return "TEST"; //remove the comment on this line to test
```

Manual Test-Cases

TC1.1 Game start successful

Use case: UC1 Start Application

Scenario: Game start successful

The main scenario of UC1 is tested where the gamer chooses to start a game.

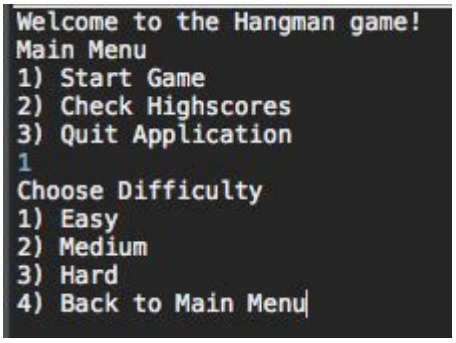
Precondition: None

Test steps

- Start the application
- The system presents the main menu with the option to start game, check high scores, and quit the application.
- Enter the key “1” and press enter.
- The system “starts” the game (see Use Case 2).

Expected

- The system should “start a game” by bringing the user to the difficulty menu (see Use Case 2).



TC1.2 Quit application confirmation successful

Use case: UC1 Start Application

Scenario: Quit application confirmation successful

Alternative scenario of UC1 is tested where the gamer chooses to quit the application.

Precondition: None

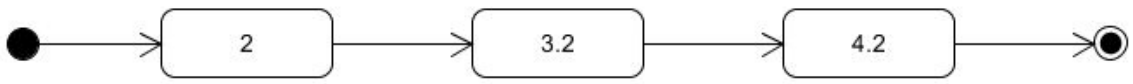
Test steps

- Start the application
- The system presents the main menu with the option to start game, check high scores, and quit the application.
- Enter the key “3” and press enter.
- The system prompts the user to confirm termination (see Use Case 3).

Expected

- The system should prompt the user to confirm termination by bringing the user to the next menu (see Use Case 3).

```
Welcome to the Hangman game!
Main Menu
1) Start Game
2) Check Highscores
3) Quit Application
3
Are you sure you want to quit the application?
1) Yes
2) No
```



TC1.3 Check highscores successful

Use case: UC1 Start Application

Scenario: Check highscores successful

Alternative scenario of UC1 is tested where the gamer chooses to check the highscores.

Precondition: None

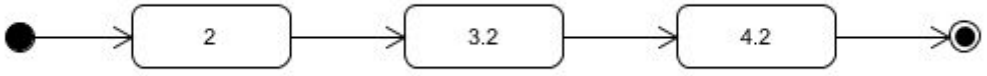
Test steps

- Start the application
- The system presents the main menu with the option to start game, check high scores, and quit the application.
- Enter the key “2” and press enter.
- The system starts displays the highscores (see Use Case 4).

Expected

- The system should print the highscores (see Use Case 4).

```
Welcome to the Hangman game!
Main Menu
1) Start Game
2) Check Highscores
3) Quit Application
2
Player | Score

test3 - 600
test4 - 450
Tanner - 400
test2 - 300
itworks:) - 200
Larry - 200
Hank - 150
test1 - 100
test:) - 100
Frank - 50

1) Back to Main Menu
```



TC1.4 Return on invalid input successful

Use case: UC1 Start Application

Scenario: Return on invalid input successful

Alternative scenario of UC1 is tested where the gamer enters an invalid input and gets redirected back to step 2.

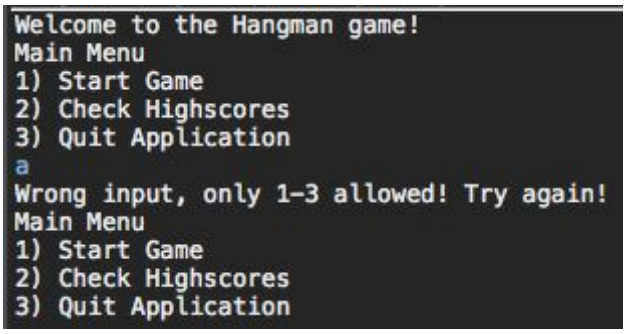
Precondition: None

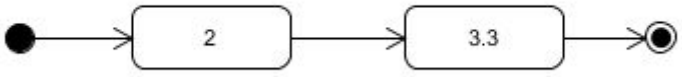
Test steps

- Start the application
- The system presents the main menu with the option to start game, check high scores, and quit the application.
- Enter the “a” key and press enter.
- The system prints an error message and redirects the gamer back to step 2.

Expected

- The system should print an error message and redirect the gamer back to step 2.





TC2.1 Game launch successful

Use case: UC2 Launch Game

Scenario: Game launch successful

The main scenario of UC2 is tested where a game is launched.

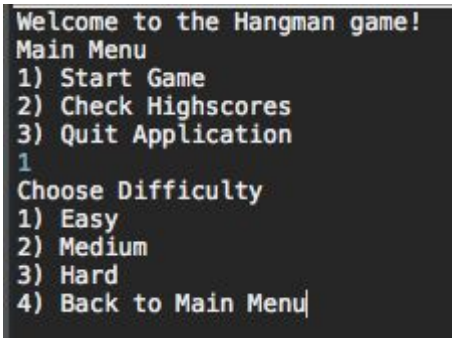
Precondition: The gamer chose to start a game in UC1.

Test steps

- None.

Expected

- The system should launch a game by displaying the difficulty menu (see Use Case 5).



TC3.1 Application termination confirmed successful

Use case: UC3 Quit Application

Scenario: Application termination confirmed successful

Main scenario of UC3 is tested where the gamer chooses to confirm the termination of the application.

Precondition: UC1 / UC6 was successful.

Test steps

- The system prompts the gamer for confirmation regarding the termination.
- Enter the key “1” and press enter.
- The application terminates.

Expected

- The system should terminate successfully.

TC3.2 Return to previous state successful

Use case: UC3 Quit Application

Scenario: Return to previous state successful

Alternative scenario of UC3 is tested where the gamer chooses to not confirm the termination of the application.

Precondition: The application is running & the gamer chose to quit the application.

Test steps

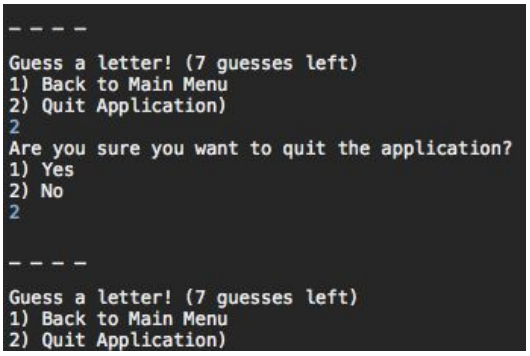
- The system prompts the user for confirmation regarding the termination.
- Enter the key “2” and press enter.
- The system returns to its previous state (Main Menu (UC1) / current game (UC6)).

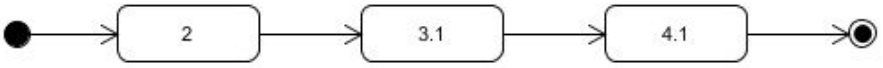
Expected

- The system should return to its previous state successfully.

Previous state = Main Menu (UC1).

Previous state = current game (UC6).




2
3.1
4.1

TC3.3 Return on invalid input successful

Use case: UC3 Quit Application

Scenario: Return on invalid input successful

Alternative scenario of UC3 is tested where the gamer enters an invalid input and gets redirected back to step 2.

Precondition: The application is running & the gamer chose to quit the application.

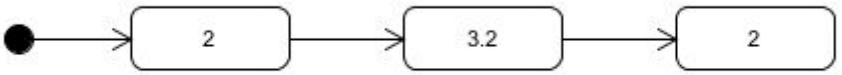
Test steps

- The system prompts the user for confirmation regarding the termination.
- Enter the key “a” and press enter.
- The system prints an error message and redirects the gamer back to step 2.

Expected

- The system should print an error message and redirect the gamer back to step 2.

```
Welcome to the Hangman game!
Main Menu
1) Start Game
2) Check Highscores
3) Quit Application
3
Are you sure you want to quit the application?
1) Yes
2) No
a
Wrong input, only 1-2 allowed! Try again!
Are you sure you want to quit the application?
1) Yes
2) No
```



TC4.1 Highscores printed successful

Use case: UC4 Check highscores

Scenario: Highscores printed successful

Main scenario of UC4 is tested where the gamer chooses to check the highscores and return to Main Menu.

Precondition: The gamer chose to check the highscores.

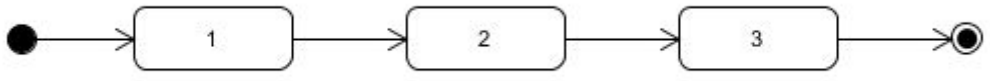
Test steps

- The system prints the highscores and gives the option to return to Main Menu.
- Enter the key “1” and press enter.
- The application returns to the Main Menu. (see Use Case 1, step 2)

Expected

- The system should print the highscores and return to the Main Menu (see Use Case 1, step 2)

```
Player | Score

test3 - 600
Name - 600
test4 - 450
Tanner - 400
test2 - 300
itworks:) - 200
Larry - 200
Hank - 150
test1 - 100
test:) - 100

1) Back to Main Menu
1
Main Menu
1) Start Game
2) Check Highscores
3) Quit Application
```



TC4.2 Return on invalid input successful

Use case: UC4 Check highscores

Scenario: Return on invalid input successful

Alternative scenario of UC4 is tested where the gamer enters an invalid input and gets redirected back to step 1.

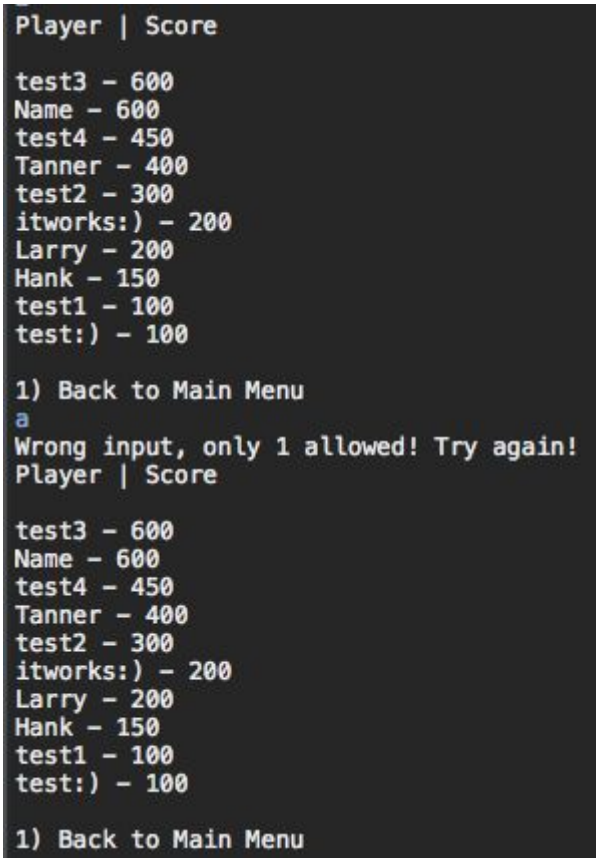
Precondition: The application is running & the gamer chose to quit the application.

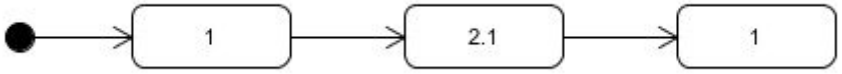
Test steps

- The system prompts the user for confirmation regarding the termination.
- Enter the key “a” and press enter.
- The system prints an error message and redirects the gamer back to step 1.

Expected

- The system should print an error message and redirect the gamer back to step 1.




1
2.1
1

TC5.1 A round of hangman launched on “easy” difficulty successful

Use case: UC5 Choose Difficulty

Scenario: A round of hangman launched on “easy” difficulty successful

Main scenario of UC5 is tested where the gamer chooses to start a round of the hangman game on “easy” difficulty.

Precondition: The gamer successfully launched a game (UC2).

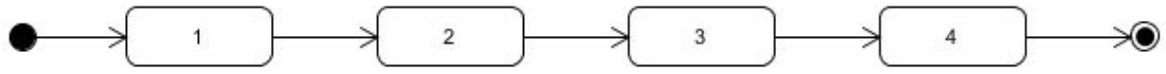
Test steps

- The system presents the difficulty menu
- Enter the key “1” and press enter.
- The system starts a round of the hangman game on “easy” difficulty. (see Use Case 6)

Expected

- The system should start a round of the hangman game on “easy” difficulty. (see Use Case 6)

```
Choose Difficulty
1) Easy
2) Medium
3) Hard
4) Back to Main Menu
1

_ _ _ _

Guess a letter! (13 guesses left)
1) Back to Main Menu
2) Quit Application)
```



TC5.2 A round of hangman launched on “medium” difficulty successful

Use case: UC5 Choose Difficulty

Scenario: A round of hangman launched on “medium” difficulty successful

Alternative scenario of UC5 is tested where the gamer chooses to start a round of the hangman game on “medium” difficulty.

Precondition: The gamer successfully launched a game (UC2).

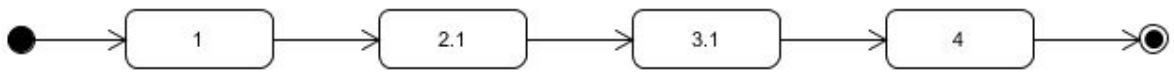
Test steps

- The system presents the difficulty menu
- Enter the key “2” and press enter.
- The system starts a round of the hangman game on “medium” difficulty. (see Use Case 6)

Expected

- The system should start a round of the hangman game on “medium” difficulty. (see Use Case 6)

```
Choose Difficulty
1) Easy
2) Medium
3) Hard
4) Back to Main Menu
2


_ _ _ _

Guess a letter! (10 guesses left)
1) Back to Main Menu
2) Quit Application)
```



TC5.3 A round of hangman launched on “hard” difficulty successful

Use case: UC5 Choose Difficulty

Scenario: A round of hangman launched on “hard” difficulty successful

Alternative scenario of UC5 is tested where the gamer chooses to start a round of the hangman game on “hard” difficulty.

Precondition: The gamer successfully launched a game (UC2).

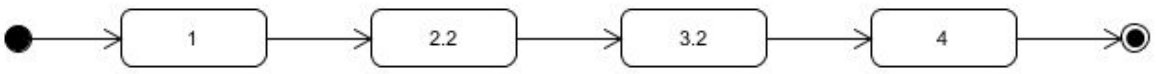
Test steps

- The system presents the difficulty menu
- Enter the key “3” and press enter.
- The system starts a round of the hangman game on “hard” difficulty. (see Use Case 6)

Expected

- The system should start a round of the hangman game on “hard” difficulty. (see Use Case 6)

```
Choose Difficulty
1) Easy
2) Medium
3) Hard
4) Back to Main Menu
3

_ _ _ _

Guess a letter! (7 guesses left)
1) Back to Main Menu
2) Quit Application)
```



TC5.4 Return to Main Menu successful

Use case: UC5 Choose Difficulty

Scenario: Return to Main Menu successful

Alternative scenario of UC5 is tested where the gamer chooses to return to the main menu (UC 1).

Precondition: The gamer successfully launched a game (UC2).

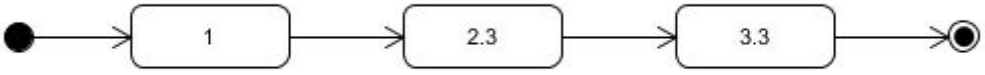
Test steps

- The system presents the difficulty menu
- Enter the key “4” and press enter.
- The system returns to the Main Menu (see Use Case 1).

Expected

- The system should return to the Main Menu (see Use Case 1).

```
Choose Difficulty
1) Easy
2) Medium
3) Hard
4) Back to Main Menu
4
Main Menu
1) Start Game
2) Check Highscores
3) Quit Application
```



TC5.5 Return on invalid input successful

Use case: UC5 Choose Difficulty

Scenario: Return on invalid input successful

Alternative scenario of UC5 is tested where the gamer enters an invalid input and gets redirected back to step 1.

Precondition: The gamer successfully launched a game (UC2).

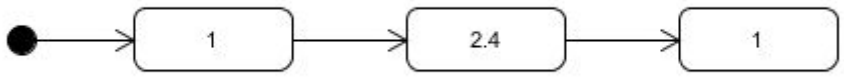
Test steps

- The system displays the option to guess a letter, go back to Main Menu, or Quit the application.
- Enter the key “a” and press enter.
- The system prints an error message and redirects the gamer back to step 1.

Expected

- The system should print an error message and redirect the gamer back to step 1.

```
Choose Difficulty
1) Easy
2) Medium
3) Hard
4) Back to Main Menu
a
Wrong input, only 1-4 allowed! Try again!
Choose Difficulty
1) Easy
2) Medium
3) Hard
4) Back to Main Menu
```



TC6.1 Correct letter guess successful

Use case: UC6 Guess Letter

Scenario: Correct letter guess successful

Main scenario of UC6 is tested where the gamer enters a correct letter and the system displays the position(s) of the guessed letter, within the word.

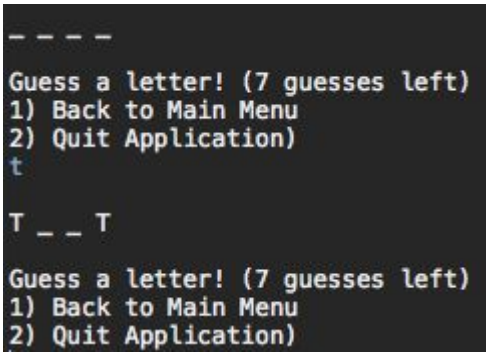
Precondition: The gamer successfully chose a difficulty (UC5).

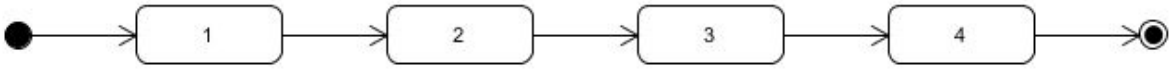
Test steps

- The system displays the option to guess a letter, go back to Main Menu, or Quit the application.
- Enter the key “t” and press enter.
- The system displays the position(s) of the guessed letter, within the word.

Expected

- The system should display the position(s) of the guessed letter, within the word.





TC6.2 Incorrect letter guess successful

Use case: UC6 Guess Letter

Scenario: Incorrect letter guess successful

Alternative scenario of UC6 is tested where the gamer enters an incorrect letter and the system displays the hangman figure with an additional part.

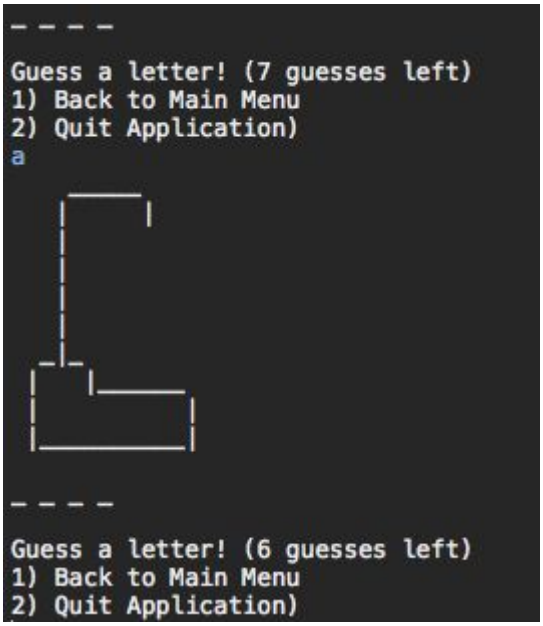
Precondition: The gamer successfully chose a difficulty (UC5).

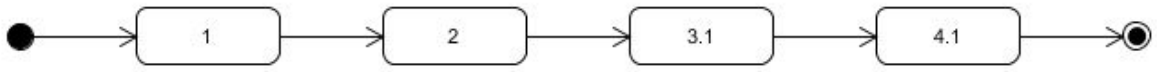
Test steps

- The system displays the option to guess a letter, go back to Main Menu, or Quit the application.
- Enter the key “a” and press enter.
- The system displays the hangman figure with an additional part.

Expected

- The system should display the hangman figure with an additional part.





TC6.3 Return to Main Menu successful

Use case: UC6 Guess Letter

Scenario: Return to Main Menu successful

Alternative scenario of UC6 is tested where the gamer chooses to return to the Main Menu (UC 1).

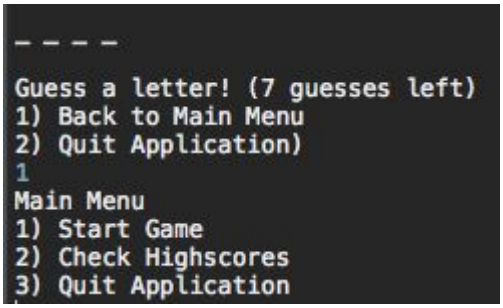
Precondition: The gamer successfully chose a difficulty (UC5).

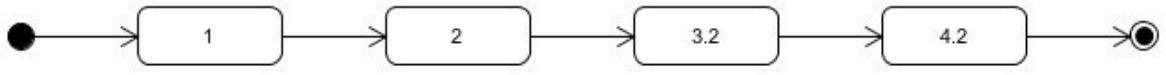
Test steps

- The system displays the option to guess a letter, go back to Main Menu, or Quit the application.
- Enter the key “1” and press enter.
- The system returns the gamer to the Main Menu. (see Use Case 1).

Expected

- The system should return the gamer to the Main Menu.(see Use Case 1).





TC6.4 Quit application successful

Use case: UC6 Guess Letter

Scenario: Quit application successful

Alternative scenario of UC6 is tested where the gamer chooses to quit the application (UC 3).

Precondition: The gamer successfully chose a difficulty (UC5).

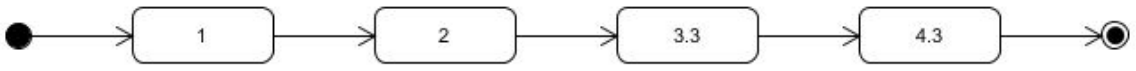
Test steps

- The system displays the option to guess a letter, go back to Main Menu, or Quit the application.
- Enter the key “2” and press enter.
- The system displays the confirm termination menu. (see Use Case 3)

Expected

- The system should display the confirm termination menu. (see Use Case 3)

```
_ _ _ _

Guess a letter! (7 guesses left)
1) Back to Main Menu
2) Quit Application)
2
Are you sure you want to quit the application?
1) Yes
2) No
```



TC6.5 Return on invalid input successful

Use case: UC6 Guess Letter

Scenario: Return on invalid input successful

Alternative scenario of UC6 is tested where the gamer enters an invalid input and gets redirected back to step 2.

Precondition: The gamer successfully chose a difficulty (UC5).

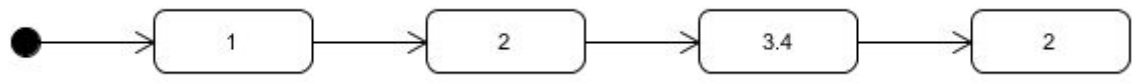
Test steps

- The system displays the option to guess a letter, go back to Main Menu, or Quit the application.
- Enter the key “3” and press enter.
- The system prints an error message and redirects the gamer back to step 2.

Expected

- The system prints an error message and redirects the gamer back to step 2.

```
_ _ _ _

Guess a letter! (7 guesses left)
1) Back to Main Menu
2) Quit Application)
3
Wrong input, only a-z & 1-2 allowed! Try again!

_ _ _ _

Guess a letter! (7 guesses left)
1) Back to Main Menu
2) Quit Application)
```



TC7.1 Save highscore successful

Use case: UC7 Save highscore

Scenario: Save highscore successful

Main scenario of UC7 is tested where the gamer chooses to save the score.

Precondition: The gamer won a game of hangman (UC 6).

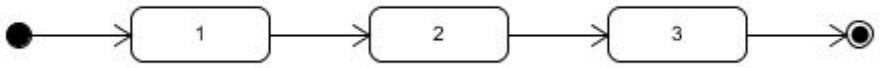
Test steps

- The system displays the option to save the score.
- Enter the key “1” and press enter.
- The system asks the gamer to enter a name (UC 8).

Expected

- The system should prompt the gamer to enter a name & move to use case 8.

```
E N D
You won, congratulations!
Your score was: 600. Do you wish to save your score?
1) Yes
2) No
1
Please enter your name (1-10 characters):
```



TC7.2 Return to main menu successful

Use case: UC7 Save highscore

Scenario: Return to main menu successful

Alternative scenario of UC7 is tested where the gamer chooses to not save the score.

Precondition: The gamer won a game of hangman (UC 6).

Test steps

- The system displays the option to save the score.
- Enter the key “2” and press enter.
- The system returns the gamer to the main menu (UC 1, step 2)

Expected

- The system should display the main menu (use case 1, step 2)

```
T O
You won, congratulations!
Your score was: 350. Do you wish to save your score?
1) Yes
2) No
2
Main Menu
1) Start Game
2) Check Highscores
3) Quit Application
```

TC7.3 Return on invalid input successful

Use case: UC7 Save highscore

Scenario: Return on invalid input successful

Alternative scenario of UC7 is tested where the gamer enters an invalid input.

Precondition: The gamer won a game of hangman (UC 6).

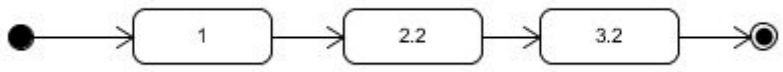
Test steps

- The system displays the option to save the score.
- Enter the key “a” and press enter.
- The system prints an error message and redirects the gamer back to step 1.

Expected

- The system should print an error message and redirect the gamer back to step 1.

```
T E S T
You won, congratulations!
Your score was: 150. Do you wish to save your score?
1) Yes
2) No
a
Wrong input, only 1-2 allowed! Try again!
Your score was: 150. Do you wish to save your score?
1) Yes
2) No
```



TC8.1 Name enter successful

Use case: UC8 Enter gamer name

Scenario: Name enter successful

Main scenario of UC8 is tested where the gamer enters a correct name.

Precondition: The gamer chose to save the score (UC 7)

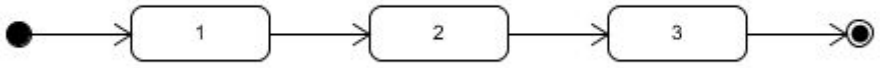
Test steps

- The system prompts the gamer to enter a name between 1-10 characters.
- Write “Name” and press enter.
- The system returns the gamer to the main menu (UC 1, step 2)

Expected

- The system should display the main menu (Use Case 1, step 2)

```
Please enter your name (1-10 characters):
Name
Main Menu
1) Start Game
2) Check Highscores
3) Quit Application
```



TC8.2 Return on incorrect name successful

Use case: UC8 Enter gamer name

Scenario: Return on incorrect name successful

Alternative scenario of UC8 is tested where the gamer enters an incorrect name.

Precondition: The gamer chose to save the score (UC 7)

Test steps

- The system prompts the gamer to enter a name between 1-10 characters.
- Write “Incorrectname” and press enter.
- The system prints an error message & returns the gamer to the step 1 in the main scenario.

Expected

- The system should print an error message & return the gamer to step 1 in the main scenario.

```
Please enter your name (1-10 characters):
Incorrectname
Wrong input, name must be 1-10 characters. Try again!
Please enter your name (1-10 characters):
```

Test Report

Manual tests

Manual test traceability matrix and success:

Test	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8
TC1.1	1/OK	0	0	0	0	0	0	0
TC1.2	1/OK	0	0	0	0	0	0	0
TC1.3	1/OK	0	0	0	0	0	0	0
TC1.4	1/OK	0	0	0	0	0	0	0
TC2.1	0	1/OK	0	0	0	0	0	0
TC3.1	0	0	1/OK	0	0	0	0	0
TC3.2	0	0	1/OK	0	0	0	0	0
TC3.3	0	0	1/OK	0	0	0	0	0
TC4.1	0	0	0	1/OK	0	0	0	0
TC4.2	0	0	0	1/OK	0	0	0	0
TC5.1	0	0	0	0	1/OK	0	0	0
TC5.2	0	0	0	0	1/OK	0	0	0
TC5.3	0	0	0	0	1/OK	0	0	0
TC5.4	0	0	0	0	1/OK	0	0	0
TC5.5	0	0	0	0	1/OK	0	0	0
TC6.1	0	0	0	0	0	1/OK	0	0
TC6.2	0	0	0	0	0	1/OK	0	0
TC6.3	0	0	0	0	0	1/OK	0	0
TC6.4	0	0	0	0	0	1/OK	0	0
TC6.5	0	0	0	0	0	1/OK	0	0

TC7.1	**0**	**0**	**0**	**0**	**0**	**0**	**1/OK**	**0**
TC7.2	**0**	**0**	**0**	**0**	**0**	**0**	**1/OK**	**0**
TC7.3	**0**	**0**	**0**	**0**	**0**	**0**	**1/OK**	**0**
TC8.1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1/OK**
TC8.2	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1/OK**
Coverage & Success	**4/4 OK**	**1/1 OK**	**3/3 OK**	**2/2 OK**	**5/5 OK**	**5/5 OK**	**3/3 OK**	**2/2 OK**

Comments

The random word was set to always be “TEST” during these test cases, to keep consistency.

All test cases were successful!

Automated Unit Tests

ATC 1 - DrawHangmanTest (DrawHangman.java)

```java
class DrawHangmanTest {

private static int test_count = 0;

    DrawHangman test;

    /* Is executed before every test method (not test case).*/
    @BeforeEach
    public void setUp() {
        test_count++;
        System.out.println("Test "+ test_count);
    }


    @Test
    void shouldDrawAllHangmanStagesOnEasy() {

        DrawHangman sut = new DrawHangman();
        for (int i = 0; i < 12; i++) {
        assertEquals(sut.printHangMan("Easy", i), i);
        assertTrue(sut.printHangMan("Easy", i) < i+1);
        }
    }
    @Test
    void shouldDrawAllHangmanStagesOnMedium() {
        DrawHangman sut = new DrawHangman();
        for (int i = 0; i < 9; i++) {
        assertEquals(sut.printHangMan("Medium", i), i);
        assertTrue(sut.printHangMan("Medium", i) < i+1);
        }
    }

    @Test
    void shouldDrawAllHangmanStagesOnHard() {
        DrawHangman sut = new DrawHangman();
        for (int i = 0; i < 6; i++) {
        assertEquals(sut.printHangMan("Hard", i), i);
        assertTrue(sut.printHangMan("Hard", i) < i+1);
        }
    }

}
```

ATC 2 - HangmanGameTest (HangmanGame.java)

```
10  class HangmanGameTest {
11
12  private static int test_count = 0;
13
14      /* Is executed before every test method (not test case).*/
15      @BeforeEach
16      public void setUp() {
17          test_count++;
18          System.out.println("Test "+ test_count);
19      }
20
21      @Test
22      void shouldReturnZeroIfGameHasNotBeenLaunched() {
23          HangmanGame sut = new HangmanGame();
24          assertEquals(0, sut.guessesLeft());
25      }
26
27
28
29  }
```

ATC 3 - HangmanMenusTest (HangmanMenus.java)

```
12  class HangmanMenusTest {
13
14      private static int test_count = 0;
15
16      /* Is executed before every test method (not test case).*/
17      @BeforeEach
18      public void setUp() {
19          test_count++;
20          System.out.println("Test "+ test_count);
21      }
22
23      @Test
24      void test() throws InterruptedException{
25          HangmanMenus sut = new HangmanMenus();
26          System.setIn(new ByteArrayInputStream("1".getBytes()));
27          assertTrue(sut.confirmTermination());
28      }
29  }
```

ATC 4 - HighScoresTest (HighScores.java)

```java
class HighScoresTest {

private static int test_count = 0;

    /* Is executed before every test method (not test case).*/
    @BeforeEach
    public void setUp() {
        test_count++;
        System.out.println("Test "+ test_count);
    }

    @Test
    void shouldReturnZero() {

        HighScores sut = new HighScores();
        assertEquals(sut.getHighScore(), 0);
    }

    @Test
    void shouldNotReturnNullOrNegativeInteger() {

        HighScores sut = new HighScores();
        sut.readFile();
        for (int i = 0; i < 10; i++) {
            assertTrue(sut.getName(i) != null);
            assertTrue(sut.getScore(i) > -1);
        }
    }

    @Test
    void shouldReturn50() {
        HighScores sut = new HighScores();
        sut.calculateScore("Easy", 1);
        assertEquals(sut.getHighScore(), 50);
    }

    @Test
    void shouldReturn75() {
        HighScores sut = new HighScores();
        sut.calculateScore("Medium", 1);
        assertEquals(sut.getHighScore(), 75);
    }
```

```java

    @Test
    void shouldReturn100() {
        HighScores sut = new HighScores();
        sut.calculateScore("Hard", 1);
        assertEquals(sut.getHighScore(), 100);
    }

}
```

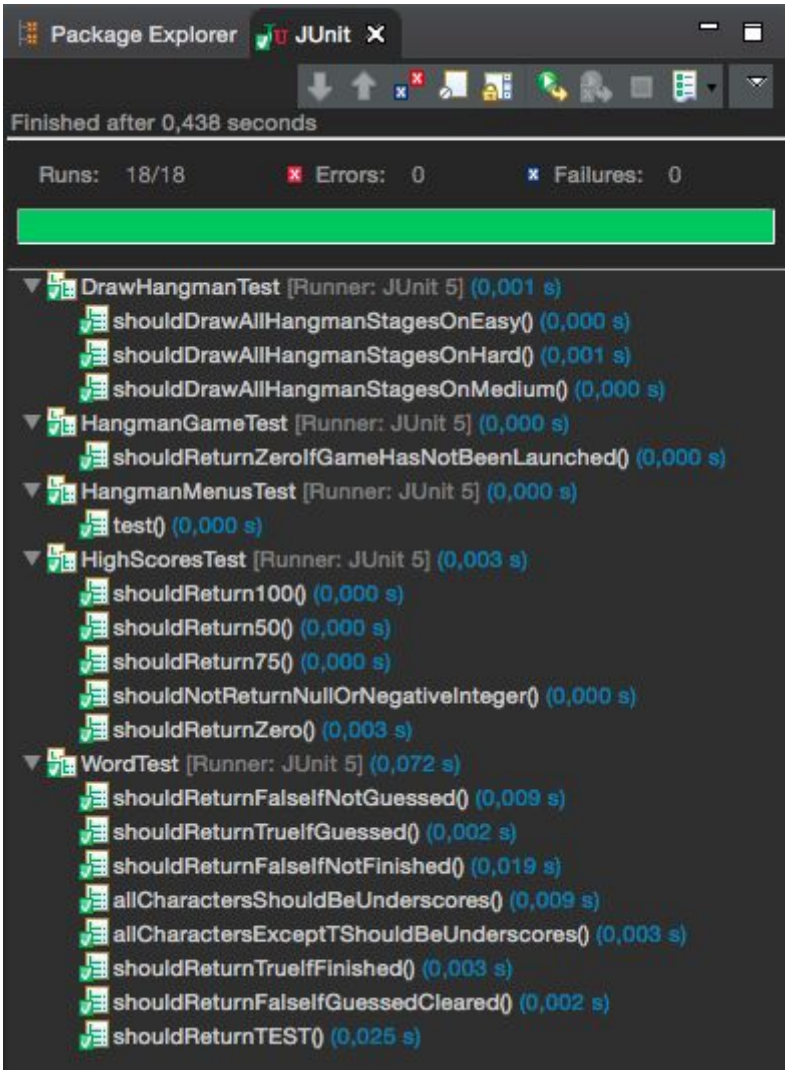
ATC 5 - WordTest (Word.java)

```java
class WordTest {

private static int test_count = 0;

    /* Is executed before every test method (not test case).*/
    @BeforeEach
    public void setUp() {
        test_count++;
        System.out.println("Test "+ test_count);
    }

    @Test
    void shouldReturnFalseIfNotGuessed() {
        Word sut = new Word();
        assertFalse(sut.contains("a"));
    }

    @Test
    void shouldReturnTrueIfGuessed(){
        Word sut = new Word();
        sut.add("a");
        assertTrue(sut.contains("a"));
    }

    @Test
    void shouldReturnFalseIfGuessedCleared(){
        Word sut = new Word();
        sut.add("a");
        sut.clear();
        assertFalse(sut.contains("a"));
    }

    @Test
    void allCharactersShouldBeUnderscores(){ //_ _ _ _
        Word sut = new Word();
        sut.setupWord();
        sut.printUnderscores();
        for (int i = 0; i < sut.getWordLetters().length; i++) {
            String actual = sut.getWordLetters()[i];
            String expected ="_";
            assertEquals(actual, expected);
        }
    }
```

```
53
54      @Test
55      void allCharactersExceptTShouldBeUnderscores(){ //T _ _ T
56          Word sut = new Word();
57          sut.setupWord();
58          sut.updateWordLetters("T");
59              String actual = sut.getWordLetters()[0];
60              String expected = "T";
61              assertEquals(actual, expected);
62              actual = sut.getWordLetters()[1];
63              expected = "_";
64              assertEquals(actual, expected);
65              actual = sut.getWordLetters()[2];
66              expected = "_";
67              assertEquals(actual, expected);
68              actual = sut.getWordLetters()[3];
69              expected = "T";
70              assertEquals(actual, expected);
71      }
72
73      @Test
74      void shouldReturnTrueIfFinished(){ //T E S T
75          Word sut = new Word();
76          sut.setupWord();
77          sut.updateWordLetters("T");
78          sut.updateWordLetters("E");
79          sut.updateWordLetters("S");
80
81          assertTrue(sut.checkIfWordFinished());
82      }
83
84      @Test
85      void shouldReturnFalseIfNotFinished(){ //T E S T
86          Word sut = new Word();
87          sut.setupWord();
88          assertFalse(sut.checkIfWordFinished());
89      }
90
91      @Test
92      void shouldReturnTEST(){ //T E S T
93          Word sut = new Word();
94          sut.setupWord();
95          assertEquals(sut.getWord(), "TEST");
96      }
```

Automated unit test coverage and success

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
jd222qd_dv600	59,0 %	994	690	1 684
src	59,0 %	994	690	1 684
hangman_project	44,7 %	554	685	1 239
HighScores.java	32,9 %	122	249	371
HangmanGame.java	6,9 %	18	243	261
HangmanMenus.java	17,5 %	34	160	194
DrawHangman.java	93,3 %	237	17	254
HangmanMain.java	0,0 %	0	10	10
ReadWords.java	86,0 %	37	6	43
Word.java	100,0 %	106	0	106
Automated_tests	98,9 %	440	5	445
DrawHangmanTest.java	97,3 %	107	3	110
HighScoresTest.java	98,0 %	97	2	99
HangmanGameTest.java	100,0 %	29	0	29
HangmanMenusTest.java	100,0 %	34	0	34
WordTest.java	100,0 %	173	0	173



Comments

I was unable to test some methods due to the way that they were structured. For example void methods which need some of mock to be tested properly, which is what lead to pretty low code coverage. Also, some of the methods in HangmanMenus.java automatically calls other methods when given certain inputs. To test these methods, they either need to be refactored or you need to use some sort of mock tool.

All the tests that were performed, were successful.

Reflection

Writing and performing tests on the code turned out to be more challenging than expected. I had to refactor parts of it to even make it testable in the first place, and some other parts I just wasn't able to refactor with the time I had. The manual test cases were fairly simple to perform because the application worked as expected and with no errors. However, because of the way that it was written, I had trouble performing the automated unit tests. This is something to think about in the future, i.e write code that is easily testable. All in all it was a valuable iteration for me personally, because I learned a lot about how the code should be structured and written for testability.

9 | Time log

Iteration 1

Date	Task	Estimated Time	Actual Time
27-01-2019	Read chapter 2 in *Software Engineering (10th edition)* by Ian Sommerville.	1h	50min
27-01-2019	Read chapter 3 in *Software Engineering (10th edition)* by Ian Sommerville.	1h	50min
23-01-2019	Learn about Software Processes (Lecture 2).	2h	2h
28-01-2019	Read chapter 22 in *Software Engineering (10th edition)* by Ian Sommerville.	1h	50min
28-01-2019	Read chapter 23 in *Software Engineering (10th edition)* by Ian Sommerville.	1h	50min
30-01-2019	Learn about Planning and Managing projects (Lecture 3).	2h	2h
05-02-2019	Write a project summary.	30min	30min
05-02-2019	Write a vision for the project.	1h	45min
05-02-2019	Write a reflection about the vision.	30min	30min
05-02-2019	Write an introduction.	10min	5min
05-02-2019	Write a justification.	15min	10min
05-02-2019	Write about stakeholders.	10min	5min
05-02-2019	Write about resources for the project.	30min	10min
05-02-2019	Write about hard- and software requirements	15min	10min
05-02-2019	Write the project schedule.	30min	30min
05-02-2019	Write about the project scope.	30min	45min
05-02-2019	Write about the project constraints.	30min	20min
05-02-2019	Write about project assumptions.	15min	15min
05-02-2019	Write a reflection about the project plan.	30min	25min
06-02-2019	Write the list of risks for the project.	15min	10min

06-02-2019	Write about strategies concerning the risks.	30min	20min
06-02-2019	Write a reflection about the risk analysis.	30min	20min
06-02-2019	Write skeleton code for the game.	30min	15min
06-02-2019	Write revision history for the iteration.	10min	5min
06-02-2019	Write time log for the iteration.	30min	5min
08-02-2019	**ITERATION 1 DEADLINE**		

Discussion: When I made the estimations for each task I didn't have much previous data to base it upon. Most of the estimations are therefore a little higher than the actual time it took for each task. However, I think this is preferred compared to underestimating the time, because that could lead to potential issues with the planning and not being able to perform tasks on time. There was only 1 task that took longer than I expected it to, and that was writing the scope for the project. The reason for this is because I was unsure of what was actually supposed to be written there.

Iteration 2

Date	Task	Estimated Time	Actual Time
12-02-2019	Read chapter 4 in *Software Engineering (10th edition)* by Ian Sommerville.	1h	50min
12-02-2019	Read chapter 5 in *Software Engineering (10th edition)* by Ian Sommerville.	1h	50min
12-02-2019	Read chapter 20 in *Software Engineering (10th edition)* by Ian Sommerville.	1h	50min
06-02-2019	Learn about Systems and Software Modeling Lecture 4).	2h	2h
13-02-2019	Read chapter 7 in *Software Engineering (10th edition)* by Ian Sommerville.	1h	50min
06-02-2019	Learn about Modeling with UML (Lecture 5).	2h	2h
13-02-2019	Read chapter 6 in *Software Engineering (10th edition)* by Ian Sommerville.	1h	50min
13-02-2019	Learn about Software Architecture (Lecture 6).	2h	2h
13-02-2019	Read chapter 15 in *Software Engineering (10th edition)* by Ian Sommerville.	1h	50min

20-02-2019	Learn about Software Design (Lecture 7).	2h	2h
20-02-2019	Learn about Transformation From Software Design to Implementation (Lecture 8).	2h	2h
18-02-2019	Create Use Case Diagram	30min	40min
19-02-2019	Create Use Case 1 - Start Application	15min	20min
19-02-2019	Create Use Case 2 - Launch Game	15min	20min
19-02-2019	Create Use Case 3 - Quit Application	15min	20min
19-02-2019	Create Use Case 4 - Check Highscores	15min	20min
19-02-2019	Create Use Case 5 - Choose Difficulty	15min	20min
19-02-2019	Create Use Case 6 - Guess Letter	15min	20min
20-02-2019	Create 'Play Game' State Machine.	30min	20min
20-02-2019	Create Extended State Machine.	45min	1h
20-02-2019	Implement Use Case 1 - Start Application	1h	20min
20-02-2019	Implement Use Case 2 - Launch Game	1h	30min
20-02-2019	Implement Use Case 3 - Quit Application	1h	45min
21-02-2019	Implement Use Case 5 - Choose Difficulty	1h	30min
21-02-2019	Implement Use Case 6 - Guess Letter	2h	3h
21-02-2019	Create Class Diagram	30min	20min
21-02-2019	Write revision history for the iteration.	10min	5min
21-02-2019	Write time log for the iteration.	30min	5min
22-02-2019	**ITERATION 2 DEADLINE**		

Discussion: In this iteration I had some previous data to use when estimating the times. However I still like to leave some room for extra time incase any unexpected event occurs.This is why a lot of the estimations are higher than the actual time. There were also some new tasks, unique to this iteration, for example the use cases. I underestimated the time it would take to create the use cases, the use case diagram, the extended state machine, as well as the implementation for Use Case 6. The reason for these differences is because I didn't have any prior knowledge about roughly the time each of these tasks would take. I did my best to estimate the time, but this time it turned out to be too short.

Iteration 3

Date	Task	Estimated Time	Actual Time
01-03-2019	Read chapter 8 in *Software Engineering (10th edition)* by Ian Sommerville.	1h	50min
27-02-2019	Learn about Software Testing (Lecture 9).	2h	2h
27-02-2019	Learn about Testing Techniques (Lecture 10).	2h	2h
27-02-2019	Learn about Testing (Lecture 10.5).	2h	2h
02-03-2019	Implement visual drawing of hangman feature	1h	1,5h
02-03-2019	Write what to test and how.	30min	20min
03-03-2019	Create and perform Manual Test-Case on Use Case 1	30min	45min
03-03-2019	Create and perform Manual Test-Case on Use Case 2	30min	45min
03-03-2019	Create and perform Manual Test-Case on Use Case 3	30min	45min
03-03-2019	Create and perform Manual Test-Case on Use Case 4	30min	45min
03-03-2019	Create and perform Manual Test-Case on Use Case 5	30min	45min
03-03-2019	Create and perform Manual Test-Case on Use Case 6	30min	45min
04-03-2019	Create activity diagrams for all Manual Test-Cases.	2h	1,5h
04-03-2019	Write test report for Manual Test-Cases	45 min	45min
05-03-2019	Create Automatic Unit test for HangmanGame.java	45min	15min
05-03-2019	Create Automatic Unit test for HangmanMenus.java	45min	20min
05-03-2019	Create Automatic Unit test for Word.java	45min	1h
05-03-2019	Create Automatic Unit test for DrawHangman.java	45min	20min
06-03-2019	Create automated unit test coverage and success report.	10min	5min
06-03-2019	Write reflection about testing.	30min	25min
07-03-2019	Write revision history for the iteration.	10min	5min
07-03-2019	Write time log for the iteration.	30min	10min
08-03-2019	**ITERATION 3 DEADLINE**		

Discussion: Same as in iteration 2, I had some previous data to base my estimations upon here, however there is also many completely new tasks concerning testing. For these, I had

no idea how much they would take, so I made rough estimates. Creating and performing the manual test-cases proved to take quite a lot longer than I had first estimated, because of my lack of knowledge within the field.

Iteration 4

Date	Task	Estimated Time	Actual Time
17-03-2019	Implement highscore feature.	2h	2,5h
18-03-2019	Write Use Case 7 - Save highscore	15min	25min
18-03-2019	Write Use Case 8 - Enter Gamer Name	15min	25min
19-03-2019	Create and perform Manual Test-Case on Use Case 7	30min	45min
19-03-2019	Create and perform Manual Test-Case on Use Case 8	30min	45min
19-03-2019	Create Automatic Unit test for HighScores.java	30min	25min
19-03-2019	Create Activity Diagram for Test-Cases on UC 7-8.	20min	15min
20-03-2019	Update Use Case Diagram.	5min	5min
20-03-2019	Update Class Diagram.	5min	5min
20-03-2019	Update Manual Test-Case report	10min	10min
20-03-2019	Update Automated Unit test Code Coverage and Success report.	5min	5min
21-03-2019	Write revision history for the iteration.	10min	5min
21-03-2019	Write time log for the iteration.	30min	5min
22-03-2019	**ITERATION 4 DEADLINE**		

Discussion: This time I had more previous data to base my estimations upon. However, I also took into account the knowledge I gathered about each task from previous iterations. I thought some of the tasks would go quicker now that I had some previous experience, but writing use cases and creating + performing manual test-cases still took more time that I had estimated. The rest is fairly accurate.

10 | Handing in

Files:

HangmanMain.java → Used to launch the application
HangmanGame.java → Contains code needed to play the game.
ReadWords.java → Used retrieve a word from a file.
HangmanMenus.java → Used to navigate menus.
Word.java → Contains information about the randomly chosen word
HighScores.java → Will contain highscore related things.
DrawHangman.java → Draws the hangman figure
HangmanProject.pdf → Contains the complete documentation about the hangman project.

Github repository link:
https://github.com/jd222qd/jd222qd_dv600