

# Software Development Project

**SINISA GVOJIC**

Linnaeus University, Sweden

7/2/2019

## Table of Contents

1 Revision History .....	5
2 General Information.....	6
3 Vision.....	7
3.1 Reflection .....	7
4 Project Plan .....	8
4.1 Introduction .....	8
4.2 Justification .....	8
4.3 Stakeholders .....	8
4.4 Resources .....	8
4.5 Hard- and Software Requirements .....	9
4.6 Overall Project Schedule .....	9
4.7 Scope, Constraints and Assumptions.....	9
4.8 Reflection .....	9
5 Iterations .....	10
5.1 Iteration 1 .....	10
5.2 Iteration 2 .....	10
5.3 Iteration 3 .....	11
5.4 Iteration 4.....	11
6 Risk Analysis .....	13
6.1 List of Risks.....	13
6.2 Strategies .....	13
6.3 Reflection.....	14
7 Use Cases .....	15
UC 1 Start Game.....	15
UC 2 Play Game .....	16
UC 3 Quit Game .....	17
UC 4 User registration .....	17
UC 5 Display High Scores .....	17
UC 6 Inputting new words.....	18
UC 7 Log In .....	18
UC 8 Play Timed Game.....	19
8 Test log.....	22

What to test .....	22
Manual Test Cases .....	23
TC 1.1 Start game .....	23
TC 1.2 Empty input.....	23
TC 1.3 Invalid input.....	24
TC 2.1 Play game with no input errors .....	24
TC 2.2 Play game, with an invalid language choice .....	25
TC 2.3 Play game, with an empty letter input while playing the game .....	26
TC 2.4 Play game with a multiple character sequence as a guess .....	27
TC 2.5 Play game with a non-letter character as an input .....	27
TC 3.1 Quit game.....	28
TC 3.2 Quit game.....	29
TC 3.2 Quit game, wrong input .....	29
TC 4.1 User registration.....	30
TC 4.2 User registration.....	31
TC 4.3 User registration.....	31
TC 5.1 Display high scores .....	32
TC 6.1 Inputting new words, English set.....	32
TC 6.2 Inputting new words, Swedish set .....	33
TC 6.3 Inputting new words, canceling the adding .....	34
TC 7.1 Log in.....	35
TC 7.2 Log in.....	35
TC 7.3 Log in, password mismatch .....	36
TC 7.4 Log in, empty email .....	37
TC 7.5 Log in, email mismatch .....	37
TC 8.1 Play timed game.....	38
TC 8.2 Play timed .....	39
game, with an invalid language choice .....	39
TC 8.3 Play timed game, with an empty letter input while playing the game .....	39
TC 8.4 Play timed game with a multiple character sequence as a guess .....	40
TC 8.5 Play timed game with a non-letter character as an input .....	41
TC 8.6 Play timed game, letting the timer run out.....	42
Unit tests .....	43

Unit test method printLettersAndLines() in the Game class.....	43
Unit test methods getEnglish() and getSwedish() in the Game class.....	43
Unit test method logIn() in the Player class.....	44
Unit test method registerUser() in the Player class.....	44
Unit test method checkEmail() in the Player class .....	44
Failing method .....	45
Reflection.....	45
Time Log.....	46

## 1 Revision History

<b>Date</b>	<b>Version</b>	<b>Description</b>	<b>Author</b>
7/2/2019	0.25	First project iteration and first version	Sinisa Gvojic
17/2/2019	0.50	Second iteration with added features and patches	Sinisa Gvojic
17/3/2019	0.75	Almost finished product with polished features and most of the testing complete	Sinisa Gvojic
21/3/2019	1.0.1	Finished product with the complete documentation	Sinisa Gvojic

## 2 General Information

Project Summary	
Project Name	Project ID
Hangman	Sg222xg
Project Manager	Main Client
Sinisa Gvojic	People interested in learning either the Swedish or English language
Key Stakeholders	
Project Leader, Project Manager, Head Developer, Head Tester, End user	
Executive Summary	
<p>The project is made for educational purposes for people who wish to learn proper spelling in either English or Swedish. The project will include several features other than the base game. It will be created through four iterations, with the final one being delivered on the 22/3/2019.</p>	

### 3 Vision

To design and develop a system based on the popular Hangman game. The system, from now on referred to as “the game”, will be text-based rather than creating a GUI with a graphical representation of the man being hanged. This is done due to time, knowledge and resource constraints under which the game is developed. The game will be developed using the Eclipse IDE, as the developer is most familiar with this environment. The user, henceforth referred to as “the player” will be presented with lines corresponding to the number of letters a word contains. The task of the player is to guess the letters within a certain limit of attempts in order to win and then will be given a score. ~~The score system will function as such: the player starts off with 1000 points and with each correct guess will be awarded with another 500, however if the player does not guess correctly 300 points will be subtracted instead.~~ The scores will be stored in a text file along with the name of the player. The game will have two languages available, Swedish and English. The player can select a language and then the word to be guessed will be in the selected language. ~~The game will implement a multi-player option where the players will take turn to guess the randomly selected word.~~ Additional features will include optional user registration, a survival with a time limit as well as a feature to add more words to the word pool.

#### 3.1 Reflection

The vision for this project was created to display an overarching depiction of what the development of the project will look like, as well as what features and requirements will be necessary for the project to run as intended. It is written with the general, not-detailed idea of what features will be included in the game, as well as the developer’s expectations of what the final project will look like.

## 4 Project Plan

There will be 4 iterations of the project. The first iteration, with the due date 8/2/2019 will include the basic functions of the program. The basic functions include generating random words in the Swedish and English language, giving the user a certain amount of guesses and displaying a message whether or not the user won the round of the game or not. The second iteration, due to the 22/2/2019 will expand on the first by including additional features such as drawing the man to be hanged from different keyboard characters and counting the score and storing it along with the name the user input before starting the round. The third iteration will expand on the user and score side of the system by implementing user registration with an email and a single-player high score system, as well as test both manually and automatically the implemented features by the 8/3/2019. The final project is to be delivered by the 22/3/2019 at 23:55.

### 4.1 Introduction

The Hangman game is a Java program that allows the user to play a game of hangman in a text-based fashion.

### 4.2 Justification

The game is being developed in order to help people learn the correct spelling of words in either Swedish or English, since non-native speakers of the languages are the target audience

### 4.3 Stakeholders

Project manager – the person in charge making a schedule of the planned courses of action in regards to developing the project as well splitting resources to help develop the project faster and in higher quality

Head developer – the person in charge of making sure that the project requirements are fulfilled in a timely matter as well as communicating with the project manager and negotiating the proper course of action in case of an unpredictable circumstance (see Risks). Additionally, the head developer is in charge of making the code clear and well structured for easier maintainability

Head Tester – the person in charge of seeing through that the project has been thoroughly tested, both manually and via automated tests. The head tester communicates with the head developer to sort out any bugs or flaws in the source code.

### 4.4 Resources

The development and testing of this game will be done in the Eclipse Java IDE, using Java 1.8.201, installed on an Acer laptop with an i7-7700 processor, 16GB



of RAM and 256GB of SSD memory. Additional resources for development will be gathered utilizing developer knowledge and the available material.

Note: The user does not need a GPU to play this game

#### 4.5 Hard- and Software Requirements

As stated in the resources, the game will be developed in a Java IDE, and it being a text-based game played in a terminal, it does not require significant computing power regarding graphics. The user has to have java 1.8 installed on his/her device in order for the game not to run into any problems while running

#### 4.6 Overall Project Schedule

The first iteration and the project skeleton is due to be delivered by the 8/2/2019, and the expansion with added additional core features(main menu and the option to quit the game without just closing the terminal) is to be developed, tested and delivered by the 22/2/2019 along with the use case documentation and the use case, class and state machine diagrams. The next iteration, due to the 8/3/2019 will implement more features such as the option to register users log in and track high scores. The project iteration will be delivered, alongside testing documentation showing both manual test cases and automated tests and their respective result.

#### 4.7 Scope, Constraints and Assumptions

Under the scope of this project falls the development and deployment of the game. The game will not be released for sale/download on any online/digital game store nor will it be sold as physical copies. The features of the game will include selecting the language for the word to be guessed (English or Swedish), a high score count, user registration via email, multiplayer, survival mode as well as the option to add new words.

The main constraint for the development of the game is time, as there are very clear and strict deadlines in order. That being said, with the lack of time comes a lack of top quality. The game will be text based with no graphics added through another program, such as Unity™. Seeing as the game is to be developed by a single student, there are no budget constraints for the game development. This is due to a lack of employees and other personnel related expenses.

The user/player of the game is assumed to have a functional computer that is able to run java programs with a mouse and keyboard connected to the device.

#### 4.8 Reflection

Writing the project plan took a large portion of the time, strictly due to checking and re checking the deadlines listed on the MyMoodle page and making sure everything was detailed. This, however is done for the sake of precision which was lacking the first time I wrote this document.

## 5 Iterations

### 5.1 Iteration 1

Iteration 1 features the basic functions of the game. The basic functions include giving the user a set number of attempts to guess all the letters. The words will be randomly selected from the appropriate file in either Swedish or English. The player will then guess a letter one by one until he/she either guesses all the letters or the number of guesses reaches a certain threshold. This bare skeleton of the program is to be delivered by the 8<sup>th</sup> of February this year.

Tasks:

- Read chapters 2, 3, 22, 23 in the coursebook – getting familiar with the process of developing software. Estimated time for each chapter: 150 minutes
- Write vision – coming up with the software and how it is supposed to roughly function. Estimated time: 60 minutes
- Write project plan – a detailed plan for the project, excluding the writing of the source code. Estimated time: 120 minutes
- Write the first iteration plan – a fine grained plan about the tasks needed to be completed before submitting the first iteration of the project. Estimated time: 90 minutes
- Write a list of risks – potential factors that can negatively impact the development of the project. Estimated time: 45 minutes
- Write a time log. Estimated time: 30 minutes
- Write the method for the English word set in hangman. Estimated time: 30 minutes
- Write the method for the Swedish word set in hangman – same method as English, just the file name is different. Estimated time: 15 minutes
- Write the basic game source code. Estimated time: 50 minutes

### 5.2 Iteration 2

The second iteration of the project will expand upon the game skeleton with new features. The game will implement a menu that will enable repeatability as well as an option to quit the game in between sessions.

Tasks:

- Write Iteration 2 section in the documentation. Estimated time: 60 minutes
- Write UC 1. Estimated time: 30 minutes
- Write UC 2. Estimated time: 45 minutes
- Write UC 3. Estimated time: 15 minutes
- Write UC 4. Estimated time: 15 minutes
- Write UC 5. Estimated time: 15 minutes
- Write UC 6. Estimated time: 30 minutes

- Write UC 7. Estimated time: 30 minutes
- Read chapters 6, 7, 15. Estimated time per chapter: 90 minutes
- Watch lectures 6-8 at double speed. Estimated time per lecture 45 mins
- Write the play game class with its methods. Estimated time: 5 hours
- Test the new code. Estimated time: 2 hours
- Drawing the Use case diagram. Estimated time: 60 minutes
- Drawing the Play Game state chart. Estimated time: 75 minutes
- Drawing the Hangman State Chart. Estimated time: 60 minutes
- Drawing the Class Diagram. Estimated time: 30 minutes

### 5.3 Iteration 3

Iteration 3 is mainly focused on testing. However, the features for which a use case diagram was drawn in the previous iteration are also implemented into the code. Testing of the previous and new features is to be done via manual and automated tests.

Tasks:

- Write the Iteration 3 section in the documentation. Estimated time: 30 minutes
- Write the test log for the tests. Estimated time: 35 minutes
- Write manual TC 1. Estimated time: 90 minutes
- Write manual TC 2. Estimated time: 120 minutes
- Write manual TC 3. Estimated time: 90 minutes
- Write manual TC 4. Estimated time: 150 minutes
- Write manual TC 5. Estimated time: 60 minutes
- Write manual TC 6. Estimated time: 90 minutes
- Write manual TC 7. Estimated time: 120 minutes
- Write the unit test for the Game class methods. Estimated time: 210 minutes
- Write the unit test for the Player class methods. Estimated time: 180 minutes
- Write the unit test for the NewWord class methods. Estimated time: 120 minutes
- Read chapter 8 in the book. Estimated time: 45 minutes

### 5.4 Iteration 4

Iteration 4 will add a new timed game mode, update the use case diagram, and the class diagram. Other than this, it will polish the already implemented features of the game, as this is the final version of the project.

Tasks:

- Write a time log. Estimated time: 30 minutes
- Write the Iteration 4 section in the documentation. Estimated time: 40 minutes
- Implement the new timed feature. Estimated time: 120 minutes
- Write a use case for the new feature. Estimated time: 30 minutes
- Write the manual test case for the feature. Estimated time: 30 minutes
- Write an automated test case for the feature. Estimated time: 60 minutes
- Draw a new use case diagram. Estimated time: 45 minutes
- Draw a new PlayGame state machine diagram. Estimated time: 60 minutes
- Draw a new Hangman state machine diagram. Estimated time: 90 minutes
- Draw a new Class diagram. Estimated time: 45 minutes

## 6 Risk Analysis

### 6.1 List of Risks

Risk	Probability	Effects
The developer is ill during the key times in the development cycle	Low moderate	Tolerable
A critical flaw in the code is discovered with no possible way of handling it without affecting the overall code structure	Extremely low	Catastrophic
The device on which the project is developed is irreparably damaged in some way	Extremely low	Serious
The developer gets in a serious traffic accident	Low	Catastrophic
Environment used to develop the project does not generate code properly	Low	Insignificant
The developer needs to urgently return to his home country	Extremely low	Catastrophic
The workload is underestimated	Moderate	Serious

### 6.2 Strategies

The developer is ill during the key times in the development cycle – appropriate medication will be provided to mitigate the symptoms of the illness, ensuring the timely delivery of the project

A critical flaw in the code is discovered with no possible way of handling it without affecting the overall code structure – structuring the code in such a way that no hard-coded component causes a fatal flaw

The device on which the project is developed is irreparably damaged in some way – regular antivirus scans are to be done and the device is to be kept in a safe container to avoid hardware damage

The developer gets in a serious traffic accident – reflective bands are provided to the developer to improve visibility. The developer is to avoid travelling by foot on busy roads during rush hours

Environment used to develop the project does not generate code properly – regular updates are to be installed to minimize the chances of code not being generated

The developer needs to urgently return to his home country – in case of this occurring the code will be uploaded to github for the possible continuation of project development

The workload is underestimated – a detailed analysis of the tasks is to be done and a work plan is to be assembled

### 6.3 Reflection

The list of risks was assembled by thinking about every-day-life that could actually have an impact on the project development. After the those, I moved on to thinking about the technical risks. As a one person project, risks such as specialists in different fields not being available and unable being to recruit qualified staff are not applicable, and many risks that usually involve multiple people are non existant.

## 7 Use Cases

### UC 1 Start Game

Precondition: none.

Postcondition: the game menu is shown.

#### **Main scenario**

1. Starts when the user wants to begin a session of the hangman game.
2. The system presents the main menu with a title, the option to play and quit the game, an option to register a new user, to log in, view high scores and add new words to the word pool.
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 game.

1. The system quits the game (see Use Case 3)

3.2 The Gamer makes a choice to register himself

1. The system registers the user (see Use Case 4)

3.3 The Gamer makes a choice to view the high score

1. The system displays the high score list (see Use Case 5)

3.4 The Gamer chooses to add another word to the word pool

1. The new word is added (see Use Case 6)

3.5 Invalid menu choice

1. The system presents an error message.
2. Go to 2

## UC 2 Play Game

Precondition: none

Postcondition: The player score is shown along with the option to play again

### Main scenario

1. The player wants to play a game of hangman
2. The system prompts the user to select a language
3. The player selects the language of the word that he/she will have to guess, English or Swedish
4. The system selects a word from an appropriate predefined list of words in a file and presents a player a number of dashes corresponding with the number of letters in the selected word
5. The player inputs a letter as a single character from the keyboard
6. A new set of dashes is drawn with the guessed letters in their appropriate positions, and the score is updated and increases the guess counter by one
7. If the amount of guesses is not exceeded repeat from step 5.
8. The system presents the user with a score and an option to play again
9. The user selects the option to play again (repeat from step 2).

### Alternative Scenarios

3.1 The user inputs an invalid selection, the system then informs the user and asks for a different input

5.1 The input was a single non letter character

1. The system asks the user to confirm exiting the game, after which the system exits the current session and displays the main game menu (see UC1)

5.2 The input was multiple characters

1. The system takes the first character of the input and checks whether it is a letter or not.

1.1 If it is, the system then checks if it is contained the word.

1.1 If it is not a letter, asks the user to confirm exiting the game

7.1 The amount of guesses is exceeded, displaying the player's score and asking the user if he would want to play again



### UC 3 Quit Game

Precondition: The game is running.

Postcondition: The game is terminated.

#### **Main scenario**

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

#### **Alternative scenarios**

3.1. The user does not confirm

1. The system displays the main menu

### UC 4 User registration

Precondition: none

Postcondition: The user is successfully registered

#### **Main scenario**

1. The user wishes to register himself
2. The system prompts the user to input his username, email and password
3. The user inputs the abovementioned information
4. A message is shown saying the user is successfully registered, and returns the user to the main menu

#### **Alternative scenarios**

2.1 The user input is invalid, the system asks the user to correct his inputs

### UC 5 Display High Scores

Precondition: none

Postcondition: high scores are displayed

#### **Main scenario**

1. The user wishes to view the list of highscores
2. System displays the names of the users and their high scores, and asks the user to input any key to return to the main menu

## UC 6 Inputting new words

Precondition: The previous use case was executed without any errors

Post condition: The user is notified that a new word is added

### Main Scenario

1. The user wants to input a new word
2. The system presents two options for the language choice
3. The user selects a language, Swedish or English
4. The system prompts the user to input a new word
5. The user inputs a new word
6. The system checks if the word already exists in the selected language list, and if it does not, displays that a new word is successfully added and is returned to the main menu

### Alternative scenario

3.1 Invalid user choice, the system asks the user for a different input

6.1 If the word the user input is already in the list, the system notifies the user asks for another word (repeat from step 2)

## UC 7 Log In

Precondition: none

Postcondition: A message saying the user is logged in is displayed with the option to return to the main menu

### Main Scenario

1. The user wants to log in
2. The system asks the user to enter his/her username and password
3. The user inputs the username and password
4. The system displays a message that the user is logged in, and is provided with an option to return to the main menu

### Alternative Scenarios

3.1 The username or password do not match, the user is asked to input his/her log in details again

## UC 8 Play Timed Game

Precondition: none

Postcondition: The score is displayed, as well as an option to play again

### Main scenario

1. The player wants to play a game of hangman
2. The system prompts the user to select a language
3. The player selects the language of the word that he/she will have to guess, English or Swedish
4. The system selects a word from an appropriate predefined list of words in a file and presents a player a number of dashes corresponding with the number of letters in the selected word
5. The player inputs a letter as a single character from the keyboard
6. The system checks if the time is not exceeded and a new set of dashes is drawn with the guessed letters in their appropriate positions, and the score is updated and increases the guess counter by one
7. If the amount of guesses or the time is not exceeded repeat from step 5.
8. The system presents the user with a score and an option to play again
9. The user selects the option to play again (repeat from step 2).

### Alternative Scenarios

3.2 The user inputs an invalid selection, the system then informs the user and asks for a different input

5.1 The input was a single non letter character

1. The system asks the user to confirm exiting the game, after which the system exits the current session and displays the main game menu (see UC1)

5.2 The input was multiple characters

1. The system takes the first character of the input and checks whether it is a letter or not.

1.1 If it is, the system then checks if it is contained the word.

1.2 If it is not a letter, asks the user to confirm exiting the game

6.1 The time limit is exceeded, immediately stopping the game and displaying the option to play again or to quit.

7.1 The amount of guesses is exceeded or the time is up, displaying the player's score and asking the user if he would want to play again

## 7.1 Diagrams

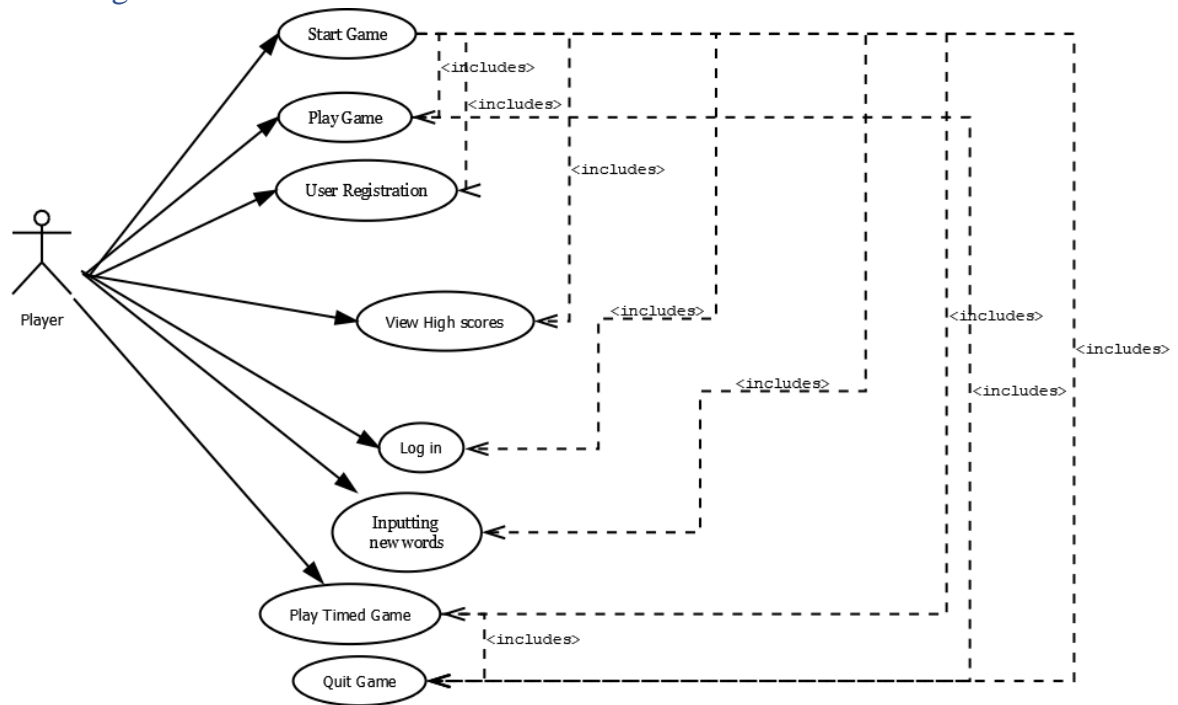


Figure 1: Use case diagram

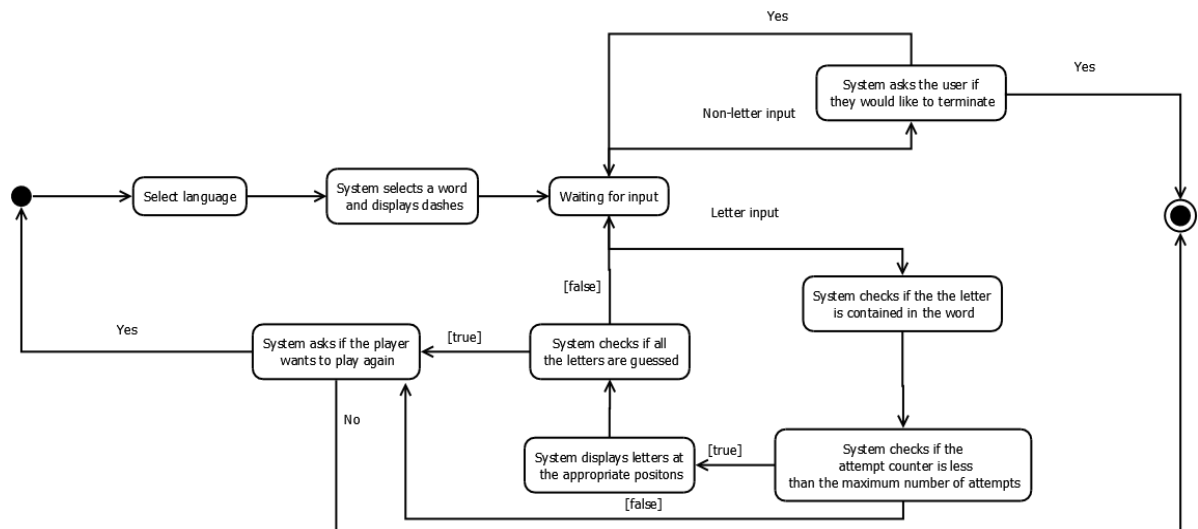


Figure 2: Play game state chart

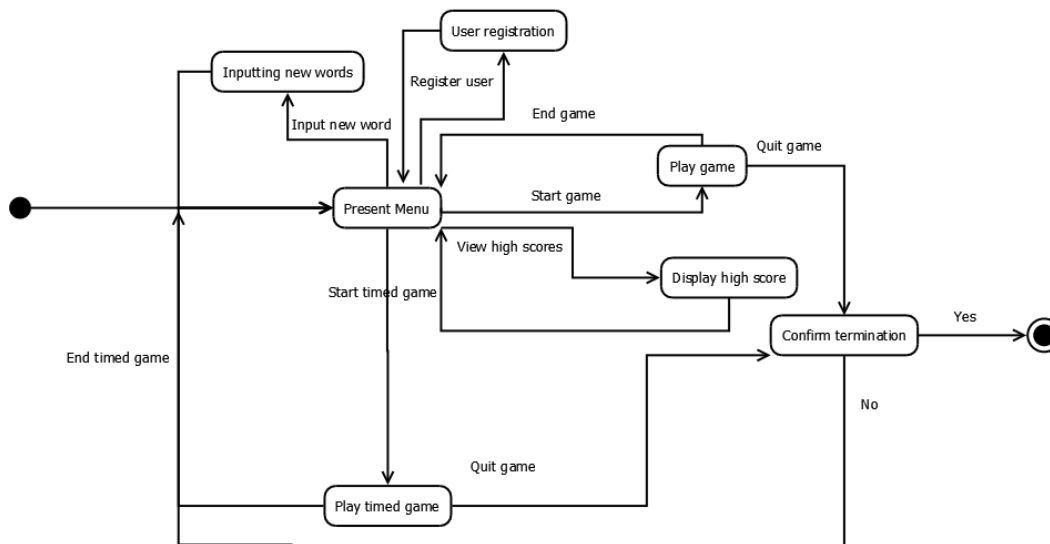


Figure 3: Hangman State Chart

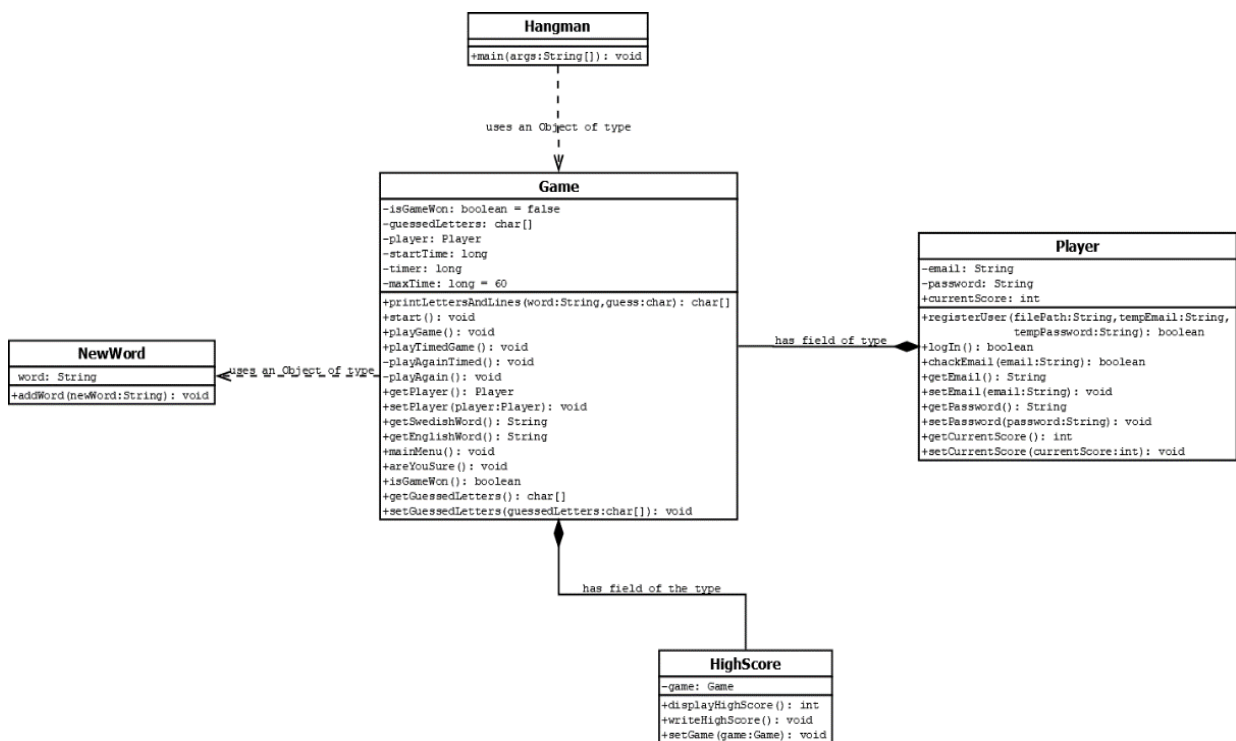


Figure 4: Class diagram

## 8 Test log

### What to test

In the test cycle of the project development, I will attempt to test and validate every method that is vital for the full functionality of the program. To do this, the tests will be divided into test cases that will each test a branch of the project use cases. I intend to first test the methods via dynamic manual testing followed up by statically inspecting the code in order to see how we can implement unit tests to classes and methods.

Test being done	Pass	Fail
TC 1.1	<input checked="" type="checkbox"/>	
TC 1.2	<input checked="" type="checkbox"/>	
TC 1.3	<input checked="" type="checkbox"/>	
TC 2.1	<input checked="" type="checkbox"/>	
TC 2.2	<input checked="" type="checkbox"/>	
TC 2.3	<input checked="" type="checkbox"/>	
TC 2.4	<input checked="" type="checkbox"/>	
TC 2.5	<input checked="" type="checkbox"/>	
TC 3.1	<input checked="" type="checkbox"/>	
TC 3.2	<input checked="" type="checkbox"/>	
TC 4.1	<input checked="" type="checkbox"/>	
TC 4.2	<input checked="" type="checkbox"/>	
TC 4.3	<input checked="" type="checkbox"/>	
TC 5.1	<input checked="" type="checkbox"/>	
TC 6.1	<input checked="" type="checkbox"/>	
TC 6.2	<input checked="" type="checkbox"/>	
TC 6.3	<input checked="" type="checkbox"/>	
TC 7.1	<input checked="" type="checkbox"/>	
TC 7.2	<input checked="" type="checkbox"/>	
TC 7.3	<input checked="" type="checkbox"/>	
TC 7.4	<input checked="" type="checkbox"/>	
TC 7.5	<input checked="" type="checkbox"/>	
TC 8.1	<input checked="" type="checkbox"/>	
TC 8.2	<input checked="" type="checkbox"/>	
TC 8.3	<input checked="" type="checkbox"/>	
TC 8.4	<input checked="" type="checkbox"/>	
TC 8.5	<input checked="" type="checkbox"/>	
TC 8.6	<input checked="" type="checkbox"/>	

## Manual Test Cases

### TC 1.1 Start game

Use case: UC 1 Start game scenario: Starting the game was successful

Description: This specific use case is tested to ensure the proper functionality of the starting menu option for playing the game. If the game does not start when it is supposed to, the rest of the system becomes redundant and unfunctional. Other than this, the off cases need to be handled correctly rather than letting the game crash.

Precondition: The text files SwedishWords.txt and EnglishWords.txt exist in the src folder

Test steps:

- Start the app
- System displays a main menu
- Enter 1 in the console and press enter

Expected result:

- The game had started successfully
- Also output from UC 2

Comment: none

- ☒ Successful
- ☐ Failed

### TC 1.2 Empty input

Use case: UC 1 Start game scenario: Starting the game failed

Precondition: The text files SwedishWords.txt and EnglishWords.txt exist in the src folder

Test steps:

- Start the app
- System displays a main menu
- Do not enter any characters and just press the enter key

Expected result:

- The state of the game does not change, the same menu is still displayed

Comments: none

### TC 1.3 Invalid input

Use case: UC 1 Start game scenario: Starting the game failed

Precondition: The text files SwedishWords.txt and EnglishWords.txt exist in the src folde

Test steps:

- Start the app
- System displays a main menu
- Enter an option not provided in the menu

Expected result:

- A message is displayed informing the user that the input is not valid and displaying the menu again

```
1. Play game
2. Quit game
3. Add new word to the set
4. Register
5. Log in
6. View high score
8
Wrong input, please try again
1. Play game
2. Quit game
3. Add new word to the set
4. Register
5. Log in
6. View high score
```

Comments: none

### TC 2.1 Play game with no input errors

Use case: UC 2 Play Game scenario

Description: This test case verifies that the core feature of the project is working as intended. The inputs tested will be character inputs, multiple character inputs and empty inputs

Precondition: The text files SwedishWords.txt and EnglishWords.txt exist in the src folder

Test steps:

- Start application
- System displays the main menu
- Enter 1 in the console



- The system offers the choice of the language of the word by entering 1 or 2 in the console
- Select a language by pressing either 1 or 2
- The system displays lines corresponding with the number of characters in the word
- Keep entering single letter characters until you reach the maximum number of mistakes or you guess the word

Expected result:

- The game displays a message informing you if you won or lost the game
- The game also displays a menu to either play again or exit the game

```

Guess a letter: d
c e p h a l o p o d
- - - - -
YOU WON!!!
1. Play again
2. Quit game

```

Comments: none

## TC 2.2 Play game, with an invalid language choice

Use case: UC 2 Play Game scenario

Precondition: The text files SwedishWords.txt and EnglishWords.txt exist in the src folder

Test steps:

- Start application
- System displays the main menu
- Input 1 in the console
- The system offers a language choice English or Swedish by inputting 1 or 2 respectively
- Input 7 into the console

Expected result:

- The game informs you of the invalid input, and asks for a different one

```
For english words , press 1. För svenska ord, tryck 2: 7
Wrong input, please try again:
For english words , press 1. För svenska ord, tryck 2:
```

### TC 2.3 Play game, with an empty letter input while playing the game

Use case: UC 2 Play Game scenario

Precondition: The text files SwedishWords.txt and EnglishWords.txt exist in the src folder

Test steps:

- Start application
- System displays the main menu
- Input 1 in the console
- The system offers a language choice English or Swedish by inputting 1 or 2 respectively
- Select 1
- Input nothing into the console and press enter

Expected result:

- The game continues functioning without any interruptions or changes

```
Guess a letter:
a
|
a
- - - - -
Guess a letter:
```

## TC 2.4 Play game with a multiple character sequence as a guess

Use case: UC 2 Play Game scenario

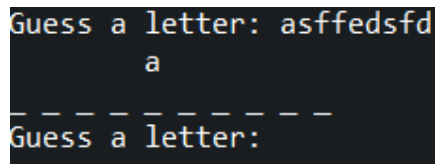
Precondition: The text files SwedishWords.txt and EnglishWords.txt exist in the src folder

Test steps:

- Start application
- System displays the main menu
- Input 1 in the console
- The system offers a language choice English or Swedish by inputting 1 or 2 respectively
- Select 1
- Input asffedsfd into the console and press enter

Expected result:

- The game takes the first character of the input and checks if it is contained within the word, and the system continues to function as intended



```
Guess a letter: asffedsfd
      a
-----
Guess a letter:
```

Comment: The program takes the first character of the input

## TC 2.5 Play game with a non-letter character as an input

Use case: UC 2 Play Game scenario

Precondition: The text files SwedishWords.txt and EnglishWords.txt exist in the src folder

Test steps:

- Start application
- System displays the main menu
- Input 1 in the console
- The system offers a language choice English or Swedish by inputting 1 or 2 respectively
- Select 1
- The system will display the lines responding to the number of characters in the randomly selected word

- Input @ into the console and press enter

Expected result:

- The menu to confirm exiting the game is presented

```
Guess a letter: @  
1. Confirm exit  
2. Go back.
```

Comment: none

### TC 3.1 Quit game

Use case: UC 3 Quit game scenario

Description: Quitting the game via an in game option means that the user will not have to close the window if they want to exit the program, possibly preventing system malfunctions in the future.

Precondition: None

Test steps:

- Start application
- System displays a main menu
- Input 2 into the console
- The system asks if you are sure you want to quit
- Input 1 into the console

Expected result:

- A system message is displayed informing you that the game will soon exit

```
2. Quit game  
3. Add new word to the set  
4. Register  
5. Log in  
6. View high score  
2  
1. Confirm exit  
2. Go back.  
1  
The game will now exit.
```

Comment: none

## TC 3.2 Quit game

Use case: UC 3 Quit game scenario: Returning to the main menu

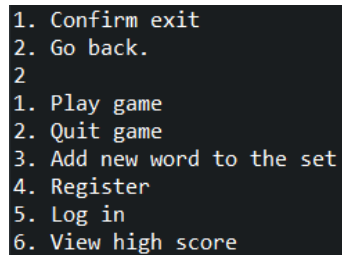
Precondition: None

Test steps:

- Start application
- System displays a main menu
- Input 2 into the console
- The system asks if you are sure you want to quit
- Input 2 into the console

Expected result:

- The main menu is displayed again



```
1. Confirm exit
2. Go back.
2
1. Play game
2. Quit game
3. Add new word to the set
4. Register
5. Log in
6. View high score
```

## TC 3.2 Quit game, wrong input

Use case: UC 3 Quit game scenario

Precondition: None

Test steps:

- Start application
- System displays a main menu
- Input 2 into the console
- The system asks if you are sure you want to quit
- Input t in the console

Expected result:

- A message is displayed saying that that was the wrong input and prompts you to input something else

Comment: none

## TC 4.1 User registration

Use case: UC 4 User registration: A valid email address is input

Description: Testing this functionality is done to satisfy those who are more competitive in the game of hangman and those who attempt to break the highscore. Also, refinement of this functionality in future iterations cannot be done unless the basic version functions as intended

Precondition: The file Players.txt has to exist in the src folder

Test steps:

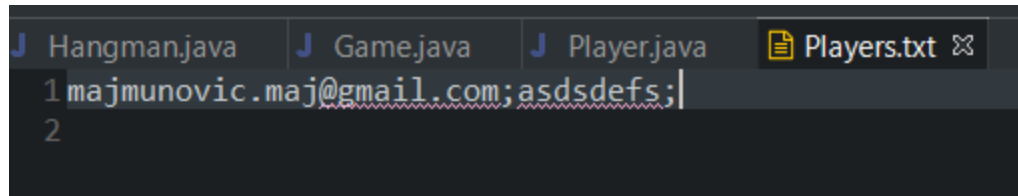
- Start application
- System displays a main menu
- Input 4 into the console
- The systems asks for a valid email address
- Input [majmunovic.maj@gmail.com](mailto:majmunovic.maj@gmail.com) and press enter //a valid address I made specifically for this test
- The system asks for a password
- Input asdsdefs as the password and press enter
- Open the file Players.txt

Expected result:

- The email followed by a semicolon and password should be printed into the file
- The system notifies you that the user was successfully registered and the main menu is displayed

```
Input a valid email adress: majmunovic.maj@gmail.com
Input a password (at least 8 characters long): asdsdefs
User successfully registered!

1. Play game
2. Quit game
3. Add new word to the set
4. Register
5. Log in
6. View high score
```



```
J Hangman.java J Game.java J Player.java Players.txt
1 majmunovic.maj@gmail.com;asdsdefs;
2
```

Comment: none

## TC 4.2 User registration

Use case: UC 4 User registration: An invalid email address is input

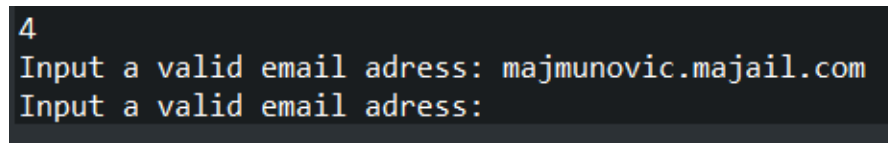
Precondition: The file Players.txt has to exist in the src folder

Test steps:

- Start application
- System displays a main menu
- Input 4 into the console
- The systems asks for a valid email address
- Input [majmunovic.majgail.com](mailto:majmunovic.majgail.com) and press enter

Expected result:

- The system will ask to input the email address again



```
4
Input a valid email address: majmunovic.majail.com
Input a valid email address:
```

Comments: none

## TC 4.3 User registration

Use case: UC 4 User registration: An invalid password is input

Precondition: The file Players.txt has to exist in the src folder

Test steps:

- Start application
- System displays a main menu
- Input 4 into the console
- The systems asks for a valid email address

- Input [majmunovic.maj@gmail.com](mailto:majmunovic.maj@gmail.com) and press enter
- The system asks for a password longer than 8 characters
- Input wqr and press enter

Expected result:

- The system will ask to input the password again

```
Input a valid email adress: majmunovic.maj@gmail.com
Input a password (at least 8 characters long): wqr
Input a password (at least 8 characters long):
```

## TC 5.1 Display high scores

Use case: UC 5 Display high scores scenario

Description: This use cases needs to access a file and display what is written inside it, making it a necessity to do a test.

Precondition: The file Highscore.txt has to exist in the src folder

Test steps:

- Start application
- The system displays the main menu
- Input 6 into the console

Expected results:

- The current high score is displayed along with the main menu

## TC 6.1 Inputting new words, English set

Use case: UC 6 Inputting new words to the English set

Description: This use case writes into a file, and so it requires testing.

Precondition: The files EnglishWords.txt and SwedishWords.txt must exist in the src folder

Test steps:

- Start the application
- Input 3 into the console
- The system will ask you for the word you would like to input



- Input hospital into the console
- The system will ask if you want to add it to the English words or Swedish word set by inputting 1 or 2 respectively
- Input 1 into the console

Expected result:

- The system displays that the new word is added to the word list along with the main menu, and the word hospital is written into the EnglishWords.txt file

```
3
Enter the word:
hospital
1. Add a new english word
2. Add a new swedish word
Any other button: Back to the main menu
1
New word added!
1. Play game
2. Quit game
3. Add new word to the set
4. Register
5. Log in
6. View high score
```

Comments: none

## TC 6.2 Inputting new words, Swedish set

Use case: UC 6 Inputting new words to the Swedish set

Description: This use case writes into a file, and so it requires testing.

Precondition: The files EnglishWords.txt and SwedishWords.txt must exist in the src folder

Test steps:

- Start the application
- Input 3 into the console
- The system will ask you for the word you would like to input
- Input pepparkakor into the console
- The system will ask if you want to add it to the English words or Swedish word set by inputting 1 or 2 respectively
- Input 2 into the console

Expected result:

- The system displays that the new word is added to the word list along with the main menu, and the word hospital is written into the SwedishWords.txt file

```

6. View high score
3
Enter the word:
pepparkakor
1. Add a new english word
2. Add a new swedish word
Any other button: Back to the main menu
2
New word added!
1. Play game
2. Quit game
3. Add new word to the set
4. Register
5. Log in
6. View high score

```

Comments: none

### TC 6.3 Inputting new words, canceling the adding

Use case: UC 6 Inputting new words, canceled

Description: This use case writes into a file, and so it requires testing.

Precondition: The files EnglishWords.txt and SwedishWords.txt must exist in the src folder

Test steps:

- Start the application
- Input 3 into the console
- The system will ask you for the word you would like to input
- Input pepparkakor into the console
- The system will ask if you want to add it to the English words or Swedish word set by inputting 1 or 2 respectively
- Input t into the console

Expected result:

- The system displays the main menu again

```

3
Enter the word:
dasdefe
1. Add a new english word
2. Add a new swedish word
Any other button: Back to the mai
t
1. Play game
2. Quit game
3. Add new word to the set
4. Register
5. Log in
6. View high score

```

Comments: none

## TC 7.1 Log in

Use case: UC 7 Log in

Description: This functionality enables other functionalities such as keeping track of the score and record so it is necessary to test this option. The critical inputs will be the blank inputs

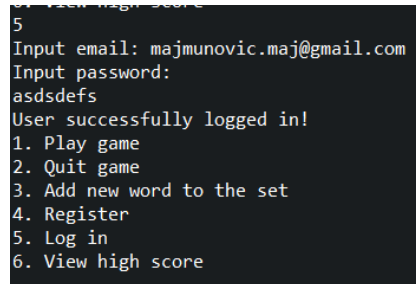
Precondition: The file Players.txt must exist in the src folder

Test steps:

- Start the application
- The system displays a main menu
- Input 5 into the console
- The system will ask for an email
- Input [majmunovic.maj@gmail.com](mailto:majmunovic.maj@gmail.com) into the console
- The system will ask for a password
- Input asdsdefs into the console

Expected result:

- The message that the user is successfully logged in is displayed along with the main menu



```
5
Input email: majmunovic.maj@gmail.com
Input password:
asdsdefs
User successfully logged in!
1. Play game
2. Quit game
3. Add new word to the set
4. Register
5. Log in
6. View high score
```

Comments: none

## TC 7.2 Log in

Use case: UC 7 Log in, empty password

Description: This functionality enables other functionalities such as keeping track of the score and record so it is necessary to test this option. The critical inputs will be the blank inputs

Precondition: The file Players.txt must exist in the src folder

Test steps:

- Start the application
- The system displays a main menu
- Input 5 into the console
- The system will ask for an email
- Input [majmunovic.maj@gmail.com](mailto:majmunovic.maj@gmail.com) into the console
- The system will ask for a password
- Just press enter

Expected result:

- The message saying that either the email or password are incorrect is displayed along with the main menu

```
5
Input email: majmunovic.maj@gmail.com
Input password:

Wrong username or password
1. Play game
2. Quit game
3. Add new word to the set
4. Register
5. Log in
6. View high score
```

Comments: none

### TC 7.3 Log in, password mismatch

Use case: UC 7 Log in, wrong password is input

Description: This functionality enables other functionalities such as keeping track of the score and record so it is necessary to test this option. The critical inputs will be the blank inputs

Precondition: The file Players.txt must exist in the src folder

Test steps:

- Start the application
- The system displays a main menu
- Input 5 into the console
- The system will ask for an email
- Input [majmunovic.maj@gmail.com](mailto:majmunovic.maj@gmail.com) into the console
- The system will ask for a password
- Input wfwetwe and press enter

Expected result:

- The message saying that either the email or password are incorrect is displayed along with the main menu

```
5
Input email: majmunovic.maj@gmail.com
Input password:
wfwetwe
Wrong username or password
1. Play game
2. Quit game
3. Add new word to the set
4. Register
5. Log in
6. View high score
```

Comments: none

#### TC 7.4 Log in, empty email

Use case: UC 7 Log in, empty email

Description: This functionality enables other functionalities such as keeping track of the score and record so it is necessary to test this option. The critical inputs will be the blank inputs

Precondition: The file Players.txt must exist in the src folder

Test steps:

- Start the application
- The system displays a main menu
- Input 5 into the console
- The system will ask for an email
- Just press enter
- The system will ask for a password
- Input asdsdefs

Expected result:

- The message saying that either the email or password are incorrect is displayed along with the main menu

Comments: none

#### TC 7.5 Log in, email mismatch

Use case: UC 7 Log in, nonexistent email

Description: This functionality enables other functionalities such as keeping track of the score and record so it is necessary to test this option. The critical inputs will be the blank inputs

Precondition: The file Players.txt must exist in the src folder

Test steps:

- Start the application
- The system displays a main menu
- Input 5 into the console
- The system will ask for an email
- Input [Maj.maj@gmail.com](mailto:Maj.maj@gmail.com)
- The system will ask for a password
- Input asdsdefs

Expected result:

- The message saying that either the email or password are incorrect is displayed along with the main menu

Comments: none

### TC 8.1 Play timed game

Use case: UC 8 Play Timed Game scenario

Description: This test case verifies that the core feature of the project is working as intended. The inputs tested will be character inputs, multiple character inputs and empty inputs

Precondition: The text files SwedishWords.txt and EnglishWords.txt exist in the src folder

Test steps:

- Start application
- System displays the main menu
- Enter 1 in the console
- The system offers the choice of the language of the word by entering 1 or 2 in the console
- Select a language by pressing either 1 or 2
- The system displays lines corresponding with the number of characters in the word

- Keep entering single letter characters until you reach the maximum number of mistakes or you guess the word

Expected result:

- The game displays a message informing you if you won or lost the game with the score (if player is logged in)
- The game also displays a menu to either play again or exit the game

### TC 8.2 Play timed game, with an invalid language choice

Use case: UC 8 Play Timed Game scenario

Precondition: The text files SwedishWords.txt and EnglishWords.txt exist in the src folder

Test steps:

- Start application
- System displays the main menu
- Input 1 in the console
- The system offers a language choice English or Swedish by inputting 1 or 2 respectively
- Input 7 into the console

Expected result:

- The game informs you of the invalid input, and asks for a different one

```
For english words , press 1. För svenska ord, tryck 2: 7
Wrong input, please try again:
For english words , press 1. För svenska ord, tryck 2:
```

### TC 8.3 Play timed game, with an empty letter input while playing the game

Use case: UC 8 Play Timed Game scenario

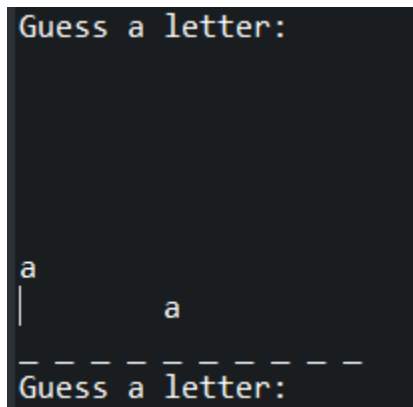
Precondition: The text files SwedishWords.txt and EnglishWords.txt exist in the src folder

Test steps:

- Start application
- System displays the main menu
- Input 1 in the console
- The system offers a language choice English or Swedish by inputting 1 or 2 respectively
- Select 1
- Input nothing into the console and press enter

Expected result:

- The game continues functioning without any interruptions or changes



#### TC 8.4 Play timed game with a multiple character sequence as a guess

Use case: UC 2 Play Game scenario

Precondition: The text files SwedishWords.txt and EnglishWords.txt exist in the src folder

Test steps:

- Start application
- System displays the main menu
- Input 1 in the console
- The system offers a language choice English or Swedish by inputting 1 or 2 respectively
- Select 1
- Input asffedsfd into the console and press enter

Expected result:



- The game takes the first character of the input and checks if it is contained within the word, and the system continues to function as intended

```
Guess a letter: asffedsfd
      a
- - - - -
Guess a letter:
```

Comment: The program takes the first character of the input

### TC 8.5 Play timed game with a non-letter character as an input

Use case: UC 8 Play Timed Game scenario

Precondition: The text files SwedishWords.txt and EnglishWords.txt exist in the src folder

Test steps:

- Start application
- System displays the main menu
- Input 1 in the console
- The system offers a language choice English or Swedish by inputting 1 or 2 respectively
- Select 1
- The system will display the lines responding to the number of characters in the randomly selected word
- Input @ into the console and press enter

Expected result:

- The menu to confirm exiting the game is presented

```
Guess a letter: @
1. Confirm exit
2. Go back.
```

Comment: none

### TC 8.6 Play timed game, letting the timer run out

Use case: UC 8 Play Timed Game scenario

Precondition: The text files SwedishWords.txt and EnglishWords.txt exist in the src folder

Test steps:

- Start application
- System displays the main menu
- Enter 1 in the console
- The system offers the choice of the language of the word by entering 1 or 2 in the console
- Select a language by pressing either 1 or 2
- The system displays lines corresponding with the number of characters in the word
- Wait 60 seconds
- Enter a single letter character

Expected result:

- The game displays the message that the player lost the game with 0 as his score

## Unit tests

### Unit test method printLettersAndLines() in the Game class

```
@Test
void testPrintLettersAndLinesAeroplane() {
    char[] guessedLetters = {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '};
    game.setGuessedLetters(guessedLetters);
    char[] testarray = {'a', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'a', ' ', ' ', ' '};
    game.printLettersAndLines("aeroplane", 'a');
    for(int i = 0; i < guessedLetters.length; i++) {
        assertEquals(testarray[i], guessedLetters[i]);
    }
}

@Test
void testPrintLettersAndLinesSand() {
    char[] guessedLetters = {' ', ' ', ' ', ' ', ' ', ' '};
    game.setGuessedLetters(guessedLetters);
    char[] testarray = {'s', ' ', ' ', ' ', ' ', ' '};
    game.printLettersAndLines("sand", 's');
    for(int i = 0; i < guessedLetters.length; i++) {
        assertEquals(testarray[i], guessedLetters[i]);
    }
}
```

Runs: 2/2    ✖ Errors: 0    • Failures: 0

GameTest [Runner: JUnit 5] (0.015 s)

- ✓ testPrintLettersAndLinesSand() (0.015 s)
- ✓ testPrintLettersAndLinesAeroplane() (0.000 s)

Comments: Comparing the arrays with the equals() method always displays a failure

### Unit test methods getEnglish() and getSwedish() in the Game class

```
@Test
void testGetSwedishWord() throws IOException {
    assertEquals("Yxa", game.getSwedishWord());
}

@Test
void testEnglishWord() throws IOException {
    assertEquals("Hypopotomus", game.getEnglishWord());
}
```

✓ testEnglishWord() (0.015 s)

✓ testGetSwedishWord() (0.000 s)

Comments: I clumped these two methods as one test since they do the same thing, except they access different files. I deleted the words from the files, except one word in each file to eliminate the randomness element

### Unit test method `login()` in the Player class

```
@Test
void testLoginCorrectData() throws FileNotFoundException {
    assertEquals(true, player.login("gvojics@gmail.com", "damaica1.2"));
}

@Test
void testLoginIncorrectData() throws FileNotFoundException {
    assertEquals(false, player.login("asfgsf@asf.com", "asgfhwdfghft"));
}
```

```
testLoginIncorrectData() (0.016 s)
testLoginCorrectData() (0.000 s)
```

Comments: The original version of the method had no arguments making it very difficult to test

### Unit test method `registerUser()` in the Player class

```
@Test
void testRegisterUserCorrectFile() throws IOException {
    assertEquals(player.registerUser("src/Players.txt", "awerad@gmail.com", "agddwefgfrewdef"), true);
}

@Test
void testRegisterUserIncorrectFile() throws IOException {
    assertEquals(player.registerUser("src\\Player.txt", "awwead@gmail.com", "afdsdfssd"), false);
}
```

```
testRegisterUserIncorrectFile() (0.000 s)
testRegisterUserCorrectFile() (0.000 s)
```

Comments: Same as for the `login()` method

### Unit test method `checkEmail()` in the Player class

```
@Test
void testCheckEmailValidAddressFormat() {
    assertEquals(true, player.checkEmail("nngvjc@gmail.com"));
}

@Test
void testCheckEmailInvalidAddressFormat() {
    assertEquals(false, player.checkEmail("asafdwdfewdefed.com"));
}
```

```
✓ testCheckEmailValidAddressFormat() (0.000 s)
✓ testCheckEmailInvalidAddressFormat() (0.000 s)
```

Comments: This method does not check whether or not the email address exists, but rather if the email address is of a valid format

### Failing method

```
@Test
void testDisplayHighScoreActualHighScore() throws FileNotFoundException {
    assertEquals(2700, highscore.displayHighScore());
}
@Test
void testDisplayHighScoreFraudulentHighScore() throws FileNotFoundException {
    assertEquals(5500, highscore.displayHighScore());
}
```

```
HighScoreTest [Runner: JUnit 5] (0.016 s)
✗ testDisplayHighScoreFraudulentHighScore() (0.016 s)
✗ testDisplayHighScoreActualHighScore() (0.000 s)
```

```
public int displayHighScore() throws FileNotFoundException {
    File file = new File("src\\Highscore.txt");
    //File file = new File("src\\Highscore.txt");
    @SuppressWarnings("resource")
    Scanner sc = new Scanner(file);
    ArrayList<Integer> scores = new ArrayList<>();
    while(sc.hasNextLine()) {
        scores.add(Integer.parseInt(sc.nextLine()));
    }
}
```

Comment: This is the failing method, since the file it is supposed to access does not exist

### Reflection

The assignment taught me to plan ahead when I code to make my methods tester-friendly. I have not been able to test almost half of the methods via unit testing simply due to the fact that they cannot be tested unless I change the source code almost entirely. The methods I did test I had to make some quite large alterations to. The largest thing I took time was writing and fixing the code through thorough testing

## Time Log

First iteration/game skeleton:

Action	Estimated time	Actual Time
Reading chapters 2, 3, 22, 23 in the coursebook	150 minutes per chapter	On average 90 minutes per chapter
Writing the vision	60 minutes	73 minutes
Writing the project plan	120 minutes	158 minutes
Writing the first iteration plan	90 minutes	82 minutes
Writing the list of risks	45 minutes	52 minutes
Writing the time log	30 minutes	25 minutes
Write the method for the English word set in hangman	30 minutes	26 minutes
Write the method for the Swedish word set in hangman	15 minutes	4 minutes
Write the basic game source code	45 minutes	48 minutes

Reflections:

- Reading the book took less time than expected since I played it safe and assumed it would take a significantly longer period to read the material
- Writing the project went above the estimated time due to the lack of experience in planning for a project
- Writing the method for the Swedish word set took less time than expected because it's mostly a copy-paste of the English word set method

Second iteration:

Action	Estimated time	Actual time
Programming the Game class	4 hours	4 hours 8 minutes
Writing the Iteration 2 section	60 minutes	123 minutes
Writing use case 1	30 minutes	25 minutes
Writing use case 2	45 minutes	42 minutes
Writing use case 3	15 minutes	17 minutes
Writing use case 4	15 minutes	23 minutes
Writing use case 5	15 minutes	10 minutes

Writing use case 6	30 minutes	27 minutes
Writing use case 7	30 minutes	32 minutes
Reading chapters 6, 7, 15	90 minutes/chapter	88 minutes/chapter
Watching lecture at double speed	50 minutes/lecture	50 minutes/lecture
Test the new code	120 minutes	207 minutes
Drawing Use case diagram	60 minutes	62 minutes
Drawing the Play Game State Chart	75 minutes	83 minutes
Drawing the Hangman State Chart	60 minutes	57 minutes
Drawing the Class Diagrams	30 minutes	24 minutes

Reflection:

- Writing the section in the documentation took much longer because I underestimated the amount of tasks that needed to be done
- Testing the new code took longer because of many bugs that were discovered (words not being selected randomly, but in order, etc)

Third iteration:

Action	Estimated	Actual
Writing Manual TC 1	90 minutes	74 minutes
Writing Manual TC 2	120 minutes	47 minutes
Writing Manual TC 3	90 minutes	15 minutes
Writing Manual TC 4	150 minutes	104 minutes
Writing Manual TC 5	60 minutes	17 minutes
Writing Manual TC 6	90 minutes	98 minutes
Writing Manual TC 7	120 minutes	97 minutes
Unit testing Game methods	210 minutes	125 minutes
Unit Testing Player methods	180 minutes	84 minutes
Unit Testing the NewWord methods	120 minutes	95 minutes
Writing the Iteration 3 section in the documentation	30 minutes	37 minutes
Writing the time log	30 minutes	15 minutes

Reflection:

- The second test case was the most important one to test, due to it being the core of the game, so I predicted it would take a lot longer than it did
- User registration TC4 has proved difficult to test, so I set apart the most time for it, however I overestimated how long it would take
- The log in function was really difficult to thoroughly test because of unexpected results and exceptions that showed up
- Unit testing methods proved to be difficult due to most of them not having a clear input and output

Final iteration:

Action	Estimated time	Actual time
Writing a time log	30 minutes	32 minutes
Writing the Iteration 4 section in the documentation	45 minutes	39 minutes
Implementing the new timed game mode	120 minutes	54 minutes
Writing a use case for the new feature	30 minutes	30 minutes
Writing a manual test case for the feature	30 minutes	13 minutes
Writing an automated test case for the feature	60 minutes	Not possible
Drawing a new use case diagram	45 minutes	45 minutes
Drawing a new PlayGame state machine diagram	60 minutes	45 minutes
Drawing a new Hangman state machine diagram	90 minutes	45 minutes
Drawing a new Class diagram	45 minutes	43 minutes

Reflection:

- The new timed game mode is essentially the same as the regular game, except with a condition that will make the player lose the game after the time is up
- The manual test case for the timed game mode was based off of the play game test case, shortening the time it took to write it
- Drawing new diagrams took significantly less time as I reused the old diagrams, making the placement of the elements easier to organize