SOFTWARE DEVELOPMENT
Hangman Game

Name: Johan Dahlberg
Email: jd222qd@student.lnu.se
Date: 07-02-2019

1 | Revision History

Date	Version	Description	Author
07-02-2019	1	First iteration	Johan Dahlberg
22-02-2019	2	Second iteration	Johan Dahlberg
	3	Third iteration	Johan Dahlberg
	4	Fourth iteration	Johan Dahlberg

2 | General Information

Project Summary	
Project Name	Project ID
Hangman Game	HANGMAN_GAME
Project Manager	Main Client
Johan Dahlberg	University teachers
Key Stakeholders	
Programmer & Tester Product Manager End user	
Executive Summary	
Goal: Create a Hangman Game Language: Java IDE: Eclipse	

3 | Vision

The goal of this project is to implement the game "Hangman" in a text based fashion, using Java as the coding language. The basics of this game is that a random word will be generated from a list of words, which the player then has to guess by suggesting one letter at a time. The words will be imported from a site on the internet, such as an online dictionary.
The amount of guesses (parts used to hang the man) a player has will depend on the difficulty that they choose. There will be 3 difficulties, Easy, Medium & Hard.
Easy - 13 guesses/parts (ground, vertical pole, horizontal pole, rope, head, body, left arm, right arm, right leg, left leg, eyes, nose, mouth)
Medium - 10 guesses/parts (ground, vertical pole, horizontal pole, rope, head, body, left arm, right arm, right leg, left leg)
Hard - 7 guesses/parts (rope, head, body, left arm, right arm, right leg, left leg), the structure (ground, vertical pole, horizontal pole) will already be built on this difficulty.

When starting the program, the player will be greeted by a 'Main Menu', with the options to:

1) Start Game
2) Check High scores
3) Quit Game

Pressing '1' will take the player to a new 'Choose difficulty' menu where they can choose difficulty and start the game, or choose to go back to the main menu:

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

Pressing '2' in the main menu will bring up a list of high scores with the "player name" and their score. The top 10 best scores will be shown.

Pressing '3' will quit the program.

Once a game has been started, a random word will be chosen and will be matched by the same amount of underscores. For example:
Dog → _ _ _

For each incorrect guess the player makes, an image will appear, with the use of Unicode, which represents the hanging of the man. The letter used to guess will also show up on the screen under "letters used" so the player can easily keep track of the guessed letters. If the player runs out of guesses, they lose and will be given the option to 'try again' (same difficulty but different word), or 'go back to main menu'.
If the player manages to guess the word, they have beat the game, and will be given a score. They then have the option to save their score using 'save score' or 'go back to main

menu’. Choosing to save the score will make the player enter a “player name”. If the score is good enough to make it into the top 10, they can view it the main menu by choosing ‘Check Highscores’. If the player chooses not to save their score, it will not be added to the high score list, even if it would’ve been in the top 10.

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 school project which is important because it teaches us how to plan bigger projects in the future.

4.3 Stakeholders

Programmer & Tester - Creates the product and tests it to make sure the code works as expected.

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.

End user - The person who will use the finished product.

4.4 Resources

The resources we have available are:
The time to complete the project.
Computers to run all the software.
Java language to create the code required.
Eclipse to code in, books/internet to find useful information, lectures for additional important information about the project planning process.

4.5 Hard- and Software Requirements

To create the program you need:

Hardware requirements:
Computer

Software requirements:
Eclipse
JRE (java runtime environment)
JDK (java development kit)

To use the program you need:

Hardware requirements:
Computer

Software requirements:
Eclipse
JRE (java runtime environment)

4.6 Overall Project Schedule

Iteration 1 - (08-02-2019)
Iteration 2 - (22-02-2019)
Iteration 3 - (08-03-2019)
Iteration 4 - (TBD)

4.7 Scope, Constraints and Assumptions

Scope: This project will include highscores, different difficulties and graphical unicode representations of images. It will not include multiplayer or a GUI.

Constraints: The allocated time for the project and the functional requirements for the project such as highscores, difficulties. Another constraint might be potential lack of knowledge in a specific field.

Assumptions: The user knows how to use a terminal/console and understand english.

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

In this iteration a plan for the project will be created and the included features will be decided. In addition to this, skeleton code will be written for the project. Chapters 2-3, 22-23 in the Software Engineering book, by Ian Sommerville will also be read.

5.2 Iteration 2

In this iteration the features will be modelled using UML and then added to the project's code. The diagrams will be implemented in the way that they are modelled, and also included in the project documentation. Chapters 4-7, 15, 20 in the Software Engineering book, by Ian Sommerville will also be read.

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

5.3 Iteration 3

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. More to be added later...

5.4 Iteration 4

In this iteration, the project will be completed. For a new set of features, reiterate the steps in iteration 1-3. More to be added later...

6 | Risk Analysis

6.1 List of risks

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 | Time log

Iteration 1

- **Estimations**
- Read chapters 2-3 + 22-23 (3 hours)
- Finish the study material questions (4 hours)
- Finish the project plan document (5 hours)
- Write skeleton code (30 min)

- Date Time Task
- **27-01-2019 (2h)** Read Chapter 2 & 3 in Software Engineering by Ian Sommerville
- **28-01-2019 (2h)** Read Chapter 22 & 23 in Software Engineering by Ian Sommerville
- **29-01-2019 (1h)** Started working on the study material questions
- **30-01-2019 (2h)** Finished working on the study material questions
- **05-02-2019 (2h)** Started working on the project plan
- **06-02-2019 (4h)** Finished working on the project plan
- **(30 min)** Wrote the skeleton code
- **08-02-2019 DEADLINE**

- **Results**
- Read chapters 2-3 + 22-23 (4 hours)
- Finish the study material questions (3 hours)
- Finish the project plan document (6 hours)
- Write skeleton code (30 min)

Iteration 2

- **Estimations**
- Read Chapters 4-7 + 15 + 20 (6 hours)
- Finish the study material questions (4 hours)
- Create use case diagram (2 hours)
- Write fully dressed use case model (3 hours)
- Create a 'play game' state machine (1 hour)
- Create an extended state machine (2 hours)
- Make game playable (basic requirements) (5 hours)
- Create a class diagram (1 hour)

- Date Time Task
- **12-02-2019 (3h)** Read Chapter 4-6 in Software Engineering by Ian Sommerville
- **13-02-2019 (3h)** Read Chapter 7, 15, 20 in Software Engineering by Ian Sommerville
- **14-02-2019 (3h)** Finished working on the study material questions
- **18-02-2019 (4h)** Created a use case diagram

- **19-02-2019 (4h)** Wrote a fully dressed use case model
- **20-02-2019 (2h)** Created a ‘play game’ static machine
- **20-02-2019 (2h)** Created an extended state machine
- **20-02-2019 (2h)** Started working on making the game playable
- **21-02-2019 (4h)** Finished working on making the game playable (includes difficulty feature)
- **21-02-2019 (1h)** Created a class diagram
- **22-02-2019 DEADLINE**

- **Results**
- Read Chapters 4-7 + 15 + 20 (6 hours)
- Finish the study material questions (3 hours)
- Create use case diagram (4 hours)
- Write fully dressed use case model (4 hours)
- Create a ‘play game’ state machine (2 hours)
- Create an extended state machine (2 hours)
- Make game playable (basic requirements + difficulty feature) (6 hours)
- Create a class diagram (1 hour)

8 | Handing in

Files:

HangmanMain.java → Used to launch the application
HangmanLogic.java → Contains everything needed to navigate menus and play the game (excluding the retrieval of a word)
ReadWords.java → Used retrieve a word from a file

Diagrams:

UML State Machine.png → The basic 'Play game' state machine, containing the basic play game use cases' states and transitions.
UML State Machine Extended.png → The extended state machine, containing all states and transitions.
UML Class diagram.png → The class diagram, showing all the classes, methods and variables.
UML Use Case.png → The use case diagram, showing the user-system interactions.

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