HANGMAN GAME DEVELOPMENT PROJECT

ADAM ELFSTRÖM
Programvaruteknik
22nd March 2019

Contents

1 | Revision history

Date	Version	Description	Author
08/02/2019	1.0	Project plan created, basic structure and functionality, such as file organization and user input implemented.	Adam Elfström
22/02/2019	1.1	Updated Iteration 2 plan. Updated Time Log with estimates for Iteration 2. Added Use cases section. Created diagrams. Implemented basic gameplay.	Adam Elfström
8/03/2019	1.2	Updated Iteration 3 plan, mostly with plans for major refactoring. Added use case re-writing to the time log (as Iteration 3). Completely re-wrote or tweaked all use cases.	Adam Elfström
22/03/2019	1.3	Updated Iteration 4 plan. Updated Execute Summary. Updated vision. Updated project plan (Resources, Hard/software reqs, overall schedule).	Adam Elfström

2 | General information

Project summary	
Project Name	Project ID
Hangman Game	ae223nv_1dv600
Project Manager	Main Client
Adam Elfström	Young adults (age 16-20)
Key Stakeholders	
End user, project manager, developer, customer, tester	
Executive Summary	
The Hangman Game project aims to produce an interactive application that can be run in a terminal on any personal computer. This application allows users to play a game of "hangman" with differing rule sets and scoring systems. The game consists of guessing on which letters a hidden word contains, where each correct guess reveals the guessed letter in the word. An incorrect guess adds a part to a picture of a man being hanged. The game ends when the entire word is guessed or after 10 incorrect guesses, when the entire picture has been drawn. The project is being done as an exercise in project planning and management and the produced product is not meant for commercial use.	

3 | Vision

The finished application will be an interactive word-guessing game. Before the game starts, the user is presented with a menu in which they can select different options such as game mode and previous scores. The difficulty of the game will be at it's highest in the beginning, as the user is only presented with blank slots where the letters that make up the word will be placed. As the game progresses and the user guesses letter after letter the previously empty slots are filled in with each correct guess. The user will be able to control the level of difficulty in the sense that the length of the word might be chosen before the game starts, or left to be randomly picked. To further increase difficulty, the user will be able to play the game in a timed mode, where the word has to be guessed before the time runs out. The user will also be able to choose an "infinite" mode, where the objective is to guess as many words correctly as they can. The game will in this mode persist until one word is not able to be guessed, or if the timed mode is selected, until the timer runs out. The timed modes might make the user more stressed as they play, much like the text-based illustration of a man slowly being hanged as they guess a letter incorrectly being shown on the screen might. This illustration will be shown regardless of the mode the user chooses, and it will communicate how many wrong guesses the user has made. When the man is fully drawn, the user has lost the game. If the user were to lose, the word remaining hidden letters will be presented in addition to an option to return to the main menu.

3.1 Reflection

Specifying the vision for the project and its application is in a sense difficult, as it must illustrate the functionality and usecases for the application without being as detailed and precise as the requirements. I found that the process of detailing the vision involved me rewriting sentences often that ended up being either too vague or too specific, bordering on technical instructions or requirements. As a conclusion, it was a good exercise because it meant thinking and describing what you wanted in an approachable way, instead of keeping it very technical and terse.

4 | Project Plan

4.1 Introduction

The application to be made is in the form of an interactive game where the objective is to, letter by letter, guess what a randomly chosen hidden word is. The only information shown to the user is a sequence of empty spaces which illustrate how many letters the word contains. As letters are correctly guessed the empty spaces are filled out, which slowly decreases the difficulty of guessing the word. With each incorrect guess a part of an image is drawn of a stick-figure getting hanged, after 10 incorrect guesses the user has lost and the game ends.

4.2 Justification

This application will be the resulting product from a project that aims to educate a student of the different activities involved in software engineering, such as managing and planning a project and subsequent implementation and testing of said plans. This application is not being created for commercial use and will only be available to a small group of individuals.

4.3 Stakeholders

End user: The person using the finished application. This person will not scrutinize the application from a technical perspective, but rather from a a perspective of enjoyment and usability.

Manager: The person responsible for the project and its completion. This person wishes to see that the project plan and schedule is followed and that the increments and the finished application is delivered to the customer.

Developer: The person who will be implementing all required features. This person will try to write code that follows the requirements of the application's features and will also be concerned with good code structure for better maintainability.

Customer: The customer is in this project a select group of teachers and tutors that have requested that the project starts and the application is produced. These persons will be concerned with quality control and that the application and its development process is according to specifications.

Tester: The person testing the code. This person will only be concerned with whether the application performs as expected or not.

4.4 Resources

Total project budget is 700 SEK. Every tool and resource used during the project - such as a computer to work on, word-processing software for documentation writing, the terminal used to execute the application and the IDE used to write and test the code in - is already available to all involved parties or available for free. The available money will be spent on the book Software Engineering, 10:th ed.

by Ian Sommerville. This book contains most of the knowledge that is needed to complete the project and will be used by the developer, project manager and the tester. Travels to and from the University where all involved parties work and/or study involve walking or biking and the travel budget is therefor 0 SEK.
Personnel is 1 person. Planning, implementing, testing and documenting is all done by the same person.
The duration of the entire project is 8 weeks and it will be alongside a university course in Java programming. This means that a maximum of 20 hours is estimated to be available each week, making a total of 160 hours for the entire project. Each of the four iterations will take two weeks and will involve both the studying of the literature and the actual work. The estimation is that around 40% of the time spent on each iteration (except the last iteration) will be used to study and learn the literature, as well as conducting additional research needed when the literature is insufficient.

4.5 Hard- and Software Requirements

The following hardware requirements are the recommended hardware for the system that wishes to run the application. They do not reflect actual minimum hardware requirements, since the program can be run on virtually any machine with java installed due to the small size of the program. Note: if any operating system other than Windows is used the program will still work, but it will look strange.

4.5.1 Hardware Requirements

Processor: 1 or more cores @ 1 GHz or higher
Storage: 64 GB or more
Memory: 4 Gb or more
Graphics: Intel or AMD integrated graphics
Network: Ethernet connection (LAN) or wireless connection (WiFi)
Display: 800x600 or higher resolution
Input devices: Mouse and keyboard

4.5.2 Software Requirements

Developer and tester

Operating system: Windows 7 or newer
Browser: Google Chrome, Firefox or similar
Java: JDK 11 or newer
IDE: Eclipse IDE v4.10 (2018-12) or newer
To run the program: Command Prompt or PowerShell

End user

Operating system: Windows 7 or newer
Java: JDK 11 or newer
To run the program: Command Prompt or PowerShell

4.6 Overall Project Schedule

The following schedule contains the deadlines for the different deliverables, based on which iteration they belong to.

4.6.1 8th February 2019

- Initial project plan
- Time log
- Skeleton code

4.6.2 22nd February 2019

- Updated project plan (with iteration 2)
- Time log
- Use case diagram
- Use cases
- State machine
- Playable basic version of the game
- Class diagram of the basic game

4.6.3 8th March 2019

- Updated project plan (with iteration 3)
- Time log
- Test plan
- Manual test cases
- Automated unit tests
- Test results
- Manual and automated test coverage

4.6.4 22nd March 2019

- Final project plan
- Time log
- Updated use case diagram
- Updated use cases
- Updated state machine
- Finished game
- Class diagram of the finished game
- Updated test plan
- Manual test cases
- Automated test cases
- Test results
- Manual and automated test coverage

4.7 Scope, Constraints and Assumptions

Scope: The application must contain a working version of the traditional hangman game, meaning that words must be able to be chosen (completely random or random within some parameters), the length must be presented to the user, the letters must be able to be guessed and the game must be able to be either won or lost depending on whether the user guesses correctly or incorrectly. To make the game more interesting and increase the replayability, some different game modes should be available. These game modes should be a standard version (random word), a timed standard version, an infinite version and a choose length version (random word of chosen length). The standard modes only involve one word that needs to be guessed and the infinite mode can contain an infinite sequence of words to be guessed, where the end of the game is determined either by time or a wrong guesses. The scores that are generated from these different modes, such as the time taken when guessing a word or the number of words/characters correctly guessed during the infinite mode, should be saved and accessible to the user from the in-game menu. The words that will be used in the game are to be fetched from a database to avoid having to store them locally. All words must be nouns and must not contain any special characters to be guessed, only normal English letters, to avoid having to implement language/character recognition. This means that words such as "add-on" will only require the user to guess the "add" and "on". The application could have some features allowing for customisation, meaning that certain aspects of the game can be changed by the user. This customisation could be in the form a list of words the user do not want to see when playing, and this list could be expanded every time the user fails to correctly guess a word, if the player chooses.

Out of Scope: The application won't have a feature to allow for online activities such as local multi-user games or any form of online play to reduce the complexity of the program and ensure that it is completed during the limited time-frame of the project. The application must not contain a graphical user interface, as everything will be displayed in a terminal window. This is because the application will focus on features and only implement visual improvements if time allows. The application might not be optimised to run on operating systems other than Windows, as some things related to the printout on the console window is purely operating system dependant. Implementing something that recognises which operating system the program is run on is a feature that might only be considered to be added if time allows. The application will not contain the option to choose words in any other way than word length, since this requires a deeper analysis of the (very large) word list that might result in a total reduction the amount of available words.

Constraints: The main constraints for this project are time, lack of experience and lack of personnel. The project involves one person that is responsible for all parts, such as management, documentation, implementation and testing. This means that effort must be split among the different activities and this results in a lack of time for finesse and detailed work. In addition to this, the time will be further constrained due to a lack of experience for the person that works on the project. This is lack of experience managing and planning a project, and making a working application such as the game to be developed. Time is therefore needed to research questions that arise during the project, time that could otherwise be spent on implementation, testing, etc.

Assumptions: The end user has to have access to a computer to be able to run the application. The user will have some knowledge about how hangman works. Some previous knowledge about how to operate a terminal window exist, since the game will be executed in this environment.

4.8 Reflection

Before I could start writing the project plan, a lot of thinking was required. This was mainly because many of the parts to be written down contained things that, for this particular project, were somewhat difficult to identify. This meant that a significant part of the time spent on the plan was not actually constructing it, but simply thinking about how it should be constructed and what it should contain.

5 | Iterations

5.1 Iteration 1

Firstly, an account on github.com has to be created to allow for easy version control of the application and easier distribution once the project is finished. To start using this account for the intended purposes, git has to be downloaded and installed onto at least the computer where the source code will be written. The local files can then be uploaded to a GitHub repository. The GitHub setup process should take around 30 minutes, since some instructions on process has been provided.

Before any implementation can take place, a document containing all planning must be created and subsequently filled with information about the project. This document should be well structured and organized with easily distinguishable parts. After the document structure is finished, some basic information about the project, such as names and dates, can be added to the title page and the footer of the pages. Creating the document will be done using LaTeX, which is more time-consuming than alternatives like Microsoft Word. This should therefore take around two hours.

The vision of the project can then be added to the document, along with a short reflection about writing this vision. The vision must contain information that makes it clear what type of application will be developed and what it should be able to achieve. It should not take longer than one hour to write the vision.

The general project plan should be created and added to the document. This plan should contain information about what the project is about, why the project exists, who might be interested in the project, what resources are available and required, main deadlines for deliverable increments, and some specification about the features of the finished application. Additionally, the project plan should also include a short reflection about its creation. The project plan should take around two hours to write.

Risks that might affect the project should then be identified, analysed and planned for. This information should be terse and presented in a table for easier reading. The entire process of identification, analysis and planning for the risks should take a total of one hour.

After all of that is finished, the detailed planning for each iteration will commence, where more technical details about what should be done is presented. This planning should be added to before each iteration begins and as more information about each iteration becomes clear. The initial iteration planning will however still contain an overview of what will be done, instead of leaving all the planning for when the iteration starts. It should take around 30 minutes to plan for each of the three first iteration, and around 10 minutes for the last iteration. The last iteration takes less time to plan since it will consist of the parts already mentioned in the first three iterations.

After the planning is completed, implementation can begin. Implementation should start with the creation and structuring of the files that are expected to be part of the finished application. These files should not contain much actual code, as the focus for this iteration is on the documentation. The code that is produced should be more of an outline for what is to be added in future iterations rather than a functioning game, meaning that it is only skeleton code. The class structuring and initial code writing should take around 20 minutes.

Around 20 minutes should also be spent confirming that the application can actually run outside of Eclipse IDE, since the end user will execute it from a console/terminal window.

Lastly, the project plan and the skeleton code will be uploaded to GitHub and a release of the project will be created. This release creates a "frozen" copy of the files so that work can continue. The release should not have to take more than 15 minutes.

5.2 Iteration 2

To start, the plan for Iteration 2 will be updated with new information regarding the activities that the iteration will involve. This should take around 30 minutes to write.

The requirements for the application will then start to be created. These requirements are extensions of the the plan made in iteration one, meaning that they contain more details for features than what was specified earlier. Instead of simply writing a large spreadsheet, the requirements will mostly take the form of use cases. These use cases will present scenarios of intended usage by end users and will be presented in a "Fully dressed" form, meaning that they contain detailed steps on how the user is expected to interact with the system and what decisions the user might make. The first and third use case (Start game & Quit game) have already been created and should be updated to fit this particular application. This should take around 20 minutes. The most important use case, "Play Game", should be the next use case to be written in the fully dressed style. This should take up to 45 minutes. The remaining use cases will then be written in slightly less detail. These use cases should take around one hour to write.

All written use cases will also be presented in a Use Case Diagram. This diagram shows what the system will be able to do for the end user. Instead of containing details about each use case, the diagram will only show the name of the use cases which makes it more high level than the written use cases. The use case diagram should take around 30 minutes to draw.

The behaviour of the application will then be modelled using a state machine, based on the use cases previously designed. The state machine is a graphic involving different states of the application (what the application is doing at a specific time), and the transitions that occur between these states (based mainly on user input). Two versions of the state machine will be created, one for the Play Game use case and one that shows all the states the application can be in (including the state machine for the Play Game use case). The state machine for the the "Play Game" use case should take around 45 minutes to draw and the rest of the states should take around one hour to add.

The modelled behaviour of the application should now be able to be implemented in its most basic form. This means that the game should be playable and the player should be able to navigate different menus. The majority of the effort and time spent implementing is spent on functionality instead of appearance, meaning that things like customisation options and the drawing of the hanged man will not be implemented. The features to be implemented in this iteration are:

- An Exit/quit menu - 15 min
- Game mode selection menu - 20 min
- Word length selection menu - 15 min
- Word fetching - 30 min
- Word selection (random and filtered random)- 15 min
- Hidden word presentation - 15 min
- Letter guessing - 35 min
- Store wrong guesses - 10 min
- Game ending - 15 min
- Dash/hyphen handling - 10 min
- Instant replay after game has ended - 45 min

After implementation is completed, a diagram detailing the attributes of the different classes will be created. This diagram will also specify which classes are depending on other classes during execution. The class diagram should take around 45 minutes to draw.

5.3 Iteration 3

Because both the code and the use cases written in Iteration 2 had room for improvement, a review of both the code and the use cases shall be made to pinpoint what should be changed. Some things that influence if the code/use case should be changed are related to how easy it is to test and whether or not it is easy or difficult to understand. The code review should take around 40 minutes to complete and the use case review should take around 20 minutes to complete.

The reviews should be used to change the code and use cases before the tests are finalised to ensure that no extra work has to be made re-writing all the tests. The refactoring should take around two hours to complete, but this may not be accurate depending on how many things are found to be in need of a change after the code review. Updating the use cases should take around one hour.

After the use cases are updated, activity diagrams should be created for each use case to make it easier to illustrate during manual testing which paths (scenarios) are being tested. These diagrams should take around 45 minutes in total.

Updated use cases and refactored code will have made it so that some states and state transitions in the state machine will not be false, so the state machine will have to be updated. This should take around 40 minutes.

The final thing to do before starting with the testing will be to updated the now invalidated class diagram to reflect the refactored code and class structure. This should take around 40 minutes.

A test plan should be the first thing created in the testing phase. The test plan includes decisions about what to test, why those specific features were chosen to be tested and what techniques are to be used when testing. The document should take around 30 minutes to create, the basic info and use cases should take around 1 hour to include and the actual test plan should take around 30 minutes to write.

After the initial planning is complete, the actual tests can be written. Manual test cases will be written that test every scenario in two of the implemented use cases, which means that if UC1 - Start Game is manually tested, at least four different scenarios will have tests written for them. Activity diagrams should be drawn which illustrate which paths in the use cases that the tests cover. The manual test cases should take around three hours to write, 20 minutes to run and 15 minutes to report. The diagrams should take around 20 minutes to draw.

Automated unit tests will also be written and run of a few different methods (both finished and unfinished methods). To be able to run the tests, some fake/mock classes might have to be created to remove random functionality/use input and console output. The fake classes should take around 1 hour to create. The automated tests should take around around 2.5 hours to write and run. A coverage tool will also be run to measure how much of the implemented code that the unit tests cover. Running the coverage tool and documenting the results should take around 10 minutes. The results of the the unit tests should be added to the test report and it should take around 30 minutes.

After the the rest of the iteration is completed, a short reflection should be written about the experience of creating and documenting tests. This should not take more than 20 minutes.

5.4 Iteration 4

Iteration 4 will begin with correcting previous mistakes in the project documentation. This means that feedback received on iteration 1 will be read and notes will be taken regarding what needs to be changed. When feedback for iteration 2 and 3 is received, this will be repeated. It should take around 45 minutes to read and write down necessary changes from the feedback.

The section "General Information" will be updated in regards to the feedback with the information that was previously missing or false. This should take around 20-25 minutes.

The section "Resources" should then be updated with information that was previously missing. This should take around 30 minutes.

The section "Hard- and Software Requirements" will then be updated like in the previous tasks. This should also take around 30 minutes.

The section "Scope, Constraints and Assumptions" will then be updated to make it clearer what is in scope and what is out of scope and a few additional requirements that are out of scope will also be added to further clarify what the project entails. This should take around 15 minutes.

The section "Vision" will be updated to better reflect the project if necessary and a short justification as to why the project exists will be added. This should take a maximum of 30 minutes.

The section "Overall Project Schedule" will be changed from only including due dates for iteration completions to including what deliverables are fall under the four major due dates. This should take around 45 minutes.

One last use case should added called "View banned words" that lets the user view previously banned words. Along with this, any discrepancies noticed with the use cases should be addressed. In total, this should take around 30 minutes.

Simple activity diagrams illustrating the paths for each use case should then be drawn if none exist or updated if the use case was changed. This should take around 30 minutes.

The state machine should then be updated to reflect what the application contains in this iteration, as some states were marked as not implemented previously. The fourth game mode should also be added, since it never added previously. This should take around 25 minutes.

Implementation will start with implementing the ability to guess on an entire word at once instead of being limited to guessing letter-for-letter. This should not take more than 45 minutes.

Replacing the printout of the number of guesses left is a drawing of a stick figure being hanged. The figure will be progressively drawn whenever a guess is incorrect. This should take around 45 minutes to implement.

To prepare for the new features to be implemented in this iteration, the menu system should be overhauled. This will basically mean that the menu controls will be moved from the main start-up java file to the actual menu java file and reworked from being messy loops to purely using methods. This should make the menus easier to add options to in the future. It should take around 1.5 hours to rework the menus.

The timed game mode should then be implemented and it will contain a timer that is running on a different thread than the rest of the application to allow for printing the time taken on the screen. This should 45 - 60 minutes to implement.

The "endless" game mode should then be implemented. This means that whenever a word is correctly guessed, the game will immediately restart. This should not take more than 30 minutes to code.

The scoring systems should then be implemented. For the endless mode, the score is the number of words correctly guessed and for the timed mode, the score is the time it took to correctly guess the word. It should take around 1 hour to code these two scoring systems.

After the scoring systems are implemented, a system of saving the scores should be implemented. This should allow for the user to at the end of the game save their score to later be viewable from the main menu. This should take around 2 hours to implement.

The score viewing use case should then be implemented, meaning that the previously saved scores should be able to be viewed from an option in the main menu. This should take around 1 hour to implement.

The ability to remove a word after a loss should then be implemented. This means that the word will not be able to be randomly chosen later on. This should take around 1 hour to code.

After implementation is completed, the class diagram should be updated to reflect the new class structure created during implementation. This should take 30 minutes.

The testing phase should then be started, and much like in iteration 3, this will contain manual and automated tests. The first thing is to update the Test Plan to include that the intention is to test all use cases. This should take around 10 minutes.

The manual test cases should then be updated. The old manual cases should still be able to be succeed, but a few more will be added that test the newly implemented game modes. This should take around 2 hours to write.

The manual test cases should then be run and should all succeed. This should take around 30 minutes to do. The test report should also be updated with these results. This should take around 30 minutes also, since some new test cases have been added.

The test environment should then be updated to prepare for automated tests. This will include creating fakes/mocks to removed random functionality and user input/output. This should take around 2 hours to do.

The automated unit tests should then be written and executed, and should all succeed. This should take around 3 hours to do. A coverage tool should also be run to see how much of the code was covered with the tests. This should take around 20 minutes to run and report. The test report should then be updated with the result of the tests. This should take around 40 minutes to do.

6 | Risk analysis

6.1 List of risks

Risk	Probability	Impact
Staff becomes ill	Moderate	Catastrophic
The application only works on the development computer	Low	Catastrophic
Git releases are not publishable/visible	Low	Serious
Requirements change dramatically and require major redesign	Low	Serious
Development time and effort is underestimated	High	Tolerable
The chosen word database doesn't work as expected/can't be used	Moderate	Tolerable
Development computer failure	Low	Tolerable
The development software does not work correctly	Low	Tolerable
The programmer's inexperience results in inefficient problem solving	High	Insignificant

6.2 Strategies

Avoidance: Avoiding the risks specified above mostly involves choosing the safest option where several options are available. This might mean that instead of choosing a database full of words that the programmer has never used before, the words are instead accessed from a place where the complexity is reduced, such as local text files. This also the main way of keeping to the schedule. If a feature is implemented in the safest and most simple way possible instead of the best way possible, the estimates will be more accurate and the schedule easier to follow. Developing the application in Java will mean that the likelihood of the application not working on other computers is reduced, as Java is designed to be platform independent. Avoiding changes to the requirements and the planning means making the vision and the scope of the project as accurate as possible, as early as possible.

Minimisation: Dealing with illness is in this project very important since only one programmer is working on it. If time allows, some things can be worked on before they are scheduled, meaning that the schedule should have some slack planned. Hardware failure is an easy problem to deal with, since all files for the project are backed up in a cloud storage service and therefore always accessible from other computers. Estimation risks can both be avoided and their impact minimised by the same strategy; choosing the least complex solution to problems. Software issues are minimised by having a good understanding of the software the project uses, so that if problems arise, they are more easily solved and require less trouble shooting.

6.3 Reflection

I found it difficult to identify and analyse risks for this project, as this is something I'm very inexperienced with. During previous tasks in other courses, risks have also been a factor that could have been analysed in more depth, but never was. It has just been something that you rarely think about explicitly as risks, but you still take them into consideration when doing rough planning. I think I have managed to come up with somewhat plausible risks, but I feel like I might have missed some very obvious things that can go wrong.

7 | Use cases

7.1 UC1 - Start Game

Precondition: none.
Postcondition: The game is started (See UC2).

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 start playing the game, changing the game mode, view previous scores and quit the game
3. The user chooses to start playing the game
4. The system starts the game (See UC2).

Alternative scenarios

3.1 The user chooses to quit the game
   1. The system terminates (See UC3).

3.2 The user chooses to change the game mode
   1. The system shows the game mode selection menu (See UC5).

3.3 The user chooses to view previous scores
   1. The system shows the previous scores (See UC4).

3.4 The user chooses to view previously banned words
   1. The system shows the banned words (See UC9).

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

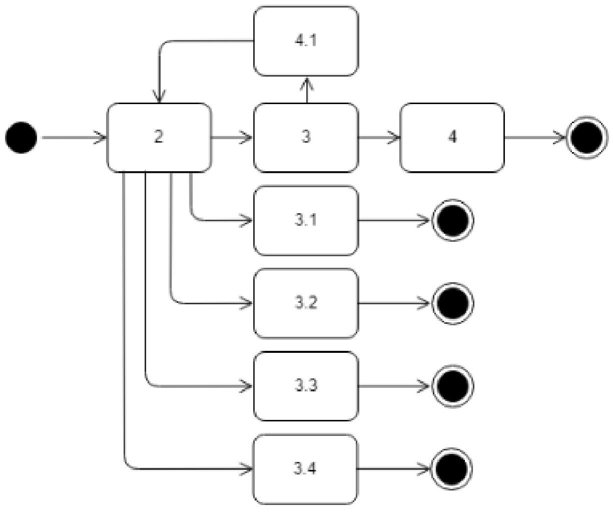


Figure 1: Paths for UC1

7.2 UC2 - Play Game

Preconditions: none.
Postcondition: a victory/loss screen is presented.

Main scenario

1. Starts when the user chooses to play the game
2. The system selects a random word and presents a row of underscores, each representing a letter in the chosen word
3. The user inputs a letter, or an entire word, as a guess to what letters the hidden word might contain
4. The guess is correct and all letters have been guessed, the system presents the victory screen containing a menu with options to "Play again with the same settings", "Go to the main menu", "Save score" (See UC8), "Ban word" (See UC9) or to "Quit" the game (See UC3).

Alternative scenarios

4.1 Correct letter guess, but not all letters have been guessed

1. The system updates and presents the row of underscores with the correct characters now shown
2. Goto 3 in main scenario.

4.2 Incorrect letter guess, but it is not the 10th incorrect guess

1. The system adds the guess to the list of all incorrect guesses
2. Goto 3 in main scenario.

4.3 Incorrect letter guess, 10 total incorrect guesses have been made

1. The system presents the loss screen containing a menu with options to "Play again with the same settings", "Go to the main menu", "Save score" (See UC8), "Ban word" (See UC9) or to "Quit" the game (See UC3).

4.4 Incorrect word guess, but it is not the 10th incorrect guess

1. The system adds the guess to the list of all incorrect guesses
2. Goto 3 in main scenario.

4.5 Incorrect word guess, 10 total incorrect guesses have been made

1. The system presents the loss screen containing a menu with options to "Play again with the same settings", "Go to the main menu", "Save score" (See UC8), "Ban word" (See UC9) or to "Quit" the game (See UC3).

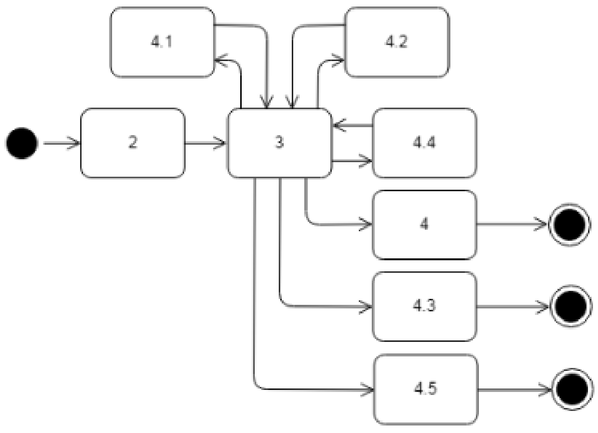


Figure 2: Paths for UC2

7.3 UC3 - 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 returns to its previous state.

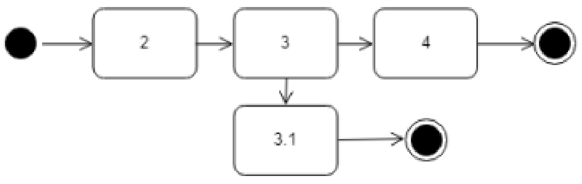


Figure 3: Paths for UC3

7.4 UC4 - View Scores

Precondition: none.
Postcondition: all previous scores are presented.

Main scenario

1. Starts when the user wants to view the list of previous scores
2. The system presents all previously saved scores, the option to return to the main menu and the option to clear all saved scores

3. The user chooses to return to the main menu
4. The system shows the main menu.

Repeat from step 2

Alternative scenarios

3.1 The user chooses to clear all previous scores

1. The system deletes all saved scores
2. Goto 2 in main scenario.

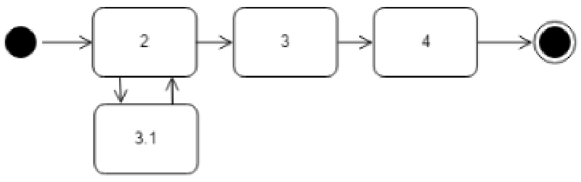


Figure 4: Paths for UC4

7.5 UC5 - Choose game Mode

Precondition: none.
Postcondition: the game mode choice is saved, main menu is presented with the updated game mode.

Main scenario

1. Starts when the user wants to choose which game mode to play
2. The system shows all available game modes
3. The user chooses the basic game mode
4. The system saves the game mode choice and presents the main menu.

Alternative scenarios

3.1 The user chooses the specific word length game mode (game mode 2)

1. The system presents the word length selection prompt (See UC5).

3.2 The user chooses the timed or endless game mode (game mode 3 & 4)

1. The system saves the game mode choice and presents the main menu.

4.1 Invalid menu choice

1. The system presents an error message
2. Goto 2 in main scenario.

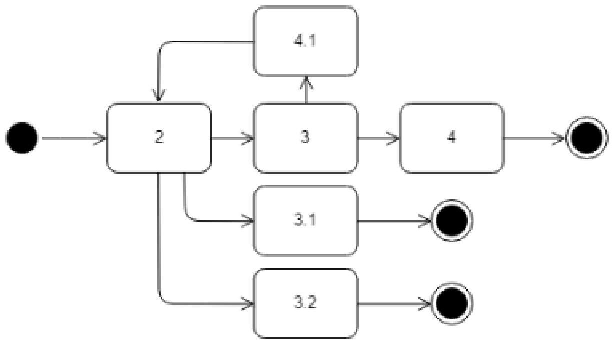


Figure 5: Paths for UC5

7.6 UC6 - Choose Word Length

Precondition: game mode 2 has been selected.
Postcondition: the word length choice is saved, the main menu is presented.

Main scenario
1. Starts when the user wants to choose the length of available words
2. The system prompts the user to specify a length between 2 and 17
3. The user inputs the desired length
4. The system saves the choice and presents the main menu.

Alternative scenarios
4.1 Invalid word length/invalid input
1. The system presents an error message
2. Goto 2 in main scenario.

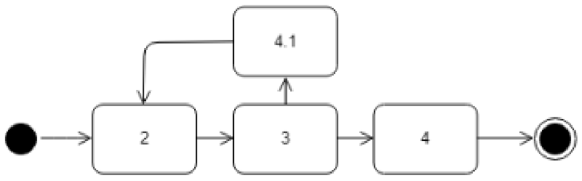


Figure 6: Paths for UC6

7.7 UC7 - Ban Word

Precondition: none.
Postcondition: the last word is removed from the list of available words.

Main scenario
1. Starts when the user wants to ban the last word they guessed on
2. The system prompts for confirmation
3. The user confirms

4. The system removes the last word from the list of available words and returns to the end of game screen.

Alternative scenarios
3.1 The user does not confirm
1. The system returns to the end of game screen.

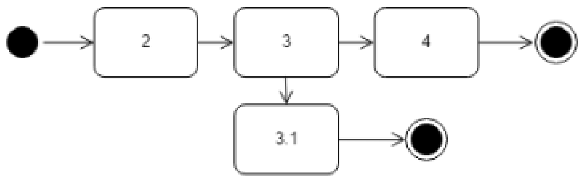


Figure 7: Paths for UC7

7.8 UC8 - Save Scores

Precondition: the user won the game in game mode 3 or 4.
Postcondition: the score is saved.

Main scenario
1. Starts when the user wants to save their last score
2. The system prompts the user to write their name
3. The user enters their name
4. The system saves the score.

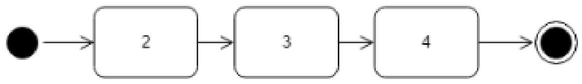


Figure 8: Paths for UC8

7.9 UC9 - View Banned Words

Precondition: none.
Postcondition: all banned words are presented.

Main scenario
1. Starts when the user wants to view which words have previously been banned
2. The system presents all previously banned words, the option to return to the main menu and the option to clear all banned words
3. The user chooses to return to the main menu
4. The system shows the main menu.

Repeat from step 2

Alternative scenarios
3.1 The user chooses to clear all banned words
1. The system deletes all banned words
2. Goto 2 in main scenario.

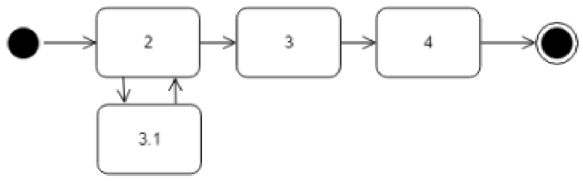


Figure 9: Paths for UC9

7.10 Summary: Use Cases

The following image is an overview of the different use cases and how they are related to one another.

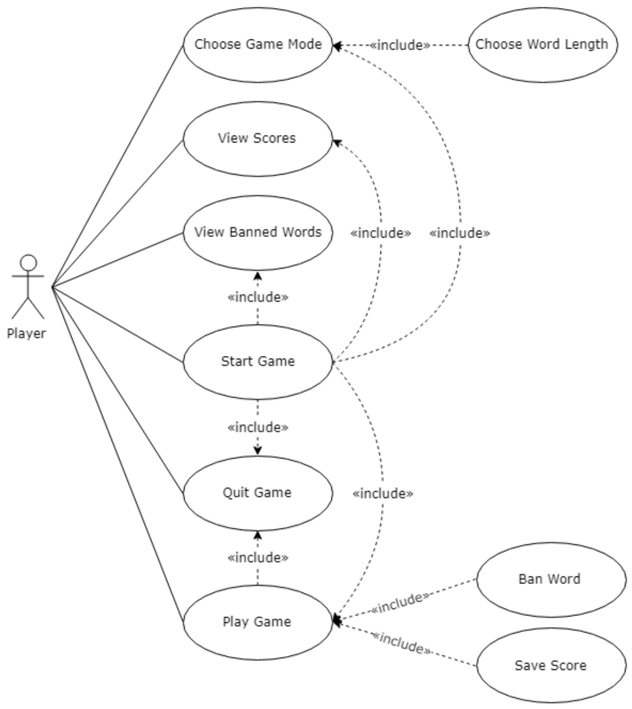


Figure 10: Use Case diagram

8 | Final Implementation

The following image provides an overview of the class structure of the final implementation. See the raw image in the Documentation folder for better resolution. An image of the dynamic behaviour of the system can ben found as a state machine in the same folder.

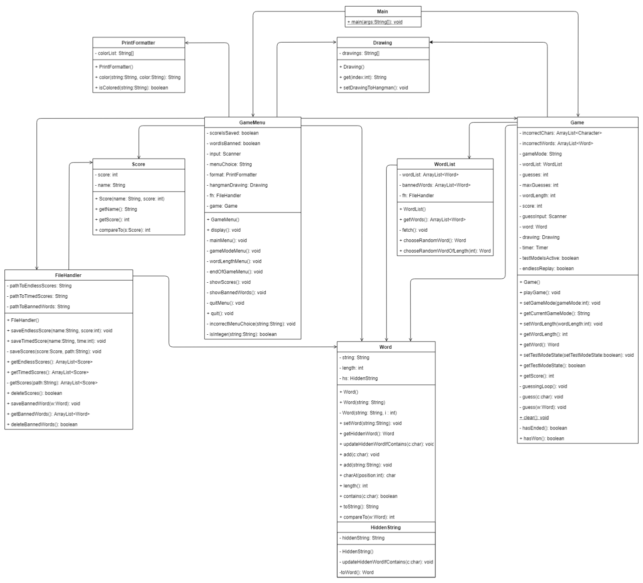

Main
+ main(args:String[]): void
PrintFormatter
- colorList: String[]
+ PrintFormatter()
+ color(string:String, color:String): String
+ isColored(string:String): boolean
Drawing
- drawings: String[]
+ Drawing()
+ get(index:int): String
+ setDrawingToHangman(): void
GameMenu
- scoreIsSaved: boolean
- wordIsBanned: boolean
- input: Scanner
- menuChoice: String
- format: PrintFormatter
- hangmanDrawing: Drawing
- fh: FileHandler
- game: Game
+ GameMenu()
+ display(): void
- mainMenu(): void
- gameModeMenu(): void
- wordLengthMenu(): void
- endOfGameMenu(): void
- showScores(): void
- showBannedWords(): void
- quitMenu(): void
+ quit(): void
- incorrectMenuChoice(string:String): void
- isInteger(string:String): boolean
Score
- score: int
- name: String
+ Score(name: String, score: int)
+ getName(): String
+ getScore(): int
+ compareTo(s:Score): int
FileHandler
- pathToEndlessScores: String
- pathToTimedScores: String
- pathToBannedWords: String
+ FileHandler()
+ saveEndlessScore(name:String, score:int): void
+ saveTimedScore(name:String, time:int): void
- saveScores(score:Score, path:String): void
+ getEndlessScores(): ArrayList<Score>
+ getTimedScores(): ArrayList<Score>
- getScores(path:String): ArrayList<Score>
+ deleteScores(): boolean
+ saveBannedWord(w:Word): void
+ getBannedWords(): ArrayList<Word>
+ deleteBannedWords(): boolean
WordList
- wordList: ArrayList<Word>
- bannedWords: ArrayList<Word>
- fh: FileHandler
+ WordList()
+ getWords(): ArrayList<Word>
- fetch(): void
+ chooseRandomWord(): Word
+ chooseRandomWordOfLength(int): Word
Game
- incorrectChars: ArrayList<Character>
- incorrectWords: ArrayList<Word>
- gameMode: String
- wordList: WordList
- guesses: int
- maxGuesses: int
- wordLength: int
- score: int
- guessInput: Scanner
- word: Word
- drawing: Drawing
- timer: Timer
- testModeIsActive: boolean
- endlessReplay: boolean
+ Game()
+ playGame(): void
+ setGameMode(gameMode:int): void
+ getCurrentGameMode(): String
+ setWordLength(wordLength:int): void
+ getWordLength(): int
+ getWord(): Word
+ setTestModeState(setTestModeState:boolean): void
+ getTestModeState(): boolean
+ getScore(): int
- guessingLoop(): void
- guess(c:char): void
- guess(w:Word): void
+ clear(): void
- hasEnded(): boolean
+ hasWon(): boolean
Word
- string: String
- length: int
- hs: HiddenString
+ Word()
+ Word(string: String)
- Word(string: String, i : int)
+ setWord(string:String): void
+ getHiddenWord(): Word
+ updateHiddenWordIfContains(c:char): void
+ add(c:char): void
+ add(string:String): void
+ charAt(position:int): char
+ length(): int
+ contains(c:char): boolean
+ toString(): String
+ compareTo(w:Word): int
HiddenString
- hiddenString: String
- HiddenString()
- updateHiddenWordIfContains(c:char): void
-toWord(): Word

Figure 11: Class diagram

9 | Test Plan

9.1 Objective

The objective of the following tests described in the test plan is to test the implemented code to verify that it works as intended and to see to that the final product is of good enough quality to be usable.

9.2 What Will Be Tested And How

The intention is to test all use cases dynamically. There will be manual Test Cases of the running client application that intend to cover all paths, within reason, in UC1 and UC2. These two use cases have been selected for manual testing because they cover the functionality most essential to the product, meaning selecting to start playing from after the program starts and actually playing the game. All paths in these two use cases should be covered by the tests. At least one manual tests will also be conducted for each of the remaining use cases, although not as thorough because the functionality covered is less important. Additionally, automated unit tests will be written for the application with the intention to cover as many of the methods in each class, that do not deal with user input and console output, as possible. The reason for not covering all methods in every class is due to some implemented methods relying on randomness and many methods in the largest classes containing user input/output. Due to lack of time, this type of code has not been optimised for automated testing and will therefor be tested manually instead.

10 | Manual Test Cases

10.1 TC1.1 - Start standard game successfully

Use Case: UC1 - Start Game
Scenario: Start standard game successfully
Description: Main Scenario of UC1 ends with the start of UC2, user is able to start playing the game
Precondition: None
Postcondition: The game is started (See UC2 and appendix A.1)

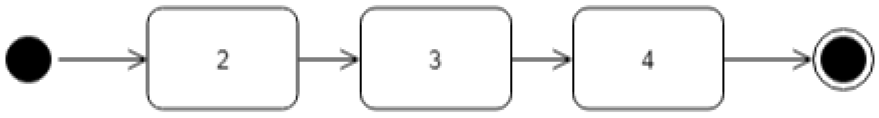


Figure 12: Paths covered in UC1

Test steps

1. Start the application
2. The system shows the main menu, see appendix A.1
3. Press 1 and press enter.

Expected result

1. The system starts the game and shows the game play screen, see appendix A.1

Test result

☑ Pass
☐ Fail

Comments

None.

10.2 TC1.2 - Enter game mode selection menu

Use Case: UC1 - Start Game
Scenario: Enter game mode selection menu
Description: Alt Scenario of UC1, user wants to change game mode instead of playing
Precondition: None
Postcondition: The game mode selection menu is shown (See UC5 and appendix B.2)

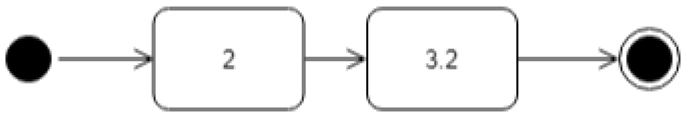


Figure 13: Paths covered in UC1

Test steps
1. Start the application
2. The system shows the main menu
3. Press 2 and press enter.

Expected result
1. The system shows game mode selection menu, see appendix B.2

Test result
- ☑ Pass
- ☐ Fail

Comments
None.

10.3 TC1.3 - View saved scores

Use Case: UC1 - Start Game & UC4 - View Scores
Scenario: View saved scores
Description: Alt Scenario of UC1, user wants to view previous scores instead of playing
Precondition: None
Postcondition: The previous scores are shown (See UC4 and appendix B.3)

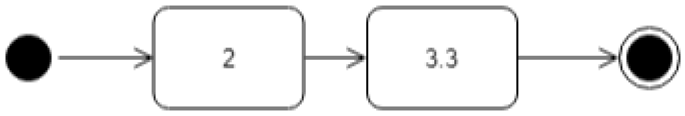


Figure 14: Paths covered in UC1

Test steps

1. Start the application
2. The system shows the main menu
3. Press 3 and press enter.

Expected result

1. The system shows previous scores (if any exist), see appendix B.3

Test result

☑ Pass
☐ Fail

Comments

None.

10.4 TC1.4 - View banned words

Use Case: UC1 - Start Game & UC9 - View Banned Words
Scenario: View saved scores
Description: Alt Scenario of UC1, user wants to view previously banned words instead of playing
Precondition: None
Postcondition: The previous scores are shown (See UC9 and appendix B.4)

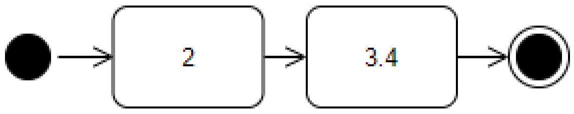


Figure 15: Paths covered in UC1

Test steps

1. Start the application
2. The system shows the main menu
3. Press 4 and press enter.

Expected result

1. The system shows previously banned words (if any exist), see appendix B.4

Test result

☑ Pass
☐ Fail

Comments

None.

10.5 TC1.5 - Enter game termination menu from main menu

Use Case: UC1 - Start Game
Scenario: Enter game termination menu from main menu
Description: Alt Scenario of UC1, user wants to quit the game instead of playing
Precondition: None
Postcondition: Game termination menu is shown (See UC3)

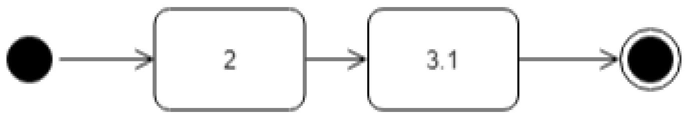


Figure 16: Paths covered in UC1

Test steps

1. Start the application
2. The system shows the main menu
3. Press 5 and press enter.

Expected result

1. The system shows the termination confirmation menu, see appendix B.5

Test result

☑ Pass
☐ Fail

Comments

None.

10.6 TC1.6 - Invalid menu choice force re-prompt

Use Case: UC1 - Start Game
Scenario: Invalid main menu choice forces re-prompt
Description: Alt Scenario of UC1, user enters invalid menu choice and is forced to enter a new menu choice
Precondition: None
Postcondition: The main menu is shown (See appendix B.1)

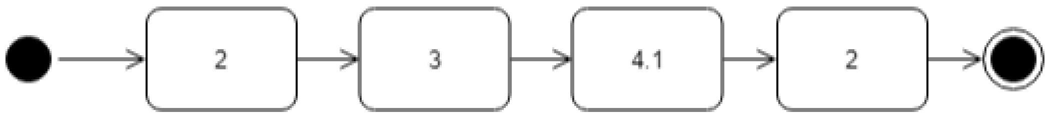


Figure 17: Paths covered in UC1

Test steps

1. Start the application
2. The system shows the main menu
3. Press 5 and press enter.

Expected result

1. The system shows the error message "'5' is an incorrect menu choice"
2. The system removes the error message after a couple of seconds and waits for a new input

Test result

☑ Pass
☐ Fail

Comments

None.

10.7 TC1.7 - Empty menu choice force re-prompt

Use Case: UC1 - Start Game
Scenario: Empty main menu choice input forces re-prompt
Description: Alt Scenario of UC1, user enters empty menu choice and is forced to enter a new menu choice
Precondition: None
Postcondition: The main menu is shown (See appendix B.1)

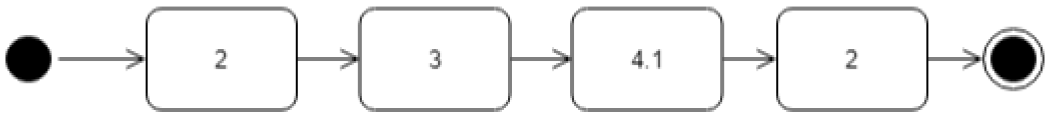


Figure 18: Paths covered in UC1

Test steps

1. Start the application
2. The system shows the main menu
3. Press enter without having typed anything else.

Expected result

1. The system shows the error message "' ' is an incorrect menu choice"
2. The system removes the error message after a couple of seconds and waits for a new input

Test result

[✓ Pass]
☐ Fail

Comments

None.

10.8 TC2.1 - Play game successfully only using letter guessing guess word with no incorrect guesses

Use Case: UC2 - Play Game
Scenario: Guess the word using only letter guesses, with no incorrect guesses
Description: Main Scenario and alt scenario 4.1 of UC2, user plays the game and successfully guesses all letters in the chosen word
Precondition: Complete step 1 in TC1.1, then the word "test" must be entered. Steps 2-3 in TC1.1 must then be successfully completed
Postcondition: The end of game screen is shown (See appendix A.2)

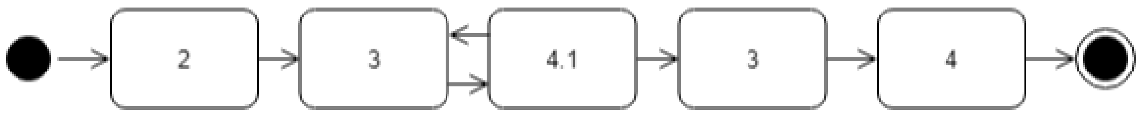


Figure 19: Paths covered in UC2

Test steps

1. The system shows the game playing screen, see appendix A.1
2. Press m and press enter
3. The system shows the same text, but the dashes above "incorrect guesses" are now "m _ _ _ _ _"
4. Press n and press enter
5. The system shows the same text, but the dashed row is now "m _ n n _ _"
6. Press e and press enter
7. The system shows the same text, but the dashed row is now "m _ n n e _"
8. Press a and press enter
9. The system shows the same text, but the dashed row is now "m a n n e _"
10. Press r and press enter

Expected result

1. The system shows the end of game screen, see appendix A.2

Test result

☑ Pass
☐ Fail

Comments

None.

10.9 TC2.2 - Play game successfully using one word guess

Use Case: UC2 - Play Game
Scenario: Guess the word using one single guess
Description: Main Scenario of UC2, user plays the game and successfully guesses the chosen word on the first try
Precondition: Complete step 1 in TC1.1, then the word "test" must be entered. Steps 2-3 in TC1.1 must then be successfully completed
Postcondition: The end of game screen is shown (See appendix A.2)

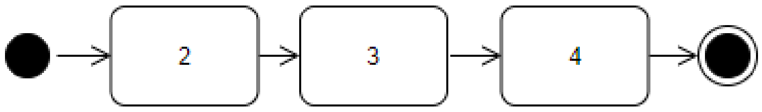


Figure 20: Paths covered in UC2

Test steps

1. The system shows the game playing screen, see appendix A.1
2. Type manner and press enter

Expected result

1. The system shows the end of game screen, see appendix A.2

Test result

☑ Pass
☐ Fail

Comments

None.

10.10 TC2.3 - Play game successfully guess word with 1 incorrect letter guess

Use Case: UC2 - Play Game
Scenario: Play game successfully and guess the word with 1 incorrect letter
Description: Main Scenario, alt scenario 4.1 and alt scenario 4.2 of UC2, user plays the game and successfully guesses all letters in the chosen word but makes a total of 1 incorrect lettter guess
Precondition: Complete step 1 in TC1.1, then the word "test" must be entered. Steps 2-3 in TC1.1 must then be successfully completed. Successfully complete steps 1-9 in TC2.1
Postcondition: The end of game screen is shown (See appendix A.2)

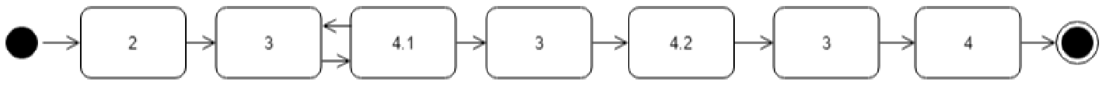


Figure 21: Paths covered in UC2

Test steps

1. The system shows the game playing screen, see appendix A.1
2. Press q and press enter
3. The system shows the same text as before, except "Incorrect guesses: q," and that the square is updated, see appendix A.4
4. Press r and press enter

Expected result

1. The system shows the end of game screen, see appendix A.2

Test result

☑ Pass
☐ Fail

Comments

None.

10.11 TC2.4 - Play game successfully guess word with 1 incorrect word guess

Use Case: UC2 - Play Game
Scenario: Play game successfully and guess the word with 1 incorrect word
Description: Main Scenario, alt scenario 4.1 and alt scenario 4.4 of UC2, user plays the game and successfully guesses all letters in the chosen word but makes a total of 1 incorrect word guess
Precondition: Complete step 1 in TC1.1, then the word "test" must be entered. Steps 2-3 in TC1.1 must then be successfully completed. Successfully complete steps 1-9 in TC2.1
Postcondition: The end of game screen is shown (See appendix A.2)

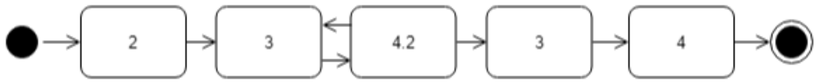


Figure 22: Paths covered in UC2

Test steps
1. The system shows the game playing screen, see appendix A.1
2. Type mannex and press enter
3. The system shows the same text as before, except "Incorrect guesses: mannex" and that the square is updated, see appendix A.4
4. Press r and press enter

Expected result
1. The system shows the end of game screen, see appendix A.2

Test result
☑ Pass
☐ Fail

Comments
None.

10.12 TC2.5 - Play game unsuccessfully

Use Case: UC2 - Play Game
Scenario: Play game unsuccessfully
Description: Alt scenario 4.2 and alt scenario 4.3 of UC2, user plays the game and unsuccessfully guesses on the word, guessing wrong a total of 10 times and correct a total of 0 times
Precondition: Complete step 1 in TC1.1, then the word "test" must be entered. Steps 2-3 in TC1.1 must then be successfully completed.
Postcondition: The end of game screen is shown (See appendix A.3)

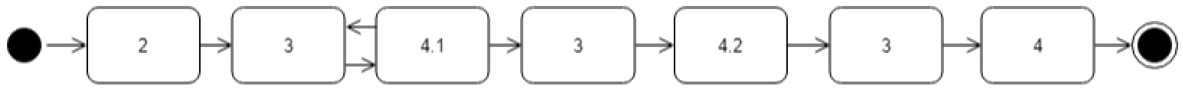


Figure 23: Paths covered in UC2

Test steps

1. The system shows the game playing screen, see appendix A.1
2. Press q and press enter
3. The system shows the same text as before, except "Incorrect guesses: q," and that the square is updated, see appendix A.4
4. Press y and press enter
5. The system shows the same text as before, except "Incorrect guesses: q, y," and that the square is updated, see appendix A.5
6. Press w and press enter
7. The system shows the same text as before, except "Incorrect guesses: q, w, y," and that the square is updated, see appendix A.6
8. Press t and press enter
9. The system shows the same text as before, except "Incorrect guesses: q, t, w, y," and that the square is updated, see appendix A.7
10. Press s and press enter
11. The system shows the same text as before, except "Incorrect guesses: q, s, t, w, y," and that the square is updated, see appendix A.8
12. Press c and press enter
13. The system shows the same text as before, except "Incorrect guesses: c, q, s, t, w, y," and that the square is updated, see appendix A.9
14. Press i and press enter
15. The system shows the same text as before, except "Incorrect guesses: c, i, q, s, t, w, y," and that the square is updated, see appendix A.10
16. Press h and press enter
17. The system shows the same text as before, except "Incorrect guesses: c, h, i, q, s, t, w, y," and that the square is updated, see appendix A.11
18. Press k and press enter
19. The system shows the same text as before, except "Incorrect guesses: c, h, i, k, q, s, t, w, y," and that the square is updated, see appendix A.12

20. Press z and press enter

Expected result

1. The system the end of game screen, see appendix A.3

Test result

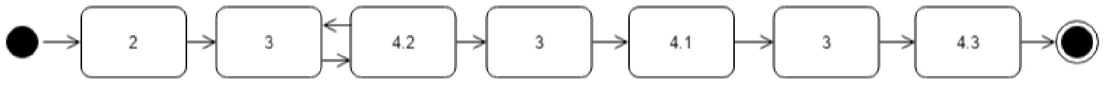
☑ Pass
☐ Fail

Comments

None.

10.13 TC2.6 - Play game unsuccessfully with one correct guess

Use Case: UC2 - Play Game
Scenario: Play game unsuccessfully with one correct guess
Description: Alt scenario 4.1, 4.2 and 4.3 of UC2, user plays the game and unsuccessfully guesses on the word, guessing wrong a total of 10 times and correct a total of 1 times
Precondition: Complete step 1 in TC1.1, then the word "test" must be entered. Steps 2-3 in TC1.1 must then be successfully completed. Then successfully complete steps 1-19 in TC2.5.
Postcondition: The end of game screen is shown (See appendix A.3)



Test steps
1. The system shows the game playing screen, see appendix A.1, except "Incorrect guesses: c, h, i, k, q, s, t, w, y," and that the square is updated, see appendix A.12
2. Press m and press enter
3. The system shows the same rows of text, the dashed row is now "m _ _ _ _ _"
4. Press p and press enter

Expected result
1. The system shows the end of game screen, see appendix A.3

Test result
☑ Pass
☐ Fail

Comments
None.

11 | Test Report

11.1 Manual Testing

Test	UC1	UC2
TC1.1	1/OK	0
TC1.2	1/OK	0
TC1.3	1/OK	0
TC1.4	1/OK	0
TC1.5	1/OK	0
TC1.6	1/OK	0
TC1.7	1/OK	0
TC2.1	0	1/OK
TC2.2	0	1/OK
TC2.3	0	1/OK
TC2.4	0	1/OK
TC2.5	0	1/OK
TC2.6	0	1/OK
COVERAGE & SUCCESS	6/OK	4/OK

11.2 Automated Unit Testing

Test	Drawing	PrintFormatter	Word	WordListMock
DrawingTest	100%/OK	0%/NA	0%/NA	0%/NA
WordTest	0%/NA	0%/NA	100%/OK	0%/NA
WordListTest	0%/NA	0%/NA	25%/OK	91.2%/OK
PrintFormatterTest	0%/NA	100%/OK	0%/NA	0%/NA
COVERAGE & SUCCESS	100%/OK	100%/OK	100%/OK	91.2%/OK

11.3 Reflection and comments

Due to an overall lack of time and an unexpected amount of time needed to update old test cases, not many new test were added. The intention was to have at least one test case for each use case to see that at least the core functionality of each test case worked as intended. As it is now, many use cases remain untested. The same is true for unit tests; the time ran out and only a small part of the code was actually tested (total project code coverage: 30.1%). The project is still an overall success, but the product might contain unwanted behaviour as a result of insufficient testing. This is also why the time log estimations are incorrect for testing, since different work was expected to be done.

12 | Time log

12.1 Iteration 1

Task	Estimate	Actual time	Comments
Create GitHub repository setup and file organization	30 min	1 h 30 min	Setup for GitHub was easy, but organizing the file structure locally presented a few problems related to the chosen software.
Construct documentation and write basic information	2 h	5 h	Working with LaTeX to structure the document in a way that tries to replicate a template ended up being a larger challenge than anticipated.
Write vision	1 h	40 min	This part was shorter and easier to write than first thought.
Write project plan	2 h	2h 30 min	The time spent thinking about what to include took slightly longer than anticipated.
Plan iteration 1	30 min	30 min	
Plan iteration 2	30 min	40 min	Some additional time was needed to understand what iteration 2 will entail.
Plan iteration 3	30 min	10 min	Not much information is currently known about iteration 3, not much could be specified.
Plan iteration 4	10 min	5 min	Iteration 4 will contain everything in previous iterations, will be specified later.
Identify risks	15 min	25 min	
Analyse risk impact	15 min	15 min	
Plan for risks	30 min	20 min	
Create classes and write some skeleton code	20 min	45 min	Some additional organization was required.
Conduct initial application testing	20 min	2 h 30 min	Encountered unexpected problems when trying to execute files from the terminal, mostly related to java versions.
Create Git release	15 min	20 min	

12.2 Iteration 2

Modelling

Task	Estimate	Actual time	Comments
Update Iteration 2 plan	30 min	20 min	
Use case diagram	30 min	1 h	I added a few more use cases and re-structured many times.
Update use case 1 and 3	20 min	35 min	My Use Case Diagram is quite different from the one that UC1 and UC3 were based on, so it took some extra time to edit.
Write fully dressed use case for "Play Game"	45 min	50 min	
Write remaining use cases	1 hr	45 min	
Draw state machine for "Play Game" use case	45 min	1h 30 min	The extra 45 minutes is time spent correcting mistakes and redrawing, based on feedback.
Draw state machine for all application states	1 h	2 h 10 min	Underestimated time needed for the placement of states so transitions never cross.
Draw class diagram	45 min	1 h	

Implementation

Task	Estimate	Actual time	Comments
Code quit menu	15 min	15 min	
Code Game Mode selection menu (after choosing main menu option Play)	20 min	30 min	
Code word length selection menu (after choosing a Game Mode)	15 min	20 min	
Code word fetching	30 min	1 h 30 min	After not finding an online word list to use, I instead started to try using local files. This presented new problems, so I once again searched for an online solution, and found something immediately.
Code completely random and random of specific length word selection	15 min	20 min	
Code letter "tiles" (hide letters, show word length)	10 min	20 min	Also implemented a screen-clear method to show the hidden word on a cleared console window.
Code letter guessing and "tile" updating	35 min	45 min	
Code wrong guesses: store and show	10 min	20 min	Added some extra functionality (primarily sorting).
Code end of game menu: Win/loss	15 min	25 min	
Remove "-" from guesses	10 min	10 min	
Code replay functionality	45 min	1 h 45 min	I thought my estimation was generous, but I ended up having to re-organize some of the menu structure to make it work like I wanted.

12.3 Iteration 3

Task	Estimate	Actual time	Comments
Review all code	40 min	45 min	
Re-organise file structure	30 min	35 min	
Review Use Cases	20 min	15 min	
Refactor code based on review	2h	6h 30 min	Many additional areas of improvements were discovered during refactoring.
Update Use Cases	1h	2.5h	Use cases were updated as test cases were being constructed, to make it easier to write tests.
Update Use Case Diagram	10 min	5 min	
Draw activity diagrams for updated use cases	45 min	55 min	
Update State Machines	40 min	50 min	
Update Class Diagram	40 min	50 min	

Testing

Task	Estimate	Actual time
Document creation & structuring	30 min	35 min
Basic information & Use Cases	1 h	3 h 30 min
Test Plan	30 min	25 min
Write manual Test Cases	3h	3h 30 min
Draw activity diagrams	20 min	35 min
Run manual tests	20 min	30 min
Update test report	15 min	15 min
Prepare test environment (create fakes/mocks)	1h	1h 45 min
Write and run unit tests	2 h 30 min	2 h 50 min
Write and run failing unit test	10 min	10 min
Run and save result of coverage tool	10 min	20 min
Update test report	30 min	1 h
Write reflection	20 min	20 min

12.4 Iteration 4

Planning

Task	Estimate	Actual time	Comments
Read feedback & make notes	1 h	1 h	
Write Iteration 4 plan	1 h	1 h 15 min	
Update General Information	25 min	35 min	
Add details to Resources	30 min	20 min	
Add detail to Hard- and Software Requirements	30 min	50 min	
Move what's out of scope to a separate paragraph	15 min	25 min	Some additional details were added to the scope.
Update Vision (add "why")	30 min	-	
Add detail to Overall Project Schedule	45 min	1 h	
Add detail to Iterations	1 h	1 h 45 min	Some things present in the time log were missing.
Update use cases	30 min	50 min	

Modelling

Task	Estimate	Actual time	Comments
Update use case diagram	25 min	25 min	
Update/draw new activity diagrams for each use case	30 min	35 min	
Update state machine	25 min	30 min	
Update class diagram	45 min	1 h	

Implementation

Task	Estimate	Actual time	Comments
Code word guessing	45 min	35 min	
Code hangman drawing	45 min	1 h	
Menu overhaul	1 h 30 min	1 h 25 min	
Code timer	45 min	1 h 20 min	The timer needed the screen to clear every second, which was ugly, so the implementation was changed.
Code endless mode	45 min	50 min	
Code scoring systems	45 min	30 min	
Code score saving and deleting	2 h	3h 30 min	File handling never was my strong suit
Code score viewing	1 h	45 min	
Code word banning	1 h	1h 10 min	

Testing

Task	Estimate	Actual time
Update Test Plan	20 min	20 min
Write manual Test Cases	2 h	2 h
Draw remaining activity diagrams	20 min	10 min
Run manual tests	30 min	25 min
Update test report	30 min	10 min
Write and run unit tests	3 h	35 min
Run and save result of coverage tool	20 min	10 min
Update test report	40 min	20 min

A | Play Game output

A.1 Start Standard Game

```
 Current game mode: One word
 ___________________
|                   |
|                   |
|                   |
|                   |
|                   |
|                   |
|                   |
|                   |
|___________________|



_ _ _ _ _ _

Incorrect guesses:
Guess:
```

A.2 Win Standard Game

```
====== YOU WIN! ======
 ___________________
|                   |
|    ___________    |
|   |  _     _  |   |
|   | |_|   |_| |   |
|   |           |   |
|   |  \_____/  |   |
|   |___________|   |
|                   |
|___________________|

The correct word was manner



1. Play again with same settings
2. Go to main menu
3. Save score
4. Ban word
5. Quit
```

A.3 Lose Standard Game

```
====== YOU LOSE! ======

 -------------------
|      -------      |
|     _|_     |      |
|    |x x|    |      |
|    |_-_|    |      |
|      |      |      |
|     /|\     |      |
|     / \   _|_      |
|          /   \     |
|___________________|

The correct word was manner


1. Play again with same settings
2. Go to main menu
3. Save score
4. Ban word
5. Quit
```

A.4 One incorrect guess

```
 -------------------
|                   |
|                   |
|                   |
|                   |
|                   |
|                   |
|           ---     |
|          /   \    |
|___________________|
```

A.5 Two incorrect guesses

```
 -------------------
|                   |
|            |      |
|            |      |
|            |      |
|            |      |
|            |      |
|           _|_     |
|          /   \    |
|___________________|
```

A.6 Three incorrect guesses

```
 ___________________
|      _______      |
|            |      |
|            |      |
|            |      |
|            |      |
|            |      |
|           _|_     |
|          /   \    |
|___________________|
```

A.7 Four incorrect guesses

```
 ___________________
|      _______      |
|      |     |      |
|            |      |
|            |      |
|            |      |
|            |      |
|           _|_     |
|          /   \    |
|___________________|
```

A.8 Five incorrect guesses

```
 ___________________
|      _______      |
|     _|_    |      |
|    |x x|   |      |
|    |_-_|   |      |
|            |      |
|            |      |
|           _|_     |
|          /   \    |
|___________________|
```

A.9 Six incorrect guesses

```
 -------------------
|      _______      |
|     _|_    |      |
|    |x x|   |      |
|    |_-_|   |      |
|      |     |      |
|      |     |      |
|           _|_     |
|          /   \    |
|___________________|
```

A.10 Seven incorrect guesses

```
 -------------------
|      _______      |
|     _|_    |      |
|    |x x|   |      |
|    |_-_|   |      |
|      |     |      |
|     /|     |      |
|           _|_     |
|          /   \    |
|___________________|
```

A.11 Eight incorrect guesses

```
 -------------------
|      _______      |
|     _|_    |      |
|    |x x|   |      |
|    |_-_|   |      |
|      |     |      |
|     /|\    |      |
|           _|_     |
|          /   \    |
|___________________|
```

A.12 Nine incorrect guesses

```
 -------------------
|      _______      |
|     _|_     |     |
|    |x x|    |     |
|    |_-_|    |     |
|      |      |     |
|     /|\     |     |
|     /      _|_    |
|          /   \    |
|___________________|
```

B | Menus

B.1 Main Menu

```
 ============ HANGMAN GAME ============

1. Play game (One word, length: random)
2. Change game mode
3. View scores
4. View banned words
5. Quit

Select 1, 2, 3, 4 or 5
```

B.2 Game Mode Menu

```
Select a game mode:
1. Guess on a random word
2. Guess on a random word with a specific length
3. Guess on as many words as you can until you lose
4. Guess on a random word, with a timer!
```

B.3 View Scores

```
TIMED
No scores saved!



ENDLESS
```

```
No scores saved!

Type 'delete' to clear all previous scores
Type anything else to return to the main menu
```

B.4 View Banned Words

```
Banned words:
No words have been banned!

Type 'delete' to clear all banned words
Type anything else to return to the main menu
```

B.5 Quit Menu

```
Are you sure you want to quit?
1. Press 1 to confirm
2. Press any other key to return to the previous menu
```