HANGMAN GAME DEVELOPMENT PROJECT

ADAM ELFSTRÖM
Programvaruteknik
8th 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

2 | General information

Project summary	
Project Name	Project ID
Hangman Game	ae223nv_1dv600
Project Manager	Main Client
Adam Elfström	Students
Key Stakeholders	
Adam Elfström, Therese Forsberg, Daniel Toll, Tobias Andersson Gidlund, Kenny Ek, Tobias Olsson	
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.	

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.

Programmer: 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.

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 0 Swedish kronor. Every tool and resource used during the project - such as a computer, word-processing software and the terminal used to execute the application - is already available to all involved parties or available for free. Travels to and from the University where all involved parties work and/or study involve walking or biking.

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. Each iteration takes two weeks.

4.5 Hard- and Software Requirements

Hardware: The project will produce an application that will be designed to be executed in a terminal on a computer. This means that a functioning computer will be required both for the programmer during implementation and the end user.

Software: The programmer's computer needs to have a recent installation of Java during the implementation phase. The code for the application needs to be written in a basic word processing program, or a more sophisticated IDE, such as Eclipse or Netbeans. Some form of word processor is also needed for documentation.

4.6 Overall Project Schedule

Date	Description
08/02/2019	Iteration 1 completion
22/02/2019	Iteration 2 completion
08/03/2019	Iteration 3 completion
18-24/03/2019	Iteration 4 completion

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, a timed standard version, an infinite version and a timed infinite version. The standard modes only involve one word that needs to be guessed and the infinite modes 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 number of words/characters correctly guessed during the timed modes, 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. This means that words such as "add-on" will only require the user to guess the "add" and "on". The application won't have a feature to allow for online activities such as multi-user or online scores from a database, everything will be stored locally. The application could have some features allowing for customization, meaning that certain aspects of the game can be changed by the user. This customization 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. 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.

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 the game 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.

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.

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.

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. This project plan should include a short reflection about its creation.

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.

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.

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 skeleton code.

5.2 Iteration 2

To start, the plan for Iteration 2 will be updated with new information regarding the activities iteration will involve. 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 they will together form a Use Case Diagram (use case model). This diagram presents what the system will be able to do for the end user, and what is depending on external factors. The Use Case diagram will contain all written use cases. Time estimates for the diagram and the written use cases will be added to the time log.

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). Time estimates for the two state machines will be added to the time log.

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. Time estimates for the different activities of the implementation will be added to the time log.

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. Time estimate for the class diagram will be added to the time log.

5.3 Iteration 3

Iteration 3 is all about testing, and a test plan should be the first thing created. 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.

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 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. 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.

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. If a use case includes other use cases, like UC1 does, these use cases will be included also be tested due to being part of scenario in the use case. Automated unit tests will also be written and run of a few different methods (both finished and unfinished methods). A coverage tool will also be run to measure how much of the implemented code that the unit tests cover. All tests must be well documented in both text and image.

5.4 Iteration 4

Iteration 4 includes everything mentioned in all previous iterations, meaning that iteration 4 will be a complete reiteration that builds upon the previous steps to add new functionality. More details and time estimate will be added later.

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 schedule, 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 be both 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 very 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've missed some very obvious things that can go wrong.

7 | Use cases

7.0.1 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).

4.1 Invalid menu choice

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

7.0.2 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 as a guess to what the word contains
4. The guess is correct and all letters have been guessed, so the system presents the victory screen containing a menu with options to play again, save the score (See UC8), go to the main menu or to quit the game (See UC3).

Alternative scenarios

4.1 Correct guess, 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 guess, but it is not the 10th incorrect guess

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

4.3 Incorrect guess, for the 10th time

1. The system presents the loss screen containing a menu with options to play again, ban the word (See UC7), go back to the main menu or to quit the game (See UC3).

7.0.3 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.

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

Repeat from step 2

7.0.5 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 game mode where you can specify word length
1. The system presents the word length selection prompt (See UC5).

3.2 The user chooses the timed or the survival game mode
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.

7.0.6 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.

7.0.7 UC7 - Ban Word

Precondition: the user lost the game.
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 were unable to guess
2. The system prompts for confirmation
3. The user confirms
4. The system removes the last word from the list of available words.

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

7.0.8 UC8 - Save Scores

Precondition: the user won the game.
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 writes their name
4. The system prompts for confirmation
5. The user confirms
6. The system saves the score.

Alternative scenarios

5.1 The user does not confirm

1. The system returns to the end of game screen.

8 | Time log

8.1 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	

8.2 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.

8.3 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, some functionality was moved into new classes.
Update Use Cases	1h	2.5h	Use cases were updated as test cases were being constructed, to make it easier to write tests.
Draw activity diagrams for updated use cases	45 min	55 min	
Update Use Case Diagram	10 min	5 min	
Update State Machines	40 min	50 min	
Update Class Diagram	40 min	50 min	
Write the test plan	-	-	See Section 2.3 in Test.pdf for time estimates related to testing.