Module	Engineering1 (Eng1) - COM00019
Assessment Title	Assessment 2, Cohort 2
Team	Quackthulu (Team 16)
Members	Aaron Price, Alec Coates, Charlie Curedale-Rayner, Eleanor Griffin-Smith
Deliverable	Requirements

Introduction

Prior to developing and implementing code, the team decided it best to set out project requirements. This would:

- aid in the development process.
- help to keep the project on track with stakeholders' goals.
- prevent the team from working on any unnecessary features.

This was done at the start of the project to avoid common pitfalls that had been outlined through research, such as the tendency to make assumptions, and eliciting unbounded and vague requirements. [1]

The requirements were broken down into the following categories based on our research: User, Functional, and Non-Functional. [2]

Throughout the process of identifying and formatting requirements, the definition of a well-defined requirement was focused on: A well-defined requirement is a statement of system functionality that must have proof of its validation and must be met to achieve a customer's objective for a system. [3]

How requirements were elicited and negotiated

After reading the project brief document, a brainstorming session took place in which the team outlined a list of clarifications that would be needed before eliciting requirements. These were taken to a Customer meeting, in which the stakeholders were asked to make these clarifications, and their responses were noted so they could be used to outline the capabilities of the system [4]. This proved useful when finalising requirements. This method was used again during assessment 2.

Due to the team only being a team of 4 the following requirement were removed from our project: Implement five power-up packs, which can be found floating down the river and be picked up by boats. These might include patching materials that allow partially restoring robustness, or temporarily improving some other characteristic (e.g., speed).

With this information, a Single Statement of Need (SSON) was created alongside a final list of User Requirements for the project.

Our SSON is as follows: "The system should allow users to compete in an exciting and strategic single player boat race game of varying difficulty and have the ability to save and return later".

The SSON acted as a guide for maintaining focus on the system requirements that would be used in the Architecture plan.

Why requirements are presented as they are

To format the elicited set of requirements, the 'IEEE Guide for Developing System Requirements Specification' provided substantial insight into a sensible format and the necessary level of detail for the requirements.

To distinguish types of requirements, colours were used to highlight differences, making the reflection process ergonomic. A tabular approach was decided upon as it neatly stored the information and meant it would be easily adaptable in case of further change.

Alongside requirements, constraints and use cases (see Appendix) were defined to help meet the stakeholders' goals.

Risks associated with these requirements are detailed in the table below and were then addressed during risk assessment and mitigation.

User Requirements

Requirement ID	Description	Risks and Assumptions (if relevant)	Priority Level
UR_MAINTAIN_PREVIOUS	Maintain the previous assessment user requirements. (find in assessment 1 deliverables section of website).	May cause issues when implementing new requirements.	Shall
UR_CHOOSE_DIFFICUTY	The player can select at what skill level they want to play at.	The player may not enjoy the game if the difficulties re too extreme.	Shall
UR_SAVE	The player can save the status of the game at any point.	Must be fully implemented or the function could be used to cheat.	Shall
UR_LOAD	The player can load their previous save and continue playing the game.	Must be fully implemented or the function could be used to cheat.	Shall

System Requirements - Functional

Requirement ID	Description	Risks and Assumptions	Design Requirement
FR_MAINTAIN_PREVIOUS	Maintain the previous assessment functional requirements. (find in assessment 1 deliverables section of website)	May cause issues when implementing new requirements.	UR_MAINTAIN_PREVIOUS
FR_DIFFICULTY	Depending on which difficulty is selected, the obstacles, and other boat AI will change	The hard level should not be impossible.	UR_CHOOSE_DIFFICUTY
FR_DIFFICULTY_SCREEN	The game should launch a screen where the player can select a difficulty.	The player may not know where to navigate to from the screen.	UR_CHOOSE_DIFFICUTY
FR_SAVE	The game must store all positions and variable of a race at the moment it is saved.	All variables must be saves to completely save the players progress.	UR_SAVE
FR_SAVE_SCREEN	The game should launch a screen where the player can save their progress	The player may not know where to navigate to from the screen.	UR_SAVE
FR_LOAD	All variables are loaded and the user may continue	All variable must be loaded to properly restore the	UR_LOAD

	to play	game.	
FR_LOAD_SCREEN	The game should launch a screen where the player can select which save they want to load	The player may not know where to navigate to from the screen.	UR_LOAD

System Requirements - Non-Functional

Requirement ID	Description	User Requirements	Fit Criteria	Risks and Assumptions
NFR_MAINTAIN_PREVIOUS	Maintain the previous assessment non-functional requirements. (find in assessment 1 deliverables section of website)	All previous	All previous	May cause issues when implementing new requirements
NFR_DIFFICULTY	Difficulty settings should be logical and not too extreme.	UR_CHOOSE_DIFFICUTY	New player should be able win on easy most of the time. It takes experience to win on hard.	Assumes the user is aware of the rules and makes reasonable choices.
NFR_QUICK_LOAD	Player must be able to save and lead their game in a reasonable amount of time.	UR_LOAD UR_SAVE	Player must be able to load a game in under 5 seconds.	Assumes player has hard drive space.

Constraints

Requirement ID	Description
CON_APPOPRIATE	The game shouldn't contain explicit or graphic content, such as blood or gore, and should be suitable for any user in the target audience.
CON_LANGUAGE	The game must be programmed in Java.
CON_RUN	The game should be able to run on any University of York Computer Science Department computer, running Windows or Linux.
CON_ACCESSIBLE	The game must use colours with high contrast to ensure all users can play it.
CON_COURSE	Must be based on the river course in York

References

- [1] 'IEEE Guide for Developing System Requirements Specifications', *IEEE Std 1233, 1998 Edition*, pp. 14, Dec. 1998. [Accessed: 05 Nov. 2020]
- [2] D. Thakur, (2013, November.23), *What is Software Requirement? Types of Requirements*. [Online]. Available: https://ecomputernotes.com/software-engineering/softwarerequirement. [Accessed: 05 Nov. 2020].
- [3] 'IEEE Guide for Developing System Requirements Specifications', *IEEE Std 1233, 1998 Edition*, pp. 11, Dec. 1998. [Accessed: 05 Nov. 2020]
- [4] 'IEEE Guide for Developing System Requirements Specifications', *IEEE Std 1233, 1998 Edition*, pp. 16, Dec. 1998. [Accessed: 05 Nov. 2020]

Appendix

1 - Use Case For "Qualifying for the final race"

- Primary Actor: Player
- Precondition: Player has completed 3 legs and health has not fell below 0
- Trigger: Player has completed the third leg
- Main Success Scenario:
    1. System tracks fastest times across all 3 qualifying legs
    2. Player has achieved one of the fastest times
- Secondary scenarios:
    1. Player has not achieved one of the fastest times
    2. Player loses the game. Loss screen is displayed.
- Success Postcondition: Player proceeds to the final race
- Minimal Postcondition: Player does not qualify for the final race. Player loses the game.

2 - Use Case For "Achieving a Medal"

- Primary Actor: Player
- Precondition: Player has qualified for the final race
- Trigger: The start of the final race
- Main Success Scenario:
    1. System tracks time taken for boat to cross the finish line
    2. Player crosses the finish line
    3. Player achieved one of the fastest three times
- Secondary scenarios:
    1.1 Player does not complete the race, health has fallen below 0
        1.1.1 Player loses the game
    2.1 Player does not achieve one of the fastest three times
        2.1.1 Player loses the game
- Success Postcondition: Player is awarded 1st, 2nd or 3rd place medal.
- Minimal Postcondition: Player does not achieve top 3 position. Player loses the game.

3 - Use Case For "Resume game"

- Primary Actor: Player
- Precondition: Player has saved a game
- Trigger: Player clicks save file button

- Main Success scenario:
    1. Systems checks that save file exists
    2. The file exists and is in full working order
- Secondary scenarios:
    1. Systems checks that save file exists
    2. The file exists and is corrupted
- Success Postcondition: Game screen loaded at the point it was saved at.
- Minimal Postcondition: Game screen does not load and player is redirected to menu screen.