Assessment: 1

Deliverable: Requirements

Team Name: Team 8

Team members: Charlie Hayes, Matilda Garcia, Joshua Stafford, David Kayode, Ionut Manasia, Matthew Tomlinson

Requirements

Introduction

A set of requirements is necessary for a software engineering project to be successful. By laying out a set of requirements in a clear but detailed manner at an early stage, every team member can have a complete picture of what is needed for the system (in addition to what is not needed). An important result of eliciting and documenting requirements is that it ensures the resulting system we produce will meet both the customer's quality and functional expectations while avoiding unnecessary allocation of resources on features or polish that the customer neither wants or expects.

Before creating the complete requirements documentation, research was needed to fully understand the process. In addition to the information laid out in the lecture on requirements engineering, the IEEE standard on Requirements Engineering [1] was consulted to recognise the steps needed to complete the process to an appropriate level. An important part of this research revealed a list of recommended techniques of eliciting requirements [2]. The techniques that we felt were appropriate for our situation and project size were brainstorming and interviews. Given we had direct access to the customer and no previous system to examine, we felt that questionnaires and 'observation of environment' were redundant. While briefly used, an in-depth analysis of the market seemed unnecessary given the unique nature of the game required and the clear idea of the final product the customer had.

The research completed also revealed the detail a given requirement should have as well as how the collection of requirements should be laid out [3], [4]. A key piece of information was that any given user requirement should be well developed, having measurable conditions that can be verified (given as functional or non-functional requirements) [3]. All requirements should also be uniquely identifiable so they may be cross-referenced from other documentation or referred to easily by a team member [4]. To help elicit requirements, a use case may be helpful [5] so use cases that we used are included (see appendix).

After research, we could begin to elicit requirements in detail. This began with a brainstorming session in which we analysed the product brief and came up with a set of questions for the customer and noted any obvious requirements. We then arranged, and completed a meeting with the customer for our system during which we gathered and added requirements to our list. These requirements could then be added into tables. For our requirements, we have chosen to show them in 3 tables: User Requirements; Functional Requirements and Non-Functional Requirements. Each requirement is given an identifier and a priority (**Shall**, **Should**, **May**) is assigned to each user requirement. For the functional and non-functional requirements, they contain the user requirement(s)'s ID for which they address. Non-functional requirements also contain a fit criteria which quantifies the intention. A fourth and final table is included, identifying constraints on the project, determined by the customer.

SSON: "The player should control a boat to race 3 legs and a final against AI controlled boats while dodging obstacles, each boat should have unique stats and have suitable difficulty to be fun for the player."

User Requirements

ID	Description	Assumptions, risks, alternatives	Priority
UR_WIN	The user should be able to win the game	The game may be too difficult to win	Shall
UR_LOSE	The user should be able to lose the game		Shall
UR_BOATS	There should be different kinds of boats	Balancing boats may be tricky	Shall
UR_SCREENS	There should be different screens that tell the user what is happening when outside of rounds		Shall
UR_TEAMS	The player will race against other teams		Shall
UR_ROUNDS	There will be 4 rounds that the player has to win to win the game as a whole, with the 4th being a final	Too many rounds might bore the player	Shall
UR_LANES	There should be lanes in the river that the player and other teams must stay in		Shall
UR_OBJECTS	Objects must be present that the player has to avoid to not lose HP on the boat	Too many obstacles may harm game performance or make the game too difficult	Shall
UR_BGM	There should be background music throughout the game	Background music may annoy the user	May
UR_SOUND_EFFECTS	Objects and boats in the game will make sounds depending on what they are doing		May
UR_MOVEMENT	The user should be able to move their boat around the screen during a round	Assumes conventions such as understanding the arrow keys to produce movement	Shall
UR_DIFFICULTY	The rounds should get harder as you move on	Too steep of a difficulty increase may frustrate the user	Should
UR_FUN	The game should be fun to play	This assumes that the user enjoys racing games	Shall

Functional Requirements

ID	Description	Assumptions, risks	User Requirement(s)
FR_BOAT_NUM	There should be a set number of boats to choose from	This assumes that a boat selection screen has been implemented	UR_BOAT_CHOICE
FR_ROUND_WIN	Out of the times each team got in each round the fastest ones will be put in the final		UR_ROUNDS UR_WIN
FR_SIMULTANEOUS	The player races against the other teams simultaneously	May mean each boat has to be visually small to fit them all on one screen	UR_TEAMS

FR_NO_OF_BOATS	There should be 6-9 boats that the player is racing against	Too many boats will make the lanes too small, too few boats may make the game less fun	UR_TEAMS
FR_HP_RECOVERY	The robustness of the boat should recover at the end of the round		UR_BOAT_STATS
FR_COLLISIONS	The boat must lose a bit of HP and all speed when it collides with an object	Frequent collisions may harm the pacing of the game	UR_OBJECTS
FR_BOAT_COLLISIONS	The player boat should be able to collide with other teams boats, which will make both boats lose HP and speed		UR_OBJECTS UR_TEAMS
FR_HARDER_OBJECTS	In later rounds new objects could be introduced with new patterns		UR_DIFFICULTY UR_OBJECTS
FR_WELCOME	There should be a welcome screen that the user sees which leads on to the boat selection screen		UR_SCREENS
FR_BOAT_CHOICE	There will be a screen with a choice of around 6 different boats	If boats are unbalanced, the player will always choose the same boat	UR_SCREENS
FR_WIN_SCREEN	There should be a screen that shows if the player wins the final	This assumes that the player is able to win the game	UR_WIN UR_SCREENS
FR_LOSS_SCREEN	There should be a screen that shows if the player loses against another team in the final		UR_SCREENS UR_TEAMS
FR_BROKEN_SCREEN	There should be a screen for when the player boats HP gets to 0		UR_SCREENS UR_OBJECTS UR_LOSE
FR_FINAL_TO_WELCOME	The final screen should go to the welcome screen with the press of a button		UR_SCREENS
FR_BOAT_STATS	Each boat should have a different value for these stats: Maneuverability, robustness, acceleration and speed		UR_BOATS
FR_STAMINA	The player boat should have stamina that gets depleted over the course of the round	The stamina system may interrupt play in a way that frustrates the user	UR_BOATS
FR_STAMINA_RECOVERY	You can recover stamina by stopping in the race		UR_BOATS
FR_STAMINA_RESET	Stamina should reset at the end of the round		UR_BOATS UR_ROUNDS
FR_HEALTH_BAR	The player should have a visual representation of how much health they have left		UR_BOATS

FR_STAMINA BAR	The player should have a visual representation of how much stamina they have left		UR_BOATS
FR_OPPONENTS	Other teams boats should be able to dodge obstacles and try and get the best time possible	If the opponents are too good, the game may become unwinnable	UR_TEAMS UR_FUN
FR_OPPONENTS_BOATS	Opponents boats are not invulnerable and can be broken	This assumes that the opponents AI will allow them to make mistakes and collide with obstacles	UR_BOATS UR_TEAMS
FR_OPPONENT_DIFFICULTY	Other teams should get better as the rounds go on	This assumes that the opponents start at a reasonable difficulty	UR_DIFFICULTY UR_TEAMS UR_FUN
FR_LANE_PENALTY	If a boat leaves its lane, a penalty of some kind should be applied		UR_LANES

Non-Functional Requirements

ID	Description	Assumptions, risks, alternatives	User Requirement(s)	Fit Criteria
NFR_BOATS_BALANCED	All the boats the user can choose from must be balanced	This assumes that the balancing affects the AI	UR_BOAT_CHOICE UR_FUN	A boat should not be the fastest and most maneuverable, there should not be one boat that wins over 50% of the time
NFR_FIRST_ROUND	The first round will be a practice round		UR_ROUNDS	Times will not matter in this round so it doesn't matter if the player does not do too good
NFR_GAME_TIME	The full game should take between 5-10 mins		UR_ROUNDS	10 is a bit long, 7 mins will be perfect
NFR_ROUND_TIME	Each round should take 1-2 minutes		UR_ROUNDS	This will make it a long enough game overall

Constraints

ID	Description
C_DIMENSION	There is a preference for the graphics being 2D
C_LIBRARY	There is no preference to which game library is used
C_LANES	Do not make the lanes too narrow so the user doesn't have enough space, or too wide so the user doesn't have to try to keep within the lanes
C_GORE	There should be no violence gore or blood
C_CONSISTANCY	The graphical style should remain consistent
C_GRAPHIC_TIME	Don't spend too much time on graphics

Appendix

Use Cases

Name: “winning the game”
Actors: The player
Precondition: The player has selected the boat that they want to play the game with and started the first round
Trigger: The player gets to the end of the final round in the quickest time possible
Main Success Scenario:

- The player gets to the final
- They work their way to the end of the course by dodging obstacles and staying in the correct lanes
- They beat the other teams time and come first place

Secondary Scenarios:

1. The players boat breaks before the end of the final
2. Another team gets a quicker time than the player

Success Postcondition: Player gets the win screen
Minimal Postcondition: Player gets one of the two loss screens

Name: “Avoiding an obstacle”
Actors: The player
Precondition: the player has selected a boat and started a round
Trigger: the player sees an obstacle in the path of their boat
Main Success Scenario:

- The player presses on either the left or right key to move in the assigned direction
- The players boat moves in the direction that the player wants it to

Secondary Scenarios:

1. The player does not move in time

Success Postcondition: the player dodges the obstacle
Minimal Postconition: the player hits the obstacle and the boat loses some HP and all of it’s speed

References

[1] "ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes -- Requirements engineering," in ISO/IEC/IEEE 29148:2018(E) , vol., no., pp.1-104, 30 Nov. 2018, doi: 10.1109/IEEESTD.2018.8559686.

[2] "ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes -- Requirements engineering," in ISO/IEC/IEEE 29148:2018(E) , vol., no., pp.31, 30 Nov. 2018, doi: 10.1109/IEEESTD.2018.8559686.

[3] "ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes -- Requirements engineering," in ISO/IEC/IEEE 29148:2018(E) , vol., no., pp.10, 30 Nov. 2018, doi: 10.1109/IEEESTD.2018.8559686.

[4] "ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes -- Requirements engineering," in ISO/IEC/IEEE 29148:2018(E) , vol., no., pp.71, 30 Nov. 2018, doi: 10.1109/IEEESTD.2018.8559686.

[5] "ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes -- Requirements engineering," in ISO/IEC/IEEE 29148:2018(E) , vol., no., pp.30, 30 Nov. 2018, doi: 10.1109/IEEESTD.2018.8559686.