Team17 - Gnocchi Games

Henry Overton
Yousif Habib
Ben Dunbar
Lucy Newton
Adam Blanchet
Thomas Heenan

Requirements

Requirements

Our team used a 4-Phase Requirement Elicitation process.

Phase One

This involved gathering requirements, which were put together using both the product brief and the information we had received during our initial and follow-up Team-Customer meetings.

Phase Two

This had our primary developers taking the requirements gathered in Phase One and prioritising them in accordance with the importance of each requirement.

Phase Three

Requirements would be scrutinised for possibly ambiguities or incorrectness by all members of the team, with the customer providing clarification where necessary.

Phase Four

The documentation of all requirements would be formalised with clear and easy-to-read tables to give a systematic and formatted view of everything.

The requirements document begins with a SSON (Simple Statement Of Need) which gives a clear picture of what a final and complete version of the system should be able to provide.

In the documentation, requirements are split into four separate tables, one for each type of requirement. These are:

- User
- Functional
- Non-Functional
- Constraint

For *User*, *Functional*, and *Non-Functional* each is given a unique ID beginning with a prefix dependent on the type of requirement (*UR*, *FR*, *NFR*, respectively).

All requirements in the *User* table are prioritised using the key words *Shall*, *Should*, and *May*.

Shall means that requirement will absolutely be in the final version of the product.
Should means that while the requirement should be in the final version, we cannot absolutely guarantee we will be able to include it.
May means that while the requirement would be beneficial to the overall quality of the product, it is not an absolute necessity and will only be included if we have the time and resources.

Each functional and non-functional requirement is connected to a corresponding user requirement that it helps satisfy. In the *Functional* and *Non-Functional* tables, each requirement references the ID of the requirement in the *User* table it is linked to. In the *Non-Functional* table each requirement also has a “Fit Criteria”, this is a brief description of what will be done to make sure that specific requirement is met.
Finally, the *Constraint* table consists of two columns. *Constraint*, which simply contains the type of constraint, and *Description*, which details the specifics of that particular constraint.

Simple Statement Of Need
A casual single-player game based on the York Dragon Race, where users play as a boat team competing in a race, consisting of three legs and a final.

User Requirements

ID	Description	Priority
UR_PLAYABLE	The game **shall** be playable	Shall
UR_RUNNABLE	The game **shall** be runnable on standard computer hardware	Shall
UR_MOVE_BOAT	The user **shall** be able to move their boat	Shall
UR_AI	The user **shall** have competing AI boats to race against	Shall
UR_TIME_PUNISHMENT	The user **shall** receive a time punishment for leaving their designated lane	Shall
UR_INSTRUCTIONS	The user **shall** receive instructions on how to play the game	Shall
UR_TIRED	The boat paddlers **shall** get tired as the race goes on	Shall
UR_CASUAL	The game **should** be a casual gaming experience	Should
UR_CHOOSE_BOAT	The user **should** have a selection of boats to choose from	Should
UR_COMPLETABLE	The game **should** be able to be completed within a given time frame	Should
UR_FAIR	The game **should** be balanced	Should
UR_DIFFICULT	The difficulty of each leg **should** be more than the last	Should
UR_OBSTACLES	The user **should** have objects in the river to dodge	Should
UR_OBSTACLES_DAMAGE	The users robustness **should** decrease every time an object is hit	Should
UR_PRACTICE	The first leg **should** not count towards qualifying times for the final race	Should

Functional Requirements

ID	Description	User_Requirements_ID
FR_START_LEVEL	The system shall start a leg after a boat is picked	UR_PLAYABLE
FR_END_LEVEL	The system shall end the leg when the player crosses the finish line	UR_PLAYABLE
FR_END_LEVEL_DNF	The system shall end the leg when all other opponents have crossed the finish line	UR_PLAYABLE
FR_CONTROLS	The system will allow the user to use the keyboard to control their boat	UR_RUNNABLE

FR_COLLIDE_BOAT	When two boats collide both take damage	UR_MOVE_BOAT
FR_AI_REMAINING	AI shall control the remaining boats for the user to race against	UR_AI
FR_ADDITIONAL_TIME	The time spent outside of a boat's designated lane will be added to their finishing time	UR_TIME_PUNISHMENT
FR_DISPLAY_INSTR	When the game is launched the user will receive instructions on how to play	UR_INSTRUCTIONS
FR_TIRED_INCREASE	A boat's acceleration and maneuverability will decrease as time goes on	UR_TIRED
FR_CHALLENGE	The game should not expect users to perform difficult tasks	UR_CASUAL
FR_BOAT_SELECTION	The look and stats of the boat the user selects will be the same as the boat the user controls in the race	UR_CHOOSE_BOAT
FR_UNIQUE_BOAT	No two boats in the boat selection screen should be the same	UR_CHOOSE_BOAT
FR_END_BOAT_BREAK	The system will end when the user's boat breaks	UR_COMPLETABLE
FR_CALCULATE_FINAL	The system will calculate the combined times of legs 2 and 3 for each team, with the fastest three teams going to the final	UR_COMPLETABLE
FR_END_FINAL	The system will award the player a medal after the final	UR_COMPLETABLE
FR_PODIUM	The system will display the finalists on a podium	UR_COMPLETABLE
FR_AI_RULES	AI should follow the same rules as the user does	UR_FAIR
FR_AI_IMPROVE	The AI should increase in difficulty at the end of each leg	UR_DIFFICULT
FR_OBSTACLES_INCR	The number of obstacles used in each leg should be more than the last leg	UR_DIFFICULT
FR_OBSTACLES_MOVE	Obstacles should move in a variation of patterns depending on their type	UR_OBSTACLES
FR_CALC_DAMAGE	The amount of damage taken to the user's robustness should be calculated using the type of obstacle and the speed of the colliding objects	UR_OBSTACLES_DAMAGE
FR_IGNORE	Performance in first leg will not be recorded	UR_PRACTICE

Non-Functional Requirements

ID	Description	User_Requirements_ID	Fit Criteria

NFR_RESILIENCE	A malfunction in another boat or part of the race should not affect the user's performance in the race	UR_PLAYABLE	If another boat runs out of health or gets stuck it will not affect the user
NFR_USABILITY	The game shall be in plain English with no especially difficult words	UR_PLAYABLE	All text is in clearly worded English
NFR_VISIBILITY	Any text or information displayed should be clearly visible	UR_PLAYABLE	The colour of text is in a contrasting colour to the background and in a readable size
NFR_RELIABILITY	The system shall reliably run from the start state to an end state	UR_RUNNABLE	Crash Likelihood: <1%
NFR_RESOURCES	The system shouldn't demand lots of resources	UR_RUNNABLE	<1% of resources of a standard computer
NFR_PORTABLE	There should be no issue playing the game on a different OS	UR_RUNNABLE	Game file executes correctly across all Operating Systems
NFR_RESPONSE	The system should respond to user inputs quickly	UR_MOVE_BOAT	Time to respond: <0.1s
NFR_SIMPLE	Users should be ready to play the game shortly after reading the instructions	UR_INSTRUCTIONS	1-2 minutes spent reading instructions
NFR_OPERABILITY	The game should be playable with minimal training	UR_CASUAL	90% of players should be able to win Leg 1 after just looking at the instructions
NFR_TIME	The game should be finished in under X minutes	UR_COMPLETABLE	Time taken to complete game is <=10 minutes
NFR_INTEGRITY	Stats of boats should not be changeable mid-game and presets should be fairly balanced	UR_FAIR	All boats are capable of winning
NFR_ACCURATE	Time taken for a boat to complete a leg and time taken out of its own lane will be recorded accurately	UR_FAIR	>1% margin of error
NFR_PRECISION	The system should reliably detect collisions	UR_OBSTACLES	Collisions detected 99% of the time

Constraint Requirements

Constraint	Description
Process	The game must be coded in Java
Design	Game will be playable on regular computer hardware running on popular operating systems