Elicitation, Justification and Negotiation of Requirements

We began forming our set of requirements when we initially received the brief. The process initially involved each of the group members reading the brief individually and making notes on key points we found. Once we had spent around 15 minutes each individually breaking down the brief we collaborated our ideas. We cross-referenced all of our notes keeping track of which requirements came up more than once from separate individuals. We made a document of the commonly noted requirements as well as the less obvious requirements that different individuals in the group had picked out. From this initial meeting and break down we, as a group, formed a good understanding of what the general idea of our game would be and what it would look like in its early stages, as well as creating a good early set of requirements to be added to and expanded on. This meeting also allowed us to ensure good communication between all group members and coming out of the meeting we were all on the same page in terms of the direction the requirements were going.

We took our set of base requirements to our stakeholder to further discuss what they want out of the project. During this meeting we kept in our mind the definition from Zave **[2]** which says: "Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems". This helped give us a sense importance for each individual requirement which was discussed. The meeting we had lasted around 20 minutes and we discussed our ideas in much more detail. This discussion gave us a much more precise direction in terms of requirements and allowed us to expand on our initial ideas. We also took into account what the stakeholder wanted from the project and, as a result, modified some base requirements or added new requirements altogether.

We then had a solid set of base requirements which were understood well by all group members, however, they were still very vague and needed expanding. To expand on the requirements which we had, we researched the criteria for a good set of requirements. We collectively agreed that the IEEE software requirements specification (SRS) would allow us to produce a full, detailed set of clear and concise requirements. The qualities of the SRS say that requirements must be: correct, unambiguous, complete, consistent, ranked for importance and/or stability, verifiable, modifiable and traceable **[1][3]**. Due to a page limit on the requirements document, we felt that we would not be able to fully explore all types of requirements listed under the SRS. This is why constraint requirements are omitted. To allow us to still go into detail on our requirements and to make sure they were complete, we broke the requirements down into functional and non-functional. We further broke down the non-functional requirements into reliability, maintainability, safety, quality and interface.

To present our requirements, we gave them an ID so that they could be referenced in other parts of the documentation for the project, as well as possibly in further assessments. The ID was ordered in terms of importance and/or stability **[1]** to maintain the IEEE SRS standards. The definition of requirements given by Zave **[2]** really aided us when deciding the importance and stability of a requirement. We kept in mind how much a requirement fulfilled a "real-world goal" **[2]** and we then had to consider the importance of this real-world goal.

We have chosen to present the requirements in tabular form. Upon researching how to present requirements we saw that they must be easily readable and interpretable by the stakeholder. To achieve this we felt that the tabular form would allow us to not only make our requirements readable, but it would help to keep them concise too. To keep our requirements concise, yet still explore and describe each requirement in enough detail we decided to break the table down into functional and non-functional sections. Each of these sections had a series of columns. These columns include: requirement ID to allow for requirement referencing, description to allow the stakeholder to interpret the requirement with minimal ambiguity, user story to show how the user will interact with the requirement, pass criteria to give the requirement a state of completion when it is achieved and risks, assumptions and alternatives to give visualise some issues that may arise with the requirements and give an alternative if a work-around is required.

1 - Functional Requirements - Ordered by importance and stability

ID	Description	User Story	Pass Criteria	Risks, Assumptions, Alternatives
1.1	The map must contain at least 5 colleges and 3 departments and a large body of water.	Across the body of water there will be islands where these colleges and departments can be found.	1.1.1 The map will have 5 colleges and 3 departments which are laid out to represent the real-life campus. 1.1.2 The map will contain a large body of water.	R: Making the layout of the map identical to the real life campus may cause heavy imbalances in the game. ALT: Make the layout generally similar to real life and abstract away from the exact location.
1.2	There must be a constant objective throughout the game.	At all times it must be obvious to the user what their next goal should be to avoid them getting bored and quitting due to not knowing what to do.	1.2.1 The user will be able to work towards and complete a main objective.	R: The objective must be clear and concise and if not, it could cause ambiguity from the users perspective.
1.3	There must be at least 2 playable modes. - Sailing - Combat	Sailing mode will be used to travel around the map and whenever you meet an enemy ship you will be able to engage in combat with their ship with the combat mode.	1.3.1 The player will be able to switch between the two modes when required.	R: Toggling between the two modes may add too much complexity to controls especially with multiple ships. ALT: The sailing mode and combat mode can be automatically toggled by AI.
1.4	There must be currency (gold) in the game. The currency will be used to give various upgrades and/or create new units.	The currency will be obtainable through events in the world and through combat, this currency can be spent at shops to upgrade/buy next ships for your fleet.	1.4.1 The user can complete actions which will affect their income 1.4.2 The user will be able to trade their currency in for upgrades.	N/A
1.5	The game must include a points system.	As the game progresses the user will accumulate points for certain tasks and will achieve bonus points at the end of the game for completing certain things and these points will be presented in some sort of leaderboard.	1.5.1 The user will have a points value associated to them. 1.5.2 The user's points value will increase upon completing a task. 1.5.3 The value will rank the user on a leaderboard.	N/A
1.6	The game must include a minigame related to the main game. This will not impact the main game.	At certain points in the game or when not in the main game a mini-game will be playable which will be of the same theme as the main game and will only be a short game in contrast to the main game.	1.6.1 The user will be able to play a minigame which is related to the main game however this will not impact the main game.	N/A

1.7	The game will aim to last and average of 30-60 minutes.	The gametime from when the user starts a new game to when the objective is complete will be aimed to last 30-60 minutes.	1.7.1 The game will be completed in the estimated time range of 30-60 minutes.	N/A
1.8	The game will be controlled using a keyboard and mouse.	All actions in the game will be done with a keyboard and mouse.	1.8.1 The user will be able to move their ship/fleet around using a keyboard and mouse. 1.8.2 The camera can be controlled with this method	A: The user will have a keyboard and mouse available.
1.9	The camera view will be top down.	The way that everything is viewed will be top down and you will be able to move the camera across the map.	1.9.1 There will be a top down camera view.	N/A
1.10	There will be randomly generated events and features throughout the map which will affect the players.	Events in the world can be things like a hurricane happening in certain coordinates which can damage ships and events can be finding something like a floating treasure chest which gives the player extra coins.	1.10.1 There will be random events around different locations on the map 1.10.2 These events will interact with and directly affect the players in positive or negative ways.	R: The random features and events may take away from the main objective ALT: Build the events and features into the main objective.
1.11	There will be a HUD containing game state information and a minimap	Located in a corner of your screen will be a simplified smaller version of the main map that will always be open. The HUD will also contain game state info.	1.11.1 There will be a visible minimap in the corner of the screen showing the map and positions of players/Points of interest. 1.11.2 The HUD will display game state information without cluttering the screen in a confusing way.	R: Scaling the full size map down to a minimap size may cause confusion. ALT: Use an expandable map only.
1.12	There will be an expandable, fullscreen map that shows the entire map in more detail.	The main map that the user can use which shows the full detail of the explored map and will take up a large amount of the user's screen.	1.12.1 The user will be able to open an expandable map which will take up most of the screen however it will show points on the map in more detail.	R: The map taking up the screen may cause the user to be distracted and get attacked without realising. ALT: The expandable map could be translucent.
1.13	There will be restricted visibility in the form of a 'fog of war'.	Only owned regions of the map will be visible to the user on the map. Each map will be unique as it's important to keep things hidden that may impact the game if the player knew about them in advance.	1.13.1 The player will have limited vision which will increase when the map is further explored 1.13.2 The user will be able to increase their map vision by exploring new territory.	R: Implementing a 'fog of war' may cause design difficulties when creating an AI ALT: No 'fog of war'

2 - Non-Functional Requirements - Ordered by importance and stability

P - Performance, I - Interface, M - Maintainability, RI - Reliability, S - Safety, Q - Quality, O - Operational, R- Resources

ID	Description	User Story	Pass Criteria	Risks, Assumptions, Alternatives
2.1 S	Anything that may upset the user or trigger any medical conditions will be have a warning before.	If flashing lights, excessive gore or sensitive subjects are explored in the game there will be suitable warnings to the user upon starting the game well before the user can experience anything that could affect them.	2.1.1 The game will clearly display warnings for possible medical conditions that will be affected. 2.1.2 This will happen before the game begins to ensure safety.	N/A
2.2 RI	The game will not crash or excessively slow down the machine.	The game will not be too heavy on the PC and will not require an excessive amount of processing power so that it can be played on even lower end computers.	2.2.1 The game will not crash upon long periods of play-time.	A: The hardware owned by the player will be able to cope with basic graphics and processing. ALT: Sacrifice graphic quality for frames per second.
2.3 Q/ I	There will be a storyline.	When there is contact between the player and AI there will be dialogue which gives a storyline to the actions done by the user, there will also be an introduction to the game to explain it.	2.3.1 The user will feel a sense of progression and they play through the game. 2.3.2 The user will relate to the storyline as it will follow a University of York theme.	A: The user has knowledge of the University and different departments, colleges and faculty on campus and within the University. R: If made too specialised, the user may not understand the storyline.
2.4 M	The game will be designed in a flexible manner to allow for bug fixing and updates.	As the game gets older if it were to be updated it needs to be able to be done easily and not involve massive amounts of code to be rewritten.	2.4.1 The game will be easy to maintain and update upon user feedback and during development through testing.	A: All members of the development team will be able to read, interpret and modify the code.
2.5 O	The game will be easy to set up on any desktop system	It will be a simple process to start playing the game and not involve any complicated setup.	2.5.1 The game will be easy to install on many different machines and systems.	N/A
2.6 P/ R	The game runs smoothly from start to finish. The game will not be resource intensive. Code will be optimised.	The game will not freeze or lag due to too many entities for the hardware and inefficient, unoptimised code as this will ruin the experience for the player if it happens regularly.	2.6.1 The game will aim to run at around 60 frames per second. 2.6.2 The game will not drop in frames over a long time period.	N/A
2.7 Q	The game will be aesthetically pleasing through graphics and animations.	Models, interfaces and designs will be in a good looking style and well defined.	2.7.1 The game will be appealing to the player as well as clear and concise to allow the user to know what is going on.	R: The player may find the art style cluttered which may add confusion if multiple things occur. ALT: Reduce graphical features.

Bibliography

[1] : Requirements Classification - https://www.tutorialspoint.com/software_testing_dictionary/software_requirement_specification.htm

[2]: Zave, P. (1997). Classification of Research Efforts in Requirements Engineering. ACM Computing Surveys, 29(4): 315-321. - https://www.cs.toronto.edu/~sme/papers/2000/ICSE2000.pdf

[3] : IEEE Software Requirements Specification - https://ieeexplore.ieee.org/document/741940