Requirements

Stakeholders: Richard Paige, University of York Communications Office
Team: Barney Morgan, Cameron Smith, Harry Berge, Jake Phillips, Matthew Wilkie, Rob Weddell

Upon researching, it was decided that the IEEE Software Requirements Specification (SRS) [1] proved the most effective method of specifying requirements. This method lays out both functional and non-functional requirements and includes a set of use cases which explain the user interactions provided by the software. The rigorous assessment of requirements required by the SRS ensured minimal redesign later in the project and ensured a realistic basis on which we could estimate schedules, risks and planning [2]. However, it was concluded that it would not be necessary to strictly follow some aspects of the SRS given that the scope of the project was relatively small and some aspects of the guidelines are overly extensive. As such, it was decided that only the Introduction and Functional/Non-Functional sections of the SRS were needed.

The next step was to develop precise requirements through a rapid applied prototyping [3] method. This method aims to rapidly cycle through the four requirement gathering phases of elicitation, analysis, specification and validation [4] multiple times in rapid succession. This method enabled the team to quickly make decisions on conflicting requirements whilst getting ever more precise requirements with every cycle.

The team began by reading the brief and noting down an initial brainstorm of ideas *(elicitation)*. When a conflict was met, both ideas would be noted down equally to be decided later by a third party *(analysis)*. This ensured that all team members ideas were equally valued and encouraged participation from all team members. Following this, an initial game proposal was created *(specification)*. Although riddled with flaws, vague requirements, and conflicting ideas, the proposal gave the group something to work from and rapidly make changes to. It also gave the team something tangible to present to the stakeholders and the target audience in order to gauge their opinion *(validation)*.

The proposal was then offered to the stakeholder, Richard Paige, in the form of a Q&A for validation *(2nd round elicitation)* [5]. The answers were then analysed, discussed, and built on. From this, a basic agreement on the product's key features and requirements were specified and appropriate changes were made to the initial proposal. This simultaneously solved any prior conflicts and enabled us to gain the insight of our client. However, it was decided that some requirements given by the stakeholder were still too vague. To deal with this, a user survey was designed and distributed among the cohort and stakeholders in order to establish a rough idea of what the target audience would like [6]. Following the user survey, the team was able to spot trends by analysing the results to the survey and make amends to the remaining few vague user requirements. This process was repeated one more time, validating with the client to ensure the finalised specification was acceptable.

Lastly, as mentioned in the SRS, multiple use-cases [7] were designed to emulate specific user interactions with the proposed system. This ensured focus on one specific usage aspect at a time by removing all previous assumptions of the requirements. It enabled the team to view the project as if the 'system was built first and foremost for its users' [7, p. 92]. This enabled the team to envision alternative outcomes that would not have otherwise been considered, and as such, relevant requirements to deal with them. For example, the first use case [8] highlighted the need for experienced players to skip the tutorial and instantly begin the game (req 2.17).

Eventually, a final set of requirements was established with the information gained from the stakeholder interviews, client survey, use cases, and rapid applied prototyping. It was ensured that these requirements were kept as simple and clear as possible, in order to ensure that no requirements are misunderstood later in development.

Requirements

1. Introduction

1.1 Purpose -

The purpose of this document is to completely and thoroughly specify the requirements of the SEPR 2018 project to build a pirate game.

1.2 Intended audience -

This project will be used by customers who will be staff or students from the University of York Computer Science department. We also anticipate that this game will be used by The University of York Communications Office as part of open day demonstrations.

1.3 Scope -

The game is being developed by a small team of developers with little previous software engineering knowledge. As such, many ambitious features will most likely be unable to be implemented

1.4 Definitions, acronyms, and abbreviations -

1.4.1 - Functional requirement - *specifies a function that a system or system component must be able to perform - Typically core features that are imperative to functionality*

1.4.2 - Non-Functional requirement - *any other requirement than functional requirements. Categorised into data requirements (D), constraints (C), performance (P) and quality (Q) requirements.*

2. Functional requirements

ID.	Description	User interaction	Risks, Alternatives and Assumptions
2.1	The game must be set in a flooded world taken over by pirates with a alternative University of York Campus as the main scene.	The player launches the game and recognises the Campus. They understand the pirate theme and style of the game from the design choices made	**Assumption:** A flooded university taken over by pirates will make a good setting. **Risks:** Those not familiar with The University York may not be able to relate to the setting. Also the game may be limited by the setting.. **Alternatives:** We could still have the game based on the university but make it more general to all universities to increase the relatability.
2.2	Ships must be the only manner of transport to transverse the open world map using the keyboard. ~~(WASD - configurable)~~	The player is free to move on the map as they wish. (other than preventing them from leaving the map or accessing locked areas). The ships will accelerate with W, decelerate with S, and turn left or right with A, D.	**Assumptions:** Assumes that all players have a keyboard and mouse. **Risk:** Slight risk that mouse and keyboard are too confusing. **Alternative:** Using external controller (gamepad).
2.3	~~Must be able to switch between sailing mode and combat mode. In combat mode, the player must be able to attack enemy (NPC) ships.~~	When approaching an enemy, the screen will zoom and weapons will now be available for use (battle mode). If all nearby enemies are killed or the player leaves their view, the screen returns to normal (sailing mode).	**Assumption:** By using two game modes it will help make the game intuitive for the player. **Risk:** Implementing two game modes may be time consuming or difficult. **Alternative:** We could use separate 'battle' maps.

	The player should be able to use both sailing and combat modes		
2.5	Must be able to conquer other colleges (at least 5) and raid departments (at least 3).	As part of the storyline, the player will progress through all colleges - working their way up to defeating the leader of each college before recruiting them to their party	**Assumption:** The game will work better as a open-world roguelike. **Risks:** Team member might not be familiar with the game tipe we decided to create. **Alternatives:** Instead we could create a linear game.
2.6	~~The game should require an element of skill.~~ The game should encourage the players to learn the mechanics by rewarding good gameplay and punishing bad gameplay.	The player will lose all progress (i.e. start over) if they die. The game will have less randomised mechanics so that skill can be developed rather than relying on luck.	**Assumption:** Games require challenges opposed with skill and tactics to remain interesting. **Risks:** When a player first plays the game they may find it difficult and it may be too easy for repetitive players. **Alternatives:** We could implement a different for of challenge such as puzzle based, fetch or luck based.
2.7	Gameplay should last between 15 and 60 minutes.	The game can be partially completed by ignoring side quests, skipping dialogue and avoiding roaming enemies in around 15 minutes to complete. Full completion could take up to 60 minutes.	**Assumption:** A faster game will work well for what we want to implement. **Risks:** It will be hard to tell a story in that amount of time and to cause attachment between the player and ingame elements. **Alternatives:** Have a longer running game to boost these aspects.
2.8	There should be encounters with non-pirate NPCs	There will be an opportunity to encounter non-pirate NPCs at colleges and departments.	**Assumption:** It would be unrealistic to imagine a world containing only pirates so we will implement non-pirate NPCs. **Risks:** We create a world that feels unrealistic to the point where it breaks immersion. **Alternatives:** We create a world with varying degrees of pirate NPCs.
2.9	~~*A weather system which affects movement*~~	~~Environmental effects such as wind, water currents and storms will affect how the ship moves.~~	~~***Stretch requirement:** This feature will be added if there is enough time.*~~
2.10	Players should gain XP from combat, traversing bad weather, and quests.	After defeating a ship in combat, the player will receive XP in which they can level up their ship or character. Bad weather will also provide a	**Assumption:** Having experience as a limiter to gaining abilities will be a good way controlling progression through the game. **Risk:** We don't want to overwhelm players with a skill tree. **Alternative:** Have linear skill trees.

		minuscule passive XP gain. Quests will provide a larger sum of XP.	
2.11	Players should accumulate gold from combat and exploration.	After every battle, the player will be rewarded with gold and items. There will also be opportunities to earn gold by completing quests or exploring areas of the map containing hidden treasure	**Assumption:** A player who is playing a pirate will want to collect gold. **Risks:** We might bog down the player in an economy system. **Alternatives:** We don't have them collect gold to buy things instead they could get points instead automatically.
2.12	Each gameplay should have an objective (e.g., defeat the Chief Pirate of James College). The objective should not be immediately achievable (i.e., there should be tasks that need to be completed first).	Whilst the game is open world and the player can do as they please in the map, certain areas and equipment will only be unlockable by progressing through the main storyline. As the story progresses, the enemies get stronger as the player (presumably) gets stronger.	**Assumptions:** Games and stories need a build up as so we will have a number of quests between the tutorial and end boss. **Risks:** We need to find a balance between giving the player lots of options and not swamping them with quests. **Alternatives:** We have a open quest system where completing enough of them ends that game.
2.13	There should be a system in place to spend gold in order to upgrade/repair your ship.	The player can visit various shops located around the map offering ship repair and the ability to purchase new items (and possibly sell obtained items)	**Assumptions:** A store will add a aspect of skill to the game where players can direct their progression. **Risks:** We don't want to derail the players progression and make them too powerful. **Alternatives:** Have a linear gold upgrade system.
2.14	There should be a minigame separate from the main game	There will be a minigame playable at allied departments where you can gamble with your earned gold.	**Assumption:** A minigame adds an extra layer to the game, keeping the gameplay interesting and implementing the minigame as a gambling game keeps it in theme with the rest of the game.
2.15	The game should include a world map.	The player can press a keyboard button to open up a window displaying the entire map.	**Assumption:** A world map will help the player navigate our game. **Risks:** It may not be necessary for a roguelike game. **Alternatives:** Not having a world map.
2.16	The player should be able to gather items.	Items are received from quests and exploration which boost the player's stats (such as speed, damage). These items are separate from the player's skills.	**Assumptions:** Collecting item will be a way for us to manage progression and tell a story. **Risk:** We'll need to implement a lot of different items. **Alternatives:** Not having items

2.17	There should be an integrated tutorial at the beginning of the game. This tutorial should be able to be skipped	The player will automatically enter a tutorial quest when the game starts which will explain basic movement, combat and game mechanics. This quest can be skipped for advanced players.	**Assumptions:** A tutorial will be needed to teach the players how to play the game. **Risk:** The tutorial may not cover the whole rules. **Alternatives:** Have an accessible tutorial through the menu.
2.18	There should be real ship based physics.	The ship cannot be turned immediately and is subject to realistic physics.	**Assumptions:** It will break immersion if we don't have ship physics. **Risk:** The game can't be too realistic, real ships have large turning arcs and are relatively slow. **Alternatives:** We can twist the physics affecting the ship so that is still feels and acts like a ship but is faster with smaller turning circles.
2.19	There should be a realistic AI controlling the ships	The ships should follow and shoot at the player to the standard of a regular user	**Assumptions:** The programmers can code a comprehensive AI **Risk:** the AI is too good and makes the game too hard, or too bad and makes the game too easy **Alternatives:** Ships move randomly
2.20	Add a new type of crew member, who can be added to a player's ship after achieving an objective, which grants the player's ship special abilities (e.g., faster movement, stronger attacks)	The user should unlock new crew members upon defeating a college boss. Obtaining a crew member should unlock a new form of attack for the ship.	**Assumptions:** The crew members will add a sense of progression to the game **Risk:** The new weapons are unbalanced and make the game too easy/hard. **Alternatives:** Make the crew members grant different abilities such as increased ship speed
2.21	Add a new type of natural obstacle to the lake, e.g., a whirlpool, a typhoon, a giant sea serpent. The obstacle should appear randomly and, ideally, should be something that can only be avoided or endured, instead of defeated (e.g., it causes damage, or delays progress).	A random kraken NPC should appear whilst traversing the map and attack the user. The user should not be able to damage this NPC.	**Assumptions:** The programmers are able to implement a random spawn **Risk:** The random spawn makes the game too hard for players to proceed and blocks critical paths **Alternatives:** Implement an AI to force spawn the NPC semi randomly to ensure that it only appears in specific locations which do not majorly impact gameplay

3. Non-functional requirements

Categorised into data requirements (D), constraints (C), performance (P) and quality (Q) requirements.

ID.	Description	User interaction	Risks, Alternatives and Assumptions
3.1 (P)	The game must run well on computers in the Computer Science department.	The game should run smoothly at all times (no stuttering or crashes).	**Resolution:** We should resolve stuttering by optimizing the game where required (view culling?).
3.2 (Q)	The game should be aesthetically pleasing with all on-screen elements clear.	The game will have well-made, high-resolution assets.	**Resolution:** We will consult stakeholder over interface design. **Risk:** Stakeholders idea for interface might not be the most user-friendly.
3.3 (Q)	~~The code should be written clearly in order to enable a smooth transition to new development teams~~ The code will we written in accordance with Google's Java Code Style Guide [9].	n/a	**Assumption:** the team that takes over our project is capable of following coding conventions. **Risk:** Too much time spent on readability rather than functionality could result in a lower quality end product
3.4 (Q)	The game should be fit to be used as an advertisement by the university	The game will have a seperate mode which plays through a specific part of the game without any context or pre-requisite.	**Risk:** The pirate aspect of the game may be seen to misrepresent the university.

References

[1] IEEE Specification for Software/System Engineering [Online].
Available: https://ieeexplore.ieee.org/document/6146379 [Accessed 29 Oct. 2018]

[2] J. Karlsson and K. Ryan, 'A Cost–Value Approach for Prioritizing Requirements', IEEE Software, Vol. 4, Issue 5, p. 67-74, 1997. [Online] Available: http://www.robertfeldt.net/courses/reqeng/papers/karlsson_1997_cost_value_prioritization_of_requirements.pdf [Accessed 29 Oct. 2018]

[3] B. Bähr, 'Prototyping Requirements'. [Online] Available: https://link.springer.com/chapter/10.1007/978-3-319-53210-3_3 [Accessed 29 Oct. 2018]

[4] D. Kolovos, ' Introduction to Requirements Engineering', The University of York, 2018. [Online] Available: https://vle.york.ac.uk/bbcswebdav/pid-2846228-dt-content-rid-7513532_2/courses/Y2018-006404/Requirements.pdf [Accessed 29 Oct. 2018]

[5] Element of SEPRise!, Client Interview Record. [Online]
Available: https://sepr4.github.io/web/submission/assessment1/requirements/ClientInterviewRecord.pdf [Accessed: 5 Nov. 2018]

[6] Element of SEPRise!, User Survey Results. [Online]
Available: https://sepr4.github.io/web/submission/assessment1/requirements/UserSurveyResults.pdf [Accessed: 5 Nov. 2018]

[7] J. Lee and N. Xue, 'Analyzing user requirements by use cases: a goal-driven approach', IEEE Software, Vol. 16, Issue 4, p. 92-101, 1999. [Online]

Available: https://pdfs.semanticscholar.org/cfcb/5f116c53732940c9fa210d92eb33d9163c10.pdf [Accessed 29 Oct. 2018]
[8] Element of SEPRise!, Use Cases. [Online]
Available: https://sepr4.github.io/web/submission/assessment1/requirements/UseCases.pdf [Accessed 6 Nov. 2018]
[9] Google Java Style [Online] Available: https://google.github.io/styleguide/javaguide.html [Accessed 5 Dec. 2018]