Requirements

Requirements Elicitation, Negotiation, and Presentation

First, we met to discuss the basic requirements and mechanics of our game. These were decided in reference to the scenario brief provided which was ambiguous. Thus our initial requirements were also ambiguous. Once we had these requirements we met with the stakeholders to discuss them. This meeting allowed us to confirm the basic functionality of the game before going into too much detail so we didn't have to change many requirements later in the development process.

We did research into what makes a good requirement and methods for extracting them. As a result, we made sure that all our requirements were correct, unambiguous, consistent, ranked for importance, verifiable, modifiable, and traceable [1]. The main methods we used were interviewing/meetings with stakeholders, meetings between the development team, prototyping, and user stories.

After our first meeting with the stakeholders, we began to specify our requirements. Our team met multiple times to agree upon more detailed requirements. We started creating some concept art and paper prototypes of the different screens in the game [3] which itself inferred some requirements and was useful to spark discussions in meetings with stakeholders. After this we created some simple user stories to outline some of the basic functionality of the system [4], such as navigating menus. Thinking about what users might want to do with our system within context gave us more ideas for requirements.

Finally, we decided to interview the stakeholders again to verify our more specific requirements before presenting them formally. In each meeting with stakeholders, we took rough notes and then wrote them up [5] for later reference. Once we had our final requirements presented formally we had a group discussion where we went through each requirement separately looking for any inconsistency or conflicts.

Our statement of requirements is based loosely on the software requirements specification (SRS) template [1]. This template is designed to work with large projects and as a result goes into a lot of detail, with many subsections. Due to the size of our project, some of the sections outlined in the template are not necessary and have been omitted. We formatted our requirements in a table to make them easier to navigate. Our table has an additional column for fit criteria to make sure our requirements were testable [2]. The requirements table is split up into an external interface, performance, function, and non-functional requirements and then ordered by importance within each group of requirements. Keywords **must**, **should**, and **could** highlight the importance of each requirement.

Requirements Table

Scope

The software being developed is a small zombie apocalypse-themed game set in the University of York. It should be playable at University open days and UCAS events. The game will require you to clear out various stages of zombies before progressing to the next stage and furthering the story.

User Characteristics

Typical users will be college students visiting for open days and students within our cohort at the University. So the typical user will be between the ages of 17-21. These students will be either studying Computer Science or otherwise interested in the degree so we can assume they have good technical skills but to make it accessible to all it should be easy to use.

Document Conventions

P (performance requirements), E (external interface requirements), F (functional requirements), and N (non-functional requirements).

Colour Key: Requirement Removed for Assessment 3 Requirements Updated for Assessment 3

ID	Requirement	Fit criteria	Rationale and Assumptions	Risks and Alternatives
P1	The game must run on Windows 10 in Java.	P1.1 - The computer boots into Windows 10. P1.2 - Java is installed on the computer. P1.3 - The source code is written in Java.	All the computers in the computer science building have Windows 10 and Java installed.	
P2	The game must run smoothly on the university computers.	P2.1 - The game runs at a minimum of 30 frames per second at all times. P2.2 - The game responds to the user to input within 25ms.	The computers are powerful enough to achieve this.	The game is more difficult to run than anticipated. Simplify graphics and animations.
E1	The user must be able to interact with the system using an input system available to university computers.	E1.1 - The user can navigate the menus. E1.2 - User is able to move their character.	All university computers have both a keyboard and mouse already plugged in.	A learning curve for using the system. Include tutorial missions.
E2	The system must provide feedback to the user.	E2.1 - There must be some visual output from the system.	All university computers are connected to a monitor.	
F1	The game must be split up into six locations (different stages) which each have a few waves of zombies.	F1.1 - The game returns you to the stage select screen once you complete a stage. F1.2 - Zombies are spawned at the start of each wave.	"There must be at least six locations in the game" (taken from the brief).	The user gets bored of one location if they spend too long there. Make the stages shorter but this could have the opposite effect of the game being too quick. [7]
F2	The game must get progressively more difficult.	F2.1 - More zombies are spawned in later waves and stages. F2.2 - More difficult zombie types are spawned at later waves and stages.	The user gets better and has access to power-ups and better weapons so the game needs to compensate for this. "The difficulty of the game should increase with each successful location visit" (taken from the brief).	The game gets too difficult to complete. Don't increase the game difficulty as much. This could make the game too easy and boring. We need to find a balance. [6]
F3	There must be three different player types the user can choose to play as with different abilities.	F3.1 - The different player types have different stats. e.g. run speed, hit points F3.2 - The different player types have	"There must be at least three different types of player with different attributes" (taken from the brief).	The game could become unbalanced. [6] Reduce how different the player types

		special abilities. e.g weapons, armour		are., but that could make the player types redundant.
F4	There must be at least 3 zombie types (based on societies) with different abilities.	F4.1 - The different zombie types have different stats. e.g. run speed, hit points F4.2 - The different zombie types have special abilities. e.g weapons, armour F4.3 - Each zombie type must reference a university society.	There are a lot of different societies in the university. The need for at least three zombie types was discussed in our second meeting with stakeholders [5].	Can take a lot of time to design as the zombie types must be clearly distinct from each other.
F5	There must be a mini-game, completely different from the main game.	F5.1 - The mini-game has a different objective to the main game. F5.2 - It is playable from the main menu.	“There should be a mini-game, completely separate from the main game” (taken from the brief).	Takes the user out of the game ruining immersion. [8] It should make sense in the context of the game.
F6	There must be five different power-ups which are sometimes dropped when a wave is completed.	F6.1 - One power-up is dropped at the end of every wave. F6.2 - When a power-up is dropped it is selected randomly from all the power-ups.	“There must be at least five different power-ups” (taken from the brief).	The power-ups make the game too easy. Change the frequency of the power-ups being dropped.
F7	There must be two bosses. One boss half way through the game and one at the end.	F7.1 - The third and sixth stage finish with a boss.	“There should be at least two big bad bosses” (taken from the brief).	Producing fun but challenging bosses can be difficult.
F8	The game must be able to be saved and then reloaded at any time.	F8.1 - The game state is encoded into a text file to be stored in a plain text file. F8.2 - Loading a game save returns the game to the exact state it was in when it was saved.	The university computers have storage. This allows the user to complete the game without having to do it all in one go.	Could make it easy to cheat by editing the save file.
F9	The zombies must seek out the player and do damage when they are within range.	F9.1 - The player loses a number of hit points depending on the zombie type. F9.2 - The player loses the number hit points every second the zombie is within a distance of 20 from the player.		AI controlled units can take long to design. May cause multiple bugs to pop up due to unexpected movement of the zombies.
F10	The player should do damage to a	F10.1 - The zombie loses a number		Collision detection can take time to

	zombie when they are in range and the user attacks in the correction direction.	of hit points depending on the weapon the player is using. F10.2 - The zombie loses a number of hit points when the user clicks in the direction of the zombie as long as it is within the range of the weapon. Range represented as a sector.		implement and if not implemented correctly, can cause bugs or ruin the game experience.
F11	There must be a Points System to show the player's progress.	F11.1 - The player gains points for avoiding zombies for 10 seconds. F11.2 - The player gains points for killing zombies or bosses. F11.3 - The player gains points for reaching a designated "safe zone". F11.4 - The player gains points for finishing the minigame.	"Points are accumulated for reaching a safe location, and for avoiding being touched by a zombie over a period of time." (taken from the brief)	Could cheat the points system s by editing the save file.
F12	There must be a safe location within the different 5 locations	F12.1 - There is a set area where no zombies or bosses can spawn.. F12.2 - There is a set area where no zombies or bosses may harm the player. F12.3 - Once, the player enters the safe area, 1000 points are obtained.	"Points are accumulated for reaching a safe location" (taken from the brief"	Players could remain or get trapped in the safe location and not progress through the remainder of the game.
N1	The game must be easy to learn to play.	N1.1 - There is a controls option in the user manual. N1.2 - The game starts with a tutorial mission.	The brief explains the game should be playable at open days and UCAS days. These would typically be short sessions so the game must be able to pick up easily.	Users haven't played a game with similar controls before. Include the very basic controls in the tutorial mission. Include the option to skip so the user doesn't get bored.
N2	The user must clear all stages and bosses in order to complete the game.	N2.1 - Once a stage is completed the next stage is available to play. N2.2 - Once the final stage is completed the game is completed.	"The game is won when the player visits each location at least once and the big bad bosses have been defeated" (taken from the brief). Since our game will use locations as stages we decided that each all locations must also be cleared.	
N3	The different zombie types and player sprites should all be distinguishable from each other.	N3.1 - All sprites are different in design. They have different colours and features.	Our paper prototypes highlighted this issue [3].	The screen could look cluttered. Include a visual aid to point out user's player sprite.

N4	The game should guide the user through the story.	N4.1 - There are text prompts to give the user story information.	Our first meeting with the stakeholders made clear the need for a unique selling point. We had an idea for an interesting story which the stakeholders agreed to in our second meeting [5].	Too much help can make the game too easy or ruin the game experience for the user. [8] Making set guides part of the world environment (such as signposts) [8]
N6	The game could have an 8-bit pixel-art aesthetic.		8-bit pixel-art graphics will restrict the amount of gore making sure it is suitable for open days [5].	Producing the graphics takes time.

References

[1] IEEE Software Engineering Standards Committee, “IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications”, October 20, 1998.

[2] "A Fit Criterion Is A Test", *www-ist.massey.ac.nz* [Online]. Available: http://www-ist.massey.ac.nz/plyons/158.360/essays/3b%20Volere_Templates/A%20Fit%20Criterion%20Is%20A%20Test.htm [Accessed: 29- Oct- 2018].

[3] “Geese Lightning Prototypes” [Online]. Available: https://drive.google.com/a/york.ac.uk/file/d/1Uy4ibuuzl1YmKniKU0XVSeuUqYk8AwYZ/view?usp=sharing [Accessed: 29- Oct- 2018].

[4] “Geese Lightning User Scenarios” [Online]. Available: https://drive.google.com/a/york.ac.uk/file/d/1W9lrv_nXuLEZJZ4EsMifbz5BRRyQK3P_/view?usp=sharing [Accessed: 29- Oct- 2018].

[5] “Geese Lightning Stakeholder Meetings” [Online]. Available: https://drive.google.com/a/york.ac.uk/file/d/1E_7OdAomfFnTmGmsMgkiHfaOYjVK1sQ3/view?usp=sharing [Accessed: 30- Oct- 2018].

[6] Harvey Smith, “Designing Enemies with Distinct Functions”, *www.gamasutra.com* [Online], Available: https://www.gamasutra.com/view/feature/131735/designing_enemies_with_distinct_.php?print=1
[Accessed: 05- Nov- 2018]

[7] Mike Stout, “A Beginner’s Guide to Designing Video Game Levels”, *https://gamedevelopment.tutsplus.com* [Online], Available https://gamedevelopment.tutsplus.com/tutorials/a-beginners-guide-to-designing-video-game-levels--cms-25662
[Accessed: 05- Nov- 2018]

[8] Jamie Madigan, “Analysis: The Psychology of Immersion in Video Games”, *http://www.gamasutra.com* [Online], Available https://ubm.io/1rObYeZ
[Accessed: 05- Nov - 2018]