Requirements

We are creating a fun single player zombie game based at the University of York campus. It will be a top-down 2D game where players will collect objective items that once all together will allow the player to reach a final boss and finish the game. The game will be in a pixel art style similar to SNES or Gameboy games. Players will also be able to collect power-ups, weapons and consumables throughout the game and will also be able to play a mini-game within the game when they are at a certain location.

Elicitation and negotiation:

We identified that there were going to be two main groups of requirements, first the ones which were explicitly extracted from the brief given by the client. The second group is those which we have extracted from our game idea which we had to design within the restraints of the scenario.

When looking through the brief we started by extracting requirements that were very clearly stated by the client e.g. "there must be at least three different types of players". We then discussed and came to a collective interpretation of any incompleteness or ambiguities contained in the brief. After finishing the first draft of our requirements a meeting with our client was held to discuss what they thought of our first draft of requirements and to answer some of our questions. One of our main takeaways from this meeting is that our requirements must not be vague and must be testable, it was also pointed out that we need to add more requirements specific to our game idea. To help us get a clearer picture of what we wanted our game to look like we researched games that we enjoyed and took ideas from many of them. We have used some features in these games to craft requirements and can validate these requirements based on the success of these games.

After we finalised our requirements we took a paper prototype to discuss with the client and took them through a barebones user experience of our game. We discussed the things that we should change and the things that were good, this discussion was very helpful for validating many of our requirements.

Research for Specification and Presentation:

While researching requirements specifications we found that IEEE has a recommended practice for software requirements specifications [1]. Most of our method for making good software requirements and presenting them came from this. As a result, we gave each requirement a unique ID to make them easily referenceable. We made the requirements consistent with the assessment specification, scenario and elicited requirements. Also, we made sure that there was no inconsistency with our requirements. Furthermore, we made sure that each requirement was testable in an effective way, this means each requirement is an achievable goal. All of our requirements were stored in a modifiable way allowing them to be changed as the project evolves. We tried to reduce ambiguity in the requirements so that everyone could understand them. We also, prototyped our user interface to check that all features were accessible and visible to the player to improve our requirements for it. These processes have allowed us to make a more usable and readable requirements specification. Our requirements are presented in tables split up into Functional and Non-Functional requirements.

Functional Requirements (F):

ID	User requirement	System requirement	Justification	Alternatives (A)/ Risks (R)
1.0	Player has the option of at least 3 different class types.	At the start of a new game, display the choices for each class, save the chosen class and update the player variables dependant on this. Each class will have a different experience. They will have a clear description before selection.	This is clearly specified in the scenario document, varied classes will allow for the game to be enjoyed for longer.	(R) Risk that players won't find classes to be very unique and won't get much of a new experience when choosing a new class.
1.1	Computer science student class.	Social awkwardness ability could scare zombies away	Many users will relate to this at CS open days	
1.2	TFTV student class.	Ability to disguise as zombies and sneak past them.	Provides different gameplay	
1.3	Sports player class.	Stronger and quicker than other classes.	Provides different gameplay	
2.0	At least 5 different power-up options must be available for the player to pick up during the game. (could include weapons)	Power-ups must all have an equal chance of spawning at allocated locations. Must give players a different experience for each power up.	This is required by the client. This could allow for more gameplay variety.	
3.0	There must be a mini-game within the game.	The system must detect when the player is at a certain location to induce the mini-game. The minigame will have significantly different gameplay than the main game. Goose hunt(duck hunt clone)	This is clearly specified in the scenario document.	(R) Could be hard to make a game similar to Duck Hunt. (A) Could use a different type of minigame if this is hard to make.
4.0	There must be a minimum of 6 locations.	The system must be able to quickly render a large map and use a method for transitioning between east and west campus.	This is clearly specified in the scenario document.	(R) Could take a long time to create an open world. (A) Have a linear progression where you have a set path.
4.1	There will be two areas in the game each with 3 locations.	These areas will be Campus West and East and the player will be able to travel between the two after receiving the bus pass item. Could have all three locations in an area accessible when the player arrives. Allowing the player to gather the tools to get to the boss in any order.	6 locations are required by the client that are based around the University of York.	(R) Could take a long time to create two campuses. (A) Locations could be accessed in order, allowing for easier difficulty scaling and less time to create open areas.
4.2	Computer Science, Ron Cooke Hub, Constantine, Central Hall, Market Square/Nisa.	While outside the building it must be a static model with a door that allows the user to enter.	A familiar location could be more fun for players.	(R) Could be hard to model certain buildings in a 2D style.
4.3	Biology.		The story is based from here.	
4.8	The difficulty will scale with each location that is cleared.	After getting the required item from each location Zombies will become more aggravated. This may increase zombie speed, frequency or damage. At the start of the game, zombie frequency will be relatively low.	The difficulty is required to increase with each location cleared by the client.	(R) Difficulty could become too difficult making the game less enjoyable. (A) Each location could have harder enemies than the last.

5.0	At least two bosses included in the game.	Must implement two enemy characters that are much more difficult to beat than the regular enemies.	Bosses will provide a good challenge to the player but will be fun to beat. The client also requested them.	(A) If the player dies a lot reduce the difficulty of the boss.
5.1	Bosses should be based on lectures/university administration.	One at the end of each campus. The one at the end of the first campus will drop a bus pass allowing the player to progress to the next area. 1st boss - Colin Runciman/ Mike Freeman 2nd boss- Jeremy Jacobs/Vice-chancellor.	This will add to the enjoyment of the game for our audience as they are mostly uni of york students.	(R) Could cause offence to university staff.
6.0	The game should be in 2D and should be from a top-down perspective.	The game's camera should be a reasonable distance away from the player allowing the player to see a good area around them.	This will be possible in the time constraint. Many successful games such as Pokemon use this perspective.	(A) Could use a side on perspective.
6.1	The camera should follow the player from above.	Fix the camera to the player model.	This will make the movement understandable to a new player.	The motion of the player is equal to the camera's motion.
7.0	The user should be able to pause the game at any point.	Freeze all models and display the pause menu after a single key press.	Allows the player convenience to pause and save anywhere.	(A) Have a safe room where the player can stay to go AFK.
7.1	The user should be able to save the game at any point.	The current game state must be stored in a file that can be read from and loaded at the start.	Discussed with the client. Allows players the flexibility to leave at any point without losing progress.	(R) Save file could become corrupted or lead to an error when loaded. (A) Have an autosave feature.
7.2	Control mappings should be available for the player to find at any time either through hints at the start or in the menu.	Controls clearly shown to the player when they start a new game. Controls also similar to other games eg. WASD. The player should be able to change key binds.	Allows the game to be more accessible for use on open days. This allows a wider audience.	(A) Have the controls be viewable in the pause menu.
8.0	The player's health bar must be displayed in the top corner of the GUI. All features on the GUI should be visible so that no features are hidden from the user.	The system must always store the health bar's current state and as the player is damaged or healed, the health bar must update. Other features should update quickly so that the user always knows what is happening.	Players make different decisions based on health. From prototyping, we determined that the features need to be visible.	(A) Health could be displayed as percentage or bar. (A) anything that isn't always visible in the GUI should be visible when gameplay requires it.
8.1	The player must be safe when they start the game or after they respawn when they die.	When the player's health reaches 0 they should respawn in a safe location. The player will start in a safe room at the beginning of the game and will respawn in the closet safe room when they die.	This will prevent the game from becoming too difficult or frustrating for the player. Many successful games do this.	This will prevent the game from becoming unfair. (A) Could also make the player invulnerable after dying for a short time.
9.0	Multiple unique enemy types.	As the difficulty increases the variety of enemies with different mechanics will start to increase.	For our experience with games in the past, this will keep the game interesting.	(A)Just increasing health, damage and/or numbers in the zombie horde.
10.0	The introduction needs to be shown to the player at	Display visuals along with text describing the story behind the	Players shouldn't feel lost at the start of the	(R) Player accidentally skips the story.

	the start of a new game.	game. This will be skippable.	game. Gives players a purpose.	
11.0	Combat should be in real time.	A player with different weapons and abilities must have different damage and range.	This will hopefully create interesting gameplay.	(R) Could be harder to implement turn-based combat.

Non-Functional Requirements (NF):

ID	Requirement	System Requirement	Justification	Alternatives (A)/ Risks (R)
1.0	The map must be based on real locations in the University of York.	The game must incorporate locations from any of the campuses in the University.	This is clearly specified in the scenario document.	(R) Could be harder to model real University buildings.
2.0	The game must be able to run on a standard University PC.	OS: Windows 10 64-bit Processor: Intel Core i5-8500 Memory: 16 GB RAM Graphics: Intel UHD Graphics 630 DirectX: Version 12	The game will be used for University open days, so will run on University computers.	
3.0	It must be easy to pick up for new players and have a friendly user interface.	The controls should be easy to use and pick up, involving traditional keys for a computer game.	Players will not enjoy the game if they don't know how to play.	
4.0	The game should be up to the standard of the university to represent it at open days.	There must be clear instructions and no complex mechanics required to learn.	This is a constraint clearly specified in the scenario document.	
5.0	The game should be visually pleasing.	Game's graphics must look relatively similar with custom designs looking similar to that of any assets used.	The game will appeal to a larger audience.	(R) Useful assets may be difficult to design. (A) Design everything ourselves.
6.0	The game shouldn't crash during gameplay.	Situations where errors could occur need to be accounted for.	Crashes and bugs would reduce the enjoyment of the game and take the player out of the experience.	(A) Having low specifications for the game to run.
7.0	There must be an interesting selection of music and sound effects for the game.	We will use varied sound effects and music. The same sounds won't be repeated regularly.	Give a better feel to the game and make it more attractive.	(R) Music loops played over the game would become too repetitive and become less pleasant to listen to.

Bibliography:

[1] IEEE, 830-1998, "IEEE Recommended Practice for Software Requirements Specifications", 1998.