Requirements:

Requirement Elicitation:

We performed our requirements elicitation by reading through the assessment document and then having a brainstorming session as a group to create a list of well-defined questions that we felt were pivotal in clarifying several outstanding ambiguities. Once we had created this list, we then interviewed the customer while making a note of any and all useful information that came to light. Through this method, we managed to clear up what we felt were the majority of uncertainties.

There are various reasons why a Statement of Requirements is helpful during the design of a software, but the most major reason is to help guide the project from square one. With the direction provided by the Statement of Requirements, the project is far less likely to go off track. A secondary reason for a detailed Statement of Requirements is to have something clear to show to stockholders, so as to remove ambiguity further by receiving their confirmation about what it is they're looking for. The need for a well-defined set of requirements is supported by the 1995 CHAOS report by the Standish Group [1], which reported that 13.1% of project failures were caused by incomplete requirements, the biggest risk to projects at that time. This same report also said that there are 94 project restarts for every 100 projects [1], which truly shows the magnitude of the issue of project failure.

As stated in, the scenario in which one must deal with a large number of requirements makes it so

"having a well-understood, clearly documented structure for the whole requirements set is essential to the effective management of complexity." [2]

In order to ensure this, we have made efforts to make our Statement of Requirements as easily readable as possible. Each class of requirement is presented in a separate table which is colour-coded in order to stand out. Each row is headed by the description of the requirement, as this is the most concise human-readable way of differentiating meaningfully between one requirement and another. After this, we have the fit criterion to define when the requirement can be deemed to be fulfilled. Then we have the environmental assumptions and risks, which are always started with assumptions and finished with risks. If there are no assumptions or no risks, mention of them is omitted. Finally is the ID for use in further documentation. Each ID is given a meaningful name such as to make them more understandable when used in text.

Statement of Requirements:

Functional Requirements:

Description	Fit Criterion	Environmental Assumptions and Risks	Reference No.
Must provide a UI	All information user requires should be available through GUI	Assumes user requires all information be explicitly presented. Risk of presenting too much information at one time	Func.UI
Must provide some means for user to interact with UI	Game will be designed to take both keyboard and mouse input	Risk of system being difficult to use if user is disabled such that they cannot easily use a keyboard or mouse	Func.Input
Must contain a system that supports	The game will implement a points system within which the player gains points	Risk of there being some method of earning points too easily in the game,	Func.Points

the user gaining points	for avoiding enemies, defeating enemies, reaching safe areas and winning minigames	leading to users achieving unusually high scores	
Must support at least 3 distinct playable characters	We will implement a number of characters that differ from one another in both the values of their attributes (Strength, Speed, etc) and in appearance	Assumes that the different characters are played on separate runs of the game. Doesn't require having all characters be played at the same time. Risk of making characters too similar to one another	Func.Char
Must support at least 6 distinct areas that are identifiably from the University of York	We will have a number of different areas in the game that are both major landmarks of the university and undeniably separate places in reality	Assumes that the locations can be anything from the size of a single room to an entire building. Risk of attempting to have too many areas	Func.Area
Must contain locations which are classed as 'safe areas' which increase the difficulty of the game when reached	We will have some areas which increase the difficulty of the game and grant points when reached. These areas must be totally safe and failure of the game must be impossible in these areas.	Assumes that most places will be, by default, dangerous areas and there will only be a few safe areas. Risk of having either too few or too many safe areas, or changing difficulty too much, ruining balancing	Func.Safe
Must contain at least 1 minigame	We will implement at least 1 game within the main game that is functionally distinct from the main game and undeniably shorter	Risk of spending too much time making an over-complicated minigame.	Func.Mini
Must contain at least 2 boss enemies	Will contain at least 2 enemies in the game that are distinctly more difficult to combat than other enemies in the game and also visibly different from other enemies	Assumes that bosses must be more difficult and not just visibly different. Risk of making a boss enemy impossible to beat, making the game impossible	Func.Boss
Must contain a variety of enemies	Will contain at least 2 enemies that are not as difficult as 'boss enemies' but are still distinct from one another	Assumes that the difference must be both graphical and functional. Risk of trying to make too many varieties	Func.Vary
Must contain at least a total of 5 powerups and powerdowns	Will contain at least 5 powerups and powerdowns that are distinct from one another in what they	Assumes that anything that helps the player can be deemed a powerup and anything that hinders the player can be deemed a powerdown. Risk of	Func.Powers

	do to help or hinder the user	making a powerup that is too beneficial, making the game too easy	

Non-functional Requirements:

Description	Fit Criterion	Environmental Assumptions and Risks	Reference No.
Must be enjoyable by SEPR cohort	The game will be tested on the SEPR cohort during development to ensure both proper balancing of game difficulty and enjoyability of gameplay	Risk of relying on SEPR cohort feedback too much, leading to the game becoming generic as a result of trying to please everyone	Non.Enjoy
Must be playable by SEPR cohort and easy enough to learn to play	The game will be tested on the SEPR cohort to ensure that the game's controls are not too complicated and the UI conveys the information the player needs appropriately	Assumes SEPR cohort are all confident at using computer mice and keyboards as the mode of input. Risk of making game too basic.	Non.Play
Must be zombie-themed	The game will be shown to both the customer and the SEPR cohort regularly to ensure the requirement of a zombie theme is met	Risk of the game focusing so much on having a zombie-theme that it becomes a generic zombie game	Non.Theme
Must run at 60 frames-per-second	Game will be tested on the Computer Science department computers regularly to ensure they can run the game at 60 FPS	Assumes the computers at the Computer Science department are roughly equally powerful	Non.FPS

Constraints:

Description	Fit Criterion	Environmental Assumptions and Risks	Reference No.
Must have at least one area in the game that is suitable to be displayed to prospective students at University open days	We will have an initial tutorial area that serves as a completely 'PG' area that can be shown to any and all prospective students.	Assumes that the restrictions on this area are inappropriate humour, gore, sexual references and drug references. Risk of failing to ensure area is 'PG'.	Con.PG
Must be able to run on the University of York Computer Science Department computers	The system will be tested regularly on those computers to ensure they are capable of running the system	Assumes that the computers will be running Windows 10 during the presentation of the final product. Risk of testing of system not	Con.Run

		being thorough enough.	
Must be programmed in Java	In order to ensure this, the system will be created using the Java-based game library LibGDX	Assumes that no part of the system can use a different programming language.	Con.Java
Must be able to be sold	In order to ensure this, we will avoid the use of any open-source software or anything else that could cause licensing issues.	Assumes that, as the department won't actually sell the game, referential humour will still be acceptable in the final product.	Con.Sell

Bibliography

[1] The Standish Group, "The CHAOS Report(1994)," The Standish Group International, Inc., Boston, MA, 1995.

[2] J. Dick, E. Hull and K. Jackson, "Structuring Requirements Documents," in *Requirements Engineering*, Cham, Switzerland, Springer, 2017, pp. 94-96.