Requirements

a) Elicitation & Negotiation Justification

We used a combination of “creativity” and “survey” [1] based techniques in order to elicit our requirements. We began by brainstorming based on the information on the assessment documentation/brief. The brainstorming was documented by a moderator figure, and no comments were made initially. Afterwards, the requirements generated were analysed and refined. Once we had a base idea of what was required from the system, we organised an interview with the customer and asked a series of unleading and unbiased questions in order to clear up any areas which we were uncertain about. The results of this survey were used to generate the remaining requirements and adapt others to better fit the customer’s desire. We felt that by using a union of elicitation techniques, we would derive requirements to a high standard of completeness [1].

Below are our elicited requirements. The tabular presentation makes each requirement easily distinguishable and readable. By using unique requirement identifiers, a high level of traceability is maintained so that relationships between requirements can be comfortably understood. The unique IDs follow the format REQUIREMENTTYPE_NAME, for example FR_MAP. Traceability between requirements is integral to identifying cause and effect of failure (maintenance), accountability and impact analysis during change management [1].

Our requirements take the following structure:

- **User Requirements**:

Table consists of the ID, the requirement itself, and the priority. Priority is selected from *may, should, shall* and is used to influence task precedence later in the system development, as well as for testing. The requirements are written in nontechnical language, and map out the high-level tasks that the user may be able to carry out with the finished system.It is important to remember that our primary user-base is *open day visitors* - this is vital to the forming of requirements, for example when considering the game session length.

- **System Requirements:**

Derived from the URs and describes the operations that the system must carry out in order to achieve the URs, split into:

  - **Functional Requirements:**

  For traceability’s sake, each FR has at least one corresponding UR which it works towards. Also described are any assumptions or risks to be kept in mind when creating tasks, as well as a requirement type from *transformation, invariant, failure*. Including FR types helps to ensure thorough eliciting.

  - **Non-Functional Requirements:**

  These describe the qualities that the system must have upon completion. We considered a range of NFR topics. This range of topics helps to ensure that we are rigorous when reviewing all aspects of the prospective system.

- **Constraints:**

Describe the predefined factors which limit the global system, these are to be kept in mind during requirement elicitation and throughout development.

b) Statement of Requirements

SSON: “The game shall allow a player to save York from intruding, water-fearing aliens, by tactically positioning defensive fire trucks at key locations within the city.”

USER REQUIREMENTS		
ID	**Requirement**	**Priority**
UR_ENJOY	The system shall offer an enjoyable user experience	Shall
UR_AESTHETIC	The system GUI shall be clean and easy to look at	Shall
UR_INFO	The system shall present all necessary information to the player through the GUI	Shall
UR_ TUTORIAL	The system shall be intuitive to understand	Shall
UR_SHOOT	The system shall allow the user to attack the ET fortresses and patrols with water from the fire engines	Shall
UR_FORTSHOOT	The user’s fire engines should be attacked by the fortresses	Shall
UR_HEALTH	The system shall restore the health of any damaged fire engines that the user has returned to the fire station	Shall
UR_REFILL	The system shall restore the water level of any fire engines that the user has returned to the fire station	Shall
UR_DRIVE	The system shall allow the user to move the fire engines around the map	Shall
UR_DIFFICULTY	The difficulty of flooding the ET fortresses shall increase with time	Should
UR_END	The system shall award the user a victory or lose state when they have flooded all of the ET fortresses, or been defeated themselves, respectively.	Shall
UR_MINIGAME	The system shall offer a minigame unrelated to the main game but in the same theme *[see Minigame URs in website]*	Shall
UR_UPDATES	The system should be able to be updated in the future with new functionality	Should

	FUNCTIONAL REQUIREMENTS				
ID	**Corresponding UR Req. ID**	**Description**	**Assumptions**	**Risks**	**Req.Type**
FR_GOALS	UR_ENJOY	To make the game fun, it will have the goal of destroying fortresses	If a game is confusing it is not fun	There might be too many goals for the user to focus on and the game stops being fun	Invariant

FR_VARIATION	UR_ENJOY	To make the game fun, the fortresses will have different attacks	If a game is confusing it is not fun	The attacks might be too complicated and confuse the player	Invariant
FR_MAP	UR_ AESTHETIC	The map will be a simplified version of York city centre	If a game does not provide enough challenge then it is not fun	The map could be too simple to be fun, impacting UR_ENJOY	Invariant
FR_GUI	UR_ AESTHETIC UR_INFO	The GUI will be simple and not take up more than 25% of the screen	A game is not easy to look at if it is too busy or if information is too small	The GUI could be too small and cluttered, or too big and take up too much screen space	Invariant
FR_SHOOT	UR_SHOOT	The fire engines will shoot water in the direction of driving and the water will continue to travel for 2 seconds		The range of the fire engines might be too small if the distance is realistic, making it difficult to get into range If the water travels for too long then the map may become cluttered and impact UR_ENJOY	Transformation
FR_FORTSHOOT	UR_FORTSHOOT	The fortresses will shoot projectiles when a fire engine is in a range of 100 screen units and will continue to travel for 2 seconds		If the projectile travels for too long then the map may become cluttered and impact UR_ENJOY	Transformation
FR_REGEN	UR_HEALTH	The fire engine’s health is refilled at a specified rate of 60hp per second when it reaches the fire station	If the game is too difficult it is not fun	The fire engines’ health might regenerate too slowly, making the game too difficult and impacting UR_ENJOY	Transformation
FR_REFILL	UR_REFILL	The fire engine’s water is refilled at a specified rate of 60hp pers second when it reaches the fire station	If the game is too difficult it is not fun	The fire engines’ water might refill too slowly, making the game too difficult and impacting UR_ENJOY	Transformation
FR_CONTROLS	UR_DRIVE	The fire engine can be moved using the arrow keys on the keyboard	Keyboards are an exceedingly common peripheral	If the computer does not have a keyboard, the game cannot be played	Transformation
FR_SPE	UR_DRIVE	The fire engine will move		The fire engine might	Transform

ED		at a specified speed across the map Varied speed across the map from 0 pixels to 100 pixels depending on the map beneath the truck. Slower on green and faster on roads		move too slowly for the game to be fun	ation
FR_COLLISION	UR_DRIVE	The fire engines will not be able to drive through the fortresses or the city walls		The user may become frustrated if they cannot drive where they like	Transformation
FR_FORTRESSHEALTH	UR_ DIFFICULTY	As the game progresses, the fortresses will increase in strength linearly		If the fortresses are too strong, the game cannot be won	Transformation
FR_WIN	UR_END	The system will end the game and present a win screen if all the fortresses have reached 0 health		The system may also have other reasons to end	Invariant
FR_LOSE	UR_END	The system will end the game and present a lose screen if all the fire engines have 0 health		The system may also have other reasons to end	Invariant

	NON-FUNCTIONAL REQUIREMENTS			
ID	**Corresponding UR Req. ID**	**Description**	**Assumptions**	**Fit Criteria**
NFR_REPEATABLE	UR_END	The game should play similarly each time	It is confusing if different playthroughs of the game had different rules	The rules must not change each time the game is played
NFR_TIME	UR_DIFFICULTY	The game should be playable within a reasonable amount of time	The target demographic (open day visitors) will have 10 minutes to spare	The game must take no longer than 10 minutes to complete
NFR_MAINTAINABILITY	UR_UPDATES	The code should be easy to maintain in the future	The code is going to be maintained	There must be extensive documentation and the code kept to the specified standards should follow the Java

				naming conventions/coding style
NFR _DOCUMENTATION	UR_UPDATES	The code should be documented	Any changes made are justifiable	Extensive documentation, specifying the class structure and code standards, and explaining the code functionality
NFR _RESILIENCE	UR_INFO UR_ENJOY	The system should not crash if an error occurs	We are able to catch bugs in testing	Whenever an error occurs, it will be caught and reported in an error log
NFR _OPERABILITY	UR_TUTORIAL	The system shall be playable by a new player	Players will spend time to understand the start screen	A one page max tutorial page will be shown explaining how to play the game
NFR _USABILITY	UR_INFO UR_TUTORIAL	The game should be simple and easy to understand	A limited colour palette makes for an easier to look at game	Simple colour-scheme and gameplay which can be understood from the NFR _OPERABILITY specified tutorial
NFR_GUI	UR_ AESTHETIC UR_INFO	A large GUI may result in the game looking overcomplicated and a cluttered screen	Cluttered screens reduce enjoyability	The GUI will be simple and not take up more than 25% of the screen
NFR_GOALS	UR_ENJOY	To make the game fun, it will have the goal of destroying fortresses	Goals and achievement make games more enjoyable	The user is able to destroy fortresses
NFR_VARIATION	UR_ENJOY	To make the game fun, the fortresses will have different attacks	Variety is gameplay makes games more enjoyable	The fortresses all have unique attack styles

REFERENCES

[1] K. Pohl, *Requirements engineering*, 1st ed. Berlin: Springer, 2010.

[2] M. Prensky, *Digital game-based learning*, 1st ed. St. Paul: Paragon House, 2001.

[3] J. Dick, E. Hull and K. Jackson, *Requirements Engineering*, Cham, Switzerland: Springer, 2017,