Requirements

Cohort 3 Team 4

Kiran Kang

Hannah Rooke

Ben Slater

Abualhassan Alrady

Cassie Dalrymple

Charles MacLeod

Dash Ratcliffe

Harley Donger

Requirements Elicitation, Negotiation and Planning

To obtain our requirements, we carried out different styles of research to grasp what we needed to include in the overall functionality of the game. We began by conducting a meeting with the customer, which was necessary to get an idea of initial requisites. This allowed us to get a solid understanding of the foundation of our game, as it meant we were able to start brainstorming with a clear objective in mind.

Prior to the client meeting, we created a list of questions focused on the areas we needed clarification on, so that we could incorporate this information into our requirements later. An example of this was asking about the overall tone and aesthetic of the game, as this was not explicitly specified. This also provided an opportunity for the client to share any additional preferences they had.
When discussing these requirements, some new features that the customer had not thought about were also brought up - one example being our suggestion of a ‘lives’ system, which the client found to be an interesting idea that we were considering including.

Following our client brief, we researched certain aspects of the game, such as the game engine itself, as this would form the basis of our coding; by doing this prior to beginning the coding, we were able to become familiar with what to expect when using a new game engine.
We also looked deeper into the specific areas such as asset sourcing and technical constraints, such as compatibility, as this process helped identify new requirements relating to hardware limitations and accessibility that we otherwise may not have thought of.

After researching relevant areas, we were able to start creating the finalised list of game requirements to give ourselves a coherent target to start working towards. We already had a long list of desired functionalities to include, so decided it would be more efficient to work together to look through them and determine their importance - this helped us simplify what we were working towards.
To ensure nothing was redundant, whilst also covering all bases, we evaluated our list of requirements against a variety of categories such as gameplay, design, accessibility, etc. By doing so, we were able to make sure our requirements addressed all critical aspects of the game.

These requirements are shown in a table below, as this presentation made the information easier to interpret and provided a clear overview of each requirement. The table also included additional details which we felt were necessary to include, such as the reasoning behind each requirement and how we would verify its inclusion.

Requirements Specification (User, System, and Non-Functional Requirements)

User Requirements (UR)

ID	Requirement	Justification	Priority (using MoSCoW)	Validation criteria
UR_LICENSE	Must use an appropriate license. Licenses for game engines/assets must be written in the implementation document.	Needed to be documented so the permissions can be granted to our development team, providing a legal framework.	Must have	All game engines and assets have the correct licensing in the Impl1.pdf
UR_MOVEMENT	Player must move using W/A/S/D keys	Allows player to move around the maze map and interact with events efficiently.	Must have	Verified during testing input system - its behaviour is demonstrated in Arch1.pdf (Player Movement)
UR_TIMER	Game must have a visible five minute countdown timer.	Needed in product brief so the player must complete the game within a time limit.	Must have	Verified during testing the Timer.Java - its behaviour is demonstrated in Arch1.pdf. Timer is displayed and counts down from 5 minutes.
UR_MAZE_MAP	Game's map aesthetic must be a maze-like university displayed as a medieval theme. Must also have barriers to restrict movement for player.	Navigates the game's pathway for the player and prevents the player from going to restricted areas.	Must have	Verified during collision detection tests. This is implemented in the MapManager.Java code
UR_NEGATIVE_EVENTS	Game must have a negative event that decreases the player's score	Needed in product brief that are obstacles for the player that can decrease their score	Must have	Verified during events test. In EventManager.Java, negative event is triggered and decreases score.
UR_POSITIVE_EVENT	Game must have a negative event that increases the player's score	Needed in product brief that are obstacles for the player that can increase their score	Must have	Verified during event testing. In EventManager.Java, negative event is triggered and decreases score.
UR_HIDDEN_EVENT	Game must have a hidden event	Used to make the player slow down	Must have	Verified during the events test. In EventManager.Java - event is hidden, then triggered and makes player's movements slower
UR_EVENT_COUNTER	Game must display how many times a player has interacted with a positive, negative or hidden event	The event counter is required as it was stated in the product brief to keep track of how many events the player will interact with	Must have	Verified during the events test. In EventManager.Java - there is an Event counter UI displayed that increments by 1 when an event is interacted with.

UR_DEAN	Dean character must chase player/act like an obstacle in the game	Stated that it was needed in product brief but is not required to implement in assessment 1	Will not have	Not implemented.
UR_MENUS	Game must include main, pause, gamover and victory menus.	This is needed so the player can start, pause and end gameplay and identify their progress in the game	Must have	Verified during UI testing - game starts with main menu and gameplay is paused using ‘esc’. When player meets criteria to win/lose a victory/gameover menu is displayed.
UR_SCORE	Game must include having a score system to achieve a ‘Win’ or ‘Lose’ outcome.	Product brief stated that you win by the ‘best score’, needed so players can be affected by the events. Such as increasing/decreasing score for negative/positive events.	Could have	Verified during events test - whenever a player interacts with a positive/negative event score increases/decreases. Score is handled in MapManager.Java

Functional Requirements (FR)

ID	Requirement	Justification	Validation criteria	Links to UR
FR_TIMER	Implement a five minute timer that stops when the game ends	Makes sure the user follows the timing rule	Timer starts when player starts game in menu, when paused and resumed timer still keeps its accuracy, game will auto end when timer hits 0.	UR_TIMER
FR_MAIN_MENU	Allows pausing/resuming of the game, providing information of the current state to the user	Enables core control for the user	Menu displays depending on the game state, pause displayed when ‘esc’ pressed and then resumes back to its gameplay state.	UR_MENUS
FR_UI_SCREENS	Creates an appealing interface for the user, through use of a map	Provides a clear and easy navigation screen that allows simple gameplay	Map, timer and event counter are all displayed correctly and visibly.	UR_MAZE_MAP
FR_PLAYER_MOVEMENT	Accept keyboard input for directional movement, such as arrow keys	This uses the regular controls for a game, making it easy to play	Player moves up, down, left and right - stops moving after key has stopped being pressed	UR_MOVEMENT
	Detect collisions between player and maze features, such as walls	Meets obstacle objective, creating events for player	Stops when player tries to go through a barrier.	UR_MOVEMENT

FR_INTERACT_EVENTS	Implement one event to hinder gameplay	Achieves negative event logic	Negative event is triggered correctly and score decreases.	UR_NEGATIVE_EVENTS
	Implement one event to benefit gameplay	Achieves positive event logic	Positive event is triggered correctly and score increases.	UR_POSITIVE_EVENT
	Implement one hidden event that triggers invisibly on contact with player	Achieves hidden event logic	Hidden event remains invisible until it is triggered and then slows player down correctly	UR_HIDDEN_EVENT
	Maintain counters for events that have been triggered	Supports score and metrics display	Event counter is incremented by 1 when player interacts with an event	UR_EVENT_COUNTER
FR_SCORE	Display remaining time and event counters clearly	Provides user feedback during gameplay	Score updates on screen when player has encountered a positive/negative test.	UR_SCORE

Non-Functional Requirements (NFR)

ID	Requirement	Justification	Validation Criteria	Links to UR
NFR_LANGUAGE_VERSION	Execute using Java 17 on standard university lab computers.	Required by university and is compatible with libGDX.	Game runs perfectly on lab PCs using Java17	UR_LICENSE
NFR_HARDWARE	Game must run on a university standard lab PC.	Provides accessibility and makes sure all users can run the game on PC despite what hardware they have	Game has been tested on lab hardware	UR_MAZE_MAP, UR_TIMER
NFR_LICENSE	All third party libraries/assets should all be not copyrighted	Avoids any legal issues for distribution	All third party library/assets are listed and there is a brief explanation in the Impl1.pdf document.	UR_LICENSE
NFR_TIMER	Game must have enough interactive events/objectives so game can be played for 5 minutes minimum	Challenges the player to complete the game in a 5 minute limit. Required in product brief.	Implemented one positive, negative and hidden event which will last 5 minutes at least.	UR_TIMER