Requirements:

Team 1:

Megan Forrow
Jonathan Broster
Sophie Bailey
Alastair Johnston
Luiz Oliveria
Alina Merkulova
Tameem Abuhaqab

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 requirements. 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_LICEN SE	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_MOVE MENT	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_NEGAT IVE_EVE NT S	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_POSITI VE_EVE NT	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, gameover 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 the 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 sure 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_MOVE MENT	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 objectives, creating events for players.	Stops when player tries to go through a barrier.	UR_MOVEMENT

FR_INTERACT_EVE NTS	Implement one event to hinder gameplay. Achieves negative logic events.	Negative event is triggered correctly and score decreases.	UR_NEGATIVE_EVENT S
	Implement one event to benefit gameplay. Achieves positive logic events.	Positive event is triggered correctly and score increases.	UR_POSITIVE_EVENT
	Implement one hidden event that triggers invisibly on contact with the player. Achieves hidden logic event.	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 using gameplay.	Score updates on screen when player has encountered a positive/negative test.	UR_SCORE

Non-Functional Requirements (NFR)

ID

NFR_LANGUAGE_ VERSION

NFR_HARDWARE

NFR_LICENSE NFR_TIMER

Requirement	Justification	Validation Criteria	Links to UR
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
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

All third party libraries/assets should all be not copyrighted Game must have enough interactive events/objectives so game can be played for 5 minutes minimum	Avoids any legal issues for distribution Challenges the player to complete the game in a 5 minute limit. Required in product brief.	All third party library/assets are listed and there is a brief explanation in the Impl1.pdf document. Implemented one positive, negative and hidden event which will last 5 minutes at least.	UR_LICENSE UR_TIMER

Assessment 2 requirements table:

To ensure the requirements remained accurate and appropriate for Assessment 2, we assessed the requirements from the previous team and evaluated each requirement against the updated game functionality. Requirements that were still relevant were the same, while those that no longer reflected the game's behaviour were changed.

User requirements (UR):

Id	Requirements	Justification	Priority (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 third-party assets and libraries used are listed with their licences in the implementation report.
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	The player can move up, down, left, and right using the W/A/S/D keys during gameplay.
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	A countdown timer starts at 5:00, is visible during gameplay, and reaches 0:00 after five minutes.
UR_MAZE_MAP	The game's map aesthetic must be a maze-like university displayed as a medieval theme. Must also have	Navigates the pathway and prevents the player entering restricted areas	Must have	The player cannot move through walls or restricted areas.

	barriers to restrict movement for players.			
UR_NEGATIVE_EVENTS	Game must have a negative event that decreases the player's score	Needed in product brief – obstacles must decrease score	Must have	Negative event triggers and decreases score
UR_POSITIVE_EVENT	Game must have a positive event that increases the player's score	Needed in product brief – obstacles must increase score	Must have	Positive event triggers and increases score
UR_HIDDEN_EVENT	Game must have a hidden event	Used to make the player slow down	Must have	Hidden event slows movement when triggered
UR_EVENT_COUNTER	Game must display how many times the player has interacted with positive, negative or hidden events	Required in product brief to track event interactions	Must have	Event counter UI increments on interaction
UR_DEAN	The Dean must appear if the player fails the code-entry objective incorrectly and must chase the player.	Follows the gameplay vision discussed by the development team: Dean acts as the primary threat and adds tension, urgency, and failure conditions.	Must have	If the player enters an incorrect code, the Dean appears and chases the player. If the Dean touches the player, the game is over.
UR_MENUS	Game must include main, pause, gameover and victory menus	Needed so player can start, pause and end gameplay	Must have	The correct menu is displayed when the game starts, is paused, or when a player wins or loses.
UR_SCORE	Game must include having a score system to achieve a Win or Lose outcome	Product brief stated you win by "best score"	must have	The score is visible and changes immediately when the player triggers positive/negative events.
UR_ACHIEVEMENTS	The game must include an achievements system that unlocks when the player completes specific in-game conditions, and achievements may modify the final score after the game ends.	Adds replay value and provides meaningful goals. The product brief notes score-based outcomes, so post-game score modifiers integrate naturally into gameplay.	Could have	During testing, achievements unlock when their conditions are met and any score modification is applied correctly to the final score screen.

UR_LEADERBOARD	The game must provide a leaderboard that stores and displays the top five scores. Players must be able to enter their name when submitting a score.	Supports the product brief requirement of winning by best score, and encourages competition and replayability.	Could have	After several completed runs, the leaderboard displays the top five scores with associated player names, and entries can be added through a name-input interface.
UR_CHECKIN	The game must provide some kind of code that can be checked to trigger the dean.	Triggers the event of the dean which is a requirement as stated by the stakeholders and assessment brief.	Must have	The check in code is randomly generated each time it is shown to the player, once the code is inputted into the shrines the corresponding event will take place if it is correct or incorrect.

Functional requirements:

Id	requirements	justification	Validation Criteria	Links to UR
FR_TIMER	Implement a five-minute timer that stops when the game ends	Ensures user follows timing rule	Timer starts, pauses, and ends accurately	UR_TIMER
FR_MAIN_MENU	Allows pausing/resuming of the game and provides state information	Enables core control for the user	Menu displays based on state; pause via ESC	UR_MENUS
FR_UI_SCREENS	Creates an appealing interface using a map	Provides clear and easy navigation	Map, timer, event counter displayed correctly	UR_MAZE_MAP
FR_PLAYER_MOVEMENT	Accept keyboard input for directional movement	Standard game controls	Player moves and stops properly when keys released	UR_MOVEMENT
FR_COLLISION	Detect collisions between player and maze features	Meets obstacle objective	Player cannot move through walls	UR_MOVEMENT
FR_INTERACT_EVENTS	Implement positive, negative and hidden events	Achieves event logic	When the player interacts with an event, the score changes or movement slows down is observed immediately.	UR_NEGATIVE_EVENTS, UR_POSITIVE_EVENT, UR_HIDDEN_EVENT
FR_SCORE	The system must display the player's current score and update it when an event is triggered..	Provides gameplay feedback	Score is visible during gameplay and changes immediately after positive or negative events have been triggered.	UR_SCORE
FR_DEAN_AI	Implement Dean spawning after code entry, chasing the player using pathing or direct movement.	Required loss condition & challenge.	If the Dean touches the player, the game ends	UR_DEAN
FR_ACHIEVEMENTS	Monitor gameplay metrics and unlock achievements accordingly; apply final score modifiers if defined.	Adds optional progression elements.	Achievement triggers tested with known conditions.	UR_ACHIEVEMENTS

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 the required game engine and runtime environment.	The Game executes successfully on university lab PCs using Java17.	UR_LICENSE
NFR_HARDWARE	The 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 must have appropriate licences allowing the use in this project.	Avoids any legal issues for distribution	All third party libraries/assets are listed and there is a brief explanation in the Implementation report.	UR_LICENSE
NFR_TIMER	Game must have enough interactive events/objectives so game can be played	Challenges the player to complete the game in a 5 minute limit. Required in product brief.	Gameplay includes interactions to fill the five-minute timer.	UR_TIMER

	for 5 minutes minimum			
NFR_USABILITY	UI elements (including achievements and leaderboard name entry) must be easily readable and operable without training.	New features introduce additional UI interactions that must be clear for the player.	Test users operate achievement notifications and name entry without assistance.	UR_ACHIEVEMENTS, UR_LEADERBOARD
NFR_DATA_PERSISTENCE	Leaderboard data must be stored locally between game sessions.	Required so top 5 scores are preserved even after restarting the game.	Restarting the game still shows previously saved scores.	UR_LEADERBOARD

Changes report:

Overview of Changes Made to Requirements:

- the new features implemented by our team (Achievements, Leaderboard, Dean behaviour)

- The need for clearer and testable validation criteria

Where the original requirements already satisfied the brief, they were kept unchanged. Only requirements directly impacted by new features or changes to system behaviour were modified.

User Requirements (UR) Changes:

UR_DEAN — Updated (Major Change)

Original :

- UR_DEAN was marked as *Will not have.*
- No implementation was planned.

requirement was updated to:

- The Dean must appear after the player completes the code-entry objective and must chase the player. If the Dean catches the player, the game must end

- Priority changed from *Will not have* to Must have.

- New validation criteria included: spawning, chase behaviour, and game-over state.

justification:

The Dean provides the game's main obstacle and creates a fail state. Adding this ensures the game includes a required challenge and a clear game-over condition.

Updated section:

Requirements Table : Row UR_DEAN

Functional Requirements Table : FR_DEAN_AI (new)

UR_LEADERBOARD - Added (New Requirement):

storing top five scores and entering player names.

Justification:

The selected team’s implementation included partial leaderboard logic. To fully integrate and justify the feature, a requirement was added for traceability and validation.

Updated Section:

Requirements Table: Added row UR_LEADERBOARD

Improvements to Validation Criteria -Updated (Minor Changes)

Examples:

UR_SCORE: clarified score updates.

UR_EVENT_COUNTER: clarified behaviour on event triggers.

Justification:

The clarification ensures the requirement is specific and measurable, enabling consistent verification during testing.

Functional Requirements (FR) Changes:

FR_DEAN_AI - Added (Major Change):

Original:

No Dean AI requirement existed.

Updated Version:

New FR states the Dean must spawn after code entry, chase the player, and trigger game-over on collision.

Justification:

The Dean's AI is needed to make the character function as an active threat, giving the player a meaningful challenge and a way to lose the game.

FR_ACHIEVEMENTS — Added (New)

Original:

Not present.

Updated Version:

Added to define achievement-triggering logic and optional score modification.

Justification:

Adding this requirement ensures the achievement system is properly defined and can be checked during testing.

Non-Functional Requirements (NFR) Updates:

NFR_USABILITY - Added (New Requirement)

Original :

This requirement did not exist.

Updated Version:

Added *NFR_USABILITY* to ensure new user interface elements (achievement pop-ups and leaderboard name-entry) remain clear and accessible to players.

Justification:

This requirement ensures that all new interface elements are easy for users to read and interact with, supporting a smooth gameplay experience.

NFR_DATA_PERSISTENCE — Added (New Requirement)

Original:

Not presented.

Updated Version:

Added NFR_DATA_PERSISTENCE to support saving leaderboard entries between game sessions.

justification:

Scores must be saved so the leaderboard still shows results when the game is restarted.

Updated Section:

Non-Functional Requirements Table : Added row NFR_DATA_PERSISTENCE

No changes to existing NFRs

The original NFRs remained valid and unchanged:

- NFR_LANGUAGE_VERSION
- NFR_HARDWARE
- NFR_LICENSE
- NFR_TIMER

Justification:

These requirements remained relevant and sufficient implementation; therefore changes were not necessary.