Architecture

Team 3

3NG

Give diagrammatic representations (structural and behavioural diagrams) of the architecture of the team's product, with a brief statement of the specific languages (for instance, relevant parts of UML) and the tool(s) used to create these representations. Include a systematic justification for this architecture and describe how it was initially designed and how it evolved over the course of the project. Provide evidence of the design process followed (e.g. interim versions of architectural diagrams, CRC cards) on your team's website and link to them from your report. Relate the architecture clearly to the requirements, using your requirements referencing for identification, and consistent naming of constructs to provide traceability (22 marks, ≤ 6 pages).

Components of the architecture document:

- Structural and Behavioural diagrams, each with "brief statements of the specific languages" and the tools used to create the diagrams

- A systematic justification for the architecture diagrams and descriptions of how they were initially designed and how they evolved over the course of the project

- Evidence of the design process followed must be put on the website and linked to in the report

- The architecture needs linking to the requirements (using IDs), and consistent naming of constructs

<u>Submission Checklist</u>

- At least 1 structural view at each level (context, container, component)
- Brief statement of notations and tools that are used
- Systematic justification connected to drivers/nfrS WİTH TRADE-OFFS
- Evolution evidence linked (interim diagrams, CRC, ADRs)
- Traceability : requirements <-> architecture with consistent ID s.

1. Purpose & Modelling Approach
2. Structural Architecture
    a. Class diagram - core game structure
    b. Component diagram - high level architecture
3. Behavioural Architecture
    a. Sequence diagram
    b. State diagram
4. Justification & Design Evolution
5. Traceability to Requirements
6. Summary

Purpose & Modelling Approach

This section shows the comprehensive architecture of our game “Escape from Uni”, presenting how its structure and behaviour modelled before the implementation stage.
By creating both behavioural and structural aspects through UML diagrams, we can connect design decisions clearly, analyze dependencies early and establish a prototype that meets the functional constraints (one positive, one negative, and one hidden event plus timer and counters).
According to the Object-Oriented Modelling lecture, models are two types: descriptive and prescriptive. For *Escape from Uni* we used prescriptive modelling, because the purpose is to design the architecture of the game before the full game exists. The UML diagrams specifically show how each component will behave and interact with each other.
Descriptive modelling will become applicable after the game is implemented, to document and evaluate the final system. We used the Unified Modelling Language (UML) as standardised, object-oriented modelling notation to capture both structure and behaviour.
The following diagrams are produced:

- Class Diagram
- System Context Diagram
- Sequence Diagram
- State Diagram

These diagrams correspond to the official UML notation.They were created using draw.io
….CAN BE EXTENDED AS WE USE…

Using UML models at this early stage allows our team to:

- Certify design ideas without the full implementation
- Interact architecture evidently to stakeholders
- Make a foundation for modular development and reuse
- Reduce risk of design errors by checking the consistency between diagrams

Resources used:

- https://c4model.com/

Use Cases

We have created some use cases, to implement scenarios to work from. This gives us perspective on how the system will be used, in order to design diagrams to reflect this. These diagrams can then be used in implementation, and create an easy to follow structure.

Name: Player wants to turn sound off:
Precondition: Game is running
Main success scenario:

- 1. Player pauses the game
- 2. Player enters the game settings from pause menu
- 3. Turns off sound (switch)

Success postcondition:

- Sound turns off for the entire game (including pause menu)

Name: Player wants to start a new game.
Actors:
– **Primary Actor**: User
– **Supporting Actors**: N/A
Precondition: The menu screen is displayed to the user.
Trigger: The user has selected Start Game from the menu screen.
Main Success Scenario:

- The system loads maze environment
- The system initialises the game timer and the game score to zero
- The system transitions to the starting point of the game play.

Secondary scenarios:

- Game loading takes unusually long.
    - **Postcondition**: A loading screen with a progress indicator appears to prevent player confusion.
- The player wants to restart the game in the middle of a game.
    - **Postcondition**: A pop up screen checking if the player wants to continue? Or reset progress?

Success Postcondition: The player can begin to play the game.
Minimal Postcondition: The system remains on the main menu screen.

Name: Pause the Game
Actors:
- Primary Actor: User
Precondition: The player is currently playing the game.
Trigger: the player has clicked the pause button
Main Success Scenario:

- The system halts all gameplay activity which includes player movement, enemy behavior, and time countdown.
- The system displays a *Pause Menu* with options such as Resume, Restart, and Exit to Main Menu.
- The player remains able to navigate the pause menu.

Secondary scenarios:

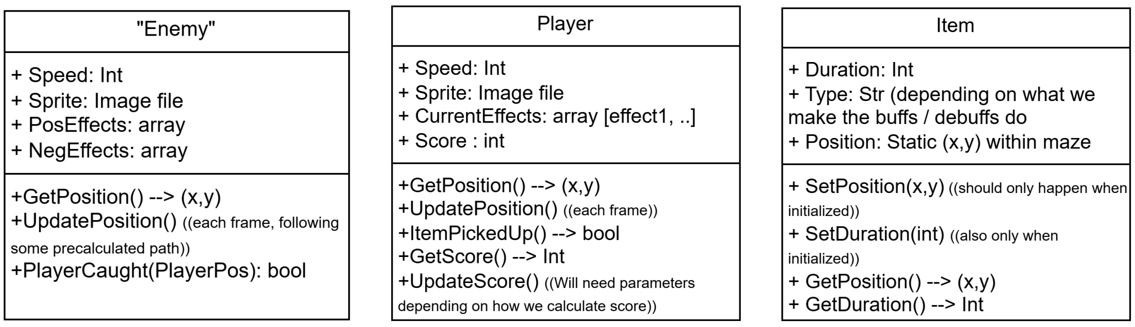
- The player selects *Resume* from the pause menu.
- The system restores normal gameplay, unfreezes the timer, and hides the pause menu.
    - **Postcondition:** Gameplay continues from exactly where it was paused; no progress is lost.

Success Postcondition: < holds for successful exit, achieves goal >
Minimal Postcondition: < holds for all exits, protects stakeholder interests >

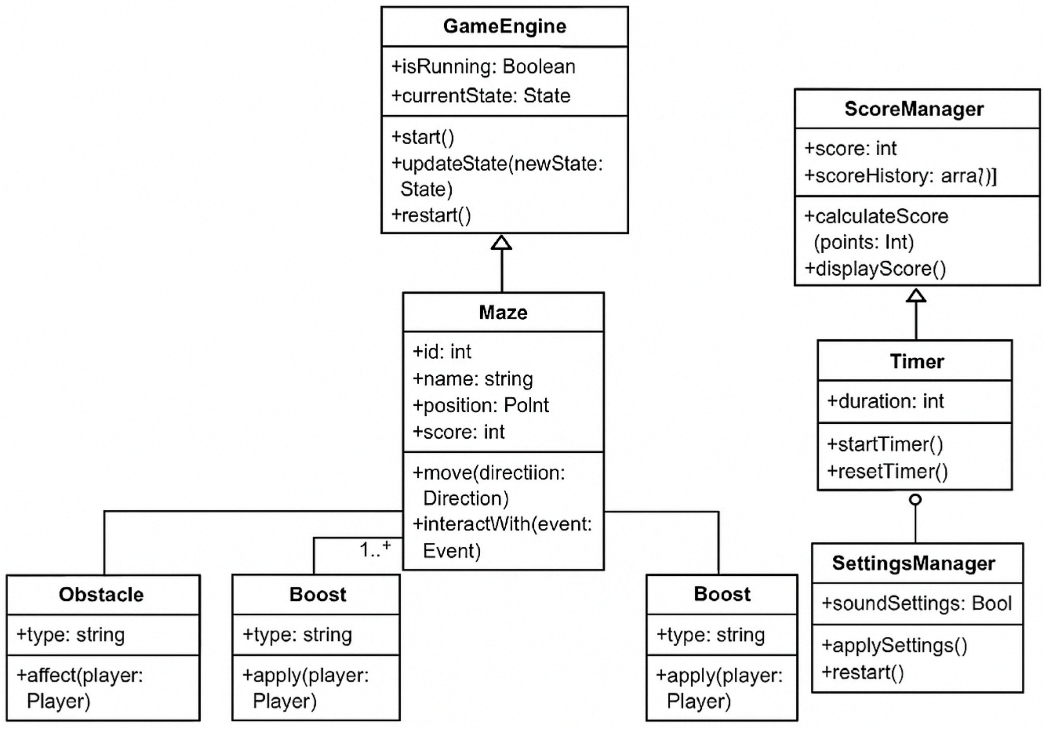
Structural Architecture

We have had to be selective of the images we show in this document, to keep within the page limit. Due to this the system context diagram is shown within this document, but the container and component diagrams are linked within the website. Descriptions of these diagrams will be presented here.

Class Diagram:


"Enemy"
+ Speed: Int
+ Sprite: Image file
+ PosEffects: array
+ NegEffects: array
+GetPosition() --> (x,y)
+UpdatePosition() ((each frame, following some precalculated path))
+PlayerCaught(PlayerPos): bool
Player
+ Speed: Int
+ Sprite: Image file
+ CurrentEffects: array [effect1, ..]
+ Score : int
+GetPosition() --> (x,y)
+UpdatePosition() ((each frame))
+ItemPickedUp() --> bool
+GetScore() --> Int
+UpdateScore() ((Will need parameters depending on how we calculate score))
Item
+ Duration: Int
+ Type: Str (depending on what we make the buffs / debuffs do
+ Position: Static (x,y) within maze
+ SetPosition(x,y) ((should only happen when initialized))
+ SetDuration(int) ((also only when initialized))
+ GetPosition() --> (x,y)
+ GetDuration() --> Int

These are different objects that will exist within the game. Enemy and player may have a parent superclass if we decide so, so that certain attributes can be inherited.
Items will be a unique class as their attributes are unique, and these exist just to apply buffs / debuffs to characters.



System context Diagram:

This diagram has been created with draw.io, and follows the C4 model standards.
It shows a big picture of the system landscape, and the systems which the game interacts with such as the menus and timer.

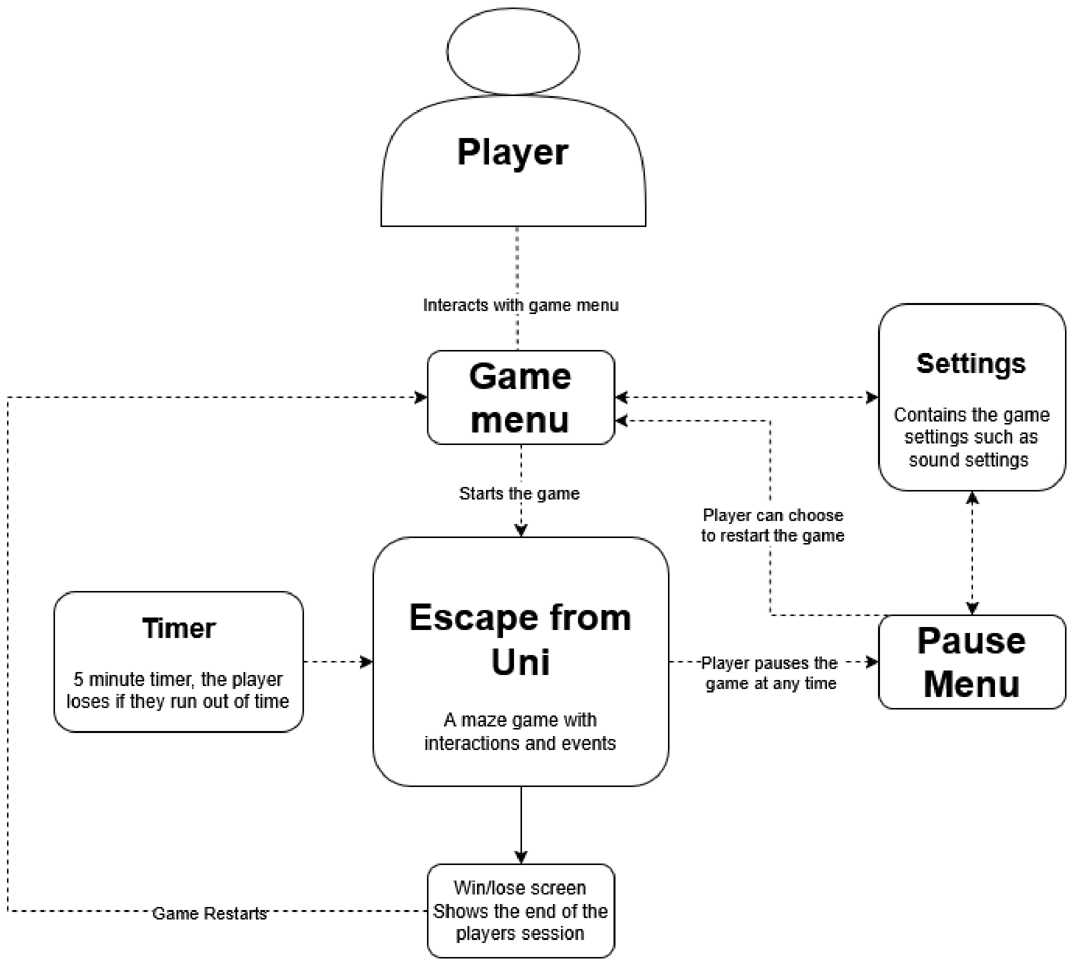

Player
Interacts with game menu
Game menu
Settings
Contains the game settings such as sound settings
Starts the game
Player can choose to restart the game
Timer
5 minute timer, the player loses if they run out of time
Escape from Uni
A maze game with interactions and events
Player pauses the game at any time
Pause Menu
Win/lose screen Shows the end of the players session
Game Restarts

Figure 1.System Context Diagram

As can be seen in *Figure 1*, the player will first interact with the game menu after loading. It should be obvious and accessible to the player to navigate the menu (NFR_UX_USABILITY), either modifying settings (NFR_SETTINGS) or loading into a new instance of the game (UR_RESTART). It is shown that the pause menu can be accessed from the game, and should be available to access at any time (UR_PAUSE_FUNCTIONALITY).

Container Diagram:

The game focused container diagram, *Figure 2 - Container Diagram,* shows a high level view of the game architecture, and the interaction between the player sprite and the elements around them, such as the map, the timer, and the game engine being used to manage and process components such as movement.

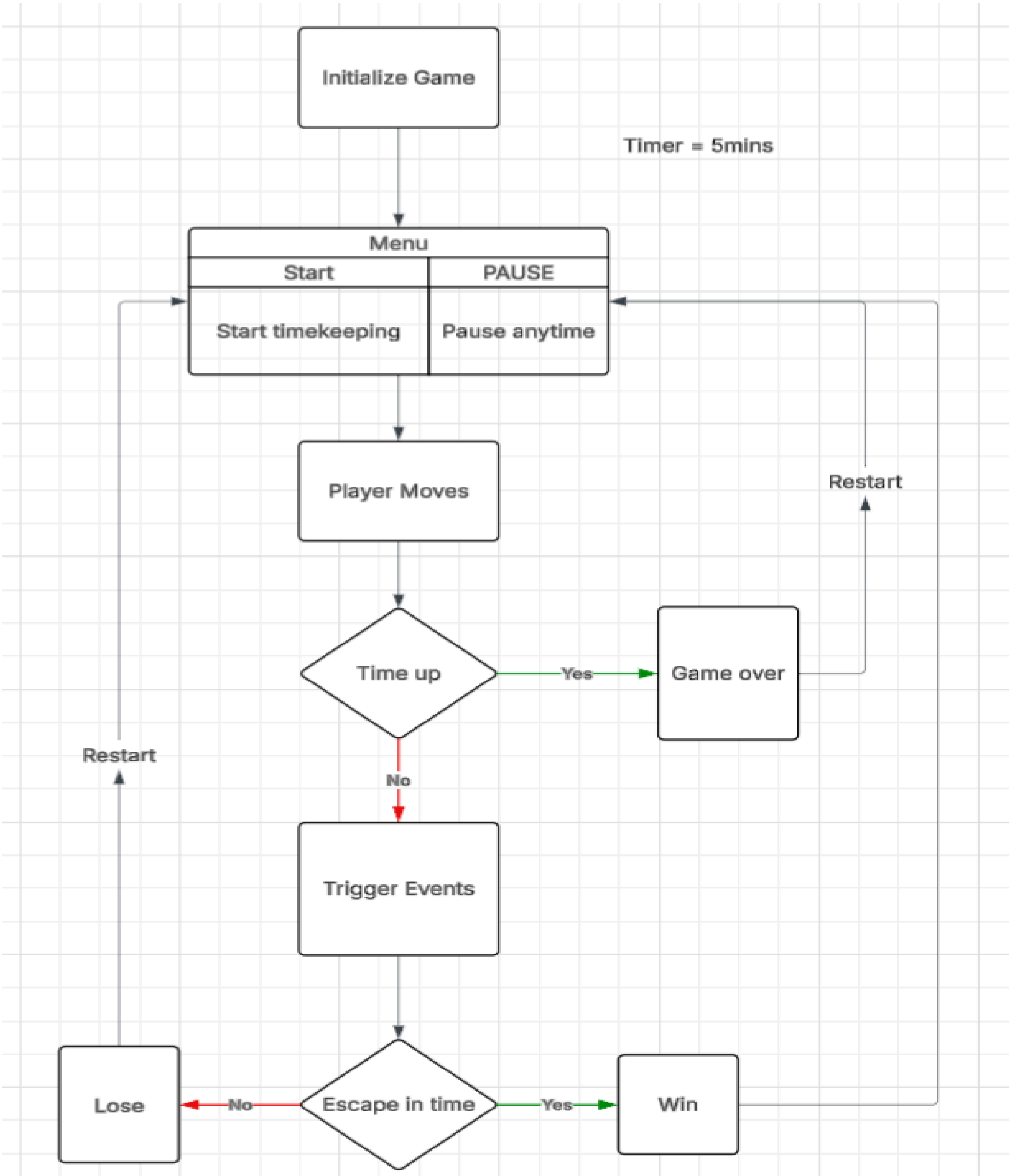
This diagram may change as the implementation of the game progresses, however at the moment, this is how the game should look based on the requirements before development.

Component Diagram:

Figure 3. Player Sprite Component Diagram
The component diagram, *Figure 3*, shows how the player will interact with the environment around it. For example, the player will move around the map - this will include collisions with areas such as walls and buildings. When exploring/traversing the map, the player will also encounter both events and objects to interact with, including positive and negative events (both visible and hidden). This fills the requirement (NFR_GAME_OBSTACLES). The player sprite will also need to display animations, for example: running, walking, picking items up. This will create a more immersive experience for the player, fulfilling UR_PLAYABILITY.

State Diagram:

The state diagram describes the player's reaction in different states. The timer will start counting down when the player starts the game. If the time is up in the middle of the game, the player will return to the menu. When the player breaks through the trap and escapes the maze, if the time is not up, he will win, otherwise he will lose and finally return to the menu.



-