# Architecture 2

Cohort 2 Team 5

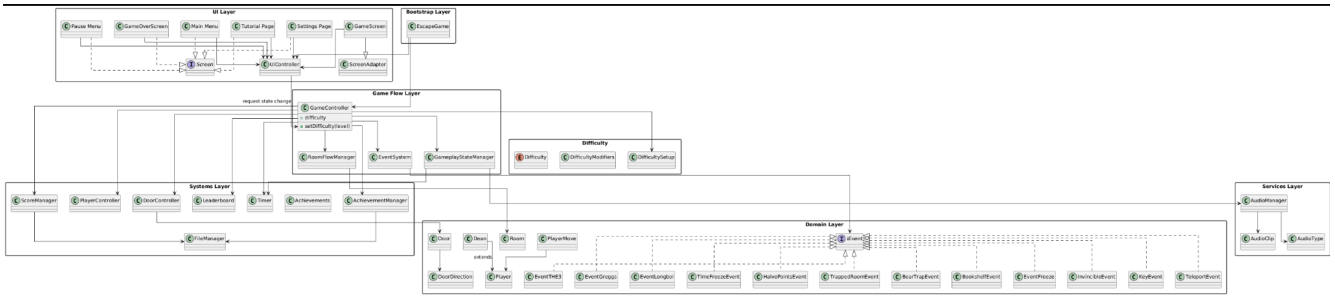
Harry Beaumont-Smith
Tom Nolan
Will Punt
Ruth Russell
Mimi Shorthouse
Lottie Silverton
Stanley Thompson

# Architecture

During the architectural design process of our game we have used various UML diagrams (class, sequence, state and component) in order to give the most concise and accurate portrayal of the games general design. We have used PlantUML in order to generate these diagrams using various IDE's and websites. To begin our design process we began by developing CRC cards, this gave us an opportunity to brainstorm the key components of the game and how it would all fit together. These cards can be found on our team website. This made designing UML diagrams much easier as we had a strong idea of the core structure of the game.

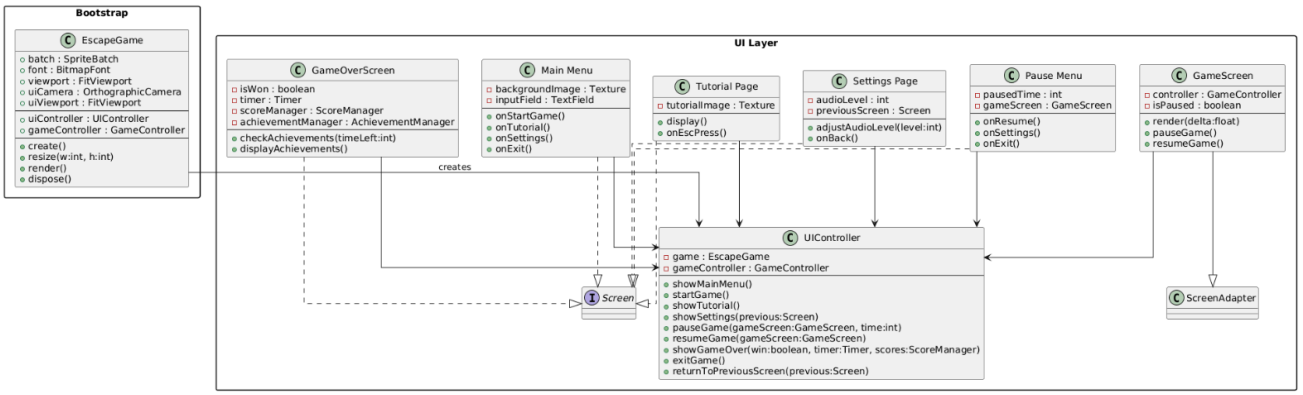
**Class Diagram:**

We began with class diagrams as these would give a great description of how all the components of the game would interact with each other, and their individual attributes and features. The process began with different team members designing various separate class diagrams that encapsulated a particular structure. Once this was completed we could combine these into one complete class diagram that describes the structure in a high level of detail.



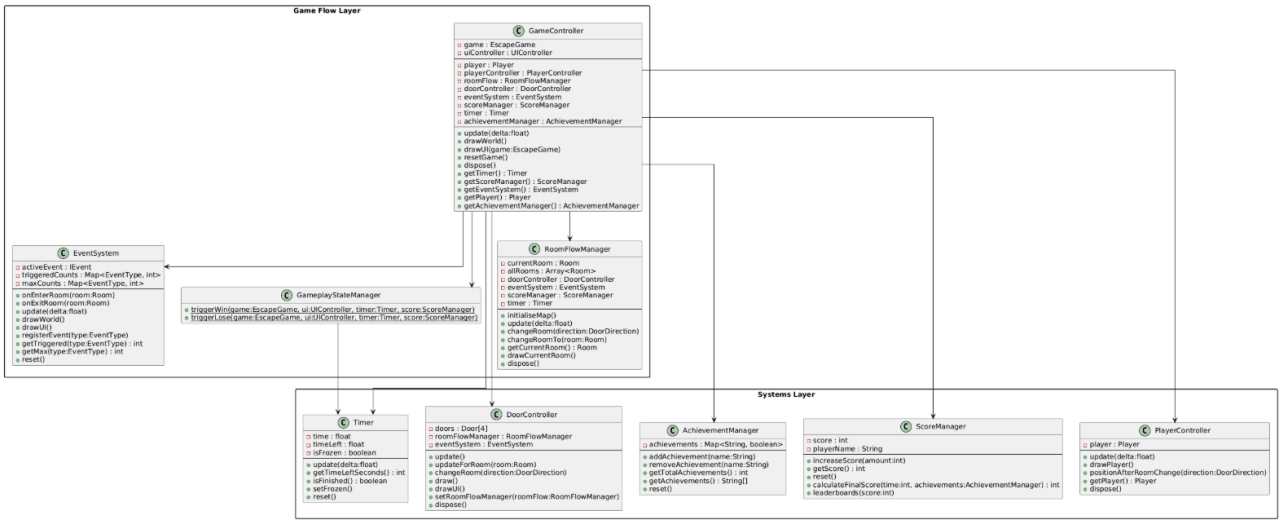

The full version of the architecture is quite wide due to the number of events present and all their components. Due to the nature of its complexity we have decided to split it into three clear, readable diagrams each representing a different part of the architecture. The first diagram is of the bootstrap and UI layer, the next is the game flow and systems layer and finally is the domain and events layer. This in turn increases the readability of architecture diagrams whilst preserving its core meaning.

UI and Bootstrap Layer:



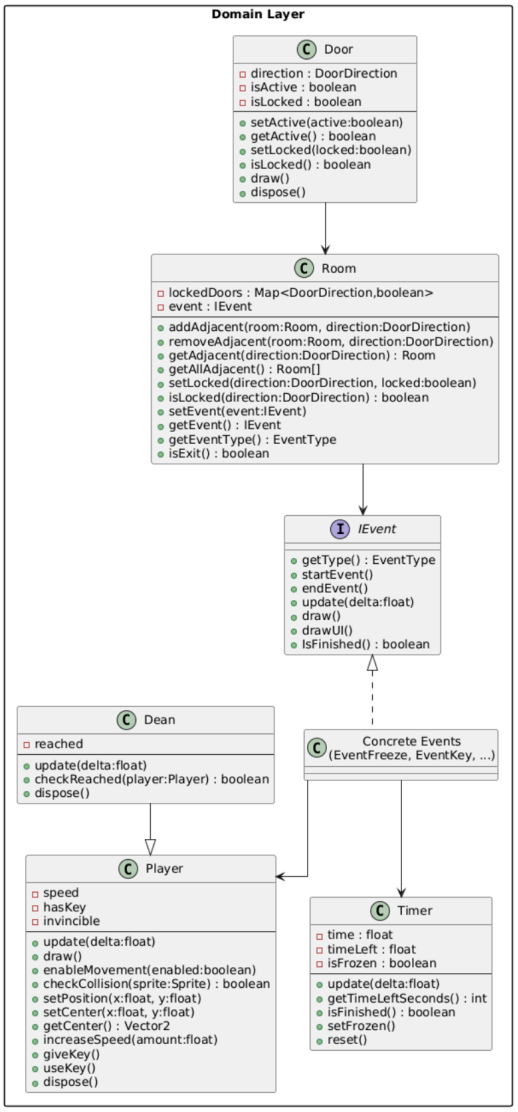
- This diagram shows the UI layer as well as the bootstrap layer, which is responsible for initialising the game through the EscapeGame class.
- This layer was set up to be very modular, as all the UI related classes communicate exclusively through the UIController instead of each other.
- Centralizing communication through the UIController reduces the direct dependencies between classes improving the modularity and maintainability.
- The UIController acts as a mediator between layers, managing data flow and interactions.
- This mediator adheres to the SOLID design principles which influenced the architecture design.

Game Flow and Systems Layer:




- This layer handles the core logic of the game and handles the room changes, event system as well as the various managers that control the subsystems
- This layer is centered around the Game Controller, which acts as a mediator between all subsystems and game flow components. This in turn reduces the direct dependencies between the classes, reducing tight coupling and limits unnecessary interconnections between the components.
- The use of the controllers and managers adds another layer of abstraction between the Game Controller and the individual classes. This further separates responsibilities across layers, improving modularity and making the architecture more maintainable.

Domain Layer:

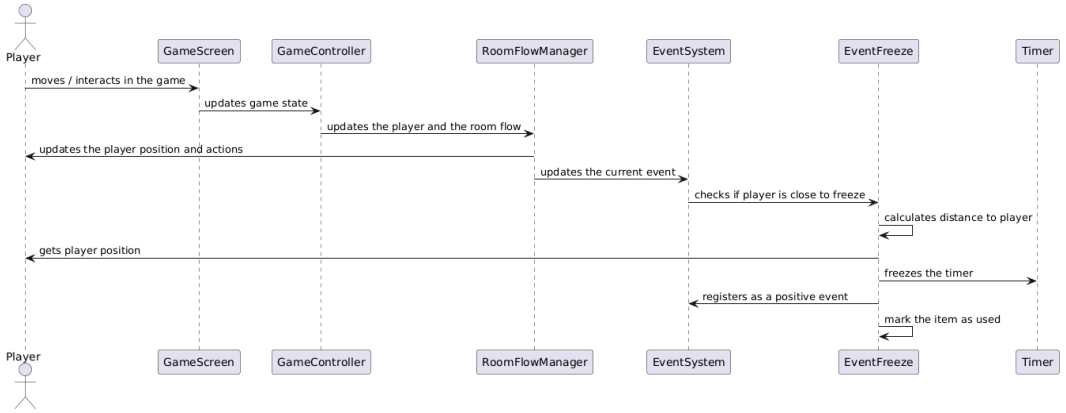



- This diagram displays the Domain layer, displaying the core gameplay entities such as Room and Door, as well as the Events (shortened for readability), including the interface for events as well.
- Interface for the event is used to improve the modularity of the events allowing for additional events to be added without modifying existing domain classes.
- Decoupling the events implementations from the rest of the system allows for the domain layer to remain focused on the core gameplay.

**Sequence diagrams:**

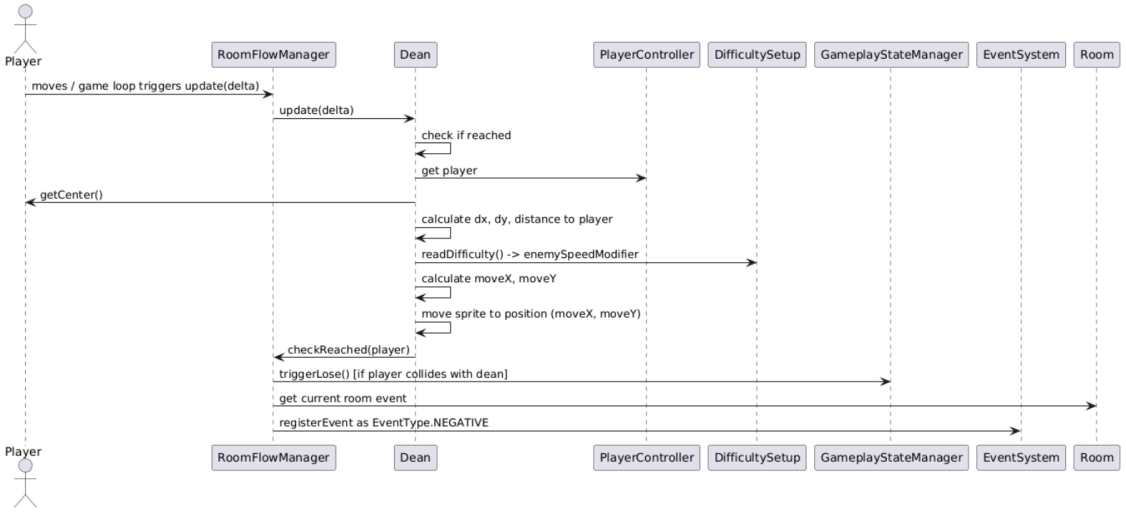
We have used sequence diagrams to describe various interactions between the player and the game; these diagrams were incredibly useful to allow us to show how requirements from the previous section would be met. The first and third diagrams describe an interaction between the player and a positive as well as a hidden event, and how the game reacts to this interaction. The second interaction describes how the Dean entity works as well as the logic behind its movement. The fourth diagram is a portrayal of how the different components work together once the player has successfully escaped. Finally we have a sequence diagram to describe an interaction where the player first pauses the game and then goes on to adjust the volume of the game.

Positive Event Sequence Diagram:



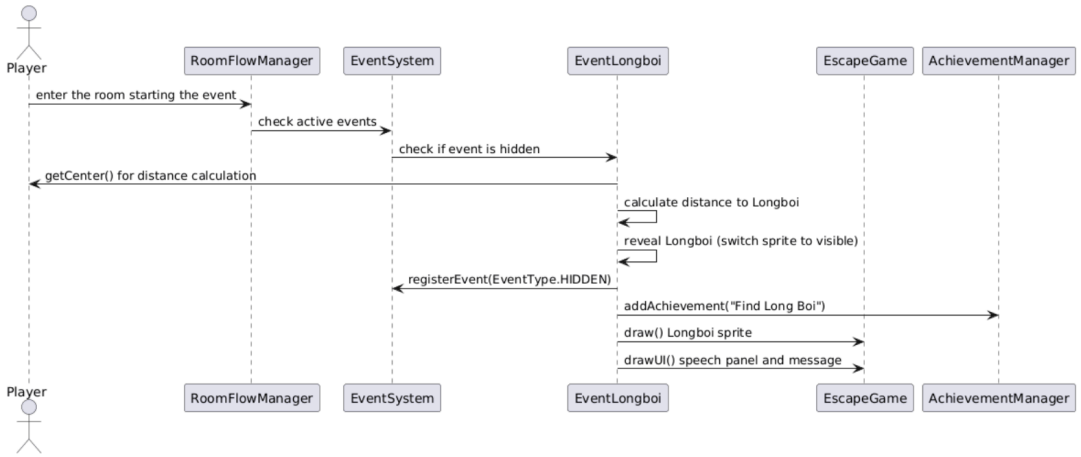

- UR_Positive - describes an implementation of a positive event subclass.
- UR_Fast - highlights a way that the player can escape quicker and obtain a better score.
- UR_Time_Tracker - the event interacts with the timer in order to freeze it, which indicates the game tracks time and thus fulfills the requirement.

The Dean diagram:



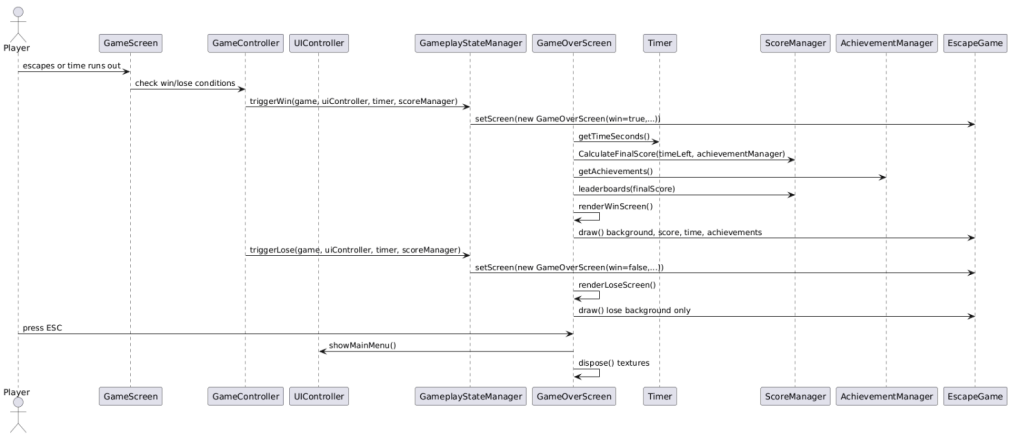

- UR_Negative - describes an implementation of a negative event subclass.
- UR_University - “Dean” refers to an over the top version of university life although still acknowledges a dean like in university.
- UR_Lose_Condition - if you are caught by the dean the game triggers its end game loss Screen.
- UR_Dean_Object - the game includes a visible dean to the player which if interacted with results in the game triggering its loose screen.
- FR_Event _Counter - upon entering the room you trigger the events counter aspect of eventsSystem which counts the event.

Hidden Event Sequence Diagram:




- UR_Hidden - describes an implementation of a hidden event subclass.
- UR_University - “longboi” is again another aspect of York University that students will enjoy, reinforcing the style of game we have chosen.
- FR_Event _Counter - Upon entering the room you trigger the events counter aspect of eventsSystem which counts the event.
- UR_Achievements - Upon discovering Longboi you activate the achievement manager which then registers the achievement.
- UR_Invisible_Events - The event stays invisible until you are near to it, after which it is visible to the player.

Game Ends Sequence Diagram:



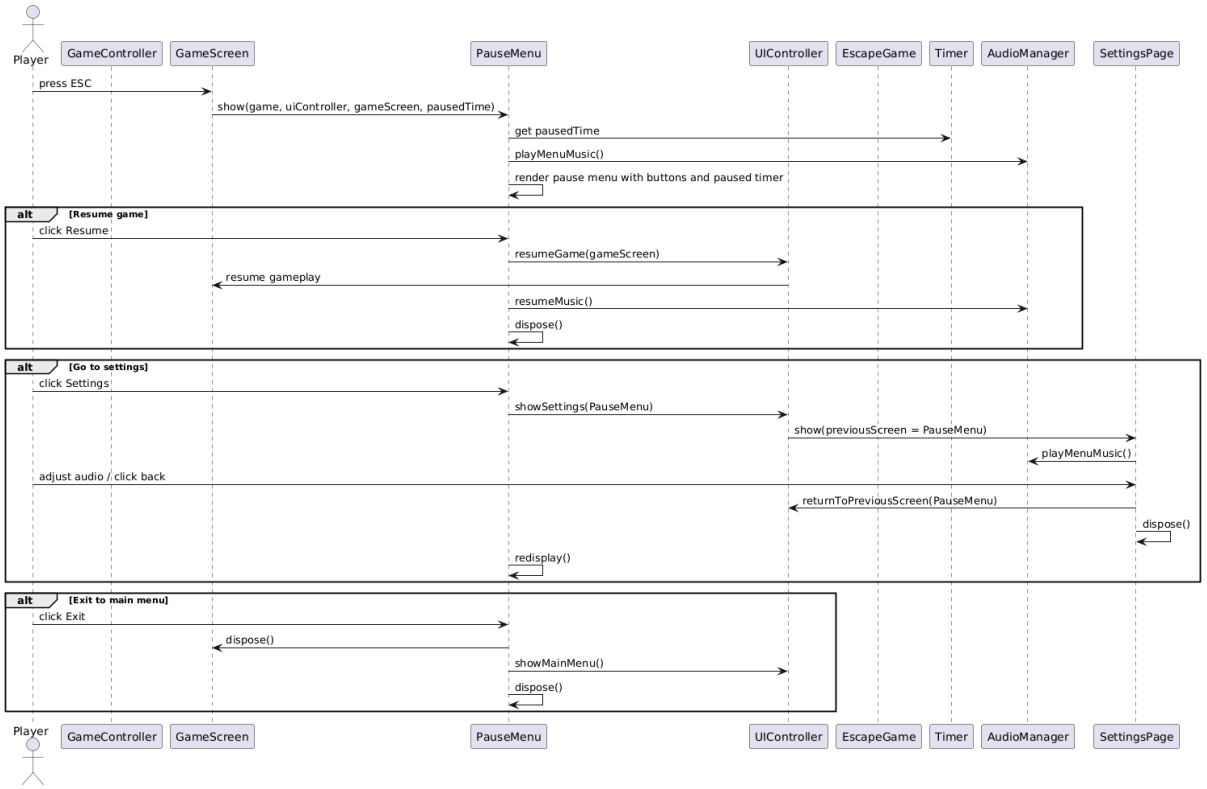
● UR_Lose_Time - highlights the outcome of what happens if the player escapes before the timer has reached 0.

● UR_Score - describes the final score being shown to the player once the game has ended.

● UR_Score - highlights the final score achieved by the player at the end of the game, once they have triggered the win condition of escaping.

● UR_Lose_Condition - highlights what happens if the player does not escape in time or is caught by the dean.

Pause Menu Sequence Diagram:




● UR_Pause - provides a description of how the player can pause the game, and the logic that follows (pausing timer and audio playing).

● UR_Settings - shows how the player can navigate the pause menu to reach the settings and the audio options that follow.

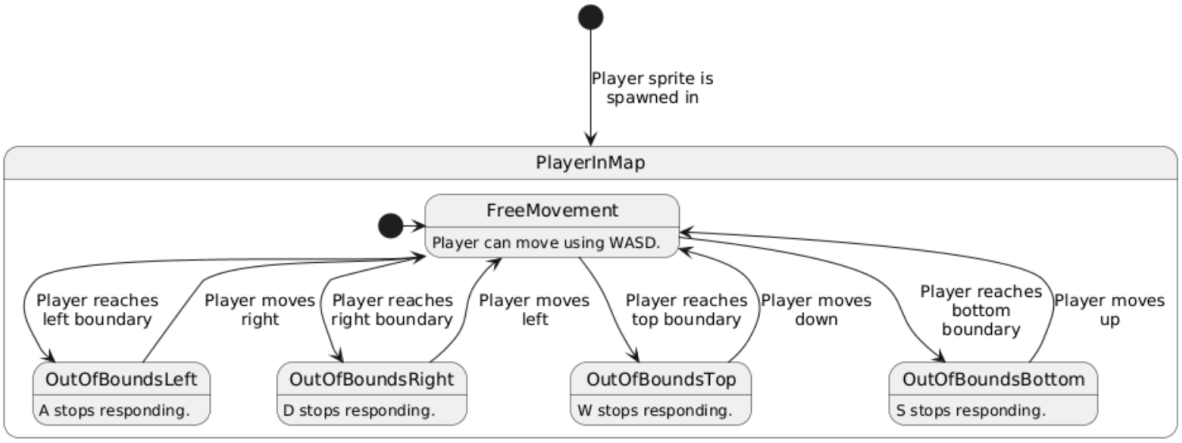
● UR_Audio - describes how the audio is adjusted by the player from the settings page and what needs to be updated.

● UR_Time_Track - gives an example of how the current time can be requested and describes the time remaining on the pause screen.

**State Diagrams:**

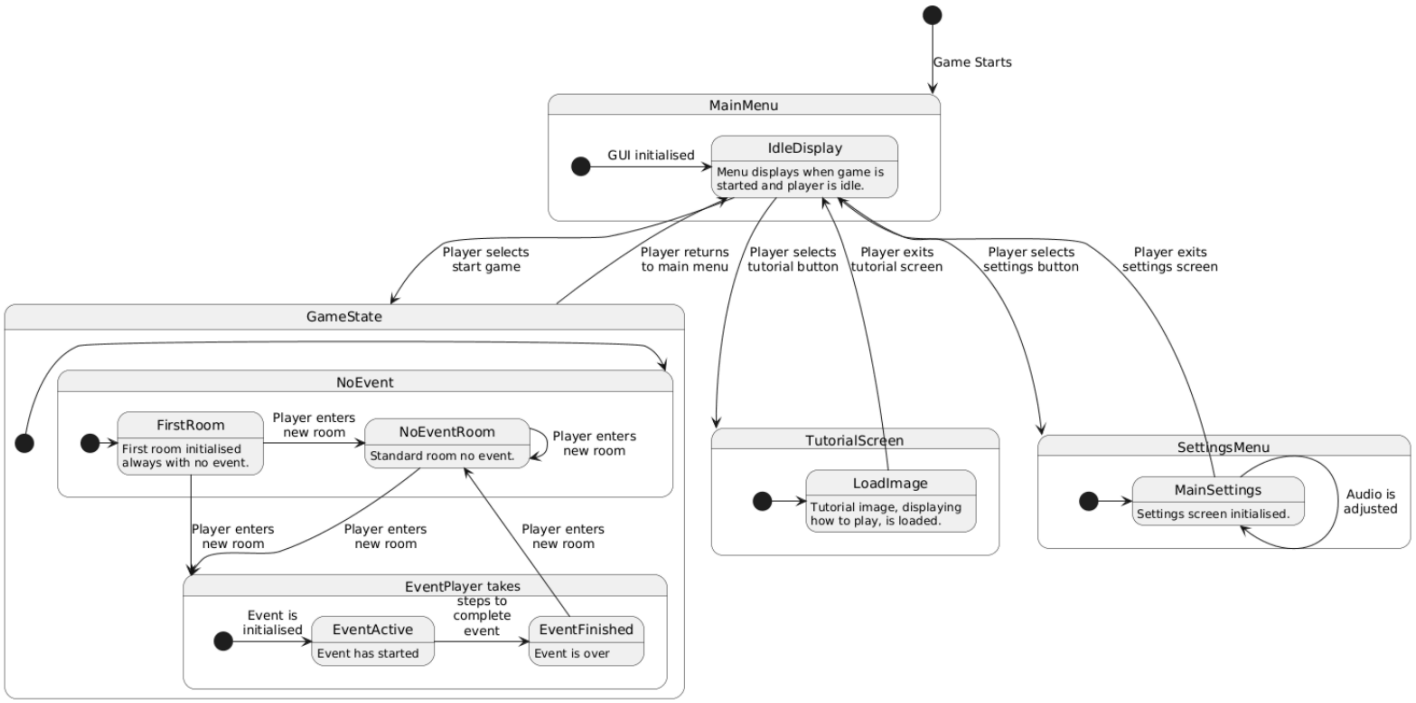
Our state diagrams have been key in portraying the different states displayed when the game is first loaded and what happens when the player interacts with this system. We have also used it to highlight how the boundaries of the map are enforced and how this links to the controller binds. In the final diagram we show a simple diagram to describe how the game checks to see if the timer has run out or if the player is caught by the dean and to end the game once this has happened.

Map Boundary State Diagram:




- UR_Player - describes without detail a player being spawned into the map as a sprite that has the ability to move around the map.
- UR_Map_Limits - clear boundaries are shown and each one is represented as a different state within the overarching state. When a player tries to exit a boundary the corresponding directional input is temporarily disabled.
- UR_Desktop_Inputs - shows the mapping between the input keys and the movement it elicits by the respective boundaries.

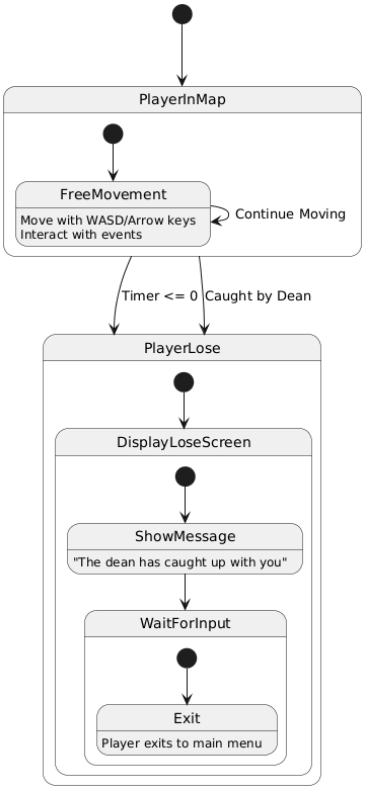
Game Start State Diagram:




- UR_Main_Menu - shows how once the game is loaded up, the main menu is immediately displayed, displaying the ability to change to various other game states.

- UR_Tutorial - this diagram shows how the state of the game changed when the player selects the tutorial from the main screen, displaying the image.
- UR_Settings - highlights how the settings menu is accessed from the main menu and gives a brief portrayal of how audio can be adjusted.
- UR_Positive/Negative/Hidden - gives a general description of how the logic works for each event type within the game state (partitioned into having an event and not having an event).

Player Loses State Diagram:

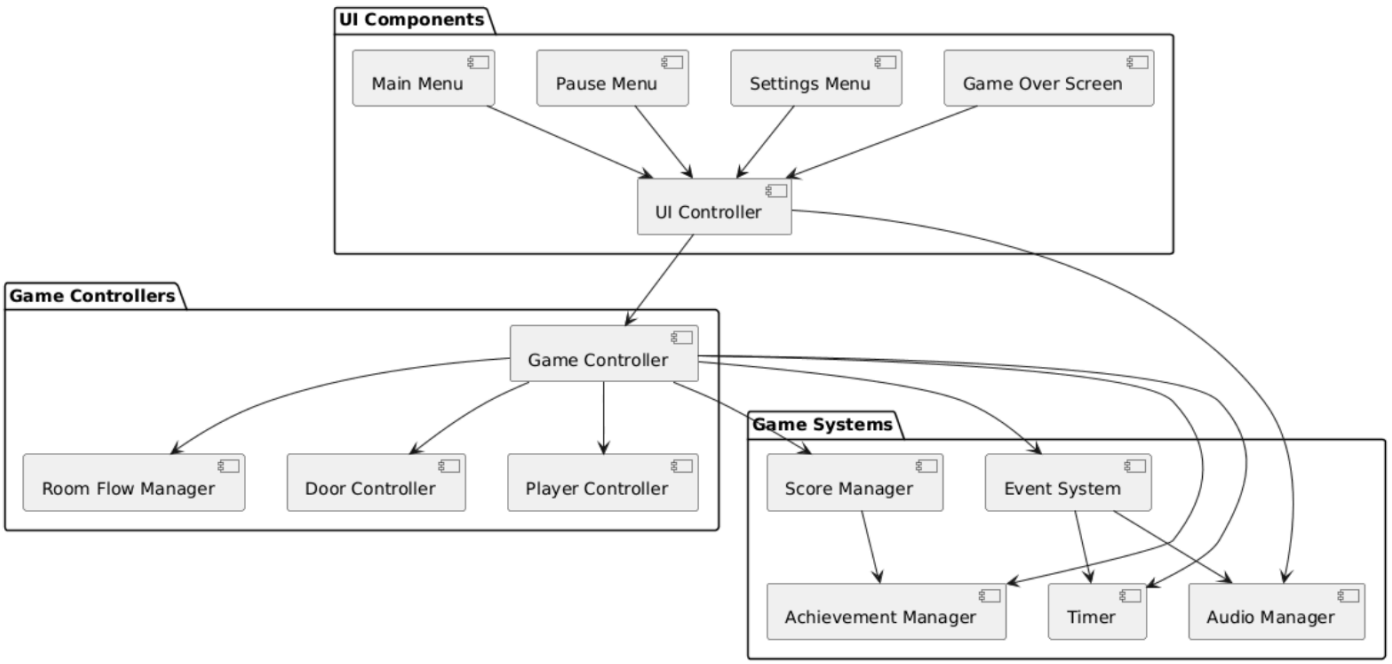



- UR_Lose_Time - this diagram primarily describes how the game checks to see if the timer has reached 0 and what happens if this does occur.
- UR_Time_Track - briefly shows how the game is always checking and “listening” to see what the value of timer is.
- UR_Dean_Object - the game ends when the dean collides with the player.

**Component Diagram:**

We created a component diagram that describes the logic and access level, each of the components of the game use to interact with one another. These are split into menu components, game managers as well as the game system. This helped to create a more abstracted version of our class diagram, with the aim of describing the logic between menus and controllers with less focus on detailed mechanics. The key aim at this point was to increase modularity in order to ensure the ease of future testing, something which can be seen as only the controller of menu components and game controllers interacting with the other packages. This reduces tight coupling and reduces cohesion, improving the architectural design of the game.

UI and Controller Component Diagram:




- UR_Main_Menu - highlights which parts of the game the main menu can access.
- UR_Pause - like earlier shows how the tutorial is accessed via the main menu.
- UR_Audio - describes how audio is accessed both via the game menus and how the room manager controls the audio playing.
- UR_Settings - the settings can be accessed via either the main menu upon booting up the game or via navigating the pause menu, during gameplay.
- UR_Achievements - the achievement Manager stores all the achievements and results in them being given to the player.
- UR_Score - the score Manager stores the players score and works it out once the player escapes.