Assessment 1: Greenfield Development - Architecture

Cohort 1 Team 2

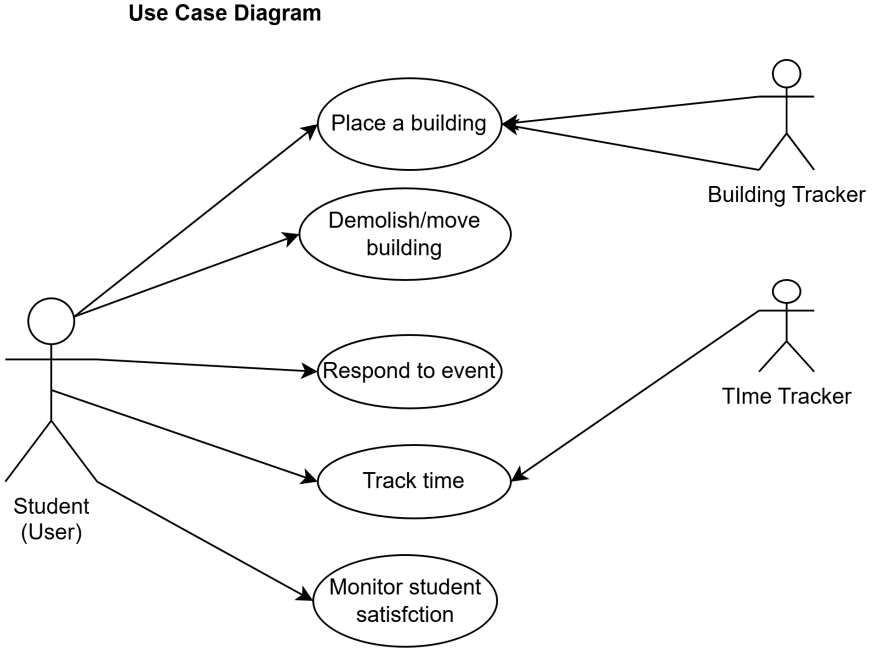
Team Members:

Trace Chinelle
Vidhi Chohan
Apollo Cowan
Siyuan Liu
Aryaman Marathe
Charlie Mason

UML- Tracey Chinelle, assisted by Charlie Mason

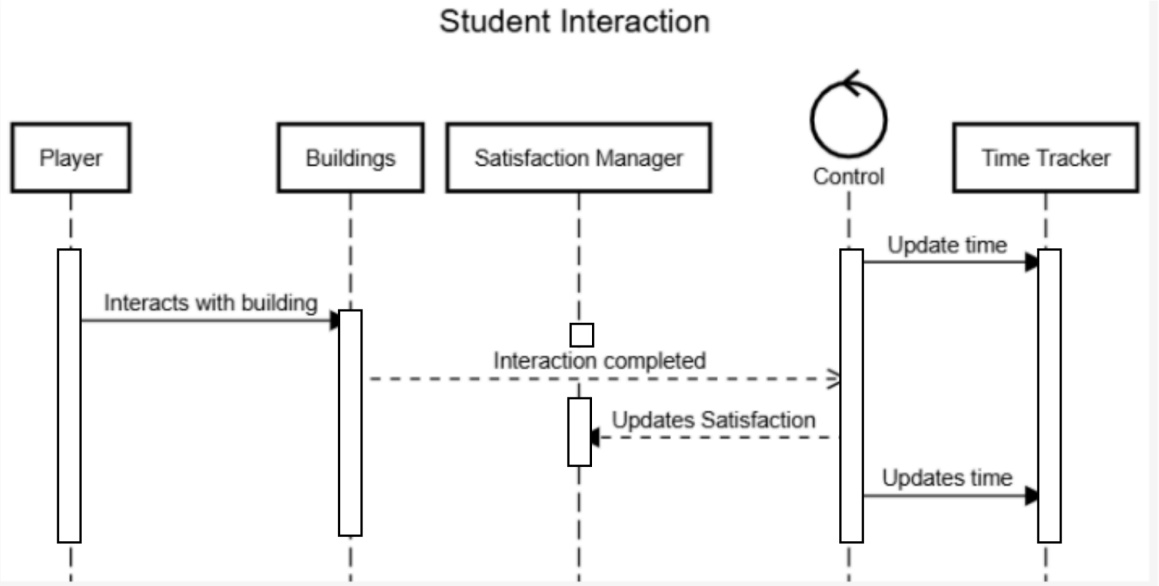
We created the UML diagrams to provide a clear and structured representation of the static architecture and dynamic behaviours of the UniSim game. This helps to break the system into manageable components, showing how objects like **Student**, **Building**, and **EventManager** interact and change over time. The class diagram defines the structure of the game ensuring modularity and scalability. The sequence and state diagrams illustrate how the components collaborate when handling actual gameplay scenarios and transitions. These diagrams directly link the project's requirements to maintain consistency and traceability. This systematic approach supports maintainability, testing, and scalability for future use.

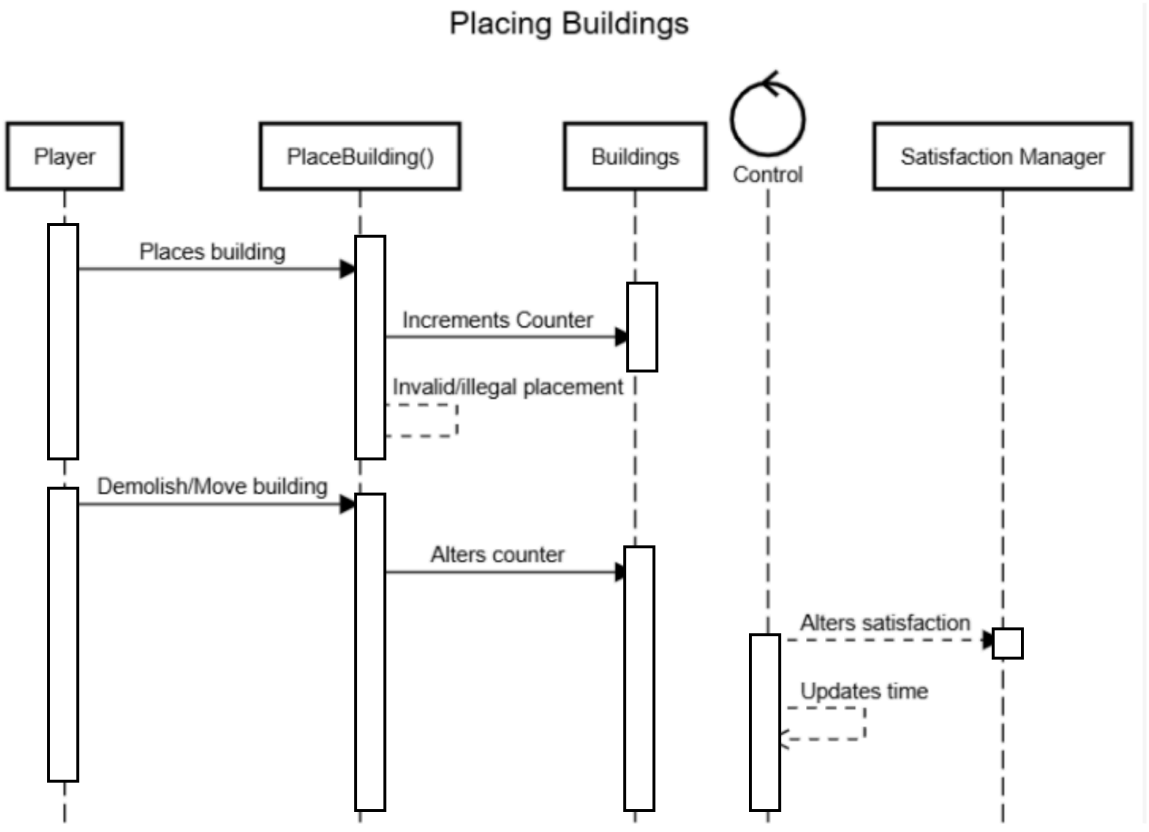
Behavioural Diagrams

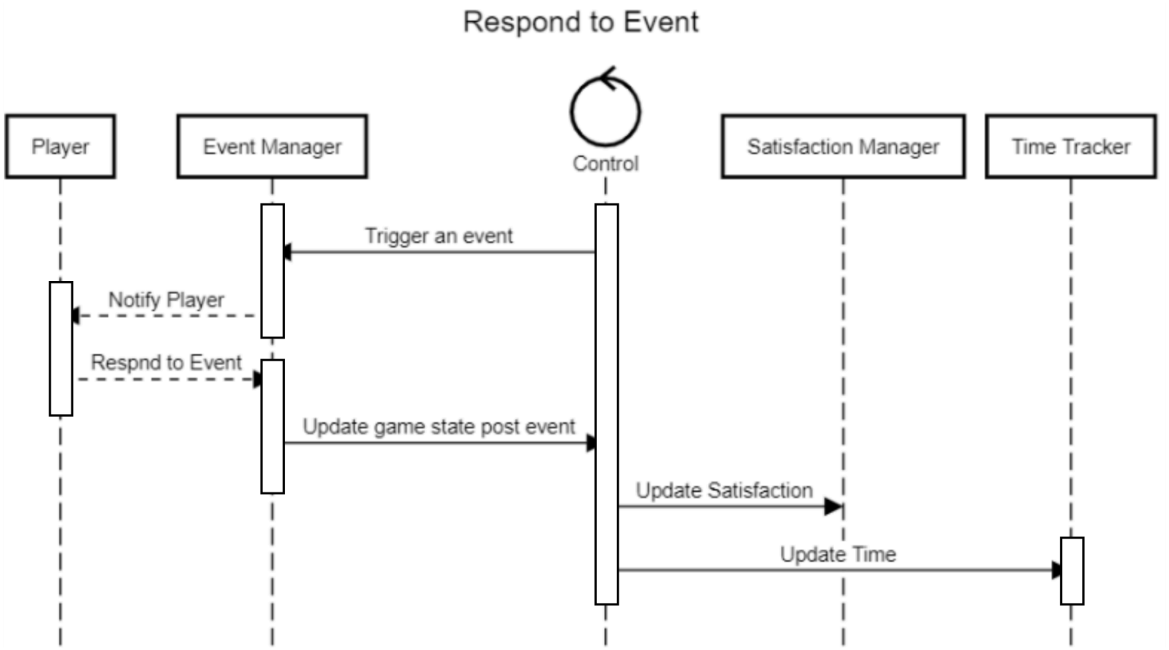


I created these to define the main interactions from the player's view. The diagrams abstract in-depth details and system requirements, providing a high-level overview of usability (what the player can do). For instance, placing a building aligns with **UR_PLACE_ACTIVITY** and **UR_PLACE_FOOD**. These requirements ensure the core aspects remain in focus during design and implementation. This user-centred design approach guarantees traceability by linking actions to specific functional requirements, such as responding to events (**UR_REACT_TO_EVENT**) and tracking time (**UR_DURATION**). Use case scenarios like **Placing a Building** and **Monitoring Student Satisfaction** also aid in validating and testing features.

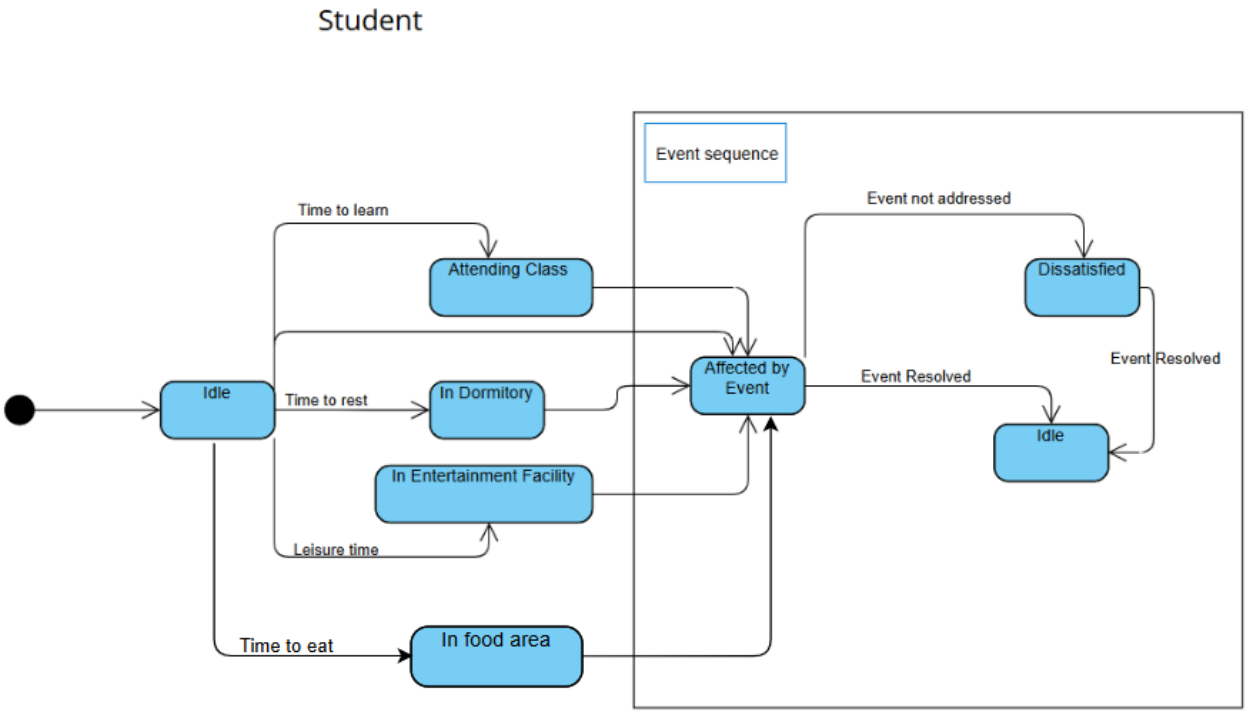
Created with draw.io

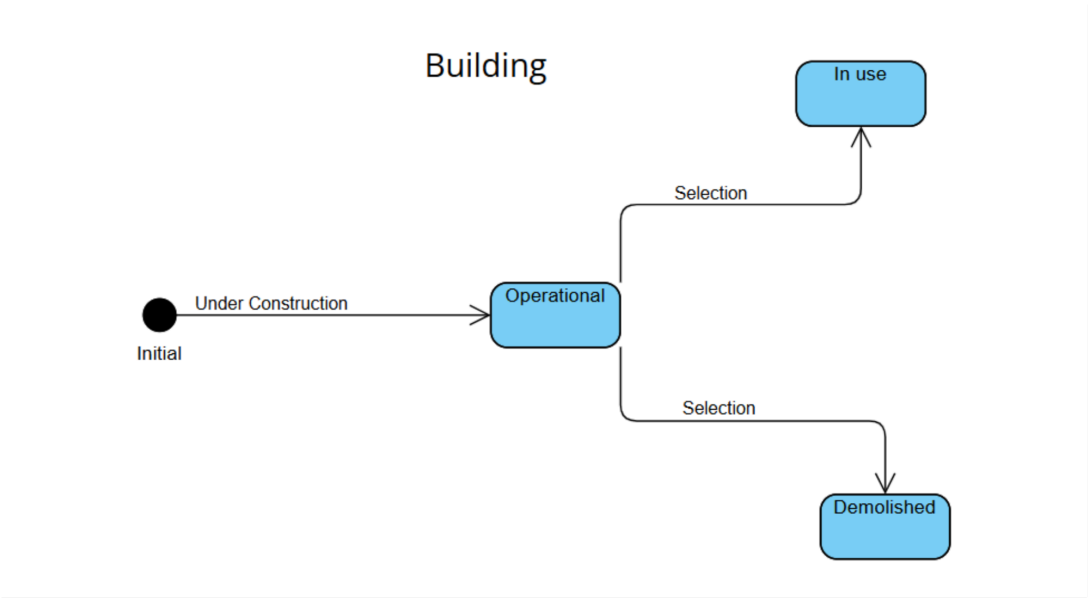

Student Interaction
Player
Buildings
Satisfaction Manager
Control
Time Tracker
Update time
Interacts with building
Interaction completed
Updates Satisfaction
Updates time

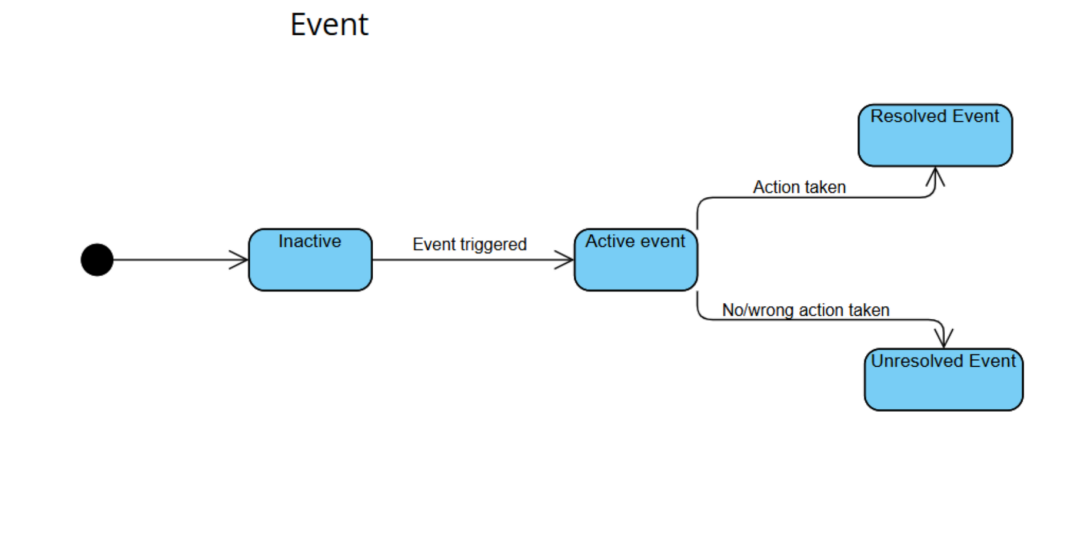

Placing Buildings
Player
PlaceBuilding()
Buildings
Control
Satisfaction Manager
Places building
Increments Counter
Invalid/illegal placement
Demolish/Move building
Alters counter
Alters satisfaction
Updates time



The sequence diagrams capture communications between objects such as **Player**, **Building**, and **EventManager**. For example, the **Place Building** sequence supports **UR_CHOOSE_PLACE** and **UR_RESTRICTED_AREA**, ensuring buildings are positioned within valid areas. The event response sequence aligns with **FR_REACT**, which allows the player to address positive, negative, and neutral events appropriately. By showing object dependencies and interaction timings, these diagrams enhance modularity and ensure gameplay actions are traceable to their corresponding requirements.
Created with sequencediagram.org



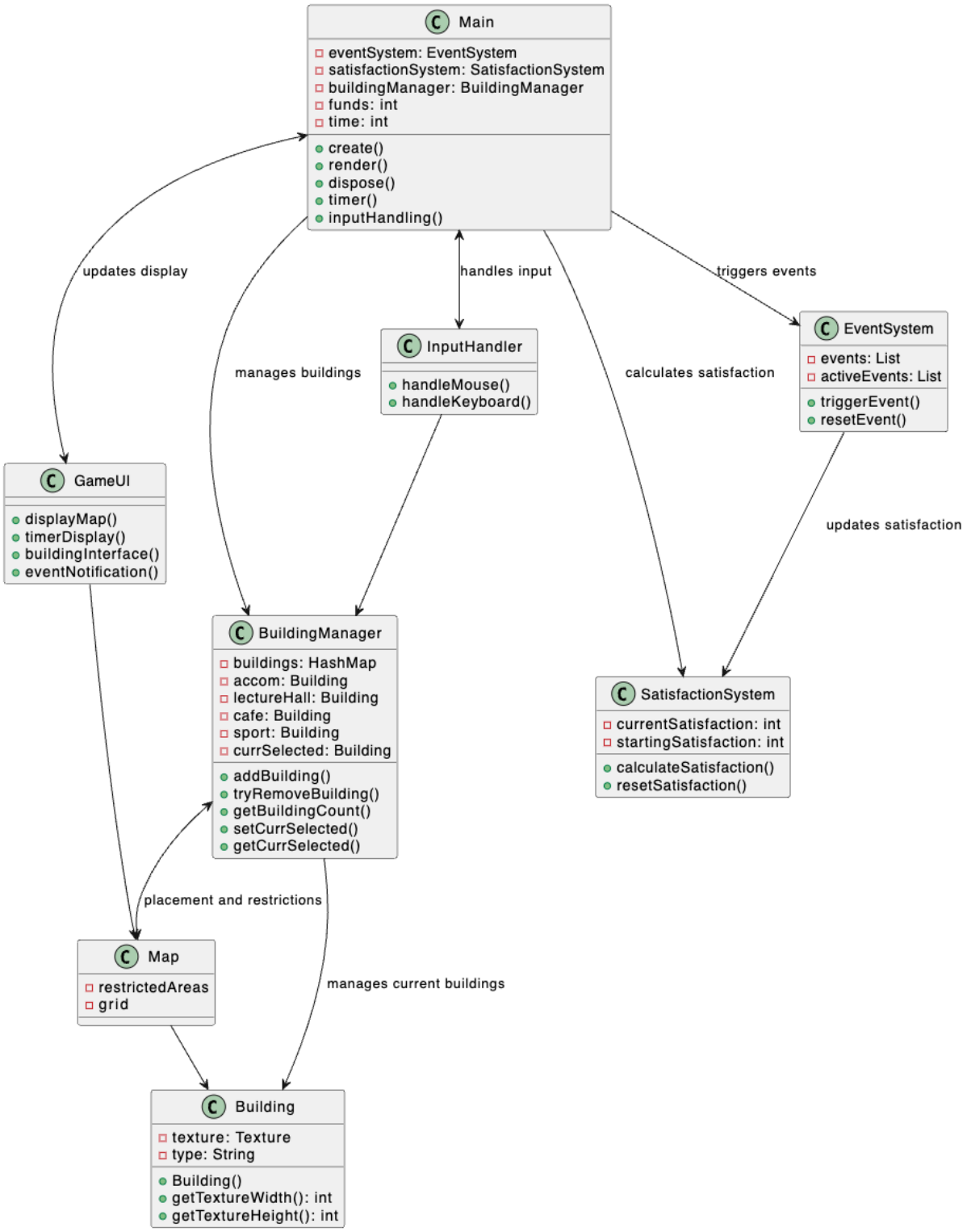




The state diagrams highlight the lifecycle and state transitions of UniSim's key components. For instance, the **Student** component state transitions - such as **Idle** to **Attending Class**- align with **UR_SS_SCORE**, which mandates that student satisfaction is updated based on gameplay events. Similarly, the **Building** component's lifecycle, moving from **Under Construction** to **Operational**, supports requirements like **FR_BUILD_TIME**. These diagrams illustrate fault tolerance and scalability by showing how the system handles transitions, ensuring the architecture supports ongoing feature expansions while meeting initial requirements.

Created with visual-paradigm.com

Structural Diagram- Charlie Mason



The **Main** will be the main part of the code, which will initialise the **Map**, **Buildings**, **EventSystem**, and **SatisfactionSystem**. The **Main** will be part of a loop that will manage the timer, process the user input, update the map when buildings are placed and process the events and satisfaction systems. The timer corresponds with **UR_DURATION/FR_DURATION** and **UR_PAUSE/ FR_PAUSE**. Having separate components for the **Map** and the building gives greater freedom in the interaction the building types have on the student satisfaction score, which helps fulfil the **UR_CHOOSE_PLACE** requirement. The **Map** is to be based on a grid system and allows the user to easily know where they can and cannot place buildings. This feature helps to satisfy **FR_SNAP_GRID** and complements **NFR_USABILITY**.

The separate classes for **EventSystem**, **SatisfactionSystem**, **Map**, **Building** and **BuildingManager** allow us to efficiently debug and test different aspects of the game without ruining its core functionality, which aligns with our **NFR_MAINTAINABILITY** requirement. The **EventSystem** is designed to trigger an event which would influence the student satisfaction score. By having a class dedicated to **EventSystem**, we can ensure that **UR_THREE_EVENTS** can be implemented with the three different types, **FR_POS_EVENT**, **FR_NEG_EVENT**, and **FR_NTRL_EVENT**. The decisions made by the user will influence the satisfaction score letting **SatisfactionSystem** recalculate and satisfy the **UR_SS_SCORE** requirement.

The **InputHandler** & **GameUI**, work in parallel to let the user interact easily with the game interface. The **GameUI** will show updated information to the user, which meets the criteria for **UR_UX** and **FR_UX_DESIGN** requirements. The **InputHandler** will process the changes the user wants to make, creating a more seamless experience for the user which corresponds to the **NFR_OPERABILITY** requirement. The modular design of the structure helps to meet the **NFR_SCALABILITY** requirement which allows for any future updates to be seamlessly added, for example, a greater variety of events added to the **EventSystem,** and a larger **Map** with a wider range of buildings to choose from.