Architecture

Cohort 3 Group 6 - Carbon Goose

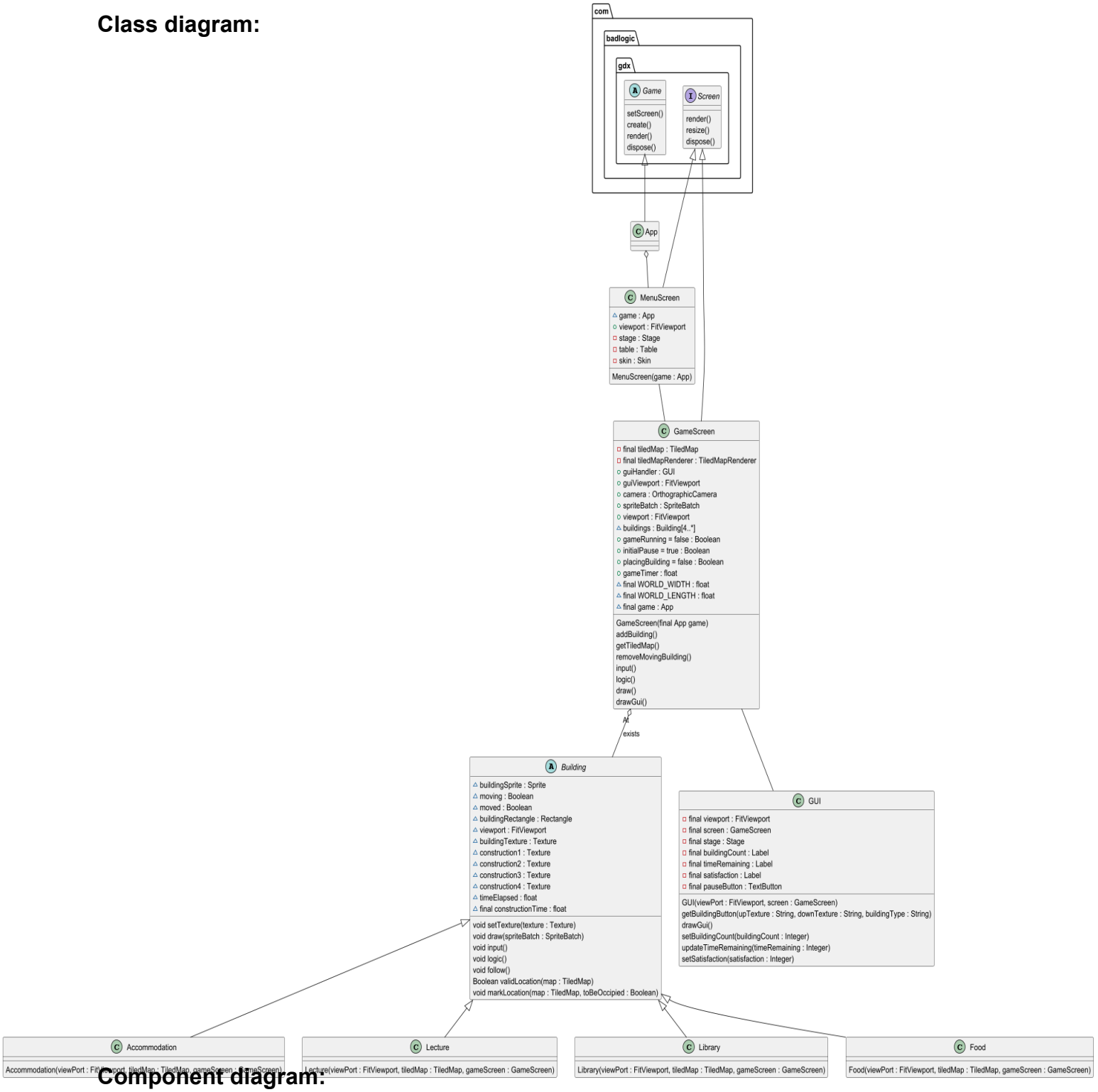
Members: Bailey Horsley, Owen Jones, Rory Ingram, Ken Jacob, Abishek Kingslin Raj, Louis Polwarth, Adam Johnson

Diagrammatic Representations

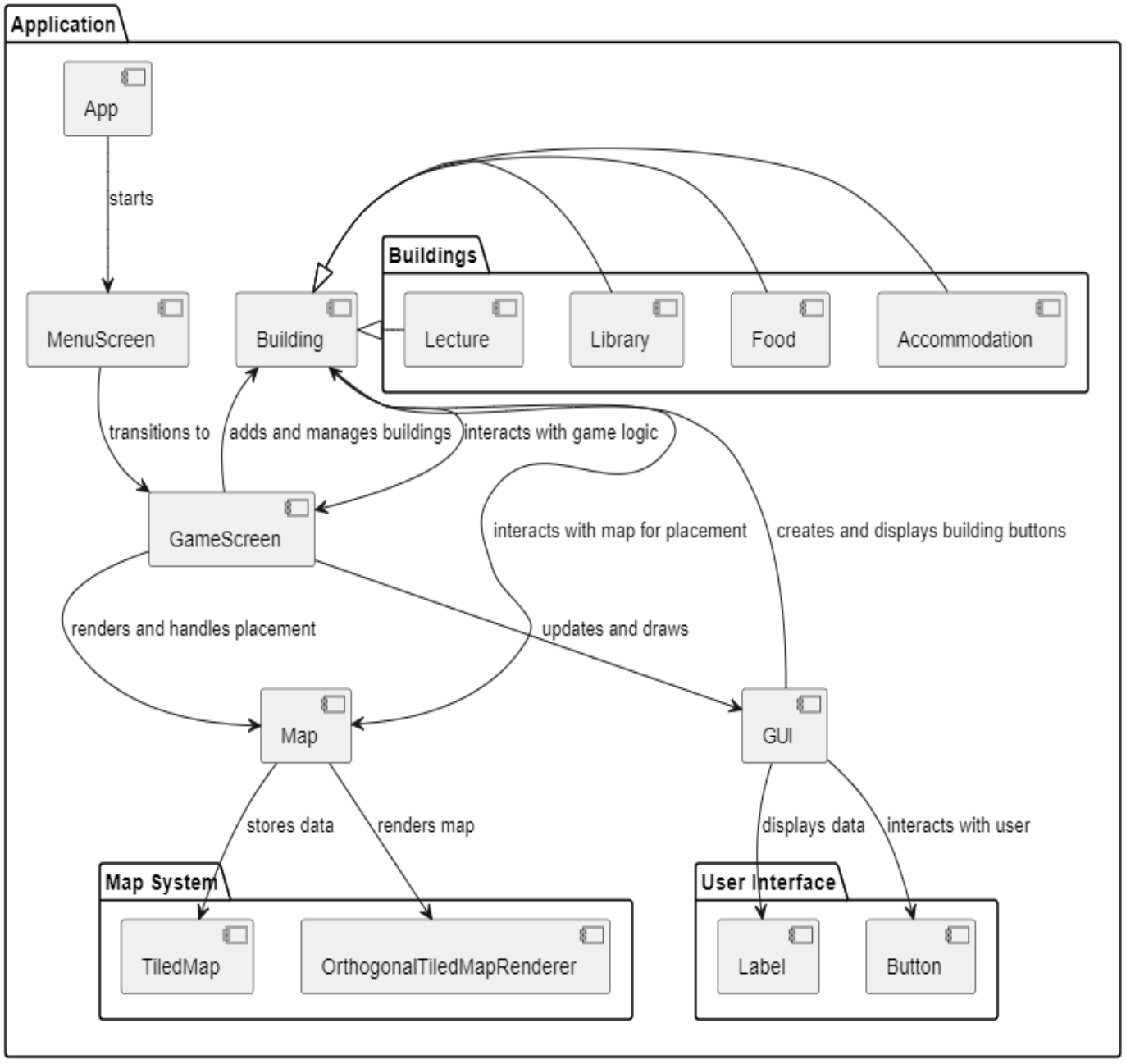
For our diagrams, we used Unified Modeling Language (UML) as the primary language. It provides a standard way to visualise the architecture, making it suitable for representing both the structural and behavioural aspects of UniSim. In order to utilise this language and produce the diagrams we used an open-source tool called PlantUML.

Structural Diagrams

Class diagram:

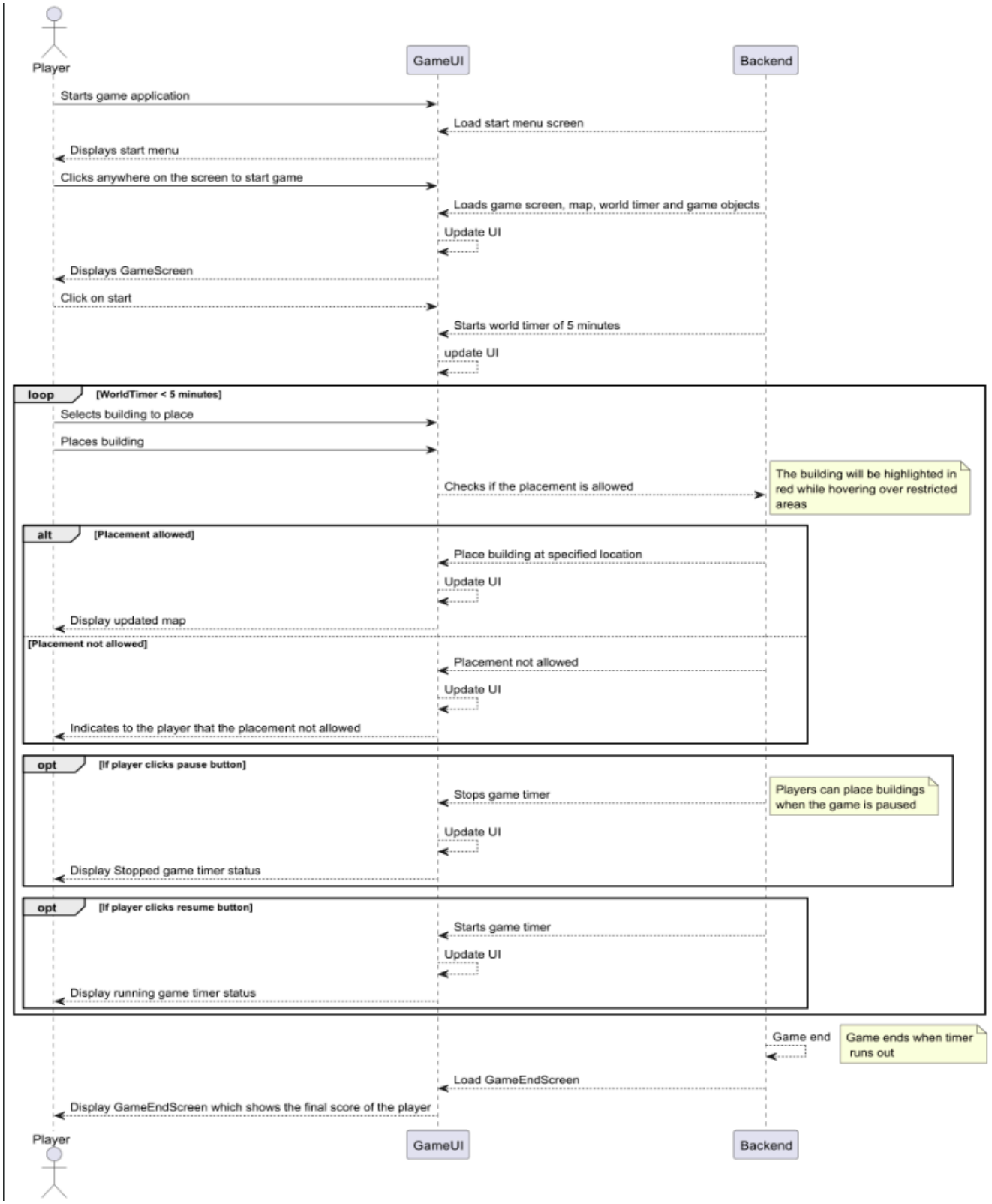


Component diagram:

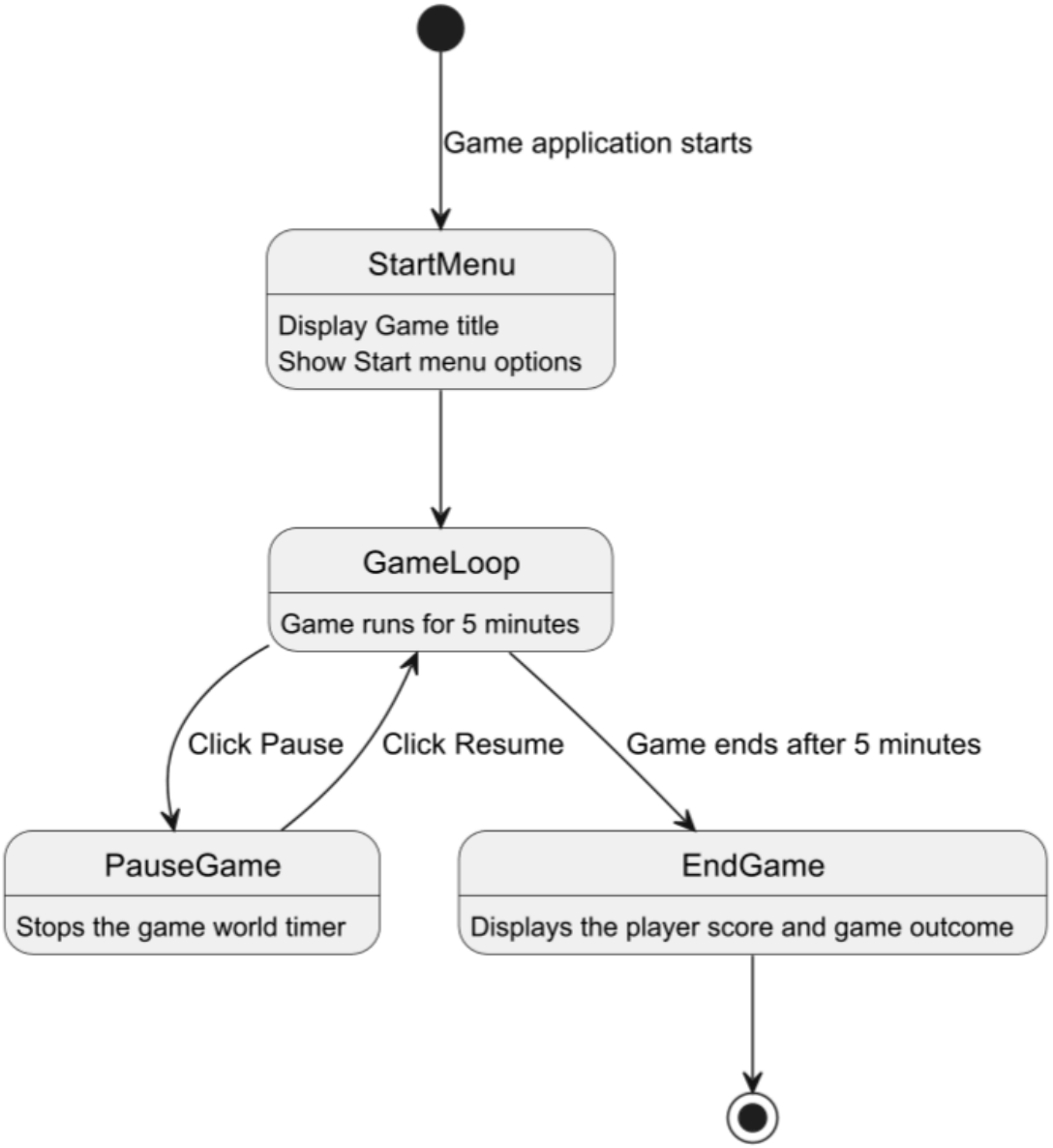

Application
App
starts
MenuScreen
Building
Buildings
Lecture
Library
Food
Accommodation
transitions to
adds and manages buildings
interacts with game logic
GameScreen
interacts with map for placement
creates and displays building buttons
renders and handles placement
updates and draws
Map
GUI
stores data
renders map
displays data
interacts with user
Map System
User Interface
TiledMap
OrthogonalTiledMapRenderer
Label
Button

Behavioural Diagrams

Sequence diagram:



State diagram:



Justification of Architecture

When planning the architecture for UniSim, it was clear that the project would require event-driven and component based architecture, as they align with the game's dynamic nature and help deconstruct the core requirements into independent components to be built. The game needs to respond to user actions such as placing buildings (FR_PLACE_BUILDING) or pausing the game (FR_PAUSE), both of which trigger events that update the game state (FR_SATISFACTION_UPDATE, FR_CURRENCY_UPDATED). These events needed to be identified in order to update the game state, allowing for decoupling of the game mechanics (component-based), making it easier to extend and modify. Overall, this architecture should allow others to easily extend the system in later stages, adding more events and features if desired, and guarantees that the game remains flexible, modular, and responsive, all while ensuring clarity.

In order to ensure that this architecture was the most suitable for UniSim, we compared it to other styles and methods and considered trade-offs as a basis for our rationale.

An alternative was a request-response type of architecture, which would require the game to constantly wait for player input and process the game's state in a sequential manner. While this could work at a very simple level, it would lead to inefficient updates in UniSim, considering our need for a real-time response to user actions like building placement (UR_BUILDING_) and events (UR_EVENTS). An event-driven architecture however, allows the game to be responsive by handling user actions as independent events that can occur at any time. This is ideal for UniSim as it can maintain fluid gameplay, and respond to specific events rather than continuously processing a queue of requests.

Furthermore, a monolithic type of architecture could have been considered, which is where all game logic would be integrated in a single codebase, allowing for a simplification in development. However, this would be difficult to maintain and extend as the game grows, as well as making it harder to isolate and fix issues. Instead, we opted for a component-based architecture, where distinct components can be independently updated, providing scalability and modularity, which allows us to update features like events (NFR_EVENTS) or building upgrades (FR_BUILDING_UPGRADES) without major disruption to existing functionality (NFR_SYSTEMS).

Design Evolution Over the Project Lifecycle

We initially made a rough sketch of the initial class diagram on a whiteboard, outlining the core components of our game. We had a "Main" class which controlled the main game loop, the GUI, game objects such as map, buildings, game timer and the player stats.

The "Player" class which represents the player's progress and stats, has a primary attribute "satisfaction" which tracks the satisfaction among the students in the university, aligning directly with the requirement UR_CUSTOMER_SATISFACTION. It also has a method "updateSatisfaction()" to update the student satisfaction in the university based on the player progress, which corresponds to FR_SATISFACTION_UPDATE.

The Map and GridSquare classes structure the game environment, with Map as a tiled map and each GridSquare representing a tile. GridSquare includes an isWater() method, which restricts building placement on water tiles, enhancing realistic gameplay.
We use an abstract Building class to define shared attributes and behaviours, such as a buildingSprite for visuals and buildingTexture for texture management. Specific building types, like LectureHall, Cafe, and Accommodation, extend the Building class, each representing a unique building the player can place on campus to build their university.

Finally, we used PlantUML to formalise this class structure, translating our initial rough sketch into a clear, organised class diagram that guided our development process as shown in week 3 of "Design evolution" part of our website.

After further research on our preferred game engine and discussion among the team, we refined our initial approach, firstly we introduced a "MenuScreen" class to handle the start menu UI. Next, we implemented the GameScreen class where we set up our map using a tile based system and configured its viewport and camera to define the visible game area. Additionally, this class also

handles essential gameplay elements such as tracking the number of buildings placed, updating game world timer and the rendering of the map and other game objects.

A “GUI” class was implemented to handle the game user interface, displaying the game stats and controls, including buttons to start, pause and resume the game, as well as labels to show the number of buildings placed, world timer and the current satisfaction level among students.

In addition to the initially planned abstract building class and the concrete classes Lecture, Food and Accommodation, we added the Library class for the university library building, where the students could engage in recreational activities and socialising. We updated our abstract Building class, adding more textures for indicating the construction progress of a building. Also added methods to check if a tile is valid for building placement, marking the occupied tiles and to allow the buildings to follow the cursor when choosing their location on the map and a dedicated method for the construction process.

Lastly, we refined our “main” class, renaming it “App”, which handles the initial setup of the game application.