Architecture

Group 4 - Cohort 2

Theo Coleman

Matias Duplock

Mohamed Eljak

Katie Schilling

Juliet Urquhart

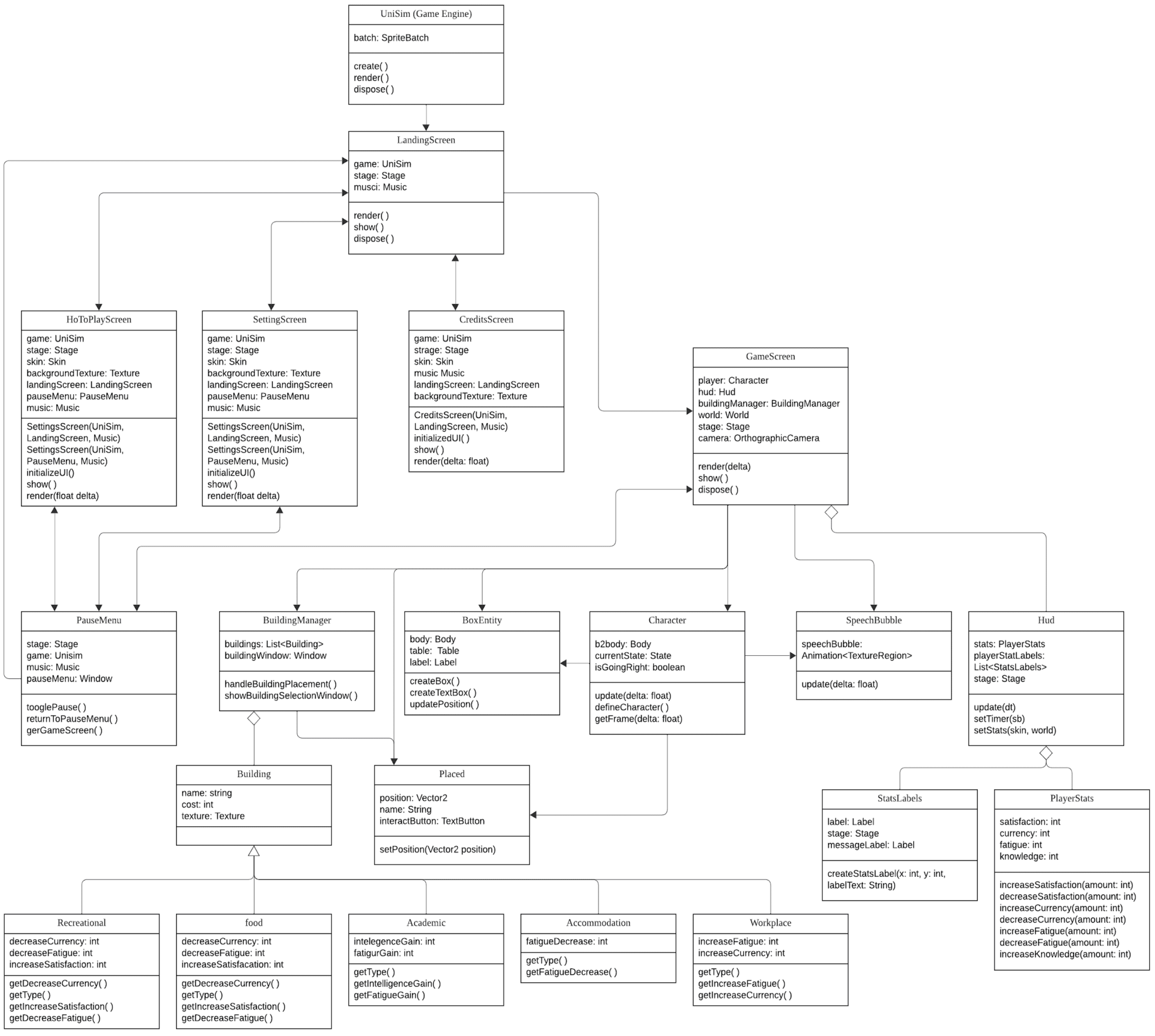
Nora Wu

Structural Diagram

We developed structural diagrams to represent the architecture of UniSim, focusing on key components and their interactions. The diagrams were designed using **Lucidchart**, which provided collaborative features and allowed us to create customised UML diagrams effectively. Our diagrams include:

- **Class Diagrams**: These visualise attributes and methods for each class, giving a clear overview of each component.
- **Relationships (Association, Aggregation, Composition)**: Represented using distinct UML notations such as arrows, diamonds, and filled diamonds to indicate the type and direction of relationships.

The UML diagrams provide a high-level overview of UniSim's architecture, depicting the relationships between major classes such as **UniSim**, **GameScreen**, **BuildingManager**, **PauseMenu**, **Character**, and more



Systematic Justification for the Architecture

The architecture of UniSim was systematically designed to fulfil functional requirements for an interactive simulation game, with an emphasis on **modularity**, **extensibility**, and **maintainability**. The key design objectives were to ensure the system handled user interactions efficiently while being flexible enough to accommodate future expansion.

Key Design Principles:

1. **Separation of Concerns**:
   - The architecture is divided into classes that handle specific responsibilities, such as **GameScreen** for gameplay logic (FR_PLAYER_MOVEMENT, UR_NAVIGATION), **PauseMenu** for pause functionality (FR_PAUSE_MENU, UR_PAUSE), and **BuildingManager** for managing buildings (FR_BUILDINGS, FR_BUILDING_PLACEMENT, UR_ADD_BUILDING).
   - This separation allows each class to be developed, tested, and maintained independently, significantly reducing system complexity.
2. **Modularity**:
   - The use of distinct screens such as **LandingScreen**, **GameScreen**, **CreditsScreen**, and **SettingsScreen** ensures modularity. Each screen manages its own UI and game features, reducing dependencies across the application (UR_INTUITIVE).
   - The **Building** class and its specialised subclasses (**Academic**, **Accommodation**, etc.) create a clear hierarchy, enabling new types of buildings to be easily added with minimal changes (FR_BUILDINGS, FR_BUILDING_PLACEMENT).
3. **Extensibility and Reusability**:
   - **Aggregation** relationships were leveraged to reuse components efficiently. For example, **GameScreen** aggregates **Hud** and **BuildingManager**, enabling these components to be reused across different screens (FR_STATS, UR_VARIABLES).
   - Components like **Hud**, **PlayerStats**, and **BuildingManager** were designed to be easily reusable, promoting scalability without significant rework (NFR_UPDATE_STATS).
4. **Decoupling**:
   - The **UniSim** class acts as the central entry point, managing screen transitions without directly controlling the internal workings of each screen. This decoupling allows the various components and screens to evolve independently, avoiding cascading changes across unrelated areas (UR_NAVIGATION, FR_PLAYER_MOVEMENT).

Evolution of the Architecture

Initial Design (First Structural Diagram)

- **Centralised Design**:
  - Initially, **UniSim** was the central component, encompassing major elements like **batch**, **camera**, **physicsWorld**, **playerStats**, **buildingWindow**, and **hudElements**. It handled all key operations, from rendering to managing resources.
- **Issues**:
  - **Tightly Coupled System**: The centralization resulted in a tightly coupled architecture, making it challenging to scale or modify parts of the system without affecting others (NFR_INTUATIVE_CONTROLS).
  - **Lack of Modularity**: There was minimal separation between game logic and different components, resulting in a complex and difficult-to-manage codebase (UR_INTUITIVE, FR_INSTRUCTIONS).

For a more detailed view of the initial structural diagram, please refer to the design process documentation available on our team's website: https://eng1website1.netlify.app/design.

Second Structural Diagram (Toward Modularity)

- **Introduction of Modular Elements**:
  - **UniSim** retained its core attributes like **physicsWorld** and **camera**, but some of its responsibilities were offloaded to newly introduced components.
  - **BuildingList** was introduced to handle building management, allowing **UniSim** to focus on core game functions while **BuildingList** managed building logic (FR_BUILDINGS, FR_BUILDING_PLACEMENT).
  - The **Hud** became a distinct class managing user interface elements, such as displaying **PlayerStats** as required in FR_STATS and UR_VARIABLES. This led to improved separation of concerns, facilitating independent modifications.
- **Key Improvements**:
  - **Reduced Responsibility for UniSim**: Delegating responsibilities for **Hud**, building management, and assets to specific modules decreased the load on **UniSim**, enhancing code maintainability.
  - **Increased Abstraction**: By introducing **BuildingList** and **Hud** as independent classes, the structure became more modular, allowing easier expansion and modification.

For a more detailed view of the second structural diagram, please refer to the design process documentation available on our team's website: https://eng1website1.netlify.app/design.

Final Structural Diagram (Fully Modular and Scalable Design)

- **Separation of Screens**:
  - The final diagram introduced independent screens such as **LandingScreen**, **GameScreen**, **CreditsScreen**, **HowToPlayScreen**, and **SettingScreen**. Each screen manages specific user experiences, simplifying the **UniSim** class and enhancing navigation flexibility (UR_NAVIGATION, UR_INTUITIVE).

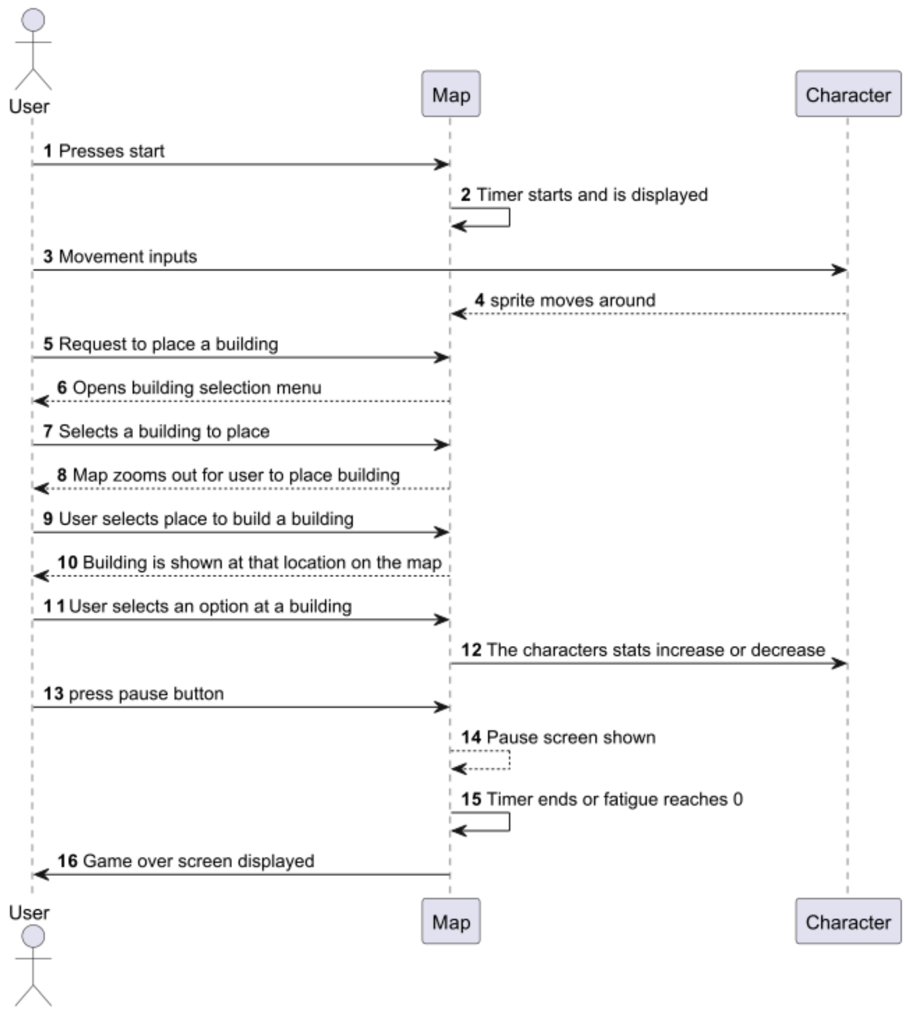
    - Each screen contains its own attributes and methods, improving readability and enabling easy modifications without affecting other screens (FR_INSTRUCTIONS, NFR_INSTRUCTIONS).
- **Building System Evolution**:
    - The **BuildingManager** class was refined to manage specialised subclasses such as **Academic**, **Food**, **Recreational**, **Accommodation**, and **Workplace** to represent the different buildings needed for the FR_BUILDINGS AND UR_ADD_BUILDINGS requirement. Each subclass has unique attributes, providing flexibility for managing different in-game buildings and the activities performed at them required by FR_ACTIVITIES and UR_INTERACT.
    - The **Placed** class was introduced to represent placed buildings, adding modularity to building placement and interactions (FR_BUILDING_PLACEMENT, NFR_BUILDING_RESTRICTIONS).
- **Player Interaction and Stats**:
    - Modular classes for handling player interactions were introduced, including **Character**, **SpeechBubble**, **PlayerStats**, **StatsLabels**, and **Hud**.
    - The **Hud** aggregates **PlayerStats** and **StatsLabels**, providing a clear separation between gameplay logic and UI rendering, supporting UR_VARIABLES.
    - **Character** handles movement (FR_PLAYER_MOVEMENT) and interactions independently of the **GameScreen**, while **SpeechBubble** offers visual cues for player interactions, enhancing the overall experience (UR_INTERACT).
- **Pause and Settings Management**:
    - The **PauseMenu** (FR_PAUSE_MENU) and **SettingsScreen** were designed to manage their respective tasks independently, which helped create a more user-friendly experience without cluttering **GameScreen** (UR_PAUSE) . The pause menu also allowed a screen to be added with game instructions to fulfil the FR_INSTRUCTIONS requirement.

Behavioural Diagrams

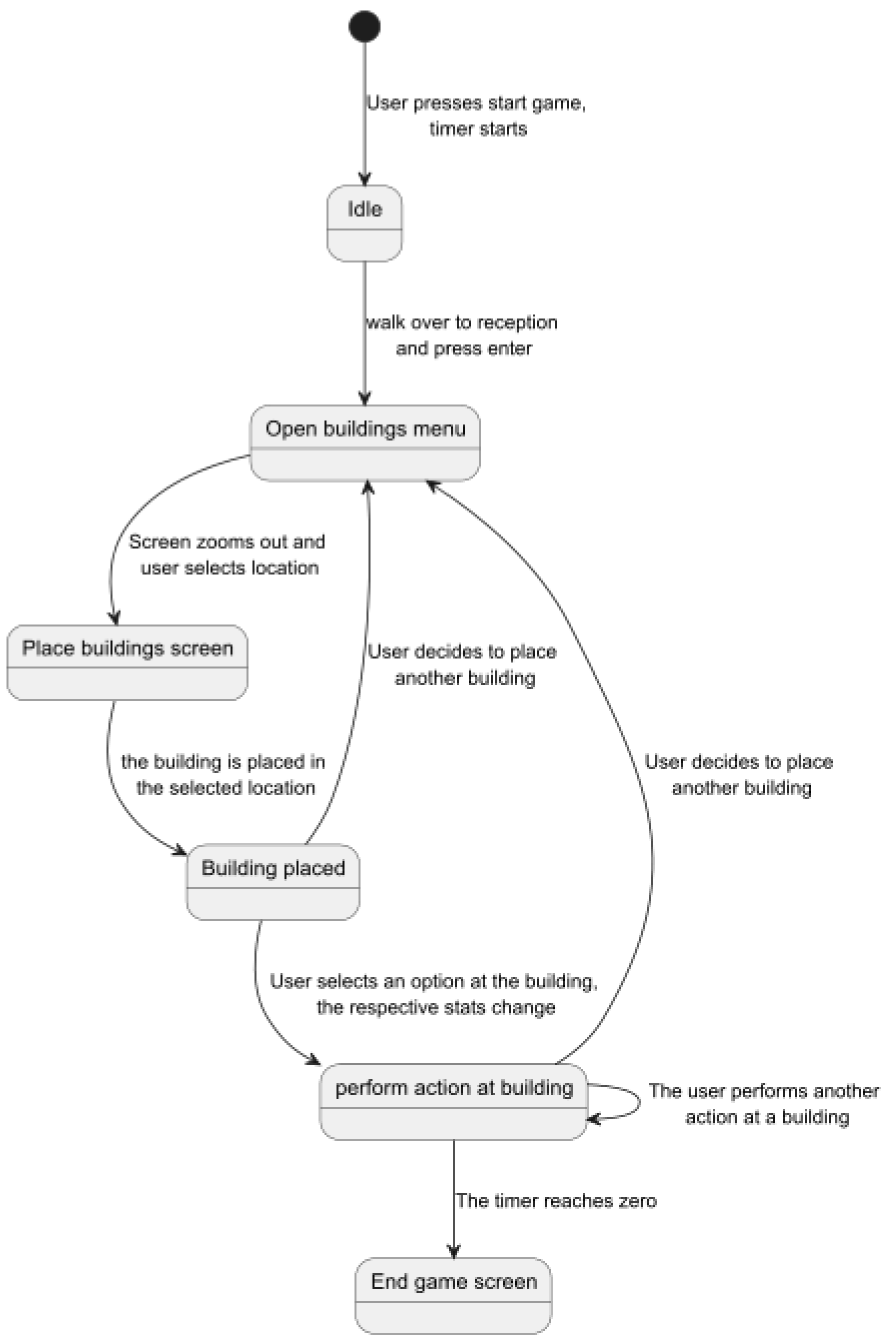
We also developed behavioural diagrams using **PlantUML** in **IntelliJ** to demonstrate the flow of the game as it is being played and describe how the functions interact with each other. These diagrams include:

- **Sequence/activity diagram**: To show how various objects within the system communicate changes as the game is being played.
- **State diagram**: To capture the different system states and how the system moves between them.

We decided on the behavioural flow of the game in a meeting after we had gathered the requirements from the client. Once we had a list of requirements and features needed for the game we were able to come up with a basic structure of how we wanted it to work. We stuck to this structure throughout the process of making our game.


User
Map
Character
1 Presses start
2 Timer starts and is displayed
3 Movement inputs
4 sprite moves around
5 Request to place a building
6 Opens building selection menu
7 Selects a building to place
8 Map zooms out for user to place building
9 User selects place to build a building
10 Building is shown at that location on the map
11 User selects an option at a building
12 The characters stats increase or decrease
13 press pause button
14 Pause screen shown
15 Timer ends or fatigue reaches 0
16 Game over screen displayed
User
Map
Character

The above activity diagram shows how the users actions affect both the game map and the character and how these changes affect each of these objects.


User presses start game, timer starts
Idle
walk over to reception and press enter
Open buildings menu
Screen zooms out and user selects location
Place buildings screen
User decides to place another building
the building is placed in the selected location
User decides to place another building
Building placed
User selects an option at the building, the respective stats change
perform action at building
The user performs another action at a building
The timer reaches zero
End game screen

The above state diagram shows the different states of the system and how the game transitions between each one. The user should be able to perform the main gameplay actions of placing buildings and interacting with those buildings (UR_INTERACT) at any time as shown in the state diagram.