Architecture

Group 8: GeNext

Aditya Mydeo

Alex Miles-Thom

Christopher Omoraka

Faisal Nabi

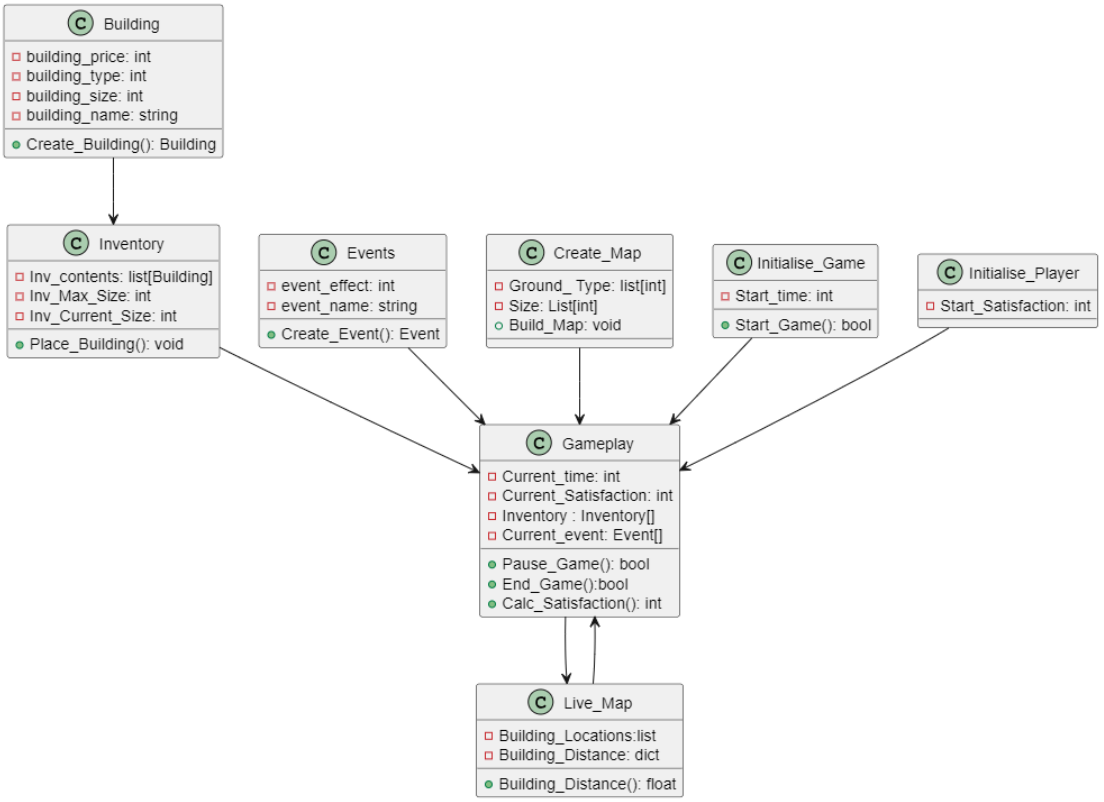
Jesly Prosper

Stefan Maxwell

Zak McGuire

After meeting with our key stakeholder- Tommy Yuan and eliciting the user requirements, we decided to use these requirements to design the basis of our game. To do this we decided to use Plant UML on VScode to illustrate our design. This is because it is industry standard and the most familiar tool we knew how to use. To help the implementation team we devised a general idea of how we want the user requirements to be met by our code. This alignment of design elements with requirements helped ensure that every component of the architecture served a clear purpose within the gameplay experience.

First Interim Version of Class Diagram



https://morako1.github.io/unisim/

To begin with, the **Building** class represents the types of structures that players can place in the game. It includes attributes such as building_price and building_type allowing the system to handle various characteristics of the building. This design satisfies UR_BUILDING_TYPE by allowing different building types and sizes. Additionally FR_BUILDING and FR_BUILDING_NAMES, are satisfied by the game allowing up to 4 building types.

Next, the **Inventory** class manages all buildings that have been created and placed by the player using different attributes such as Inv_contents. This class directly supports the UR_BUILDING_COUNTER and FR_BUILDING_COUNTER_DISPLAY by maintaining a count of placed buildings. The Place_Building method enables the addition of buildings to the inventory, ultimately enhancing the user experience.

The **Events** class is responsible for handling events that occur in the game and directly can affect the satisfaction score. it includes attributes such as event_effect and event_name to define the characteristics of the event. This class fulfills the requirements: UR_REACT_TO_EVENTS and FR_EVENT_REACTION by making the system generate and manage various types of events that the user can respond to. THe Create_Event method

generates various events which aligns with the requirement FR_EVENT and FR_EVENT_ANNOUNCEMENT.

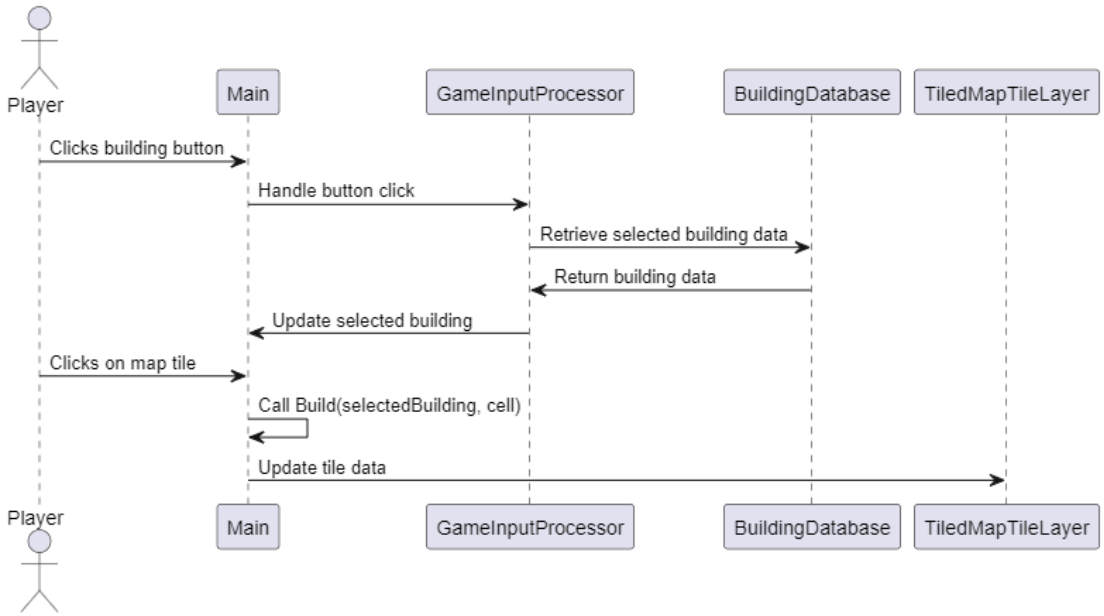
Additionally, **Create_map** class is used to generate the campus map providing the basic framework for the user. It contains attributes such as Ground_Type and Size that defines the layout and boundaries of the map, ultimately meeting the requirement UR_CAMPUS_MAP by creating an accessible map with clear boundaries. The Build_Map method constructs the game's environment ensuring that all players actions are within limits, satisfying the FR_GAME_BOUNDARIES.

The **Initialise_Game** class generates the initial setup for the games including a timer and starting the game. This setup allows the game to begin in a controlled manner aligning with UR_GAME_TIMER as a countdown begins which is clearly displayed to the player.

Initialise_Player class is responsible for establishing the players satisfaction score level with the Start_Satisfaction attribute. This sets a baseline for the player score at the start. This contributes to UR_SATISFACTION_SCORE by creating a starting point for the score which can then be modified later on.

Finally, the **GamePlay** class integrates the key elements of gameplay such as time, satisfaction, inventory and events. This supports many requirements including: UR_GAME_TIMER,FR_DISPLAY_TIMER,UR_SATISFACTION_SCORE and FR_DISPLAY_SATISFACTION. Additionally, methods like Pause_Game and End_Game address UR_PAUSE_GAME and FR_PAUSE_FUNCTION_BUTTON, by allowing used to pause and end the game at any point.

Sequence Diagram



https://morako1.github.io/unisim/
A Sequence diagram was developed to visualize the flow of the player when they selected and placed a building on the tiled map in game. The Sequence diagram coded in UML to illustrate the interactions of objects in an specific order and is focused on the steps that is involved in building placement and the actions of the player to update the map.
Elements in the Sequence diagram include:

Player: the user that is interacting with the UniSim game.
Main: The games primary source of input handling, game logic
GameInputProcessor: The manager of input processing from the player's inputs.
BuildingDatabase: Contains information about the different types of building in the game
TiledMapLayer: the map layer, where buildings are placed.

Sequence Flow

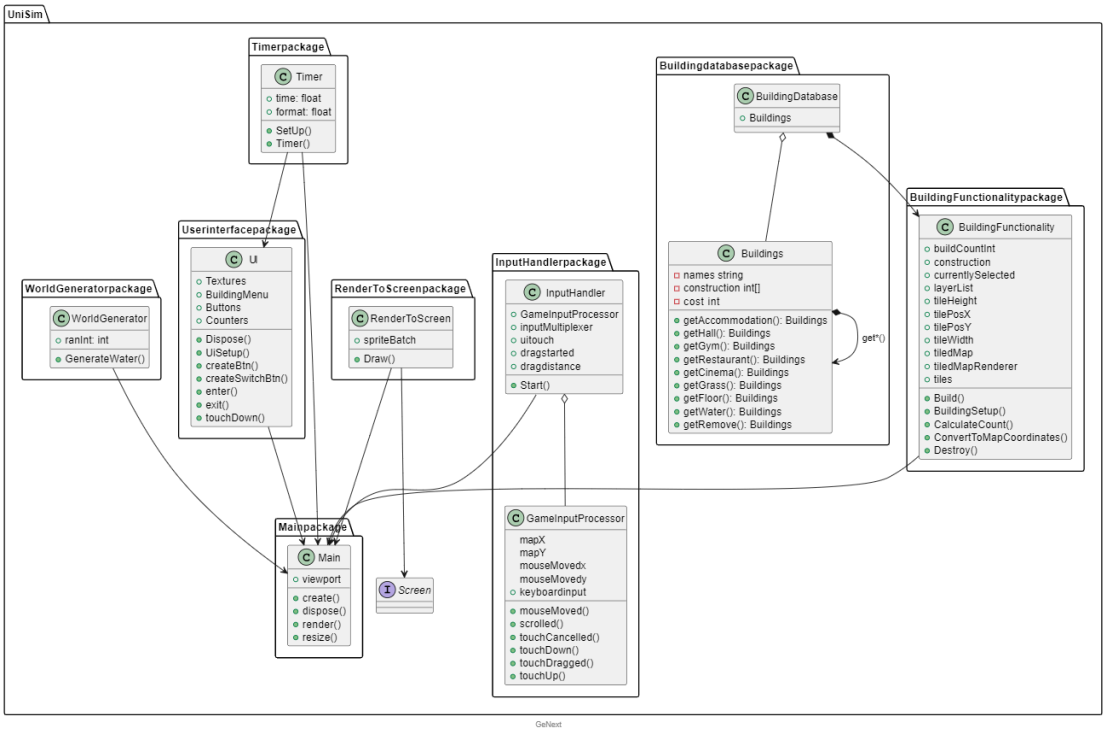
1. The player initiates the building placement by clicking a button in the game interface to select a building.
2. The main receives this click and interprets it to select a building for placement.
3. main determines which building type has been selected.
4. Main communicates with GameInputProcessor and retrieves the relevant building data
5. GameInputProcessor calls Building Data to get information about the selected building.
6. BuildingDatabase sends back the building data to the GameInputProcessor.
7. the player clicks on a specific tile on the map where they want to place a building.
8. Main component registers this tile and places the selected building.
9. Main component calls the Build function, passing selectedBuilding and map tile as parameters. to initiate the building placement on the map.
10. This checks if the placement is valid (e.g., within map boundaries or without overlapping another building) before confirming the placement.
11. TiledMapTileLayer updates the tile data to display the placed building on the map

There are few functional requirements that are implemented in the diagram:
FR_DRAG_AND_DROP is the interaction for building placement.

FR_GAME_BOUNDARIES ensures placement only in valid map areas like the grass on ground.

FR_BUILDING_COUNTER_DISPLAY: after placing a building, the building counter will update

Final Class Diagram



https://morako1.github.io/unisim/

Over the course of time we decided to change several things to improve our game. To begin with we made use of packages such as WorldGeneratorpackage, Userinterfacepackage, InputHandlerpackage, Buildingdatabasepackage, and BuildingFunctionalitypackage. Each of these packages contain classes that have specific roles leading to easier maintainability due to each part of the code being responsible for one specific task. For example, Buildingdatabasepackage now handles the building data, which includes classes like **Buildings** and **BuildingDatabase**. This allows the system to manage building data independently of other components, ensuring better scalability and alignment with our requirements such as UR_BUILDING_TYPE and FR_BUILDING.

Another key difference is the inclusion of the WorldGeneratorpackage, which contains the WorldGenerator class and its GenerateWater method. This addition introduces procedural world generation capabilities, allowing for different and new maps to be generated each time. This ultimately enhances user experience and gameplay. This feature supports requirements relating to UR_CAMPUS_MAP and FR_GAME_BOUNDARIES, by allowing boundaries to adapt to the different terrain each game.

The new architecture also includes an input management system with the InputHandlerpackage and the **GameInputProcessor** class, which specifically handles different user input like mouseMoved, touchDown and touchUp. This package improves response handling for user interactions ultimately making the gameplay more intuitive. This directly supports UR_PAUSE_GAME and UR_PLAY_GAME by allowing consistent and efficient input handling.