American University of Beirut
Faculty of Arts and Science
Department of Computer Science
CMPS 299B - Graduation Capstone Project
Technical Documentation
Advisor: Professor Maurice Khabbaz, PhD
Members:

- **Osama Iskandarani**
- **Iyad Al Arab**
- **Harout Babikian**
- **Maurice Salameh**
- **Sara Najjar**

May 20th, 2024

A Smart and Efficient IoT-Assisted Digital Twin Building Administration, Control and Monitoring Solution: AUB Bliss Building Case Study

The following showcases the technical documentation of all project components. The Digital Twin is deployed hosted on itch.io.

ABSTRACT:

This Unity-based project develops a digital twin of Bliss Hall, aiming to provide an interactive and immersive simulation for educational and operational purposes. The core objective is to facilitate user engagement and understanding through a virtual representation that mirrors real-time environmental and operational dynamics of the physical Bliss Hall building, including lecture sessions.

Key technologies employed include Unity and Blender for 3D modeling and interactive scene management, Photon.PUN for robust multiplayer networking, and real-time data integration using sensors and web scraping in the ArduinoIDE (C++) and Selenium (Python). Unity's SceneManager facilitates seamless transitions between singleplayer, multiplayer, and administrative interfaces, enhancing user interaction. Photon.PUN supports scalable multiplayer environments and synchronized object states, ensuring consistent user experiences across sessions. Data from environmental sensors—such as temperature, humidity, and various gases—are captured in real-time,

processed, and visualized within the Unity environment to reflect the actual conditions of Bliss Hall such as the prediction of fire break out which uses a custom machine learning model coded in Python3.

Compared to other university campus digital twins, this application proves to be more portable in deployment, where redeployment requires the submission of a built Unity WebGL project on itch.io. Moreover, it is flexible in code and model modification in Unity, where the code and Unity components are hosted on the Unity Version Control System and can be modified by the developers at any time. The project's integration of real-time voice communication distinctly sets it apart from conventional digital twins, which often lack such user interaction and tend to be more limited to specific components.

Unity Documentation

Player Controller

The player controller configuration in our Unity project is composed of several key components that have been integrated to enhance both the Singleplayer and Multiplayer experiences. This configuration encompasses the character model visuals, controller, and animations, which were sourced from the ReadyPlayerMe asset package.

Integration Process

Upon importing the ReadyPlayerMe package into Unity, extensive customization was carried out using C# scripts. These scripts were tailored to align with the specific requirements of our project's interactive systems. The modifications enabled the character models to interact seamlessly with environment-specific elements such as elevators and the minimap, leveraging both local and Photon Server functionalities to support networked game settings.

Controller Setup

The control scheme is built around the conventional WASD keys for player movement, complemented by the use of the left-shift key for sprinting functionalities. Camera control is managed through mouse movements, providing a fluid and intuitive user experience. This setup ensures that player inputs are responsive and consistent across different gameplay scenarios.

Unity Tagging System

In Unity, the player characters are assigned the 'Player' tag. This tagging mechanism is crucial for the interaction logic within the game environment, as it allows for the systematic identification of players during collisions with various game structures such as walls, stairs,

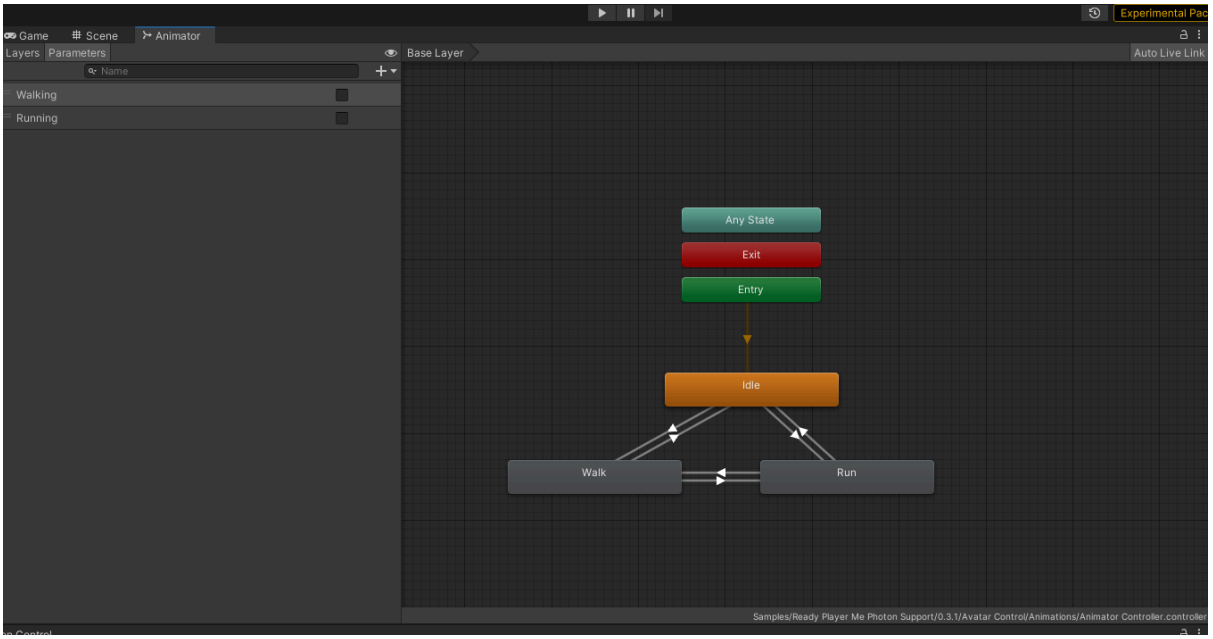
and doors. This is integral to maintaining gameplay integrity and ensuring that physical interactions within the game world behave as expected.

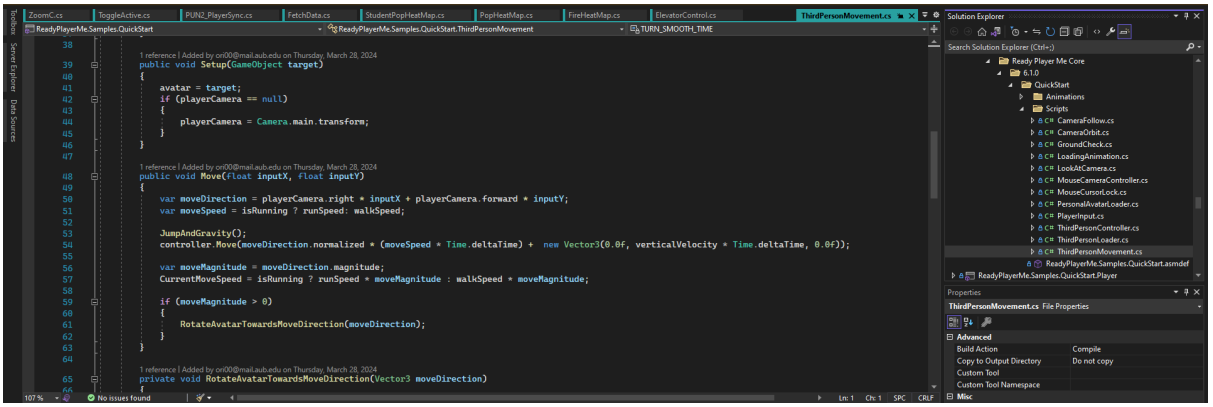
Libraries and Dependencies

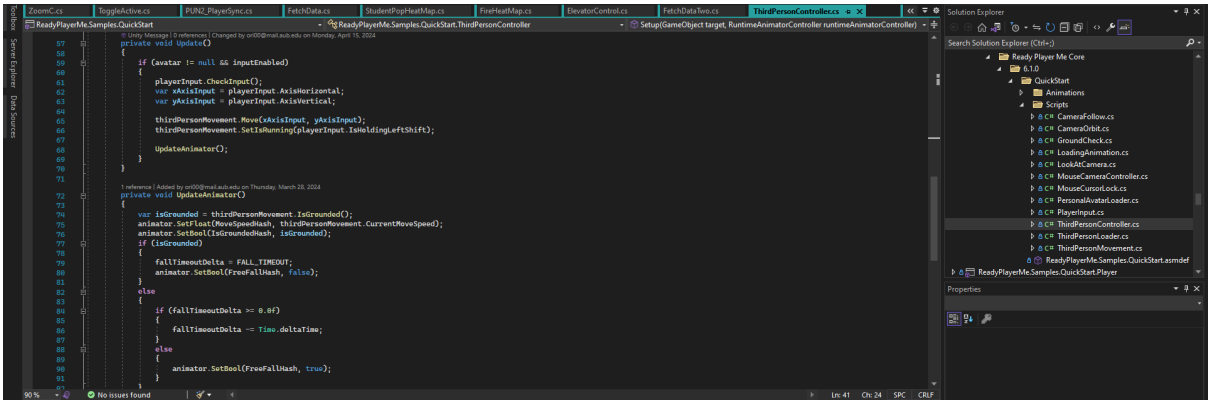
The project utilizes a robust set of Unity libraries to support the various functionalities required by the player controller. These include:

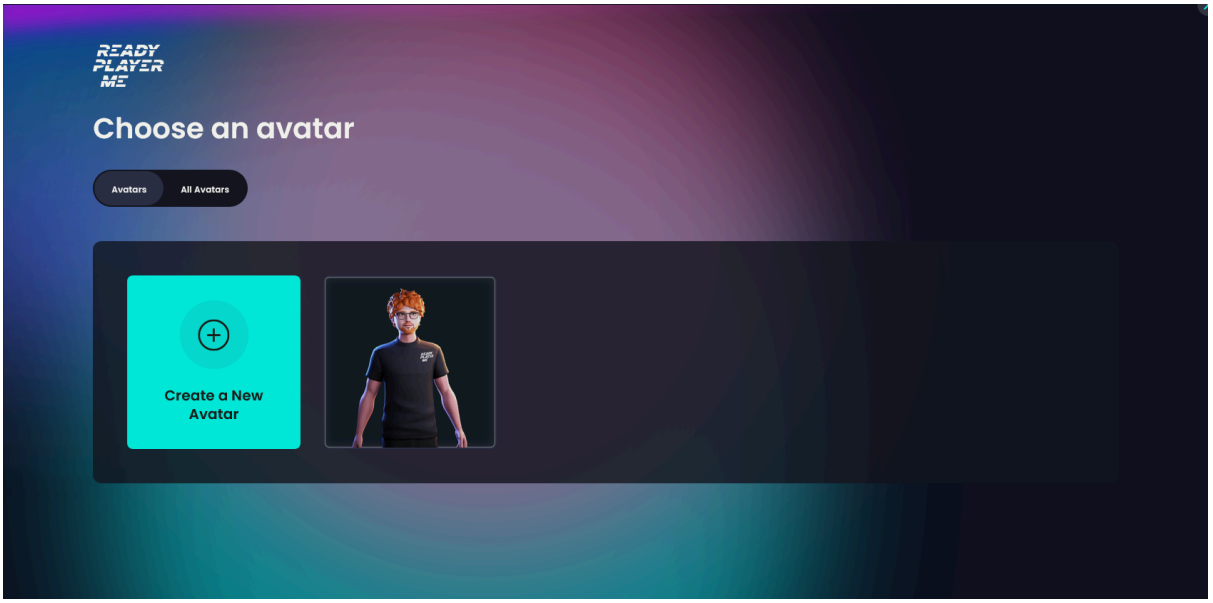
- **UnityEngine**: Used for fundamental Unity game development tasks.
- **TMPro (TextMeshPro)**: Enhances text rendering capabilities.
- **UnityEngine.Networking**: Facilitates network operations for multiplayer settings.
- **UnityEngine.UI**: Manages UI elements within the game.
- **System.Collections**: Provides classes for data structure management.
- **PhotonVoice.Pun and Photon.Pun**: These libraries from Photon are essential for implementing multiplayer capabilities, including real-time voice and data transmission.

These libraries are instrumental in providing a wide range of functionalities, from UI management to networked player interactions, thereby supporting a comprehensive and engaging game environment.


Game
Scene
Animator
Layers
Parameters
Base Layer
Auto Live Link
Name
Walking
Running
Any State
Exit
Entry
Idle
Walk
Run
Samples/Ready Player Me Photon Support/0.3.1/Avatar Control/Animations/Animator Controller.controller


ZoomC.cs
ToggleActive.cs
PUN2_PlayerSync.cs
FetchData.cs
StudentPopHeatMap.cs
PopHeatMap.cs
FireHeatMap.cs
ElevatorControl.cs
ThirdPersonMovement.cs
ReadyPlayerMe.Samples.QuickStart
ReadyPlayerMe.Samples.QuickStart.ThirdPersonMovement
TURN_SMOOTH_TIME
public void Setup(GameObject target)
{
avatar = target;
if (playerCamera == null)
{
playerCamera = Camera.main.transform;
}
}
public void Move(float inputX, float inputY)
{
var moveDirection = playerCamera.right * inputX + playerCamera.forward * inputY;
var moveSpeed = isRunning ? runSpeed: walkSpeed;
JumpAndGravity();
controller.Move(moveDirection.normalized * (moveSpeed * Time.deltaTime) + new Vector3(0.0f, verticalVelocity * Time.deltaTime, 0.0f));
var moveMagnitude = moveDirection.magnitude;
CurrentMoveSpeed = isRunning ? runSpeed * moveMagnitude : walkSpeed * moveMagnitude;
if (moveMagnitude > 0)
{
RotateAvatarTowardsMoveDirection(moveDirection);
}
}
private void RotateAvatarTowardsMoveDirection(Vector3 moveDirection)
Solution Explorer
Search Solution Explorer (Ctrl+;)
Ready Player Me Core
6.1.0
QuickStart
Animations
Scripts
CameraFollow.cs
CameraOrbit.cs
GroundCheck.cs
LoadingAnimation.cs
LookAtCamera.cs
MouseCameraController.cs
MouseCursorLock.cs
PersonalAvatarLoader.cs
PlayerInput.cs
ThirdPersonController.cs
ThirdPersonLoader.cs
ThirdPersonMovement.cs
ReadyPlayerMe.Samples.QuickStart.asmdef
ReadyPlayerMe.Samples.QuickStart.Player
Properties
ThirdPersonMovement.cs File Properties
Advanced
Build Action
Compile
Copy to Output Directory
Do not copy
Custom Tool
Custom Tool Namespace
Misc
No issues found


ZoomC.cs
ToggleActive.cs
PUN2_PlayerSync.cs
FetchData.cs
StudentPopHeatMap.cs
FireHeatMap.cs
ElevatorControl.cs
FetchDataTwo.cs
ThirdPersonController.cs
ReadyPlayerMe.Samples.QuickStart
ReadyPlayerMe.Samples.QuickStart.ThirdPersonController
Setup(GameObject target, RuntimeAnimatorController runtimeAnimatorController)
private void Update()
{
if (avatar != null && inputEnabled)
{
playerInput.CheckInput();
var xAxisInput = playerInput.AxisHorizontal;
var yAxisInput = playerInput.AxisVertical;
thirdPersonMovement.Move(xAxisInput, yAxisInput);
thirdPersonMovement.SetIsRunning(playerInput.IsHoldingLeftShift);
UpdateAnimator();
}
}
private void UpdateAnimator()
{
var isGrounded = thirdPersonMovement.IsGrounded();
animator.SetFloat(MoveSpeedHash, thirdPersonMovement.CurrentMoveSpeed);
animator.SetBool(IsGroundedHash, isGrounded);
if (isGrounded)
{
fallTimeoutDelta = FALL_TIMEOUT;
animator.SetBool(FreeFallHash, false);
}
else
{
if (fallTimeoutDelta >= 0.0f)
{
fallTimeoutDelta -= Time.deltaTime;
}
else
{
animator.SetBool(FreeFallHash, true);
}
Solution Explorer
Search Solution Explorer (Ctrl+;)
Ready Player Me Core
6.1.0
QuickStart
Animations
Scripts
CameraFollow.cs
CameraOrbit.cs
GroundCheck.cs
LoadingAnimation.cs
LookAtCamera.cs
MouseCameraController.cs
MouseCursorLock.cs
PersonalAvatarLoader.cs
PlayerInput.cs
ThirdPersonController.cs
ThirdPersonLoader.cs
ThirdPersonMovement.cs
ReadyPlayerMe.Samples.QuickStart.asmdef
ReadyPlayerMe.Samples.QuickStart.Player
Properties
No issues found



Data Visualization Overview

Our project displays two distinct types of data: data scraped from the AUBsis website and real-time data collected from sensors in a well-defined user-interface. These data sets are handled through different technological stacks and integrated into our system to provide dynamic and responsive visual representations, particularly within the Bliss building in Unity.

Data Acquisition and Processing

AUBsis Web Scraped Data

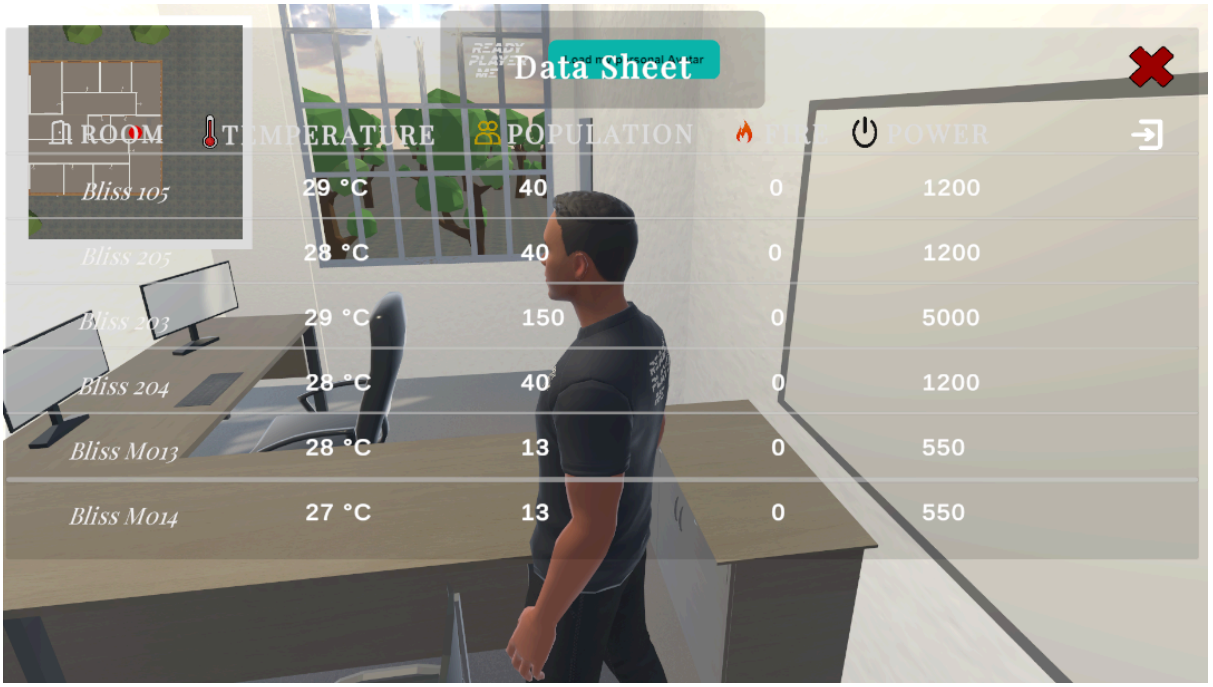
1. **Data Fetching**: The data is initially scraped from the AUBsis website using Selenium, a powerful tool for automating web browsers through Python scripts. This method allows us to programmatically navigate the website and extract data.
2. **API Integration**: Once collected, the data is accessed via a custom API. This API is structured to facilitate the fetching of the scraped data, ensuring that it can be retrieved efficiently and securely.
3. **Data Filtering and Organization**: After retrieval, the data undergoes a filtering process. This step is crucial as it organizes the data into a structured format, making it suitable for database storage and further processing. The primary objective during this phase is to prepare the data for efficient querying and integration.
4. **Database Upload**: The organized data is then uploaded to a Supabase database. Supabase, an open-source Firebase alternative, is used for its real-time capabilities and ease of use in handling CSV files. This step ensures that the data is stored securely and is readily accessible for further operations.

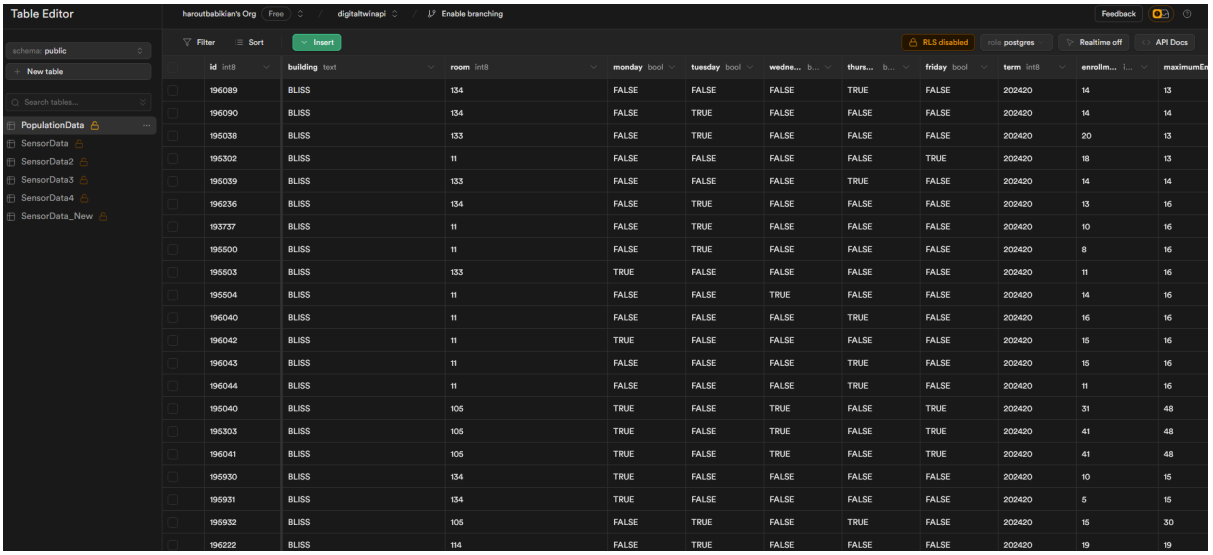
Real-Time Sensor Data

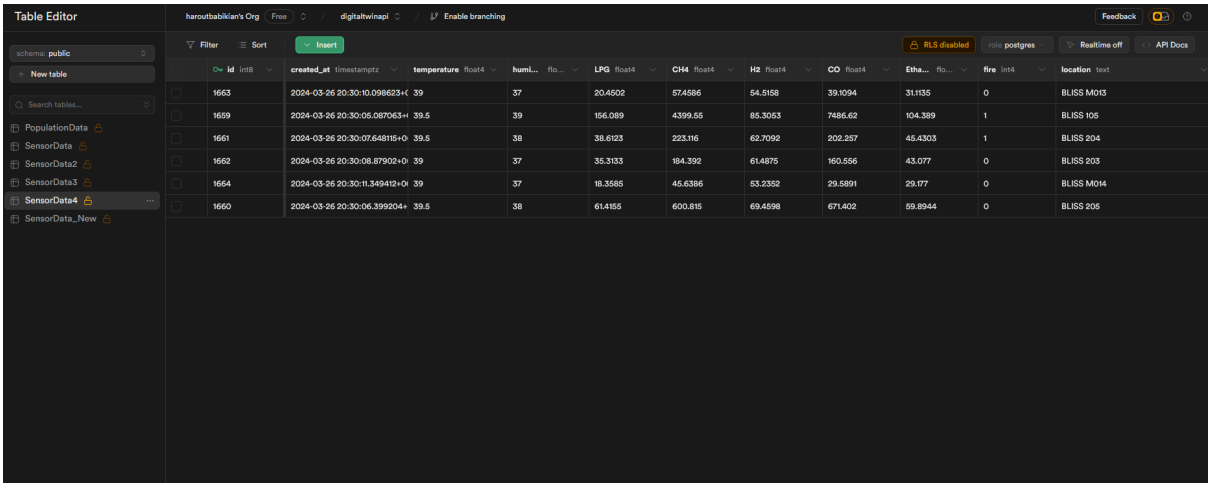
Real-time data from sensors is handled separately and is designed to provide instantaneous feedback within the application. This data is critical for applications requiring up-to-date environmental readings from various sensors located throughout the Bliss building.

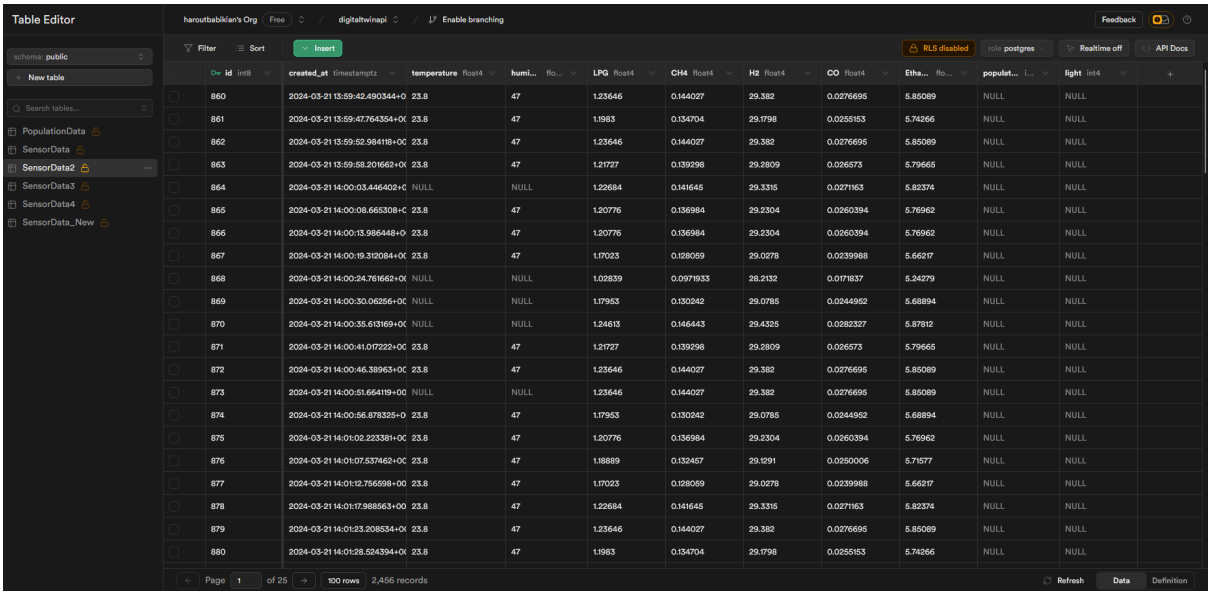
Integration and Visualization in Unity

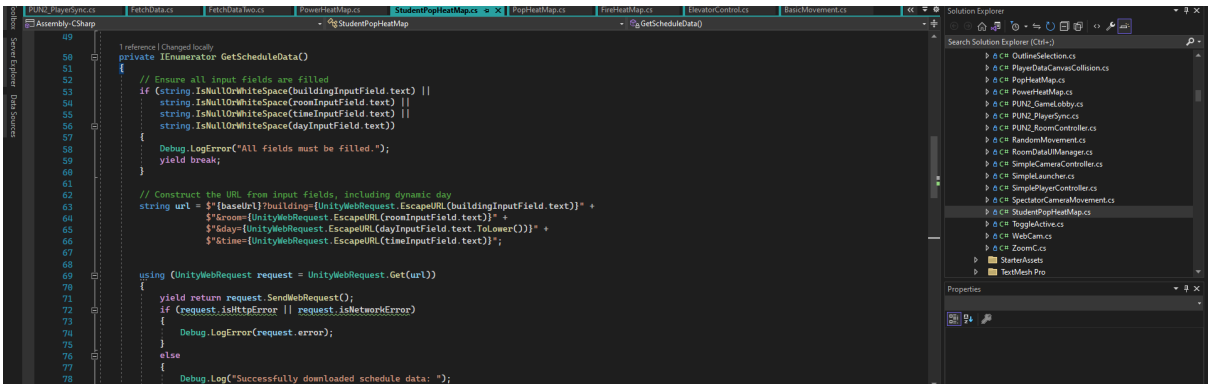
1. **Data Retrieval**: Data stored in Supabase is fetched into Unity using a JavaScript API. This API bridges the gap between the backend database and the Unity frontend, enabling the seamless flow of data into the game environment.
2. **Unity Setup**: Several endpoints are configured within Unity, specifically linked through text fields in the Unity inspector. This setup allows for dynamic assignment and updating of data values, which are displayed in corresponding rooms within the Bliss building.
3. **Data Display**: As users navigate through the virtual Bliss building in Unity, they encounter data visualizations that correspond to specific rooms. Each room displays data relevant to its function or the data collected from nearby sensors, enhancing the user's interactive experience and providing context-specific information.











Web Networking & Communication

Our project leverages the advanced networking capabilities of Photon.PUN and Unity's web request functionalities to create a robust multiplayer online experience. These components are critical for enabling real-time communication and interaction among players in a multiplayer environment.

Photon.PUN Networking

Multiplayer Online Instance

1. **Photon Server Setup**: The developer sets up a Photon Server, a powerful backend platform that facilitates online multiplayer gaming. This setup includes configuring the necessary server settings to support scalable multiplayer instances.
2. **Room Creation and Management**: Within the Unity environment, the `PhotonNetwork` class from the Photon.PUN library is used to manage online rooms. It allows for the creation of new rooms, which players can join to participate in multiplayer sessions. This is fundamental for organizing players into groups and managing game sessions effectively.
3. **Player Instantiation**: The PhotonNetwork class also handles the instantiation of player entities in the multiplayer environment. When new players join a room, their player instances are created online, ensuring each player has a unique presence in the game world.

Object Synchronization

- **Photon View**: This component is crucial for maintaining consistent object states across the network. By using `Photon View`, the project ensures that all players perceive the same state for any synchronized object within the game, thus maintaining coherence and synchronization in the multiplayer environment.

Photon Voice Integration

Real-Time Voice Communication

1. **Framework Extension**: Photon Voice Pun extends the Photon Unity Networking framework to include voice communication functionalities. It allows for the real-time transmission and reception of audio, enabling players to communicate via voice within multiplayer sessions.
2. **Setup and Configuration**:
    - **Photon Voice Package**: The initial step involves importing the Photon Voice package into Unity, which provides the necessary tools and components for voice communication.
    - **Server Configuration**: Developers need to configure the Photon Server settings specifically for voice communication by including the App ID for Voice. This links the project with Photon's voice services.
    - **PhotonVoiceView Component**: This component is attached to player GameObjects within Unity. It handles the capturing of microphone input from the player's device and the transmission of this audio to other players. Additionally, it plays incoming audio streams from other participants in the game.
3. **Data Handling**: Voice data management follows similar principles as other Photon synchronized objects. The infrastructure provided by Photon’s cloud services ensures efficient handling, synchronization, and transmission of voice data across the network.

FREE 100 CCU for development and launch with a Fusion or Quantum app available.
Activate **one app per user** in your dashboard using **+/- CCU**.

Chat 20 CCU

App ID: 24c09d39-8 ...

Chat

Public

Traffic used: **0.0 GB** of 10.0 GB

0%

ANALYZE | MANAGE | -/+ CCU

Pun 20 CCU

App ID: d7c2caf3-4 ...

Pun

Public

Traffic used: **0.0 GB** of 60.0 GB

0%

ANALYZE | MANAGE | -/+ CCU

Voice 20 CCU

App ID: 745110fe-3 ...

Voice

Public

Traffic used: **0.0 GB** of 60.0 GB

0%

ANALYZE | MANAGE | -/+ CCU

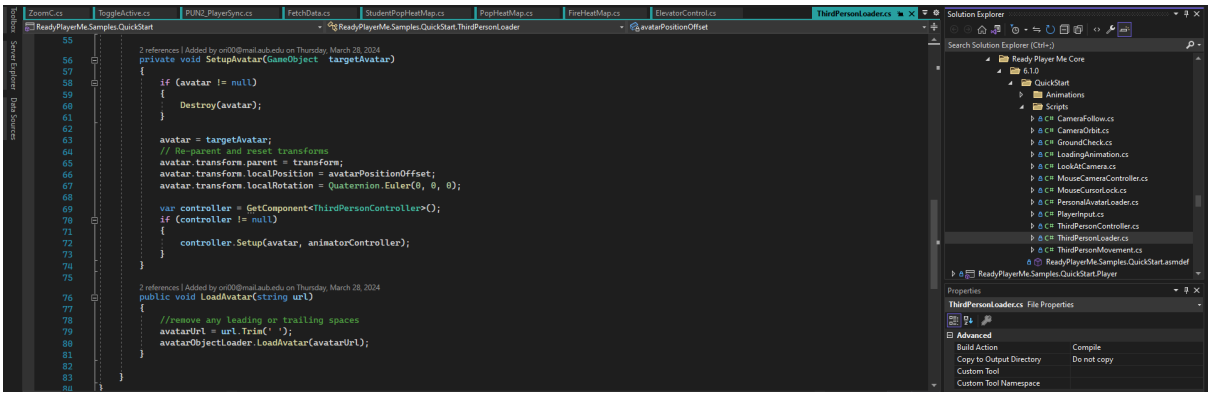
Projects

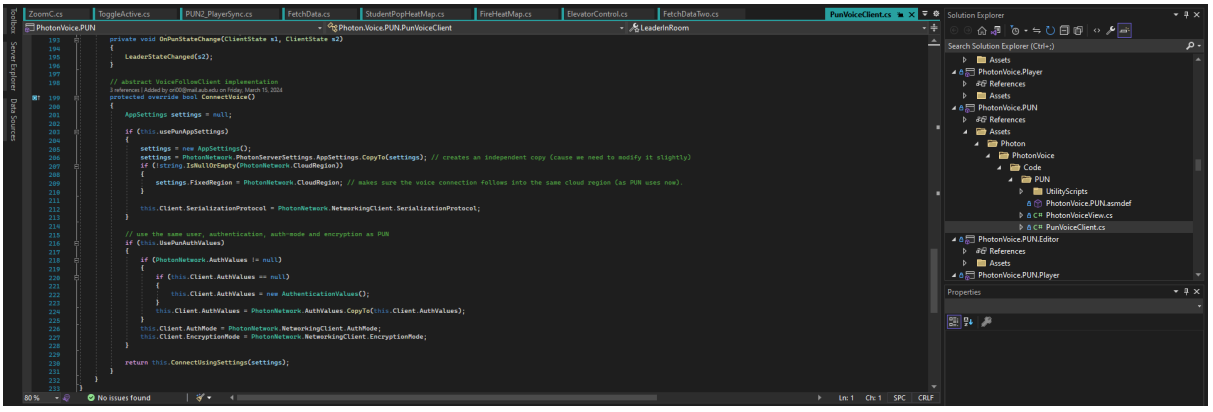
Applications

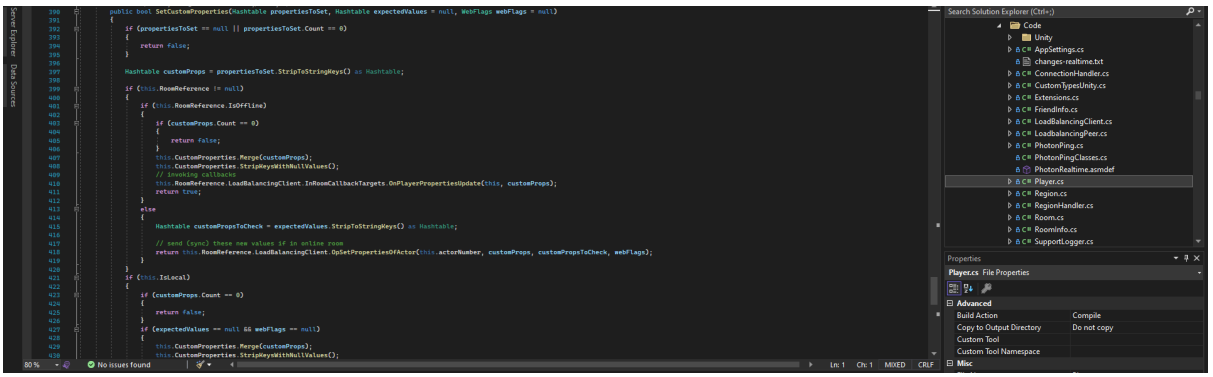
- Public Cloud
- Premium Cloud
- Self-Hosted

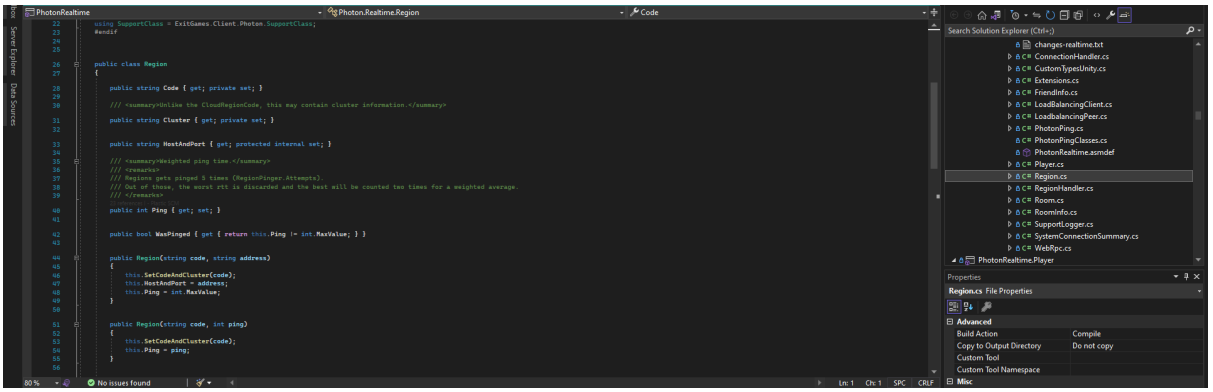
Circle Membership

Account









Data Flow Architecture

General Course Scheduling Columns

1. **id (bigint)**: A unique identifier for each record, automatically generated by the system. This field serves as the primary key in the database table.
2. **term**: Describes the academic term or semester to which the data pertains.
3. **monday** through **friday**: Each column (monday, tuesday, wednesday, thursday, friday) represents the days of the week and indicates whether the course occurs on these days, typically marked by the presence or absence of a specific scheduling notation or time slot.
4. **enrollment (integer)**: Current number of students enrolled in the course.
5. **maximumEnrollment (integer)**: The maximum number of students that can enroll in the course, defining its capacity limit.
6. **courseNumber**: A unique number assigned to the course, used for identification purposes within the academic institution.
7. **subject**: The academic subject or discipline to which the course belongs.
8. **sequenceNumber**: A number that may represent additional identifiers within a course's structure, possibly used for different sections or similar differentiations.
9. **beginTime**: The scheduled start time for the course.
10. **endTime**: The scheduled end time for the course.
11. **room**: The room number or identifier where the course takes place.
12. **building**: The building name or code where the course is located.
13. **facultyName**: The name of the faculty member teaching the course.
14. **facultyEmail**: The email address of the faculty member teaching the course.

Environmental Data Columns

1. **created_at (timestamp with time zone)**: The timestamp when the record was created, set to the current time by default.
2. **temperature (real)**: Real-time temperature data, potentially measured in Celsius or Fahrenheit.
3. **humidity (real)**: Real-time humidity level data, typically expressed as a percentage.
4. **LPG (real)**: Real-time data indicating the concentration of Liquefied Petroleum Gas in the air.
5. **CH4 (real)**: Real-time data for methane (CH4) concentration levels.
6. **H2 (real)**: Real-time hydrogen (H2) concentration levels.
7. **CO (real)**: Real-time carbon monoxide (CO) concentration levels.
8. **Ethanol (real)**: Real-time ethanol vapor concentration levels in the air.
9. **fire (integer)**: An integer indicating the presence or status of fire; could be used as a boolean or a status indicator.
10. **location (text)**: Text description or identifier of the location where these environmental readings are being taken.
11. **power (real)**: Real-time data indicating the power consumption or power level, initialized to '0' by default.
12. **population (integer)**: Indicates the number of people or the population count in the vicinity, initialized to 0 by default.

Each column is structured to efficiently capture and represent specific data relevant to either academic course scheduling or environmental monitoring, integral for database management and data analysis purposes.

UI & Scene Management

The project incorporates a robust scene management system to facilitate navigation between different modes and functionalities, including Singleplayer, Multiplayer, and Admin modes. This system is built using the Unity SceneManager library, which is part of Unity's C# scripting API.

Scene Setup

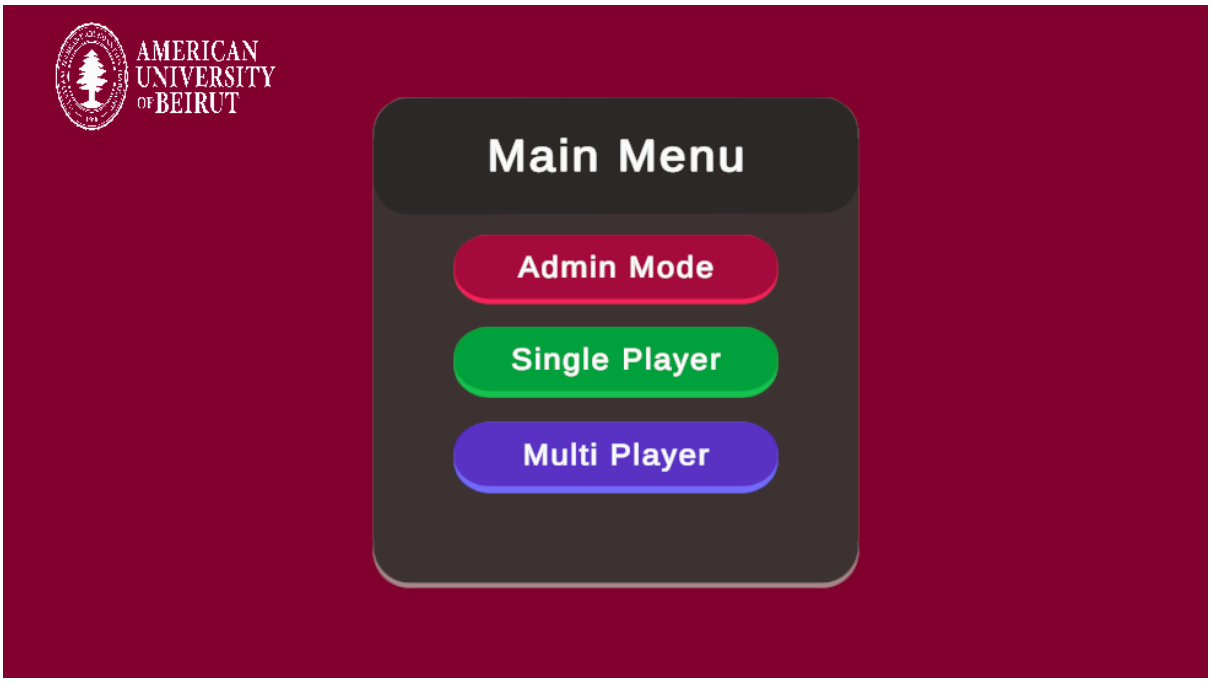
1. **Scenes Configuration**:
    - **Singleplayer**: This scene allows players to engage with the game in a solo mode, where they can explore or complete tasks independently.
    - **Multiplayer**: In this mode, players can connect and interact with each other within a shared game environment. This scene utilizes network capabilities to synchronize player actions and game states.
    - **Admin Mode**: Specifically designed for administrators or moderators, this scene provides additional controls and insights that are not available to standard players, such as user management and gameplay analytics.
2. **Scene Management Using Unity SceneManager**:
    - The `Unity.SceneManagement` library is essential for loading and switching between scenes dynamically. It handles the instantiation of scenes based on user interactions and game logic.
    - Scene transitions are managed through scene indexing, where each scene is assigned a unique index number. This method provides an efficient way to reference and load scenes programmatically.

User Interface for Scene Navigation

1. **Main Menu**:
    - A central hub where players can navigate to different parts of the game. The main menu is the first interface presented to the user upon starting the application.
    - It features options to enter Singleplayer, Multiplayer, or Admin modes, allowing players to choose their desired gameplay experience.
2. **Navigation Controls**:
    - **Scene Switching via Mouse**: Players can navigate through the menu and select scenes using the left-click button of the mouse. This input is tracked to detect user selections within the UI.
    - **Return Mechanism**: The 'ESC' key is configured to allow players to return to the main menu from any scene. This feature is crucial for providing a quick and easy way for users to back out of a current task or mode and return to the main interface.

Implementation Details

- The UI elements such as buttons and menus are linked to scene loading functions via event listeners that respond to user inputs (e.g., mouse clicks and key presses).
- Careful consideration is given to the UI layout and interaction design to ensure a seamless and intuitive navigation experience for all types of players.



Machine Learning Documentation

Fire Detection Component:

The American University of Beirut (AUB) currently hosts a fire alarm system to detect the breaking of fire in buildings. However, such a system is prone to a huge risk of false alarms, including at least three false fire alarms in Bliss Hall alone during the year of 2023. To solve that, we will be implementing an IoT-based machine learning solution, which would serve as an accurate smoke and fire detection tool thanks to the utilisation of a wide variety of sensors, specific machine learning model, and a huge dataset with over 60,000 records.

The dataset is already on Kaggle, and the full documentation and implementation stages (alongside all the software and hardware tools that were used) are up on Hackster.io, where the author, Stefan Blattmann, implemented a Real-time Smoke Detection with AI-based Sensor Fusion project using Neuton (for the machine learning part), and ArduinoIDE for the hardware configuration.

The dataset includes the following attributes:

- Air Temperature
- Air Humidity
- TVOC: Total Volatile Organic Compounds; measured in parts per billion (Source)
- eCO2: co2 equivalent concentration; calculated from different values like TVCO
- Raw H2: raw molecular hydrogen; not compensated (Bias, temperature, etc.)
- Raw Ethanol: raw ethanol gas (Source)
- Air Pressure
- PM 1.0 and PM 2.5: particulate matter size < 1.0 µm (PM1.0). 1.0 µm < 2.5 µm (PM2.5)
- Fire Alarm: ground truth is "1" if a fire is there
- CNT: Sample counter
- UTC: Timestamp UTC seconds
- NC0.5/NC1.0 and NC2.5: Number concentration of particulate matter. This differs from PM because NC gives the actual number of particles in the air. The raw NC is also classified by the particle size: < 0.5 µm (NC0.5); 0.5 µm < 1.0 µm (NC1.0); 1.0 µm < 2.5 µm (NC2.5);

The hardware includes the following sensors:

- Arduino Nicla Sense ME
    - Bosch BHI260AP: 6 axis IMU (3-Axis Accelerometer + 3-Axis Gyroscope) + MCU
    - Bosch BMP390: Pressure sensor
    - Bosch BMM150: Magnetometer
    - Bosch BME688: Humidity, temperature, and gas sensor (VOC)
- External Sensors
    - Bosch BMP388: Pressure sensor
    - Sensirion SPS30: Particular Matter sensor (Smoke detector)
    - Sensirion SHT31: Humidity and temperature sensor
    - Sensirion SPG30: Gas sensor (VOC)
    - GPS: For Timesync of the sensor readings

Our approach would be to use the Multi-Layer Perceptron (MLP) machine learning model to train the dataset from Kaggle (or through our own data if time permits) as it showed a better performance and smoke and fire detection accuracy compared to different models such as Linear Regression (LR) and Support Vector Machine (SVM) (Source).

By doing so, we will not only accurately simulate and display the breaking of fire (or the levels of severity that may lead to such an event), but also to send a notification that a fire or smoke will take place.

For Machine Learning, either Tensorflow (in Python) or Neuton can be used to train the data. As for the sensor configuration, it can be done using ArduinoIDE (in C/C++). However, there is no need to buy any hardware tools if we only utilise the current dataset from Kaggle. We can still get our own data using our current sensors in addition to new sensors that we can buy (i.e., ethanol gas, eCO2, pressure sensors). For now, we will use the current dataset until the new data from the sensors are ready.

To integrate the functionality and component into our Unity application, we detect the fire event through the ArduinoIDE logic, and upon the detection signal, the message is transmitted to our supabase database and is fetched into the Unity application as both a boolean and numerical values (where numerical is for analysis purposes, and boolean for displaying the room in a specific colour to indicate the break of fire). That is, to view this as a schema, the signal and data are sent from the hardware tool to the online database and are fetched in Unity using the already existing backend logic.

This component serves as a great addition to the existing fire alarm system at the university, as it allows us to more accurately detect any fire-related event that will take place.

Power Saving Documentation

Introduction

In the domain of digital twins and smart building management, efficient power monitoring is pivotal for optimizing energy usage and enhancing operational efficiency. This report explains the power monitoring aspect within an *IoT-Assisted Digital Twin: Building Administration, Control and Monitoring Solution*. This report clarifies the types of sensor data utilized, and the processing methodologies employed for processing to estimate power consumption in a monitored room through sensor data analysis.

Data

The following data has been retrieved from deployed sensors:

```
{
    "id": 1665,
    "created_at": "2024-03-26T21:30:12.349412+00:00",
    "temperature": 20,
    "humidity": 38,
    "LPG": 17.9823,
    "CH4": 46.2189,
    "H2": 52.8934,
    "CO": 28.721,
    "Ethanol": 28.799,
    "fire": 0,
    "location": "BLISS 105",
    "power": 70,
    "population": 10,
    "light": 1
}
```


This project monitors two main power consumption units: Air Conditions and Lights.

Data is reported at 10-minute intervals, while processing each 2 instances at once.

The main data fields used in the following processing methods are “light”, “created_at” and “temperature”. The AC and light units’ characteristics are hard-coded inside the processing, which contribute to calculate the estimated power consumed given a certain location.

The following is a snapshot of the data processed from sensors data:


```
var power_data :{…}  =
  {
    "used_ac_duration": 0,
    "used_ac_power": 0,
    "used_ac_cost": 0,
    "lost_ac_duration": 0,
    "lost_ac_power": 0,
    "lost_ac_cost": 0,
    "unused_ac_time": 0,
    "used_light_duration": 0,
    "used_light_power": 0,
    "used_light_cost": 0,
    "lost_light_duration": 0,
    "lost_light_power": 0,
    "lost_light_cost": 0,
    "unused_light_time": 0
  };
```

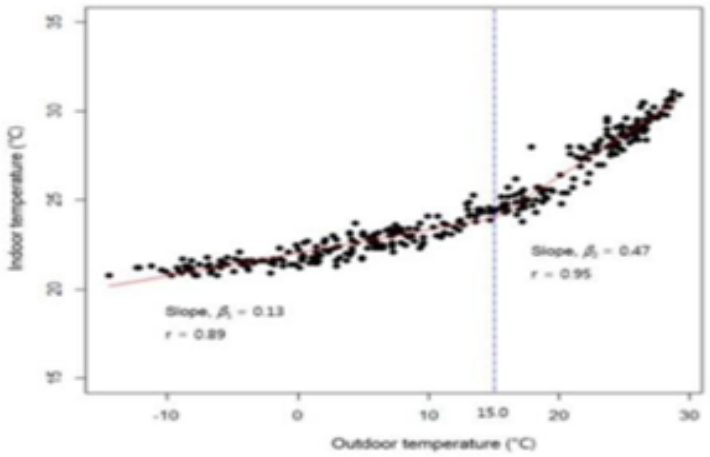

Methodologies

Air Conditioning Monitoring

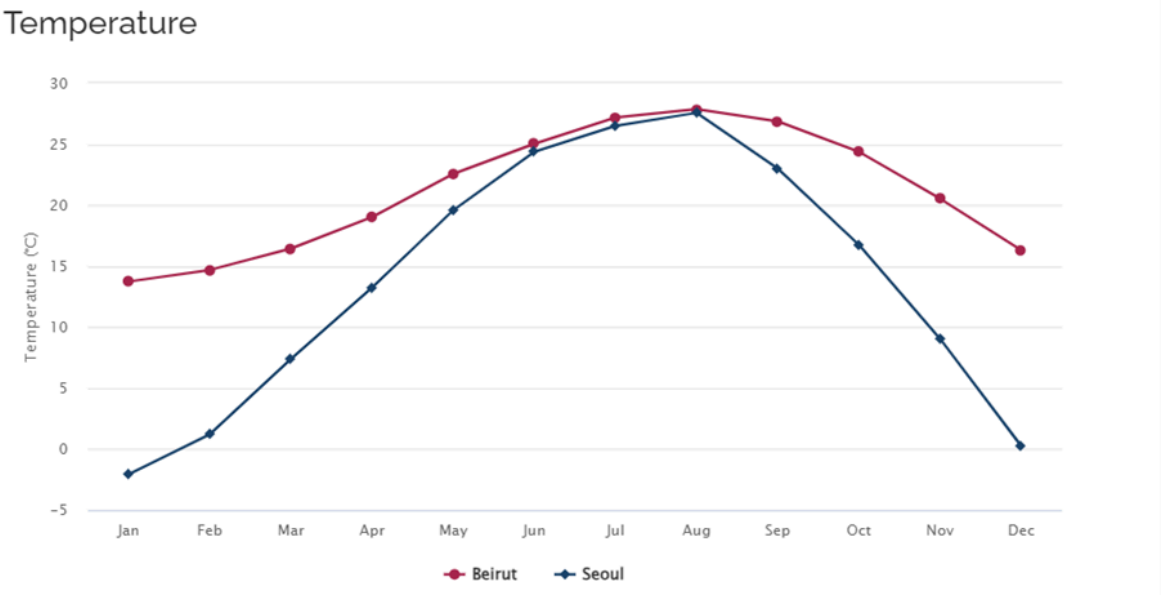
The algorithm followed consists of estimating the indoor temperature inside a room using outdoor temperature, with the help of a pre-trained machine learning model[1] which was initially designed using weather data from Seoul, Korea. The findings showed that outdoor

temperature, apparent temperature, and absolute humidity could be good indicators of indoor conditions.

Seoul has four distinctive seasons with monthly outdoor temperatures ranging from –7.2 to 26.5°C, what approximates its weather pattern quite similar to Beirut.



Given the outdoor temperature, the model estimates the indoor temperature inside the room, which would be later compared with the actual one.



With the right tweaking, the model could be made fit to estimate good enough predictions for indoor temperatures in Beirut.

A better solution would be to gather thousands of data instances from deployed sensors and train a corresponding machine learning model.

The algorithm signals that the AC are active, if there is a change in indoor temperature larger than some set threshold between two consecutive instances, or if the estimated indoor temperature is markedly different than the actual indoor temperature (fetched form the sensors).

Utilizing these methods to track at what times the units are active, the AC power consumed is deduced through:

ac_power_consumed = number_of_units * duration * ac_power_consumption;

Light Consumption Monitoring

The process of computing lights power consumption checks whether the sensors deployed in a specific room signal the presence of light inside the room. This light can be distinguished from any outside light by configuring and tuning the light sensor for the optimal threshold.

Similar to the air conditioning, the above method tracks the duration of time the lights are estimated to be active, in order for it to be able to infer the power consumed by light units.

The lost power from AC and light units is defined to be the estimated power consumed by each respective unit when the population in the room is equal to 0. The lost power cost is similarly computed to the actual power consumed:

cost = power_consumption * price_of_KWh;

---

[1] https://weatherandclimate.com/compare-beirut-and-seoul

Modelling Phase

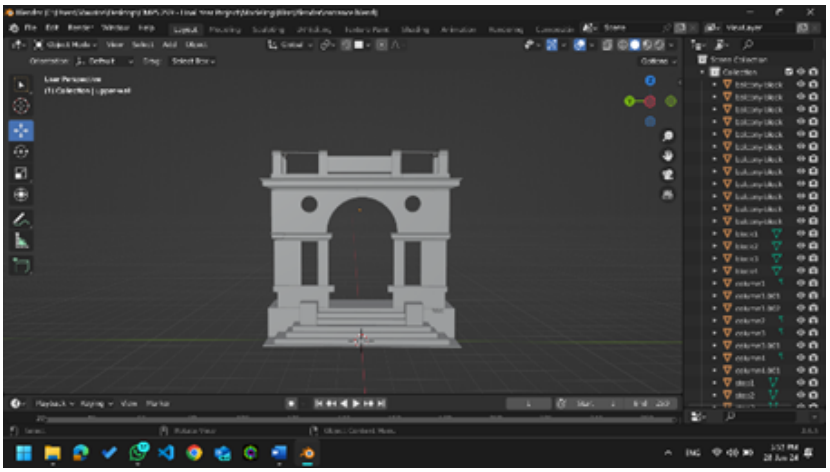
Assets Modelling

This phase includes the following steps in chronological order:

1. Choosing carefully which asset to model: Really, not any entity will add meaning to the model, so better focus on essential objects that would render the model realistic. For instance, choosing to model four different

types of doors rather than a basic one would bring no additional meaningful information to the model, and would take more time to create.

2. Get “good enough” measurements: While obtaining precise real-life measurements in our Digital Twin is certainly advantageous, our current preference leans towards approximations that mimic reality convincingly. This approach aligns more with our timeline and provides more flexibility. As a matter of fact, drawing sketches for the assets planned to be modelled with their approximated measurement, along with some pictures, helps a lot.
3. Modelling: To create the small-scale assets, we used Blender.



***Good practice side-note*:**

Files should have descriptive names, and saved preferably in 2 formats:

- **.fbx files: those are used later in assembling.**
- **.blend files: those can be opened in Blender and re-used to model something similar, in our case, we modeled many types of walls (depending on measurements) using a single blend file and exported to many fbx.**

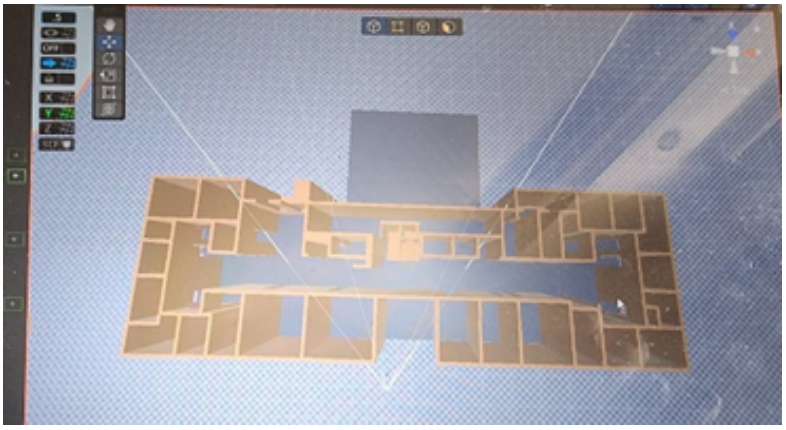
A. Assembling:

In this phase, we assemble the assets and props we created in the first stage above to construct the final model in Unity. Assets are imported into Unity as .fbx files and added to the final product.

1. **Starting Point:**

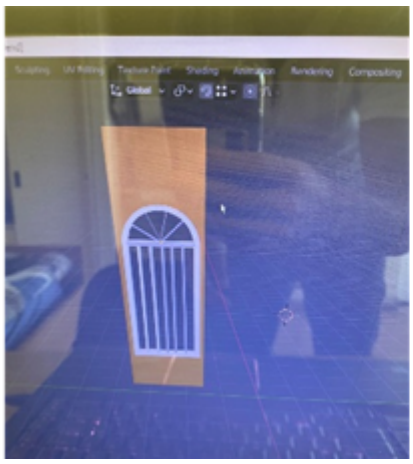
a. A good start will be to take as many pictures as you can of the building (the inside and the outside) from different angles. This helps in getting the big picture of the building's architecture.

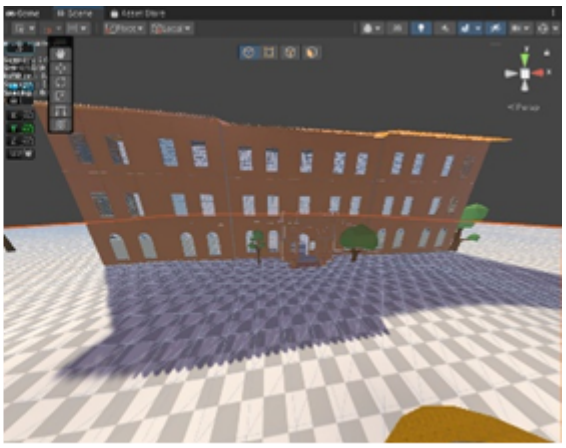
1. **Modelling:**
    a. Import the .fbx files made in Blender into the Unity Project (Import once and used multiple times). Starting with the external aspect of the building is good practice, as this will guide you with the interior architecture and its spatial division.



1. **Texturing:**
    a. The final phase includes the application of textures and colors to the assets created in Blender. This step contributes to the overall visual presentation of the model, adding vibrancy and appeal for the user.
    b. A word on the choice of colors. In our Digital Twin, we aimed for textures that closely resembled reality, but encountered the following challenges:
    c. The effort required to create realistic textures proved time-consuming and fell outside our expertise.
    d. Quality textures available in the Unity store that are actually good and fit our needs often required payment, what was not worth pursuing.

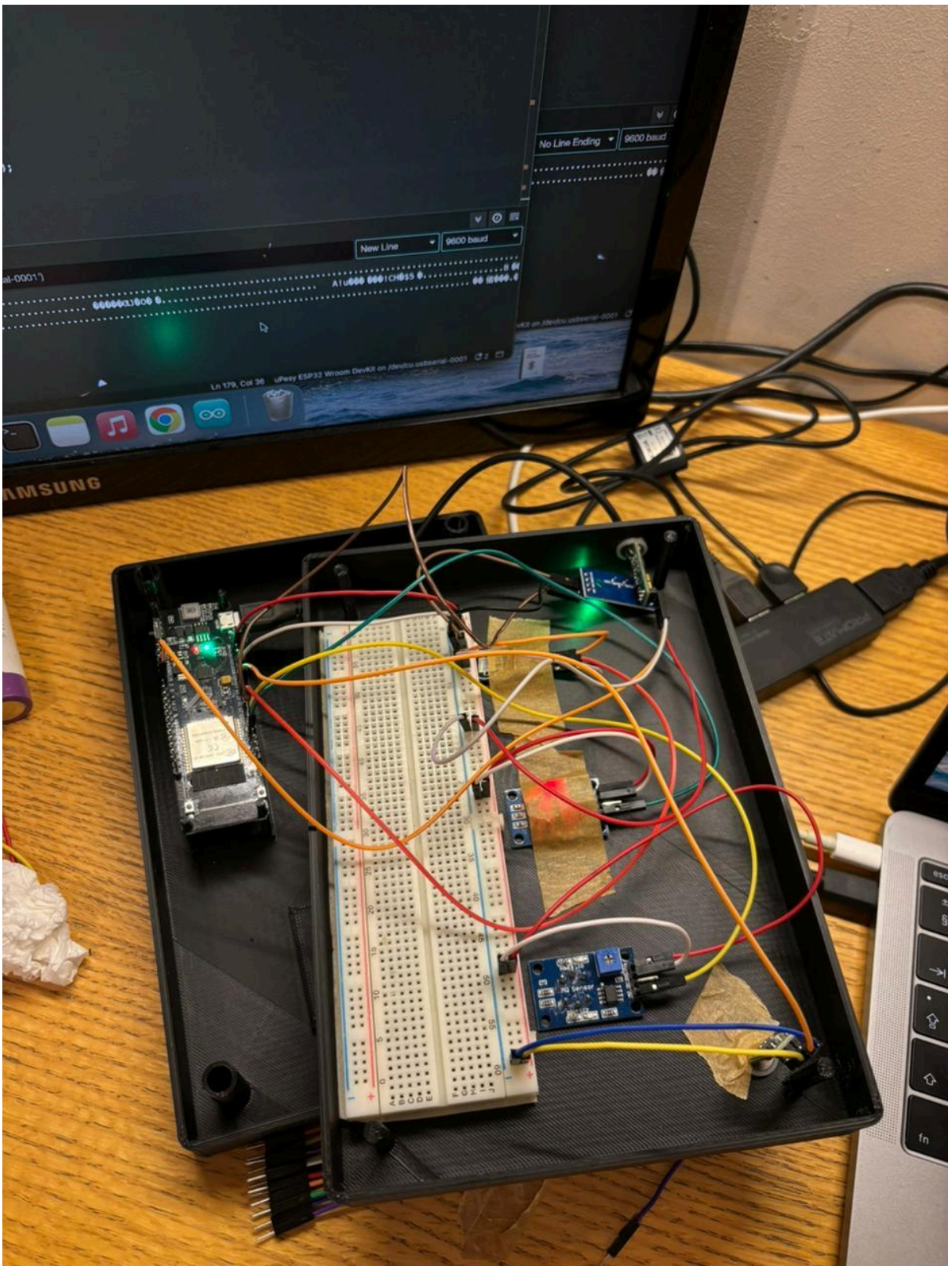
As a result, we found that simple and descriptive colors are sufficient to showcase and emphasize the distinctions in assets, as shown in the pictures.





1. **Software and Tools Utilized:**
    a. Blender: To model the assets used in constructing the larger building.
    b. Unity 3D: Where the product is assembled and built in.
    c. Unity DevOps Version Control: similar to Github that helps with collaboration, it's a version control that can offers its services for free to 3 accounts, and those include managing changes to source code, assets, and other project files over time.
    d. Alternatives: In the beginning of the development process, we were adopting Unreal Engine as our tool for modelling. It focuses a lot on unwanted graphics and rendered heavy models and textures, which was one of the main reasons we shifted to Blender x Unity.

Sensor Module Phase



This section outlines the design, implementation, and functionality of the sensor module developed for the AUB Digital Twin project. This module serves as a crucial data acquisition component, gathering real-time environmental data from Bliss Hall.

The data is then processed and transmitted to a Supabase database, contributing to the immersive and dynamic visualizations within the Unity environment.

1. Hardware Components

The sensor module is built around an ESP32 microcontroller, chosen for its powerful processing capabilities and support for wireless communication protocols like Wi-Fi. It is housed in a 3D-printed case, designed to accommodate all sensors and facilitate easy installation. The key components include:

- **ESP32 Microcontroller:**
  - Acts as the central processing unit, responsible for data acquisition, processing, and transmission.
  - Provides built-in Wi-Fi connectivity for seamless data transfer.
  - Includes integrated battery support, enabling autonomous operation.
- **MQ-3 Gas Sensor:**
  - Detects the presence of LPG (Liquefied Petroleum Gas) in the air.
  - Outputs an analog signal proportional to the gas concentration.
- **MQ-8 Gas Sensor:**
  - Measures the concentration of flammable gases like methane (CH4) and ethanol.
  - Provides an analog output corresponding to the detected gas levels.
- **DHT11 Temperature and Humidity Sensor:**
  - Measures ambient temperature and humidity levels.
  - Offers a simple digital interface for easy data acquisition.
- **PIR Motion Sensors (2):**
  - Detect movement within their field of view, indicating room occupancy.
  - Generate a digital signal when motion is detected.
- **Light Sensor:**
  - Measures the ambient light intensity.
  - Provides an analog output proportional to the light level.

2. Data Acquisition and Processing

The sensor module continuously collects data from each sensor. The ESP32 chip processes this raw data through the following steps:

- **Gas Sensor Data Processing:**
    - Raw Data Acquisition: The analog signals from the MQ-3 and MQ-8 sensors are read by the ESP32 using its analog-to-digital converter (ADC).
    - PPM Calculation: The raw analog values are converted to ppm (parts per million) concentrations using calibration data specific to each sensor type. This involves a process of mapping the raw analog output to a specific ppm value using predefined curves or algorithms. The process involves using pre-calculated calibration curves or algorithms, specific to the MQ-3 and MQ-8 sensors, to convert the raw analog data into meaningful ppm values. These curves are often supplied by the sensor manufacturer.
- **Temperature and Humidity Data Processing:**
    - Digital Data Acquisition: The DHT11 sensor provides digital data for temperature and humidity.
    - Data Interpretation: The ESP32 reads the digital data from the sensor and interprets it to extract the actual temperature and humidity values.
- **Motion Sensor Data Processing:**
    - Digital Data Acquisition: The PIR sensors output a digital signal when motion is detected.
    - Data Interpretation: The ESP32 reads the digital signals from the PIR sensors, indicating the presence of movement within the room.
- **Light Sensor Data Processing:**
    - Analog Data Acquisition: The light sensor outputs an analog signal proportional to the light intensity.
    - Data Interpretation: The ESP32 reads the analog signal from the light sensor and converts it into a measurable light intensity value.

3. Room Occupancy Calculation

The ESP32 uses the PIR motion sensors to calculate room occupancy. The logic involves:

- **Motion Detection:** When a PIR sensor detects movement, the ESP32 registers a "motion detected" event.
- **Occupancy Tracking:** The ESP32 maintains a count of the number of motion events detected within a predefined time interval. This interval acts as a window for detecting motion.
- **Occupancy Update:** If the number of motion events within the interval exceeds a certain threshold, the occupancy count is incremented. Conversely,

if the interval passes without any motion events, the occupancy count is decremented.
- **Output:** The calculated occupancy count is stored as a data point, ready for transmission.

4. Data Transmission to Supabase

The ESP32 transmits the collected and processed sensor data to a Supabase database using an HTTP request:

- **Data Formatting:** The sensor data is formatted into a structured format, compatible with the Supabase schema.
- **HTTP Request:** The ESP32 sends an HTTP request to the Supabase API endpoint, including the formatted sensor data as part of the request payload.
- **Database Update:** The Supabase server receives the data, validates it against the schema, and updates the corresponding entries in the database. This ensures data integrity and consistency.

5. Supabase Database Schema

The data transmitted from the sensor module is mapped to a specific schema within the Supabase database. This schema defines the structure and organization of the data, ensuring efficient storage and retrieval. Key elements of the schema include:

- **Room ID:** Identifies the specific room within Bliss Hall where the sensor module is installed.
- **Timestamp:** Records the exact time when the sensor data was collected.
- **LPG (PPM):** Stores the measured LPG concentration in ppm.
- **CH4 (PPM):** Stores the measured methane concentration in ppm.
- **Ethanol (PPM):** Stores the measured ethanol concentration in ppm.
- **Temperature (Celsius):** Stores the measured ambient temperature in Celsius.
- **Humidity (%):** Stores the measured humidity percentage.
- **Motion Detected (Boolean):** Indicates whether motion was detected by the PIR sensors.
- **Occupancy:** Stores the calculated room occupancy count.
- **Light Intensity:** Stores the measured light intensity value.

6. Integration with Unity Environment

The sensor data stored in the Supabase database is accessed and visualized within the Unity environment. This involves:

- **Data Retrieval:** The Unity application retrieves the sensor data from the Supabase database using a JavaScript API.

- **Data Mapping:** The retrieved data is mapped to the corresponding rooms within the virtual Bliss Hall model, based on the Room ID associated with each sensor module.
- **Visualization:** The sensor data is visualized in real-time within the Unity environment, dynamically updating based on the latest data received from Supabase. This includes:
    - **Gas Concentration Visualizations:** Visual cues like color gradients, or animated particle effects, can be used to represent the measured gas concentrations in different rooms.
    - **Temperature and Humidity Display:** Real-time updates of temperature and humidity readings can be displayed in the corresponding rooms.
    - **Occupancy Indicators:** Visual indicators, such as animated figures or changing colors, can be used to represent the calculated room occupancy levels.

7. Challenges Faced and Solutions

During the development and implementation of the sensor module, various challenges were encountered and addressed:

- **Wi-Fi Connectivity Issues:**
    - **Initial Approach:** The team initially attempted to use an ESP8266 chip connected to an Arduino Mega board. However, the ESP8266 lacked support for enterprise WPA2 user/pass login, hindering connectivity to AUB's Wi-Fi network.
    - **Alternative Solution:** A "Plan B" was devised to set up a private Wi-Fi network using a Raspberry Pi as a server and connecting the sensor modules to this network.
    - **Final Solution:** After further research, an ESP32 chip was acquired, and a third-party library was found on Github (https://github.com/martinius96/ESP32-eduroam) which enabled secure connection to AUB's Wi-Fi network using student credentials.
- **Supabase Library Availability:**
    - **Issue:** The library used for connecting to Supabase was locally installed but no longer available on Github.
    - **Solution:** The library was successfully restored, however no longer publicly available on github, overcoming this obstacle.
- **Gas Sensor PPM Calculation:**
    - **Issue:** Extrapolating ppm values from raw sensor data proved to be a complex task, requiring careful understanding of calibration techniques.
    - **Solution:** Through research and experimentation, the team successfully implemented the calibration process, accurately converting raw data to meaningful ppm values.

- **ESP32 Wifi Short Circuit:**
    - **Issue:** A misplaced jumper wire resulted in a short circuit, damaging the ESP32's Wi-Fi functionality.
    - **Solution:** A new ESP32 chip with built-in battery support was acquired to replace the damaged one, mitigating future risks.
- **Camera Integration Challenges:**
    - **Issue:** Attempting to integrate an OV7670 camera proved to be overly complex, requiring extensive wiring and driver development.
    - **Solution:** Due to time constraints and the technical complexity of the camera integration, this feature was temporarily shelved.

8. Future Enhancements

The sensor module can be further enhanced to provide even more comprehensive and insightful data for the AUB Digital Twin:

- **Camera Integration:** Revisit camera integration with a more efficient approach, perhaps utilizing a smaller camera module or exploring alternative vision-based solutions.
- **Additional Sensors:** Consider adding additional sensors, such as air pressure sensors or CO2 sensors, to expand the data collection capabilities and offer a more complete environmental picture.
- **Cloud-Based Data Processing:** Explore cloud-based data processing and analysis techniques, leveraging services like Google Cloud Platform or AWS, to gain deeper insights from the collected sensor data.
- **Real-Time Data Visualizations:** Develop more interactive and dynamic visualizations within the Unity environment, allowing for real-time monitoring and analysis of sensor data.
- **Integration with AUB Systems:** Investigate possibilities to integrate the sensor data with existing AUB systems, such as HVAC control systems, to enable more proactive and data-driven campus management.

Virtual Reality Documentation

What is VR in Unity and why a simulator?

Virtual Reality (VR) is a computer-generated simulation of an environment that allows users to interact with it in a seemingly real or physical way through the use of special electronic equipment, such as a headset with a screen or gloves fitted with sensors. VR creates a simulated world, offering immersive sensory experiences that can include sight, touch, and sometimes sound and smell.

In the context of Unity, a popular game development engine, VR is used to develop simulations and interactive experiences because of several reasons:

1. Immersive Experiences: Unity's powerful rendering engine, along with VR technology, can create highly immersive and interactive environments. This is particularly useful in simulations where the sense of presence within a virtual space enhances the experience, such as in training simulations, educational content, or entertainment.

2. Cross-Platform Development: Unity supports a wide range of VR platforms, including Oculus Rift, HTC Vive, PlayStation VR, and others. This makes it a versatile tool for developing VR applications that can reach a broad audience across different devices.

3. Interactive Training and Education: VR simulations in Unity are widely used for training and educational purposes across various fields such as medicine, aviation, military, and automotive industries. VR allows for safe, controlled, and repeatable scenarios, ideal for learning and practicing skills.

4. Architectural Visualization and Design: VR in Unity is also used for architectural visualization, allowing architects and clients to virtually walk through and interact with 3D models of buildings before they are built.

5. Accessibility: With Unity's user-friendly interface and extensive documentation, it is accessible to both beginners and professionals in the field of VR development. This has contributed to the widespread use of Unity for VR content creation.

How can VR help in the purpose of this project?

By integrating VR into the Unity simulator for Bliss Hall at the American University of Beirut (AUB), users can experience the campus in a more interactive and engaging way. This approach can significantly enhance the sense of presence, making users feel as if they are actually walking through Bliss Hall. The benefits of using VR in this context include:

- Enhanced Engagement: VR's immersive nature captivates users' attention more effectively than flat, 2D experiences.
- Educational Opportunities: VR can be used for virtual tours, historical recreations, and educational programs, offering students and visitors new ways to learn about Bliss Hall and its significance to AUB.
- Accessibility: Remote users can explore Bliss Hall as if they were there, making the campus more accessible to prospective students, alumni, and global researchers.

- Design and Planning: VR can serve as a powerful tool for facility management, allowing staff to visualize changes, maintenance needs, or upgrades in a fully immersive environment.

Combining the detailed visualization capabilities of Unity with the immersive potential of VR, and further enriched by a multiplayer component, this project can serve numerous purposes:

1. **Realistic Campus Visualization:** Unity's powerful 3D modeling tools enable the creation of a detailed and realistic digital twin of the campus. Integrating VR allows users to explore this environment as if they were physically present, which is invaluable for new or international students, visitors, or anyone looking to familiarize themselves with the campus layout and facilities.
2. **Interactive Learning Environments:**The digital twin can be augmented with interactive modules tailored to specific academic departments or labs. These modules can simulate practical experiments or provide virtual equipment interaction, offering a complimentary hands-on learning experience that is especially beneficial in fields requiring strong spatial understanding or practical skills. For example, students can register into the system and be able to watch lectures in real-time via the digital twin rather than onsite.
3. **Virtual Events and Tours:** The digital twin can host virtual events, orientations, and tours, broadening campus accessibility. This feature is particularly beneficial during events that limit physical campus access, such as public health crises or for students engaged in remote learning.
4. **Social Interaction and Community Engagement:** The multiplayer VR component fosters social interaction, enabling students to meet in virtual spaces, participate in events, or explore the campus together. This aspect helps in maintaining the vibrancy of the university community, facilitating virtual events, meetings, and group projects within a realistic campus setting.
5. **Gamification and Enhanced Engagement:** Incorporating game-like elements can make exploring the digital twin more engaging, encouraging users to interact with the campus in innovative ways. This could include orientation games, scavenger hunts, or educational challenges designed to help users, especially new students, learn about campus facilities and services in an interactive manner.

How can we use VR in this project?

To incorporate VR into the Bliss Hall digital twin, the following steps outline the process:

1. 3D Modeling with Blender: Creating detailed 3D models of Bliss Hall using Blender. This includes the building's exterior and interior, along with any significant features or furnishings. This task has already been implemented.

2. Importing into Unity: Import the 3D models into Unity. Unity's robust platform supports VR development, allowing for the integration of the models into a VR environment.
3. VR Integration:
    - Unity XR Plugin Framework: Utilize Unity's XR Plugin Framework to manage VR hardware compatibility. This framework simplifies the process of making the project accessible across various VR platforms.
    - Interactivity: Implement interactive elements using Unity's scripting capabilities. This can include opening doors, interactive learning modules, and more.
    - Navigation: Design a user-friendly navigation system within the VR environment, allowing users to move around Bliss Hall intuitively.
4. Testing and Optimization: Rigorous testing on different VR devices ensures compatibility and optimizes performance. Pay attention to frame rates, loading times, and overall user experience.

Tools to be Used

- Unity 3D: For the development of the VR environment, including programming interactivity and importing 3D models.
- Blender: For creating detailed 3D models of Bliss Hall and any associated assets.
- Unity XR Plugin Framework: To support a wide range of VR hardware and manage device-specific features.
- SteamVR Plugin for Unity (if targeting HTC Vive, Valve Index, etc.): To integrate SteamVR's tracking and input systems.
- Oculus Integration for Unity (if targeting Oculus Rift, Quest, etc.): For Oculus-specific features and optimizations.

What are the Steps to be followed in Unity to add this VR component

To add a VR component to your Unity project for the digital twin of Bliss Hall, you can follow these general steps. Please note that the specific details might vary depending on the version of Unity and the VR hardware you're targeting, but these steps provide a foundational guide. The YouTube video you mentioned could offer more detailed instructions or tips, so it's recommended to watch it alongside following these steps:

Setup Unity Environment:

- Install the latest version of Unity Hub and create a new project with the 3D template.
- Ensure your Unity version supports VR development with the XR Plugin Management system.

Import Assets:

- Import the 3D models of Bliss Hall that you created with Blender into your Unity project.
- Organize your assets within the Unity Editor for easier management.

Install XR Plugin Management:

- Go to "Edit" > "Project Settings" > "XR Plugin Management" and install the XR Plugin Management package.
- Enable the XR Plugin for your target VR platform (e.g., Oculus, HTC Vive).

Configure VR Settings:

- Within the XR Plugin Management settings, select your target platform and configure it according to your hardware's requirements.
- Ensure that the "Initialize XR on Startup" and "Stereo Rendering Mode" are correctly set.

Set Up the VR Camera:

- Replace the default Main Camera in your scene with a VR-compatible camera rig. This can be done by importing a VR package like SteamVR or Oculus Integration, which provides pre-configured camera rigs.
- Adjust the camera rig's position to match the desired starting point within Bliss Hall.

Implement VR Controls:

- If using a package like SteamVR or Oculus Integration, use the provided prefabs or scripts to implement VR controllers.
- Set up the input actions for movement, interaction, and other VR-specific functionalities like grabbing or teleportation.

Add Interactivity:

- Script interactive elements within Bliss Hall, such as doors that open or educational tools that users can engage with.
- Use Unity's physics system to add realism to object interactions.

Optimization for VR:

- Optimize your scene for VR to ensure smooth performance. This includes reducing polygon counts, optimizing textures, and ensuring efficient lighting.
- Use Unity's Profiler to monitor performance and make necessary adjustments.

Testing:

- Test your VR experience frequently on the actual VR hardware to ensure everything works as expected.
- Pay attention to user comfort, avoiding issues like motion sickness by ensuring stable frame rates and intuitive controls.

Build and Deploy:

- Once satisfied with the VR experience, go to "File" > "Build Settings" to configure your build settings for the target platform.
- Build your project and deploy it to your VR hardware for final testing and use.

The YouTube tutorial provides additional insights and visual guidance on implementing these steps in Unity: https://www.youtube.com/watch?v=TJ8yk5vVwl0

Future Vision for the VR Component

The VR component for Bliss Hall has the potential to evolve significantly in the future:

- Educational Modules: Develop interactive learning experiences, such as historical tours, engineering projects, or art exhibitions within Bliss Hall.
- Remote Collaboration: Enable students and faculty to collaborate on projects or studies within the virtual space, regardless of their physical location.
- Event Simulation: Host virtual events, conferences, or ceremonies within Bliss Hall, opening up new possibilities for community engagement.
- Expansion: Extend the VR experience to cover more areas of the AUB campus, creating a comprehensive virtual campus tour.
- Integration with Real-Time Data: Implement IoT (Internet of Things) data visualization within the VR environment, such as energy usage, room occupancy, and more, for educational and facility management purposes.