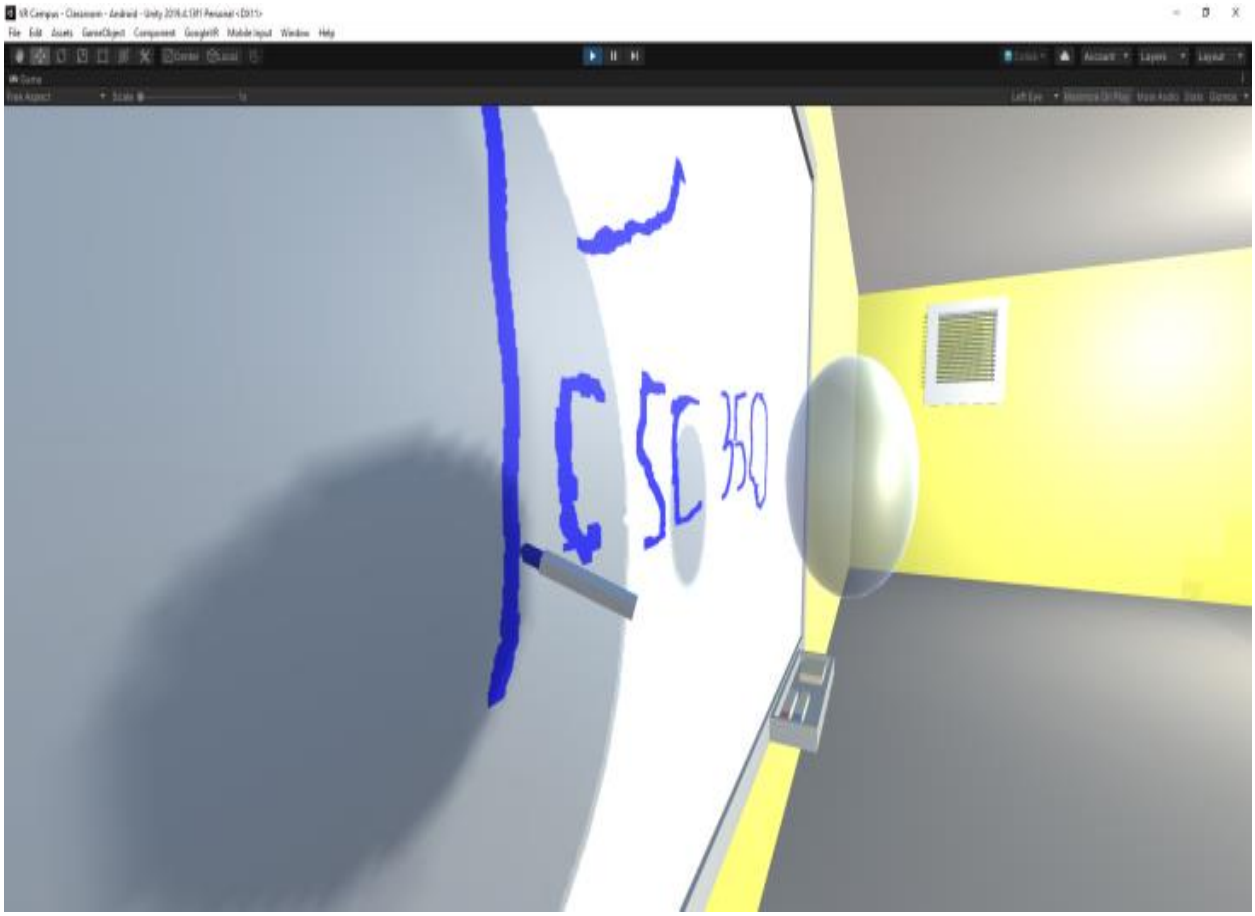
VR Campus: Team 7

Final Report



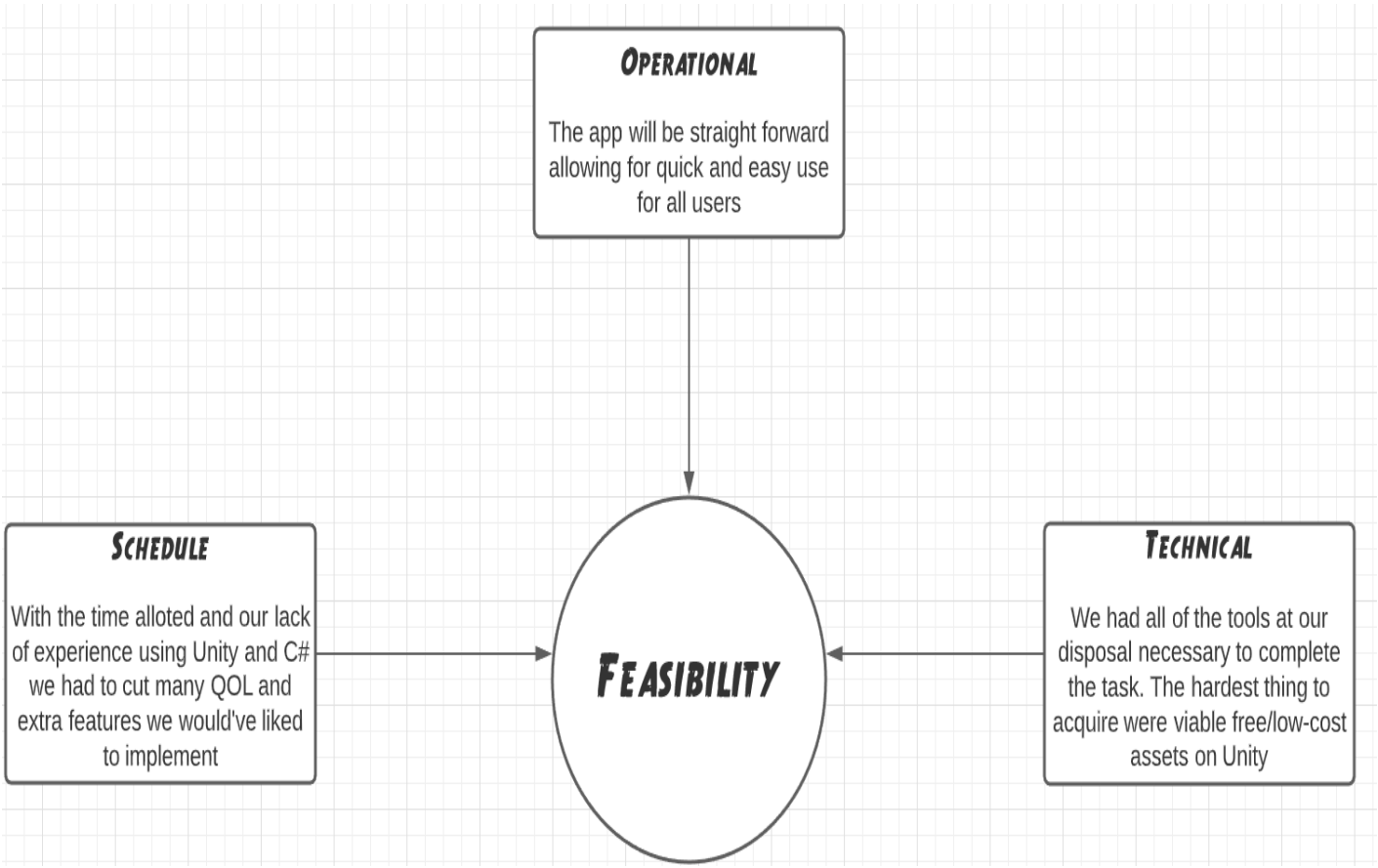
By: Jonathan Rosario, Jose Zarzuela, Yongfeng Liang, Sergey Hovhannisyan

Table of Context:

GENERAL OVERVIEW

The purpose of team 7's project was to create a virtual environment where students and professors can interact in a more personal matter while having to do schoolwork at home. We wanted to have users of our app feel like they are in an actual class or study environment without distractions of being at home. In times like these we know it's sometimes a hard task to stay focused in an online school environment with so many things going on in the background, so we wanted to tackle that issue with our project.

We wanted to include two different full-fledged environments for users of our app, a study room and a classroom. In it we were to add a fully interactable and customizable environment that included music, chat options, working whiteboard, and a web browser. With the time allotted we were not able to implement fully customizable rooms or a web browser. Our biggest hurdles came in the form of building a database and creating this whole world while learning it on the go. Trying to have meeting was also an issue at times because of how busy we all were, which cut into some of the implementation plans we had from individual tasks, all things considered we constantly kept in touch throughout the process either on discord messaging or through weekly meetings.



SYSTEM REQUIREMENTS

- Android (KitKat 4.0 or higher) / IOS (13.1 or higher) device
- Google Cardboard capable headset
- Broadband internet connection
- VR capable controller (optional)

Preliminary Design

- **Overview**
    - **The VR campus is an application designed to allow users, whether students, tutors, or professors, to work together and study/learn in a virtual environment free from real world distractions. Features allow for communication and meaningful interactions between users. Tools to aid in studying and disseminating information include a bulletin board, a whiteboard, and voice chat. The app also provides accessibility options for those with difficulty navigating the VR world. In addition to these features, we also have music and ambient sounds to help minimize real world distractions while in the virtual environment.**
- **Scenes and functions**
    - **One scene is a study room with the ability to choose music and ambience. It is more densely furnished to give the user a better experience when studying. In the study room there is a whiteboard with which the user can interact with using a virtual pen. There are also bulletin board notices where reminders can be posted. The maximum allowable capacity of users for this room is 4.**
    - **The second scene is a Classroom scene with a large capacity of 20(Max allowed by PUN 2's free plan) users. In this room the option to select music is not present as it is designed for a more instructional type of environment. There are seats in the classroom which students can sit in while listening and watching the instructor write on the whiteboard.**

- **User interface (tables and screenshots)**
    - **Main menu:**
        - **A virtual space that acts as a hub for creating rooms, accounts, and changing settings.**
    - **Radial menu to select actions and other user interactions.**
- **summary of the Implementation**
    - **Main Menu**
        - **3D Sphere Object encasing the world rendered main canvas.**
            - **Multiple Canvases nested within each other**
                - **Panels representing buttons within each canvas.**
                - **Clicking on certain buttons will activate/deactivate other canvases revealing the buttons for their respective canvas.**
                - **Buttons are animated through an event trigger system.**
    - **Whiteboard**
        - **3D Cube Object flattened out and stretched to look like a whiteboard**
            - **Texture is modifiable through the buttons located on whiteboard.**
                - **Each button is a prefab with a specific color tied to it.**
                - **Pressing button requires a virtual "hand" object to collide with the button's collider.**
            - **White board pen**

        - **Prefab created through two 3d cube objects stacked upon one another. One forms the tip, the other the handle.**
        - **Tip sends raycast to whiteboard collider, whiteboard senses the location and changes the color at that location to that of the selected tip color.**
- **Radio**
    - **Simple design created from multiple cubes and a cylinder**
        - **Smaller cube shaped buttons on the side determine which track will play.**
            - **Displays text information on users' camera to inform them of the track associated with that button.**
            - **User can use reticle to activate and change the track.**
        - **Larger cylindrical button on top for starting and stopping current track.**
            - **User can use reticle to activate the button and play currently loaded track.**
- **Network**
    - **Implemented using PUN 2 for multiplayer and voice chat**
        - **Voice is a global audio source shared by all users.**
            - **Emits sound from users currently speaking.**

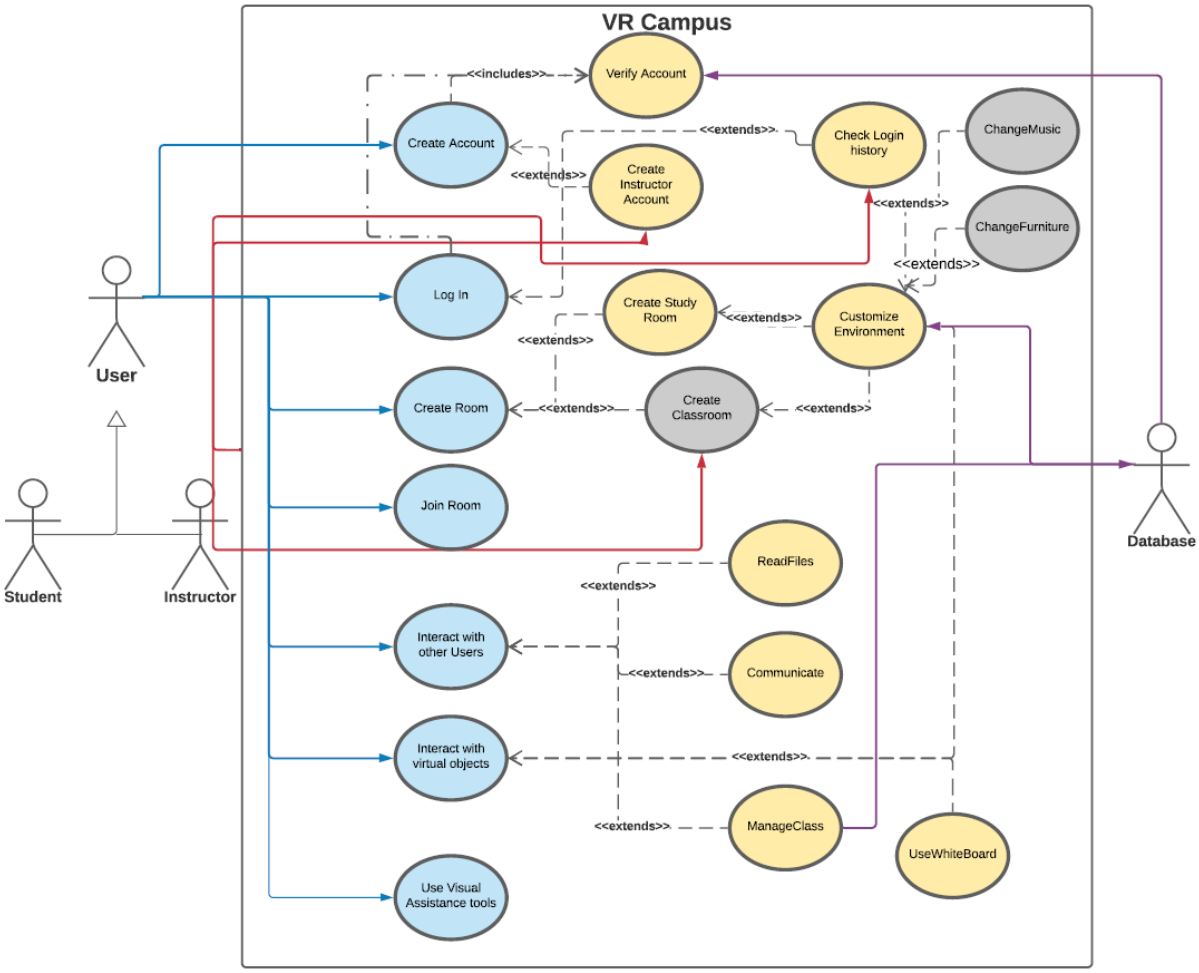
- **Room creation done through PUN 2 network. Currently hard coded to create a room named “Room 1”.**
    - **Server is hosted locally on users’ device**
    - **Other users connect to the master client**
    - **Each avatar is synchronized over the network and controlled locally**

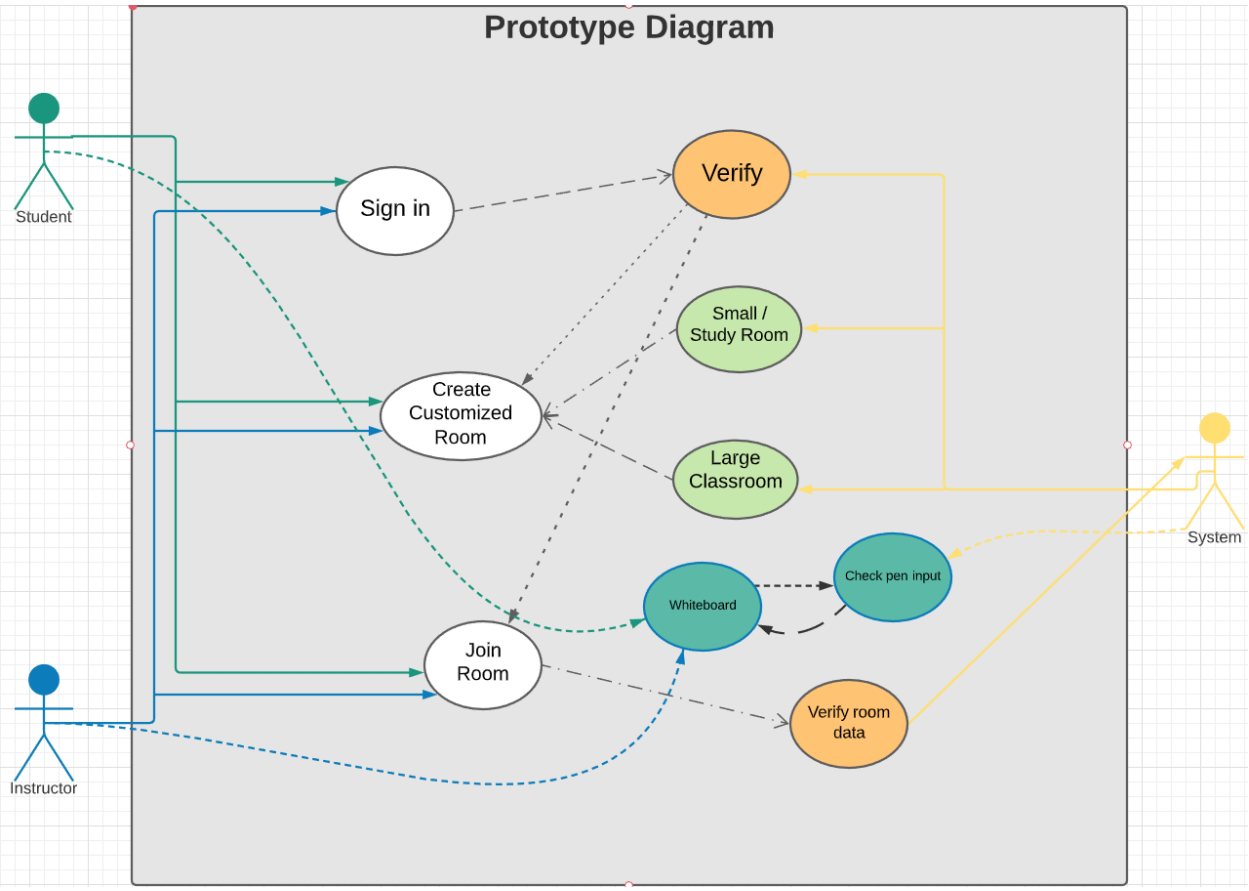
- **Manomotion**
    - **Using the prefabs provided by the developer of the SDK**
        - **Only the pinch gesture is used**
            - **Switches between virtual hand and whiteboard pen when pinch is activated.**
        - **Location of hand in space is used. Tracking information passed from the Manomotion manager which access users’ camera to locate and track users’ hand.**
            - **Camera render is disabled as to save resources on users’ device.**

- **user interactions (how users interact with VR objects)**
    - **Whiteboard:**
        - **Displays drawings user creates using their virtual pen.**
    - **Whiteboard Pen:**
        - **Object user needs to move in order to write on whiteboard. Only works on whiteboard surface. Using Manomotion to track users' hand in real life and translating that information into the virtual world space.**
    - **Whiteboard Buttons:**

        - **Used to fill the entire white board with a color. Also acts as a way to erase what is on the board. Available in four colors at present. Animated and reacts to the physics of the virtual hand “pressing” the button.**
    - **Radio:**
        - **Uses reticle to display information of each button user looks at. User can press GVR button on their device to activate the button.**
    - **Doors and Chairs:**
        - **User can use reticle and GVR button to teleport between doors and chairs. A way to navigate and “sit” in the virtual space.**
    - **Sky buttons:**
        - **Used for various accessibility features.**
- **VR navigation**
    - **Movement**
        - **Movement is accomplished through the use of teleporting to certain objects in the virtual space.**
        - **Users can also “walk” by adjusting their view towards the ground meeting a certain angle threshold causing their avatar to move forward in the direction they are facing.**
        - **Third party Bluetooth controller support also an option for movement. Allows user to use joystick to move in a direction relative to the users forward facing view.**

USE CASE DIAGRAM:




Prototype Diagram
Student
Sign in
Verify
Small / Study Room
Create Customized Room
Large Classroom
System
Check pen input
Whiteboard
Join Room
Verify room data
Instructor

USE CASE TABLES FROM EARLIER:

Number	PW	Requirements
REQ 1	1	User account management enabling users to log in and preferences to be stored.
REQ 2	2	User chooses room preferences to create a room for themselves or multiple users.
REQ 3	3	Ability to incorporate music through UI and interactable object
REQ 4	2	The program will allow users to interact with virtual items.
REQ 5	1	Multiplayer system allowing users to interact with one another through a network: handle things, share files.
REQ 6	4	The system will allow users to navigate local phone files and share them.
REQ 7	5	User will be able to navigate UI through hand gestures and buttons.
REQ 8	1	Voice chatting system for users to communicate with one another.
REQ 9	6	Whiteboard using gestures with hand tracking feature every room.
REQ 10	1	Users with visually impaired can use a set of tools to make them more Accessible to use the application.

Use Case	Use Case Name	Description	Requirements
Column1	Column2	Column3	Column4
UC-1	CreateAccount	Allows user to create a general account for logging on to the system	REQ 1
UC-2	CreateInstructorAccount	Allows user to create an instructor type account with additional privilages	REQ 1
UC-3	AddUser	Allows for adding of user after verification	REQ 1
UC-4	RemoveUser	Allows for removak of user at instructors request	REQ 1
UC-5	VerifyAccount	System checks for duplicate names/emails or invalid password before account creation finishes	REQ 1
UC-6	CheckLoginHistory	Allows user to send a signal to the system to display login history	REQ 1
UC-7	Log-in	Allows user to input account details and login to system	REQ 1
UC-8	Logout	Logs the user out of the system	REQ 1
UC-9	CreateStudyroom	Creates a study room, useable by instructors and students	REQ 1, REQ 2
UC-10	CreateClassroom	Creates a class room, useable only by instructor	REQ 1, REQ 2
UC-11	RoomTemplate	Ability to save a created room template, and delete a previous one	REQ 1, REQ 4, REQ 5
UC-12	JoinRoom	Allows user to join a room	REQ 1, REQ 2
UC-13	InteractWithOthers	Users can communicate with one another in the virtual world	REQ 2, REQ 5
UC-14	Communicate	Allows the use of voice/text chat to speak to another user	REQ 1, REQ 5
UC-15	ReadFiles	Allows to read certain file types in virtual world	REQ 1, REQ 4, REQ 5
UC-16	ShareFiles	Allows the ability to share files	REQ 7
UC-17	MenuDisplay	"Pause" menu options throught button press	REQ 1, REQ 7, REQ 10
UC-18	ManageClass	Gives instructor options to manage class such as muting or allowing/disallowing use of other features	REQ 1, REQ 2, REQ 4, REQ 5, REQ 8
UC-19	PingStudents	Allows instructor to send visual/audioo ping to students	REQ 1, REQ 7, REQ 10
UC-20	InteractWithObjects	Users can interact with virtual objects based on the object type	REQ 1, REQ 2, REQ 4
UC-21	UseWhiteBoard	Allows user to write or erase items on the whiteboard	REQ 4, REQ 9
UC-22	VisualAssistance	Gives user tools to assist with navigation of the virtual world	REQ 10
UC-23	AddItem	Ability to add an item in real time or when creating room	REQ 1,REQ 4, REQ 2
UC-24	DeleteItem	Ability to delete item in real time or when creating room	REQ 1,REQ 4, REQ 2
UC-25	ChangeMusic	User can change music/ambience in the room. Instructor is allowed to change classroom music	REQ 1, REQ 2, REQ 4, REQ 7
UC-26	ChangeFurniture	Allows user to add or remove furniture/objects to interact with in the room	REQ 1, REQ 2, REQ 4, REQ 7

Use Case 1: Create/Join Room

Use Case UC-# 1:	**Name/Identifier**
Related Requirements:	REQ 5, REQ 7 as stated in System Requirements
Initiating Actor:	Any of: Student, Professor
Actor's Goal	To create and join a virtual room
Participating Actors:	Student/Professor, System
Preconditions:	• User must be signed in • Must not currently be in a room
Postconditions:	• The instance is logged into system
Flow of Events for Main Success Scenario:	→ 1. **User** logs into app ← 2. **System** verifies user and placed in main menu → 3. **User** selects creates room ← 4. **User** customizes or chooses preset room ← 5. **System** creates and places user in room
Flow of Events for Extensions (Alternate Scenarios):	→ 1. **User** logs into app ← 2. **System** verifies user and placed in main menu → 3. **User** selects join room ← 4. **User** enters password of selected room ← 5. **System** places user in room

Number	PW	Requirements
REQ 1	1	User account management enabling users to log in and preferences to be stored.
REQ 2	2	User chooses room preferences to create a room for themselves or multiple users.
REQ 3	3	Ability to incorporate music through UI and interactable object
REQ 4	2	The program will allow users to interact with virtual items.
REQ 5	1	Multiplayer system allowing users to interact with one another through a network: handle things, share files.
REQ 6	4	The system will allow users to navigate local phone files and share them.
REQ 7	5	User will be able to navigate UI through hand gestures and buttons.
REQ 8	1	Voice chatting system for users to communicate with one another.
REQ 9	6	Whiteboard using gestures with hand tracking feature every room.
REQ 10	1	Users with visually impaired can use a set of tools to make them more Accessible to use the application.

Use Case	Use Case Name	Description	Requirements
UC-1	CreateAccount	Allows user to create a general account for logging on to the system	REQ 1
UC-2	CreateInstructorAccount	Allows user to create an instructor type account with additional privlages	REQ 1
UC-3	AddUser	Allows for adding of user after verification	REQ 1
UC-4	RemoveUser	Allows for removak of user at instructors request	REQ 1
UC-5	VerifyAccount	System checks for duplicate names/emails or invalid password before account creation finishes	REQ 1
UC-6	CheckLoginHistory	Allows user to send a signal to the system to display login history	REQ 1
UC-7	Log-in	Allows user to input account details and login to system	REQ 1
UC-8	Logout	Logs the user out of the system	REQ 1
UC-9	CreateStudyroom	Creates a study room, useable by instructors and students	REQ 1, REQ 2
UC-10	CreateClassroom	Creates a class room, useable only by instructor	REQ 1, REQ 2
UC-11	RoomTemplate	Ability to save a created room template, and delete a previous one	REQ 1, REQ 4, REQ 5
UC-12	JoinRoom	Allows user to join a room	REQ 1, REQ 2
UC-13	InteractWithOthers	Users can communicate with one another in the virtual world	REQ 2, REQ 5

UC-14	Communicate	Allows the use of voice/text chat to speak to another user	REQ 1, REQ 5
UC-15	ReadFiles	Allows to read certain file types in virtual world	REQ 1, REQ 4, REQ 5
UC-16	ShareFiles	Allows the ability to share files	REQ 7
UC-17	MenuDisplay	"Pause" menu options throught button press	REQ 1, REQ 7, REQ 10
UC-18	ManageClass	Gives instructor options to manage class such as muting or allowing/disallowing use of other features	REQ 1, REQ 2, REQ 4, REQ 5, REQ 8
UC-19	PingStudents	Allows instructor to send visual/audioo ping to students	REQ 1, REQ 7, REQ 10
UC-20	InteractWithObjects	Users can interact with virtual objects based on the object type	REQ 1, REQ 2, REQ 4
UC-21	UseWhiteBoard	Allows user to write or erase items on the whiteboard	REQ 4, REQ 9
UC-22	VisualAssistance	Gives user tools to assist with navigation of the virtual world	REQ 10
UC-23	AddItem	Ability to add an item in real time or when creating room	REQ 1,REQ 4, REQ 2
UC-24	DeleteItem	Ability to delete item in real time or when creating room	REQ 1,REQ 4, REQ 2
UC-25	ChangeMusic	User can change music/ambience in the room. Instructor is allowed to change classroom music	REQ 1, REQ 2, REQ 4, REQ 7
UC-26	ChangeFurniture	Allows user to add or remove furniture/objects to interact with in the room	REQ 1, REQ 2, REQ 4, REQ 7

Appendix: Jonathan Rosario

I. Summary of Tasks

(1) Network Implementation

a. Voice Chat

b. Small and Large Rooms Creation

c. Player Prefab

d. Networked Teleportation

(2) Manomotion Integration

a. Hand Tracking

b. Gesture Recognition

(3) User Interface (Main Menu)

a. Panels

b. Animations

c. Room Creation

(4) Whiteboard

a. Pen

b. Whiteboard Surface

c. Buttons

(5) Music

a. Track selection

b. Start/Stop

II. Challenges

1) Network Implementation

a. Learning how to implement PUN took a lot of time.

To implement it I watched a few tutorials on YouTube. InfoGamer (https://youtu.be/phDySdEKXcw), had an extremely useful playlist that assisted me in this. In addition to this, I spent time and went through the entirety of the PUN official tutorial here: https://doc.photonengine.com/en-us/pun/v2/demos-and-tutorials/pun-basics-tutorial/intro

This tutorial has since been updated and is now slightly different from when I went through it. The tutorial and in-depth documentation that PUN provided was extremely useful in getting the multiplayer to work.

b. Many moving parts to keep track of.

Once I began adding other moving parts to the multiplayer I ran into more issues. Keeping track of two "Players" which were simply spheres with scripts for movement took some work. Once I integrated Manomotion, it became much more difficult. This will require more research to solve for. Also implementing a single whiteboard pen to be synchronized across multiple clients required even more tracked data.

c. More than one player caused some erratic and unpredictable behavior.

Initially I ran into the problem of players joining in after the first player losing control of their avatar or camera. There was also a phantom third player. This issue no matter how hard I tried persisted despite following numerous tutorials. What finally resolved this was removing some duplicate code. Due to a lack of good code organization in the project and adding new features into the project without having first tried it on a separate project, there were lots of files and multiple functions attempting to do the same thing. After lots of examination, I noticed that I had forgotten the particularly important "this" keyword. My code had not checked to see whether the current avatar belonged to the local client. So, when a new player would start, the

“update()” function would run and cause the main client to control more than one camera and avatar. Additionally, by having the player fab instantiated in two separate locations, it caused a third phantom player to exist.

d. Voice worked but adding in more advanced features would have taken more time.

The voice was not too difficult to implement. However, I attempted to add a voice icon to appear over the head of an avatar when a certain threshold was met. This never worked properly, or ever. More research into PUN voice documentation is needed.

e. Player Prefabs

The simple player prefab with a photon view and movement script was created and worked. When I added in a “virtual hand” it ended up not working over the network and did not work properly even when testing a single player with that feature in combination with Manomotion.

f. Room creation

Creating rooms did not take as much time once I had gotten the PUN up and running. Through the main menu, by using a room controller, I created the option for a room to be made. At this time, the room is hardcoded to create a room named “Room 1”, however in the future I plan to create a more robust system where the user can type in a name for the room they are creating, and number of players allowed in. The reason for the lack of this feature is due to not having solved for player input. One thing I learned from watching another team, was the idea of starting off with a 2D scene, allowing for user keyboard access, and then transitioning into a 3D VR scene.

2) Manomotion Implementation

a. Implementing Manomotion required a lot of work.

Manomotion is an amazing sdk, however, it has sparse documentation as of this moment. To understand the sdk, I had to look through the code and seek help from the creator and community that worked with it. I joined their discord server where I would ask for help understanding how the sdk worked. One user had written a custom shader to allow for Manomotion to work with GVR. His code would create a shader which would be applied to a material and then applied to the user's camera to create the VR camera.

b. The documentation was lacking

At one point I had found an older tutorial for Manomotion on YouTube (https://youtu.be/NLCN1ondCxM). I attempted to follow along, but the function names and many of the features had already changed since that video. The official documentations: https://www.manomotion.com/documentation/

Is inaccessible unless you first create an account. Once you look over it you will notice that there is not a list of functions and explanation as to what each function does as of 12/15/2020. To overcome this challenge, I learned of an especially useful feature in Visual Studio. You can "click into" a function and go to the file containing that function. Through looking over the functions I was able to determine what functions would be necessary for the features I wanted to implement.

c. No native support for Google VR.

Manomotion has no native support for GVR, instead it geared towards AR and regular 3D games. Getting the demo scene working in VR required the use of a custom shader. All the issues Manomotion presented stem from a lack of documentation and tutorials. Despite the developer's willingness to help, his help was slow and geared toward more experienced developers.

d. Tracking more than one object difficult.

This became an issue due to my attempts at trying to create more than one object to follow the hand. The first issue I encountered was when tracking the hand, and then "disabling" the hand object to instead start tracking the "pen" object. This would work initially but would not allow me to reactivate the hand properly. The hand would "fly" out of the screen or never return.

e. Very time intensive due to the research involved

There were many snags in getting Manomotion running. With the lack of documentation, I attempted everything from learning how to access my android's camera through a tutorial, to trying to run both my camera and google VR at the same time.

Camera tutorial: https://youtu.be/c6NXkZWXHnc

Documentation: https://docs.unity3d.com/ScriptReference/WebCamTexture.html

Limitations in my skill, knowledge, and time made it difficult to "fully learn" features and get them working with one another.

3) User Interface

a. Google VR has limited options for input, menu selection needs to involve head movements if no other input available.

What I decided was to use head movement for selection of menu items. I created a big Sphere 3D object to encase the user and act as a sort of hub. In that sphere I created multiple canvases and had them rendered in the world space as opposed to overlaying it on the camera. On these canvases I created panels that would react to the graphics raycaster from the GVR reticle and activate when the GVR button was pressed. Linking these panel buttons to room creation was not difficult when PUN was up and running. Animating the panels took some effort, by using the Unity animate window, I created an animation that would activate and run once if it detected the graphics ray caster hovering over the panel.

4) Whiteboard

a. The board and pen

The implementation of the whiteboard required learning about how textures worked in Unity. Not only that, but it was also necessary to learn how to detect where the pen tip was currently raycasting to on the whiteboard object. To do this, I needed to detect the raycast "touch" and detect the local coordinates on the texture of where the raycast collided with the whiteboard's collider. I used the RaycastHit documentation https://docs.unity3d.com/ScriptReference/RaycastHit.html, the Physics.Raycast documentation https://docs.unity3d.com/ScriptReference/Physics.Raycast.html, and the Texture 2d documentation https://docs.unity3d.com/ScriptReference/Texture2D.html to solve for this. The RaycastHit in combination with the texture coord function allowed me to detect where precisely on the texture the ray cast had hit. By specifying the size of the raycast to be the size of the pen's tip I was able to create a change in the texture. I also made the lines smooth by using the lerp function which uses a math library to interpolate and guess where the next position is when the pen moves ensuring a smoother line.

b. Manomotion and whiteboard

Manomotion was the perfect solution for input. The main challenge was finding a way to keep the hand from shaking too much. Unfortunately, this is something I Have yet been able to solve for. I did create a function that would lock the rotation of the pen when it touched the board, but it had trouble functioning the way I wanted it to. More time and research are needed to enhance this feature.

5) Music

a. Starting and stopping music at a certain point.

The music feature was the last and final feature to be implemented before our time was up. It was surprisingly simple to get working. However, getting traditional music player features required more time and I did not have enough to fully research and learn how to implement it. Implemented by looking through official unity documentation https://docs.unity3d.com/ScriptReference/AudioSource.html

III. Lessons and Experience

This entire project has taught me a great deal. When we first began, I had a million and one ideas. Often time a lot of my frustration came from the fact that I wanted to implement all these ideas and have a fully functional application by the end of the semester. In fact, I had thought that the goal was to have a real-world app ready to be delivered.

This was unrealistic. Given the timeframe, my knowledge, and skill level, it was completely out of the realm of possibility. This was the first lesson, and the most important one. Feasibility does not necessarily only mean something that can be done, but also means something that can be done within your resources. Your resources are limited, whether time, cost, labor, or skill. If it were possible to create a fully functional app with every idea, we would not be in school learning.

It is better to focus on a few features and fully implement those. Implementing as many features as I did, left them not as robust and fully fleshed out. Another negative effect is the fact that I was unable to learn all functions related to implementing those features.

Our team also learned how important communication is when collaborating. We learned to compromise and work with each other. Working from such a long distance with vastly different schedules meant our communication had to be detailed and timely. We had to divide and conquer the project.

We did run into some issues along the way. For example, only two of us were able to integrate our parts together. Our communication improved vastly toward the end, however the initial communications left some issues, such as not realizing some of our members were working on a different Unity version.

Finally, I would say just as equally important as being grounded, is to plan better. Planning can save time and keep you from straying too far from your goals. With good planning, you will cut costs, and meet deadlines. Under planning I would also include organization and documentation. Deadlines cause immense pressure and would often cause me to forget to document properly.

Proper code and file organization would have saved me much stress and time. Quite frequently, I caught myself playing "catch-up" with trying to remember what I had been attempting to accomplish with certain lines of codes. If I had properly

documented the issues I had been working on, I may have saved time and frustration or all together avoided some problems.

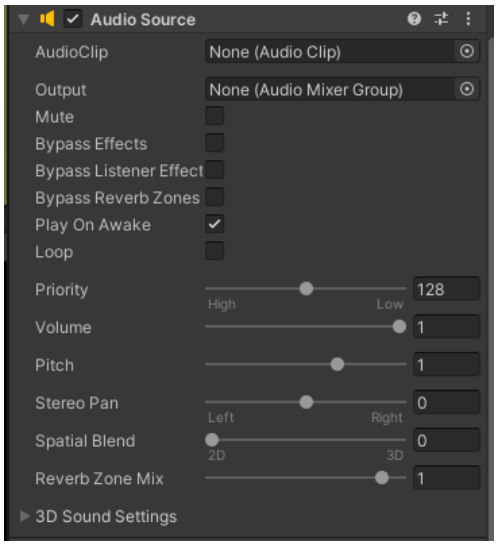
All these lessons and experiences will help me with my future development career. This project and has truly been the most valuable experience I have gained in my years at BMCC. Learning about the software development cycle and working in a group will be experiences that I will never forget.

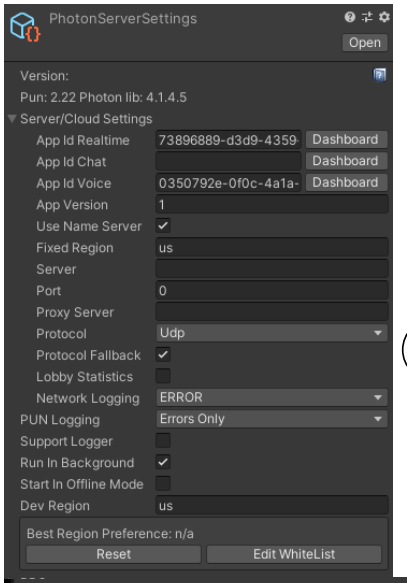
IV. Implementation Logs

Network Implementation

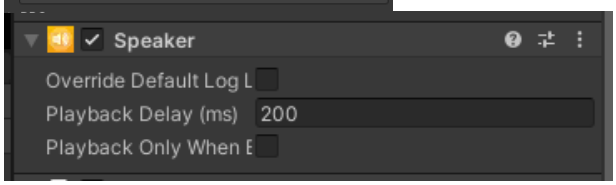
Voice Chat Settings: Implemented through inspector and PUN prefabs.
1,2,3 = player settings, 4,5 = Network Voice Manager

1

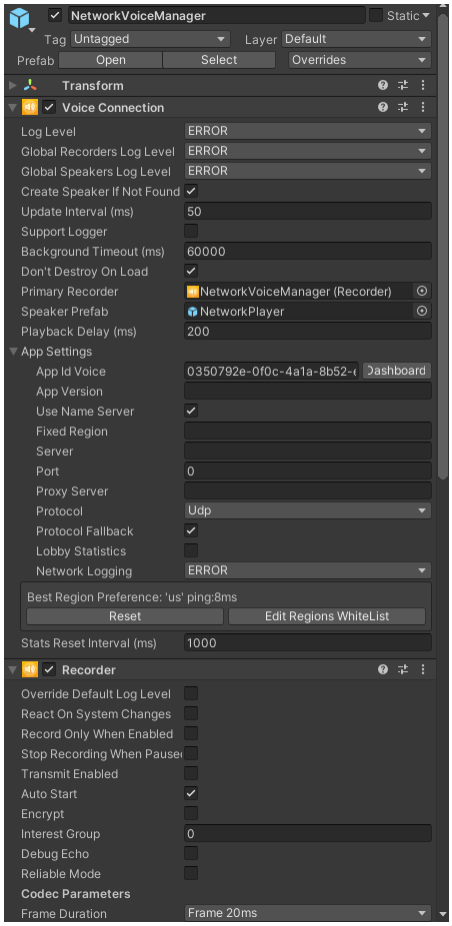




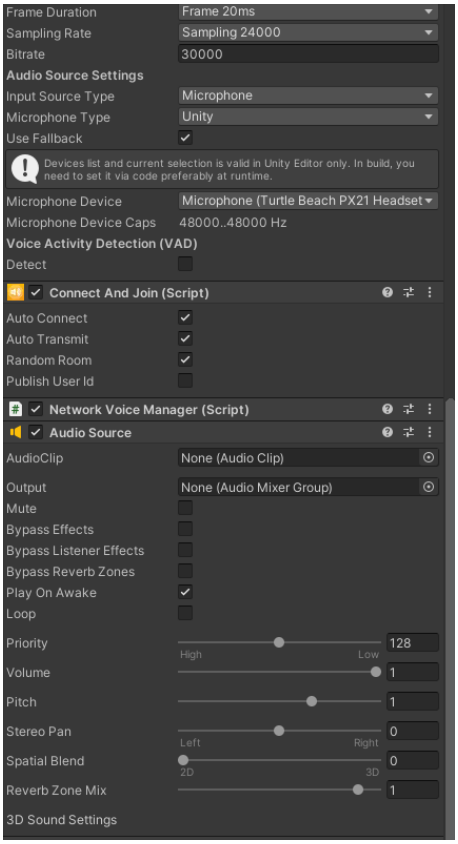
2



3



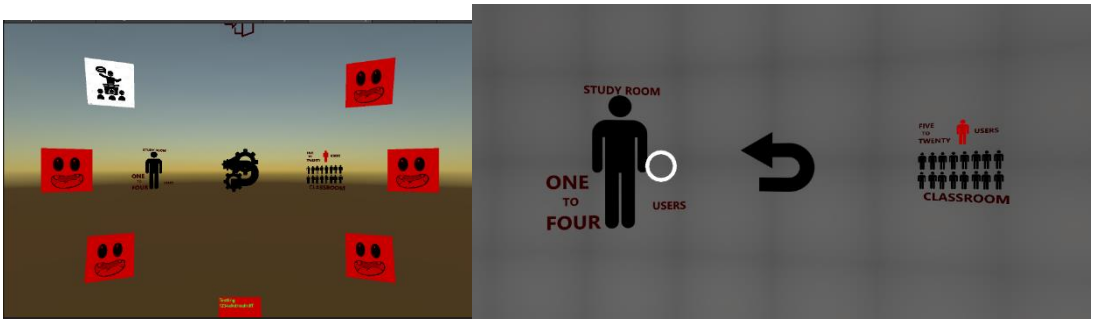
4



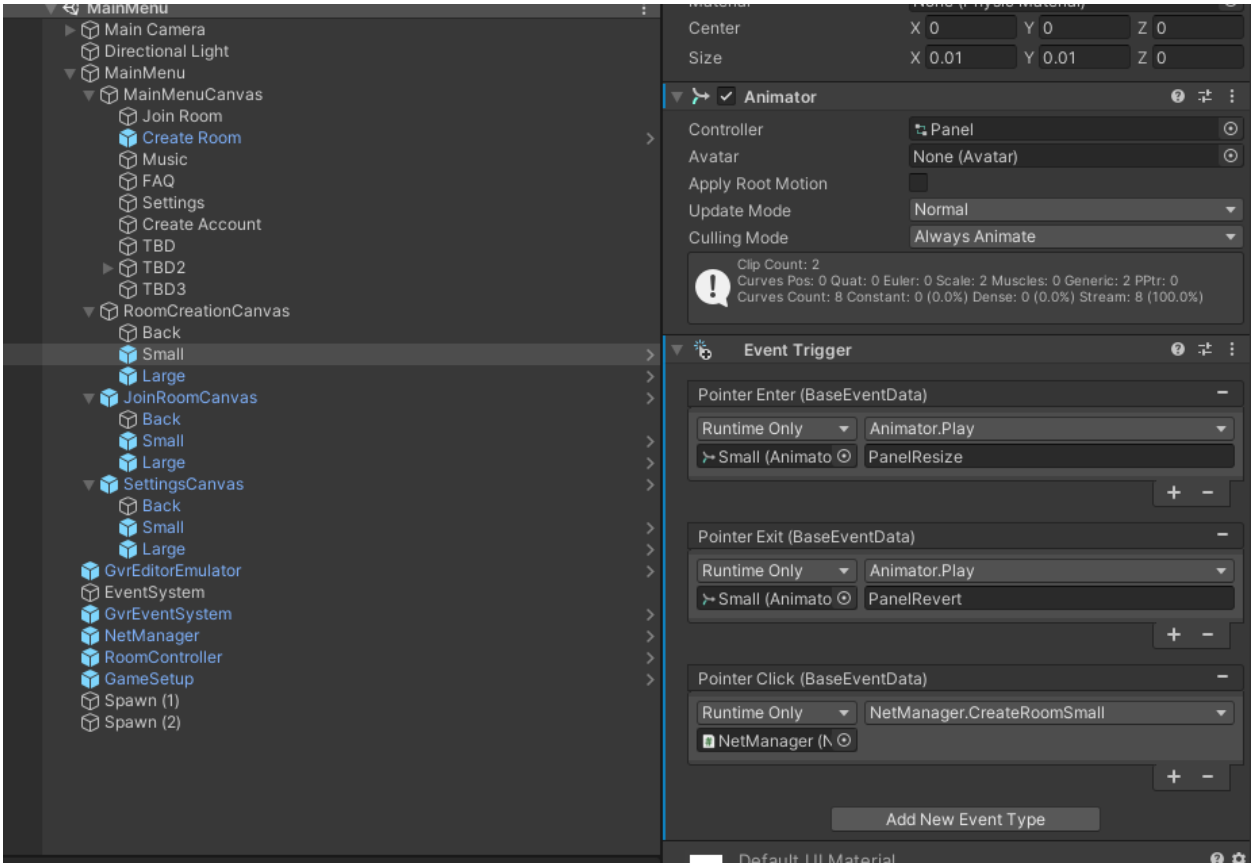
5

Small and Large Rooms Creation:

Combination of scripts and unity editor visual components for event trigger system.



Canvas Nesting and event trigger for main menu, the nesting allows for disabling and enabling different canvases in order to display one set of options at a time:



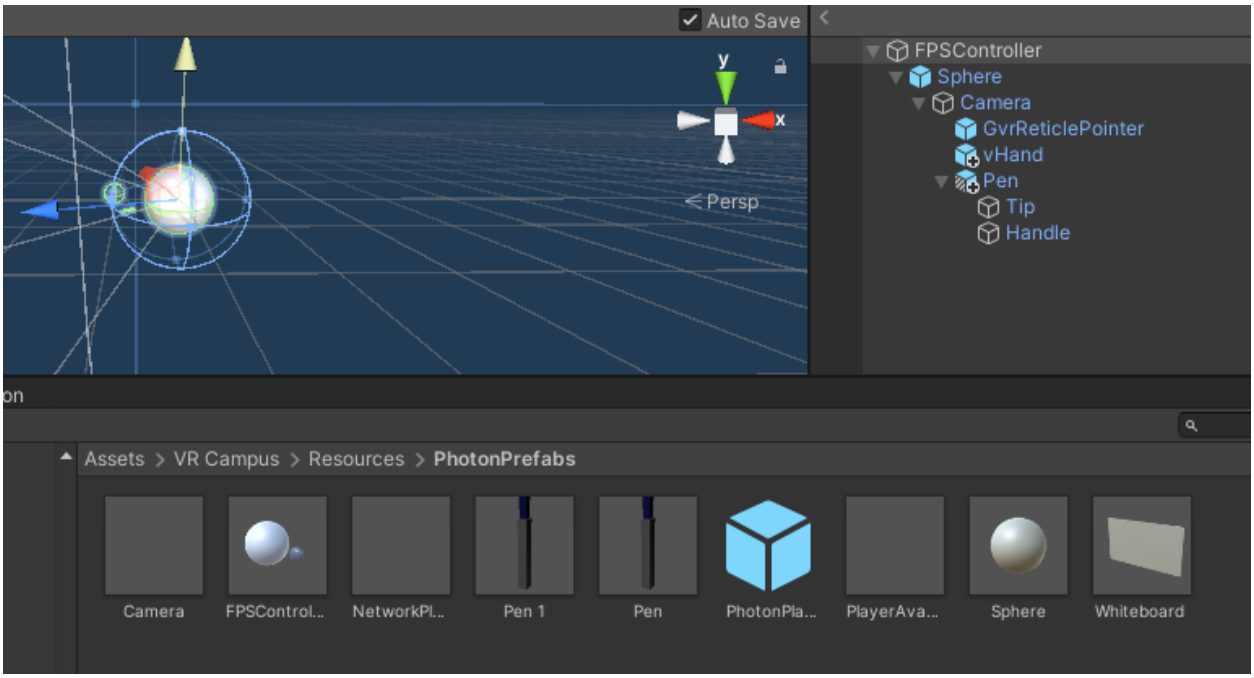
You will also notice the “Spawn” objects which are used for determining where the player prefabs are instantiated.

Functions called when using the on trigger event to create room:

```
16 references
public override void OnJoinedRoom()
{
    base.OnJoinedLobby();
    Debug.Log("We are now in a room");
    Debug.Log(PhotonNetwork.CurrentRoom.Name);
    if (!PhotonNetwork.IsMasterClient)
        return;
    StartGame();
}

1 reference
void StartGame()
{
    Debug.Log("Starting Game");
    if (PhotonNetwork.CurrentRoom.MaxPlayers == 4)
    {
        multiplayerScene = 1;
        PhotonNetwork.LoadLevel(multiplayerScene);
    }
    else
    {
        multiplayerScene = 2;
        PhotonNetwork.LoadLevel(multiplayerScene);
    }
}

2 references
void OnSceneFinishedLoading(Scene scene, LoadSceneMode mode)
{
    currentScene = scene.buildIndex;
    if(currentScene == multiplayerScene)
    {
        CreatePlayer();
    }
}

1 reference
private void CreatePlayer()
{
    GameObject myPlayer = PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs", "FPSController"), spawnPosition, Quaternion.identity, 0);
}
```

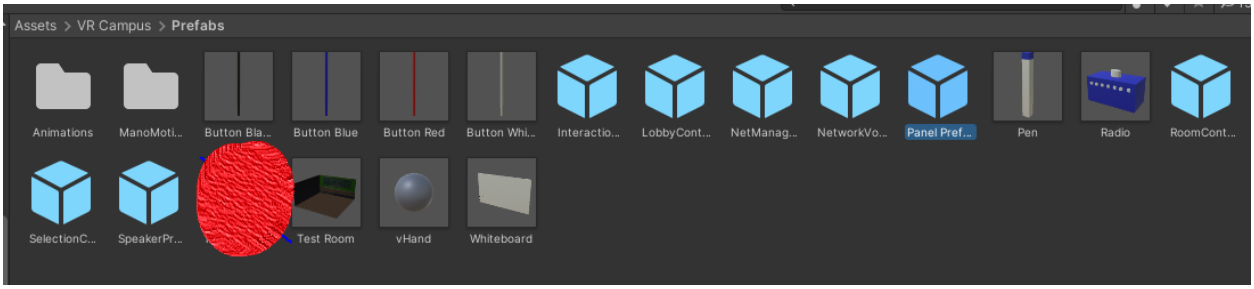
Player Prefabs:

Used for testing and spawning characters, and other prefabs.



Also included in the screenshot above are the prefab objects created for instantiation and for the whiteboard to easily share across different scenes.

Other prefabs, some are duplicates, red circle is a prefab I did not create:



Manomotion prefabs I used for integration:

Networked Teleportation:

Tested with only one player, however with some modifications in finding the local game object, I believe Jose's script can work on the network. He could not figure out how to pass the players information along when they were instantiated so I modified his code like so:

```
public GameObject player;
// Start is called before the first frame update
Unity Message | 0 references
void Start()
{
    player = GameObject.Find("FPSController(Clone)");
}

// Update is called once per frame
Unity Message | 0 references
void Update()
{

}

0 references
public void PointerClick()
{
    //move the player to the current position
    player.transform.position = new Vector3(transform.position.x + 2.0f, transform.position.y, transform.position.z);
}
```

Manomotion Integration

Hand Tracking:

Looking through the SDK I found the functions that provided me with the information I needed. I also found a function that would estimate the depth of your hand and use that to calculate a z position. The gesture information was used to switch between the “hand” and “pen”. There are also some lines that have been commented out that I had been experimenting on.

```
public class PositionAndOffset : MonoBehaviour
{
    public Camera cam;

    Vector3 offSet = new Vector3(2f,2f,2f);
    HandInfo detectedHand;
    public bool isGrabbed;
    Transform startPos;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        startPos = this.transform;
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        Grab();
        if (isGrabbed)
            currentHandPostion();


    }

    1 reference
    private void Grab()
    {
        detectedHand = ManomotionManager.Instance.Hand_infos[0].hand_info;
        ManoGestureTrigger currentDetectedGesture = detectedHand.gesture_info.mano_gesture_trigger;
        if(currentDetectedGesture == ManoGestureTrigger.CLICK)
        {
            isGrabbed = !isGrabbed;
        }
        if (!isGrabbed)
        {
            this.transform.position = cam.transform.position + (-offSet);
        }
    }
    1 reference
    private void currentHandPostion()
    {

        detectedHand = ManomotionManager.Instance.Hand_infos[0].hand_info;
       // ManoClass currentDetectedGesture = detectedHand.gesture_info.mano_class;

        Vector3 palmCenterPosition = detectedHand.tracking_info.palm_center;

        //if ( currentDetectedGesture == ManoClass.POINTER_GESTURE) {

        this.transform.position =  ManoUtils.Instance.CalculateWorldPosition(palmCenterPosition, detectedHand.tracking_info.depth_estimation);
        //this.transform.rotation = cam.transform.forward;
        //}
    }
```

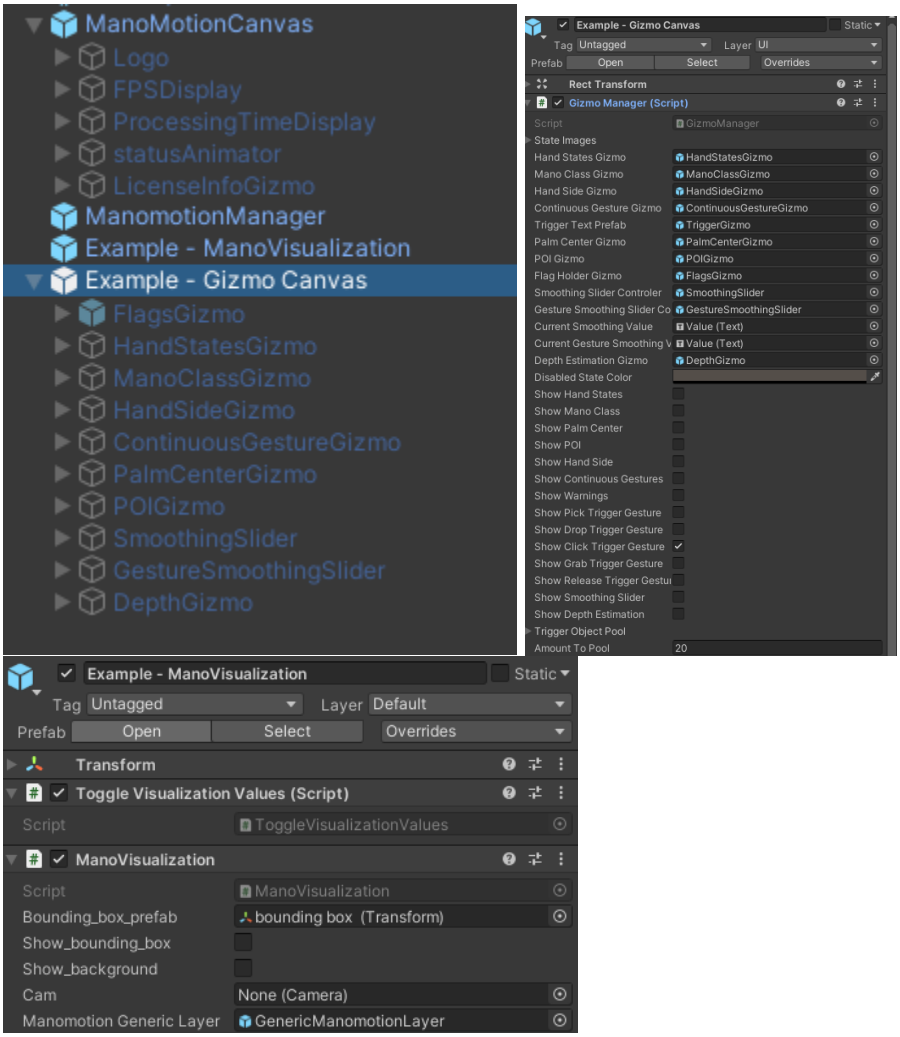
It seems like depth estimation is based on the fact that your hand gets “smaller” the further away from the camera it is causing the bounding box to shrink. Using that information Manomotion gives a rough estimate of the depth. Works better in a well lit room with a background that isn’t similar in color to your hand.

Gesture Recognition:

This was taken from the code above. Passing the information from Manomotion's manager which contains the current hand gesture information. Must be used in the Update() function to get the most up to date information.

```
detectedHand = ManomotionManager.Instance.Hand_infos[0].hand_info;
ManoGestureTrigger currentDetectedGesture = detectedHand.gesture_info.mano_gesture_trigger;
if(currentDetectedGesture == ManoGestureTrigger.CLICK)
{
    isGrabbed = !isGrabbed;
}
if (!isGrabbed)
{
    this.transform.position = cam.transform.position + (-offSet);
}
```

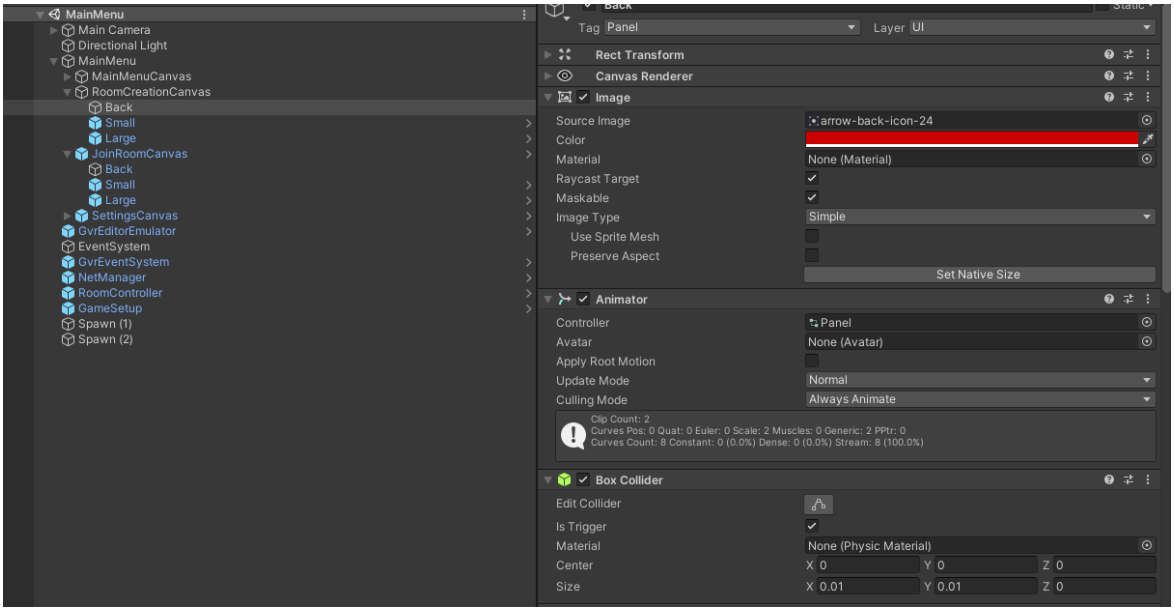
Manomotion prefabs setup:

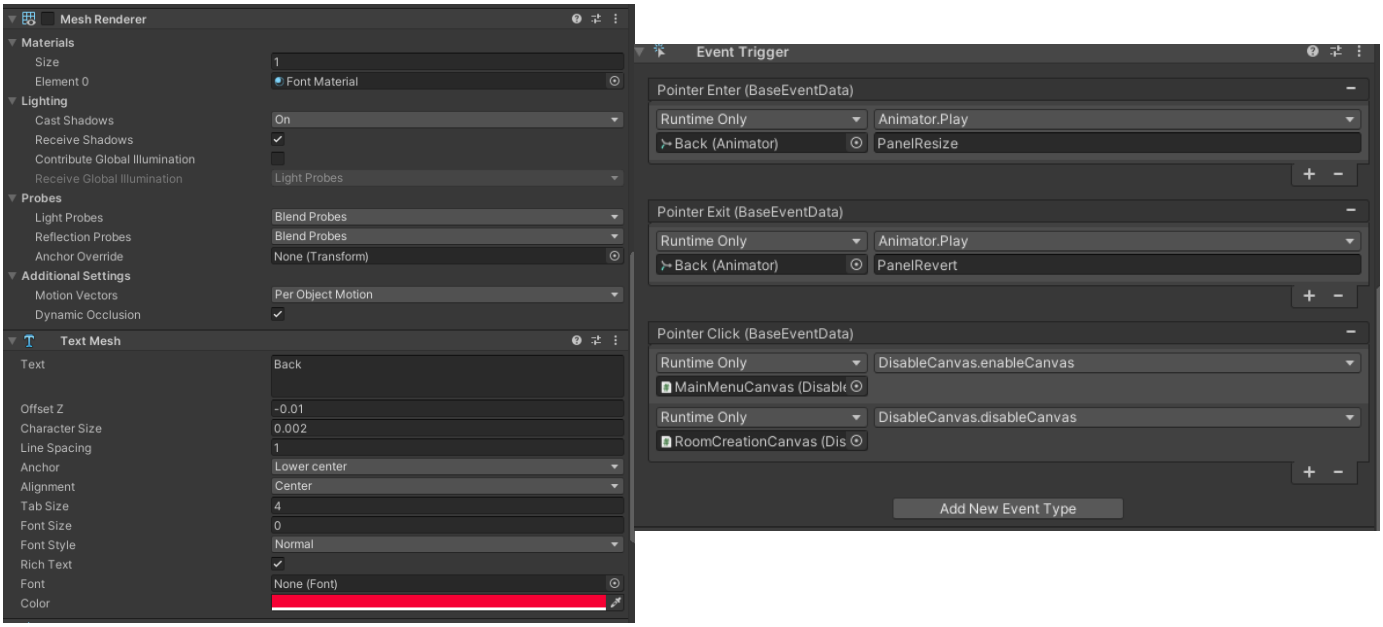


User Interface (Main Menu)

Panels:

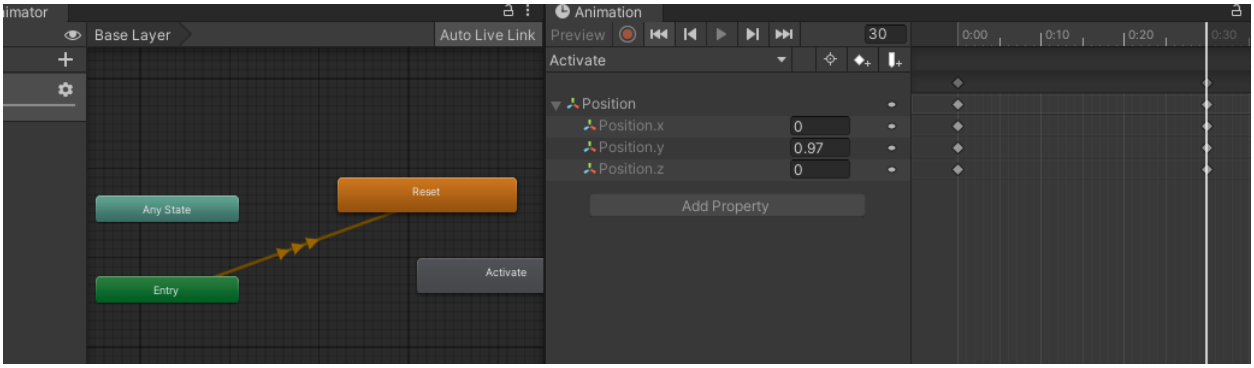
Settings for creating the panels in world space were done through the editor. Simple script created for activating and deactivating canvases and panels.

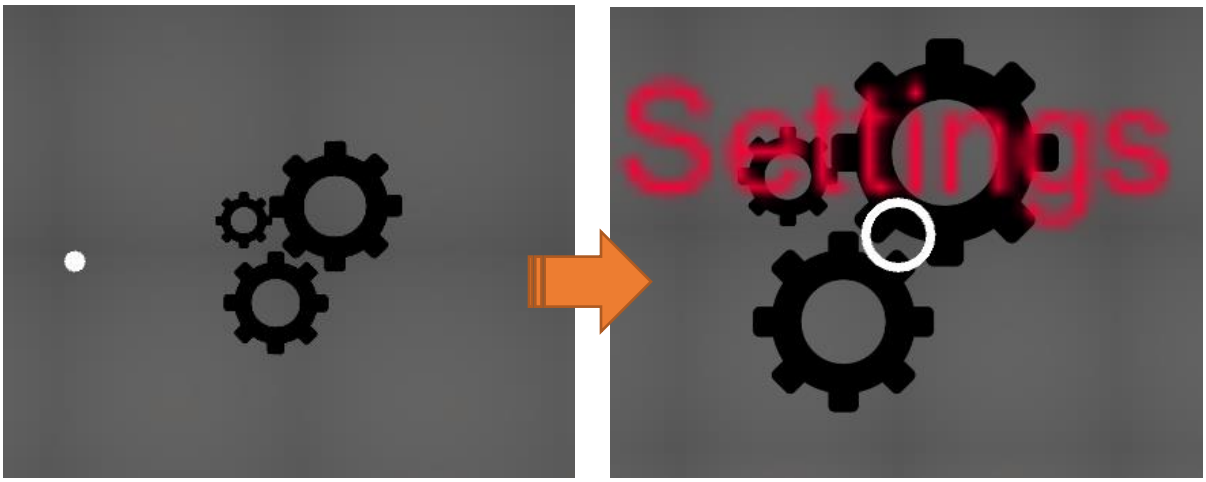




Animations:

All created using Unity's animation window and animator. Simple resizing of panel to enlarge and shrink depending on whether reticle is over it.





Sphere:

How I displayed the Sphere’s texture on the inside through reversing the normal map (without this the inside would not render and you would see right through the sphere removing the “room” effect)

```
[RequireComponent(typeof(MeshFilter))]
Unity Script | 0 references
public class ReverseNormals : MonoBehaviour
{

    Unity Message | 0 references
    void Start()
    {
        MeshFilter filter = GetComponent(typeof(MeshFilter)) as MeshFilter;
        if (filter != null)
        {
            Mesh mesh = filter.mesh;

            Vector3[] normals = mesh.normals;
            for (int i = 0; i < normals.Length; i++)
                normals[i] = -normals[i];
            mesh.normals = normals;

            for (int m = 0; m < mesh.subMeshCount; m++)
            {
                int[] triangles = mesh.GetTriangles(m);
                for (int i = 0; i < triangles.Length; i += 3)
                {
                    int temp = triangles[i + 0];
                    triangles[i + 0] = triangles[i + 1];
                    triangles[i + 1] = temp;
                }
                mesh.SetTriangles(triangles, m);
            }
        }
    }
}
```

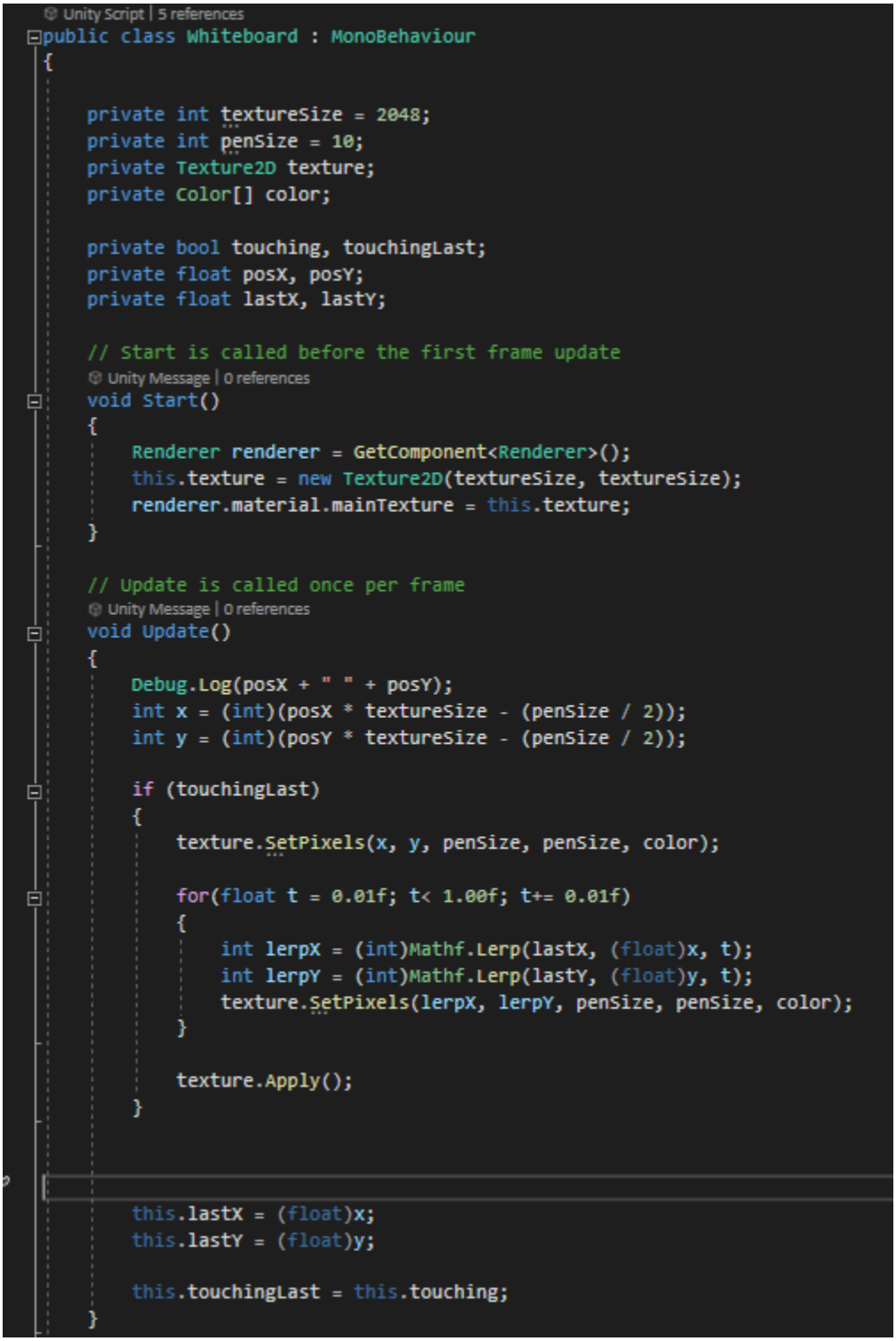
Whiteboard

Pen implementation. Works closely in conjunction with the whiteboard. Ray cast from tip of pen in the “up” or y direction is what triggers the whiteboard surface to change color. After “touching” the board, the pen attempts to lock rotation to keep it more stable.

```
// Start is called before the first frame update
Unity Message | 0 references
void Start()
{
    this.whiteboard = GameObject.Find("Whiteboard").GetComponent<Whiteboard>();
}

// Update is called once per frame
Unity Message | 0 references
void Update()
{
    float tipHeight = transform.Find("Tip").transform.localScale.y;
    Vector3 tip = transform.Find("Tip").transform.position;

    if (lastTouch)
    {
        tipHeight *= 10f;
    }
    if(Physics.Raycast(tip, transform.up, out touch, tipHeight) && Input.GetButton("Fire1"))
    {
        if (!(touch.collider.tag == "Whiteboard"))
            return;
        this.whiteboard = touch.collider.GetComponent<Whiteboard>();
        Debug.Log("touching!");

        this.whiteboard.SetColor(Color.blue);
        Debug.Log("Color set to blue");
        this.whiteboard.SetTouchPosition(touch.textureCoord.x, touch.textureCoord.y);
        Debug.Log("Position set to " + touch.textureCoord.x + " for x" + touch.textureCoord.y + " for y");
        this.whiteboard.ToggleTouch(true);


        if (!lastTouch)
        {
            lastTouch = true;
            lastAngle = transform.rotation;
        }

    }
    else
    {
        this.whiteboard.ToggleTouch(false);
        lastTouch = false;
    }
    if (lastTouch){
        transform.rotation = lastAngle;

    }
```

Whiteboard Surface

Takes the position of the pen's raycast and then if the pen is "touching" it will change the color at that position. Uses a lerp function to predict the next position of pen as it is moving.

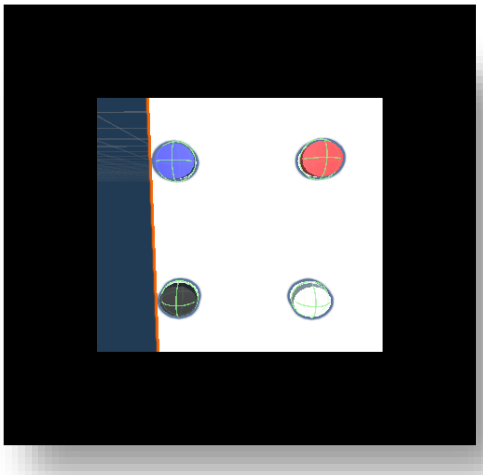


```
Unity Script | 5 references
public class Whiteboard : MonoBehaviour
{

    private int textureSize = 2048;
    private int penSize = 10;
    private Texture2D texture;
    private Color[] color;

    private bool touching, touchingLast;
    private float posX, posY;
    private float lastX, lastY;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        Renderer renderer = GetComponent<Renderer>();
        this.texture = new Texture2D(textureSize, textureSize);
        renderer.material.mainTexture = this.texture;
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        Debug.Log(posX + " " + posY);
        int x = (int)(posX * textureSize - (penSize / 2));
        int y = (int)(posY * textureSize - (penSize / 2));

        if (touchingLast)
        {
            texture.SetPixels(x, y, penSize, penSize, color);

            for(float t = 0.01f; t< 1.00f; t+= 0.01f)
            {
                int lerpX = (int)Mathf.Lerp(lastX, (float)x, t);
                int lerpY = (int)Mathf.Lerp(lastY, (float)y, t);
                texture.SetPixels(lerpX, lerpY, penSize, penSize, color);
            }

            texture.Apply();
        }



        this.lastX = (float)x;
        this.lastY = (float)y;

        this.touchingLast = this.touching;
    }
```

Fill board function is the function called when the buttons on the whiteboard are pressed.

```
2 references
public void ToggleTouch(bool touching)
{
    this.touching = touching;
}

1 reference
public void SetTouchPosition(float x, float y)
{
    this.posX = x;
    this.posY = y;
}

1 reference
public void SetColor(Color color)
{
    this.color = Enumerable.Repeat<Color>(color, penSize * penSize).ToArray<Color>();
}

1 reference
public void FillBoard(Color color)
{
    this.color = Enumerable.Repeat<Color>(color, textureSize * textureSize).ToArray<Color>();
    texture.SetPixels(0, 0, textureSize, textureSize, this.color );
    texture.Apply();
}
```

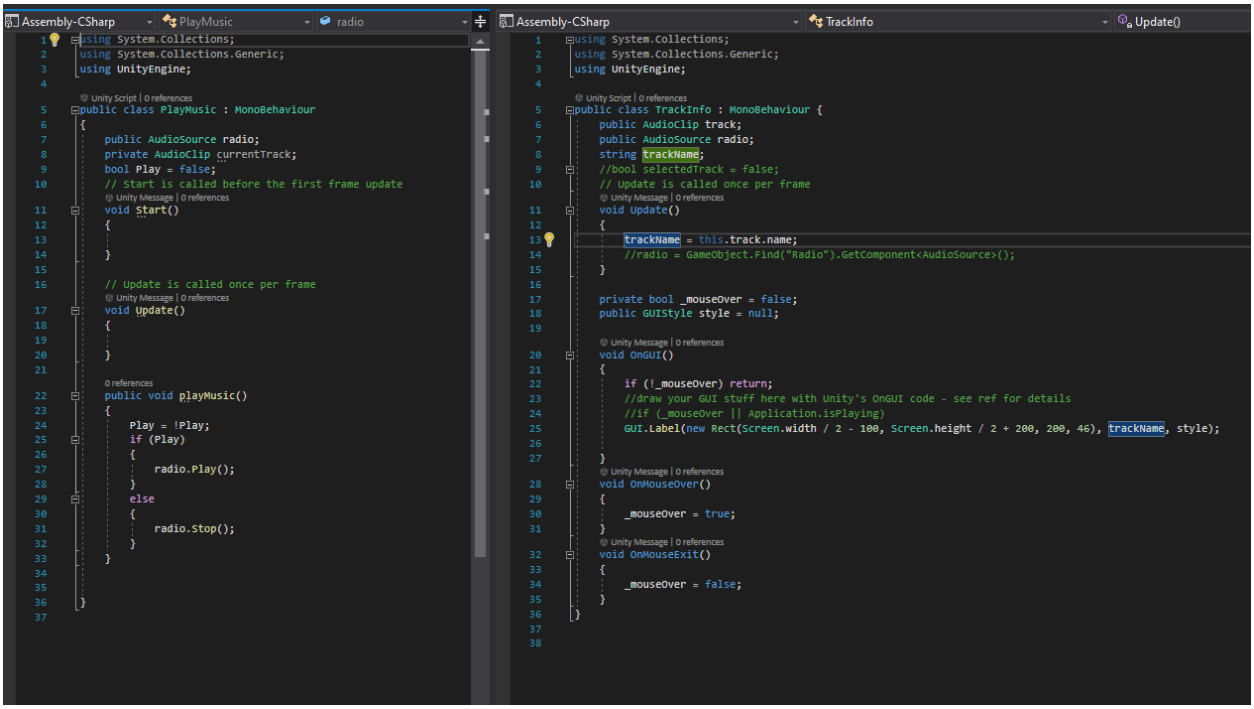
Buttons:
Animated in similar fashion to panel and uses same event trigger system. Created prefabs that set the color based on the button pressed. Uses the “this” keyword to ensure the correct button color is used.



Music

Track selection and Start/Play:

Used Jose’s prompt script and modified it to display track information while hovering over a button. Script mainly used for storing track information to display and let Unity know which track is currently loaded into the audio source.

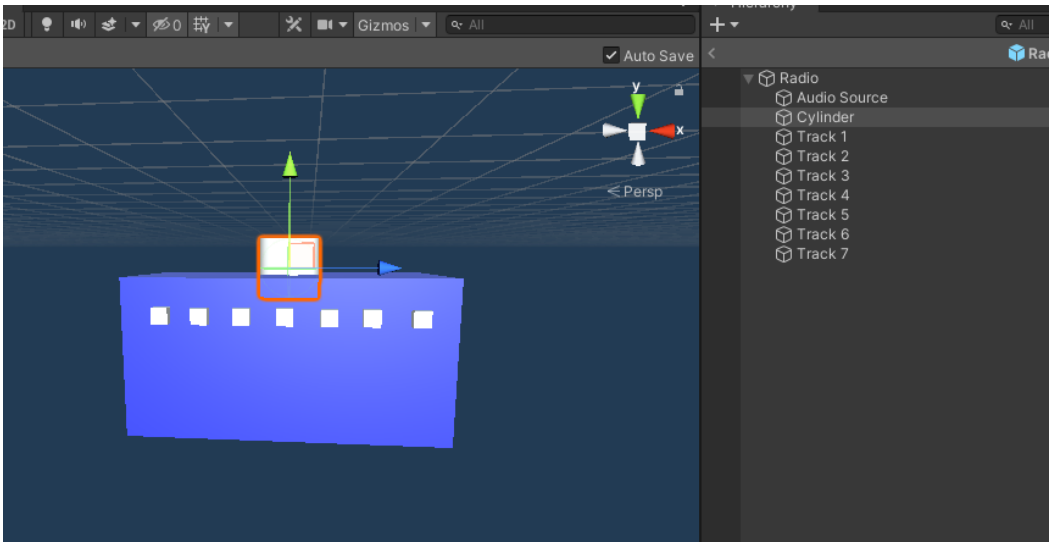


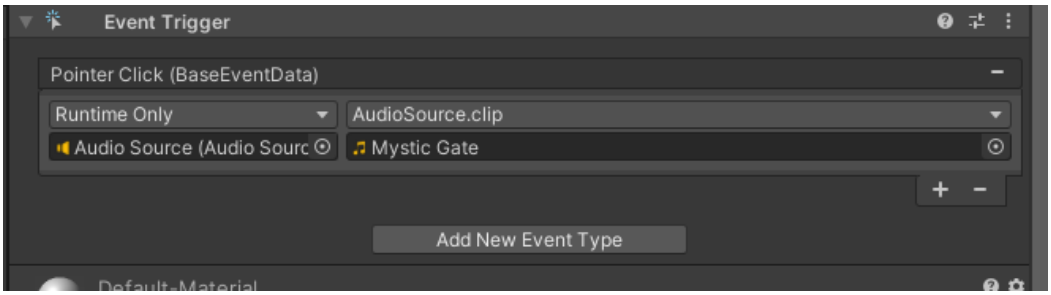
```
Assembly-CSharp | PlayMusic | radio

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script | 0 references
public class PlayMusic : MonoBehaviour
{
    public AudioSource radio;
    private AudioClip currentTrack;
    bool Play = false;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {

    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {

    }

    0 references
    public void playMusic()
    {
        Play = !Play;
        if (Play)
        {
            radio.Play();
        }
        else
        {
            radio.Stop();
        }
    }


}
```

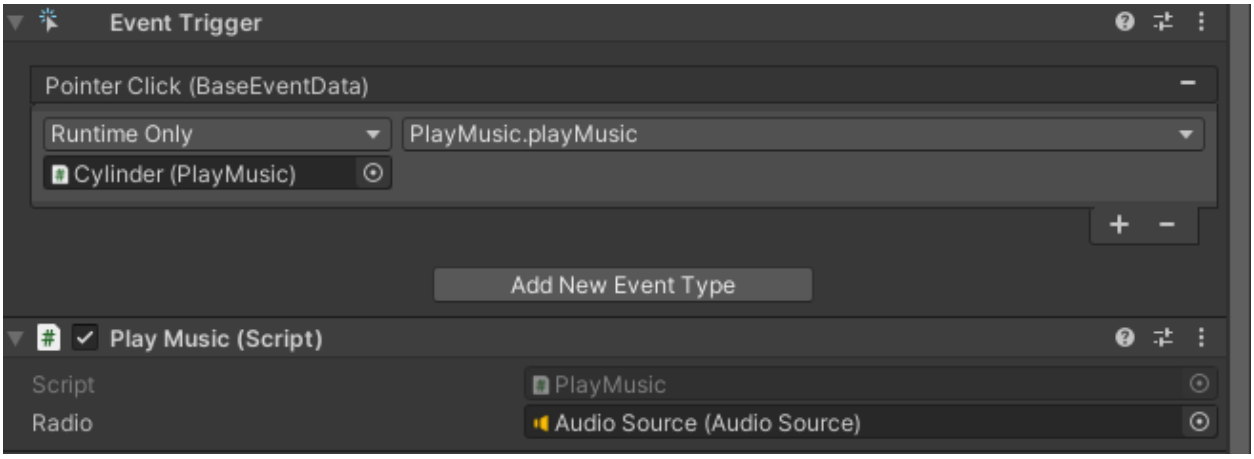
```
Assembly-CSharp | TrackInfo | Update()

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script | 0 references
public class TrackInfo : MonoBehaviour {
    public AudioClip track;
    public AudioSource radio;
    string trackName;
    //bool selectedTrack = false;
    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        trackName = this.track.name;
        //radio = GameObject.Find("Radio").GetComponent<AudioSource>();
    }

    private bool _mouseOver = false;
    public GUIStyle style = null;

    Unity Message | 0 references
    void OnGUI()
    {
        if (!_mouseOver) return;
        //draw your GUI stuff here with Unity's OnGUI code - see ref for details
        //if (_mouseOver || Application.isPlaying)
        GUI.Label(new Rect(Screen.width / 2 - 100, Screen.height / 2 + 200, 200, 46), trackName, style);

    }
    Unity Message | 0 references
    void OnMouseOver()
    {
        _mouseOver = true;
    }
    Unity Message | 0 references
    void OnMouseExit()
    {
        _mouseOver = false;
    }
}
```

Simple Radio Prefab:

Did not have time to animate it. Created a “Radio” with simple shapes, each smaller cube is assigned a track, larger cylinder button is used for pressing play/stop based on script above. Used in conjunction with visual event trigger system.







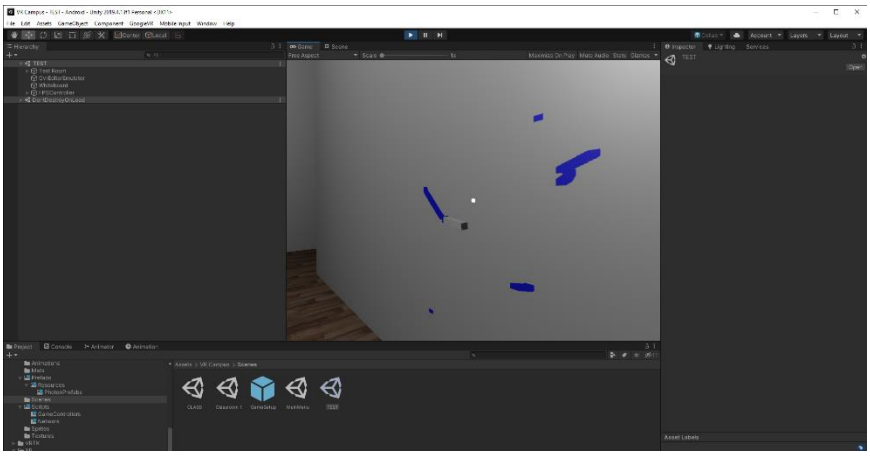
V. Progress Reports

Progress Report 11/19/20

1) Progress from last meeting:

a. I have implemented a functional white board that reacts to a "pen". Currently I have this linked to the camera of the player and it is not yet been implemented on the network. I am considering placing the pen in the center of the reticle while drawing until a better means of control is implemented.

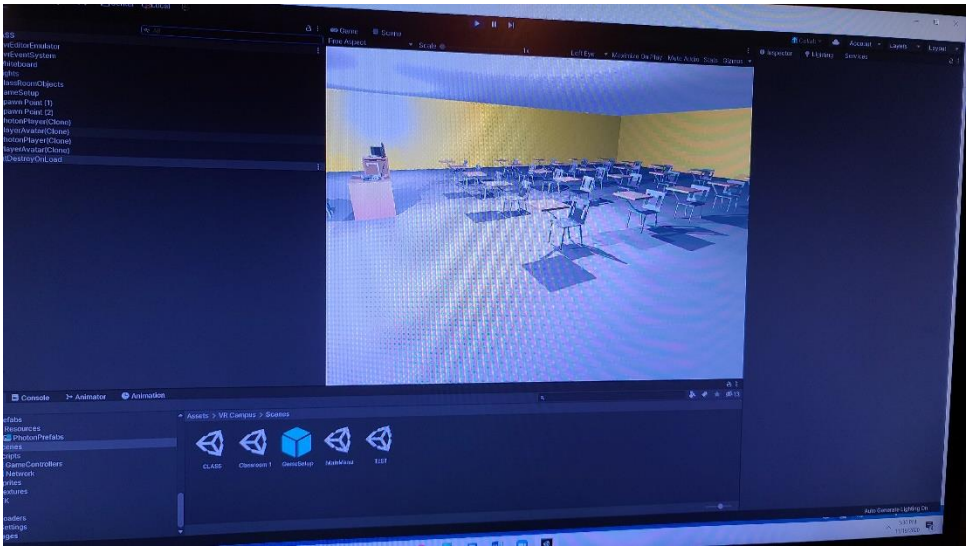
i. This was implemented through a system of a physics raycaster from the tip of the pen to the "whiteboard" tagged cube object. Grabbing the renderer component and texture size, I am able to create a "square" of color change on the texture of the whiteboard. Using the native interpolation functions of Unity I am able to smooth the line through counting 100 steps.

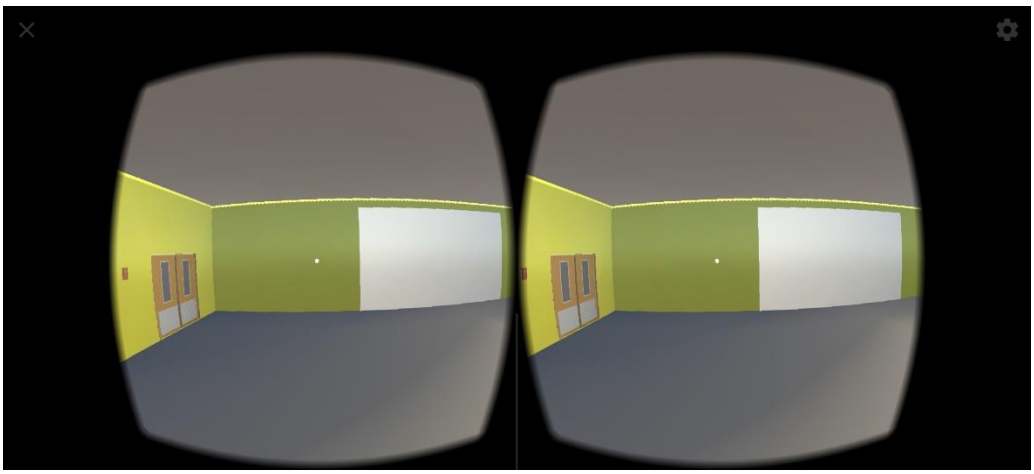


b. I have fixed the issue where the person who starts the server loses control of their camera and becomes a “ghost”. However I have yet to fix this on the second or other players that subsequently join the server.

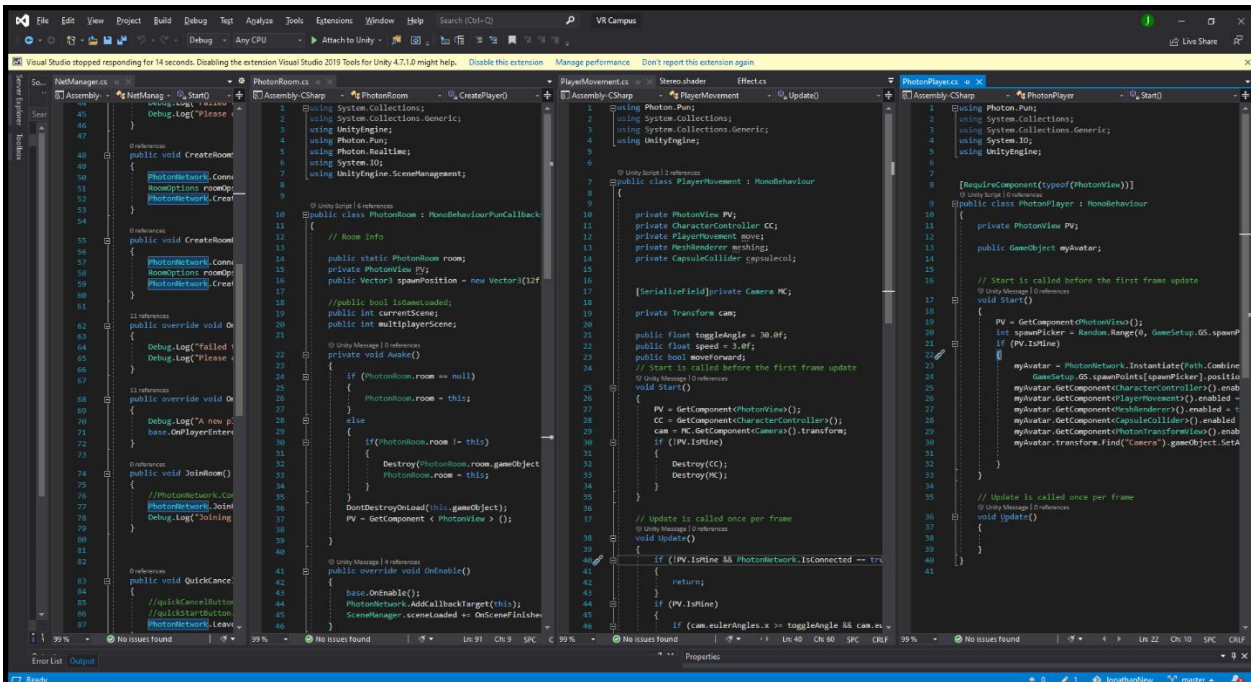
i. Grabbing the components that make up the player prefab and disabling them seemed to have assisted with the issue. By having everything disabled unless the “photonview.IsMine” it helped solve some of the player control issues with the network.

ii. On my final run, it seems like both players are now in the scene only I cannot see the second player yet.





iii. Modified most of the player movement, network manager, and photonroom scripts. Also made modifications to player prefab and instantiation process.



c. Currently in communication with some of the staff from ManoMotion and some of their community in order to attempt integrating it into our project. The complexity is a bit higher and documentation not in depth enough with no tutorials available on line. I have managed to get the sdk imported into the project. There were many issues getting it to work with the GVR but I managed to get the sample scene functioning after a while of tinkering.

i. The GVR does not natively support Manomotion, but there are work arounds. I have managed to track a person down on discord that was able to implement a

mapping of the hand position in Manomotion into the virtual world space. I will try to see if I can ask for some of his time and listen to how he managed to do so. This is dependent on his schedule, but hopefully he will be able to spend a few minutes to help guide me.

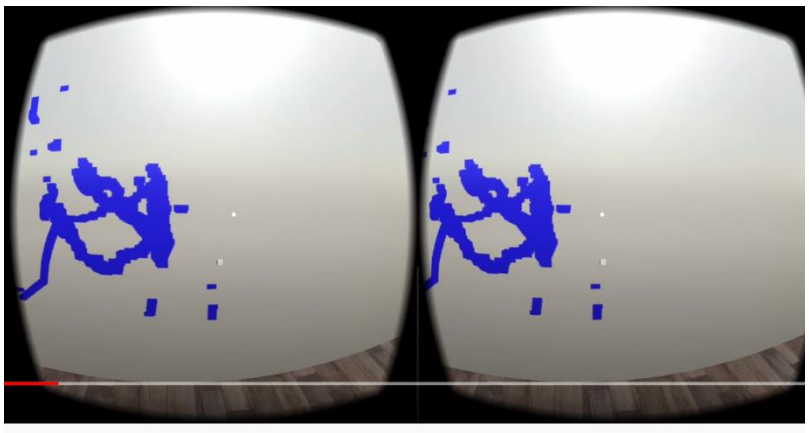
1. Your work plan until the next meeting:
    a. I plan on tackling the networking problem again from a different angle. I think the issue may be in the implementation of the join function. I will also seek out developers online who have used PUN 2 before for some direction/advice.
    b. I will continue working on implementing the Manomotion SDK. Learning all these new technologies with the limited time we have is not easy but I believe it is still possible to implement.
        i. Should the Manomotion sdk be too much to implement, I will seek other options for controlling the pen onto the whiteboard.
    c. I want to begin looking at implementing ambience/music into the application.
    d. Continue working on the lobby/room system.

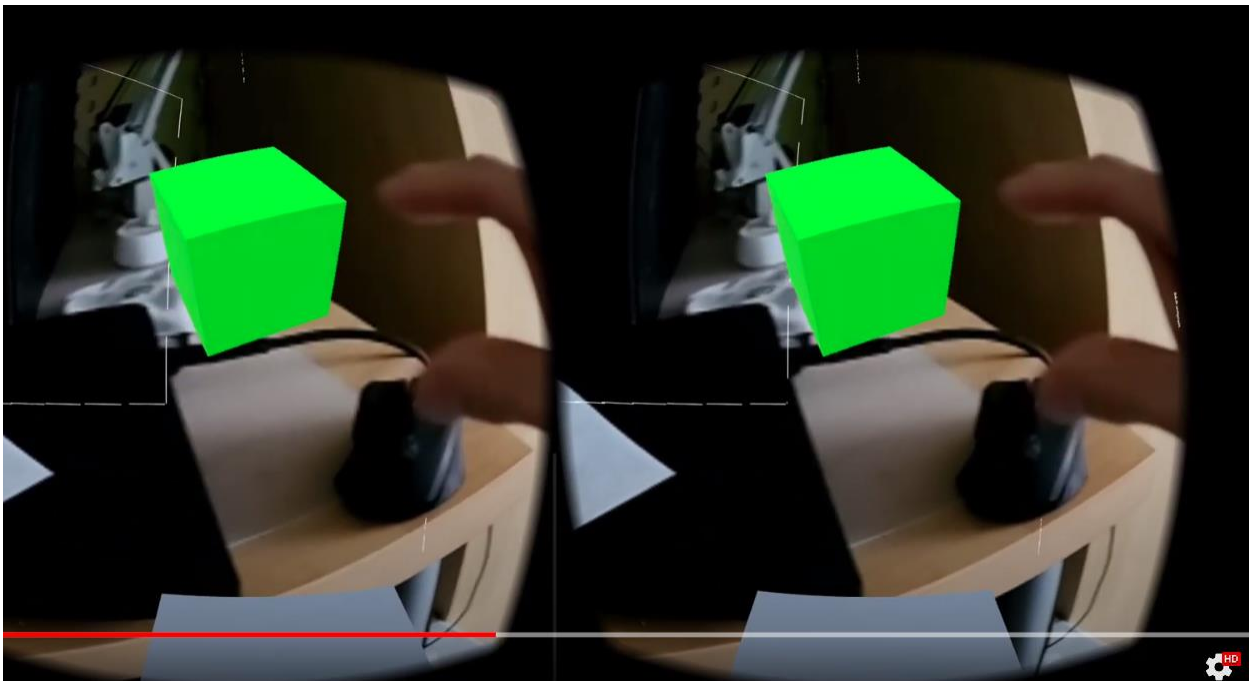
Progress Report 11/24/20

1. Progress from last meeting:

e. Implemented Manomotion with “grab” and “click” gesture. The GVR and Manomotion libraries do not mesh well together. It took a bit of “hacking” but I managed to get the Manomotion to render behind the main camera and using the location of the palmcenter I can have the “Pen” follow it. For depth estimation I use a function included with Manomotion that estimates the depth of the bounding box. The palmcenter is located in the bounding box. The click gesture will be useful for selecting or interacting with objects. That in conjunction with the Manomotion tracking should allow for some neat interactions. As this implementation took a lot longer than expected, I was unable to implement any other feature.

f. Created a Prefab of the functional Manomotion components and scripts to easily insert into the scenes needed for our VR campus.





2. Your work plan until the next meeting:

a. My plan going forward will be to make improvements with the manomotion interaction. I plan on using the click feature in conjunction with some sort of “tool tip” menu. The click can then select a tool or action, for example to grab the pen, (or maybe turn it into a pen?).

   i. Implement a “lock on” feature so that the pen is able to be steadier and not go through the wall.

   ii. Implement an erase board button.

   iii. Implement synchronization of whiteboard object as well any other object like the pen.

b. Want to implement a tool tip or action menu.

i. Need to ensure that the tool tip is only visible to the player who opens the menu.

c. Need to include music and some sounds, work on the voice chat.

i. Make music local to the player only or allow only in study room through an intractable “radio”?

d. Continue working on network features and make sure users can see the other users while disabling their own meshes to help with any performance issue.

i. Hopefully with help from others we can create some sort of database to store user preference and login information.

e. Try to implement some sort of text reader by rendering the text into a texture and display it on a surface. Could possibly also do that with other files.

i. Find a way to convert files into render-able image (if possible in time frame that is remaining.)

Progress Report 12/01/20

1. Progress from last meeting:

a. Since the last meeting I have been working on improving the Manomotion integration and whiteboard. Along the way I have

encountered many difficulties in switching between a “Pen” and hand.

i. Attempting to switch between two objects was causing some unforeseen consequences. Whenever I would switch between the pen and the hand, the hand would disappear and not reappear.

1. Initially used the “gameObject.Setactive(true/false)” function to have the hand “disappear” when the pen was active. This however never worked no matter what I seemed to do.
2. After that I attempted using the “gameObject.getComponent<MeshRenderer>.enabled” and set it to false or active depending on whether the whiteboard pen was “grabbed”. This also created issues because it would never reenable the meshrenderer properly.
    a. This also had the effect of causing the pen and hand to “collide” and behave erratically.
3. Finally, decided to go with a “Destroy(gameObject)” route. This seems to have remedied many of the issues I was initially

facing. However I had some logical errors in my code. I also read that destroying an object helps keep resources low to not impact performance which is a good thing considering the hardware being used.

a. My error included forgetting to check for when the pen was grabbed before instantiating a new instance of the hand. I only realized this when I noticed some stray hand objects floating in the middle of the room.

b. Attempted to create an "erase/fill" button for the whiteboard. Made an animation that is activated when a hand "presses" the button through use of collision detection.

i. This would then cause the board to fill in with the color that is set in the code.

1. Ran into issues because of the hand not functioning properly.

c. Due to the amount of time I spend debugging the hand and whiteboard, I was unable to complete the other features I had wanted to include and must admit I am being too ambitious with everything I wish to implement versus the

amount of time I have to do so as well as the lack of help from my team members and our various schedules.

For this week I do not have anything to "show" as some of the features broke while adding in the virtual hand to the player avatar. Hopefully will have fixed those issues by the next class to continue polishing the features.

2. Your work plan until the next meeting:

   a. My plan is to incorporate a color wheel to allow the selection of color. I have found a free asset which allows you to select color and I wish to implement it.
   b. Need to create a tool tip that will allow me to select the action I wish to perform instead of relying on the gestures solely.
      i. The way I can see this functioning is by having the tool tip menu appear in the world space around the characters avatar, disabling the movement script while the tool tip menu is open. Freezing the tool tip menu in place.
         1. Pressing and holding the google VR button for 1 second will open the menu.

2. Reticle will hover over selection and then the "Pinch" gesture can be used to select the action.

c. Based on the amount of time remaining in the semester, I do not believe all the multiplayer functions will be working properly. After figuring out the manomotion for a single person, it will then need to be implemented in a network environment which I anticipate will take a lot of time for a solo developer.

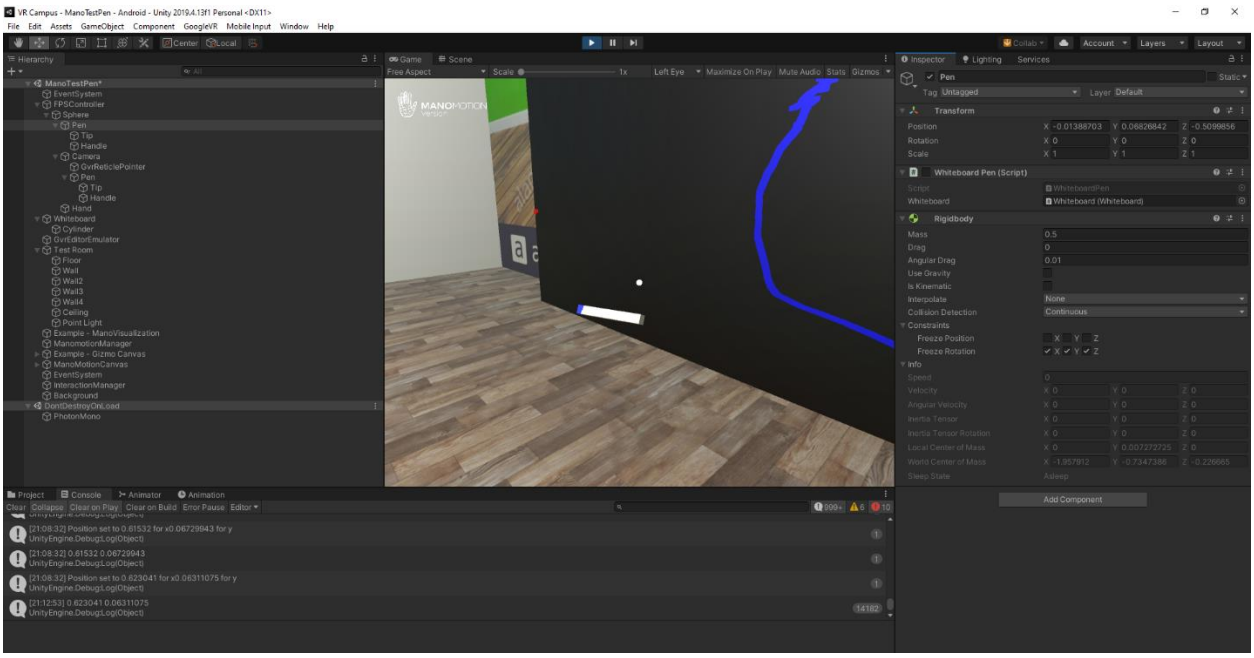
d. Want to still include a simple "music" player through a physical interaction by creating a "radio" prefab with a button.

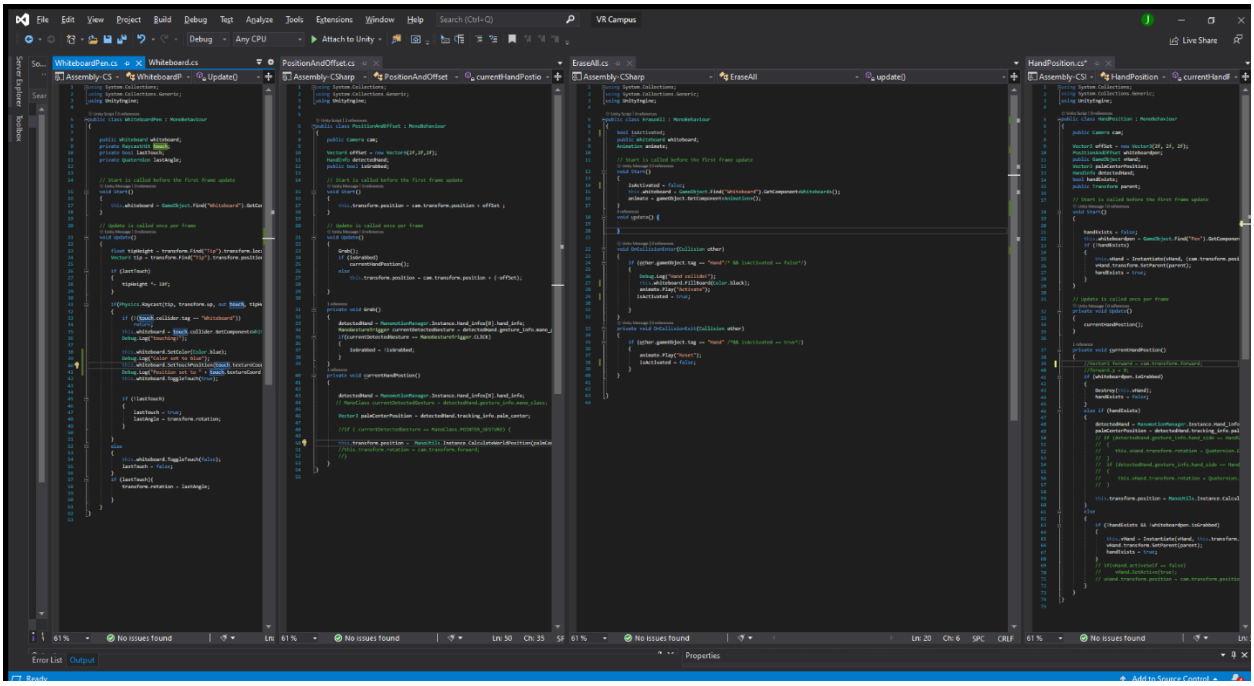
i. For the radio, I will include only one track for now if I am able to get to it before the semester ends.

UPDATE!!!! After lecture

Figured out what was causing the problem in my code. Because I was testing with a "Pen" object in editor I added in a SECOND "Pen" object. With my codes logic, it was looking for an object of name "Pen", since there were two it caused the whiteboard to stop functioning properly.

Discovered what was causing button to remain stationary and not stay in location I had assigned it. Because it had originally been created with a parent object, when I decided it wasn't necessary to have a parent object and unparented the button, it caused some logic error. I had to check the "Use Root Move" check box which then fixed the issue with the button!





Progress Report 12/3/20

1. Progress from last meeting:
    a. Implemented Manomotion more whiteboard features. Added four buttons allowing the user to change the color of entire board. Had difficulty getting it to work. No matter how often I would click the different buttons, the

only color change that would happen would be to black which was the first button I made. Played around with the color settings and the alpha channel to no avail. After some thinking, I realized this problem was similar to the one I encountered with the Pen issue prior. I needed to specify which instance of the variable "buttonColor" to use. By adding the keyword "this" in front of the variable name it solved the issue of the color change.

i. Added animations to the buttons. Found that by adding a parent object, it behaved much better than not having one. When I would do a solo object and animate it, it would lock the object to a certain position.

1. This issue was resolved by creating the animation with the object having an empty parent. Using the animation window, I added in the keyframes.

a. One other note, the object also needed to have the animation set to legacy by clicking on the more options button in the inspector and changing it to debugging mode.

2. Created one animation to activate, for when a collision with a "Hand" object is detected. Made it loop one time. Another animation for it to reset, when the "Hand object stopped colliding with it.

3. Created a public "Color" field that allows the button to change the color based a preset color picked through the unity editor.

b. Created a radio object that can play music based on a pressing a button.

i. The main button on the radio will cause the music to start or stop (from beginning, have not tried the "PlayClipAtPoint()" function yet.

1. **Button takes an audio source object attached to the radio parent and calls the Play() function if a bool is true, or the Stop() function if a bool is false.**

ii. Using the "prompt" script Jose created, I modified it to suit the radio so that the side buttons display what track is associated with the button.

1. **Currently selecting a new track will change the audio sources music clip, but it will stop the music in the process. This means the user will need to press the main button again to play the clip.**

iii. The music implementation was fairly simple. Have not tested whether it is a global sound or localized sound as the classroom and study room are too small to determine if distance would affect the sound. For the purposes of this application however, it is fine.

c. **Created a prefab of the whiteboard, whiteboardpen, and radio objects.**

i. Integrated these objects into the study room that Jose created. Overlayed it over the whiteboard he positioned and modified the size so that it would fit accordingly.

1. **Assisted Jose with how to create a "Text" on the bulletin board in the classroom using a worldspace canvas with a panel and text object.**

2. Tasks to complete for next meeting

a. **With the semester at an end our team is now shifting over to creating a simple prototype integrating the parts we have each worked on individually.**

i. For the most part, it has been integrated with Jose's scripts and models, as well

as Yongfeng's "seeing VR". Awaiting Sergey to try and integrate the animations and controllers he is working on.

b. **The next major focus will be on our presentation. We each will work on our individual portions as well as the four min group portion.**
    i. The project has taught me a lot in terms of the flow and life cycle of software development.