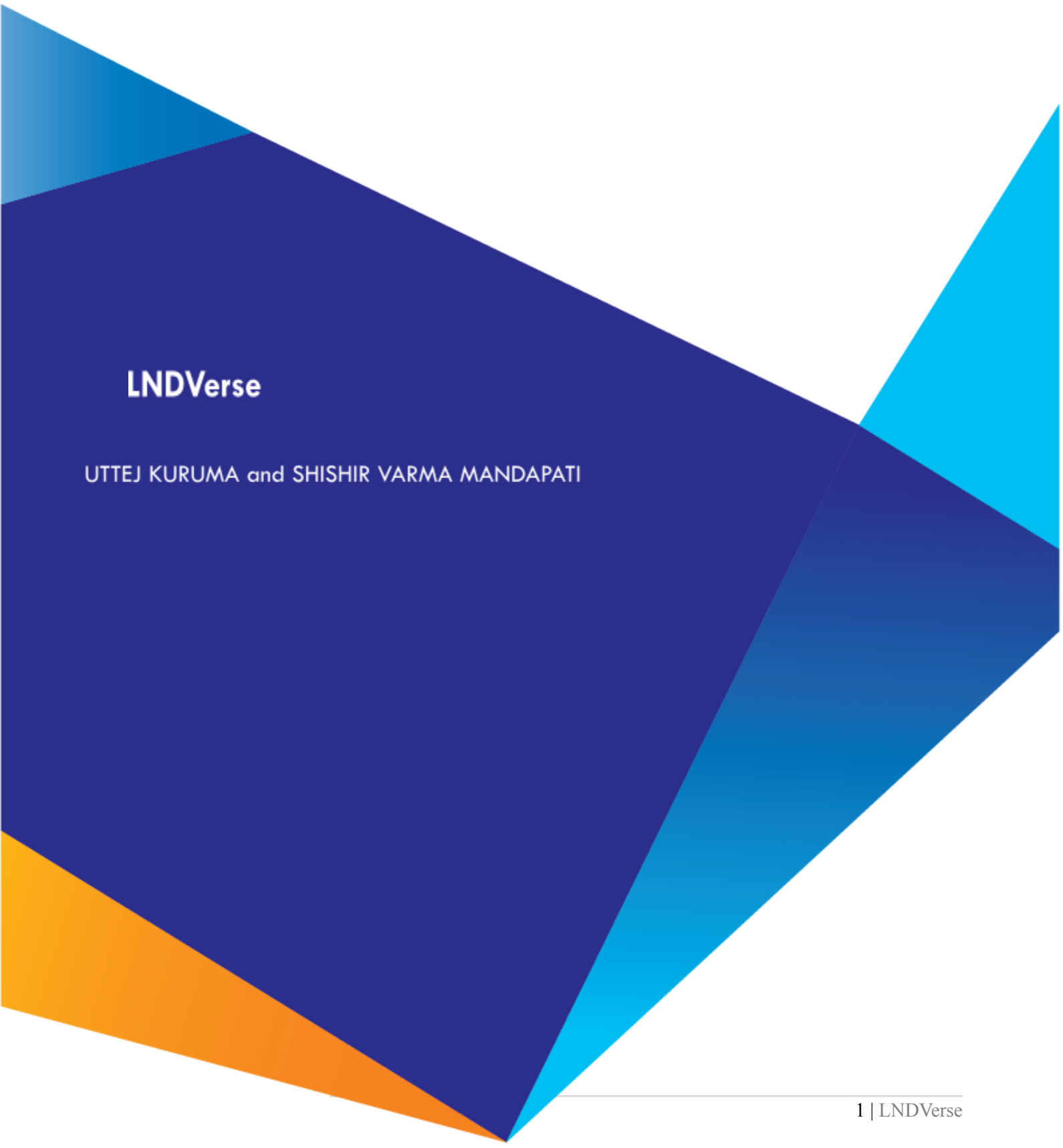


LNDVerse

UTTEJ KURUMA and SHISHIR VARMA MANDAPATI

Record of Release:

Version No.	Modified By:	Reviewed By:	Authorized By:	Release Date	Modifications Done
1	Uttej K			2-12-2022	Document Created
2	Shishir VM			9-12-2022	Document edited

Contents

1. Design Phase Introduction

The design Phase is all about how the environment is built. In this documentation we cover the following steps:

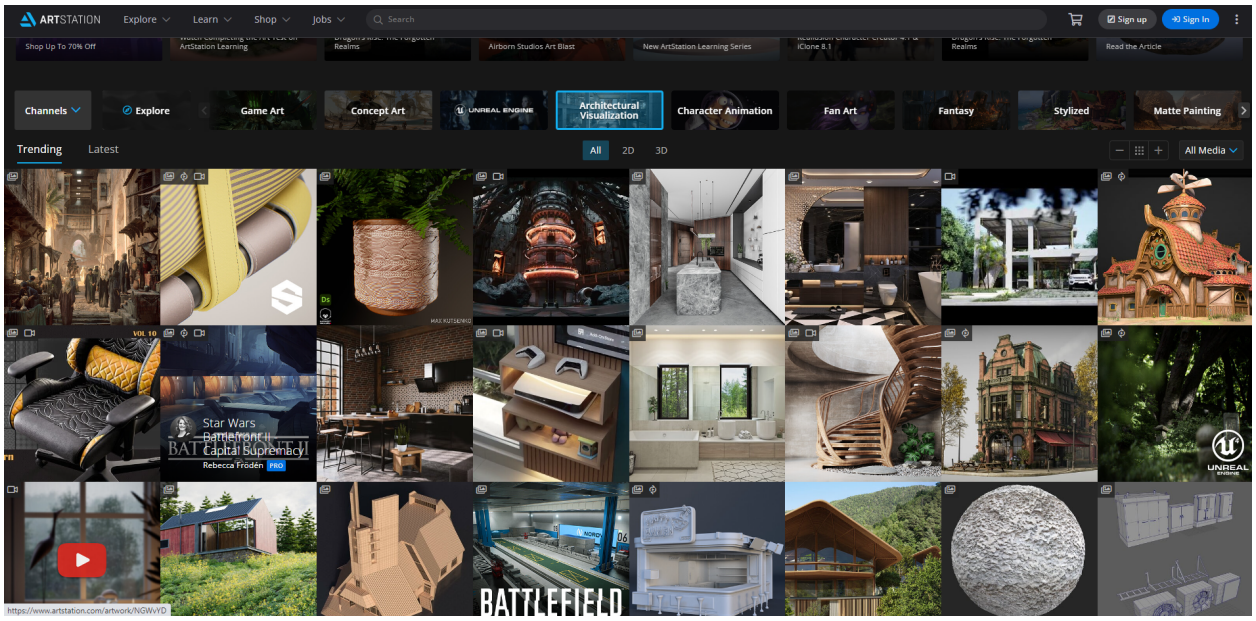
1. Reference Collection
2. Tools required
3. Environment set-up

1.1 Reference Collection

This Can be considered the most important Phase of the process. To create any environment of the style 'realistic', Reference Gathering is VERY Important. Some amazing resources would be:

1.1.1 Artstation.com

A website dedicated to artists irrespective of the tools they use.



You can navigate to the 'Architecture Visualization' section and find the images that suit your needs.

1.1.2 Blenderartists.org

A website where Blender users come together and share their progress online. Finding relevant images here might be a little challenging but it's worth it.

1.1.3 Stock Image sites

Sites like Pexels, Pixabay or even google images are sometimes the best sources for images too.

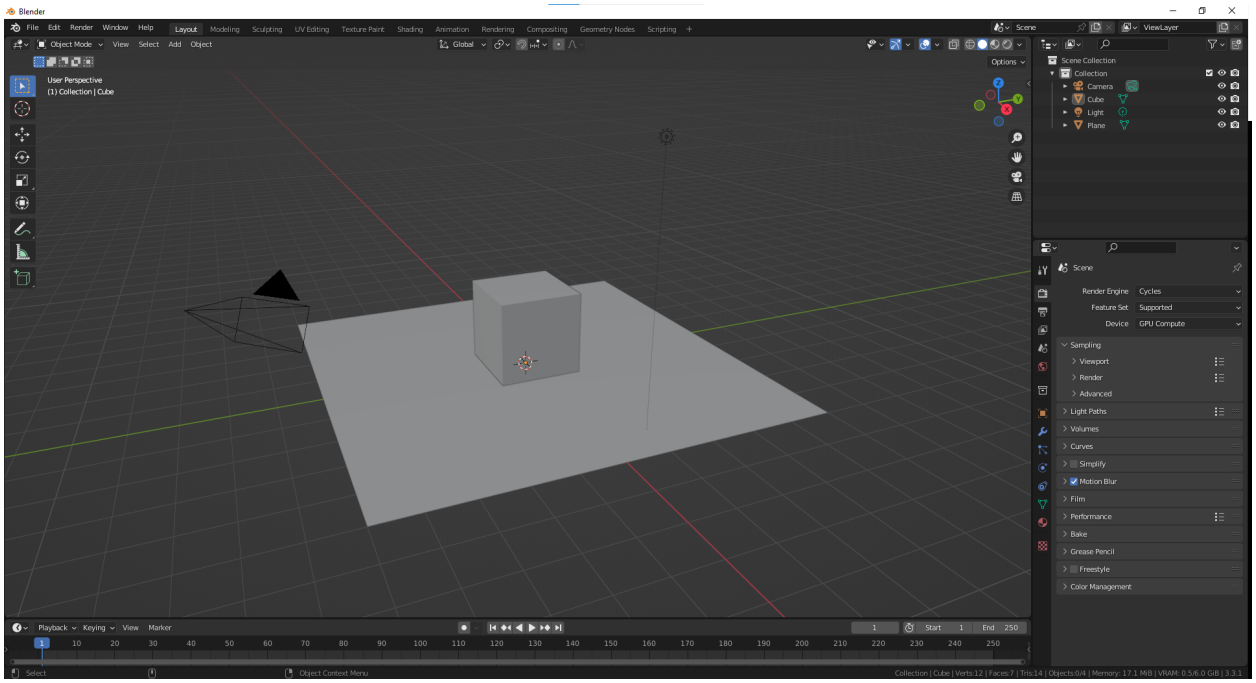
1.2 Tools Required

For this project we require a few tools:

1.2.1 Blender

A free 3D software, that can be downloaded from blender.org.

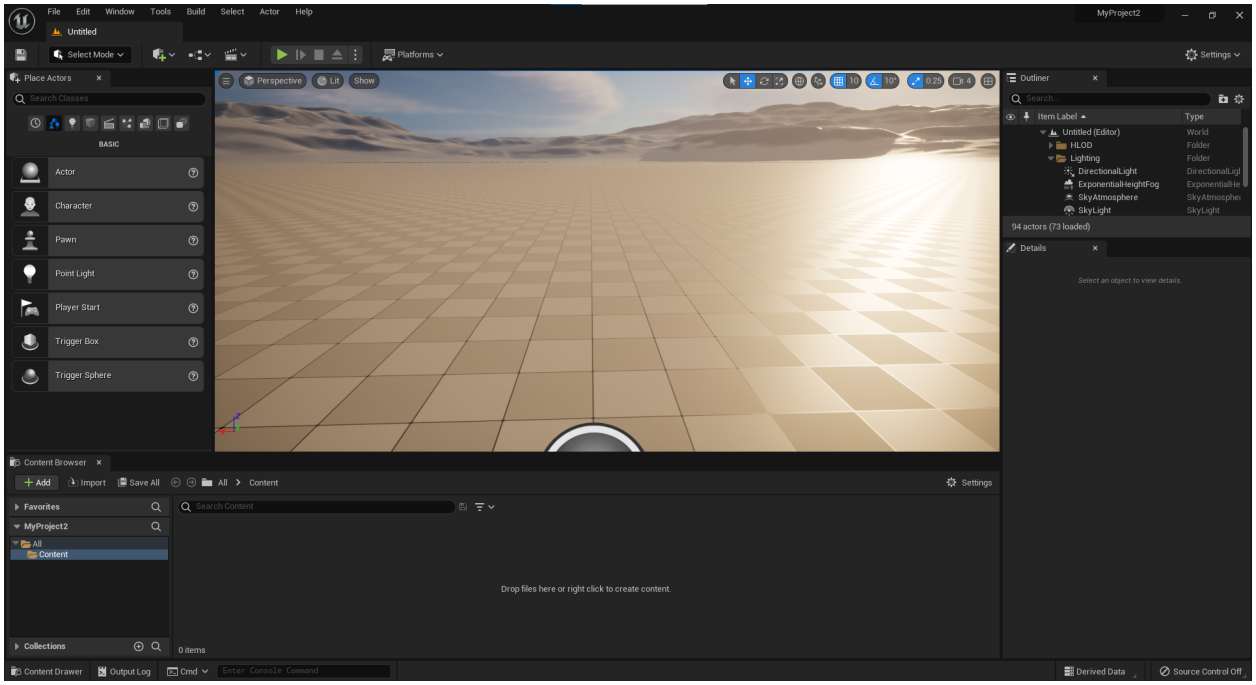
Learning this software is easy considering the number of free resources available on YouTube or anywhere on the internet.



One of the most recommended resources to learn blender would be the donut and chair tutorial from Blender Guru (Andrew Price).

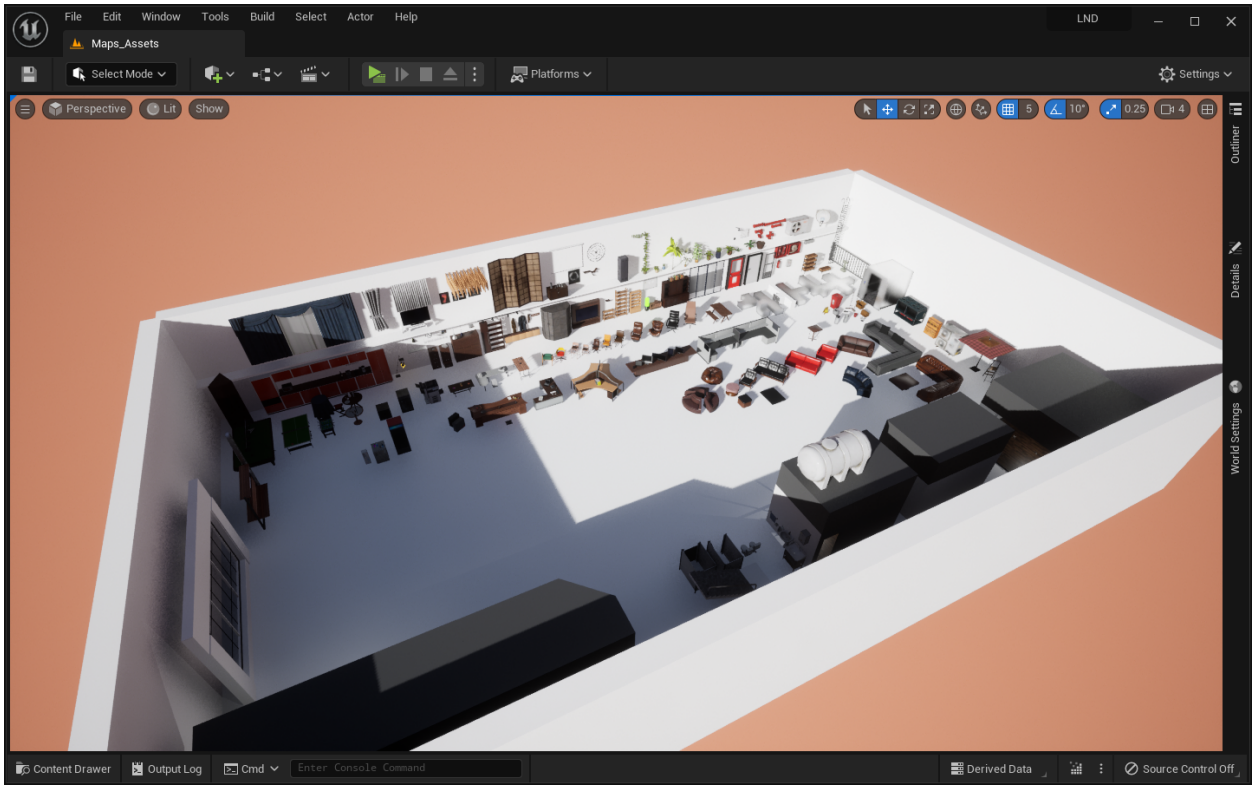
1.2.2 Unreal Engine

This is a game engine used to make some AAA Quality games with creative freedom for the artists as well as the developers. Learning curve for Unreal Engine is hard but also fun. A tutorial that can help started would be by [Unreal Sensei].

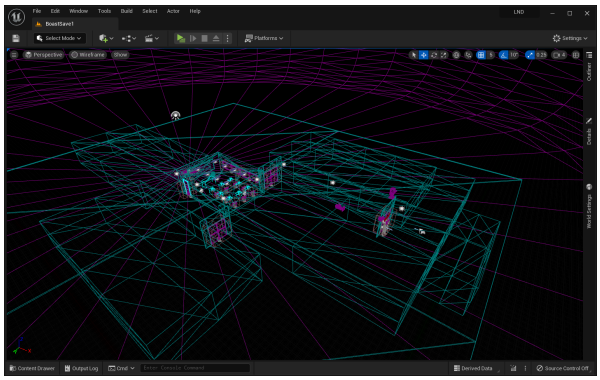


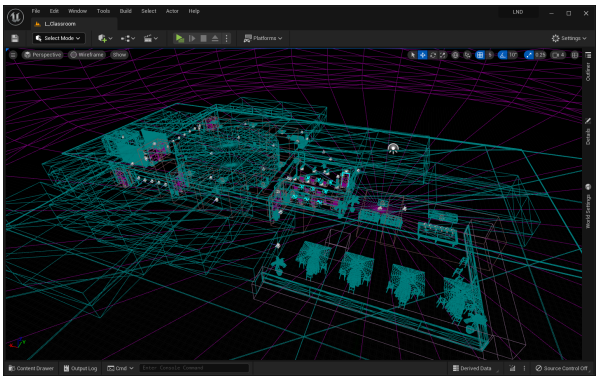
1.3 Environment Set up

We begin by creating an empty map/level, this is where most of the environment setup happens. We collect all the models we'll be using into one level so that they can be referred to whenever necessary. And importing models is as simple as dragging the FBX files right into the content drawer inside unreal engine.



We can either use the [cube grid tool] in Unreal engine or just manually place cubes around the scene to create a rough block out how we want our space to look like. I did the manual method of placing cubes all over the map to create the current version of the level. The image below (left) shows how the block out looks. And after the block out was ready, and after a few reviews from my superiors, we decided to change the design a bit (image to the right).





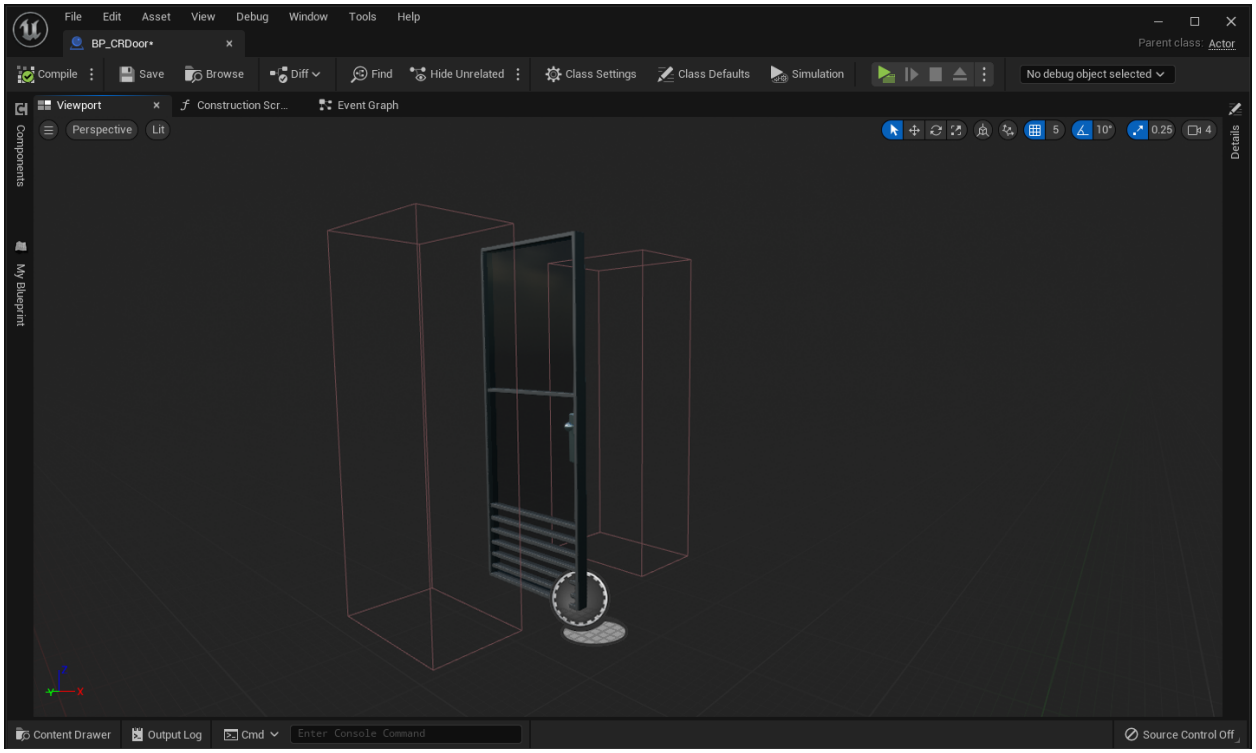
I won't be covering on how I modelled the assets or how I set up the rooms, that is something an individual must come up by themselves.

1.4 Blueprints Created

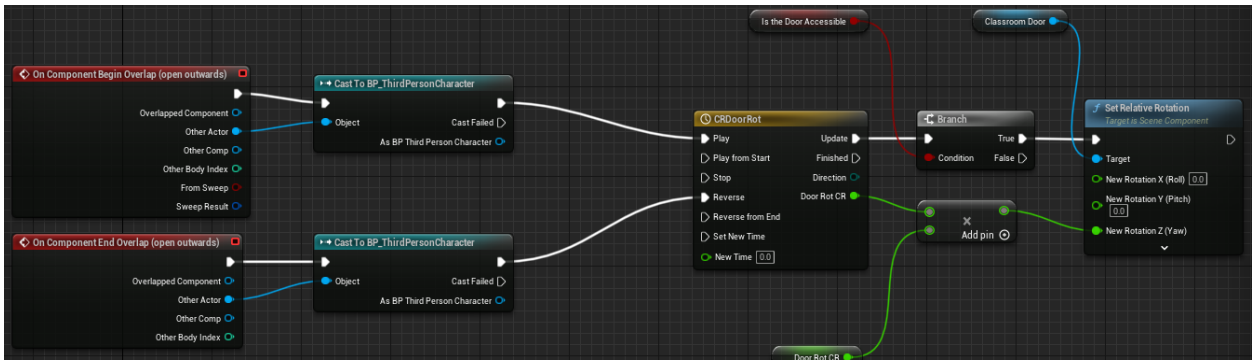
This would be a short explanation on how we can create the blueprints that are responsible for most of the user experience in this project.

1.4.1 Creating the doors

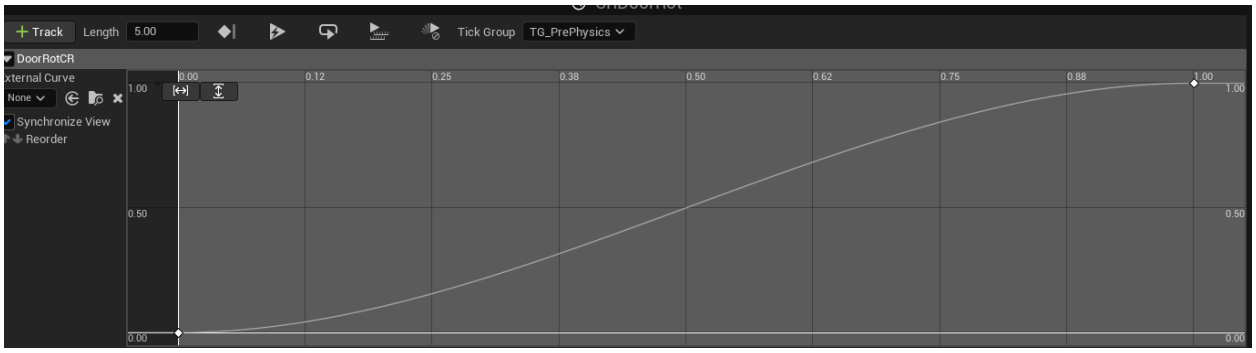
Every **Door** in this environment use a system called a blueprint to help us navigate into the room.



we begin by adding two box collisions to check if the player is near the door, and use the on begin overlap and on end overlap triggers from the collision object to drive the rotation of the door as shown in the Blueprint (BP) below:



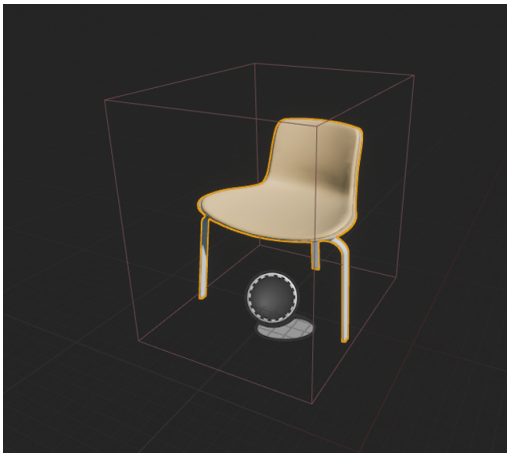
The timeline is nothing but a simple graph that smoothens the rotation of the door. I've done the same setup twice to influence the doors direction based on the user's location. The other object node must be connected to the third person BP to make sure the collision is only tested with respected to the character and not any other movable object.



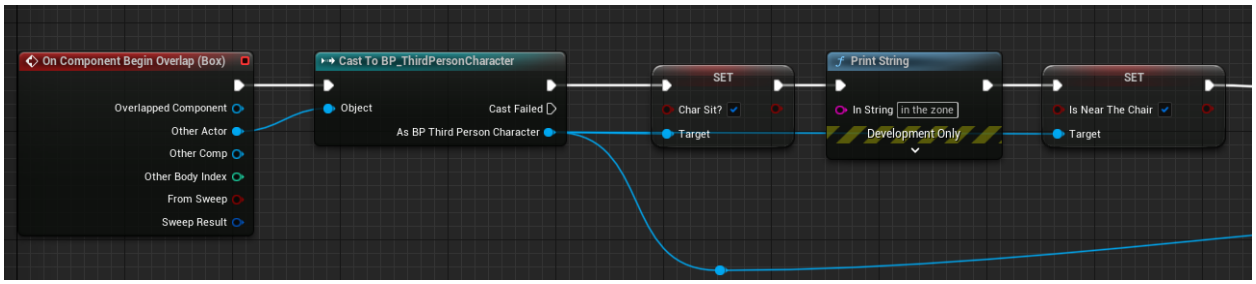
This is the timeline, which basically shows the ease-in graph. You would've also noticed that I used a Boolean function saying is the door accessible to only allow people to go into the room if they're allowed. Since this is a metaverse, that condition can be used to check if the user is registered into the training or not and then only let them enter the room.

Every chair in this room is also a BP object. It basically sets the characters location and rotation to be right in front of the chair so that the users are only able to use the sit functionality when in front of the chairs.

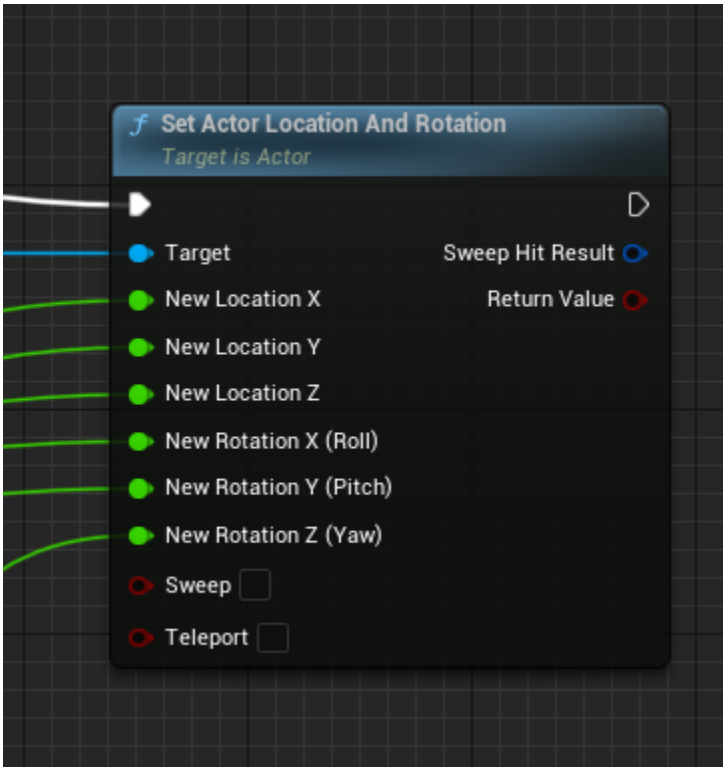
1.4.2 Creating the chairs

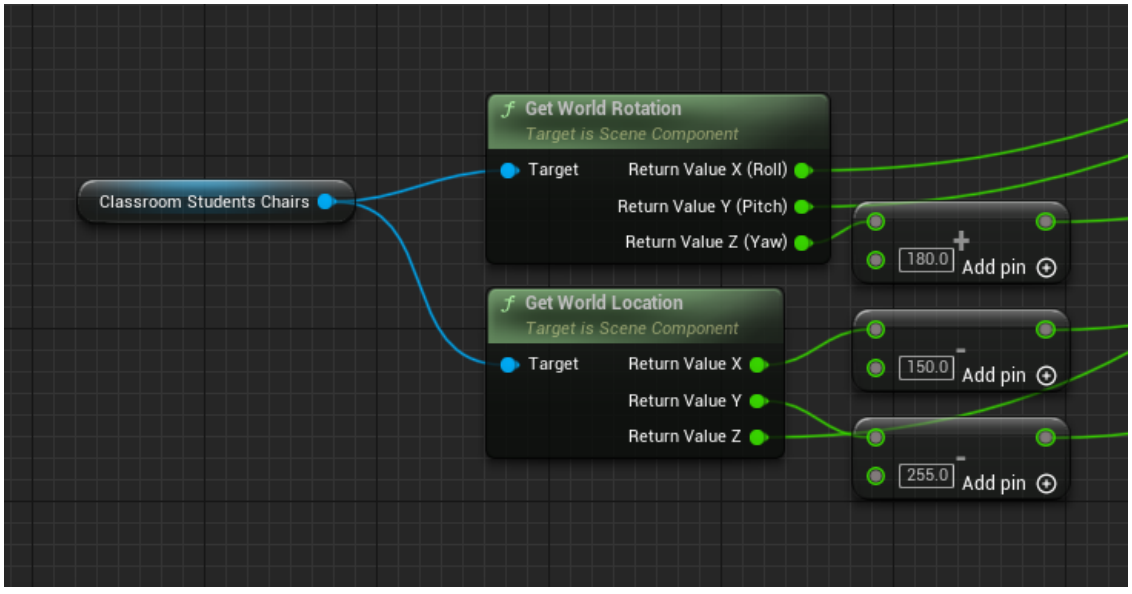


This also works quite similarly, we first add a box collision and then add in the chair mesh itself.



I'm setting the parameters of two Booleans for future reference here and setting the new transform properties for the character.
The input parameters come from a lot of behind-the-scenes math going on



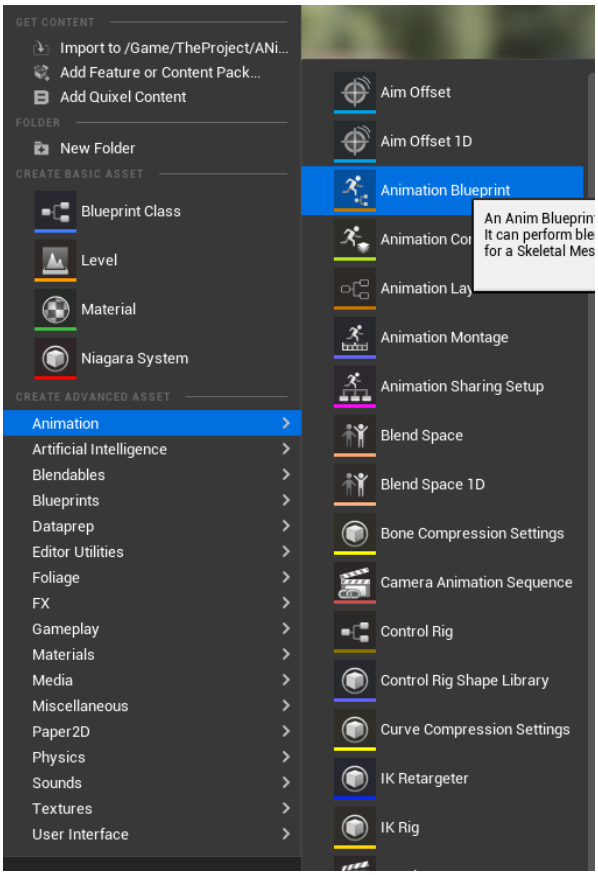


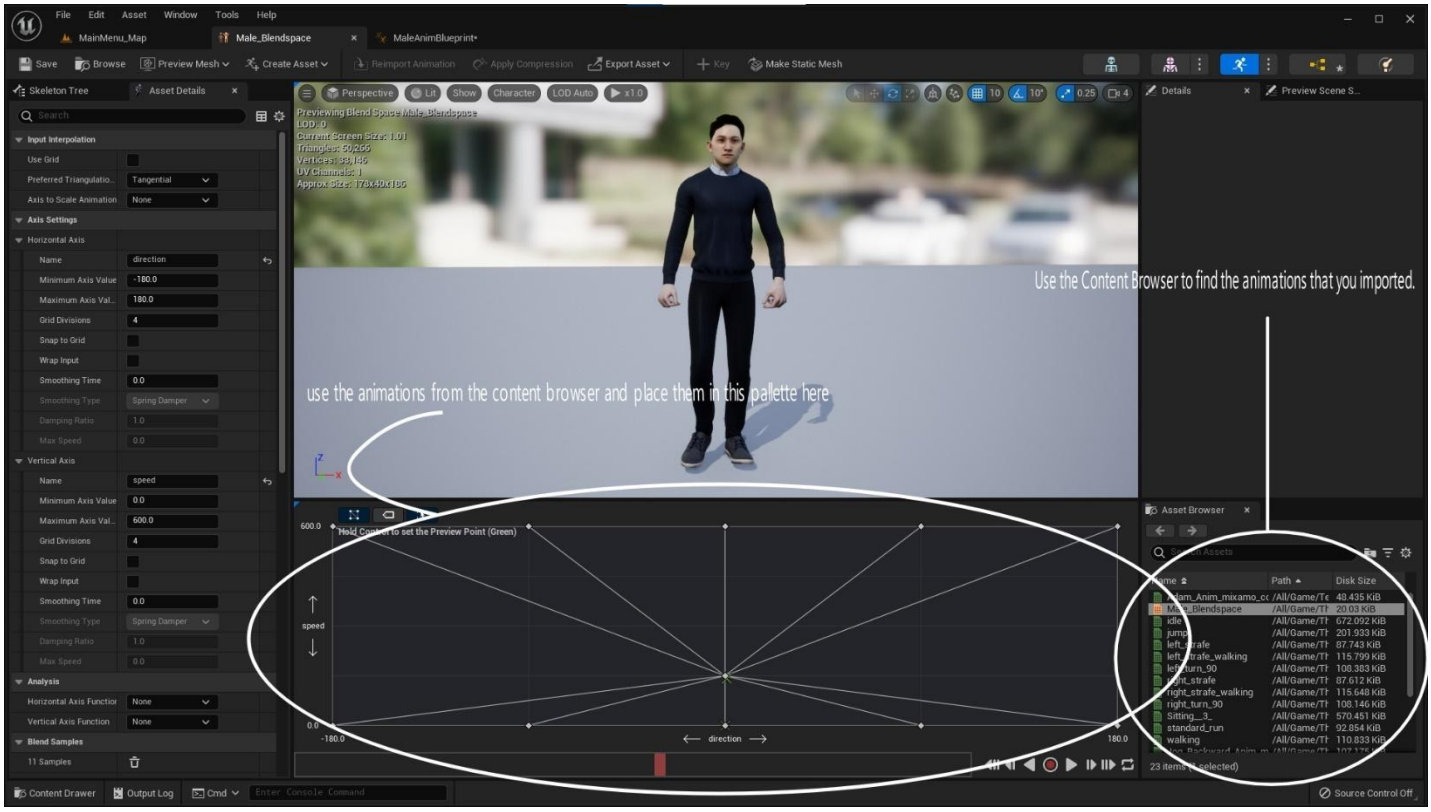
1.4.3 Creating the Character

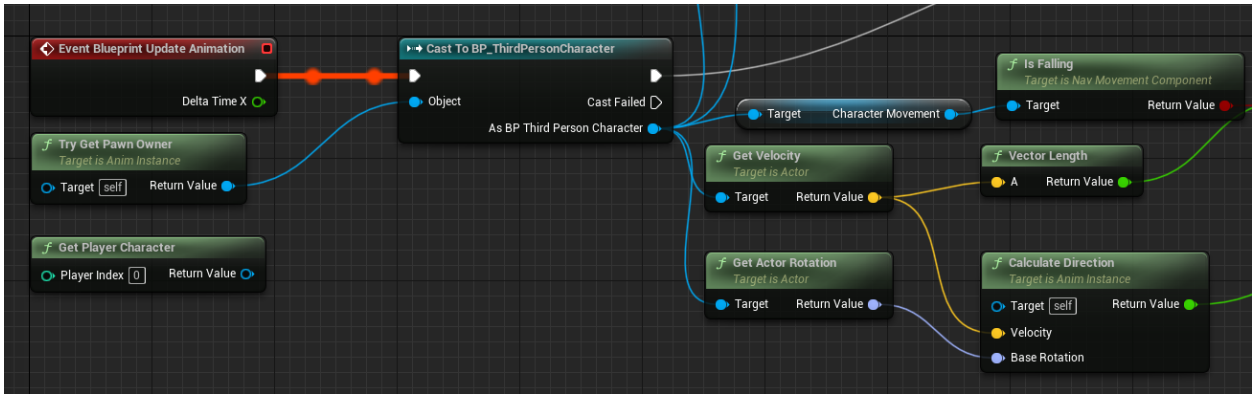
Right click in the content drawer and create an Animation Blend space and an Animation Blueprint.

Do not forget to setup the axis settings in the blend space and then we can setup the Character Blueprint.

And now you can setup the character blend space as shown below:

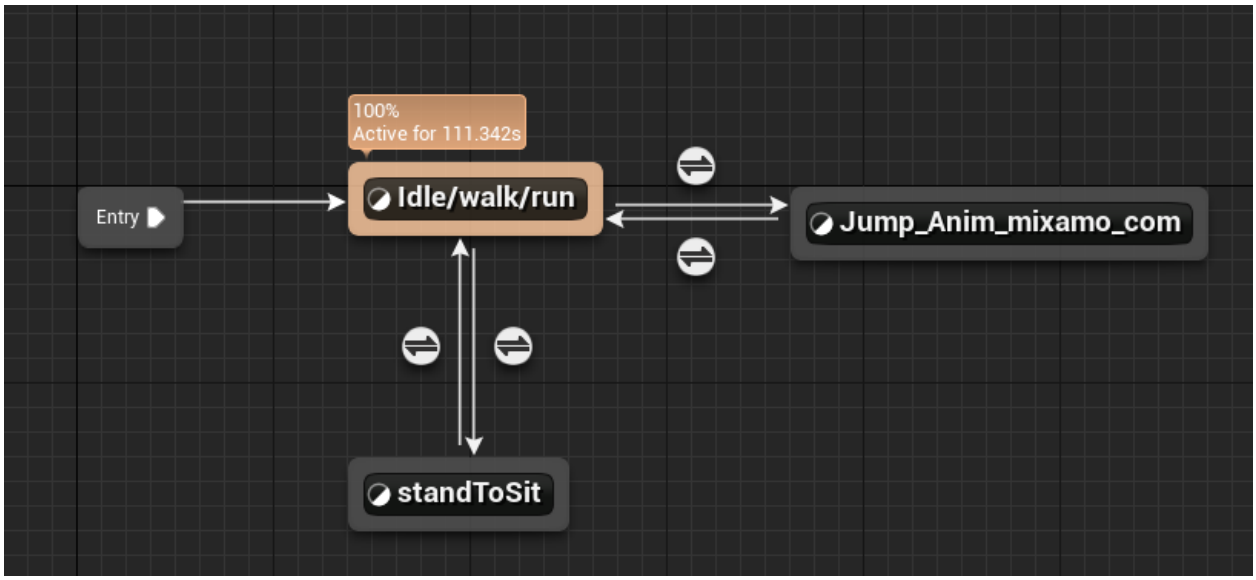






this is the minimum setup that needs to be done for the character to start moving around.

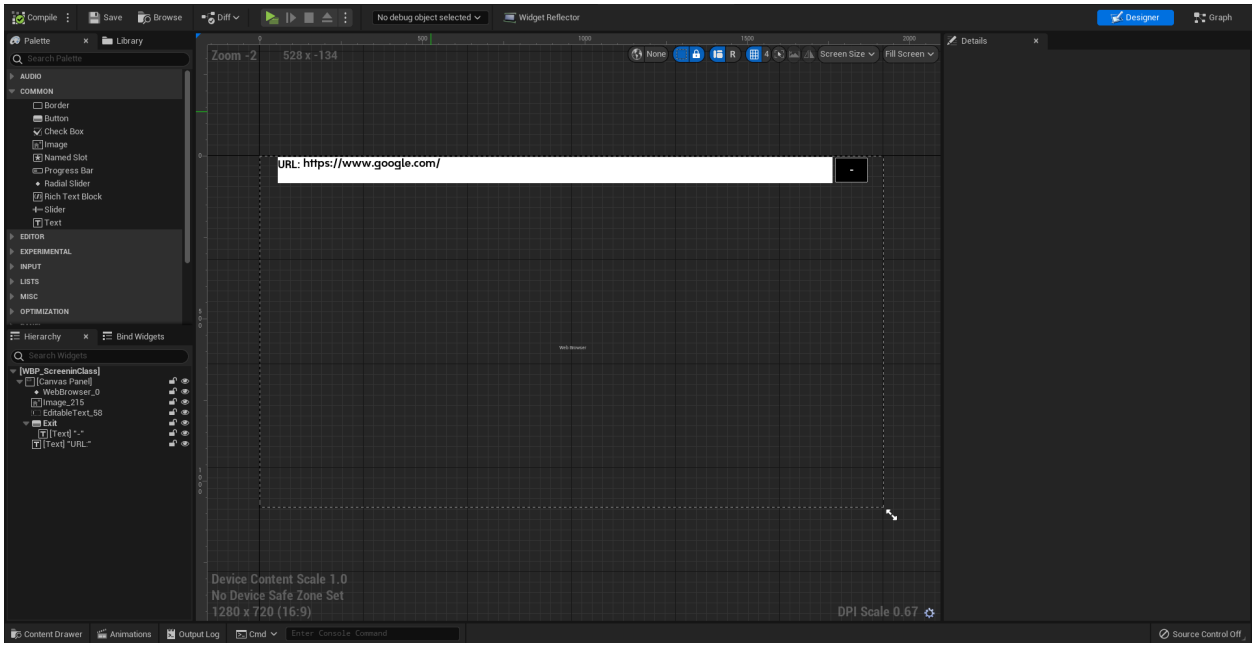
After this we need to setup the animation states as well. That can be done like shown below:



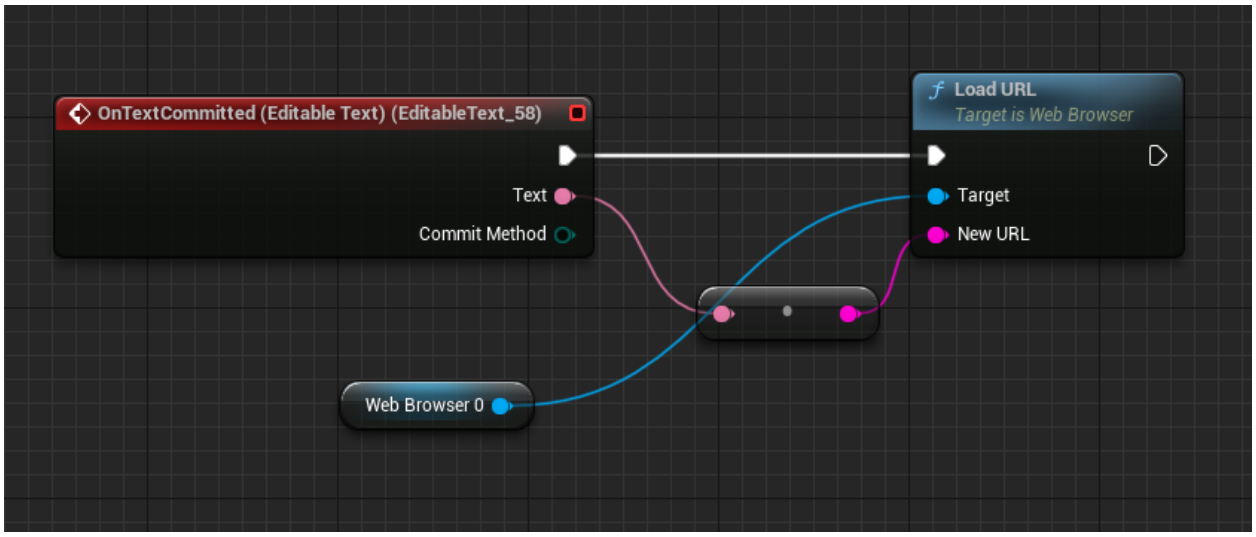
all the double outline boxes are states and the connections in between determine what state is to be applied. You would've noticed an extra state machine saying, 'stand to sit', this is the state machine responsible for the person to be able to sit on the chair. Setting this up is pretty simple; I created a Boolean called 'c is pressed' and only works if the person stands in front of a chair. So, its 2 Booleans one in the chair blueprint and one in the character blueprint. Both are true if the person is in front of the chair and the button is pressed. Then we car run a if (branch node in blueprints) condition to check if both are true and then trigger the sit animation

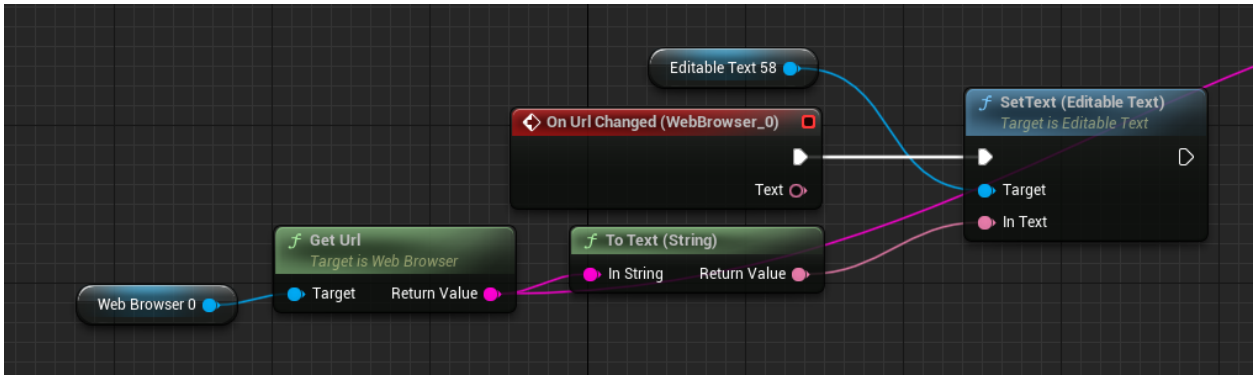
1.4.4 Creating the screen

Create a widget blueprint in the content drawer and make the widget to look as you please. The image down below shows how my screen looks. We then need to create an empty blueprint, create a widget object and link up our screen widget to it.



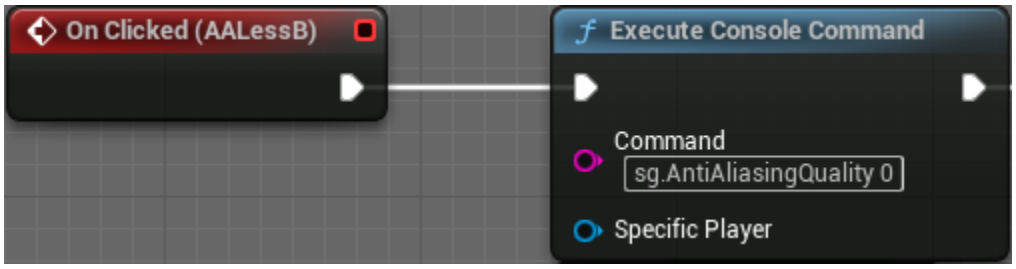
All we need to get this widget working would be to update the URL every time the input is confirmed. Fortunately, we have a node for that as well. If select the editable text box in the variables section, and click on text changed, we get a node that outputs the text every time we change it in the application. Then we send this text to a load URL node with the web browser as the target. If you want to update the text displayed in the URL box (editable text box), just use a on text changed node and pass the text to a set text node. Check the images below if needed.



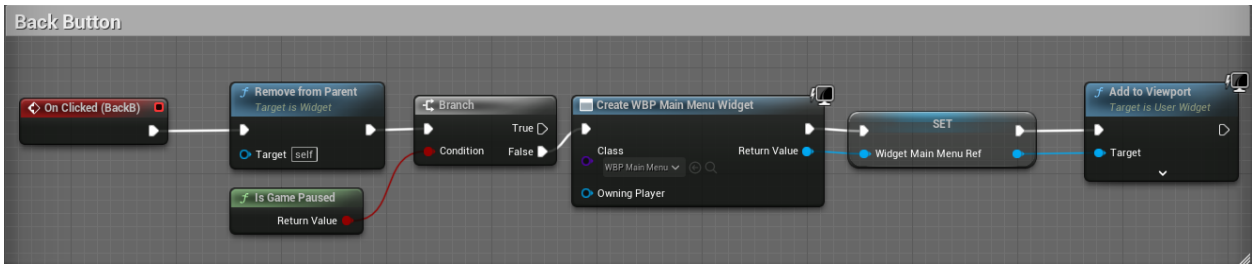


1.4.5 Creating the options/settings screen

Create another widget Blueprint and make an options screen by adding a few buttons and labelling them properly. To be able to access the screen resolution, shadow quality, anti-aliasing, and other features, we use the execute console command node and link them up to the on clicked event of the respective button. Never forget to add a back button or a resume button or the user might get stuck on the options screen. The node-tree would look like this:



And this can be used for the back button:



1.5 Setting up Photon

Setting up Photon was simple, since we followed along with the tutorial provided by the developer himself. The link to the tutorial playlist is *this*. Although we were not able to get it to work. We could not figure out what or even where the issue was. We worked on photon for nearly a week and still were not able to produce working results. We hope that whoever takes up the project next will at least be able to get photon working. LTI owns the license to Photon can be acquired by contacting Bhavesh Agarwal (our manager).

2. Networking and Backend of LNDVerse

2.1 Introduction

This part of the documentation shows how the backbone of the project has been built. We have gone through several options before choosing the ones that have been chosen now. The following explains how exacts we used Unreal Engine, PlayFab, Odin and Photon to develop this project.

2.2 BP_Gameinstance:

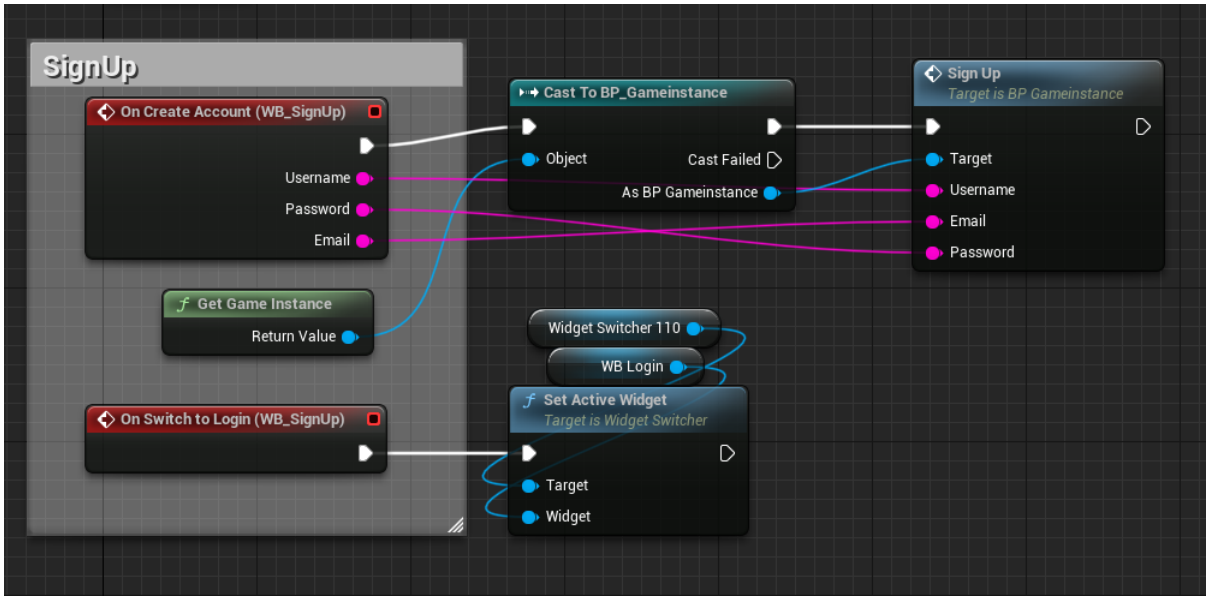
This is the Blueprint where most of the logic for this project resides. This is used because in Unreal engine this is a persistent object that can be accessed wherever and whenever you want. Throughout this documentation there will be several references to this blueprint.

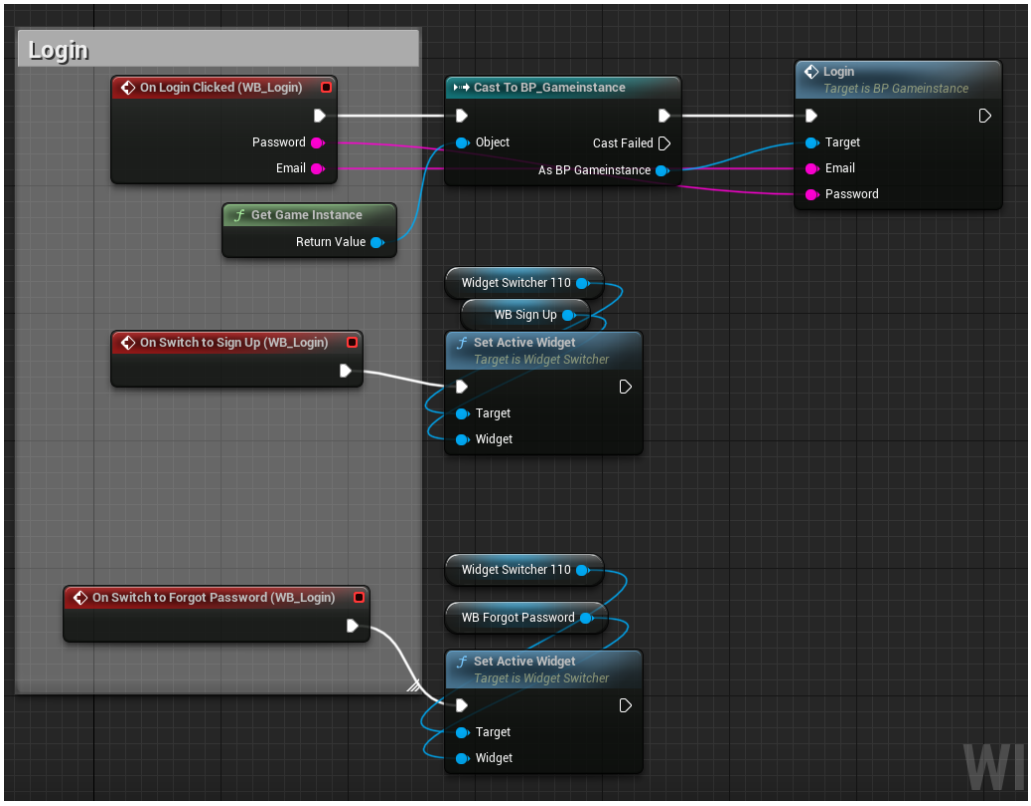
2.3 PlayFab Authentication:

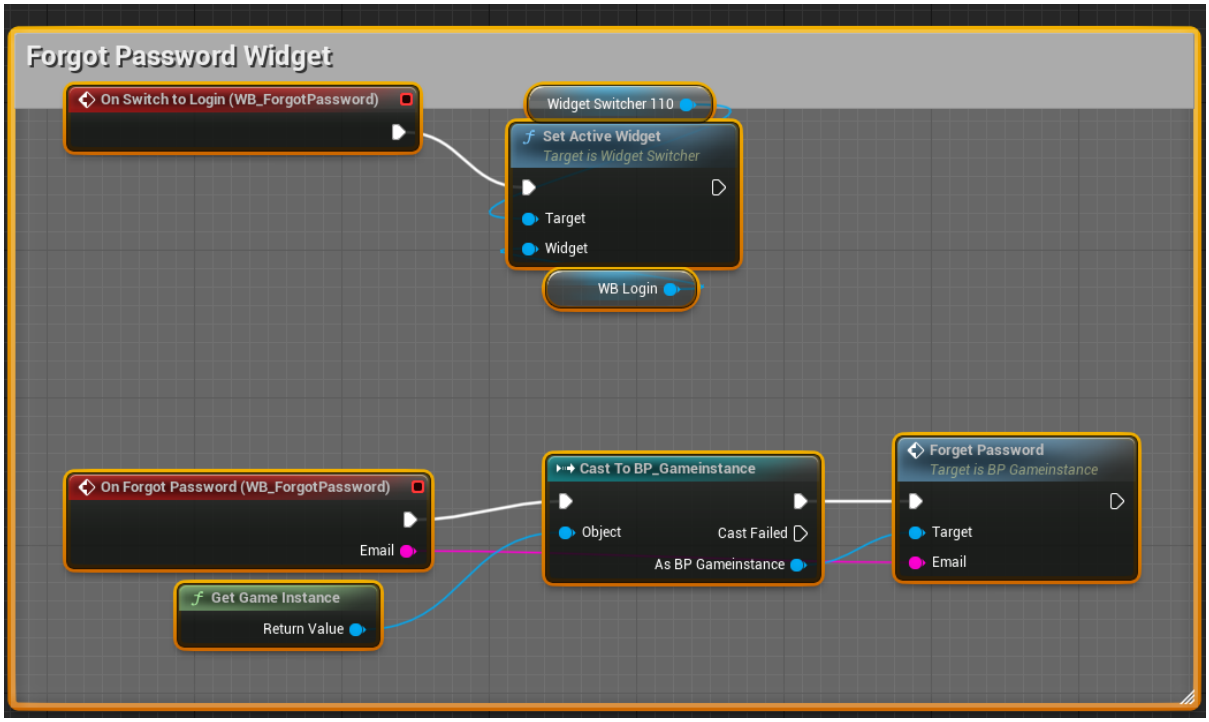
WB_PlayFabLogin is made of 4 widgets, namely **WB_SignUp**, **WB_Login**, **WB_ForgetPassword** and **WB_GenericInputField**.

We use a Widget Switcher(WidgetSwitcher110) to switch between the widgets for signup, login and forget password using the Set Active Widget Function.

WB_PlayFabLogin contains the following

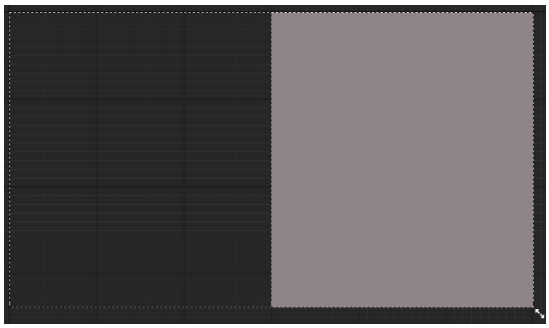

SignUp
On Create Account (WB_SignUp)
Username
Password
Email
Cast To BP_Gameinstance
Object
Cast Failed
As BP Gameinstance
Sign Up
Target is BP Gameinstance
Target
Username
Email
Password
Get Game Instance
Return Value
Widget Switcher 110
WB Login
On Switch to Login (WB_SignUp)
Set Active Widget
Target is Widget Switcher
Target
Widget


Login
On Login Clicked (WB_Login)
Password
Email
Cast To BP_Gameinstance
Object
Cast Failed
As BP Gameinstance
Login
Target is BP Gameinstance
Target
Email
Password
Get Game Instance
Return Value
Widget Switcher 110
WB Sign Up
On Switch to Sign Up (WB_Login)
Set Active Widget
Target is Widget Switcher
Target
Widget
Widget Switcher 110
On Switch to Forgot Password (WB_Login)
WB Forgot Password
Set Active Widget
Target is Widget Switcher
Target
Widget

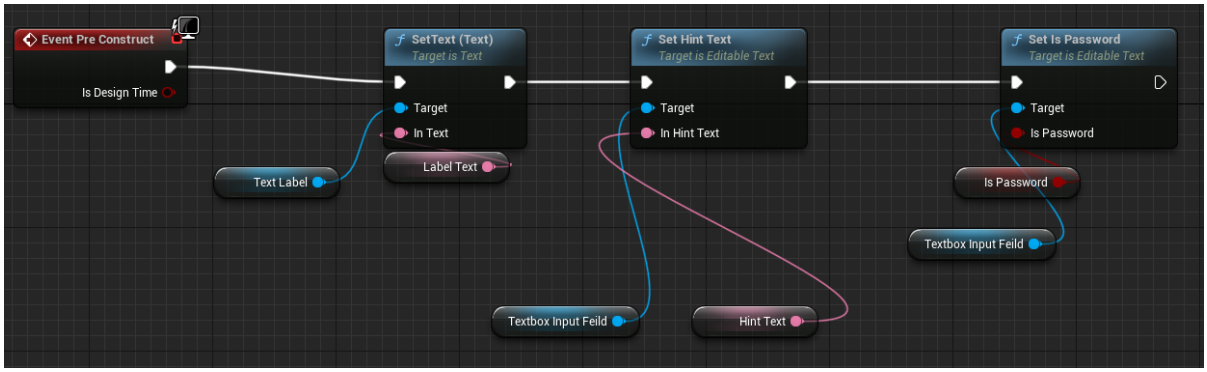


2.3.1 WB_GenericInputField:

Designer View:



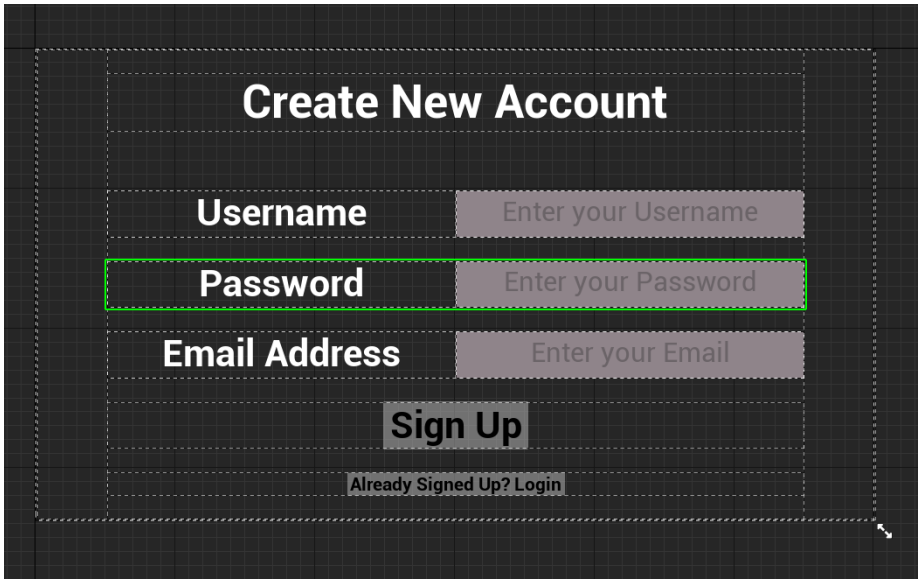
Graph View:

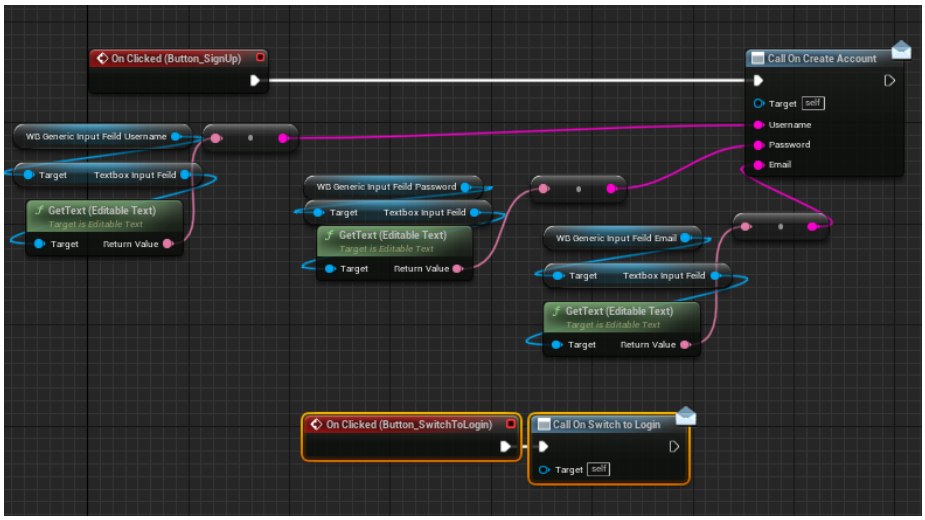


GenericInputField Widget has been used in a variety of ways. Basically, it contains a prompt on the left and text field and a hint of it on the right. There are 3 variables here Label Text, Hint Text and **isPassword**. Label text is used to store the prompt for the text that needs to be typed, so it can be stored in the **Text Label**, Text Widget.

HintText variable stores that hint for the text that needs to be typed with the target widget called the **TextBoxInputField** using the **SetHintText** function. **IsPassword** is a Boolean that contains whether the given text box stores a password or not so it can be used to mask the project.

2.3.2 WB_SignUp:

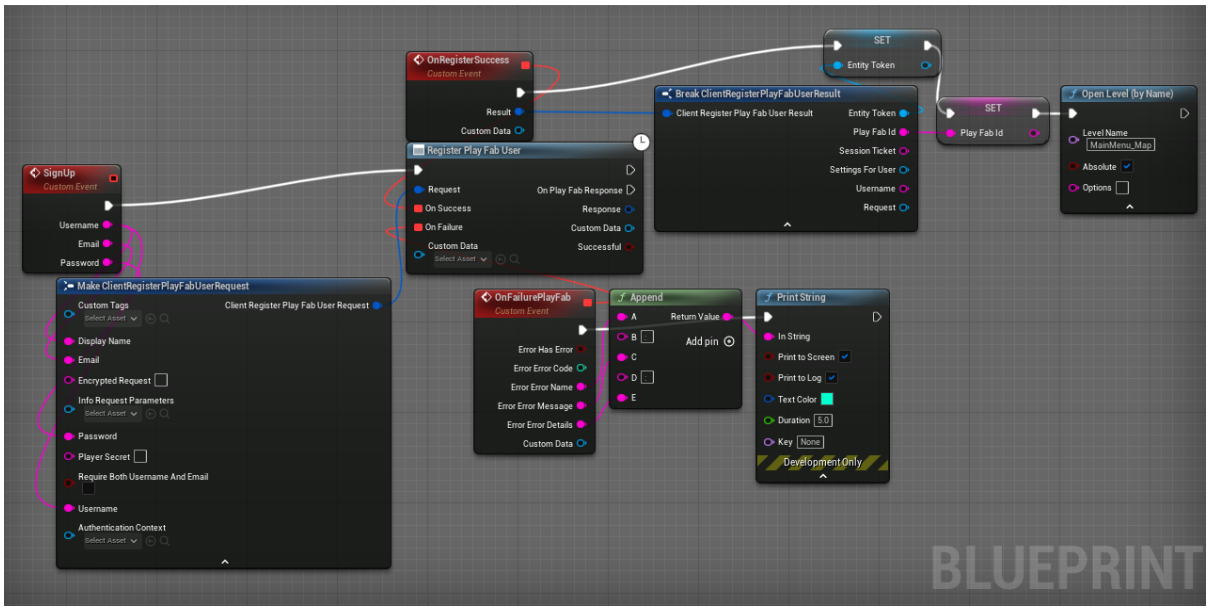




This widget contains 2 buttons, namely Button_SignUp and ButtonSwitchToLogin.
When ButtonSwitchtoLogin is triggered the CallOnSwitchToLogin event that references back to the WB_PlayFabLogin and sets the active widget to WB_Login.

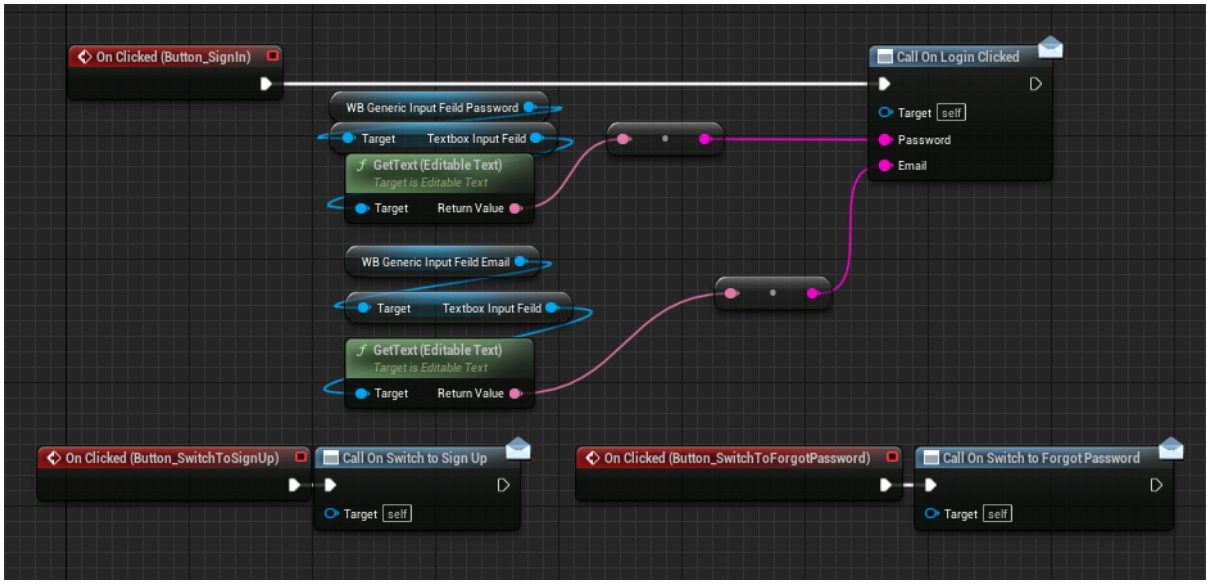
When the Button_SignUp is clicked it triggers the on CallOnCreateAccount function, which takes the Username, Password and the Email Address from the respective input fields.
CallOnCreateAccount references back to WB_PlayFabLogin where all his data is cast to BP_Gameinstance to the SignUp custom event.

This SignUp event triggers the PlayFab function called RegisterPlayFabUser, which takes the username, email address and password to pass it to the PlayFab cloud and creates an account. When this is Successful, the onRegisterSuccess event is triggered, creating an entity token and a PlayFab ID, before opening the MainMenu_Map which contains the MainMenu widget.

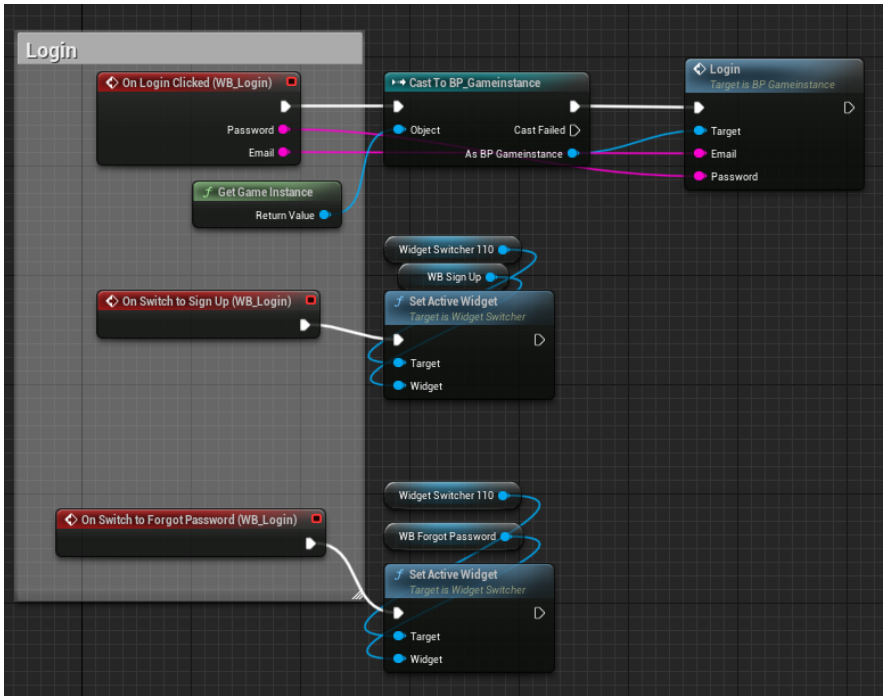

SignUp
Custom Event
Username
Email
Password
Make ClientRegisterPlayFabUserRequest
Custom Tags
Display Name
Email
Encrypted Request
Info Request Parameters
Password
Player Secret
Require Both Username And Email
Username
Authentication Context
Client Register Play Fab User Request
OnRegisterSuccess
Custom Event
Result
Custom Data
Register Play Fab User
Request
On Success
On Failure
Custom Data
On Play Fab Response
Response
Custom Data
Successful
OnFailurePlayFab
Custom Event
Error Has Error
Error Error Code
Error Error Name
Error Error Message
Error Error Details
Custom Data
Append
A
B
C
D
E
Return Value
Add pin
Print String
In String
Print to Screen
Print to Log
Text Color
Duration
Key
Development Only
Break ClientRegisterPlayFabUserResult
Client Register Play Fab User Result
Entity Token
Play Fab Id
Session Ticket
Settings For User
Username
Request
SET
Entity Token
SET
Play Fab Id
Open Level (by Name)
Level Name
MainMenu_Map
Absolute
Options
BLUEPRINT

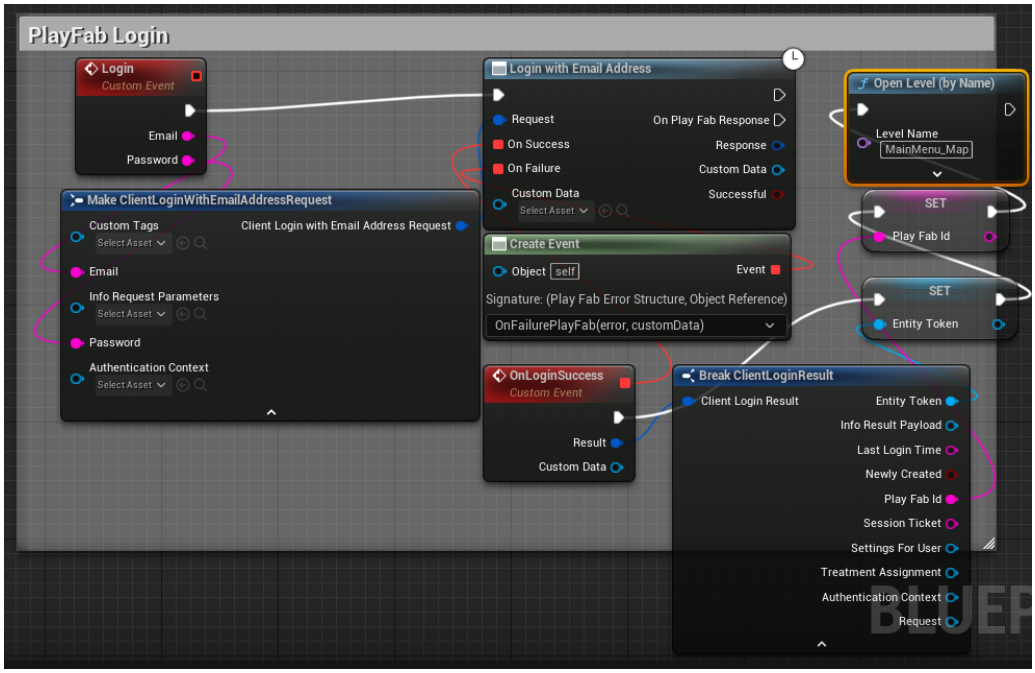
2.3.3 WB_Login:


Login
Email Address
Enter your Email
Password
Enter your Password
Sign In
New? Create an Account
Forgot Password?

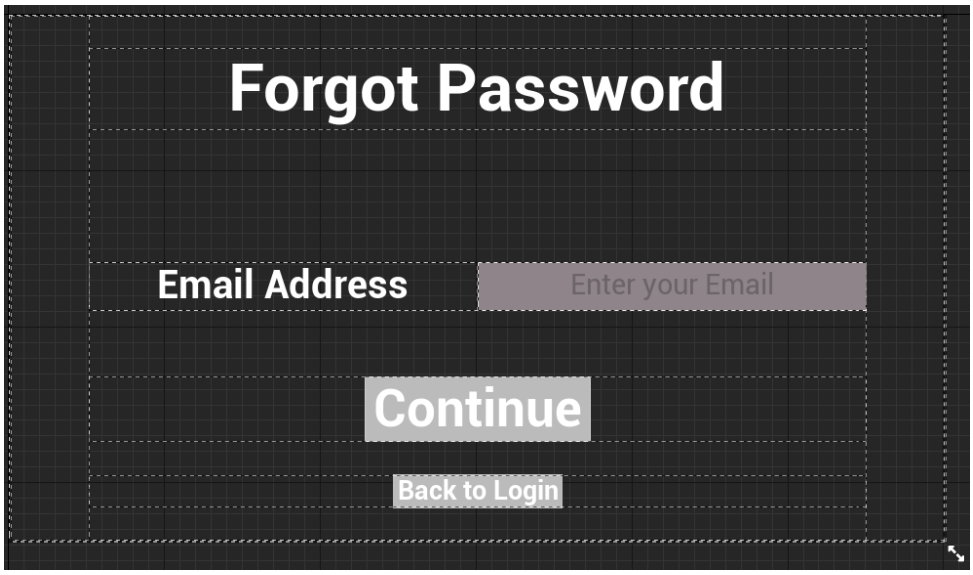


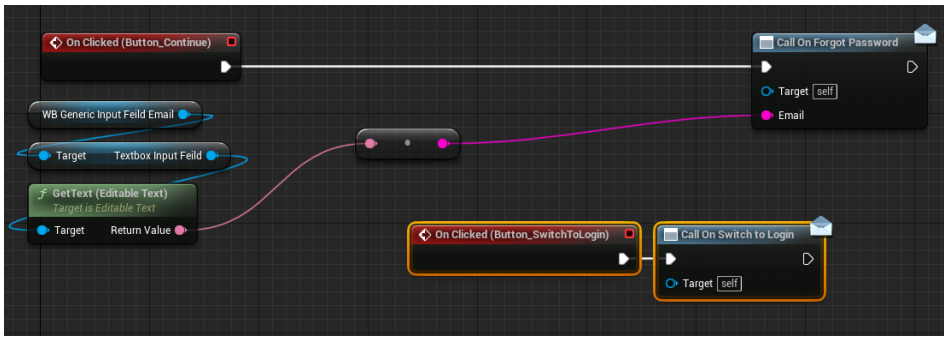
This widget has 3 buttons namely, Button_SignIn, Button_SwitchToSignUp, Button_SwitchToForgetPassword. When the username and password is filled and the Button_SignIn is pressed it triggers the CallOnLoginClicked event with intern calls the OnLoginClicked custom event that casts all these variables in the gameinstance and call the Login custom event in the Gameinstance. This Login event triggers a function from PlayFab called LoginwithEmailAddress which checks the credentials with PlayFabs servers and OnLoginSuccess lets you into the MainMenu_Map, else will trigger the OnPlayFabFailure event giving out the relevant error code and reason for the error.


Login
On Login Clicked (WB_Login)
Password
Email
Cast To BP_Gameinstance
Object
Cast Failed
As BP Gameinstance
Login
Target is BP Gameinstance
Target
Email
Password
Get Game Instance
Return Value
Widget Switcher 110
WB Sign Up
On Switch to Sign Up (WB_Login)
Set Active Widget
Target is Widget Switcher
Target
Widget
Widget Switcher 110
On Switch to Forgot Password (WB_Login)
WB Forgot Password
Set Active Widget
Target is Widget Switcher
Target
Widget


PlayFab Login
Login
Custom Event
Email
Password
Login with Email Address
Request
On Play Fab Response
On Success
Response
On Failure
Custom Data
Custom Data
Successful
Open Level (by Name)
Level Name
MainMenu_Map
Make ClientLoginWithEmailAddressRequest
Custom Tags
Client Login with Email Address Request
Email
Info Request Parameters
Password
Authentication Context
Create Event
Object
self
Event
Signature: (Play Fab Error Structure, Object Reference)
OnFailurePlayFab(error, customData)
SET
Play Fab Id
SET
Entity Token
OnLoginSuccess
Custom Event
Result
Custom Data
Break ClientLoginResult
Client Login Result
Entity Token
Info Result Payload
Last Login Time
Newly Created
Play Fab Id
Session Ticket
Settings For User
Treatment Assignment
Authentication Context
Request

2.3.4 WB_ForgetPassword

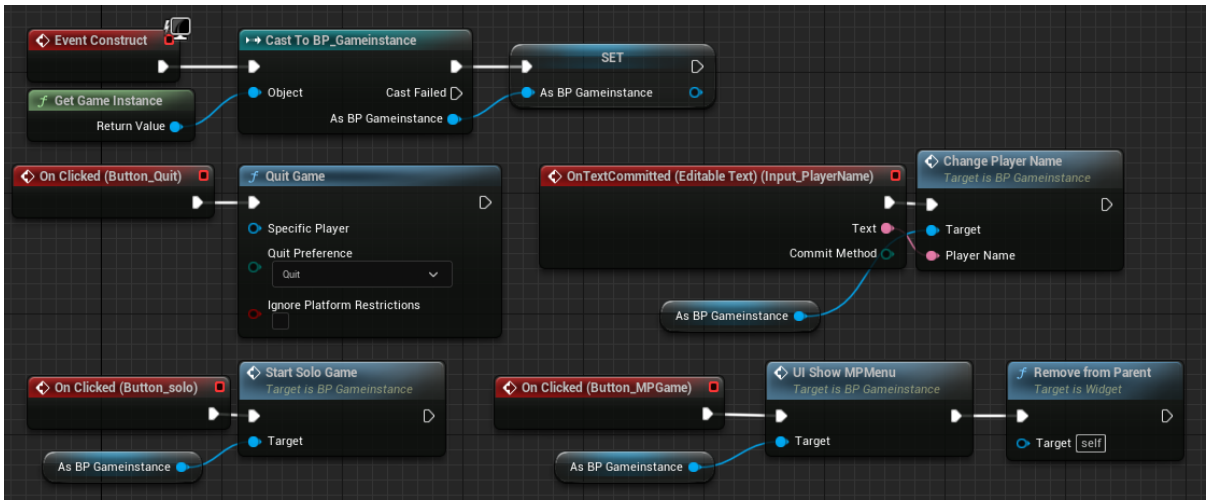




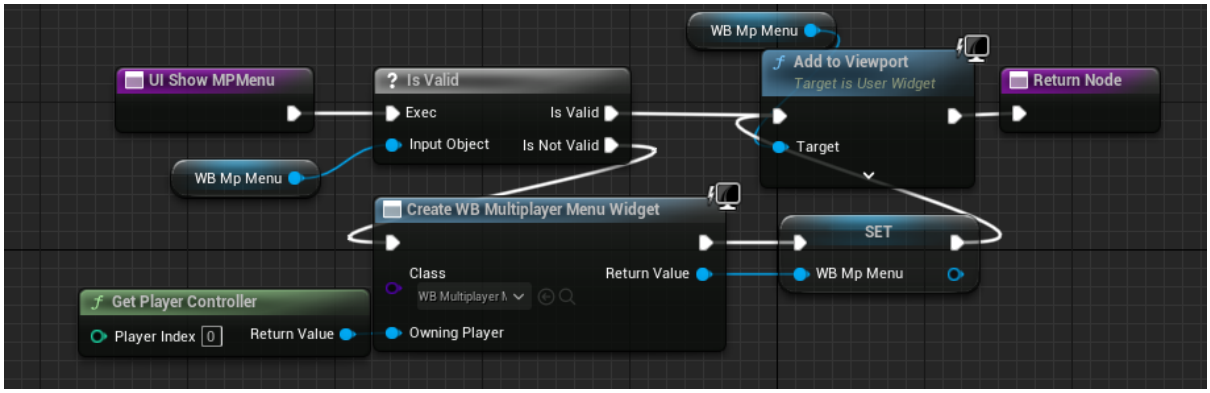
This widget contains 2 buttons and 1 text field. Button_Continue takes the email address of the user, and on clicked calls the CallOnForgotPassword function, and references the function in WB_PlayFabLogin, which intern casts the emails address to the GameInstance. ForgoetPassword custom event calls the send account recovery email function in the GameInstance and when succeeded, prints out that the recovery email has been sent. When this fails it prints out the respective error code, and message.

2.4 Base Game Logic

2.4.1 WB_MainMenu

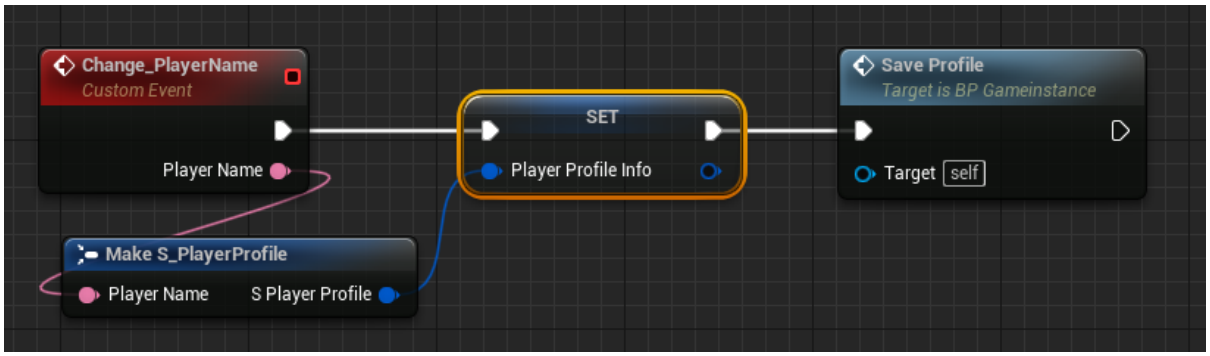


Main Menu contains 3 buttons and a text field. Button_Quit when clicked, helps you exit out of the game. Button_solo when clicked calls the Start_SoloGame method in the game instance, which intern calls the Open level by name function, with the Classroom level in it.

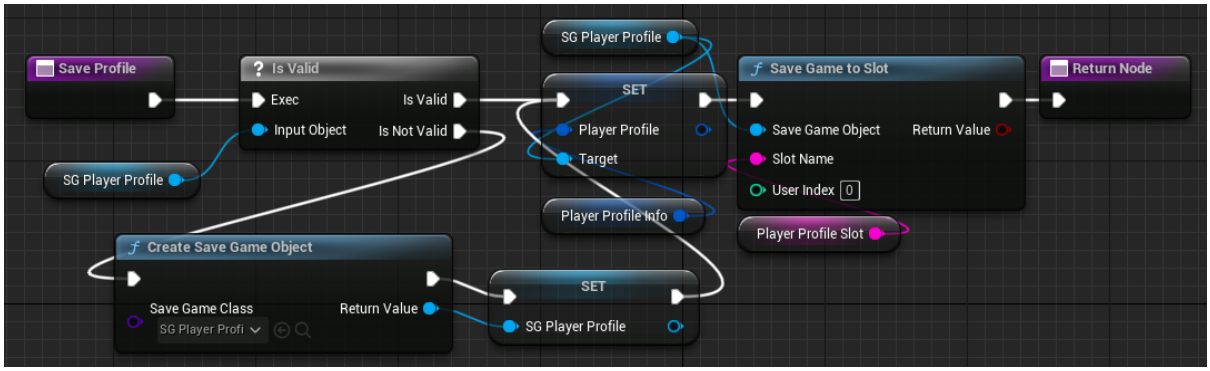


Button_MPGame when clicked calls the the UI Show MPMenu function in the the GameInstance. This function checks if the widget WB_MPMenu widget is valid and if yes, adds it to the viewport and returns it. If not, it creates the widget is not already used, adds the player controller to it and sets it as the WB_MPMenu before adding it to the viewport and returning. It also removes the

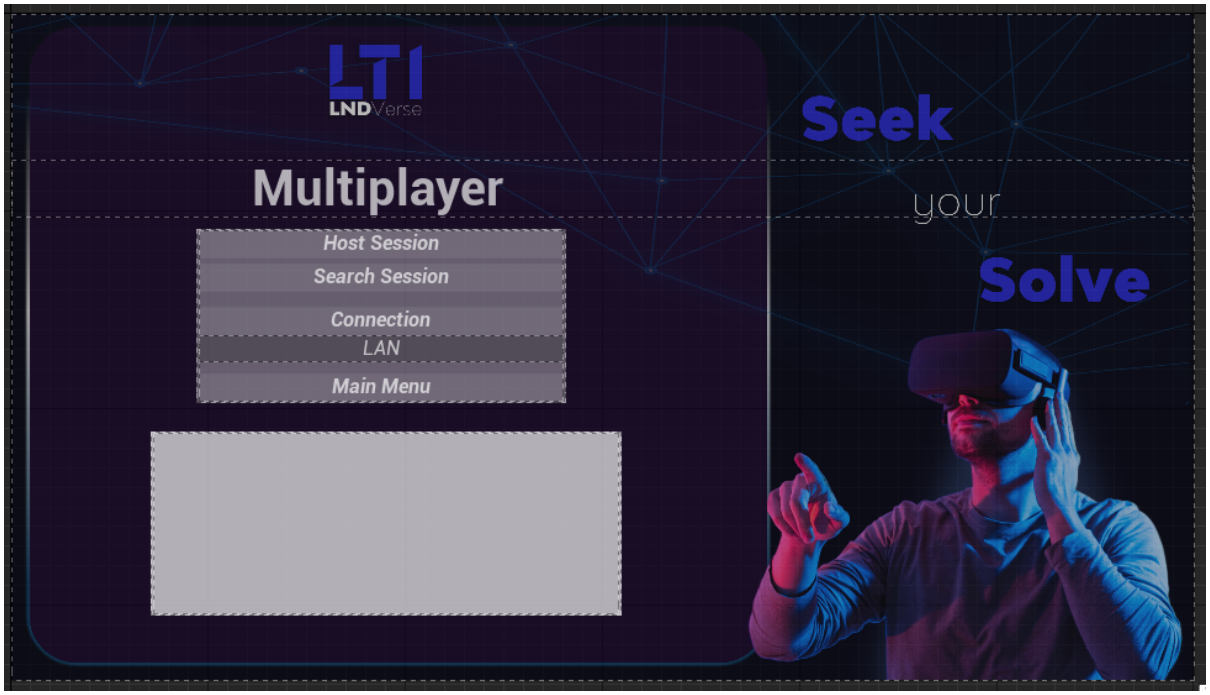
OnTextCommited is an event that is triggered whenever the user presses enter, or the text box loses focus. Then this event occurs, it calls the function Change Player name in the Game Instance. Which changes the player profile info in the cache and stores it and calls the save profile function in the Gameinstance.

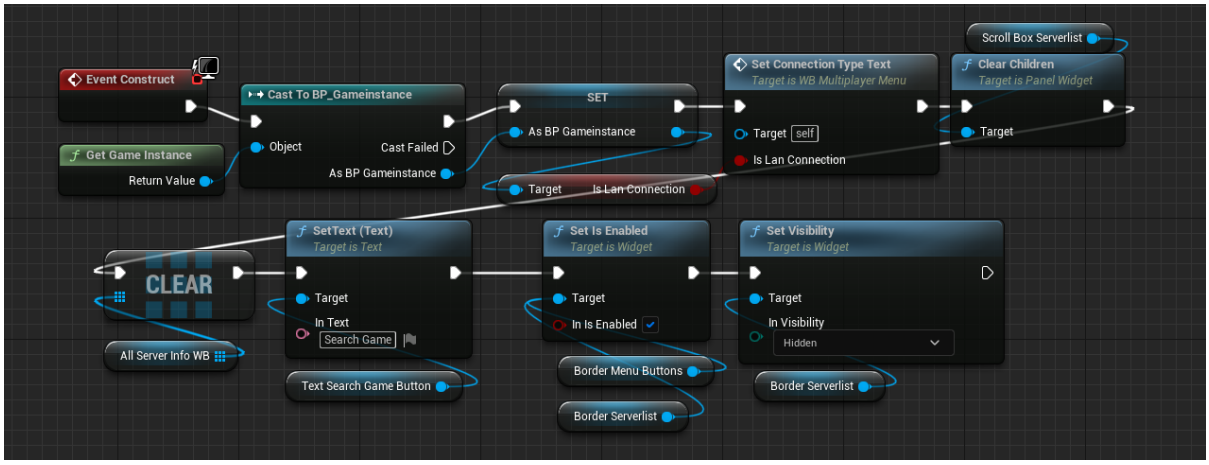


Save profile event checks if the profile is valid and if it is it sets the player profile to the new name entered and saves the name using the save game to slot function before returning. If there was never a profile stored before, it creates the save game object and sets it instead.



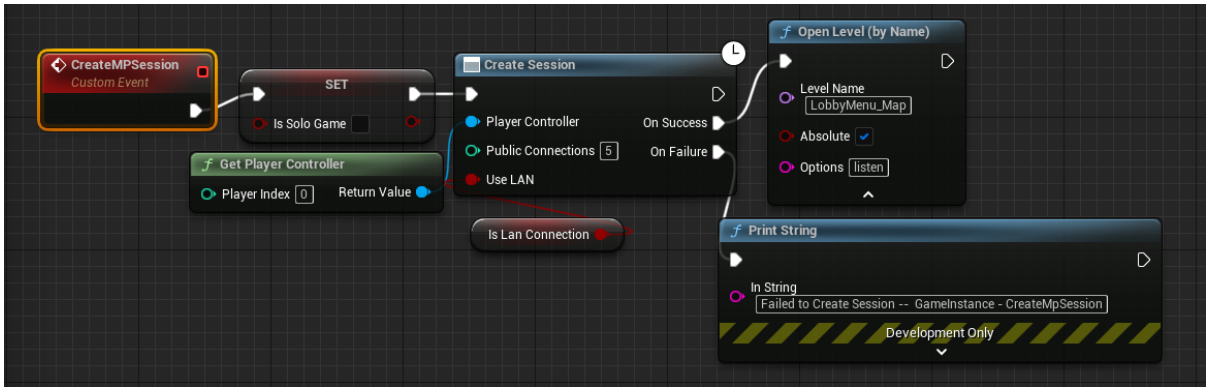
2.4.2 WB_MultiplayerMenu:





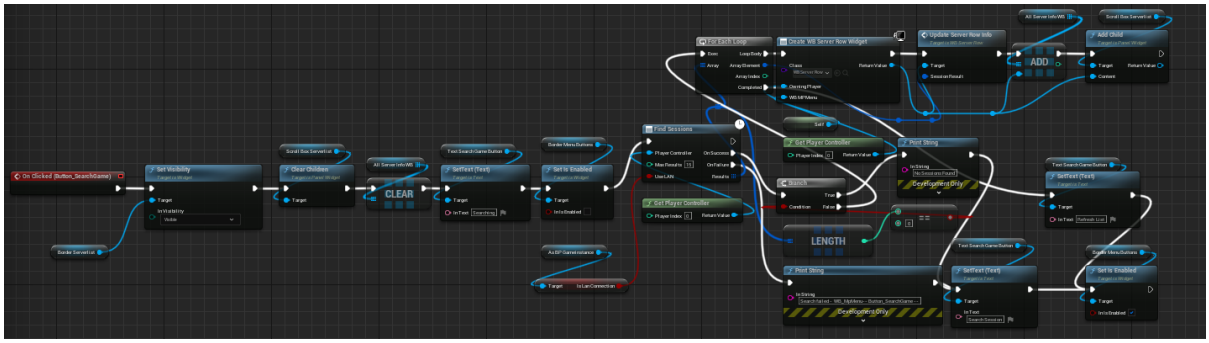
When this Level is opened from the WB_MainMenu, the first event that gets executed is the event construct. The first thing it does is initialize the GameInstance variable. Then it sets the connection type, by default it is set as LAN. It sets the clears all the servers from the server list and sets the buttons names to their defaults.

Button_HostGame calls the CreateMPSession Function in the Gameinstance, this Custom event in the GameInstance creates a session with a max number of connects as 5 and Opens the LobbyMenu_Map and prints the errors message if it fails to create a session.

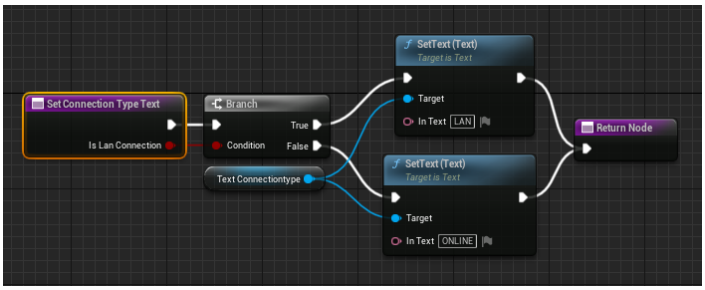


Button_SearchGame when called clears all the children in the Server lost and sets the search button to searching and disables the buttons till the search is complete. It then finds sessions available in the network, creates a Server row widget, updates the server info and ads the server to the list, with the controller. And if the search is a failure, it prints that it failed to find a session

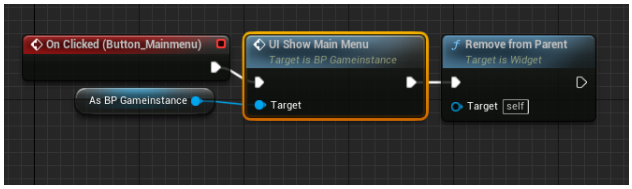
and set the buttons back to its original texts. It also enables the button after the search is complete.



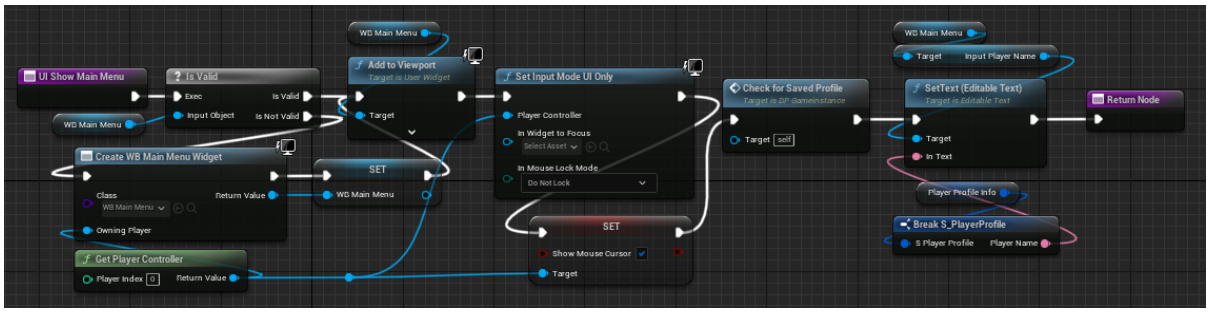
Button_Connectiontype when clicked calls the function SetConnectionTypeText, which helps you change between LAN and Online(Only used if you use the Epic Games online subsystem) and returns.



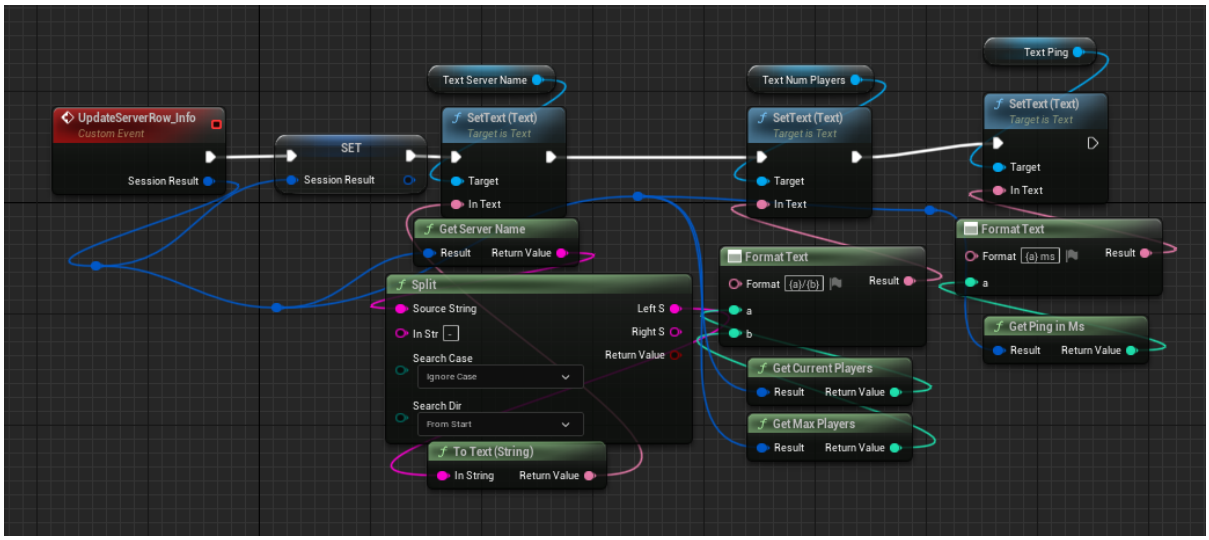
Button_MainMenu when clicked triggers a function called UI Show Main Menu



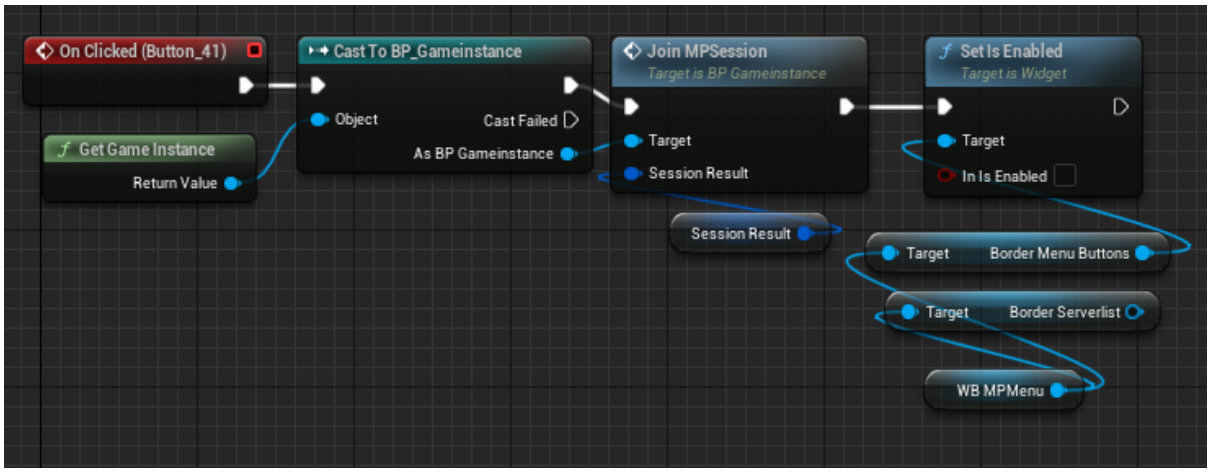
UI Show Main Menu event checks if the WB_MainNenu already exists and create if there isn't one already and adds it to the Viewport. Then Input mode is Set to UI only, giving you the cursor control. Then it checks for a save profile in the cache and loads it before going to the main menu.



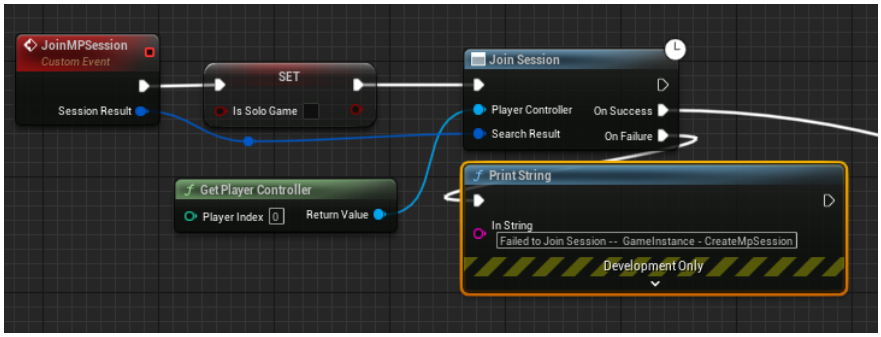
2.4.3 WB_ServerRow:



UpdateServerRow_Info function in the Multiplayer Menu triggers this event. This event takes Session result as its parameter, and adds the name, number of players in the server and ping from the session result to a Server row created in the Multiplayer Menu. This Widget mainly acts a button(Button_41).

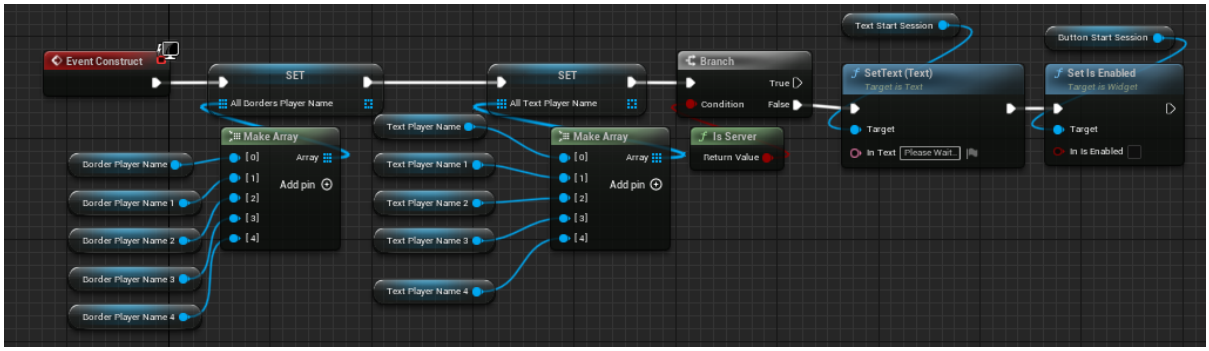


This when clicked calls the JoinMPSession function in the GameInstance, which helps the player controller to ether the session and join the Lobby.

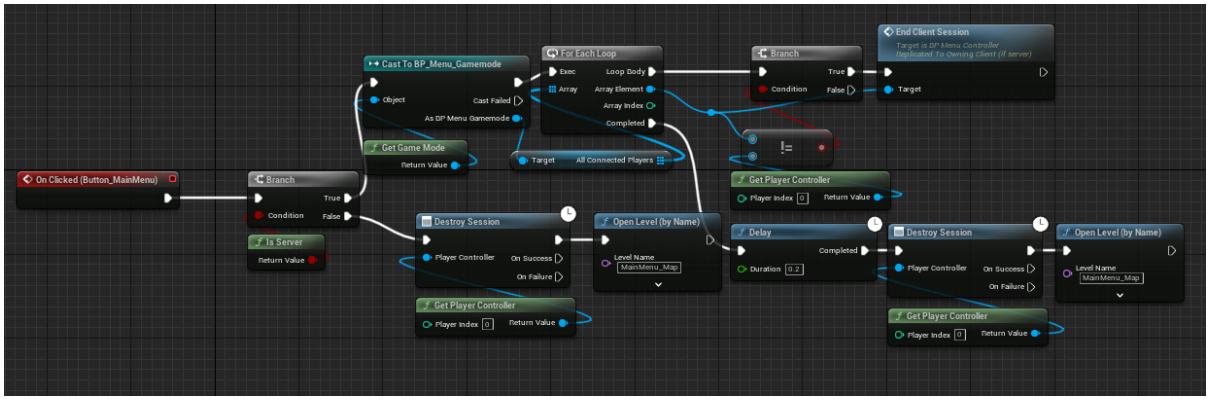


2.4.4 WB_LobbyMenu:

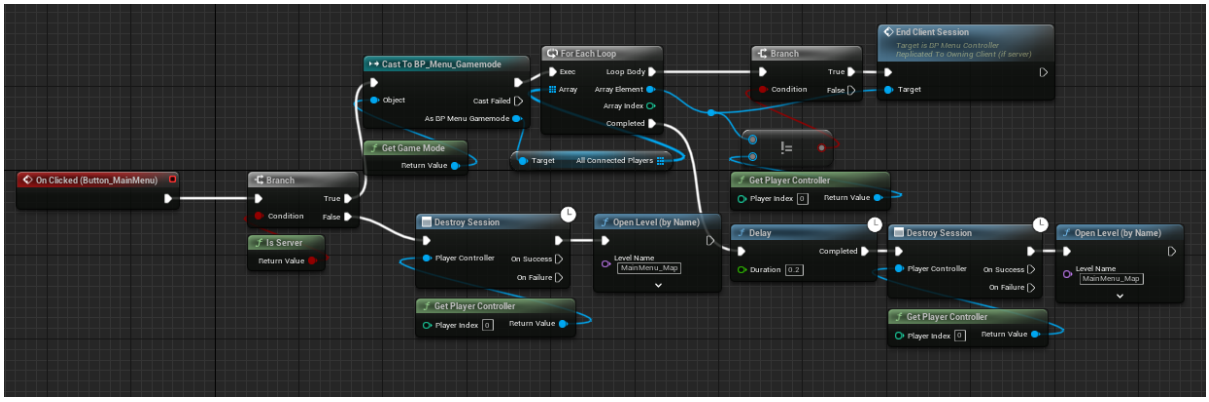
When you enter the Lobby menu, the first event that is triggered in the event construct, which takes the Player Names and adds them to the lobby player list.



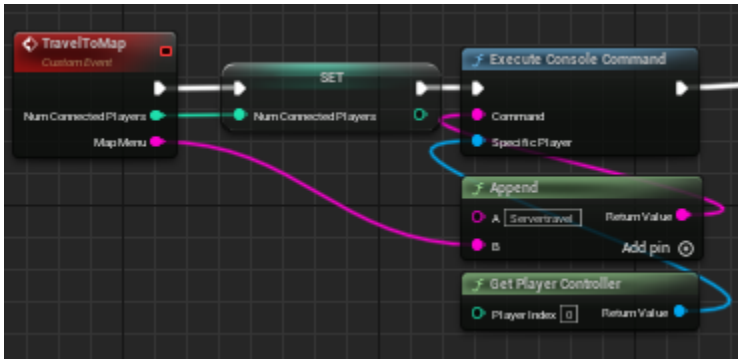
Button_MainMenu when clicked, if the controller is the server host disconnects all the clients that are connected to the session and returns them to the main menu. If the controller is a client, it destroys then session and opens the Main Menu.



Start session button when clicked, gets all the players from the lobby and makes them trigger the function TravelToMap in the GameInstance.



Travel to map event takes all the connected players and executes a console command called server travel on the player controller.

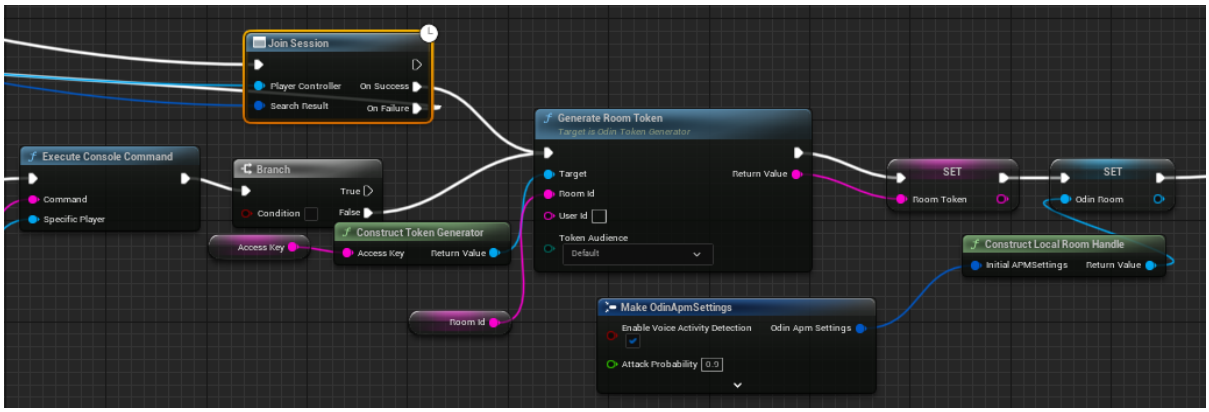


2.5 Odin 4Players:

This is a product designed to help apps games and websites have a voice chat function. We have chosen this plugin, because the support seemed really good and looked like a simple add on to the product that we already made.

This is free plugin on UEMarketplace and all you need to integrate this to your project is an access key that you are provided in the documentation. For the purpose of tests, there is also a link where u can type your access key and Room name to enter the voice chat. If anyone taking this project wants to move forward with this project, please make sure to contact them and ask if the documentation is updated, because at the time of making this project we had to contact the developers to take care of some issues, with several unresolved issues left as the documentation was not updated on their website.

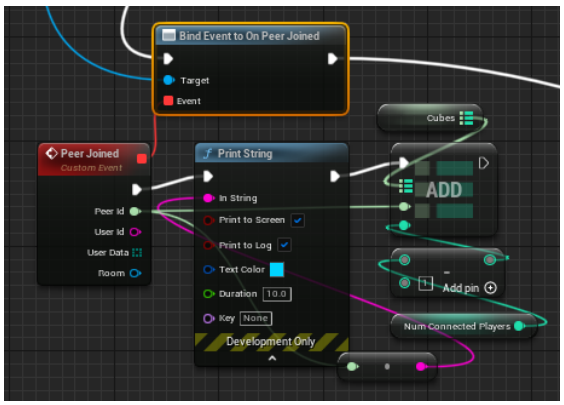
First thing any player, i.e the Server owner or the client who's joining has to generate a room token, using a room id and the access token, save the resulting tokens in the variable as shown.



There are the following events, that are required to be captured.

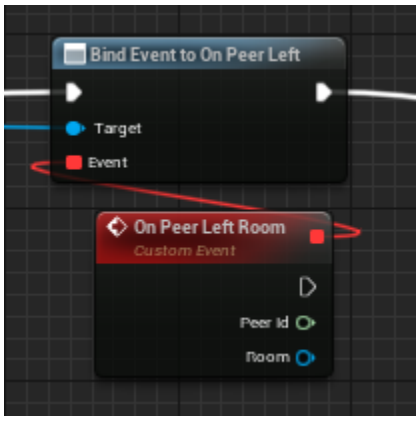
2.5.1 Bind Event on Peer Joined:

It takes the Odin Room token as an parameter and the event gives out peer ids which we store in map (Interger64, Interger) peerID to Player number.



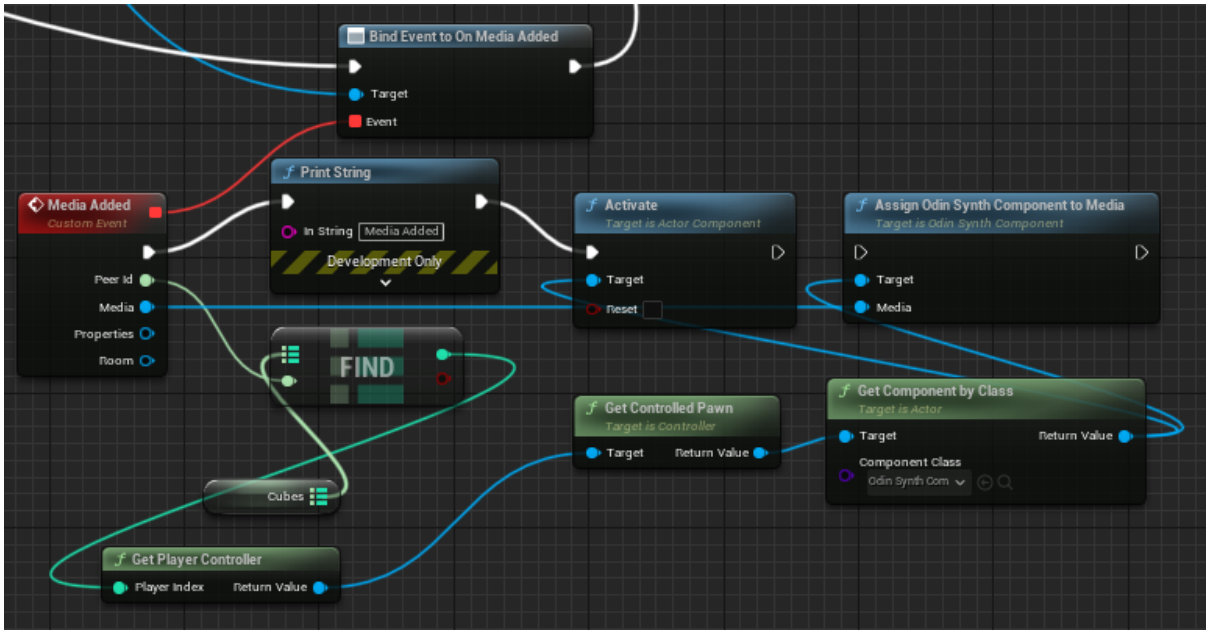
2.5.2 Bind Event on Peer Left:

This event is triggered when a peer leaves the room, this is irrelevant to us as a user does have any means to join the server after he leaves. It takes the Odin Room Token as the input



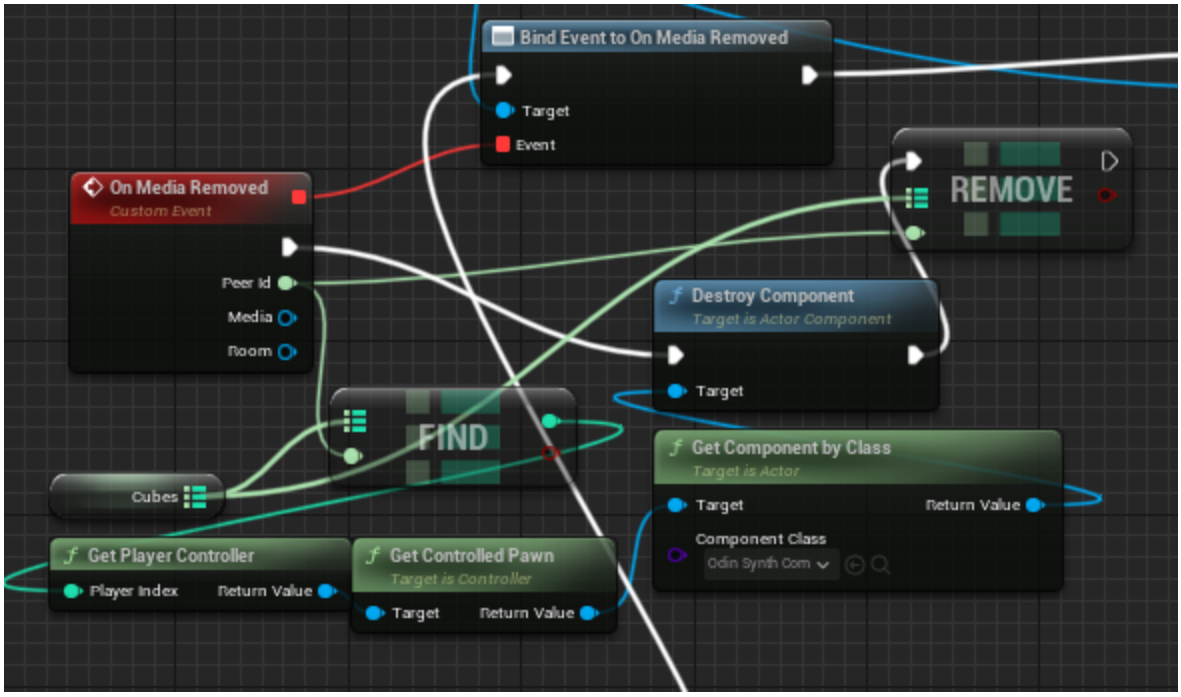
2.5.3 Bind Event on Media Added:

This event is triggered when media is added to the room. This called the activate function which activates the Odin Synth Component that is spawned with the player character, using the peer id. Odin Synth is basically a speaker that is attached to the character which sends out spatial audio to the server so players in a particular radius are able to listen when you talk.



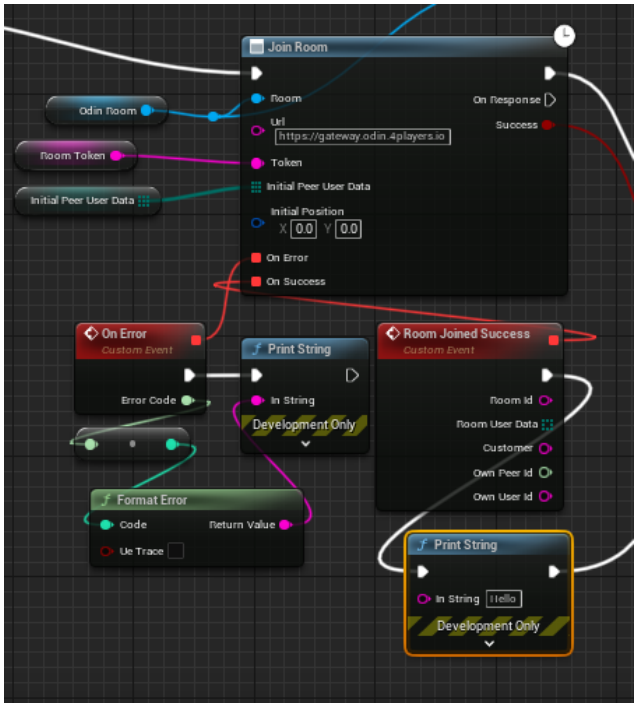
2.5.4 Bind Event on Media Removed:

This event is triggered when a peer leaves or when one of the mics stop recording audio. This event is used to destroy the components attached to the players.

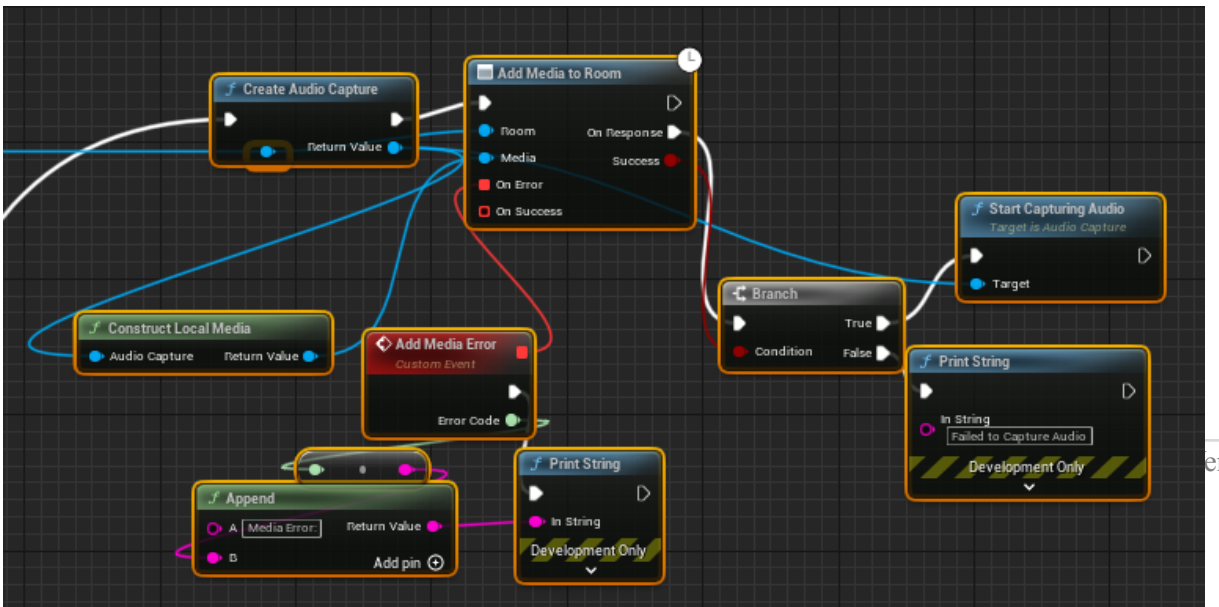

Bind Event to On Media Removed
Target
Event
On Media Removed
Custom Event
Peer Id
Media
Room
REMOVE
Destroy Component
Target is Actor Component
Target
FIND
Cubes
Get Component by Class
Target is Actor
Target
Return Value
Component Class
Odin Synth Com
Get Player Controller
Player Index
Return Value
Get Controlled Pawn
Target is Controller
Target
Return Value

2.5.5 Join Room:

This event is triggered when the server owner or the client joins the room. It takes the room token and the Odin room variable shown in the blueprints as the parameters.



On error, we print the error occurred. And on success we start capturing the audio. We call the create audio capture function and add this media to the room, and if there is an error, the error message is printed. If not the StartCapturingAudio function is triggered, thus capturing the audio from the device's mic.



3. Resources

You can download the necessary models from the sites mentioned below and many other free resources available on the internet. The only thing that needs to be made sure of is the file format, all the downloaded flies must be in **'.fbx'** or **'.obj'** formats. If you cannot find those formats, you will have to download them and export them as the formats mentioned before

- Epic games store
- Blender.org
- Sketchfab
- Cgtrader
- Bimobject.org
- Quixel.com

4. Contact the Authors

You can contact us with the following emails

- UTTEJ KURUMA: uttej09092001@gmail.com
- SHISHIR M VARMA: mshishirvarma100@gmail.com