Multiplayer Projectile Physics Plugin (MPPP)
Formerly RMPP

readme version 1.3

Customers who purchased the 4.14-4.20 project, get the plugin.

Contents

1 Usage **2**
- 1.1 Projectile Settings . . . 4
- 1.2 Customize material toughness, hit sounds, hit particle fx . . . 5
- 1.3 Plugin Structure . . . 5

2 How it Works **6**
- 2.1 Basic Multiplayer . . . 6
- 2.2 Physics . . . 7

3 The Project Won't Build! **8**

4 Enabling Apex Destruction **8**

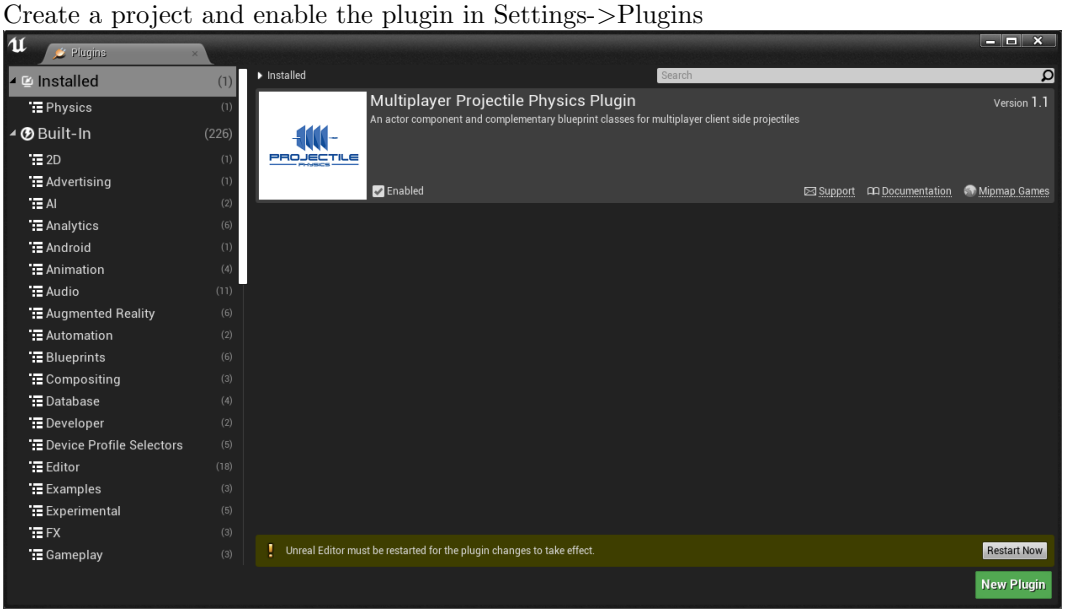
5 Custom graphics for prediction and tracers **9**

6 Lag Compensated Multiplayer Setup **10**
- 6.1 How to Implement Lag Compensation . . . 10
- 6.2 Integrating Lag Compensation into the First Person Template . . . 11
  - 6.2.1 Turn on the plugin . . . 11
  - 6.2.2 Give the player a 3rd person hitbox . . . 12
  - 6.2.3 Make a Projectile Actor Class . . . 14
  - 6.2.4 Set Up Networked Firing Code . . . 16
  - 6.2.5 Record the Player Hitboxes . . . 21
  - 6.2.6 React to Overlap Events . . . 22
  - 6.2.7 Implement Fast Forward . . . 23

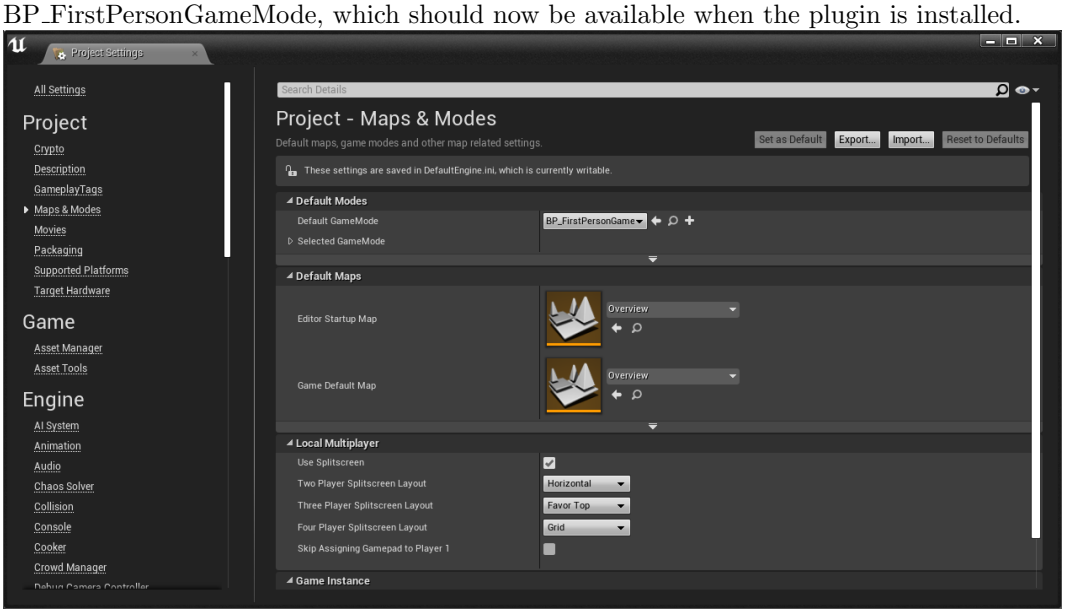
7 Hints and Tips **27**

1 Usage

1. Create a project and enable the plugin in Settings->Plugins



2. For a quick demo, in Settings->ProjectSettings->Maps&Modes set the default gamemode to BP_FirstPersonGameMode, which should now be available when the plugin is installed.



3. Hit play and you can demo the different projectiles included

Find the blueprints by clicking 'View Options' in the content browser and selecting 'Show Engine Content' and 'Show Plugin Content'. **The plugin is under RMPP** - the old name.

To fire a bullet you spawn a BP_MP_Projectile_Base actor with an initial direction. Initial speed is a setting in the blueprint. If you open the blueprint and click the RealisticProjectileComponent, you can play with all the settings. At the bottom of these properties are the events you can use to trigger your own code. These are

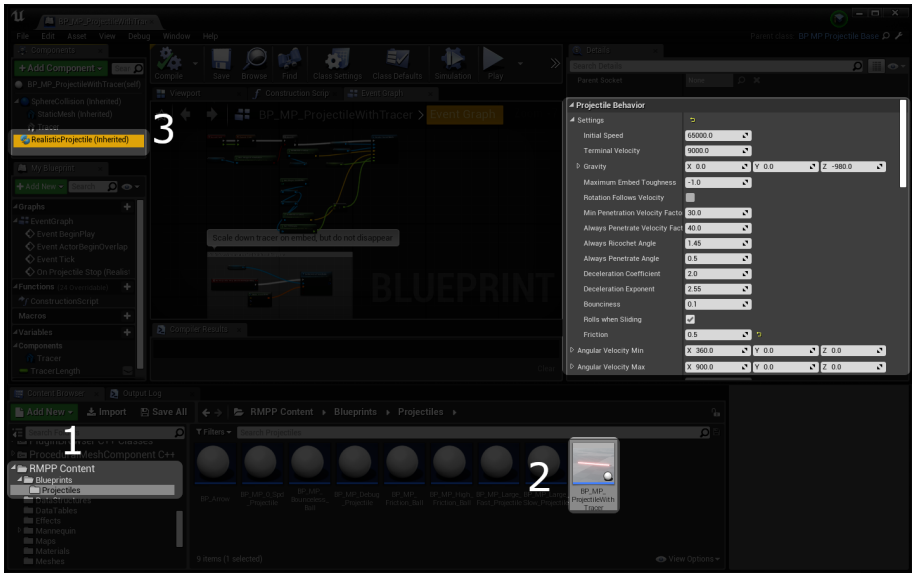
On Projectile Any Hit

On Projectile Bounce
On Projectile Penetrate
On Penetration Exit
On Projectile Stop
On Projectile Embed
On Component Begin Overlap

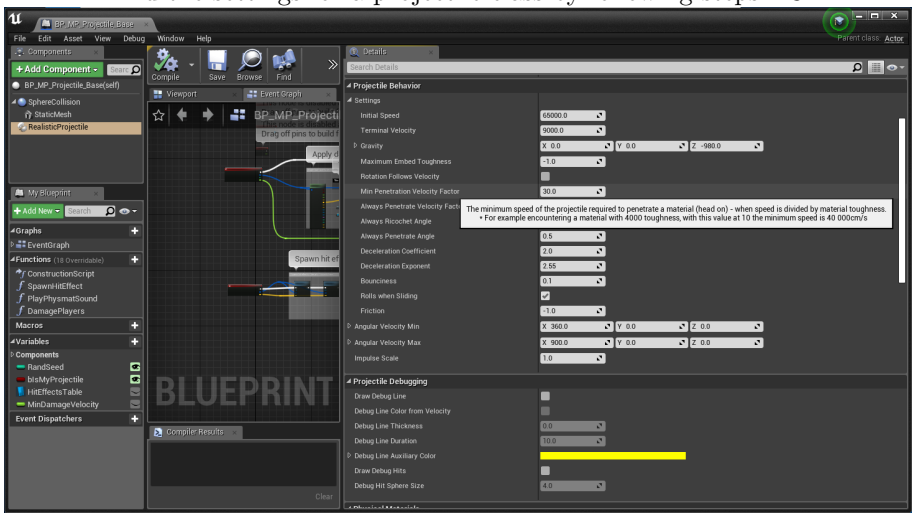
Making child classes from BP_MP_Projectile_Base is the easiest way to use the plugin, but you may wish to create your own projectile actor. Just add a RealisticProjectile component and it will move the RootComponent around and use its collision settings.

Marketplace plugins cannot contain custom collision channels or input bindings, you should make some. I recommend custom collision channels:
Projectile
CharacterMesh
Read how collision works for the plugin in the Physics section.

1.1 Projectile Settings



Find the settings for a projectile class by following steps 1-3.



All the available settings for a projectile class. Hover in the editor to see descriptions.

1.2 Customize material toughness, hit sounds, hit particle fx

Both of these are handled with datatables in RMPP\DataTables\.

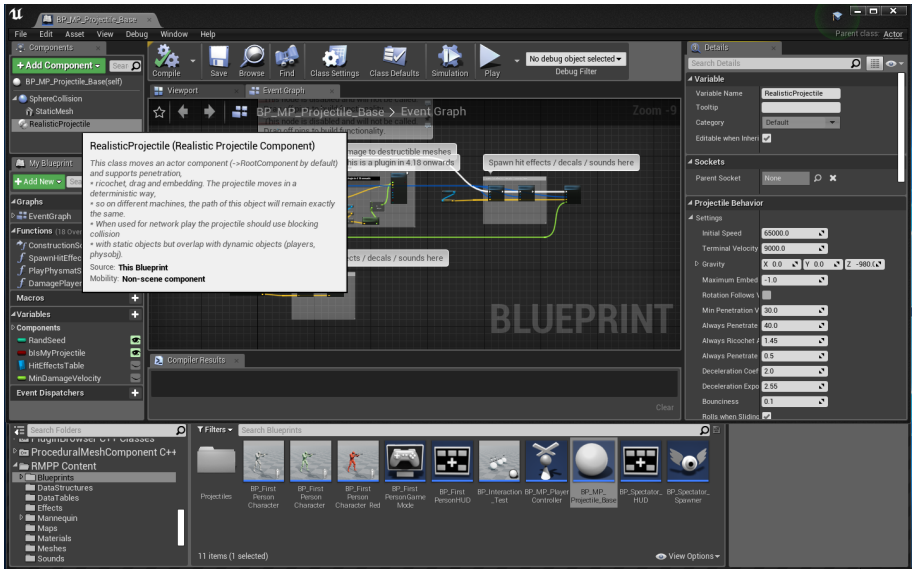
Material toughness is kept in a data table which maps the name of a physical material to a float. The row name must match the name of a Physical Material asset exactly. You can change the path of the material table in the projectile's actor blueprint under RealisticProjectileComponent->DataTablePath.

Hit effects are done in blueprints and also maps the name of a Physical Material to its sound cue and particle effect. The particle effect can be empty. If using BP_MP_Projectile_Base, you can create a child class and override the path to the table.

1.3 Plugin Structure

The multiplayer functionality is all in the Blueprint classes.
Blueprints\BP_FirstPersonCharacter
Blueprints\BP_MP_PlayerController
Blueprints\BP_MP_Projectile_Base
You should do some tutorials on replication in Unreal Engine in order to understand them.

The meat of the physics is in a class called RealisticProjectileComponent. In Unreal Engine, Actors are things you can spawn in the game, and components are things you bolt on to actors. As an example there is a "BP_MP_Projectile_Base" Actor class consisting of a sphere collision component as the scene root, and a RealisticProjectileComponent, which uses the scene root to do collision detection sweeps.



2 How it Works

The RealisticProjectileComponent is an implementation of this paper. If for some reason the link is broken, the details are:
Giliam J. P. de Carpentier, "Analytical Ballistic Trajectories with Approximately Linear Drag," International Journal of Computer Games Technology, vol. 2014, Article ID 463489, 13 pages, 2014.
Thanks Giliam! The equations are deterministic, so the calculated position of the projectile will not depend on how many times per second the position is updated (frames per second or ticks).

The RealisticProjectileComponent.h file will look for a materials data table in the project. It will look for the table at \RMPP\DataTables\TBL_MaterialPropertiesTable but this location can

be changed in RealisticProjectileComponent.h. The table contains a name of a physical material, and a toughness value for that material. If the table is not found, or the physical material name is not found in the table, a default toughness is used. Toughness affects how likely a bounce is, and how easily projectiles pass through an object.

2.1 Basic Multiplayer

When a player fires, a projectile is spawned immediately and the client also sends the location and direction to the server, who multicasts it to all clients. When projectiles collide with objects, some randomness is used. The random seed is also sent so the same random numbers are generated on each machine.

In all multiplayer games, moving objects will be in very slightly different positions on each machine. So if you want the same trajectory on two machines, bullets cannot bounce off or slow down because of other players, or physics objects. You may well want to sacrifice this so bullets can bounce off everything - just change the collision settings in BP_MP_Projectile_Base.

Bullets must remember which player fired them, as the original client will get the multicast from the server, but should not re-fire the same bullet. Clients report when their bullet hits an enemy. In the demonstration, the server then checks independently that the bullet travels there, and calculates and applies damage. This functionality is in BP_MP_PlayerController.

2.2 Physics

Each frame Unreal Engine calls a tick function in every object in the world. The name in this case is URealisticProjectileComponent::TickComponent(). One of the arguments is float DeltaTime, which gives the game time since last tick. The tick function performs these tasks:

1. Calculate the new position the projectile should be in this tick.
2. Sweep the collision primitive from the old position to the new position.
3. Handle any collisions/hits that happen.

Calculate New Position

If we are in air, the new position is calculated as per Carpentier paper. If we are in an object, there is a constant deceleration related to the material toughness. Gravity has no effect while inside an object. When sliding along a surface a frictionless parabolic arc is used, or if friction is on, the traditional non-deterministic stepping of the component is done.

Handle Collisions

First the projectile decides whether to bounce or to penetrate, based on the angle of impact, toughness of object, and velocity. Unlike real life, sometimes one object can be inside another in the game world. The projectile class remembers each ActorComponent it enters and forgets them as it exits.

When the tick function is called, imagine the projectile should advance 10 milliseconds. But if it hit something halfway, it has only moved 5 milliseconds worth. So we calculate how long it takes the projectile to get to the hit location, and then continue computing the trajectory until we use up all our time.

Collision Settings

Normally in Unreal Engine, object types that can't overlap are set to 'block' each other. Objects that can overlap and need to notify us when they do should be set to 'overlap.' For the Root component of our projectile actors, setting the collision to block means the other object will slow

the projectile down, and overlap means the same as usual. If you need the projectile to never overlap with an object, setting the toughness of its physical material very high will prevent this.

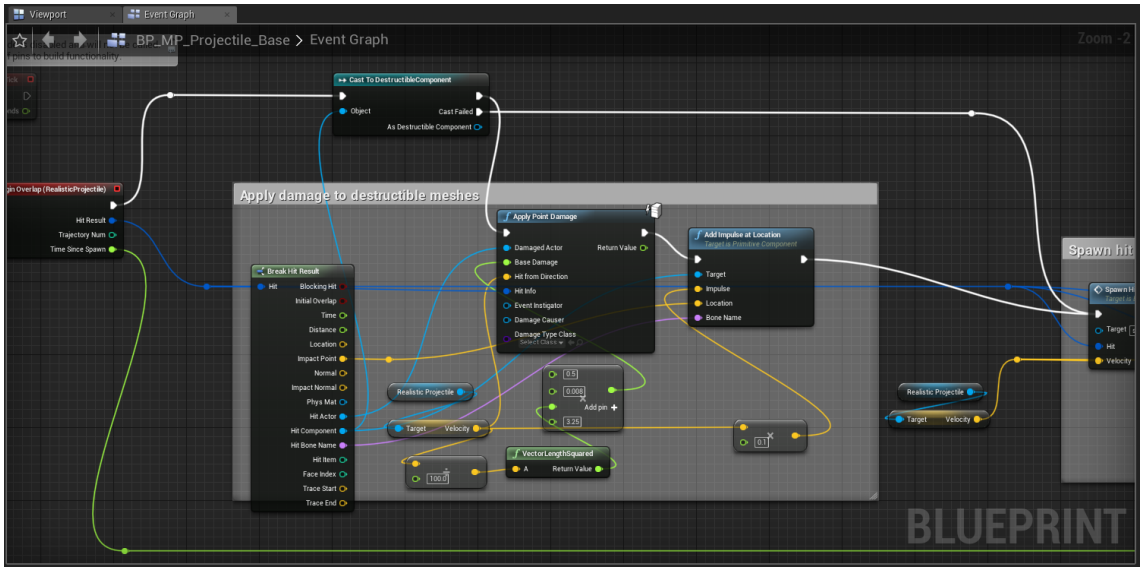
There is a special system for arrows to eject them from tough materials. The property is RealisticProjectileComponent->Settings->MaxEmbedToughness.

3 The Project Won't Build!

1. Open the project directory (you can right click the project and select Show in folder).
2. Delete the Build, Intermediate, and Saved directories.
3. Right click the .uproject file and select "Generate Visual Studio project files".
4. Open the new .sln file.
5. Make sure your project is the startup project in the solution explorer. If it is not in bold, right click the project name and select "Set as StartUp Project"
6. Run (without debugging)

4 Enabling Apex Destruction

1. In your project click Settings and select Plugins.
2. Enable Apex Destruction under the Physics Category and restart the editor.
3. Right click any static mesh you like and select "Create Destructible Mesh". There are some in RMPP\Meshes.
4. Play with the settings and click Fracture Mesh. World support and support depth=1 are useful.
5. Add the new Destructible Mesh to the Map
6. Open the blueprint at Blueprints\BP_MP_Projectile_Base. In its Event Graph, there is already a section set up for destructible meshes, you just have to cast the hit to a destructible mesh and wire the execution thread through the "Apply Radial Damage" function.
7. Run the game and shoot the mesh.



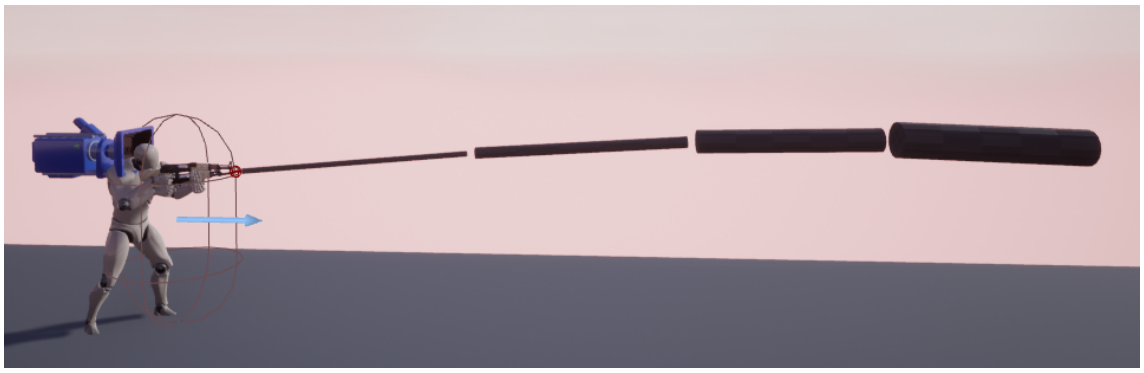
A good setup for applying APEX Destruction damage and impulse is shown.

5 Custom graphics for prediction and tracers

To create your own prediction graphics, you can use the GetPredictedTrajectory function to return an array of world location vectors. Another function is available that returns velocity and others too.

Creating a tracer is up to you, but it will probably take the form of attaching a particle effect to the actor. Useful information is the current speed which can be accessed through
RealisticProjectileComponent->Velocity, and also
RealisticProjectileComponent->InitialConditions
which can be used so tracers don't clip through their start point.

The provided tracers and prediction visual work on the same odd principle - one/many static meshes (cylinder-shaped) are scaled up as they get further away (otherwise they disappear from view). You may need to do the same for your effect.



An exaggerated view of how the path prediction and tracer effect work.

6 Lag Compensated Multiplayer Setup

From UE4.27, functions are available to help with a lag compensated setup, which offers server authorative hit detection, as described in this short article from Gamasutra about Mechwarrior Online. Recording the hitbox locations is somewhat CPU intensive. A modest Ryzen 5/Core i5 machine can run a server with 50 players at 60fps.

6.1 How to Implement Lag Compensation

The very base of the lag compensation system in MPPP is four functions -
RealisticProjectile->FastForward
Move a projectile forward in time by a small amount (eg 100ms), calling its event listeners on the way (OnComponentHit). Call on replicated clients.
RealisticProjectile->FastForwardwithHistory
Move a projectile forward in time by a small amount (eg 100ms), calling its event listeners on the way (OnComponentHit) for non-pawns. Also tests collision with a provided history of pawn hitbox locations. Call on server.
AddFrameToRewindHistory
Given an actor and a FRewindHistory struct, record the locations of actor components with a specific tag. Also clear out old frames. You may choose to use this or the alternative skeletal mesh version.
AddFrameToRewindHistoryMesh
Given a skeletal mesh component and a FRewindHistory struct, record the locations of collision shapes as per the physics asset. Also clear out old frames. You may choose to use this or the alternative tag version.

The plugin comes with a small framework that demonstrates how to implement these in c++, consisting of
AMpppLagCompProjectile - A projectile actor
AMpppLagCompPlayerState - A playerstate that records hitbox locations as long as the player

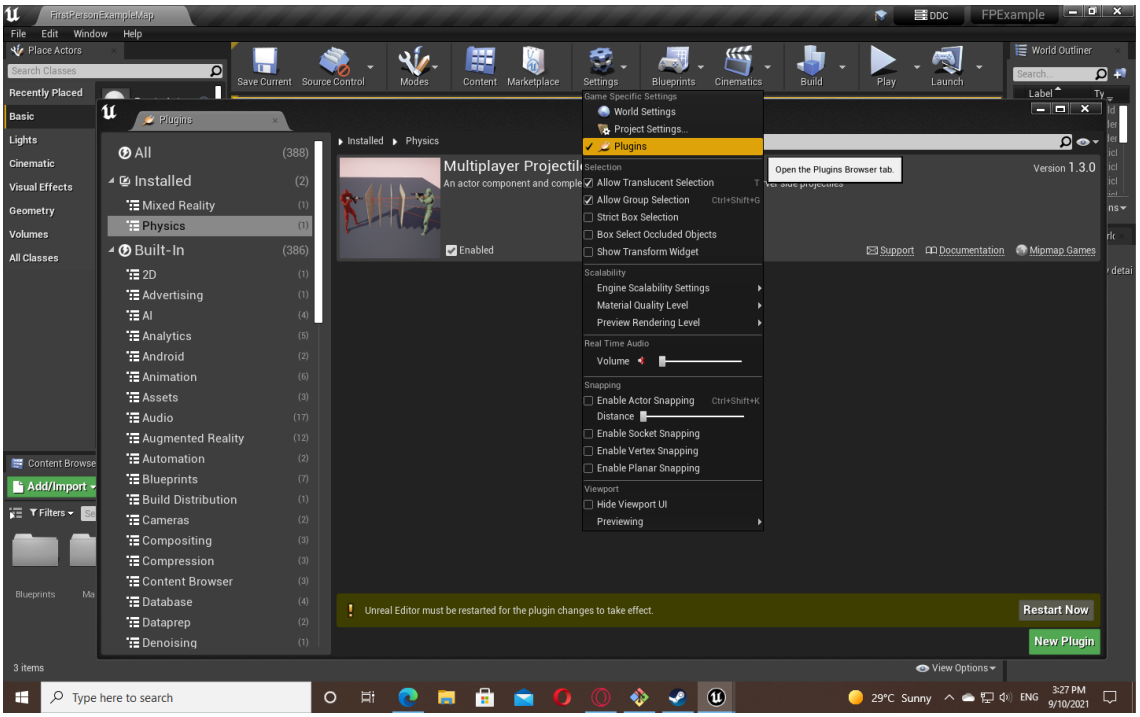
pawn blueprint implements the HitboxRecorder interface.
UMpppLagCompLauncherComponent - An actor component that handles the netcode of firing projectiles and hit testing with hitbox histories.
You may choose to use classes derived from the framework to quickly build a new game, or build from scratch/integrate with the functions mentioned earlier.

The following section describes using the functions to integrate into an existing game.

6.2 Integrating Lag Compensation into the First Person Template

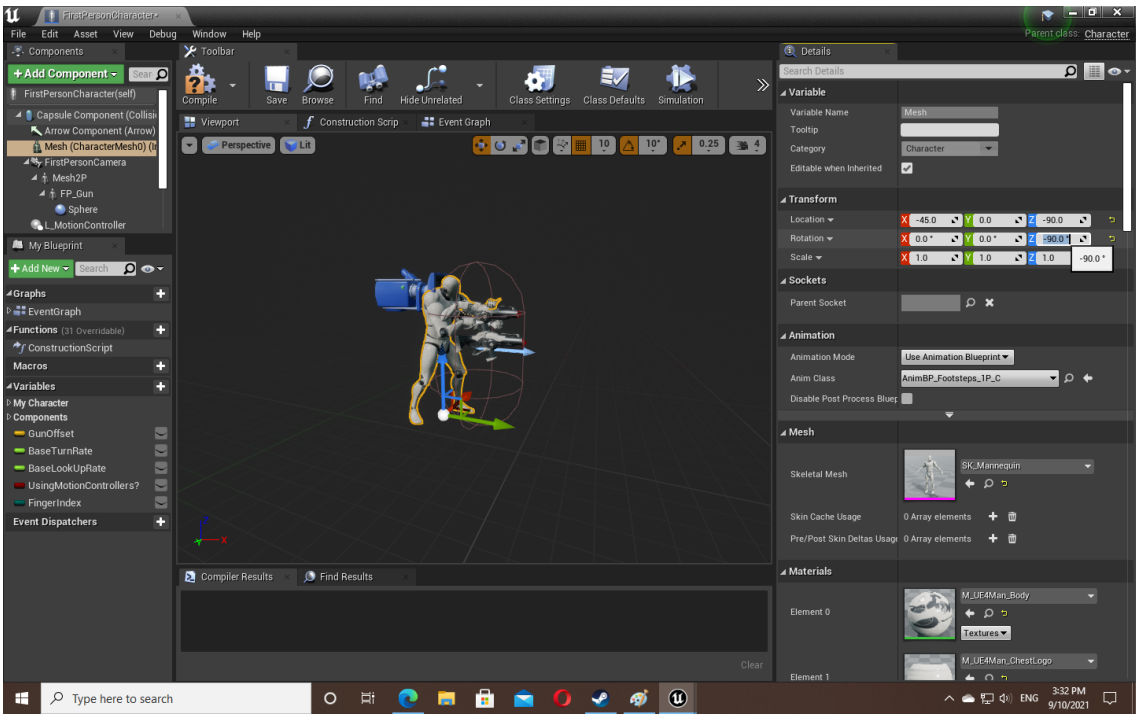
6.2.1 Turn on the plugin

Make a new blueprint project using the FirstPerson template. Go to Settings->Plugins->Physics and turn on MPPP.



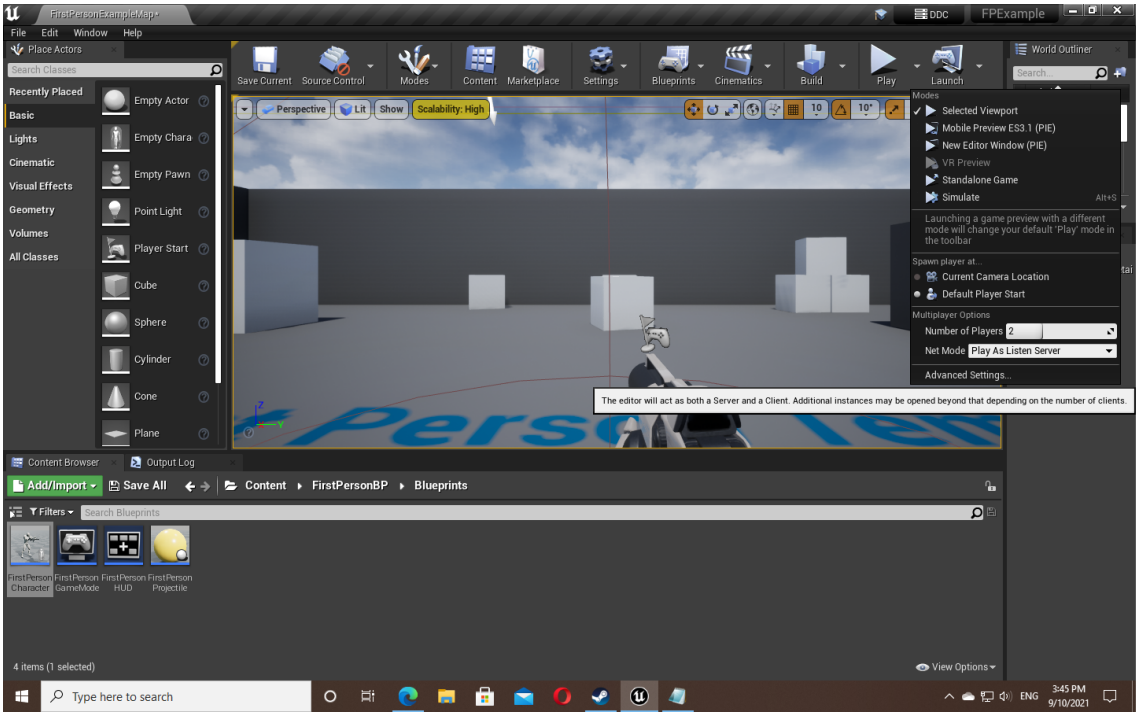
6.2.2 Give the player a 3rd person hitbox

At the moment other players can't see each other properly. In the project content open Content/-FirstPersonBP/FirstPersonCharacter. Use the existing Mesh (CharacterMesh0) and change the skeletal mesh to the UE4 mannequin which is included in the MPPP Plugin.



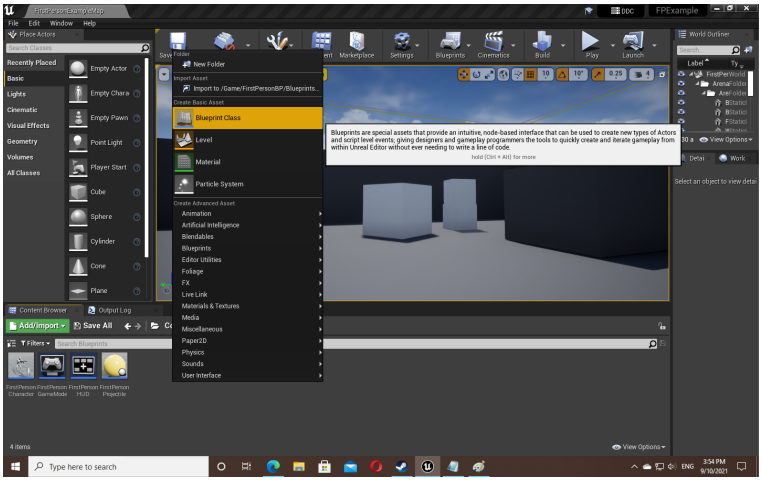
Change the mesh's location and rotation to match the FP mesh. Change the Anim Class as pictured. Change the collision settings to custom and set it to Query Only, WorldDynamic type, and Overlap with projectile types. Make sure Generate Overlap Events is checked.

Now to test that other players can see each other. In the play menu, set to listen server and 2 players. The editor window will be the server and another window opens as the client. Click play and run around. Delete the existing character object in the map if the rotation is off.

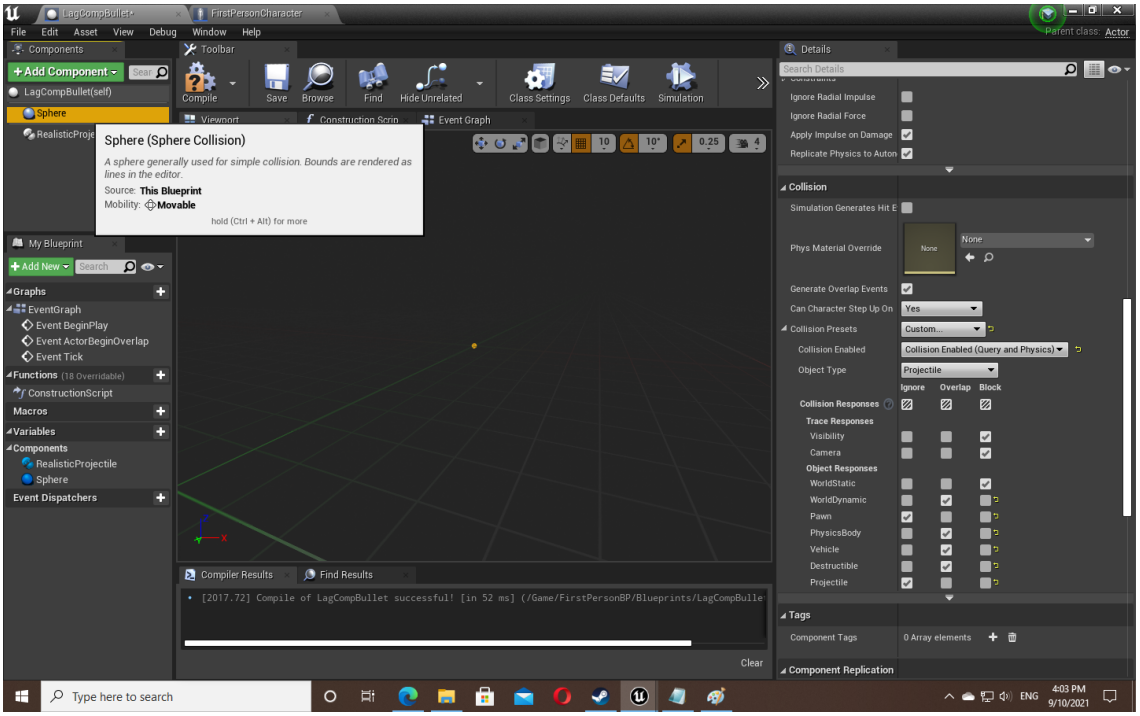


6.2.3 Make a Projectile Actor Class

Right click the content browser and create a new Blueprint class, derived from Actor.



Once created, click Add Component, and add a Sphere Collision Component.

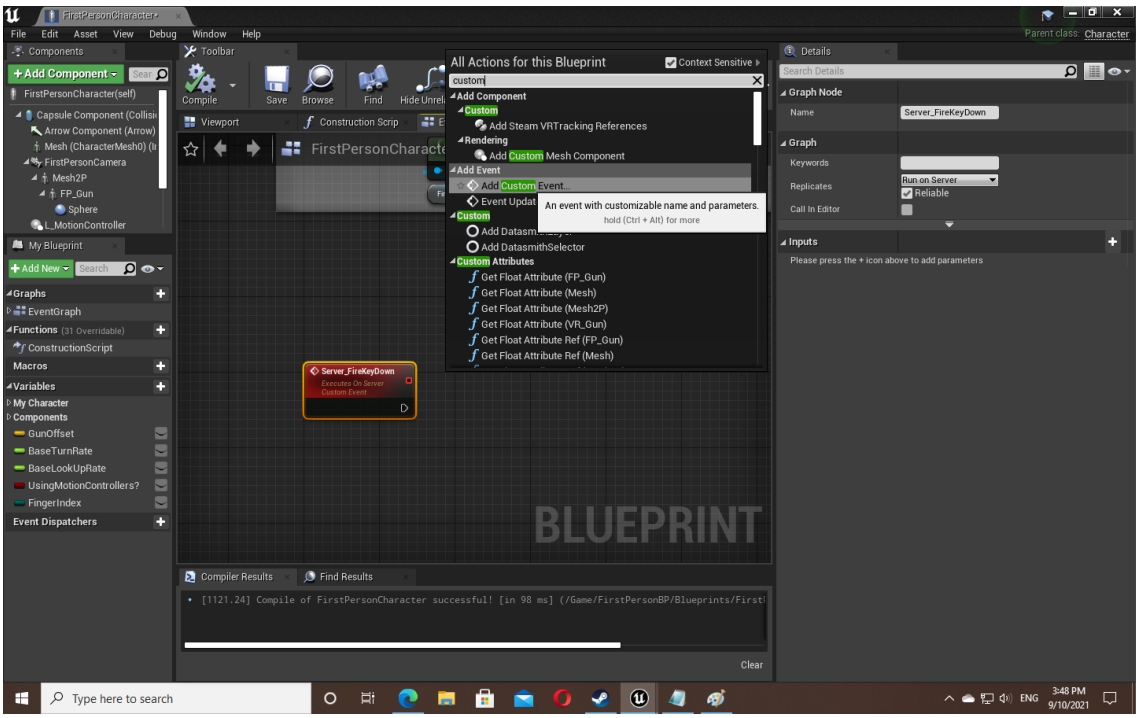


With this selected, set its collision settings to custom, Block WorldStatic, Overlap moving things, and ignore Pawns and Projectiles. Tick Generate Overlap Events. Drag the sphere collision into Default Scene Root to make it the root component. Change its radius to 1.0.

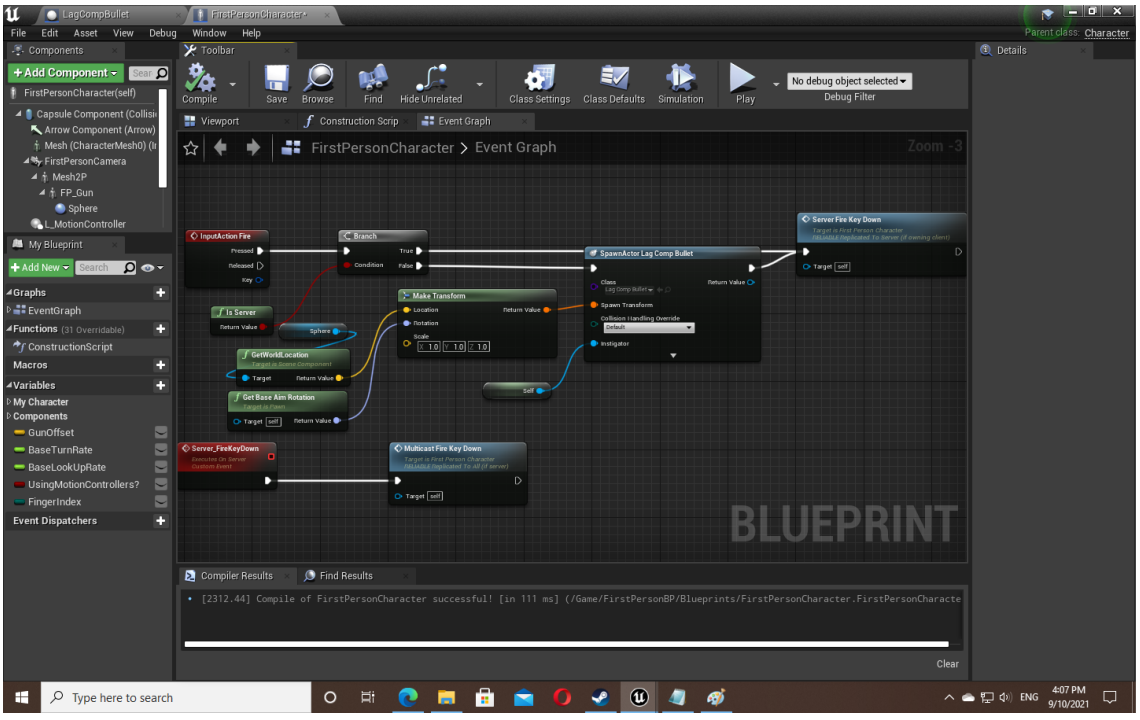
Add a Realistic Projectile Component, and under details, set Draw Debug Lines to true.

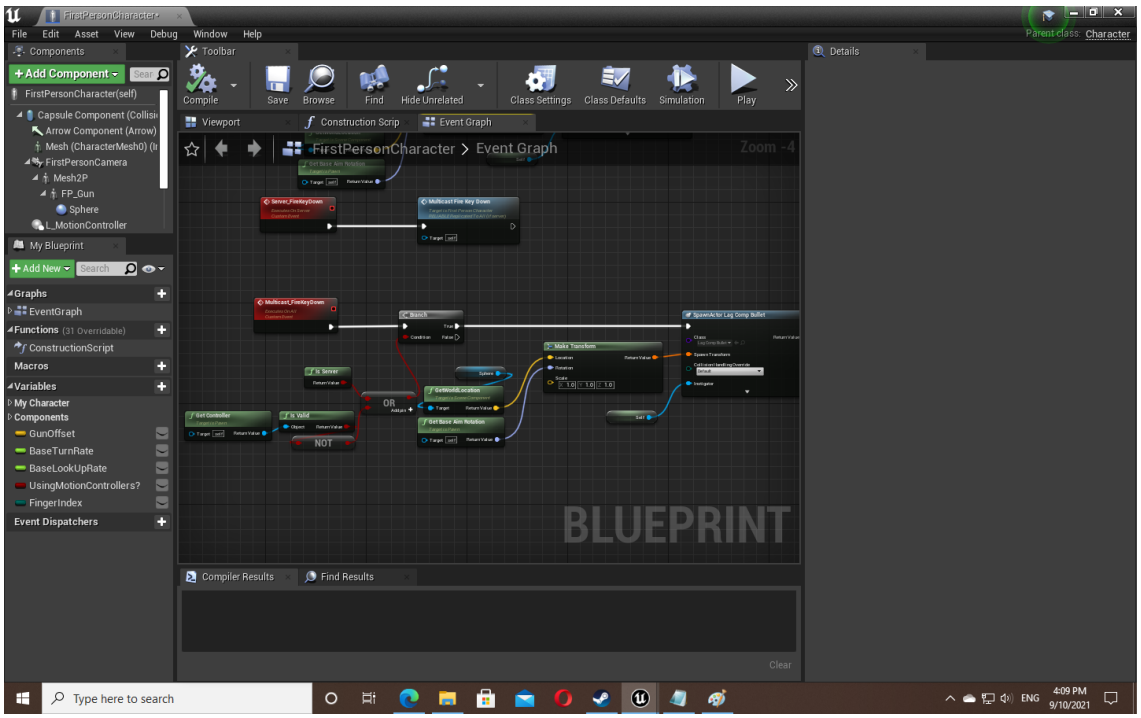
6.2.4 Set Up Networked Firing Code

Open the FirstPersonCharacter blueprint again.

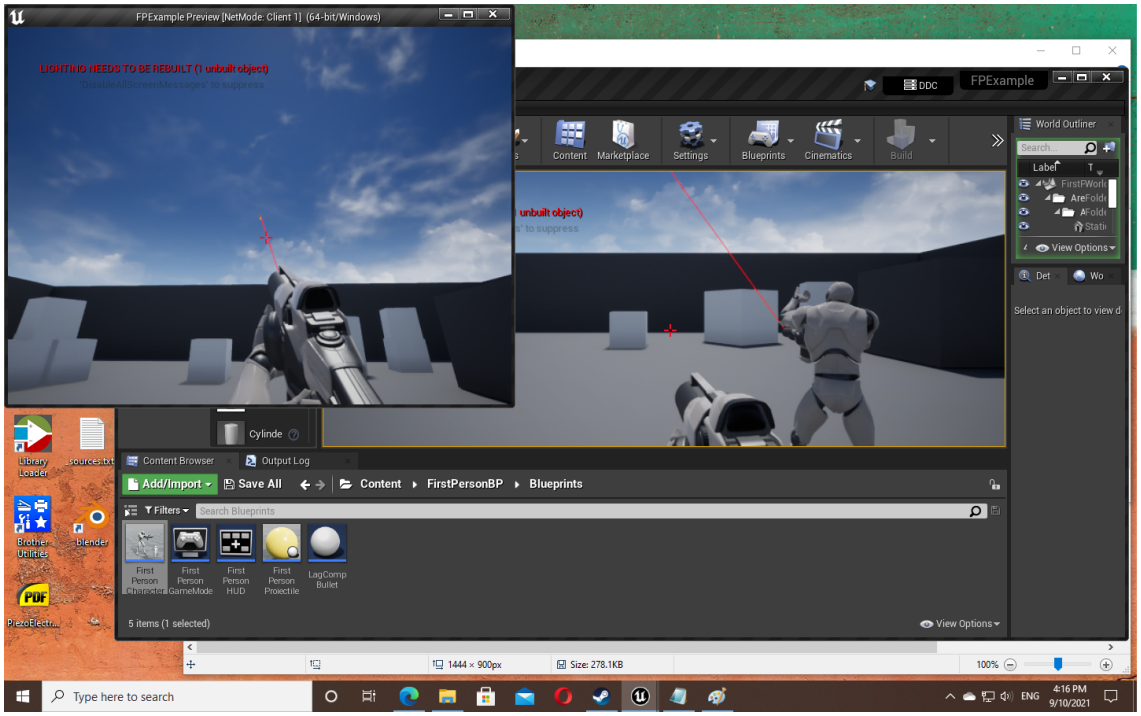


Right click the event graph and add some custom events. Server_FireKeyDown and Multicast_FireKeyDown. Make sure to change their replication setting to server and multicast, and both are reliable.

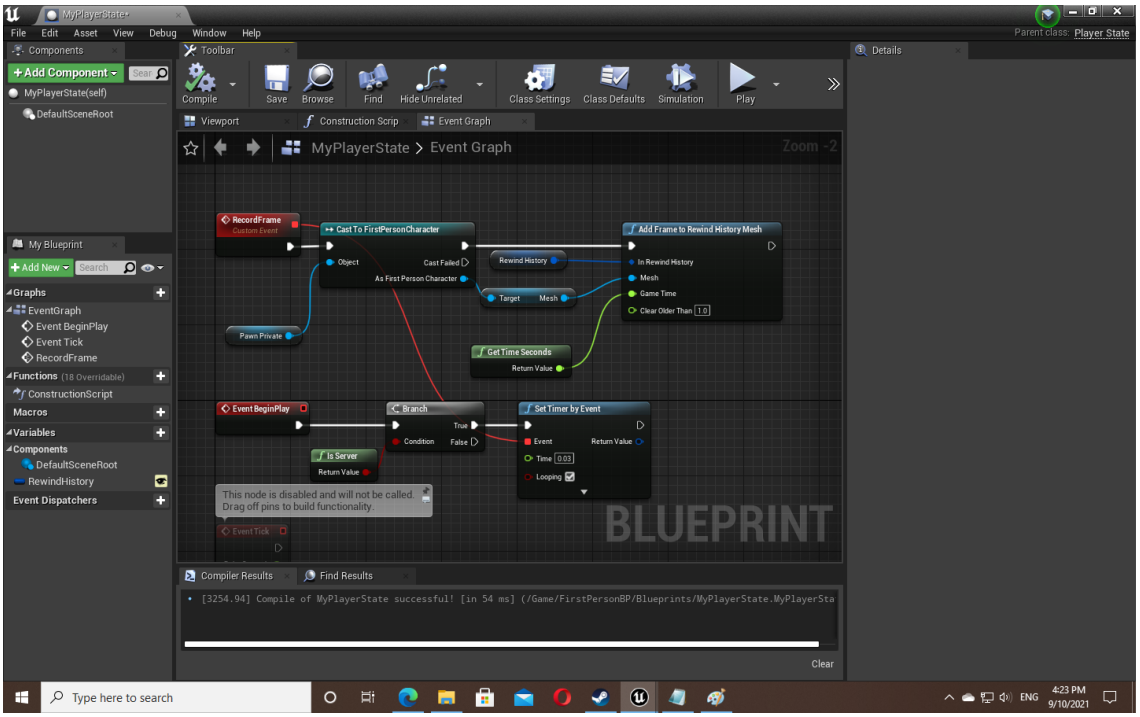




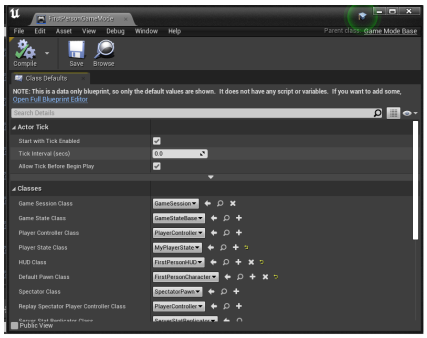
Disconnect input action fire key pressed, and instead if we are client, spawn a bullet, and then in all cases call Server_FireKeyDown. Server_FireKeyDown should then call Multicast_FireKeyDown. Copy the pictured blueprint, and then in game, check that firing on a client replicates to the server and vice versa.



6.2.5 Record the Player Hitboxes



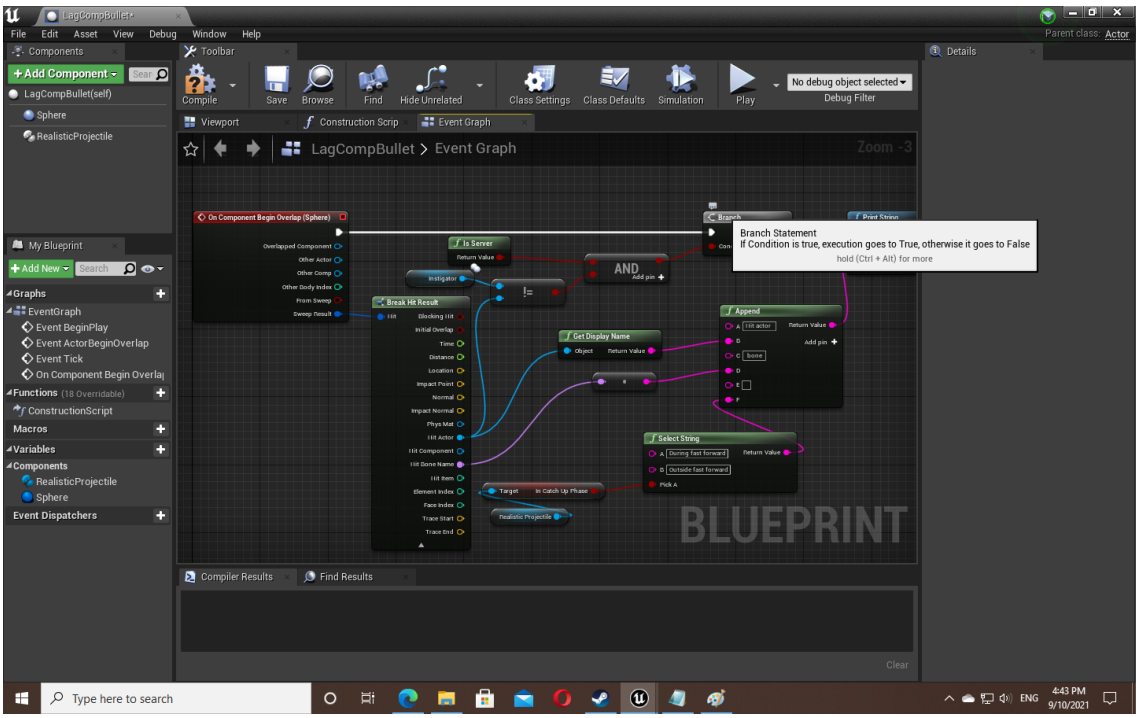
Make a new blueprint derived from player state. Add a variable to it of type RewindHistory. Copy the pictured blueprint to store the hitbox locations every so often.



Open FirstPersonGameMode and change the player state class to the new one just made.

6.2.6 React to Overlap Events

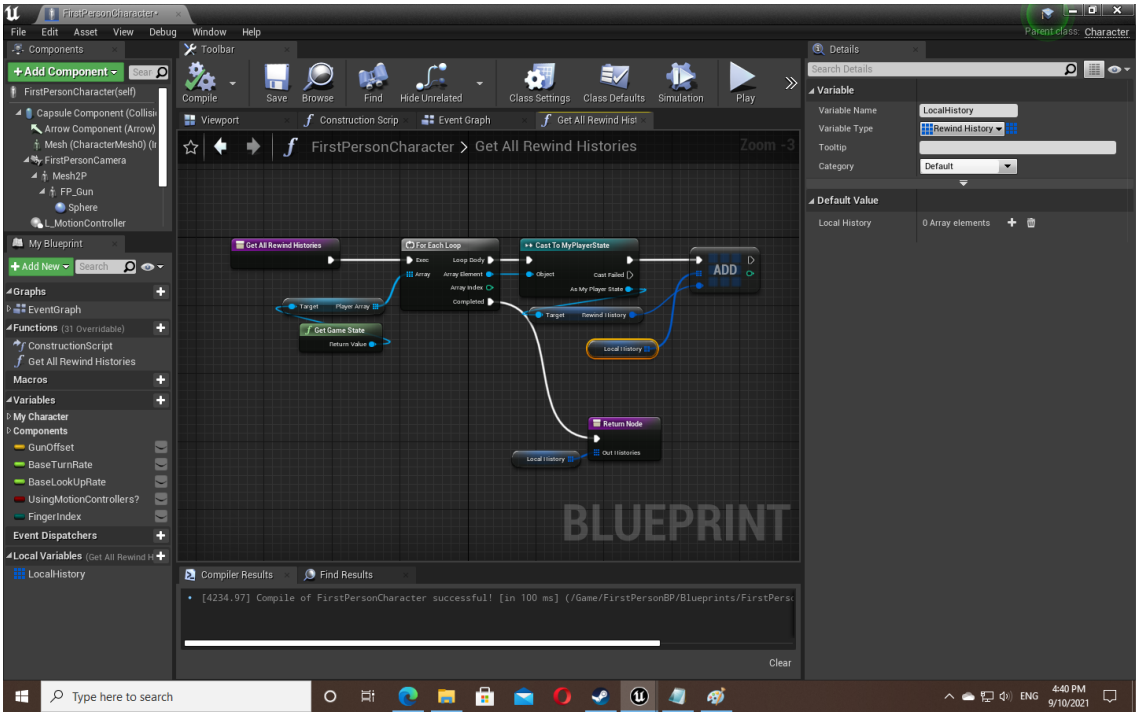
Open the bullet actor that was created and click the sphere collision component. Scroll to the bottom of its details and add a OnComponentBeginOverlap Event.



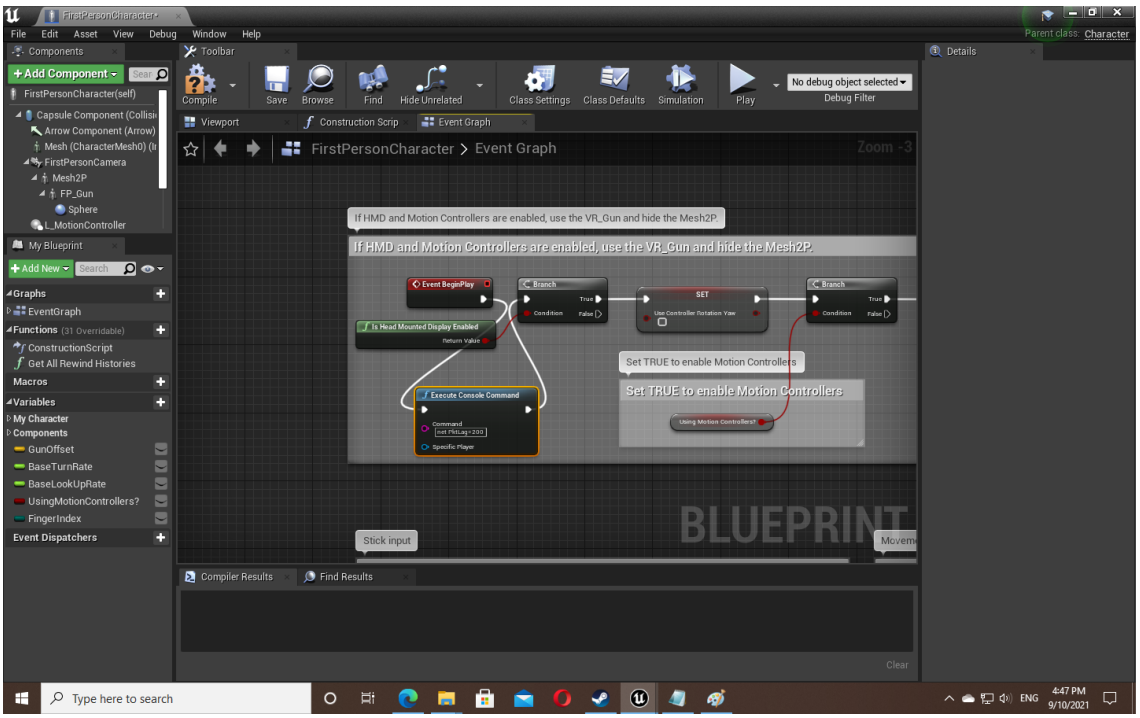
Copy the pictured blueprint and check by playing that the bullet prints on an overlap. If nothing happens, double check the collision settings of each object.

6.2.7 Implement Fast Forward

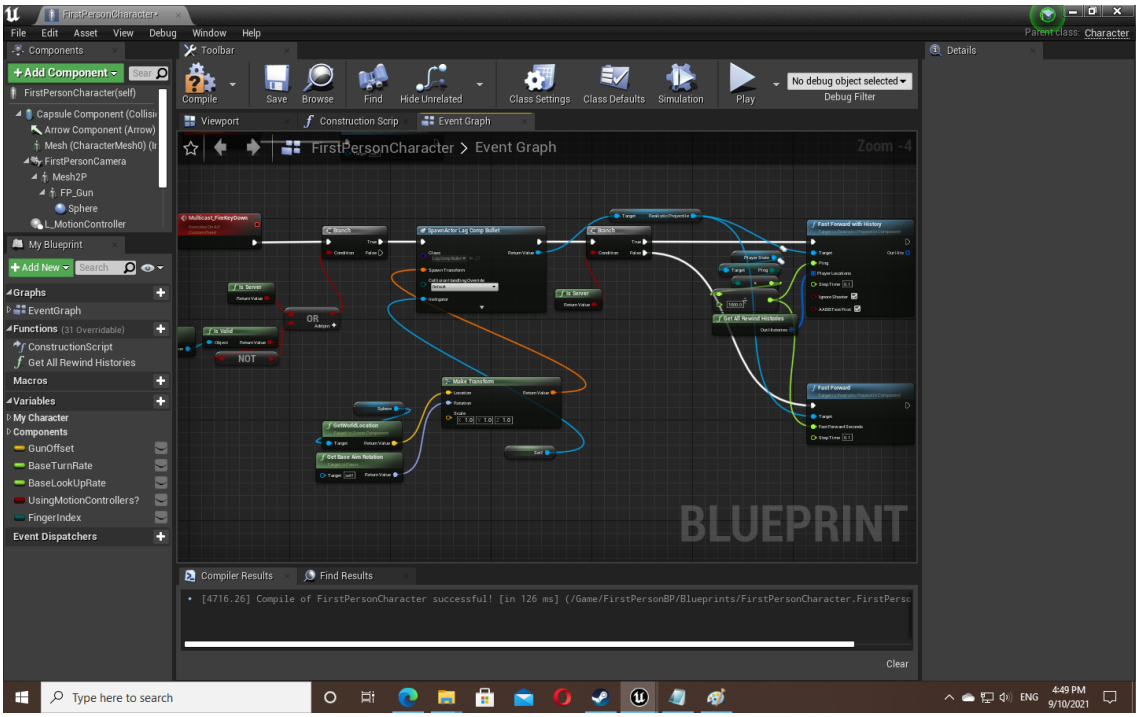
Open the FirstPersonCharacter Blueprint again, and add a function called GetAllRewindHistories.



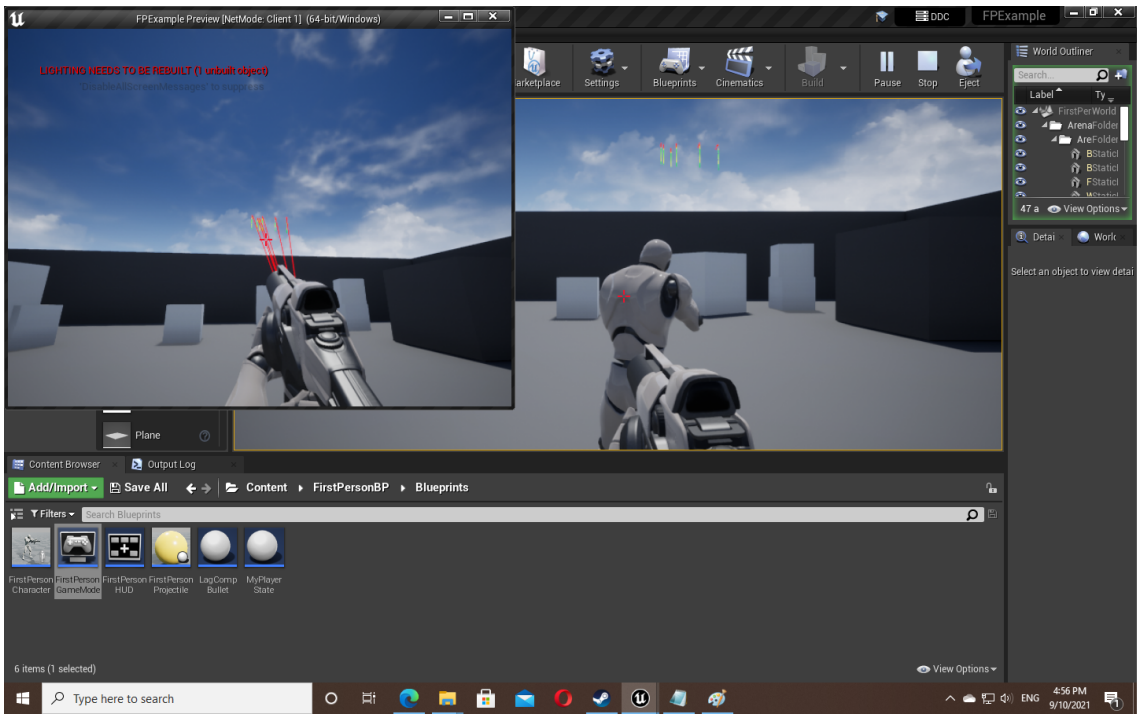
It should Output an array of RewindHistory types. Set the function to pure so it doesn't need execution.



After beginplay, make an exec console command and exec "net PktLag=200". This should simulate a nice big lag amount that is noticeable for testing. Now test that the character's movement is delayed between the client and the server.



Copy the pictured nodes after multicast_FireKeyDown and check that when firing on the client, the projectile 'jumps' forward by the ping amount, and the overlap appears as a rewind hit. The print-out should show a hit during the fast forward phase.



7 Hints and Tips

- If you cannot see the tracers, use slower bullets. Real != fun.
- The basic anti-cheat function supplied (ServerCheckClientHit) is a little expensive so if your game involves plenty of hits, consider enlarging the step time parameter, only checking some hits, or lastly optimizing the c++. The function spawns an actor each call, so maintaining a pool of actors would be more efficient.
- If your game involves large distances make sure the 'net cull distance squared' setting on relevant actors is very high. If players further than the net cull distance shoot, it is not replicated to other machines.
- The damage system comprises of the character's skeletal mesh + its physics asset, and damage is applied in the PlayerController blueprint. A bullet hits a capsule in the physics asset, which corresponds to a bone in the skeletal mesh eg. spine_01.
- Physics traces and sweeps in Unreal will stop short when a blocking hit is encountered - you will not get more than one blocking hit (although blocking objects the trace started inside will appear too).