Week 5 – Weapons

**Section 1 – Creating the base class**

Create a new C++ class called WeaponBase, derived from Actor.

For starters lets give it a protected mesh component in the header.

UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Components")

USkeletalMeshComponent\* MeshComp;

Also we are going to need to get this mesh from the character so add a getter

USkeletalMeshComponent\* GetWeaponMesh() { return MeshComp; };

In the constructor create the mesh

MeshComp = CreateDefaultSubobject<USkeletalMeshComponent>(TEXT("MeshComp"));

RootComponent = MeshComp;

Also make sure to include #include "Components/SkeletalMeshComponent.h"

Go into your Character header and give it a protected Type reference to a WeaponBase and a protected pointer to a WeaponBase, don’t forget the include too

UPROPERTY(EditDefaultsOnly, Category = "Weapons")

TSubclassOf<class AWeaponBase> WeaponType;

UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Weapons")

AWeaponBase\* Weapon;

In the Characters BeginPlay() function (in C++) spawn the weapon based on the type specifier and attach it to the “GripPoint” socket on the Character mesh, also make sure to set the owner so the weapon knows who is holding it.

if (WeaponType != NULL)

{

Weapon = GetWorld()->SpawnActor<AWeaponBase>(WeaponType);

Weapon->GetWeaponMesh()->AttachToComponent(Mesh1P, FAttachmentTransformRules(EAttachmentRule::SnapToTarget, true), TEXT("GripPoint"));

Weapon->SetOwner(this);

}

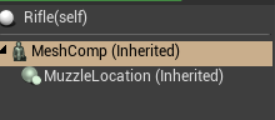
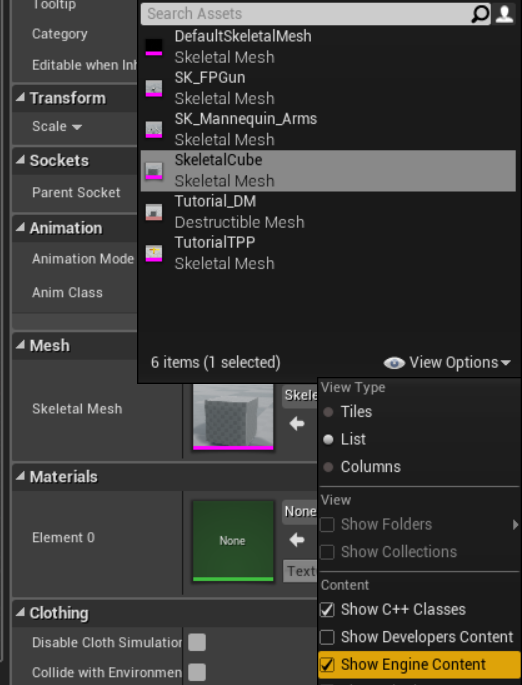
**Section 2 – Creating and assigning a weapon**

Compile and open up the editor

Create a new blueprint deriving from WeaponBase and call it “Rifle”

Select the MeshComp and Give it a mesh, for now use SkeletalCube so we can see it worked.

*You may need to “Show Engine Content” – Select the drop down->View Options -> Show Engine Content*

**

Open up your character blueprint and set the WeaponType in the details to Rifle.

Run the game and you should have a Cube in your hands as well as the old default weapon.

**Section 3 – Refactoring the Character into WeaponBase**

For this part you will need to replicate what I did in the demo.

1. Anything in Character that involves the weapon should be refactored into WeaponBase (Other than the code we created in section 1)
2. FP\_Gun will not be used now so delete/comment any code relating to it, instead use our new “Weapon”
3. Create a FireWeapon() function in WeaponBase that handles the firing of projectiles, then call it from Character. It could return a bool if the firing was successful.
4. Your OnFire function in the character should look something like this:

void AGEPCourseworkCharacter::OnFire()

{

if (Weapon->FireWeapon())

{

WeaponFired();

// try and play a firing animation if specified

if (FireAnimation != NULL)

{

// Get the animation object for the arms mesh

UAnimInstance\* AnimInstance = Mesh1P->GetAnimInstance();

if (AnimInstance != NULL)

{

AnimInstance->Montage\_Play(FireAnimation, 1.f);

}

}

}

}

1. You may also want to comment/delete all the VR specific code.
2. When you refactor the projectile spawning code into your FireWeapon() function you will need to change this line:

const FRotator SpawnRotation = GetControlRotation();

To this:

const FRotator SpawnRotation = UGameplayStatics::GetPlayerController(GetWorld(), 0)->GetControlRotation();

1. If you are successful then the game should function exactly as it did before refactoring.

**Section 4 – Line Tracing**

*For this section we are going to set up line tracing (also called raycasting) for simulating fast moving bullets and hit detection.*

Include #include "DrawDebugHelpers.h" this will let us draw lines to visualize our line tracing.

In your FireWeapon() function in you WeaponBase comment out the code for firing the projectiles and add the following code:

//get the "owner" of the weapon, i.e the character holding it

AActor\* MyOwner = GetOwner();

if (MyOwner)

{

//set the eye location and rotation by accessing the characters "Eyes view point"

//in 1st person this is the center of the viewport

FVector EyeLocation;

FRotator EyeRotation;

MyOwner->GetActorEyesViewPoint(EyeLocation, EyeRotation);

//set a vector of huge length(or as far as you want bullets to hit) from the eye location outwards

FVector TraceEnd = EyeLocation + (EyeRotation.Vector() \* 10000);

//get the position of the muzzle of the gun

FVector MuzzlePos = MuzzleLocation->GetComponentLocation();

//set up params to ignore the character and the weapon

FCollisionQueryParams QueryParams;

QueryParams.AddIgnoredActor(MyOwner);

QueryParams.AddIgnoredActor(this);

QueryParams.bTraceComplex = true;

//create a hit structure to store data about the trace

FHitResult hit;

//trace from the eye location to the trace end

if (GetWorld()->LineTraceSingleByChannel(hit, EyeLocation, TraceEnd, ECC\_Visibility, QueryParams))

{

//draw a red line if it hits anything

DrawDebugLine(GetWorld(), MuzzlePos, TraceEnd, FColor::Red, false, 1.0f, 0, 1.0f);

//Get the hit actor from hit

AActor\* hitActor = hit.GetActor();

//hitActor->Destroy();

}

else

{

//draw a white line if it does not hit anything

DrawDebugLine(GetWorld(), MuzzlePos, TraceEnd, FColor::White, false, 1.0f, 0, 1.0f);

}

Compile and run, you should see a red line projecting out of the gun when hitting something or white if not.

**Section 5 – Creating the tracer effect**

1. Download the Assets from Blackboard
2. Open the project directory and find the Content folder
3. Drag the WeaponEffects folder into the Content folder.
4. Reopen UE4 and wait for the shaders to compile.

In your WeaponBase header create a new property for the tracer called BeamEffect

UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = Effects)

UParticleSystem\* BeamEffect;

In the .cpp include #include "Particles/ParticleSystemComponent.h"

In the WeaponFired() function find an approprtiate place and enter the following code:

if (BeamEffect)

{

//Spawn the particle effect at the muzzle socket and store it in a variable

UParticleSystemComponent\* BeamComp = UGameplayStatics::SpawnEmitterAtLocation(GetWorld(), BeamEffect, MeshComp->GetSocketLocation("Muzzle"));

//Set the vector parameter "BeamEnd"

//If the hit returns an actor set it to the impact point else set it to TraceEnd

BeamComp->SetVectorParameter("BeamEnd", (hit.Actor != NULL) ? hit.ImpactPoint : TraceEnd);

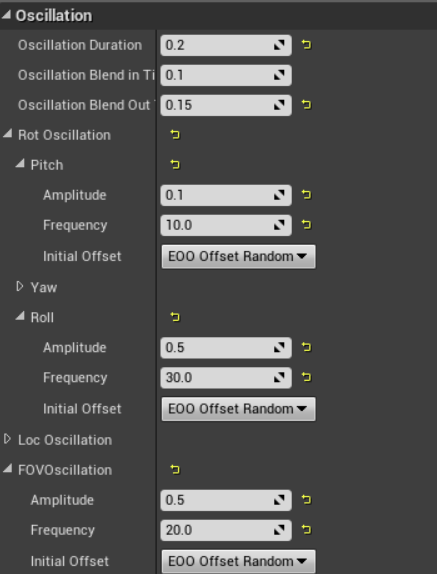
}

Compile, then in your Rifle blueprint set the BeamEffect property to P\_SmokeTrail

Comment out the debug lines and a trail should follow the path of the linetrace.

**Section 6 – Camera Shake**

1. Create a new blueprint and derive it from CameraShake
2. Call it something like “CameraShake\_RifleFire”
3. Open it up and adjust the properties. i.e:

****

1. In your WeaponBase header add a new property for a type of UCameraShake

UPROPERTY(EditDefaultsOnly, Category = Effects)

TSubclassOf<class UCameraShake> FireCamShake;

1. In the WeaponFired() function add the following call

GetWorld()->GetFirstPlayerController()->ClientPlayCameraShake(FireCamShake);

1. Set the property in your Rifle blueprint to “CameraShake\_RifleFire”

**Challenges (C++)**

1. Spawn a muzzle flash effect when shooting *\*hint\* Use the muzzle socket of the mesh*
2. Spawn a blood splatter effect when you hit an actor with a linetrace.
3. Refactor the projectile launching code into a class that derives from WeaponBase called ProjectileWeapon. Make sure to override WeaponFired().
4. Add rate of fire to your WeaponBase. *\*Hint\* You could use tick for shot interval cooldown*
5. Create a slow firing weapon with a big recoil/camera shake
6. Create an automatic weapon *\*Hint\* In character.cpp you could bind another function to IE\_Released*
7. Create a shotgun that scatters linetraces.
8. Allow the character to “carry” multiple weapons, switching on key press.