

Whiskey Business

Architecture/Design Document

Weapons System

Table of Contents

Change History	2
1. Introduction	3
2. Design Goals	3
3. System Behavior	3
4. Logical View	4
4.1 High-Level Design	4
4.2 Mid-Level Design	5
4.3 Detailed Class Design	6
5. Process View of the Weapons System	7
Light Melee Attack	7
Light Ranged Attack	8
6. Use Case View	9
6.1 Adding New Weapons	9

Change History

Version: 0.1

Modifier: Filipe Botelho

Date: 2025-02-17

Description of Change: Initial setup of Base and derived classes

Version: 0.2

Modifier: Filipe Botelho

Date: 2025-02-24

Description of Change: Added Aiming with a laser pointer

Version: 0.3

Modifier: Filipe Botelho

Date: 2025-03-02

Description of Change: changed laser pointer to draw line for more accuracy and changed activation times using AnimNotifyState

1. Introduction

This Document is to describe the Design of the Weapons System in the game Whiskey Business. It will go over the Design Goals and the decisions made in creating the weapons and how they interact in the game. It will also aid future development in creating new weapons in the game.

2. Design Goals

- Transitive Design: Weapons will be using Inheritance OOP design. Sharing as much similarities in the base class to all weapon types.
- Ease of Addition: The Design will make it as seamless as possible to add additional weapons in the future.

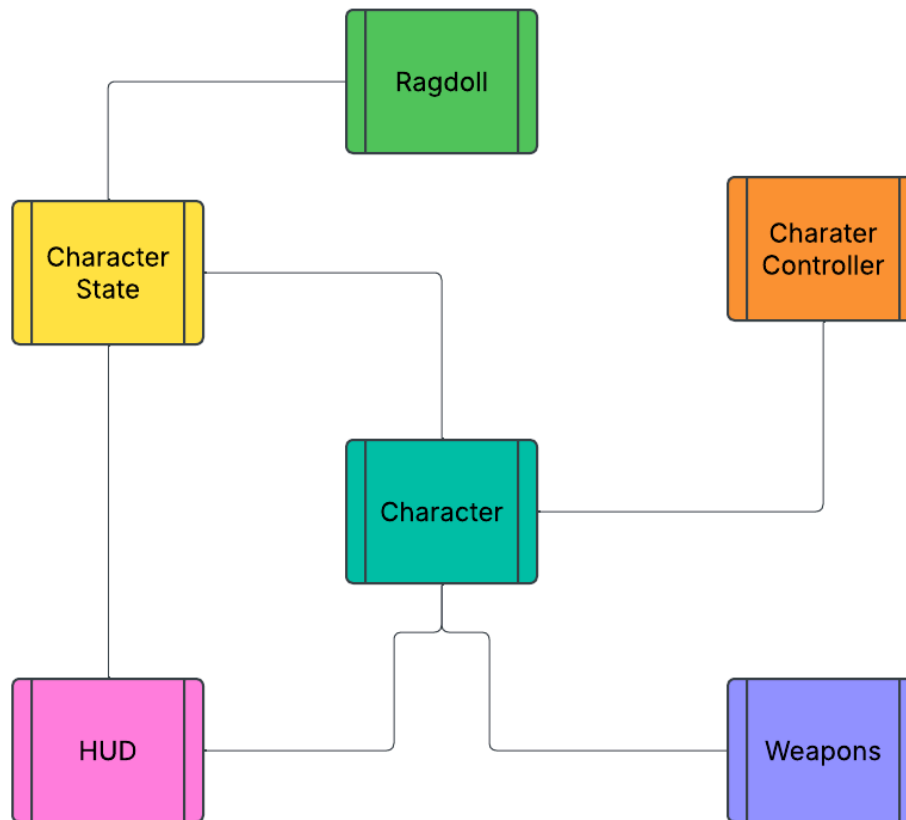
3. System Behavior

Weapons will appear on the map and can be picked up by the players. Using weapons will apply varying amounts of Damage, Stagger Damage (for ragdolling) and force dead on targets depending on the weapon.

4. Logical View

Diagrams showing the relationships and interaction between systems.

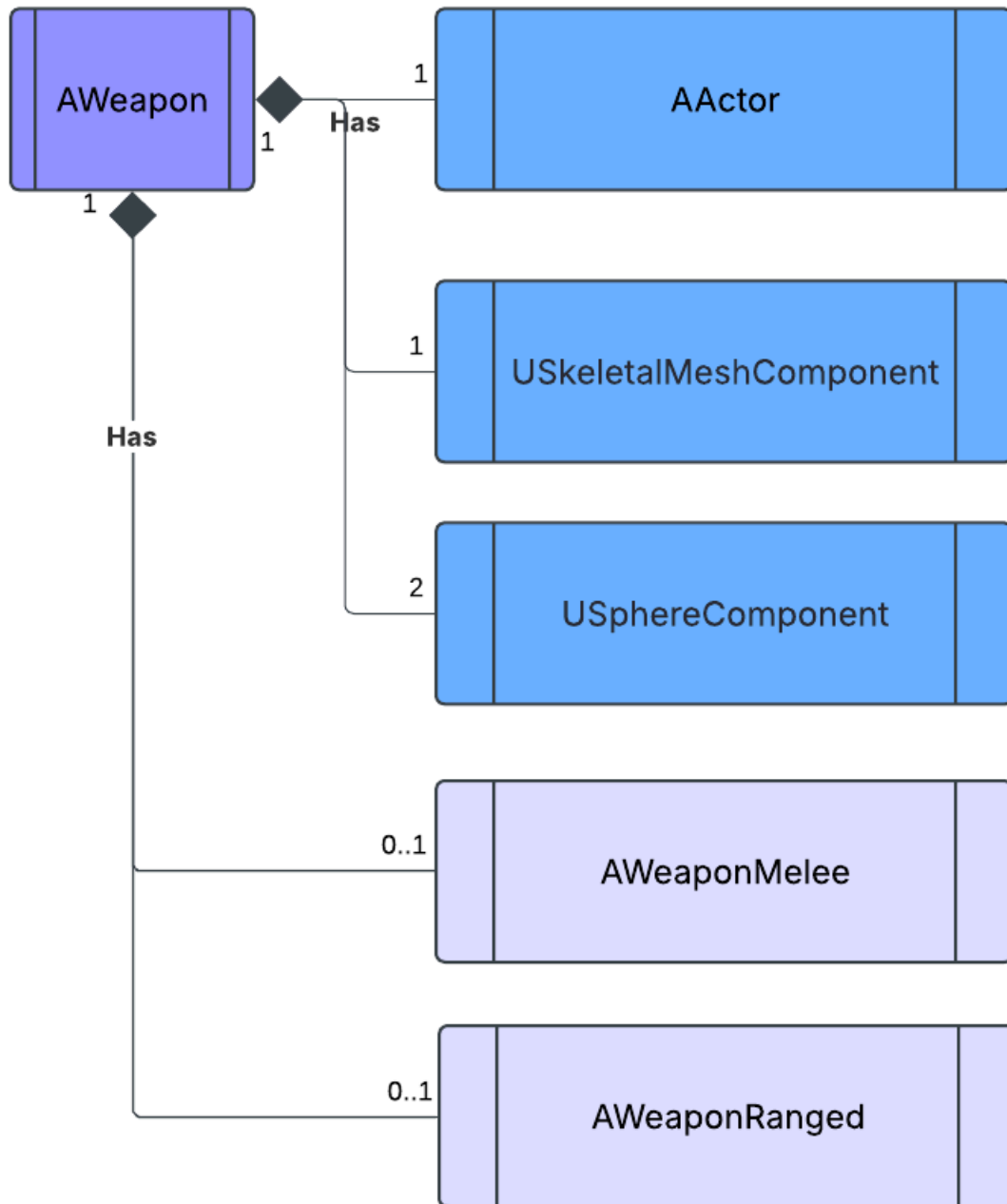
4.1 High-Level Design



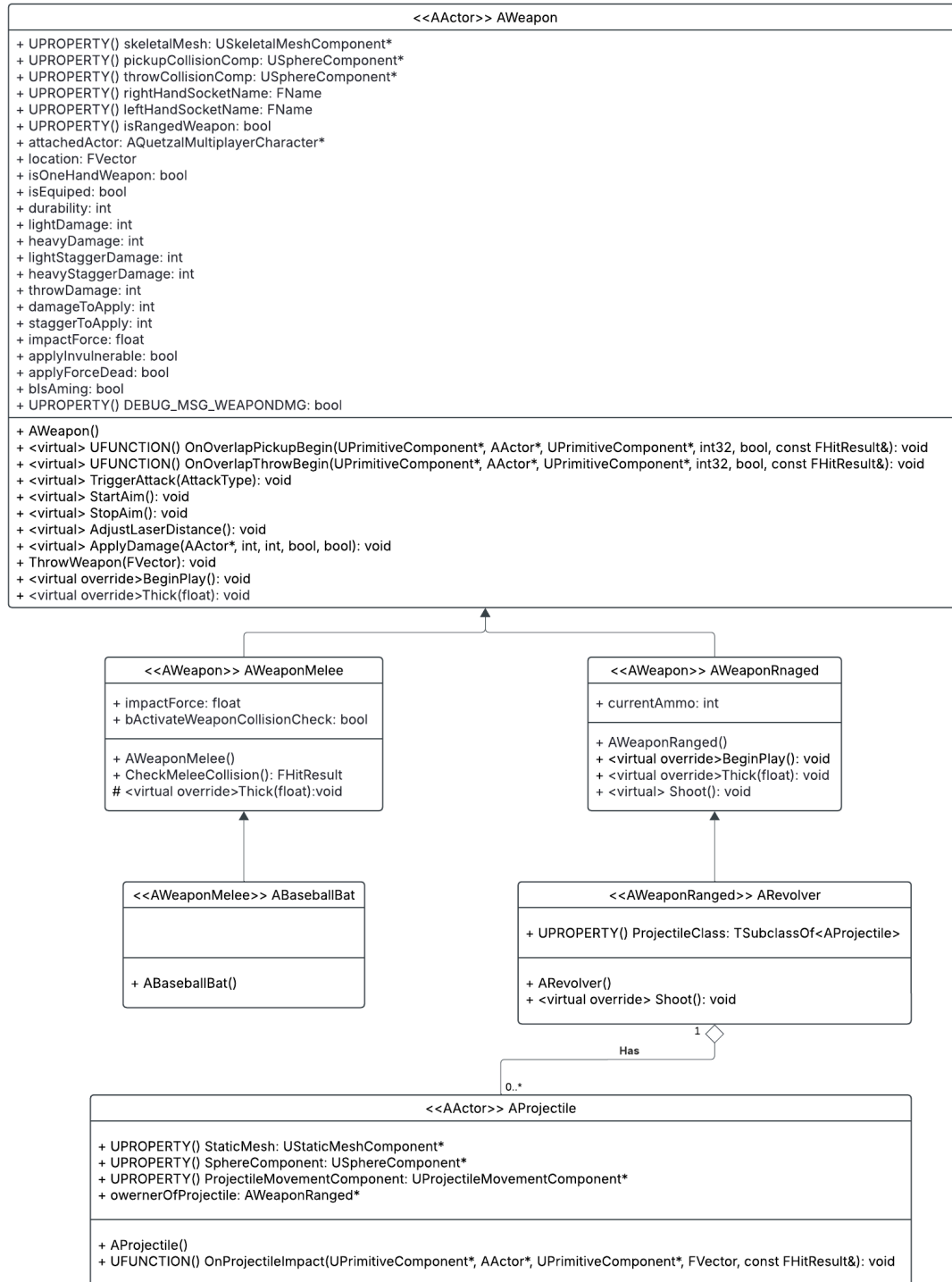
High-Level Design of varying systems in the game.

- The Character Controller will handle the input from the user that will control the player.
- The HUD will keep the players informed of the current health and score of the players using values from Character and Character State.
- Weapons will interact with the player by activating equipped weapons or receiving damage from weapons.
- Depending on the Character State the player can enter into a Ragdoll state.
- At the center is the Character interacting with all systems in some varying ways.

4.2 Mid-Level Design

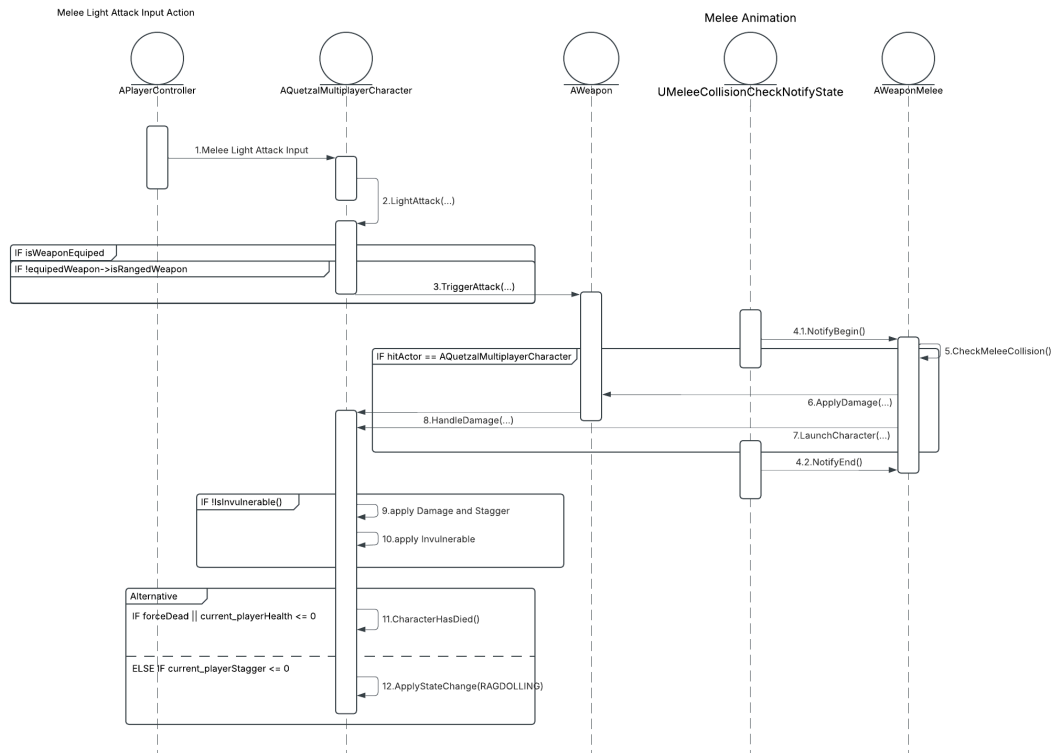


4.3 Detailed Class Design



5. Process View of the Weapons System

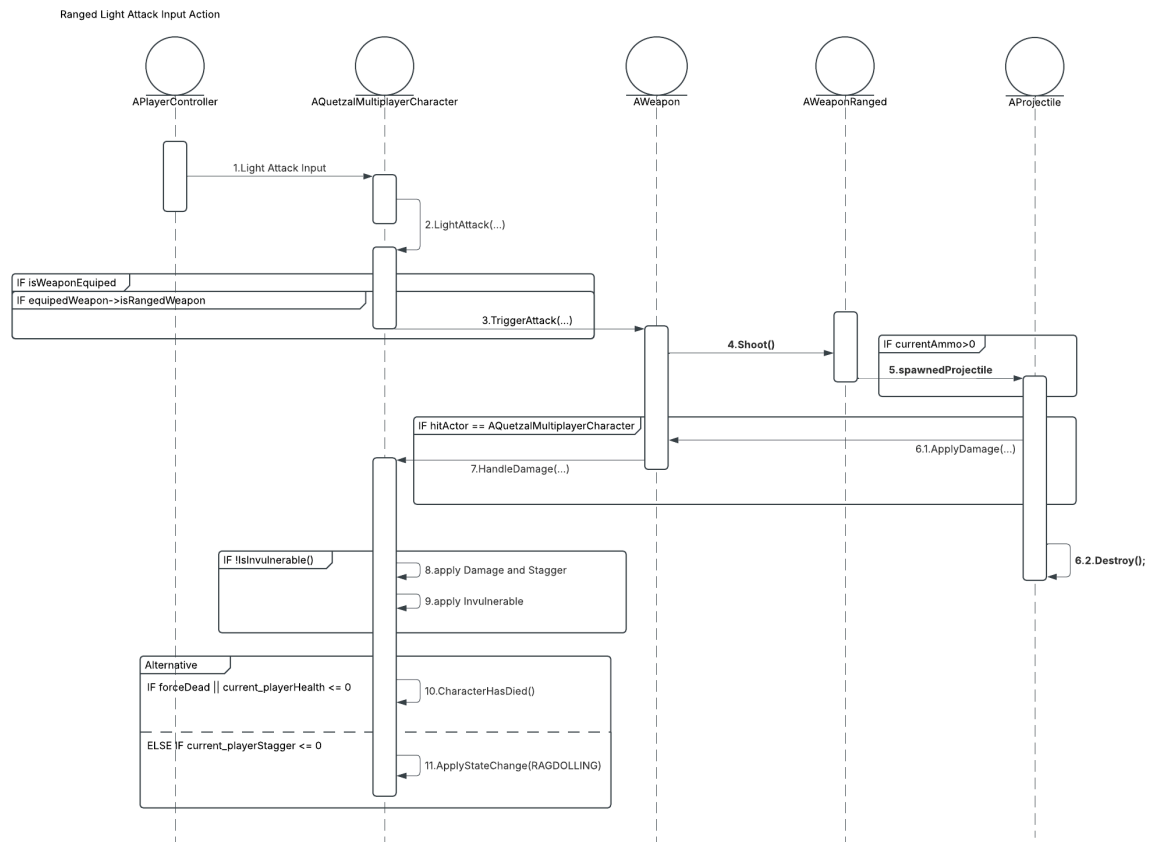
Light Melee Attack



1. Player will press the light attack action
2. The Character Controller will trigger the light attack action in Character.
3. If the Character has a melee weapon it will trigger the attack on said weapon.
4. The weapon will trigger an animation for light melee attack which at a certain point (frame 10) will trigger 4.1.NotifyBegin from MeleeCollisionCheckNotifyState.
5. In turn will initiate the CheckMeleeCollision function in WeaponMelee. It will keep checking until 4.2.NotifyEnd from MeleeCollisionCheckNotifyState is called.
6. Any Actor hit will trigger apply damage function from weapon which will call handle damage from character
7. In addition it will launch the character with some amount of force depending on the melee weapon.

8,9,10,11, and 12. Finally the Character will apply the damage, stagger damage if not invulnerable. Then in turn will check if the player is dead or initiate ragdoll.

Light Ranged Attack



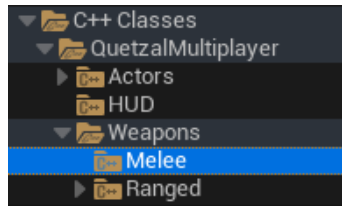
1. Player will press the light attack action
2. The Character Controller will trigger the light attack action in Character.
3. If the Character has a ranged weapon it will trigger the attack on said weapon.
4. The Weapon will then call the Shoot function from WeaponRanged. If the ranged weapon has ammo left it will spawn a projectile.
5. The projectile will then go on a forward path until it collides with something then 6.2destroy itself.
6. If the projectile hits a player it will call the apply damage function from Weapon
7. in turn it will call the Handle damage function from Character.

8,9,10, and 11 Finally the Character will apply the damage, stagger damage if not invulnerable. Then in turn will check if the player is dead or initiate ragdoll.

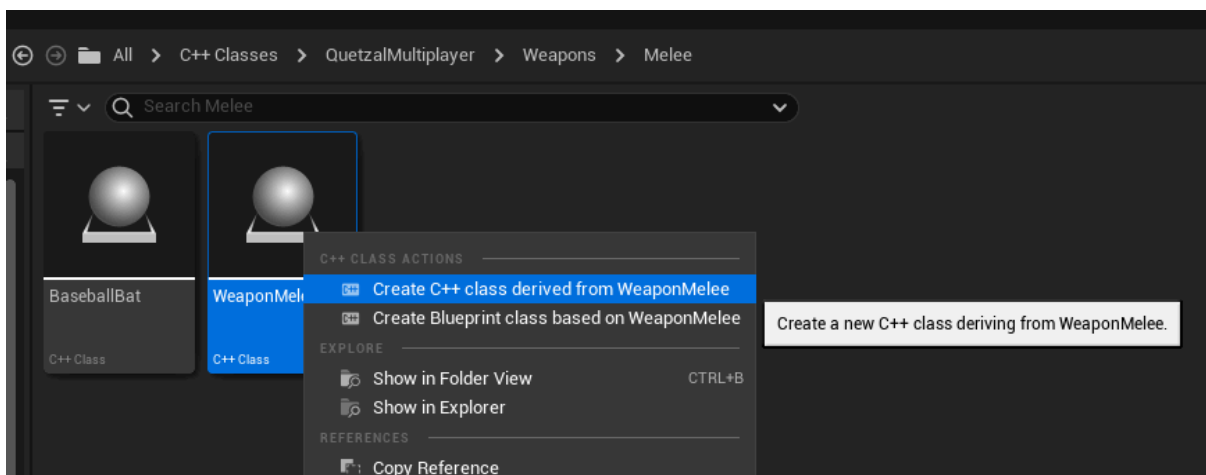
6. Use Case View

6.1 Adding New Weapons

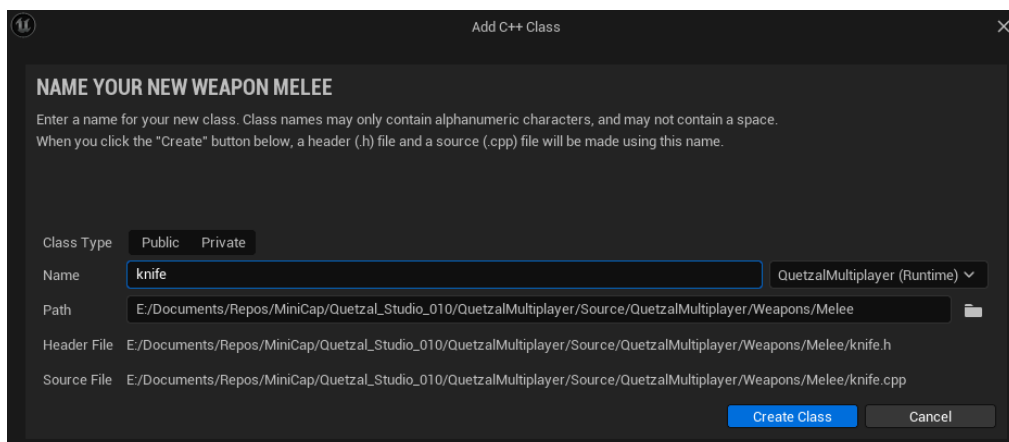
1. In Unreal, navigate to the melee c++ folder



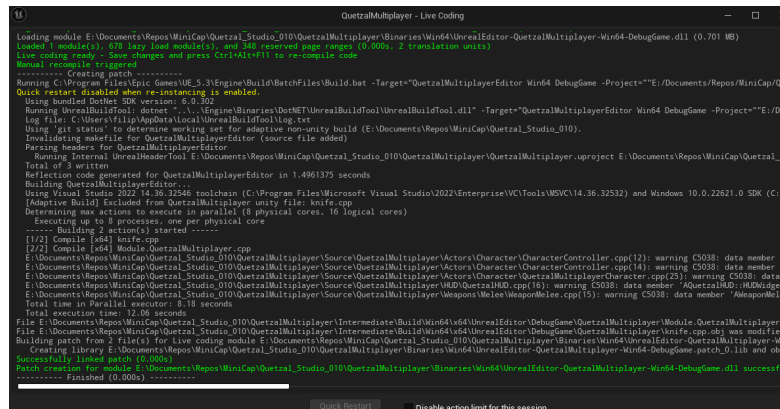
2. Right-Click WeaponMelee class and click on Create C++ derived from WeaponMelee



3. Give the weapon a name

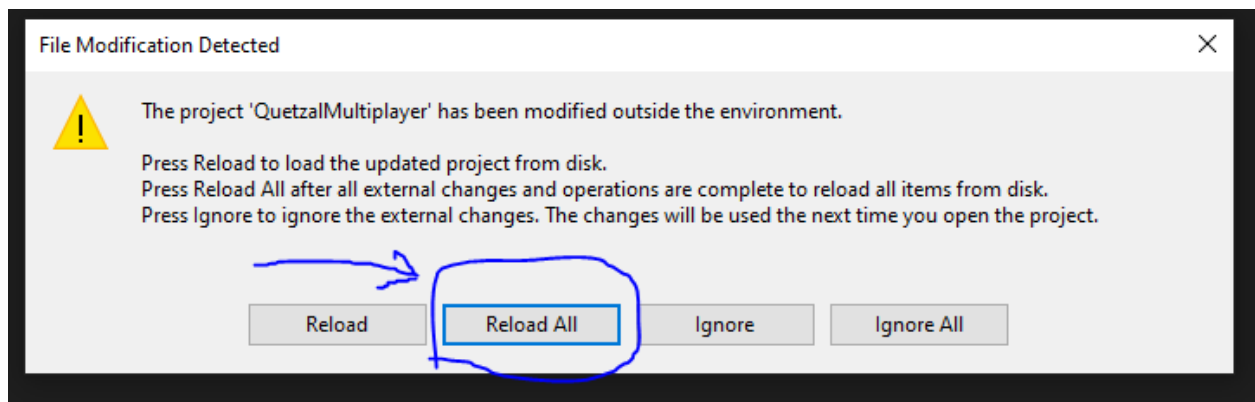


- Wait for Unreal to finish doing its thing... Once Successful reload all in Visual Studio.

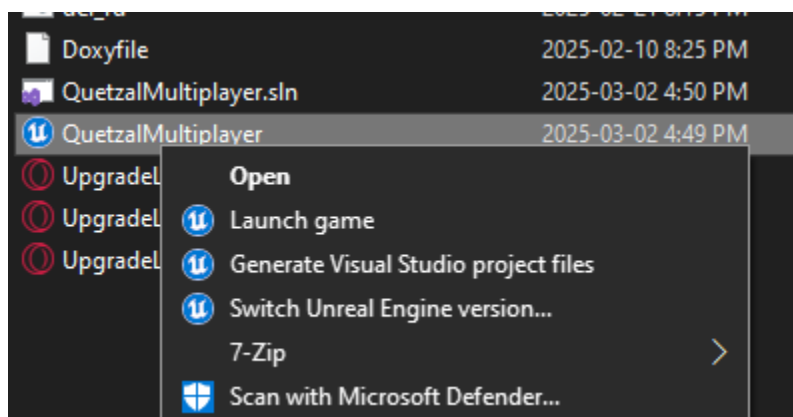


```
QuetzalMultiplayer - Live Coding
Loading module E:\Documents\Repos\MiniCap\Quetzal_Studio_010\QuetzalMultiplayer\Binaries\Win64\UnrealEditor-QuetzalMultiplayer-Win64-DebugGame.dll (0.701 MB)
Loaded module E:\Program Files\Epic Games\UE_5.3\Engine\Build\BatchFiles\Build.bat -Target="QuetzalMultiplayerEditor Win64 DebugGame -Project="E:\Documents\Repos\MiniCap\Q
Live coding ready - Save changes and press Ctrl+Alt+F11 to re-compile code
Manual recompile required

----- Creating patch -----
Running C:\Program Files\Epic Games\UE_5.3\Engine\Build\BatchFiles\Build.bat -Target="QuetzalMultiplayerEditor Win64 DebugGame -Project="E:\Documents\Repos\MiniCap\Q
Quick Restart disabled when re-instantiating is enabled
Using bundled Docker SDK version: 6.0.302
Running UnrealBuildTool -deterministic -v -x -EngineBinaries\Out\UE4\UnrealBuildTool\UnrealBuildTool.dll -Target="QuetzalMultiplayerEditor Win64 DebugGame -Project="E:\D
Log file: C:\Users\Vilip\AppData\Local\UnrealBuildTool\log.txt
Using 'git status' to determine working set for adaptive non-unity build (E:\Documents\Repos\MiniCap\Quetzal_Studio_010).
Invalidating makefile for QuetzalMultiplayerEditor (source file added)
Parsing headers for QuetzalMultiplayerEditor
Running Internal UnrealHeaderTool E:\Documents\Repos\MiniCap\Quetzal_Studio_010\QuetzalMultiplayer\QuetzalMultiplayer.uproject E:\Documents\Repos\MiniCap\Quetzal_
Total of 3 written
Reflection code generated for QuetzalMultiplayerEditor in 1.4961375 seconds
Building QuetzalMultiplayerEditor...
Using Visual Studio 2022 14.36.32546 toolchain (C:\Program Files\Microsoft Visual Studio\2022\Enterprise\VC\Tools\MSVC\14.36.32532) and Windows 10.0.22621.0 SDK (C:
(Adaptive Build) Excluded from QuetzalMultiplayer unity file: knife.cpp
Determining max actions to execute in parallel (8 physical cores, 16 logical cores)
----- Building 2 action(s) started -----
[1/2] Compile [64] knife.cpp
[2/2] Compile [64] Module.QuetzalMultiplayer.cpp
File E:\Documents\Repos\MiniCap\Quetzal_Studio_010\QuetzalMultiplayer\Source\QuetzalMultiplayer\Actors\Character\CharacterController.cpp(12): warning C5038: data member
E:\Documents\Repos\MiniCap\Quetzal_Studio_010\QuetzalMultiplayer\Source\QuetzalMultiplayer\Actors\Character\CharacterController.cpp(14): warning C5038: data member
E:\Documents\Repos\MiniCap\Quetzal_Studio_010\QuetzalMultiplayer\Source\QuetzalMultiplayer\Actors\Character\QuetzalMultiplayerCharacter.cpp(251): warning C5038: data
E:\Documents\Repos\MiniCap\Quetzal_Studio_010\QuetzalMultiplayer\Source\QuetzalMultiplayer\HUD\QuetzalHUD.cpp(16): warning C5038: data member 'AdrenalineHUD::HUDWidget
E:\Documents\Repos\MiniCap\Quetzal_Studio_010\QuetzalMultiplayer\Source\QuetzalMultiplayer\Weapons\Melee\WeaponMelee.cpp(15): warning C5038: data member 'AWeaponMele
Total time in Parallel executor: 0.11 seconds
Total execution time: 12.08 seconds
File E:\Documents\Repos\MiniCap\Quetzal_Studio_010\QuetzalMultiplayer\Intermediate\Build\Win64\UE4\UnrealEditor\DebugGame\QuetzalMultiplayer\Module.QuetzalMultiplayer
File E:\Documents\Repos\MiniCap\Quetzal_Studio_010\QuetzalMultiplayer\Intermediate\Build\Win64\UE4\UnrealEditor\DebugGame\QuetzalMultiplayer\Knife.cpp.obj was modify
Building patch from 2 file(s) for Live coding module E:\Documents\Repos\MiniCap\Quetzal_Studio_010\QuetzalMultiplayer\Binaries\Win64\UnrealEditor-QuetzalMultiplayer-W
Creating library E:\Documents\Repos\MiniCap\Quetzal_Studio_010\QuetzalMultiplayer\Binaries\Win64\UnrealEditor-QuetzalMultiplayer-Win64-DebugGame.patch.0.lib and ob
Successfully linked patch (0.000s)
Patch revision for module E:\Documents\Repos\MiniCap\Quetzal_Studio_010\QuetzalMultiplayer\Binaries\Win64\UnrealEditor-QuetzalMultiplayer-Win64-DebugGame.dll success
----- Finished (0.000s) -----
```



Tip: if you don't see the new cpp and h file, Generate Visual Studio project files again.



5. In the header and cpp file, create the default constructor.

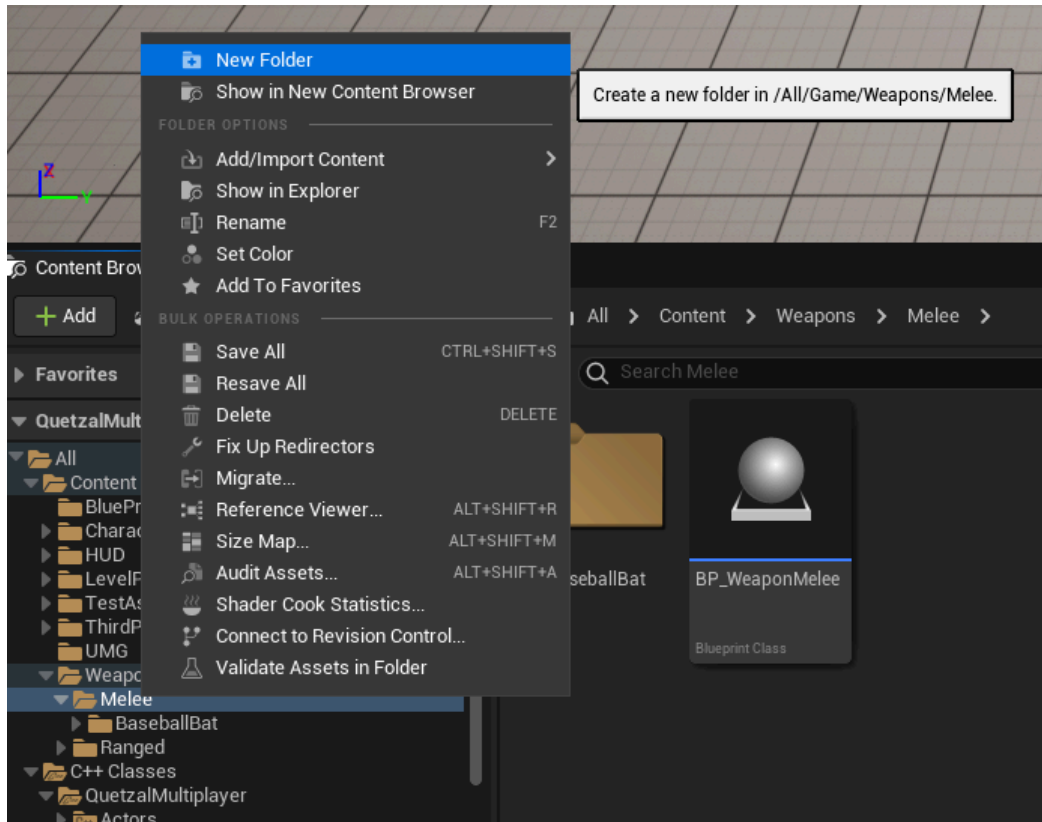
```
1 // Fill out your copyright notice in the Description page of Project Settings.
2
3 #pragma once
4
5 #include "CoreMinimal.h"
6 #include "Weapons/Melee/WeaponMelee.h"
7 #include "knife.generated.h"
8
9 /**
10  *
11  */
12 UCLASS()
13 class QUETZALMULTIPLAYER_API Aknife : public AWeaponMelee
14 {
15     GENERATED_BODY()
16
17 public:
18     Aknife();
19 };
20
21
```

6. Change the damage values for the weapon in the constructor.

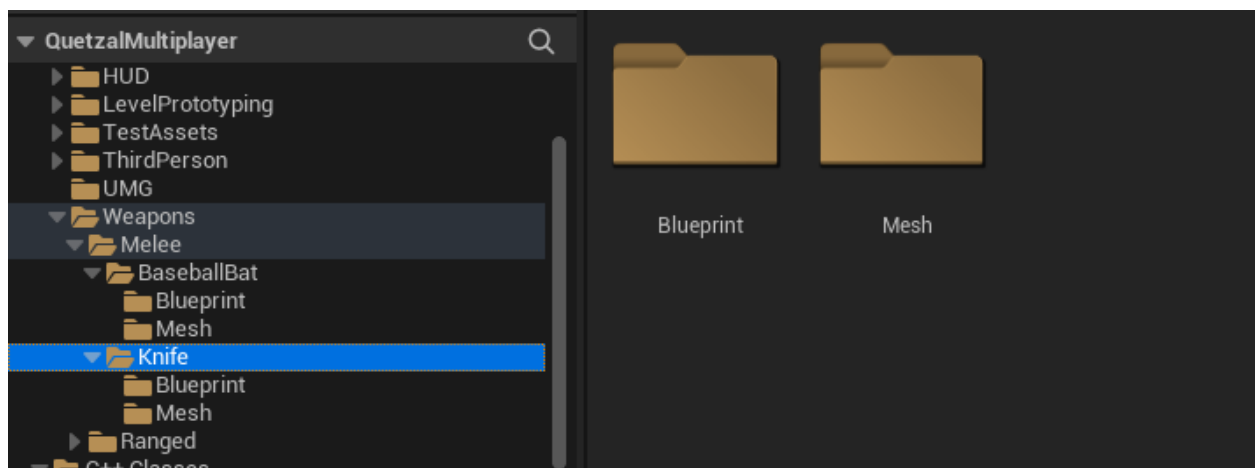
```
1 // Fill out your copyright notice in the Description page of Project Settings.
2
3
4 #include "Weapons/Melee/knife.h"
5
6 Aknife::Aknife()
7 {
8
9     lightDamage = 2;
10    heavyDamage = 4;
11    lightStaggerDamage = 1;
12    heavyStaggerDamage = 3;
13
14    impactForce = 10.0f;
15 }
16
```

7. Add any special feature for the weapon if needed.

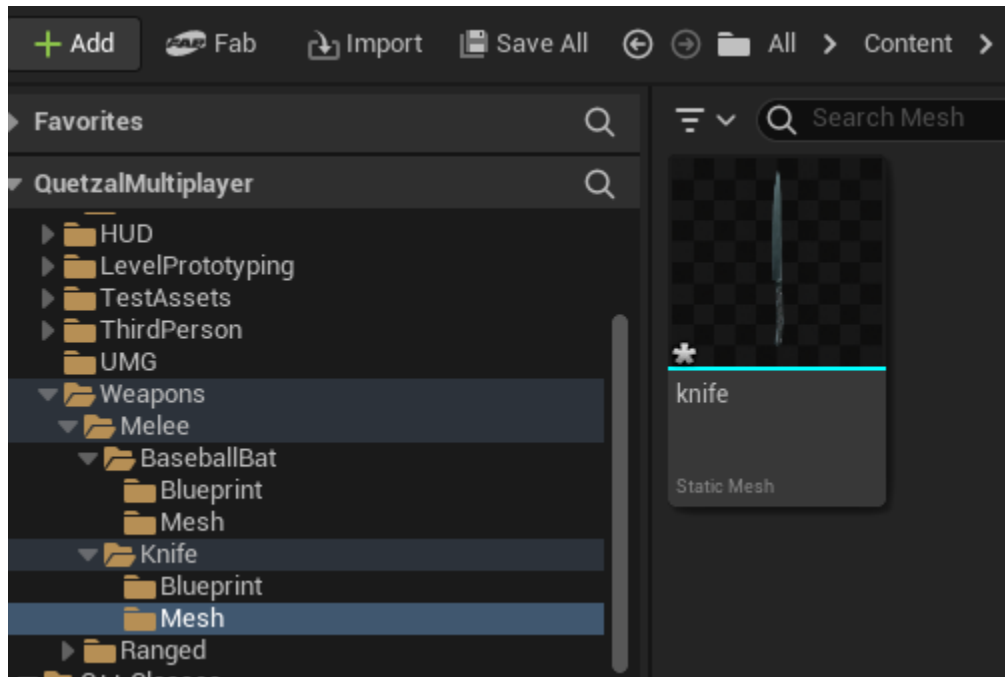
8. Back in Unreal, Create a new folder in the melee folder and give it the weapon name



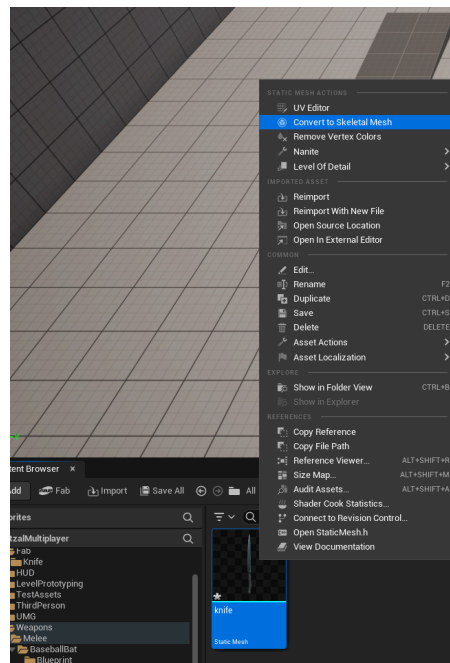
9. In the newly created folder, create a Blueprint and a Mesh folder.



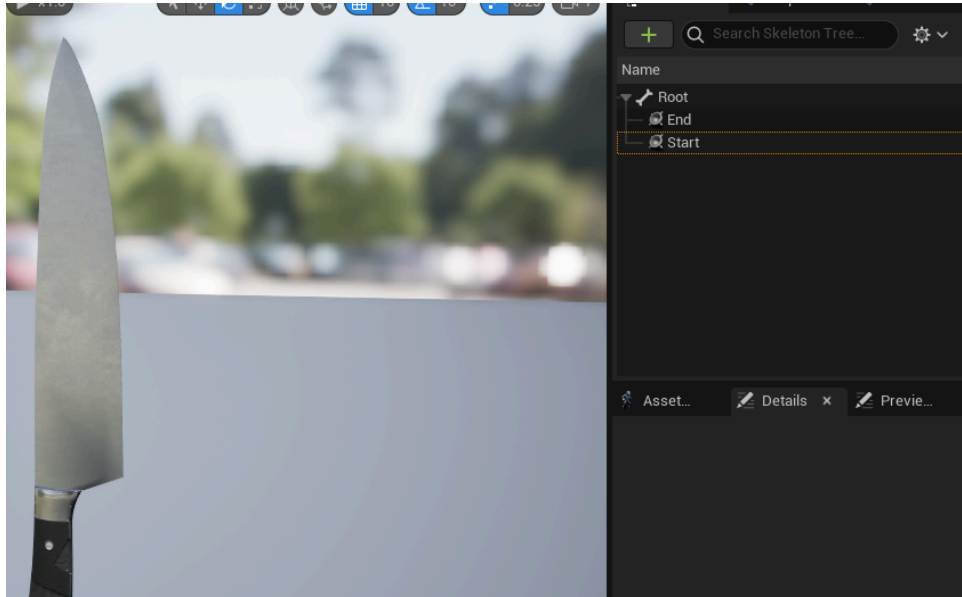
10. Put the mesh of the new weapon in the mesh folder.



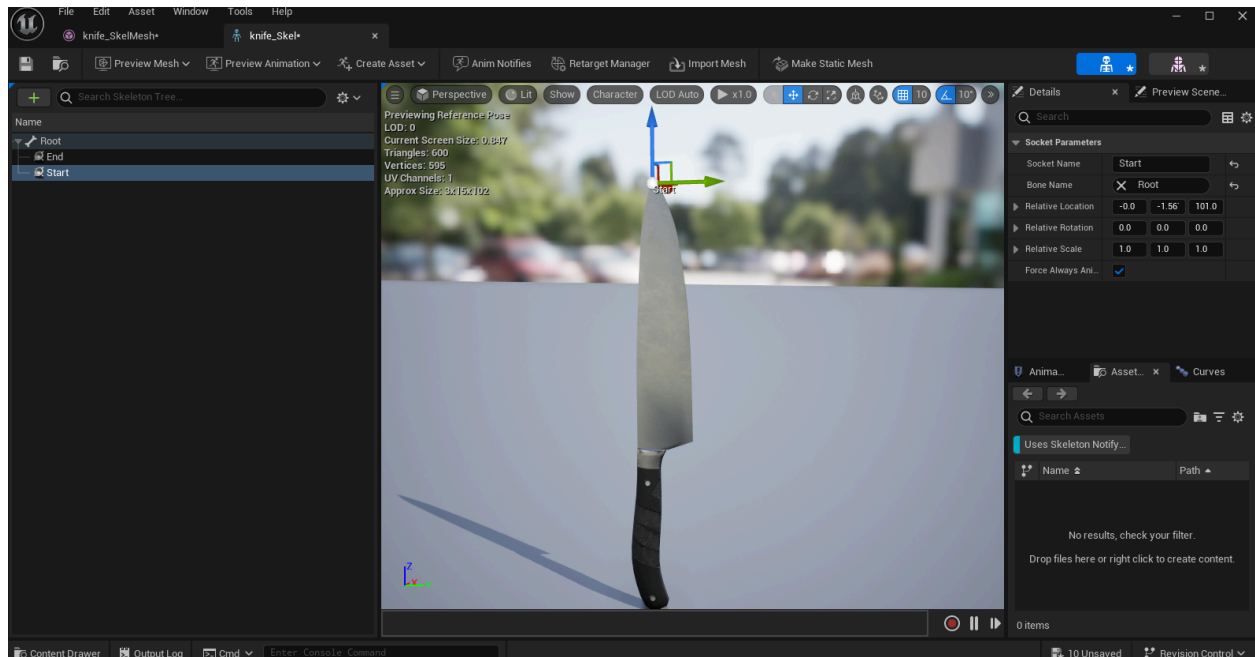
Note: Convert Static Mesh into Skeletal Mesh



11. Double click the skeletal mesh, click on the root socket, click the green plus symbol then add 2 Sockets. One Called End the other Start.



12. Next click on the Skeleton button top right, then click on the Start socket and drag it to the tip of the weapon.

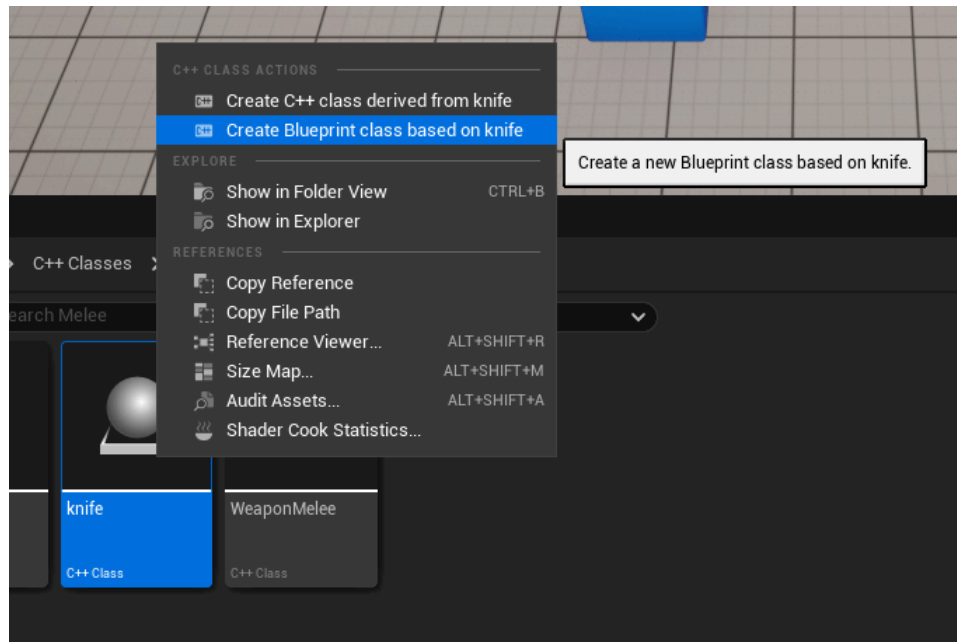


13. Now drag the End socket to the handle of the weapon. Save and close when done.

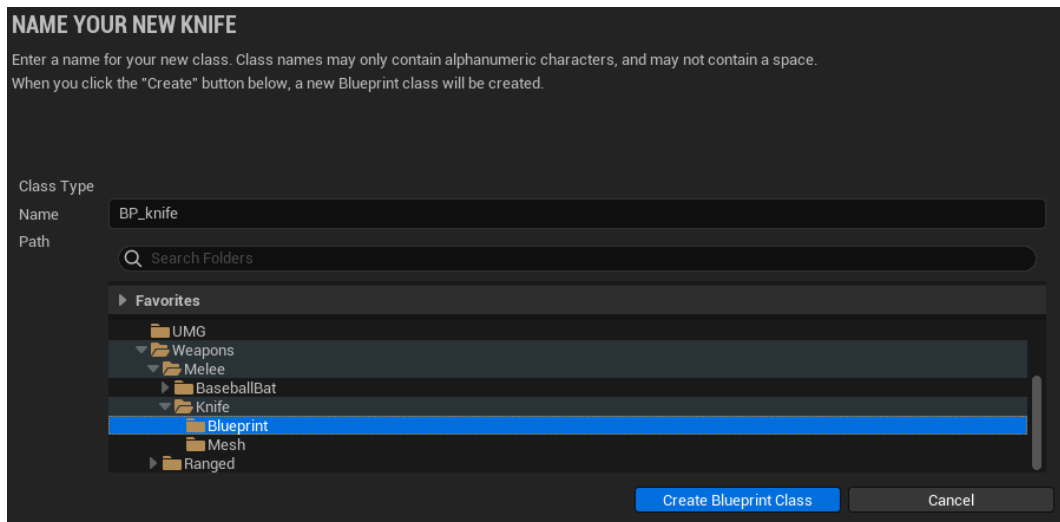
Note: The End socket is where it will attach to the character.



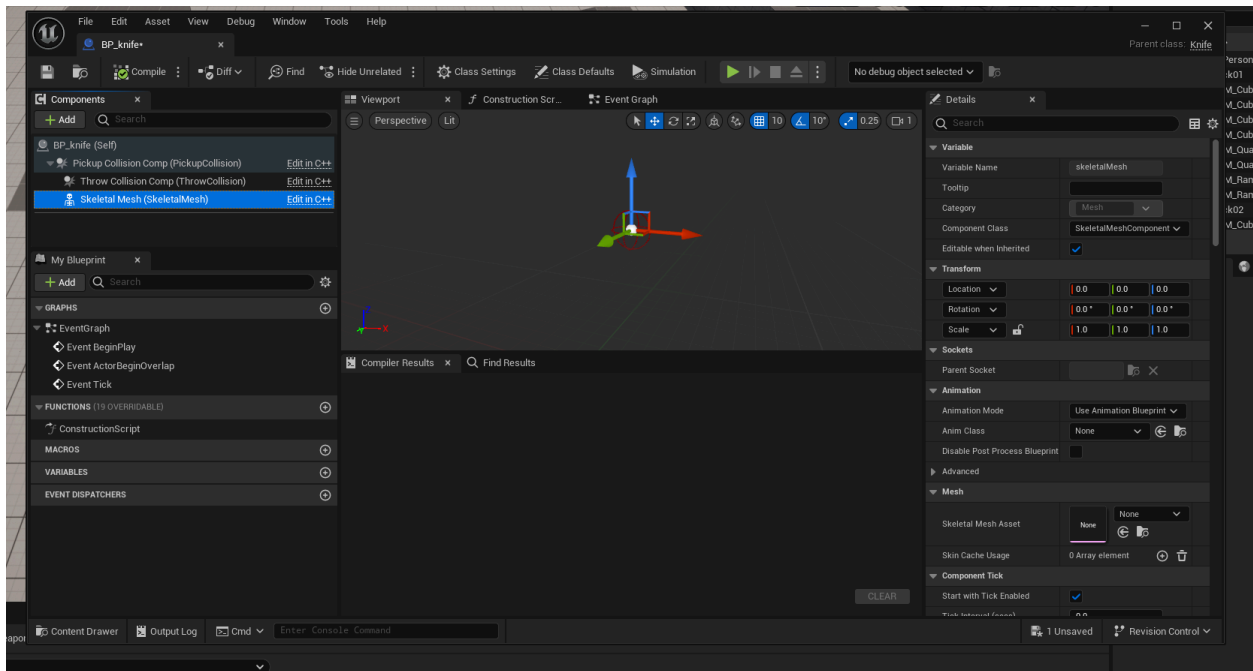
14. Navigate to the cpp file of the new weapon, create a Blueprint of the new weapon.



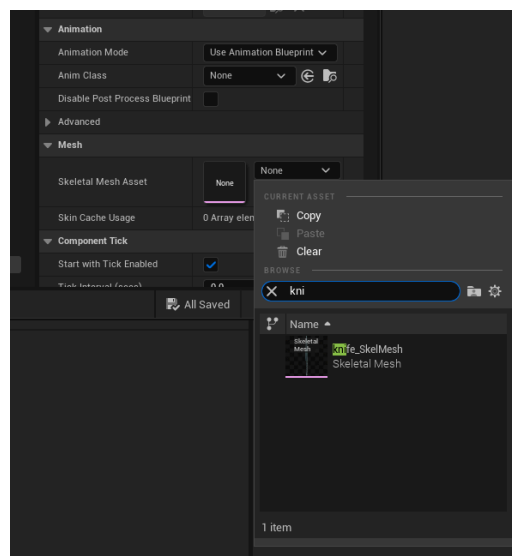
15. Give the blueprint a name, BP_WeaponName. And put it in the newly created blueprint folder.



16. A blueprint window will popup, click on SkeletalMesh in the left window.



17. Under Mesh- Skeletal mesh on the right side, search for the new skeletal mesh weapon created earlier. Once Selected, Save and close the window.



18. Congrats! You've created a new weapon for the game. Drag and drop your blueprint of the new weapon into the map and test it out.