# Setting Up Inputs

How to set up user input

Choose your implementation method

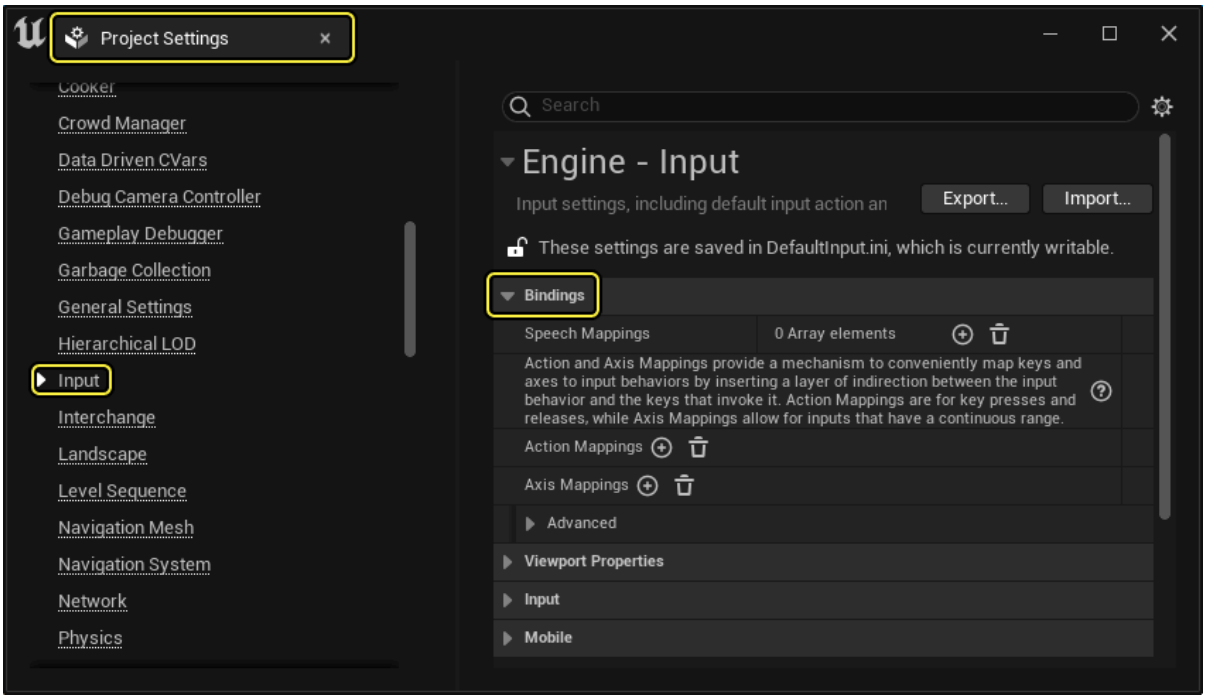
Blueprints C++

In this tutorial, you will create a Character and set it up to receive input, then assign the character to a GameMode so that it is the default pawn during gameplay. After you create your Character, you will define how it reacts to Player Input.

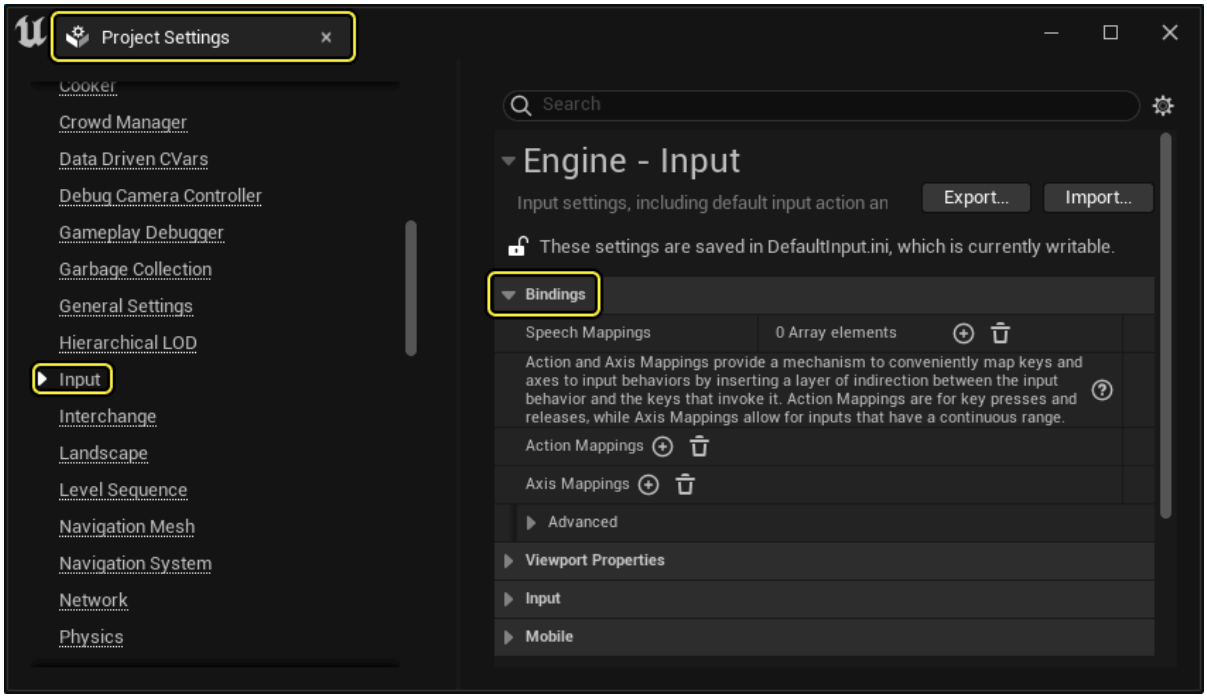
Unreal provides for more complex Input mappings for a variety of project types. Refer to Enhanced Input for additional documentation.

# Project Setup

1. Create a new **Games** > **Blank > Blueprint** project named "**SettingUpInput**".
2. From the Editor, navigate to **Edit** > **Project Settings** > **Input > Bindings**.




1. Create a new **Games** > **Blank > C++** project named "**SettingUpInput**".
2. In the **Editor**, navigate to **Edit** > **Project Settings** > **Input > Bindings**.

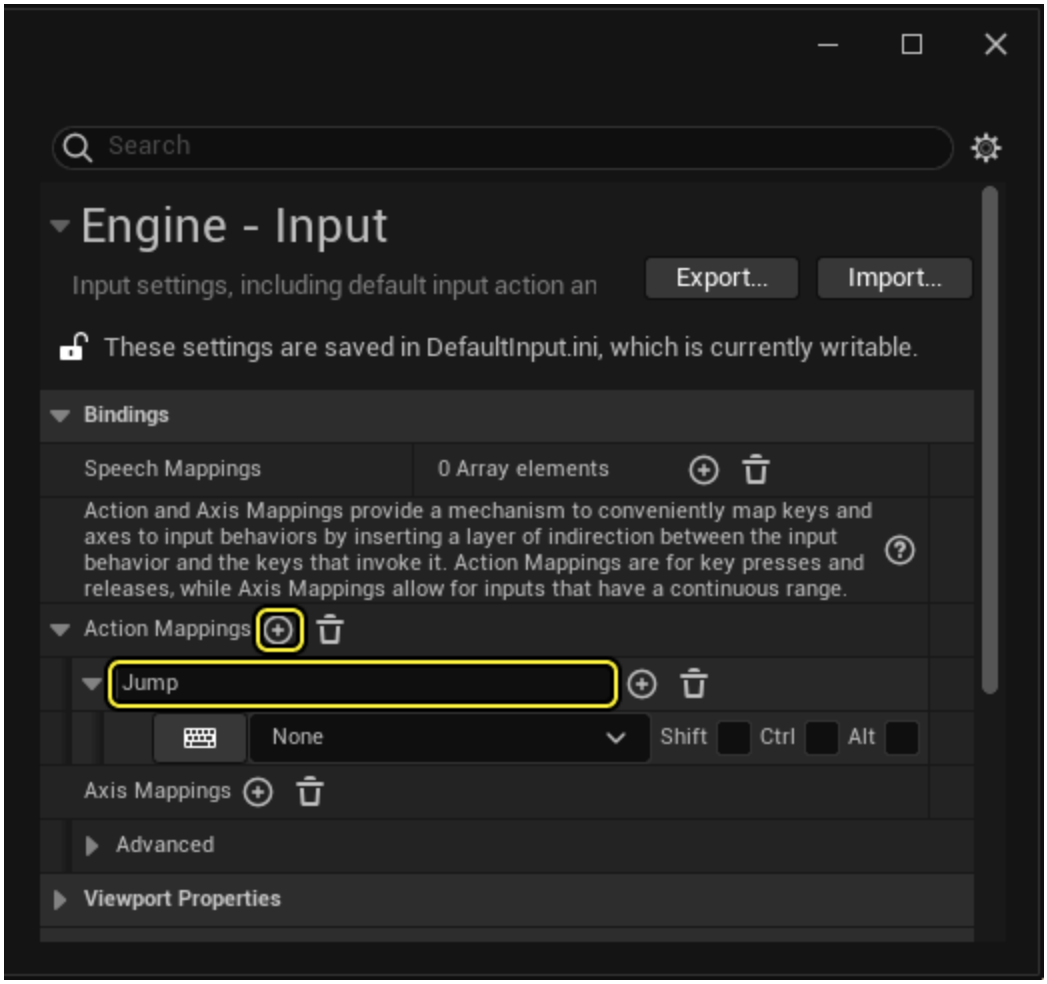


# Action and Axis Mapping Setup

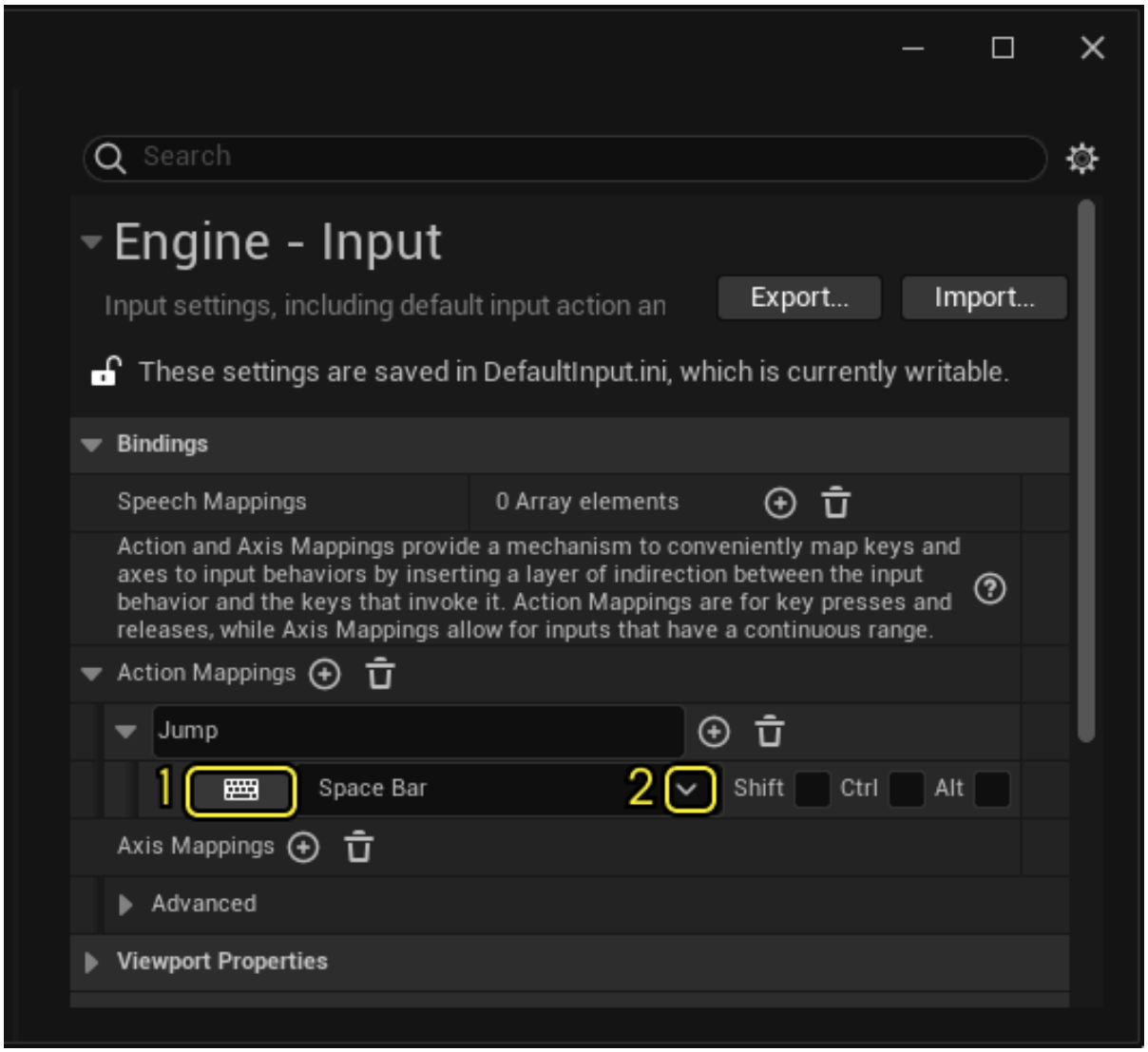
Defining Input is done through user-defined **Bindings** of **Action** and **Axis Mappings**. Both mappings provide a mechanism to conveniently map keys and axes to input behaviors by inserting a layer of indirection between the input behavior and the keys that invoke it.

Action Mappings are for key presses and releases, while Axis Mappings allow for inputs that have a continuous range. Once you have defined your Mappings, you can then bind them to behaviors in Blueprints or C++.

1. Click **Add**(**+**) next to **Action Mappings** to create a new Action named **Jump.**



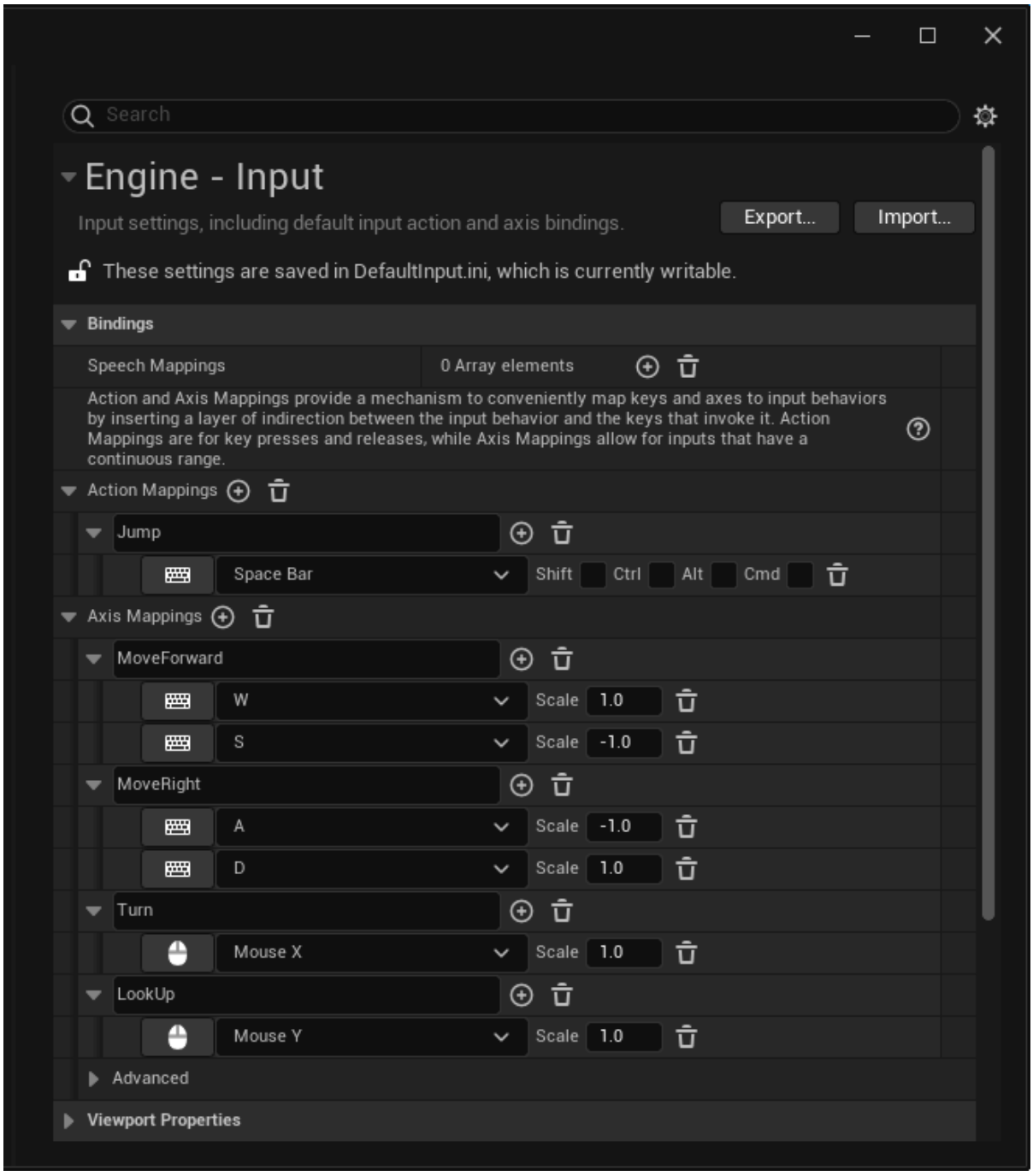

2. From either the drop down arrow(1) or the **Select Key Value** button(2), search for and select the **Space Bar** key value.




3. Navigate to the **Axis mappings** and click **Add**(**+**) to create the following **Axis Mapping** names, **Key** values, and **Scale** values:

| Axis Mapping Name | Key Value | Scale Value |
|---|---|---|
| MoveForward | W | 1.0 |
| | S | -1.0 |
| MoveRight | A | -1.0 |
| | D | 1.0 |
| Turn | Mouse X | 1.0 |

| Axis Mapping Name | Key Value | Scale Value |
| --- | --- | --- |
| LookUp | Mouse Y | -1.0 |



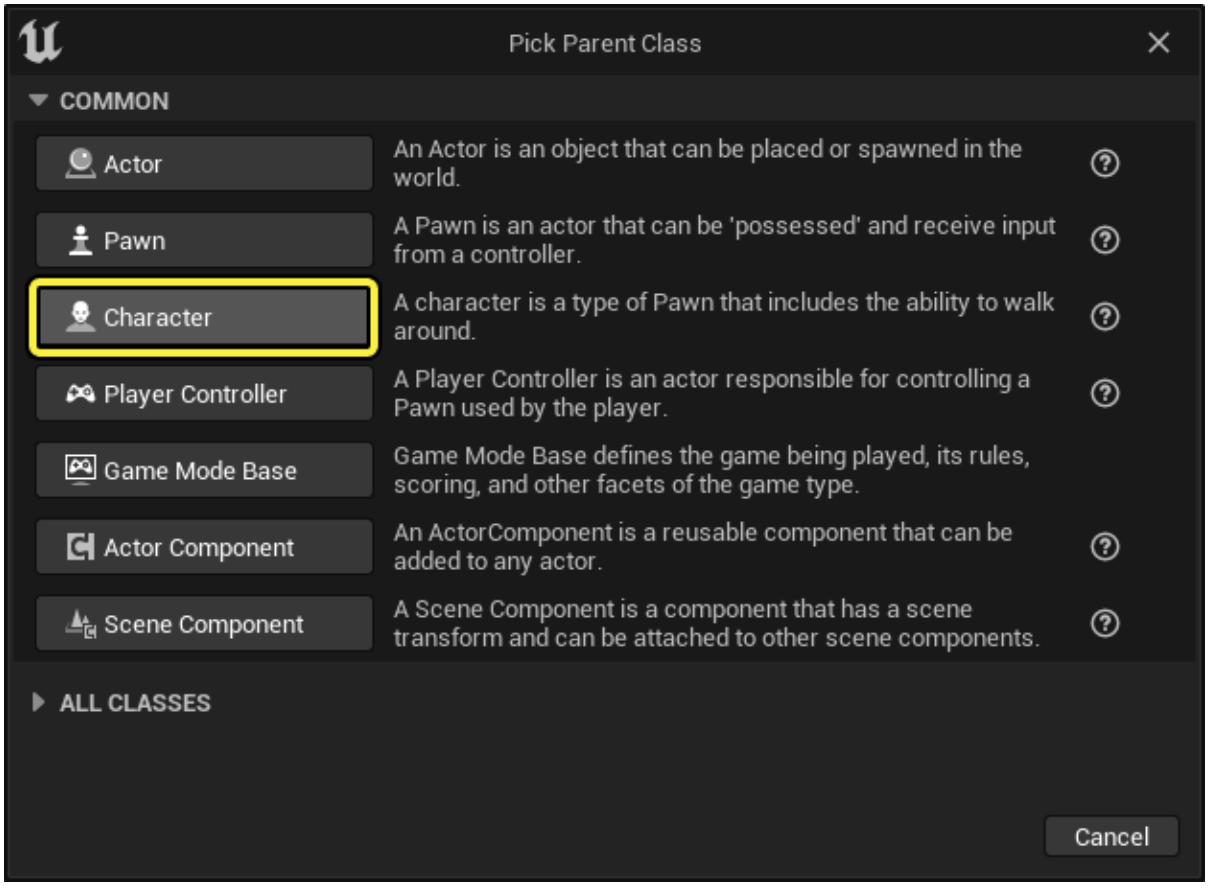

# Creating the Example Character

# Creating the Example Character

A Character is a special type of **Pawn** that has the ability to walk around. Characters extend from the Pawn class, and inherit similar properties such as physical representation of a player or AI entity within the world.

> Refer to the Pawn and Input pages for additional documentation.

1. From the **Content Drawer**, Click **Add**(**+**), then from the **Pick Parent Class** menu choose **Character** as your parent class.

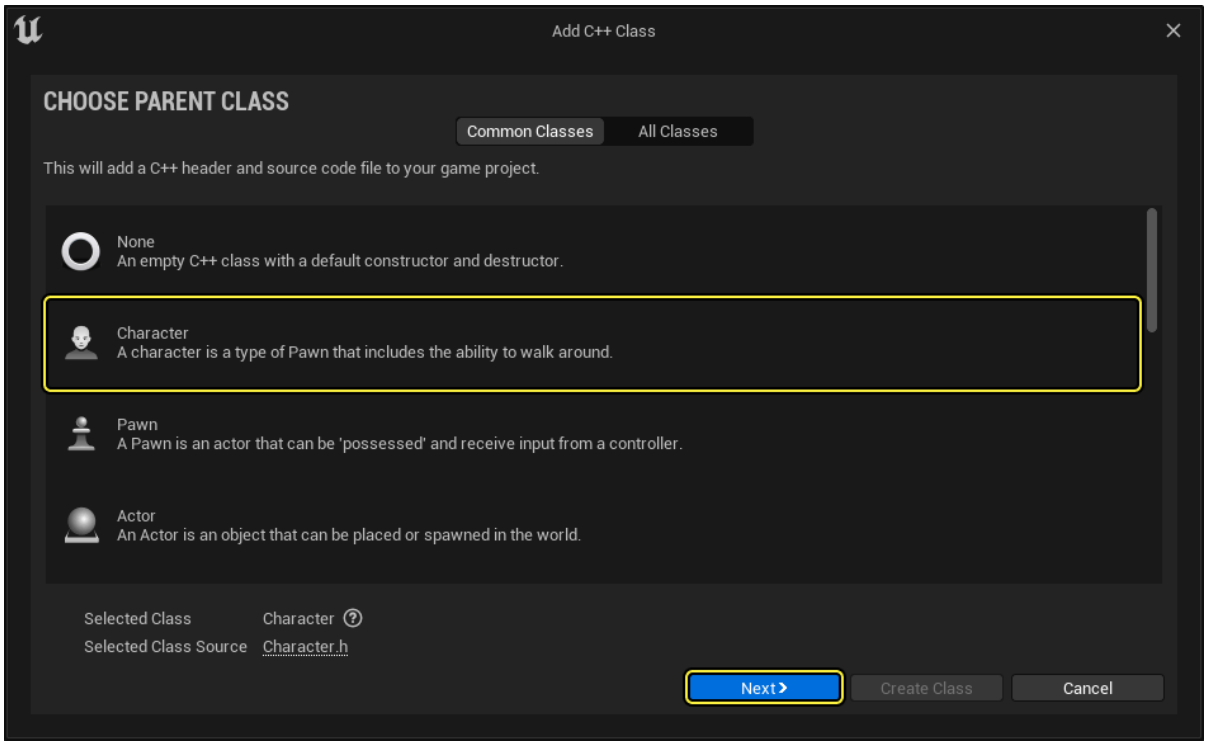



2. Name your Blueprint **BP_ExampleCharacter**, and double-click it to open its class defaults.

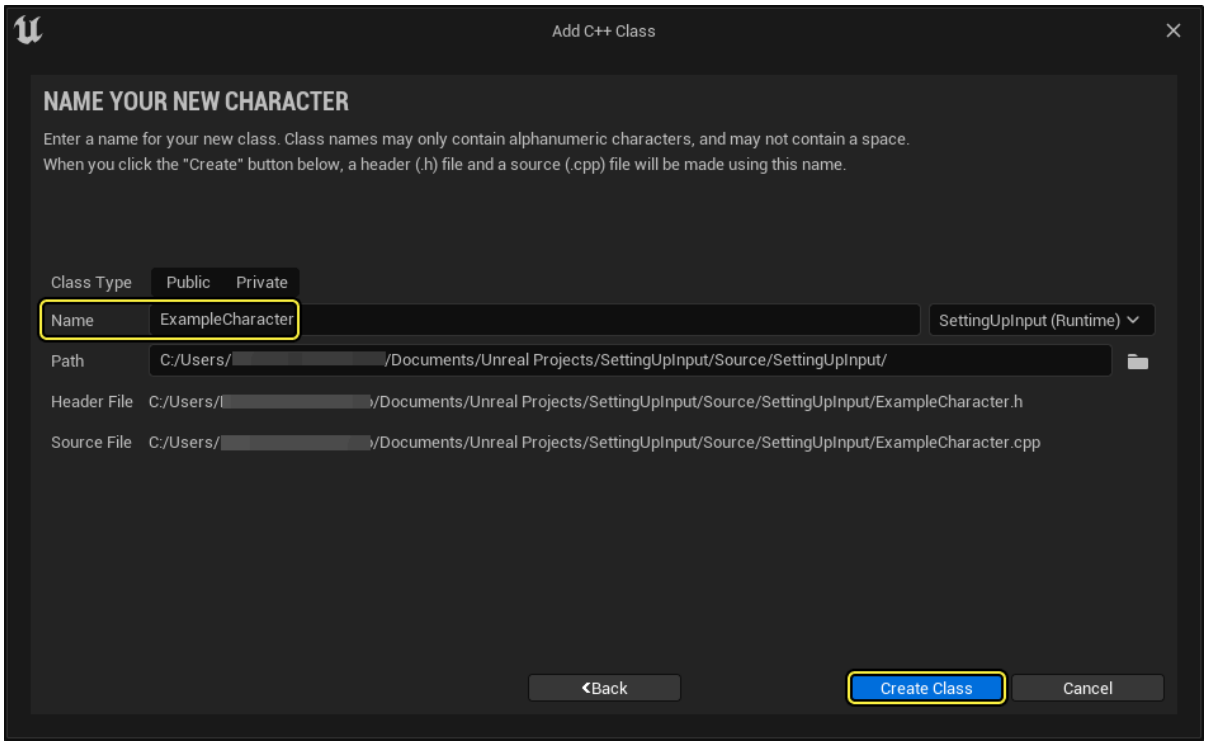
A Character is a special type of **Pawn** that has the ability to walk around. Characters extend from the Pawn class, and inherit similar properties such as physical representation of a player or AI entity within the world.

> Refer to the Pawn and Input pages for additional documentation.

1. From the **Content Drawer**, navigate to the **C++ classes** folder, right-click and select **New C++ Class**, then choose **Character** as your parent class.




2. Name your character class "**ExampleCharacter**", then click **Create Class**.

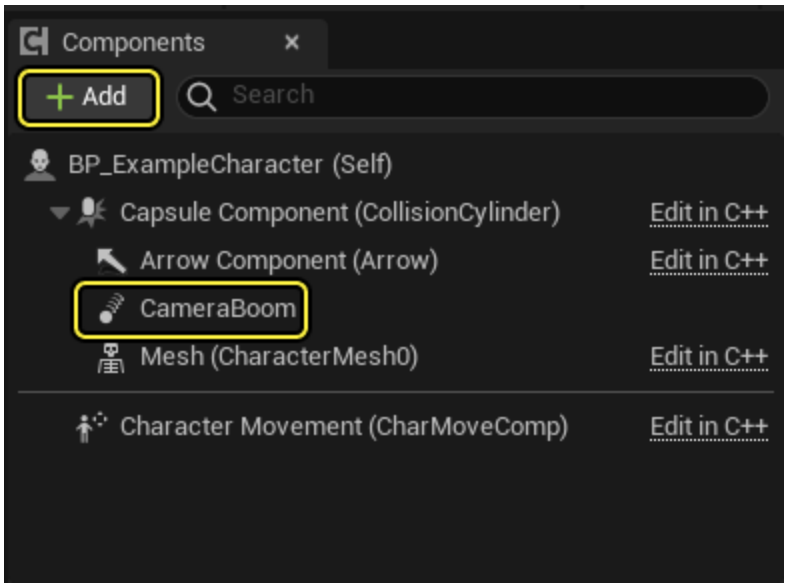



# Creating the SpringArm and Camera Components

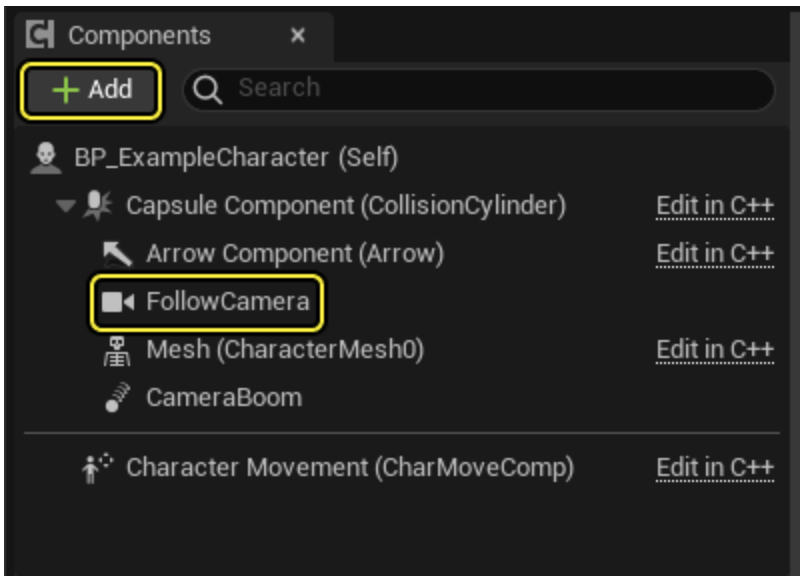
## Creating the SpringArm and Camera Components

When the **Camera** and **SpringArm Components** are used together, they provide functionality for a third-person perspective that can dynamically adjust to your game world. The camera component contains a camera that represents the player's point of view or how the player sees the world. The SpringArm component is used as a "camera boom" to keep the camera for a player from colliding into the world.

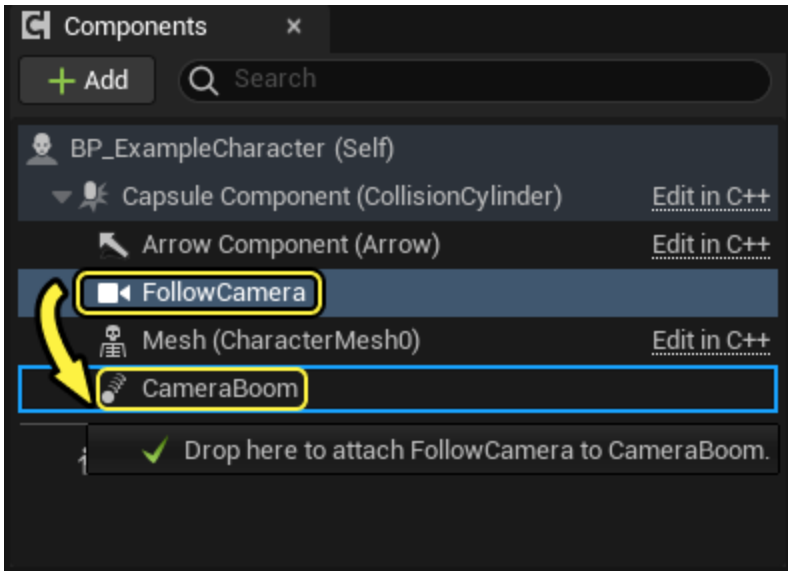
1. In the **Components** tab, click **Add(+)**, then from the drop down search for and select **Spring Arm**. Rename your Spring Arm component **CameraBoom**.



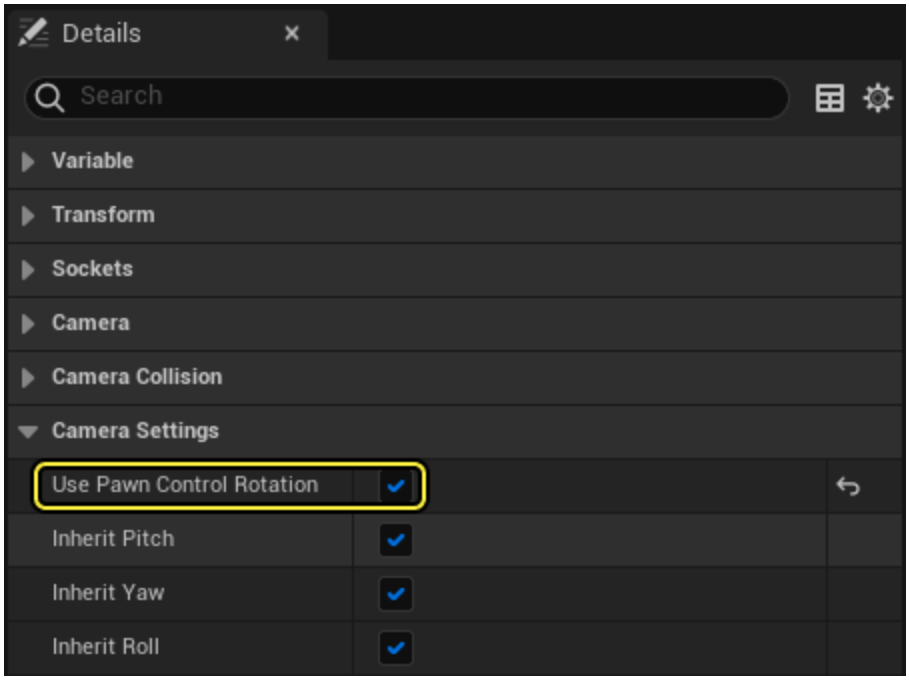

2. Repeat step 1, but search for and select the **Camera**. Rename your Camera component **FollowCamera**.




3. From the **Components** tab, drag your **FollowCamera** onto the **CameraBoom** to attach it.

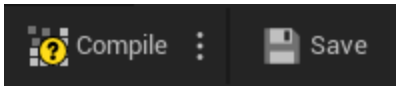



4. Select the CameraBoom in the Components tab, then navigate to the **Details** > **Camera Settings** and click the checkbox to enable the **Use Pawn Control Rotation** variable. When enabled, the Camera parent uses the view / control rotation of the pawn (**ExampleCharacter**).




> For additional Spring arm and Camera settings that you could use for your Character's perspective. Refer to the Using Cameras documentation

5. **Compile** and **Save**.



When the **Camera** and **SpringArm Components** are used together, they provide functionality for a third-person perspective that can dynamically adjust to your game world. The camera component contains a camera that represents the player's point of view or how the player sees the world. The SpringArm component is used as a "camera boom" to keep the camera for a player from colliding into the world.

1. In your code editor, navigate to ExampleCharacter.h**.** In the **Class defaults,** declare the following classes.

```
protected:
UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Components")
class USpringArmComponent* CameraBoom;

UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Components")
class UCameraComponent* FollowCamera; |
```

Copy full snippet

> UProperty Specifiers are used to provide visibility of the component in the Blueprint Editor.

2. Navigate to your `ExampleCharacter.cpp` file. Add the following libraries to the include line.

```
#include "GameFramework/SpringArmComponent.h"
#include "Camera/CameraComponent.h"
```

Copy full snippet

3. Next, implement the following in the `AExampleCharacter` constructor.

```
AExampleCharacter::AExampleCharacter()
{
//Initialize the Camera Boom
CameraBoom = CreateDefaultSubobject<USpringArmComponent>
(TEXT("CameraBoom"));

//Setup Camera Boom attachment to the Root component of the class
```

```
CameraBoom->SetupAttachment(RootComponent);

//Set the boolean to use the PawnControlRotation to true.
CameraBoom->bUsePawnControlRotation = true;

//Initialize the FollowCamera
FollowCamera = CreateDefaultSubobject<UCameraComponent>
(TEXT("FollowCamera"));

//Set FollowCamera attachment to the Camera Boom
FollowCamera->SetupAttachment(CameraBoom);
}
```

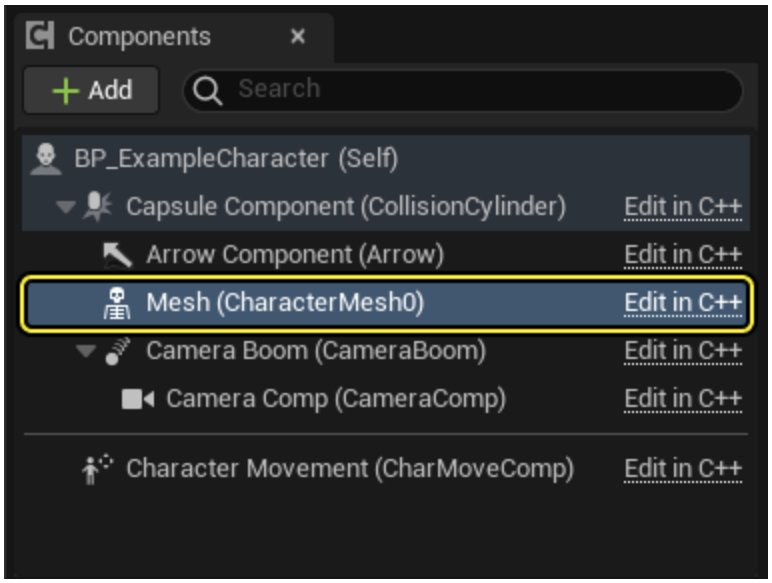
Copy full snippet

> The component calls the FObjectInitializer::CreateDefaultSubobjecttemplate, then uses the SetupAttachment method to attach to a parent Scene Component. When setting the Camera Boom to use the Pawn's control rotation, it uses its parent pawn's rotation instead of its own.

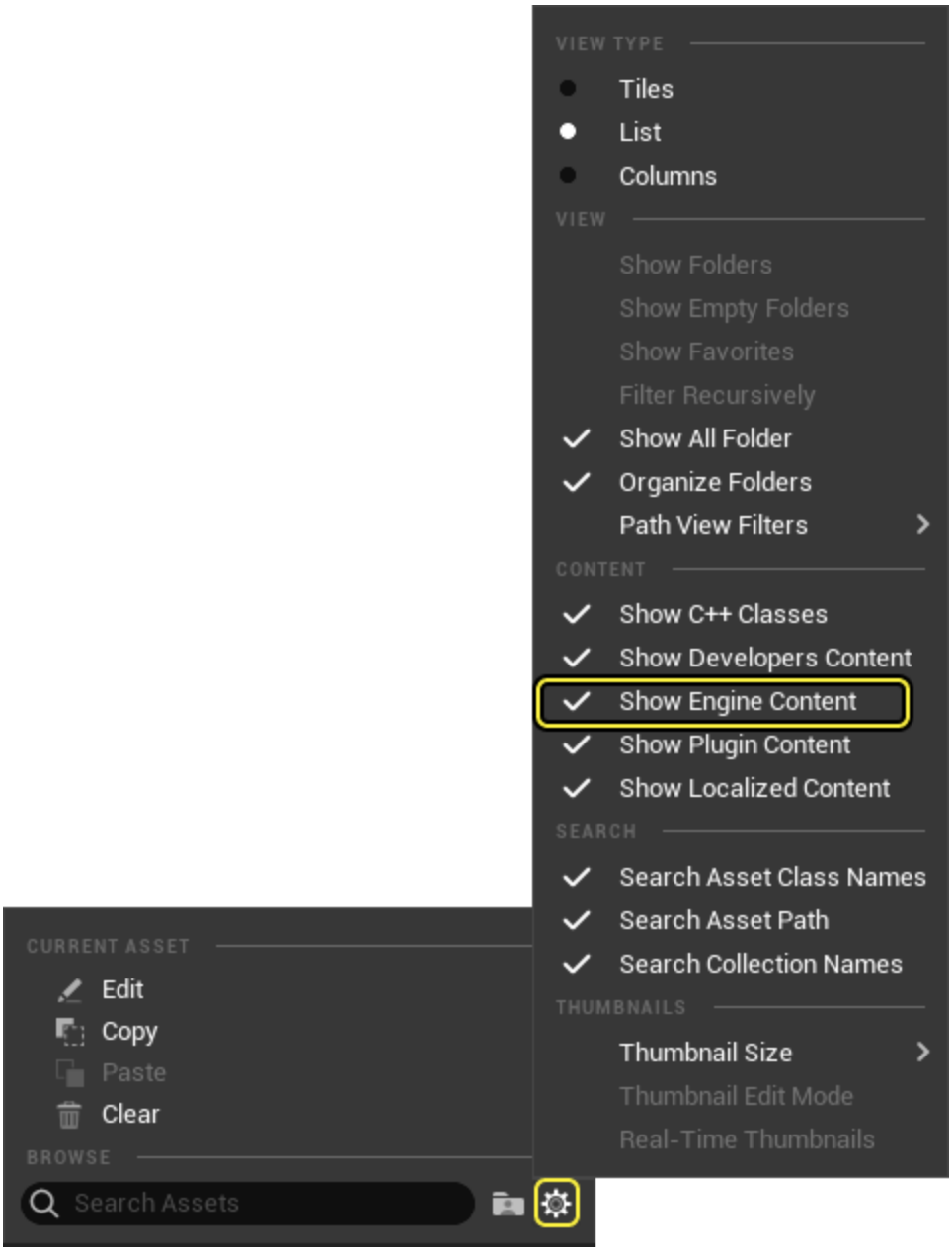
4. Compile your code.

## Setting the Character Mesh

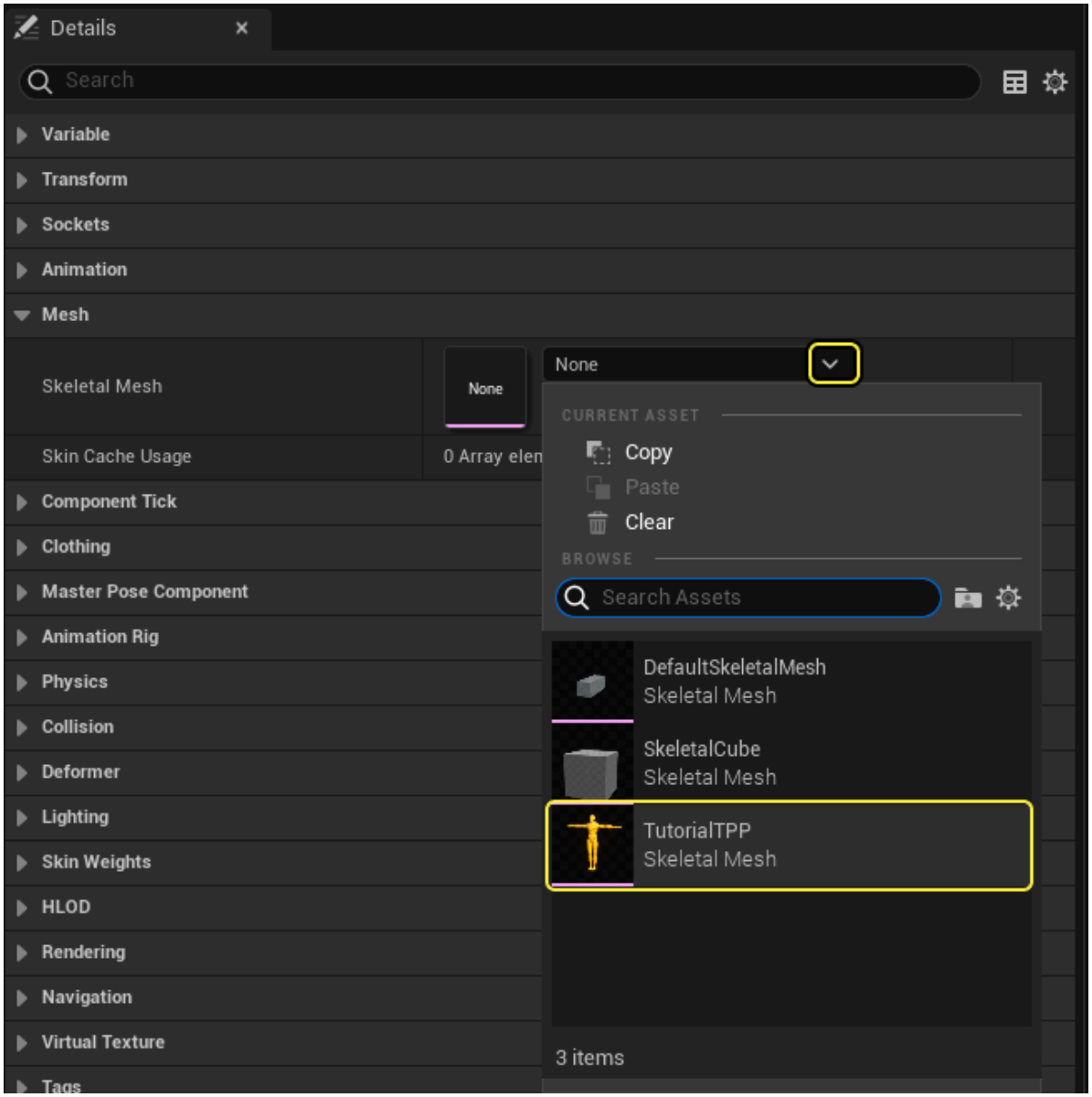
1. In the **Components** panel, select the **Mesh** Skeletal Mesh Component.



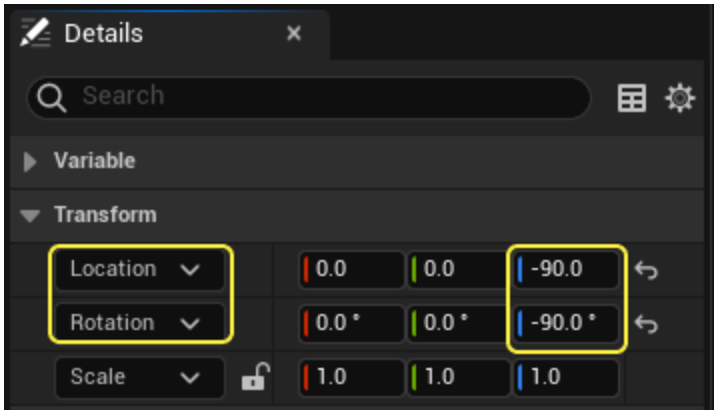

2. Navigate to **Details** > **Mesh** > **Skeletal Mesh** and expand the drop-down menu. Select **Browse Asset Options** > **Content** > **Show Engine Content**.




3. Search for and select the **TutorialTPP** Skeletal Mesh.




4. Navigate to the **Transform** category, then set the **Location** and **Rotation** vector values to (**0.0, 0.0, -90**)




5. **Compile** and **Save**.

Compile Save

# Creating the Action/Axis Functions to your Input Component

1. In your `ExampleCharacter.h` class defaults, declare the following Input functions.

```
protected:

void MoveForward(float AxisValue);

void MoveRight(float AxisValue);
```

Copy full snippet

2. Navigate to your `ExampleCharacter.cpp` and implement your `MoveForward` and `MoveRight` methods.

```
void AExampleCharacter::MoveForward(float AxisValue)
{
if ((Controller != NULL) && (AxisValue != 0.0f))
{
// find out which direction is forward
const FRotator Rotation = Controller->GetControlRotation();
const FRotator YawRotation(0, Rotation.Yaw, 0);

// get forward vector
const FVector Direction =
FRotationMatrix(YawRotation).GetUnitAxis(EAxis::X);
AddMovementInput(Direction, AxisValue);
}

}

void AExampleCharacter::MoveRight(float AxisValue)
{
if ((Controller != NULL) && (AxisValue != 0.0f))
{
// find out which direction is right
const FRotator Rotation = Controller->GetControlRotation();
```

```
const FRotator YawRotation(0, Rotation.Yaw, 0);

// get right vector
const FVector Direction =
FRotationMatrix(YawRotation).GetUnitAxis(EAxis::Y);

// add movement in that direction
AddMovementInput(Direction, AxisValue);
}
}
```

Copy full snippet

3. Navigate to the SetupPlayerInputComponent(UInputComponent* PlayerInputComponent) method, then implement the following code.

```
void AExampleCharacter::SetupPlayerInputComponent(UInputComponent*
PlayerInputComponent)
{
Super::SetupPlayerInputComponent(PlayerInputComponent);

PlayerInputComponent->BindAction("Jump", IE_Pressed, this,
&ACharacter::Jump);
PlayerInputComponent->BindAction("Jump", IE_Released, this,
&ACharacter::StopJumping);

PlayerInputComponent->BindAxis("MoveForward", this,
&AExampleCharacter::MoveForward);
PlayerInputComponent->BindAxis("MoveRight", this,
&AExampleCharacter::MoveRight);
PlayerInputComponent->BindAxis("Turn", this,
&APawn::AddControllerYawInput);
PlayerInputComponent->BindAxis("LookUp", this,
&APawn::AddControllerPitchInput);
}
```

Copy full snippet

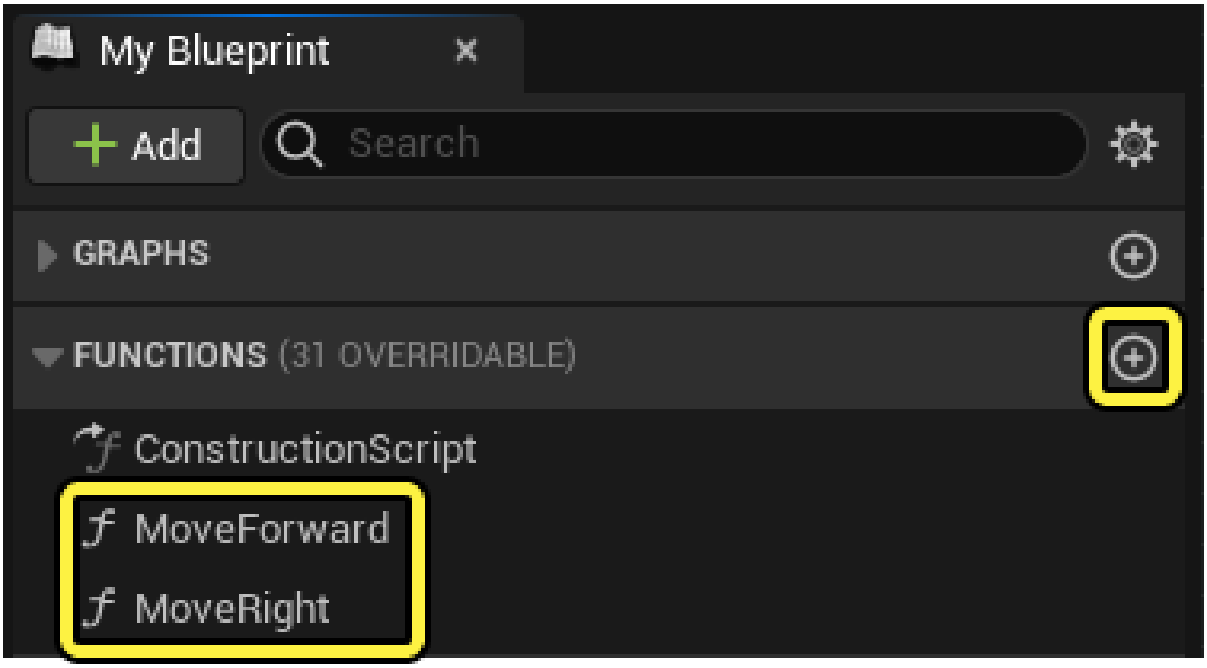
The Player Input Component links the AxisMappings and ActionMappings in your project to game actions. Both the Pawn and Character class contain methods that are inherited and can be used or extended for your custom characters. In our example, we've used

the Pawn's AddControllerYawInput and AddControllerPitchInput functions, and the Character's Jump and StopJumping functions.

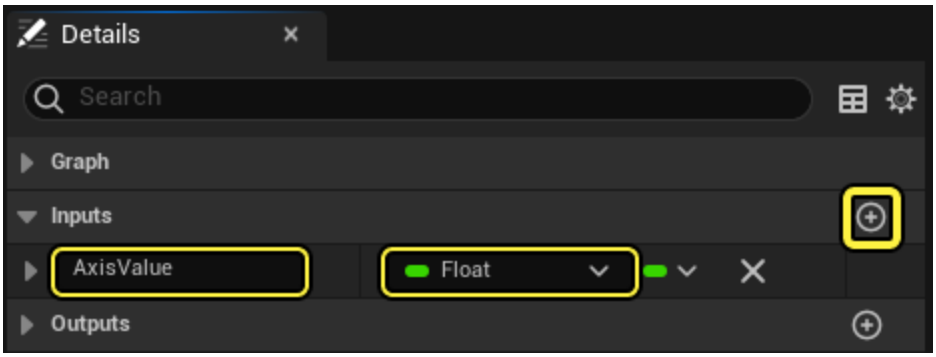
4. Compile your code.

## Creating the Action/Axis Functions to your Input Events

1. Navigate to **My Blueprint** > **Functions**, click **Add**(**+**) to add two new functions. **MoveForward** and **MoveRight**.



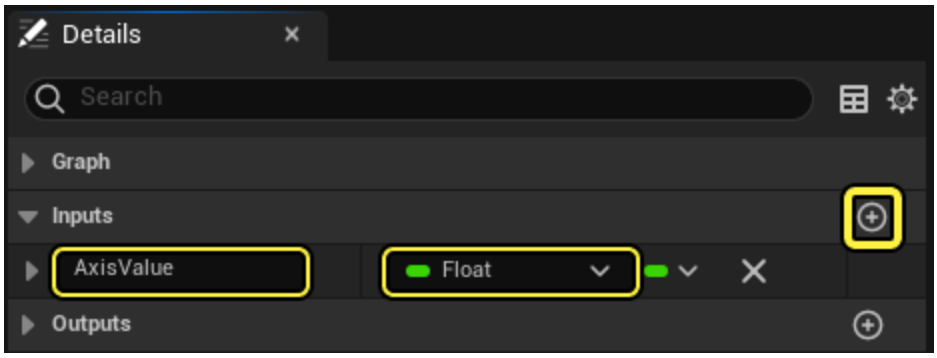

2. Select the **MoveForward** function, navigate to the **Details** > **Inputs**, then click **Add**(**+**) to add a new **float** value input named **AxisValue**.




3. Copy or Implement the logic below for the **MoveForward** function.

Copy code

4. Select the **MoveRight** function, and add a new **float** value as instructed in step 2.




5. Copy or Implement the logic below for the **MoveRight** function.

Copy code

6. Navigate to the **Event Graph**. You need to set up your Axis and Action input events to call their respective functions. Copy or implement the logic below.

Copy code

# Finished Code

## ExampleCharacter.h

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "ExampleCharacter.generated.h"

UCLASS()
class SETTINGUPINPUT_API AExampleCharacter : public ACharacter
{
GENERATED_BODY()

public:
// Sets default values for this character's properties
AExampleCharacter();

protected:
// Called when the game starts or when spawned
virtual void BeginPlay() override;

UPROPERTY(EditDefaultsOnly,BlueprintReadOnly)
class USpringArmComponent* CameraBoom;
```

```
22
23 UPROPERTY(EditDefaultsOnly,BlueprintReadOnly)
24 class UCameraComponent* FollowCamera;
25
26 void MoveForward();
27
28 void MoveRight();
29
30 public:
31 // Called every frame
32 virtual void Tick(float DeltaTime) override;
33
34 // Called to bind functionality to input
35 virtual void SetupPlayerInputComponent(class UInputComponent*
   PlayerInputComponent) override;
36
37 };
```

Copy full snippet

## ExampleCharacter.cpp

```
1 // Sets default values
2 AExampleCharacter::AExampleCharacter()
3 {
4 //Initialize the Camera Boom
5 CameraBoom = CreateDefaultSubobject<USpringArmComponent>
  (TEXT("CameraBoom"));
6
7 //Setup its attachment to the Root component of the class
8 CameraBoom->SetupAttachment(RootComponent);
9
10 //Set the boolean to use the PawnControlRotation to true.
11 CameraBoom->bUsePawnControlRotation = true;
12
13 //Initialize the Camera Comp
14 FollowCamera = CreateDefaultSubobject<UCameraComponent>
  (TEXT("FollowCamera"));
15
16 //Set its attachment to the Camera Boom
```

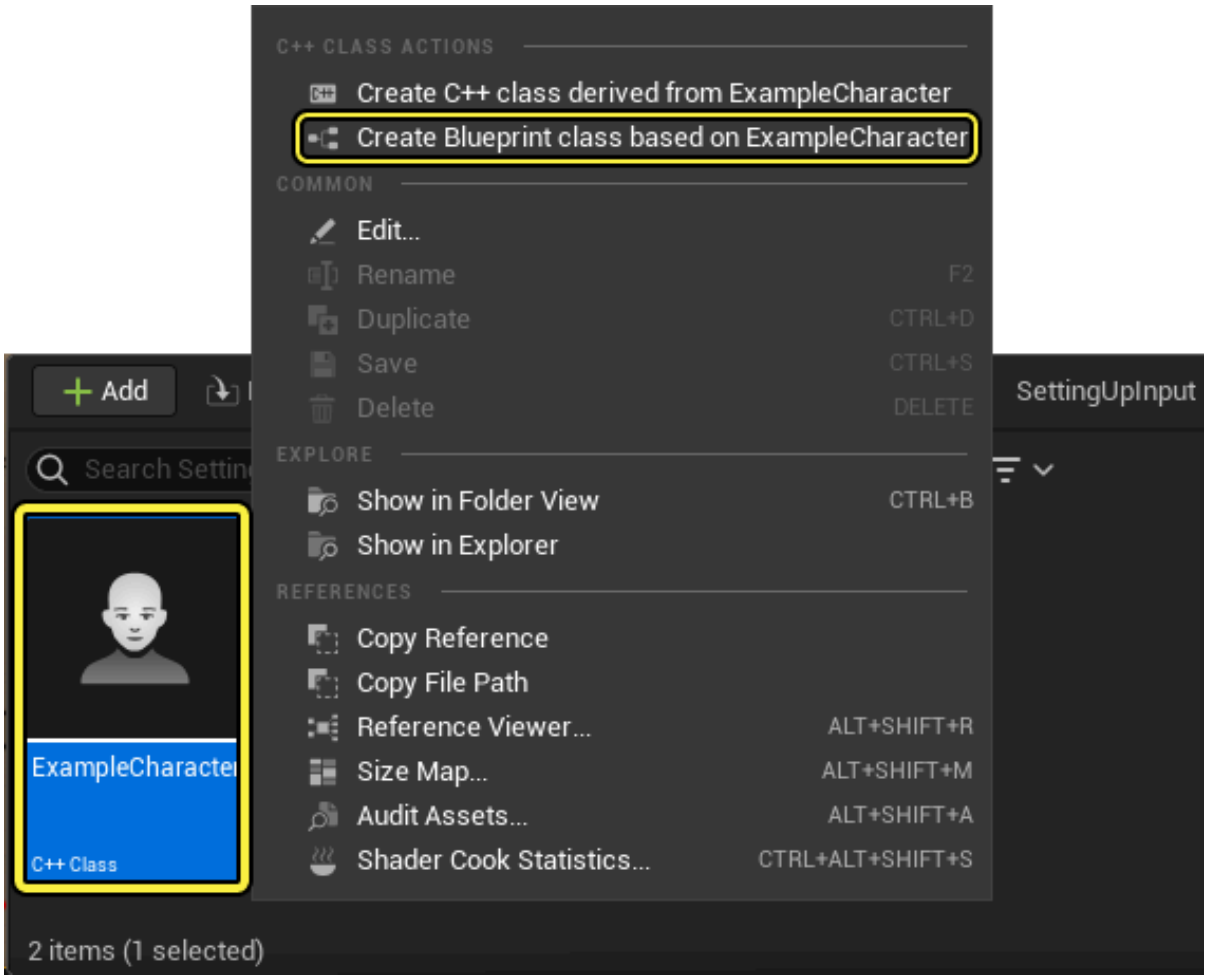
```
FollowCamera->SetupAttachment(CameraBoom);
}

// Called when the game starts or when spawned
void AExampleCharacter::BeginPlay()
{
Super::BeginPlay();

}

void AExampleCharacter::MoveForward(float AxisValue)
{
if ((Controller != NULL) && (AxisValue != 0.0f))
{
// find out which direction is forward
const FRotator Rotation = Controller->GetControlRotation();
const FRotator YawRotation(0, Rotation.Yaw, 0);

// get forward vector
const FVector Direction =
FRotationMatrix(YawRotation).GetUnitAxis(EAxis::X);
AddMovementInput(Direction, AxisValue);
}

}

void AExampleCharacter::MoveRight(float AxisValue)
{
if ((Controller != NULL) && (AxisValue != 0.0f))
{
// find out which direction is right
const FRotator Rotation = Controller->GetControlRotation();
const FRotator YawRotation(0, Rotation.Yaw, 0);

// get right vector
const FVector Direction =
FRotationMatrix(YawRotation).GetUnitAxis(EAxis::Y);

// add movement in that direction
AddMovementInput(Direction, AxisValue);
}
}
```

```
57
58 // Called every frame
59 void AExampleCharacter::Tick(float DeltaTime)
60 {
61 Super::Tick(DeltaTime);
62
63 }
64
65 // Called to bind functionality to input
66 void AExampleCharacter::SetupPlayerInputComponent(UInputComponent*
   PlayerInputComponent)
67 {
68 // Set up gameplay key bindings
69 check(PlayerInputComponent);
70
71 PlayerInputComponent->BindAction("Jump", IE_Pressed, this,
   &ACharacter::Jump);
72 PlayerInputComponent->BindAction("Jump", IE_Released, this,
   &ACharacter::StopJumping);
73
74 PlayerInputComponent->BindAxis("Move Forward", this,
   &AExampleCharacter::MoveForward);
75 PlayerInputComponent->BindAxis("Move Right", this,
   &AExampleCharacter::MoveRight);
76 }
```

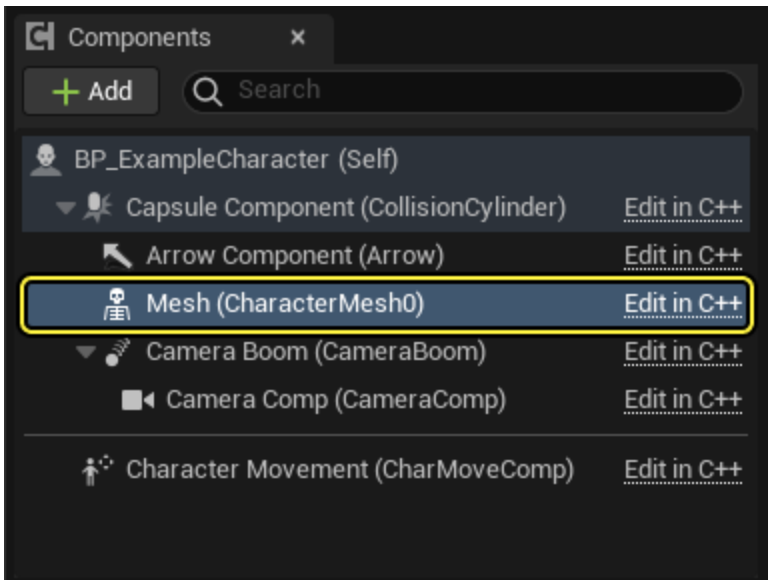
Copy full snippet

# Creating the Character Blueprint

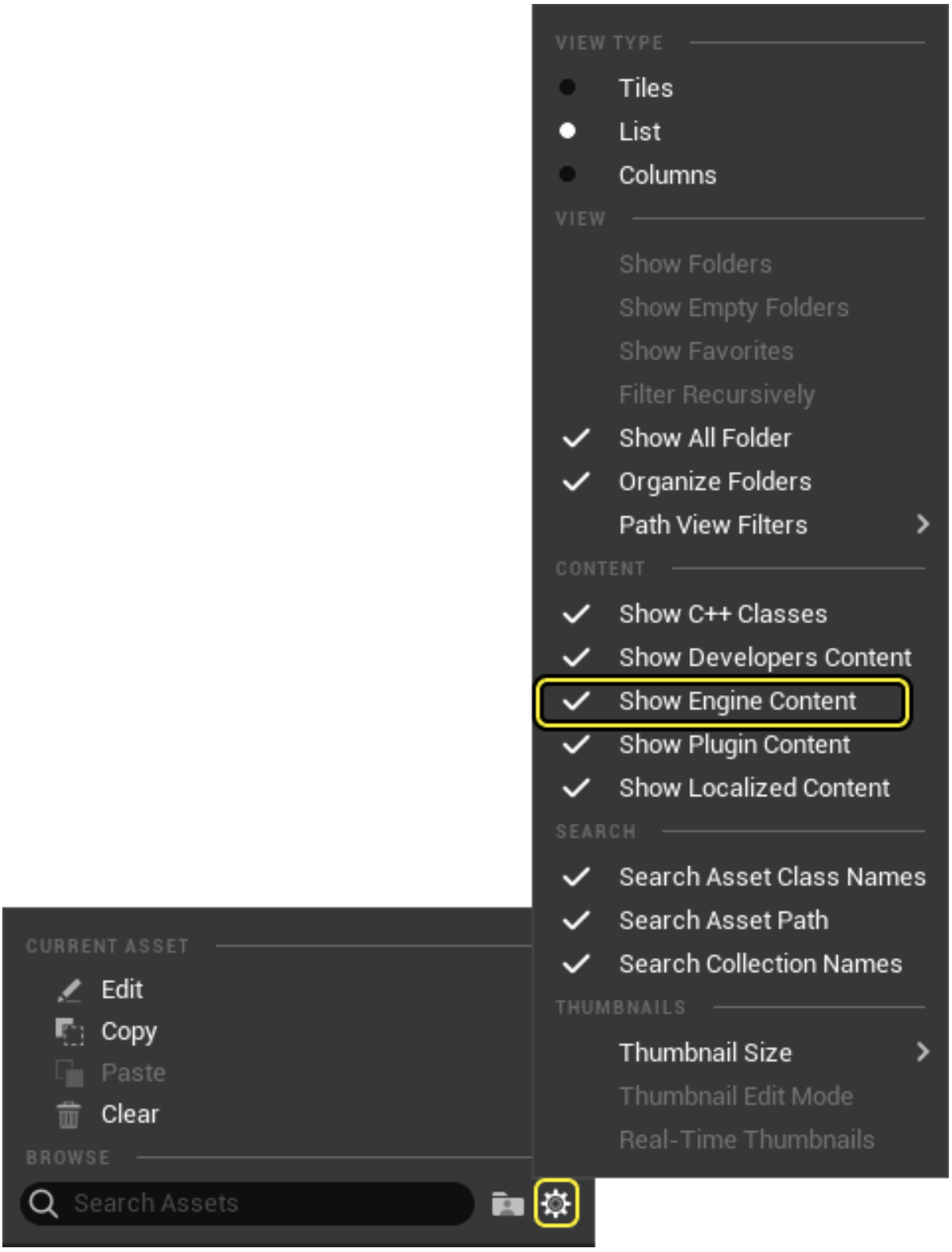
1. Navigate to your **C++ Classes Folder** and right click your **ExampleCharacter**, from the drop down menu select **Create Blueprint class based on ExampleCharacter**. Name your Blueprint **BP_ExampleCharacter**.



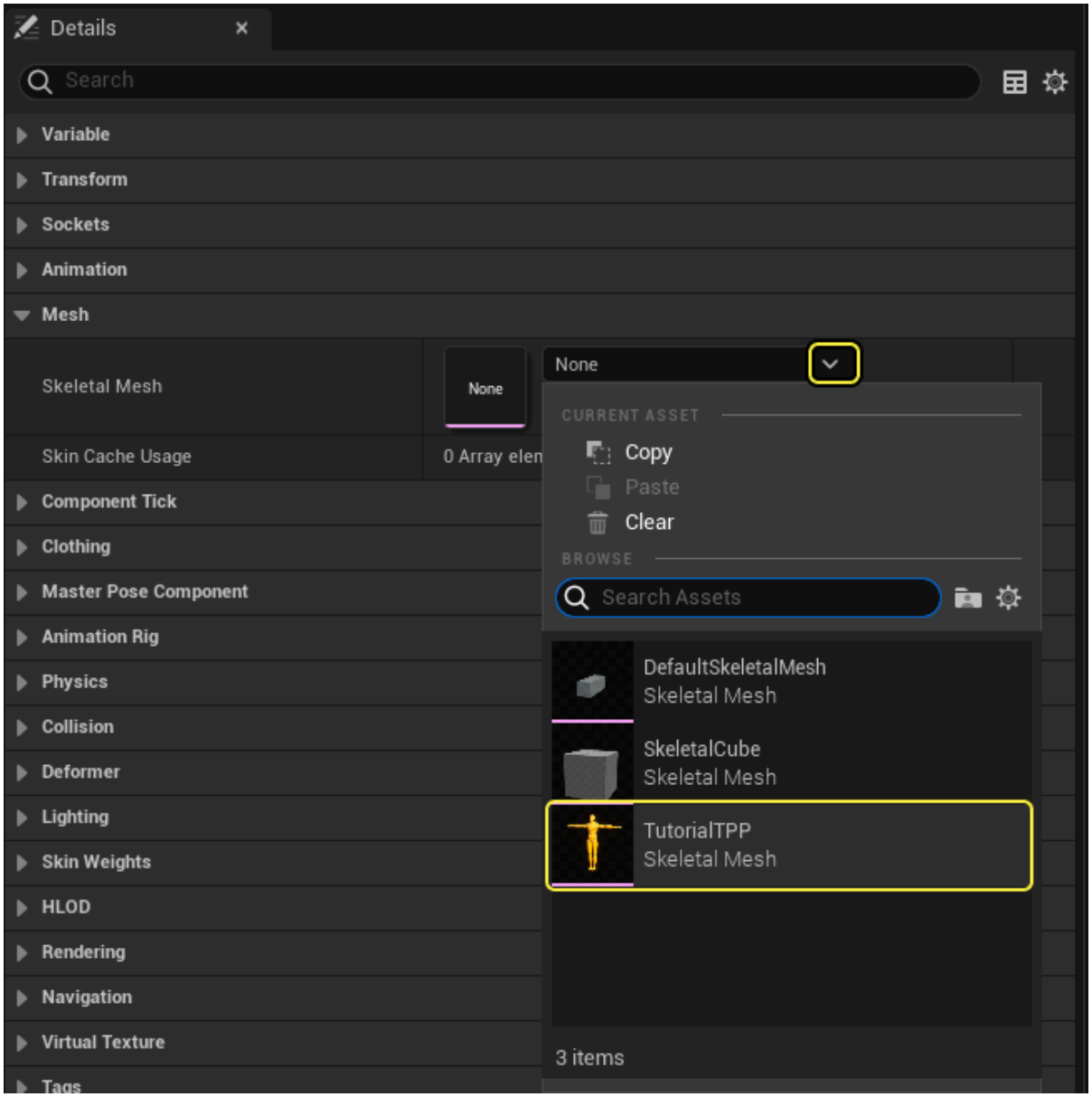

2. In the **Components** panel, select the **Mesh** Skeletal Mesh Component.



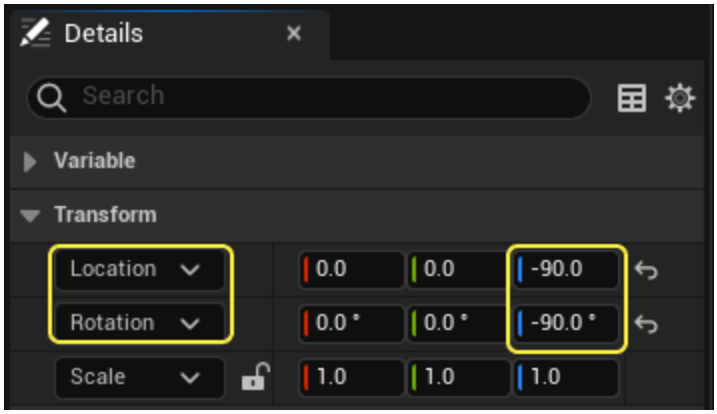

3. Navigate to **Details** > **Mesh** > **Skeletal Mesh** and expand the drop-down menu. In the Browser section, click the **Settings** Icon. Then from the context menu, select **Content** > **Show Engine Content**.




4. Search for and select the **TutorialTPP** Skeletal Mesh.

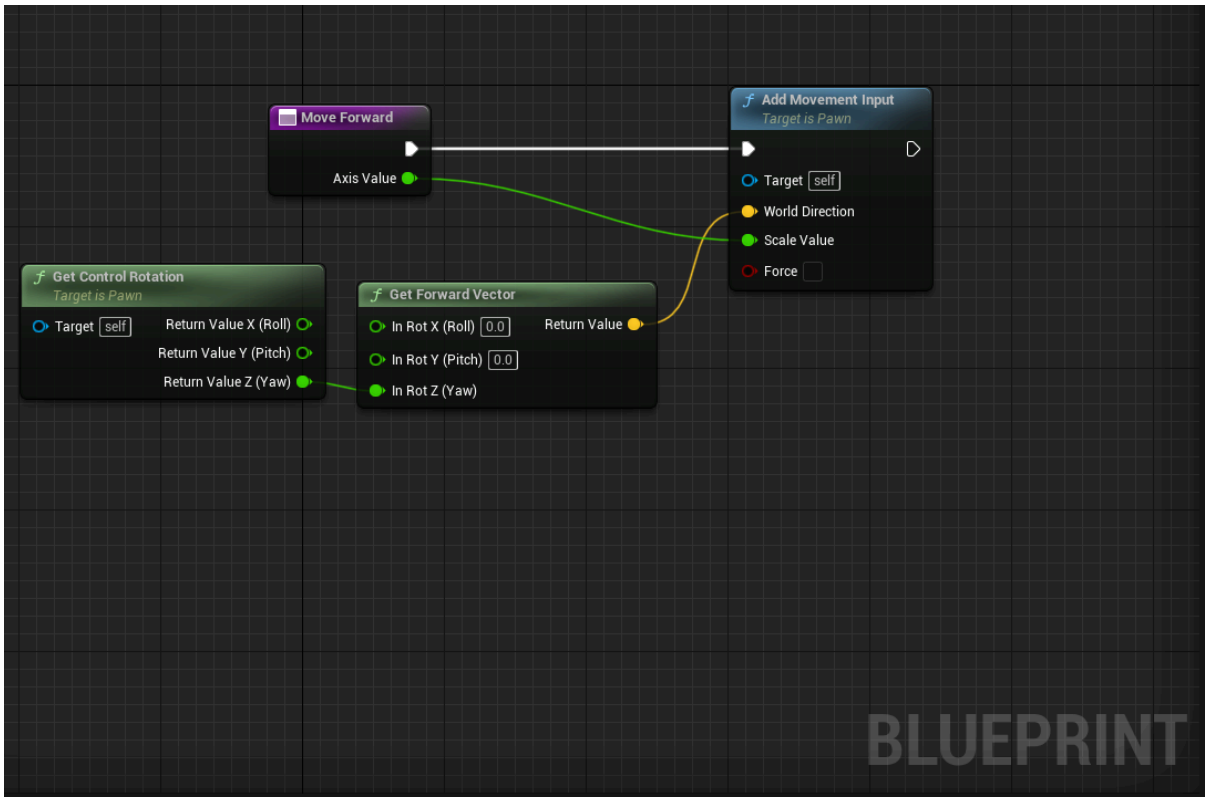



5. Navigate to the **Transform** category, then set the **Location** and **Rotation** vector values to (**0.0, 0.0, -90**)

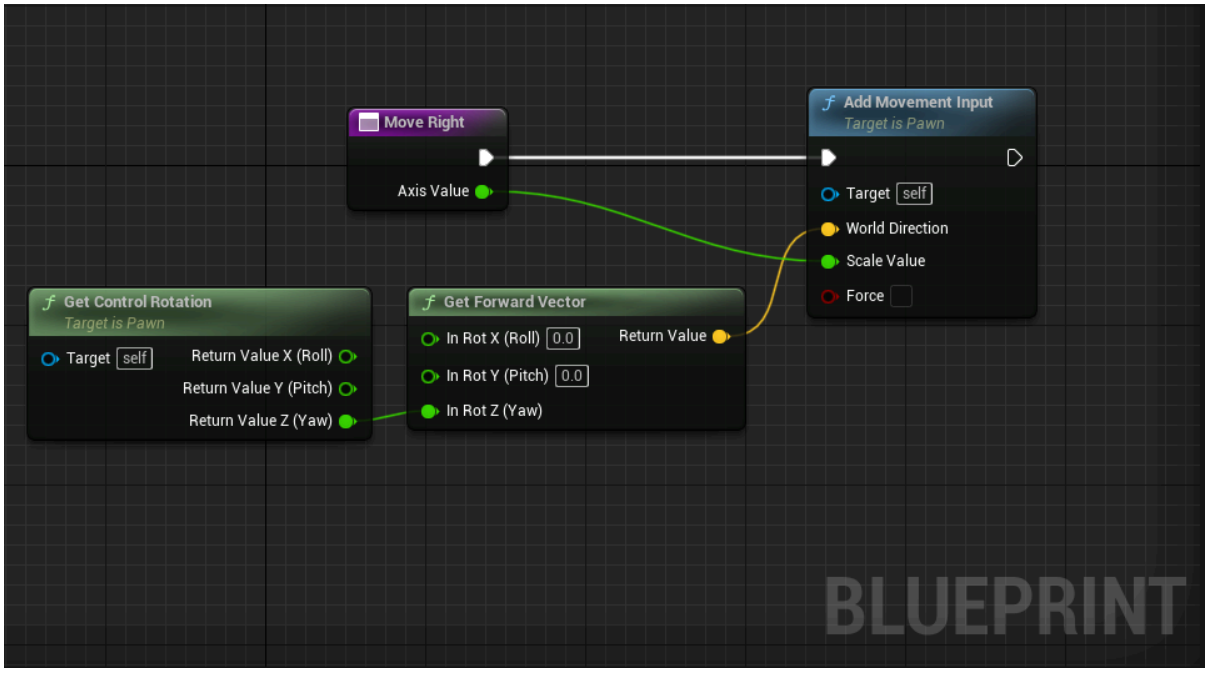



# Finished Blueprint

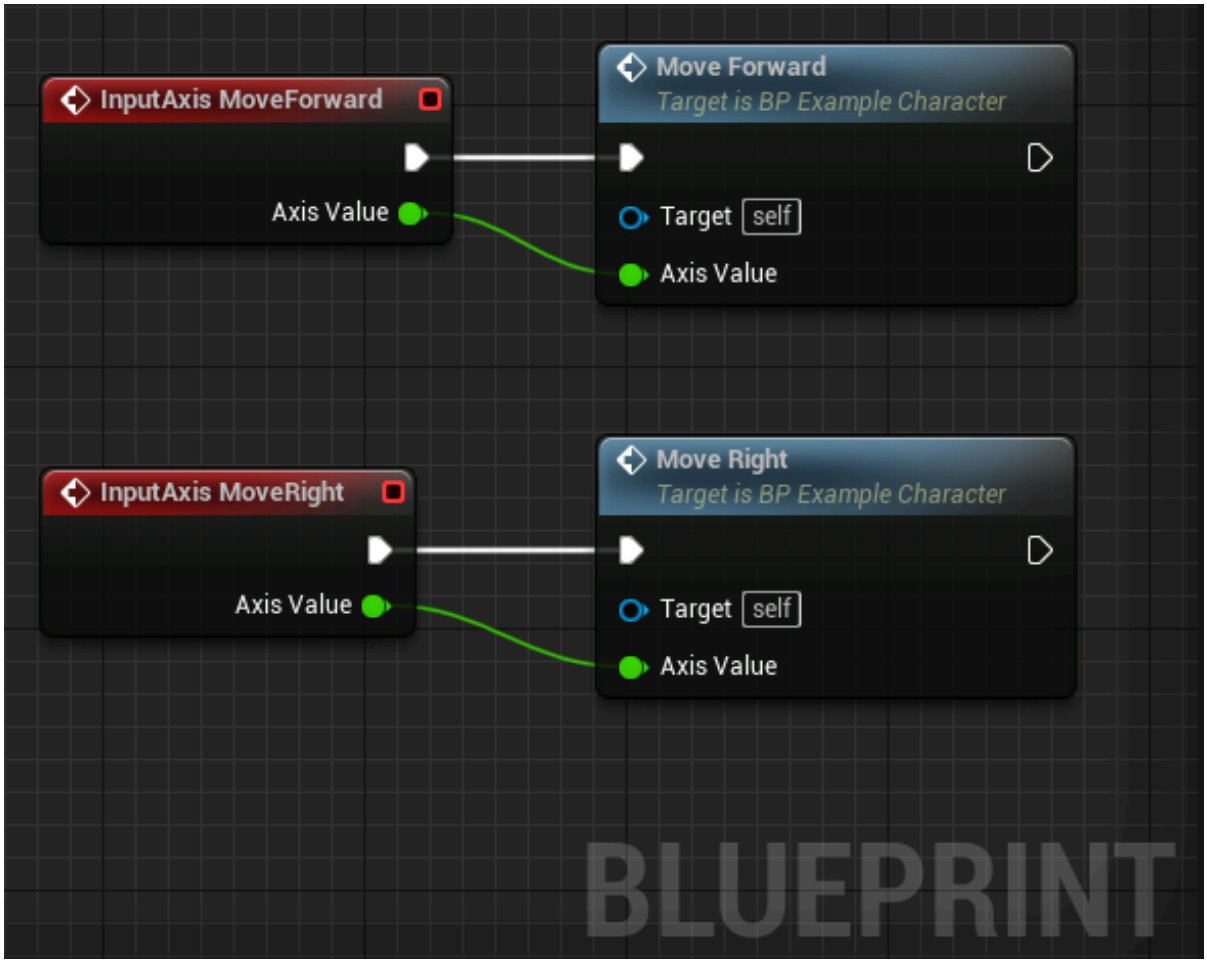
## MoveForward




## MoveRight



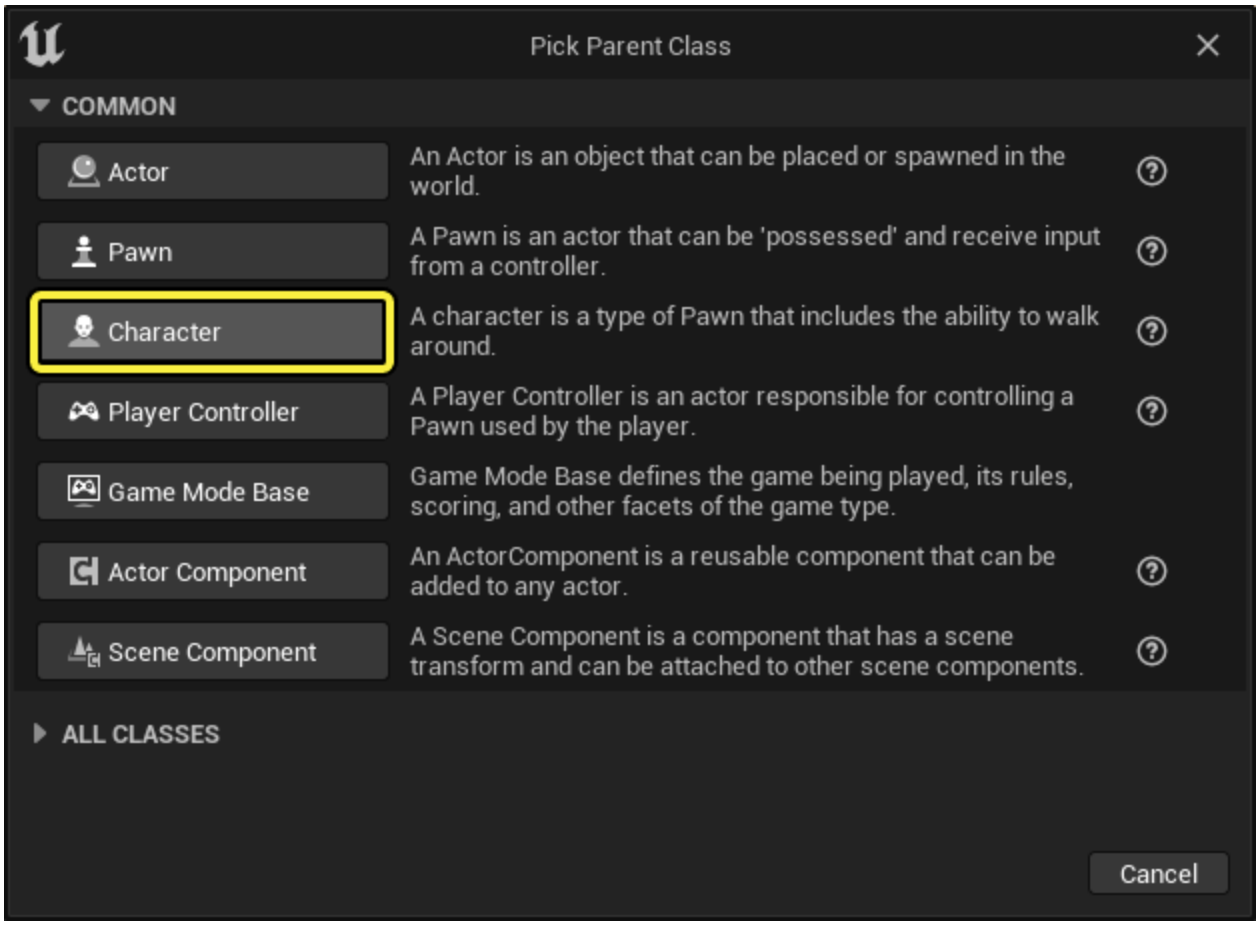

## Event Graph



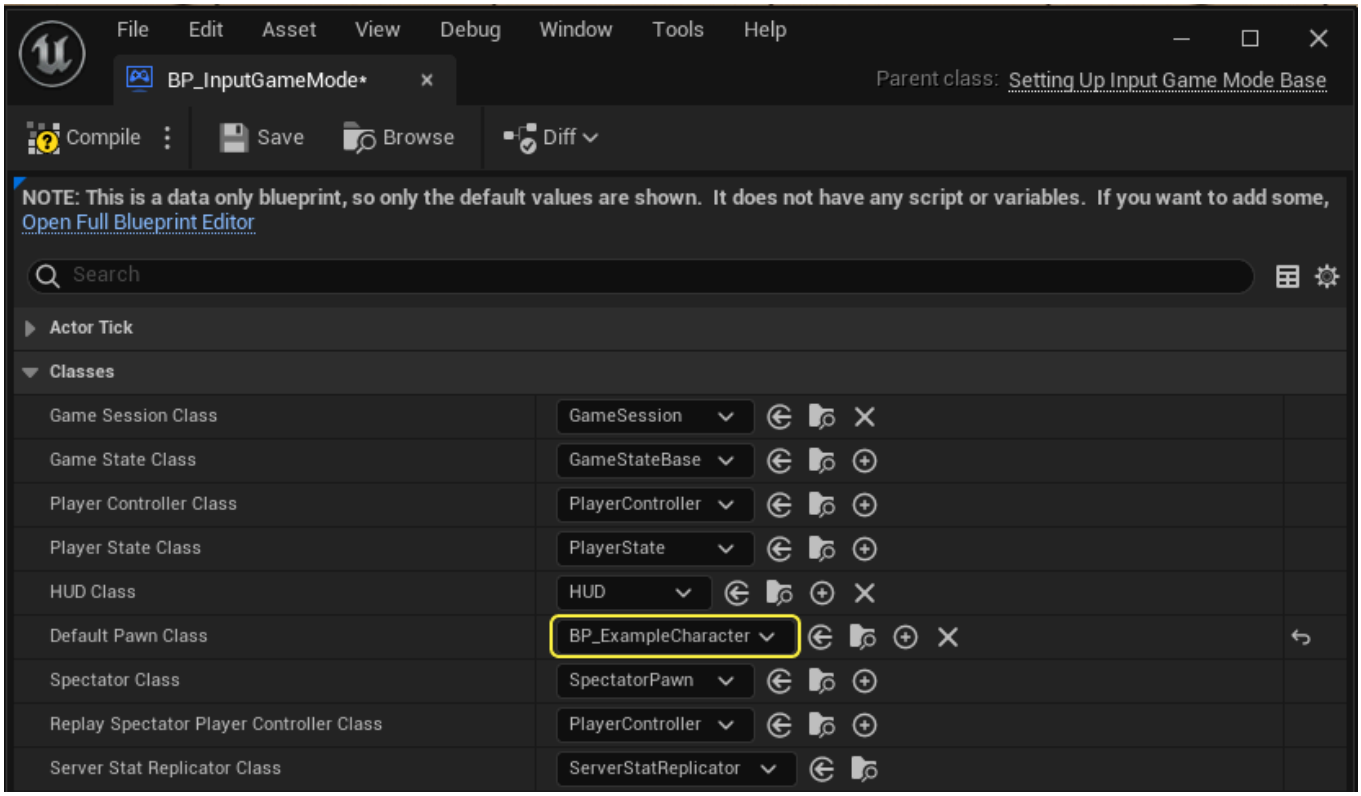

# Creating the GameMode Blueprint

The GameMode defines the game's set of rules. These rules include what default pawn the player will spawn as when the game is launched. You need to set up these rules to spawn the Player Character you created.

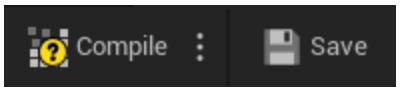
1. In the **Content Drawer**, click **Add(+)** to create a new **Blueprint Class**, then from the drop down menu choose **Game Mode Base** as your **Parent Class**. Name your game mode "**BP_InputGameMode**".



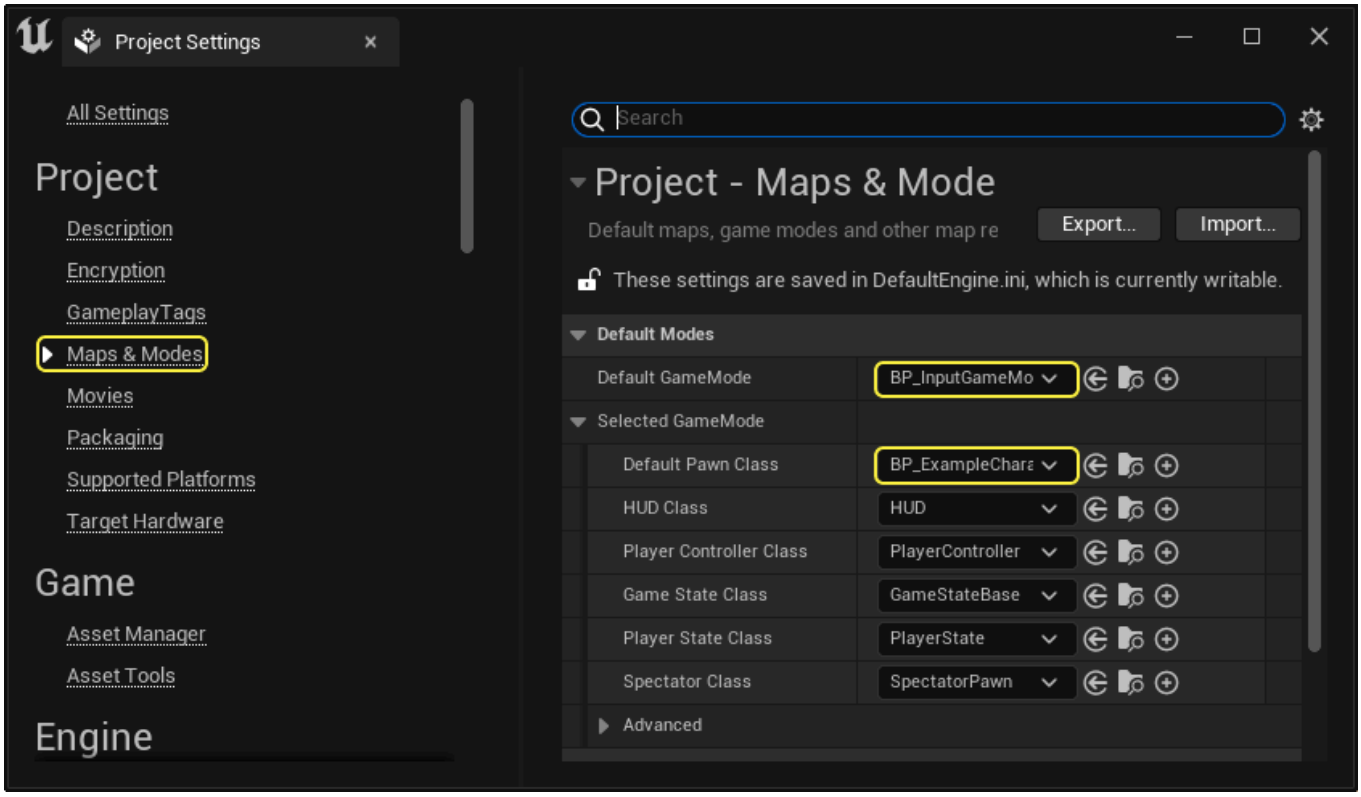

2. In the **Class** defaults, navigate to **Classes** > **Default Pawn Class**, and select **BP_ExampleCharacter**.



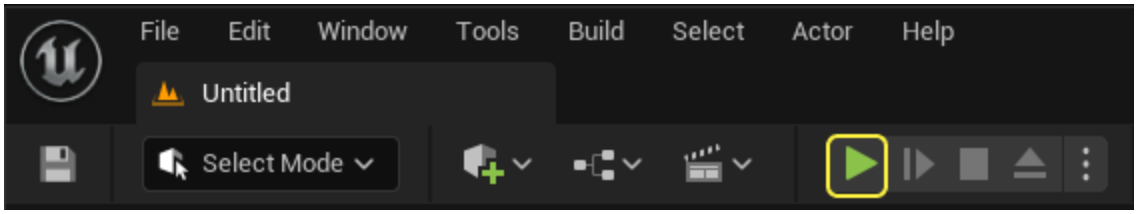

3. **Compile** and **Save**.




4. Navigate to **Edit** > **Project Settings** > **Maps and Modes**. Set the **Default GameMode** to **BP_InputGameMode**.

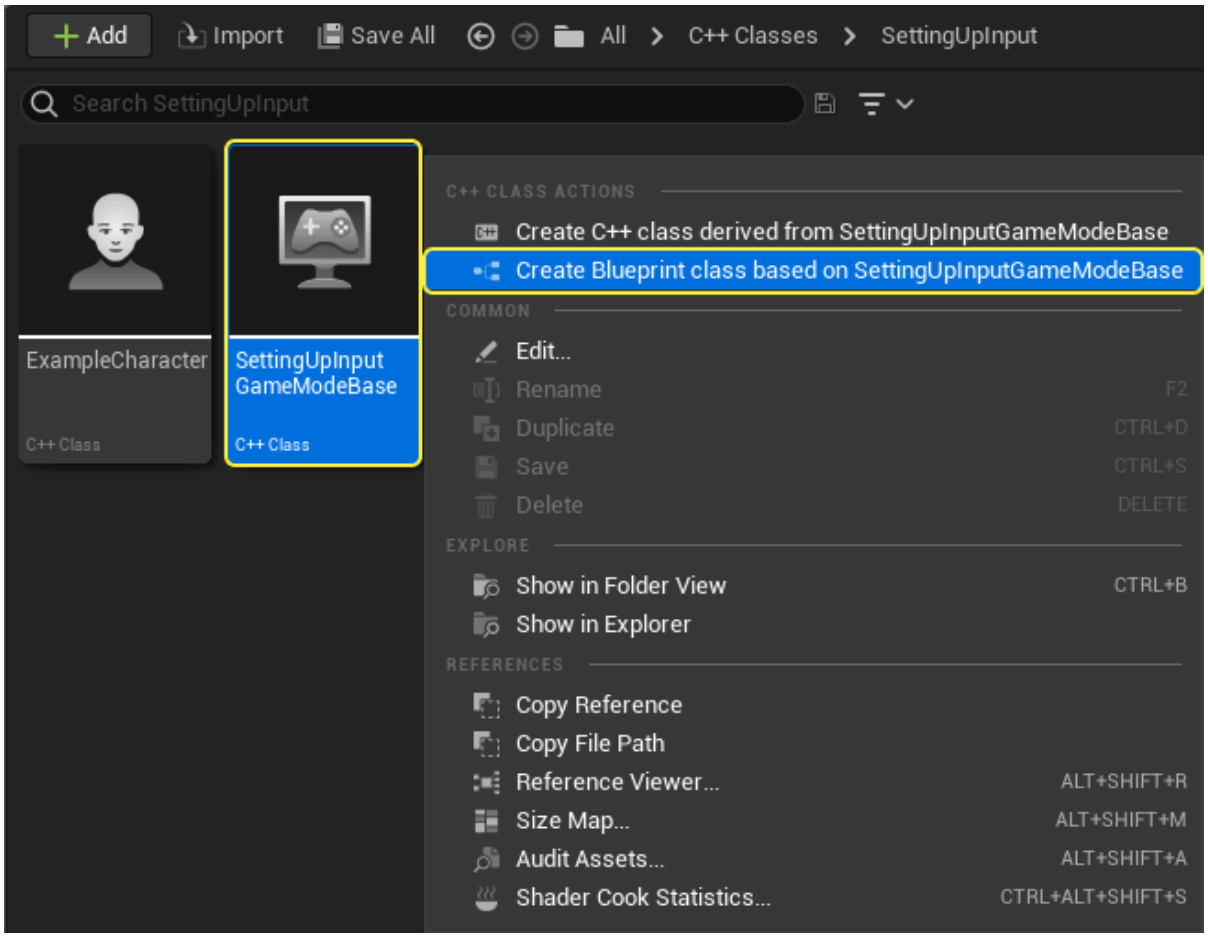



5. Navigate to the **Editor** and select **Play** (**Play in Editor**)

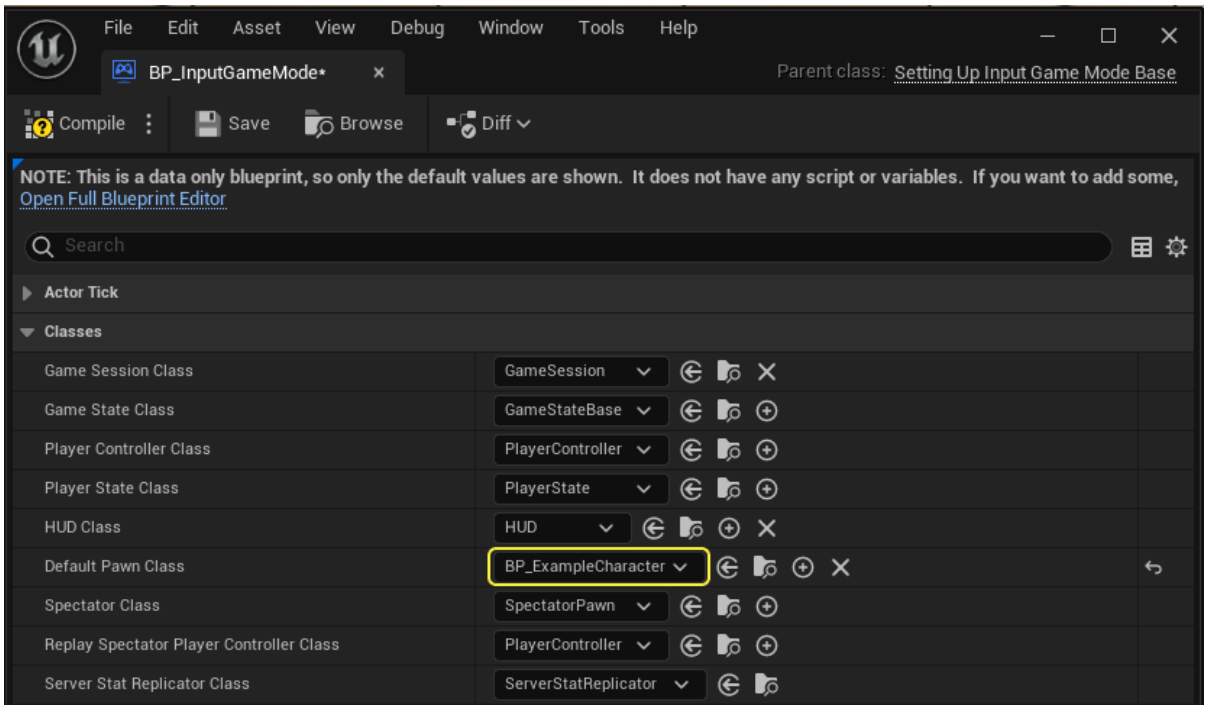



You can now control your character's movement using the W, A, S, D keys. Moving the mouse moves the camera, and pressing the spacebar causes the character to jump.
The GameMode defines the game's set of rules. These rules include what default pawn the player will spawn when the game is launched. You need to set up these rules to spawn the Player Character you created.

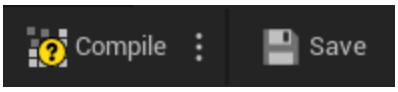
1. In the **Content Drawer**, navigate to your **C++ Classes** folder, right-click the **SettingUpInputGameModeBase**, then in the drop-down menu select **Create Blueprint Based on SettingUpInputGameModeBase**. Name your game mode Blueprint "**BP_InputGameMode**".



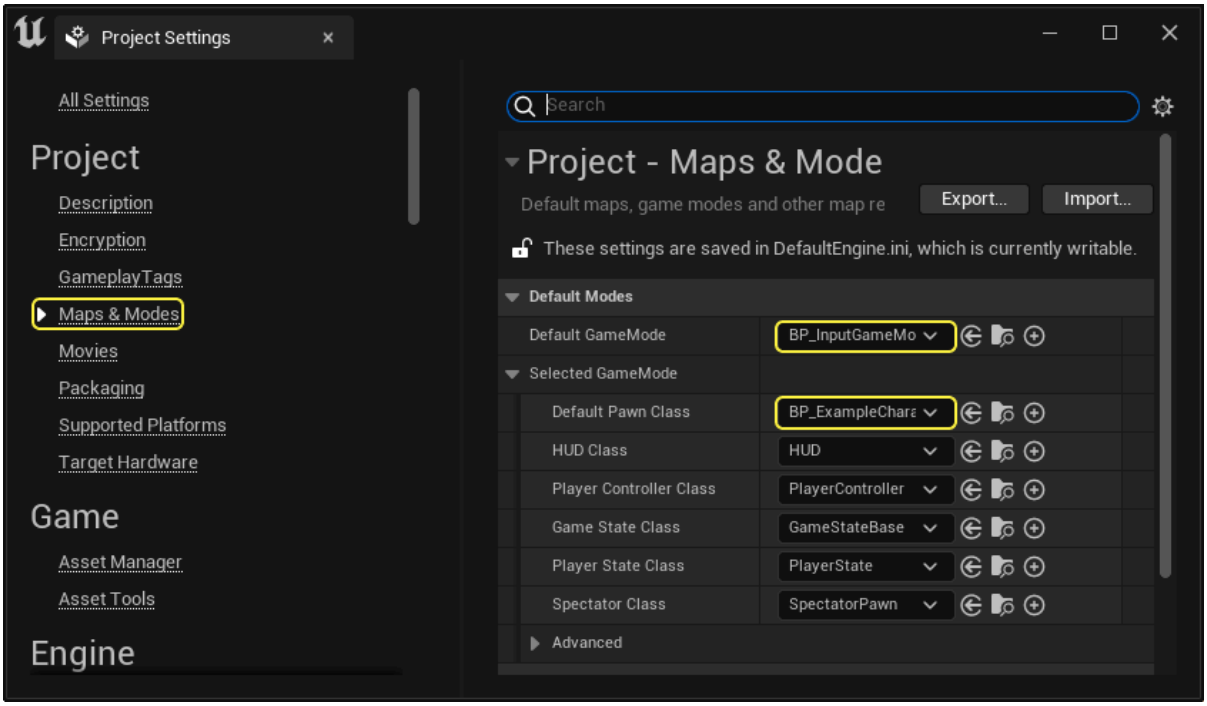

2. In the **Class** defaults, navigate to **Classes** > **Default Pawn Class**, and select the **BP_ExampleCharacter**.



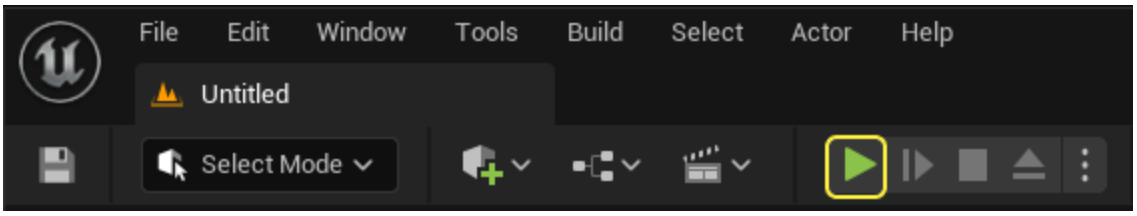

3. **Compile** and **Save**.




4. Navigate to **Edit** > **Project Settings** > **Maps and Modes**. Set the **Default GameMode** to **BP_InputGameMode.**




5. Navigate to the **Editor** and select **Play** (**Play in Editor**)




You can now control your character's movement using the W, A, S, D keys. Moving the mouse moves the camera, and pressing the spacebar causes the character to jump.

# Result