Gameplay Ability System

Links and sources

Example of using the GAS: GASShooter GitHub project

Unofficial but detailed GAS Documentation

Epic livestream in depth look at GAS

Highly recommended: Setting up GAS youtube tutorial

Notes

Any Actor that wishes to use GameplayAbilities, have Attributes, or receive GameplayEffects must have one AbilitySystemComponent (ASC) attached to them.
ASC can be assigned to weapons, players, or AI

The Actor with the ASC attached to it is referred to as the OwnerActor of the ASC.
The physical representation Actor of the ASC is called the AvatarActor
The OwnerActor and AvatarActor can either be the same or different depending on the use case

If your Actor will respawn and need persistence of Attributes or GameplayEffects between spawns (like a hero in a MOBA), then the ideal location for the ASC is on the PlayerState.

Brief

In the sections below, my game is named `unrealgame5`, and any appearances of this string should be replaced by your own project name. This page outlines the process of setting up the Gameplay Ability System for use in an Unreal Engine 5 game using C++. Blueprints can still be used for prototyping new abilities, which can later be translated to C++.

Project Plugins and Modules

To setup out UE5 project to use the required modules, we need to edit our `<PROJECT_NAME>.build.cs` file. This file should have been generated by Unreal Engine when your project was created, and for me my file is named `unrealgame5.build.cs`

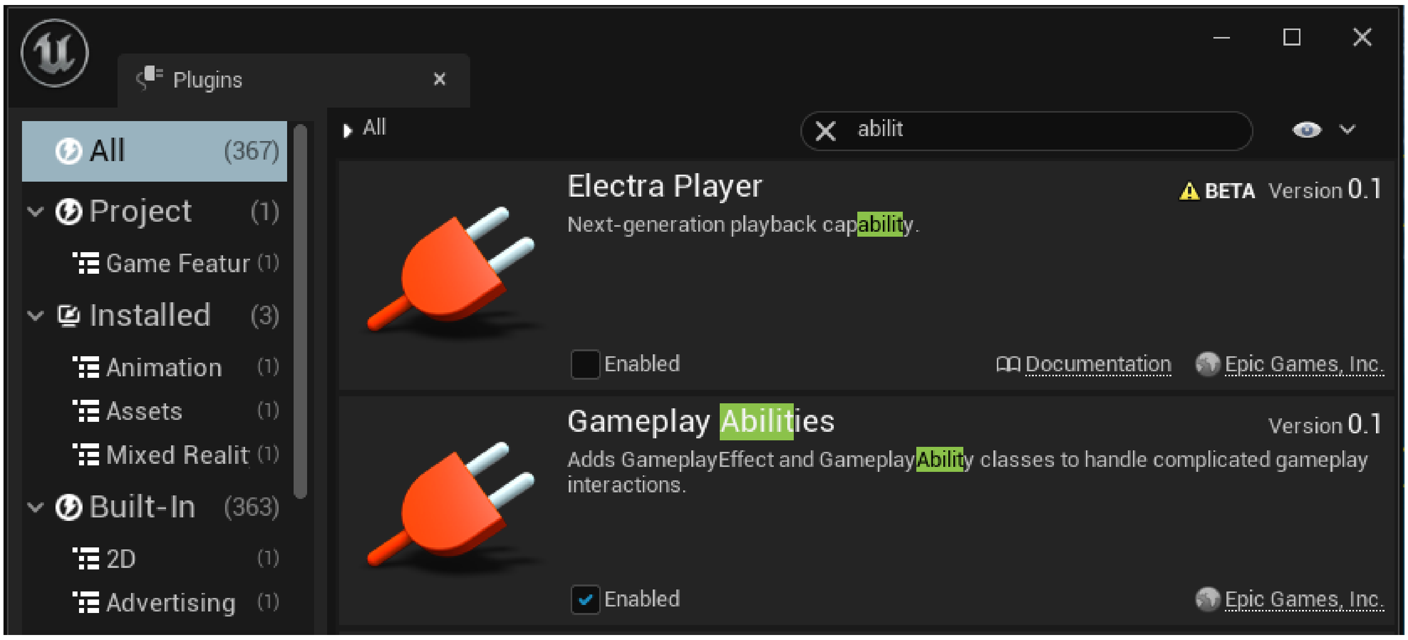
Initial contents of `unrealgame5.build.cs`, before I made any changes.

```
using UnrealBuildTool;

public class unrealgame5 : ModuleRules
{
	public unrealgame5(ReadOnlyTargetRules Target) : base(Target)
	{
		PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;

		PublicDependencyModuleNames.AddRange(new string[] {"Core", "CoreUObject", "Engine", "InputCore",
"HeadMountedDisplay"});
  }
}
```

Add `"GameplayAbilities", "GameplayTags", "GameplayTasks"` modules to `unrealgame5.build.cs` so your file looks like the following.

```
using UnrealBuildTool;

public class unrealgame5 : ModuleRules
{
	public unrealgame5(ReadOnlyTargetRules Target) : base(Target)
	{
		PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;

		PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject", "Engine", "InputCore",
"HeadMountedDisplay" });
		PublicDependencyModuleNames.AddRange(new string[] { "GameplayAbilities", "GameplayTags",
"GameplayTasks" });
	}
}
```

Then within the editor go to `Edit->Plugins...` and enable the `Gameplay Abilities` plugin. You will need to restart the editor for the changes to apply.



Electra Player — BETA Version 0.1
Next-generation playback capability.
Enabled — Documentation — Epic Games, Inc.

Gameplay Abilities — Version 0.1
Adds GameplayEffect and GameplayAbility classes to handle complicated gameplay interactions.
✓ Enabled — Epic Games, Inc.

C++ Initial Setup

To set up the scaffolding for the GAS, we need a few things -

- Enumeration of abilities to correlate with input actions
- Core GameplayAbilitySystem
- Set of attributes for our player and / or enemies
- Player character inheriting from `IAbilitySystemInterface` class
- GameplayEffect to apply default attributes for our player / enemies
- AnimGraphs with Montage slots for handling abilitiy animations

So, to complete this setup you will at least need to define a few new classes for your project.

For my project I follow the same naming convention used with GASDocumentation. I'll paste it below, but for the first few files required to set up the Gameplay Ability System (GAS), we are defining the backend of the system so none of these conventions apply. For these files, I added the `GAS_` convention.

```
Prefix Asset Type
GA_ GameplayAbility
GC_ GameplayCue
GE_ GameplayEffect
GAS_   GameplayAbilitySystem (Core Configurations)
```

Ability Enumeration

First, we need to modify the contents of the header file for our unreal project. My project is named `unrealgame5` so the file is `unrealgame5.h`, and the contents are below. If you already have information here, just make sure the `EGASAbilityInputID` enumeration is added to the header file and save your changes. This enumeration is used to correlate input actions to activate certain abilities in our game.

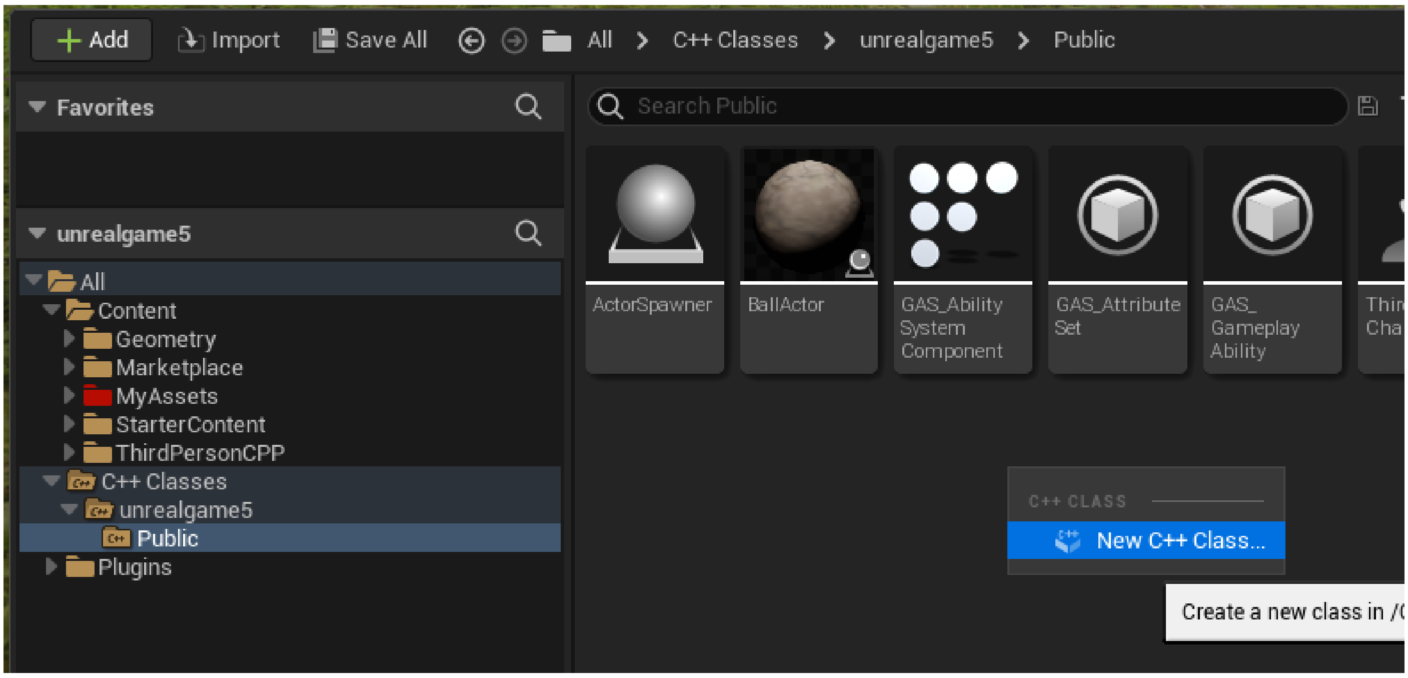
Note: `Attack` below must either be changed to match or made to match some keybind within your `Edit->Project Settings->Input` options menu.

```
// Copyright Epic Games, Inc. All Rights Reserved.


#pragma once


#include "CoreMinimal.h"


UENUM(BlueprintType)
enum class EGASAbilityInputID : uint8
{
  None,
  Confirm,
  Cancel,
  Attack
};
```

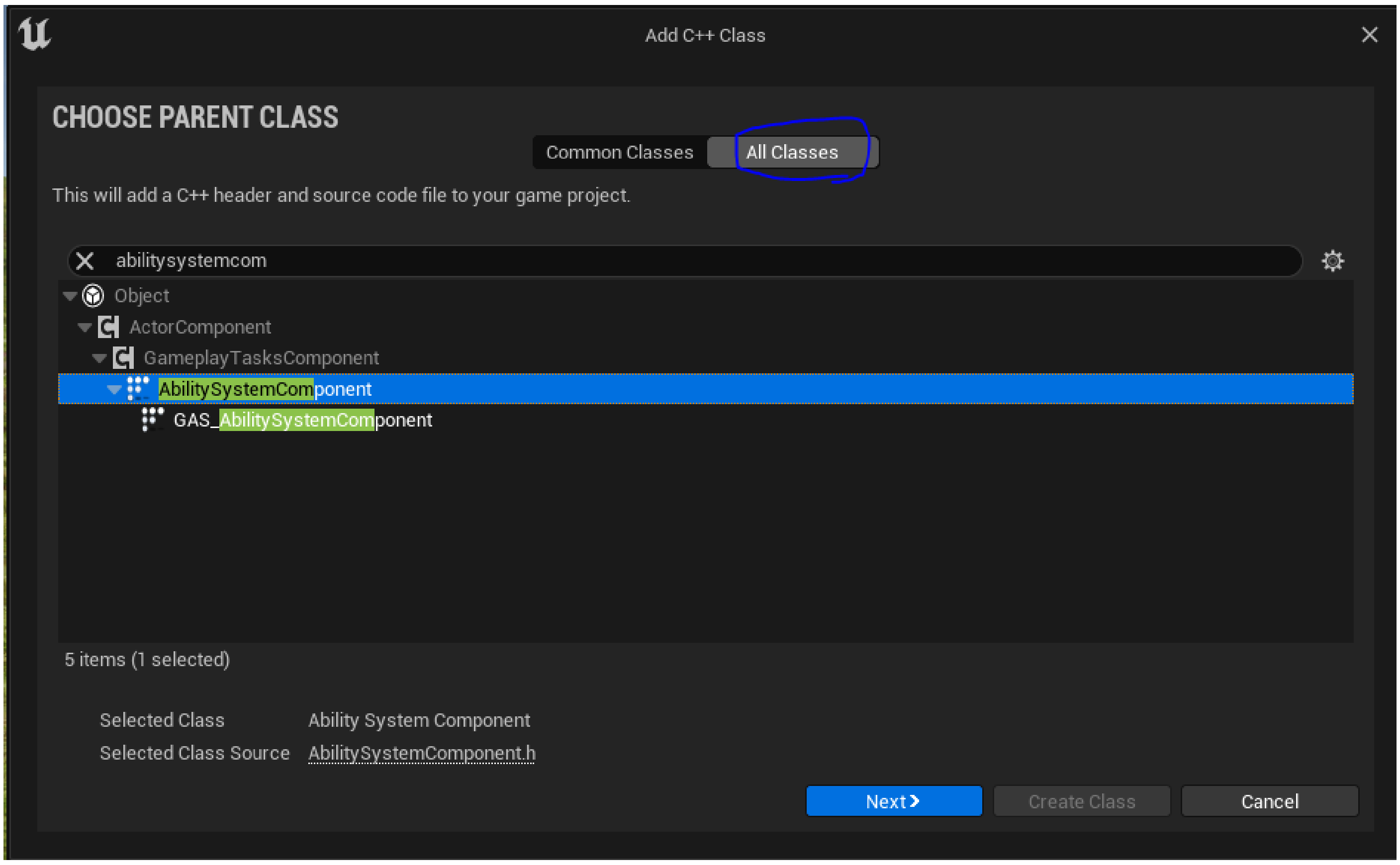
Ability System Component

Next, we need to create an `AbilitySystemComponent`. This will be the component that we attach to actors that we want to take use with the GAS. To create this component, open your project in unreal and create a new C++ source file, inheiriting from the `AbilitySystemComponent` base class.

This base class is provided by UE5, assuming you have the `GameplayAbilities` plugin installed to your project. In order to inherit from it, we need to create a new C++ Class. We must be in the `C++ Classes` subdirectory of our project in order to do this.



Next click `All Classes` and search for `AbilitySystemComponent`.

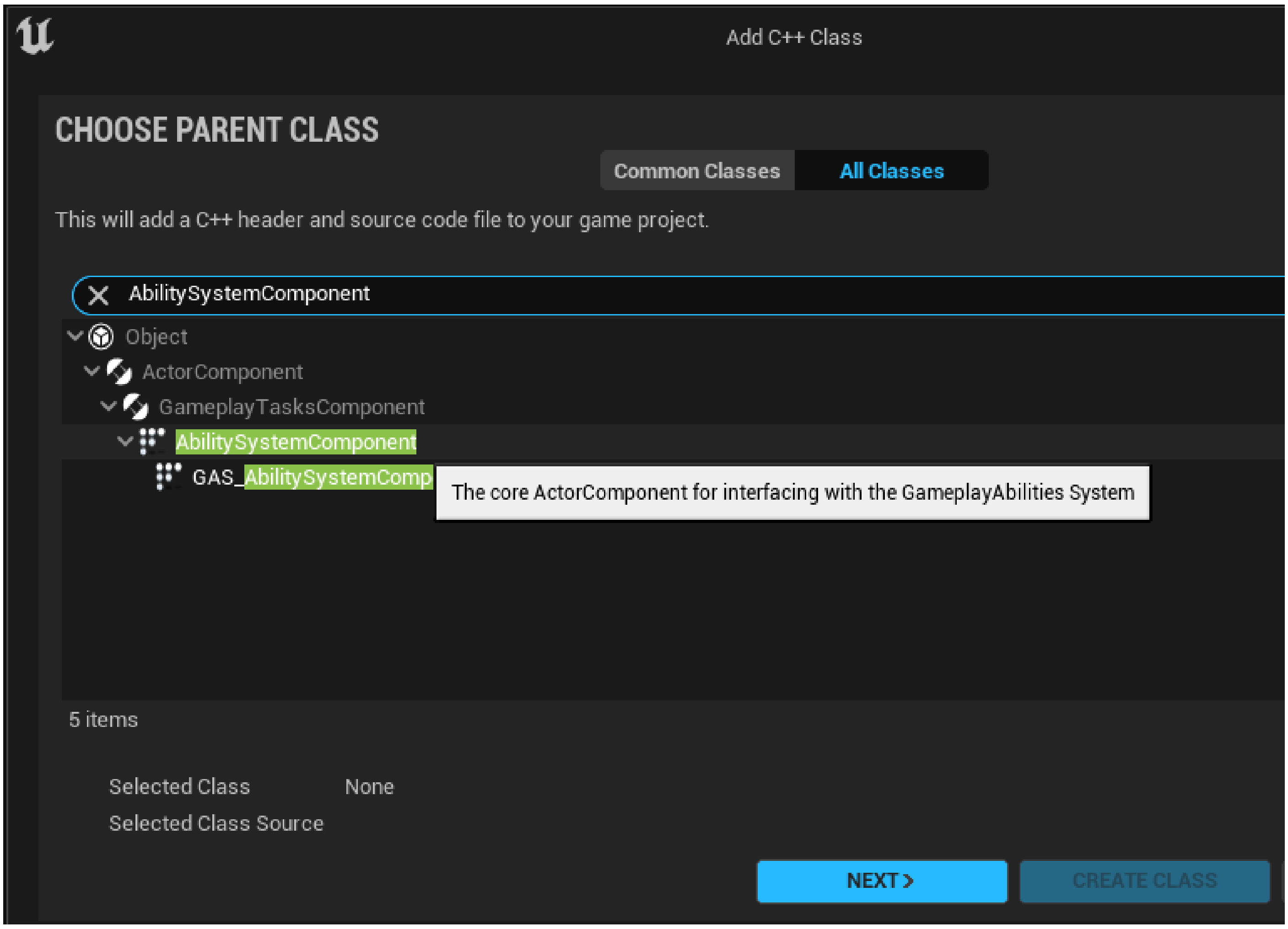


This will add a C++ header and source code file to your game project.

5 items (1 selected)

Selected Class	Ability System Component
Selected Class Source	AbilitySystemComponent.h

Click next and name your class, I'll name this class `GAS_AbilitySystemComponent`.



The generated files are seen below. You don't need to put anything else in here for now. Note that UE5 prefixed our original class name `GAS_AbilitySystemComponent` with a `U` - it's name in the source code is `UGAS_AbilitySystemComponent`, this is normal and to be expected.

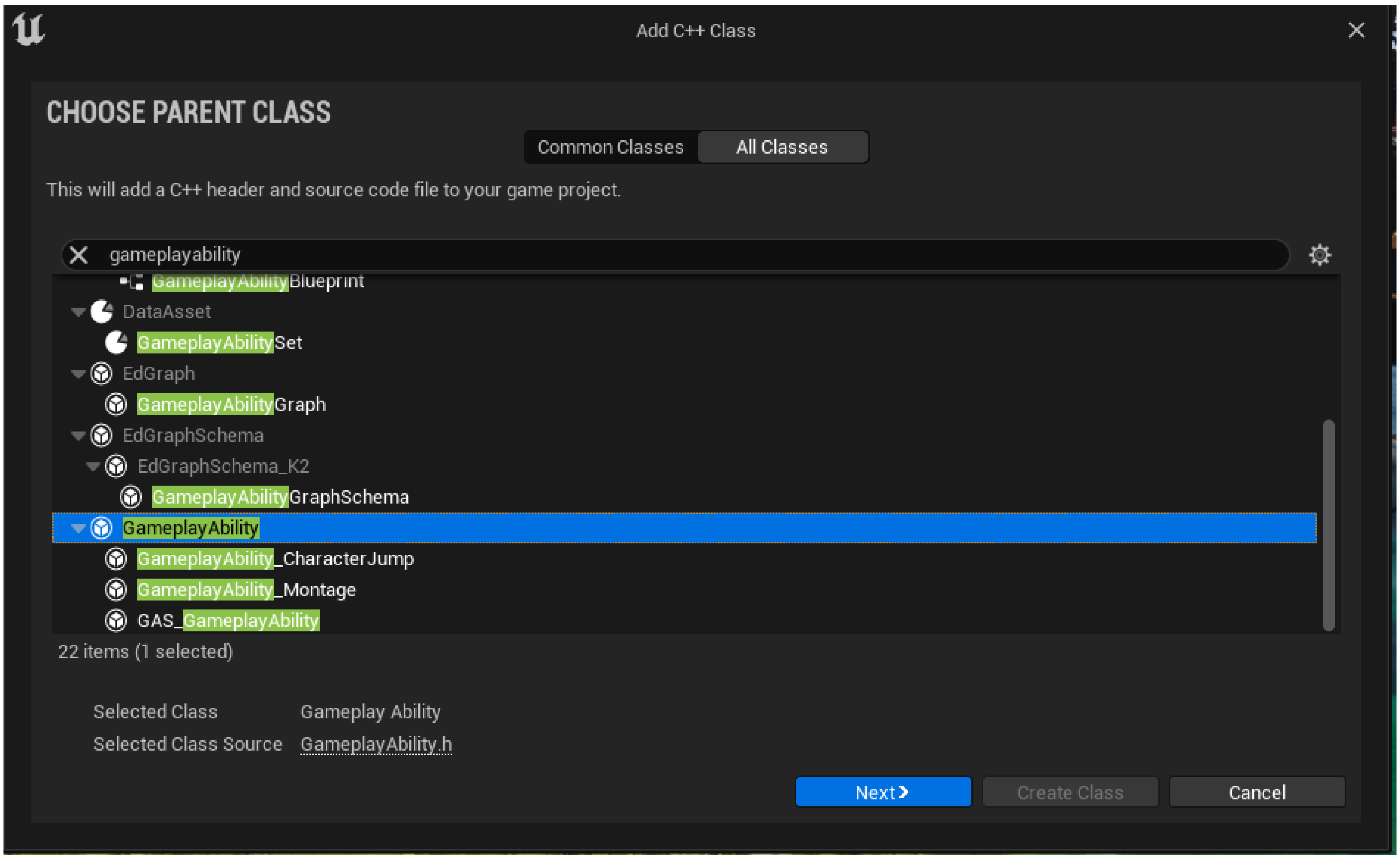
```
// GAS_AbilitySystemComponent.h
// All content (c) Shaun Reed 2021, all rights reserved

#pragma once

#include "CoreMinimal.h"
#include "AbilitySystemComponent.h"
#include "GAS_AbilitySystemComponent.generated.h"

/**
 *
 */
UCLASS()
class UNREALGAME5_API UGAS_AbilitySystemComponent : public UAbilitySystemComponent
{
```

```
GENERATED_BODY()


};
```

```
// GAS_AbilitySystemComponent.cpp
// All content (c) Shaun Reed 2021, all rights reserved


#include "GAS_AbilitySystemComponent.h"
```

Gameplay Abilities

Next we'll setup the base class that we will use for adding abilities to our game. To do this we need to create another new C++ Source file like we did in the previous step, only this time we will inherit from the `GameplayAbility` class provided with the `GameplayAbilities` UE5 plugin.



I named this class `GAS_GameplayAbility` and the source code is seen below

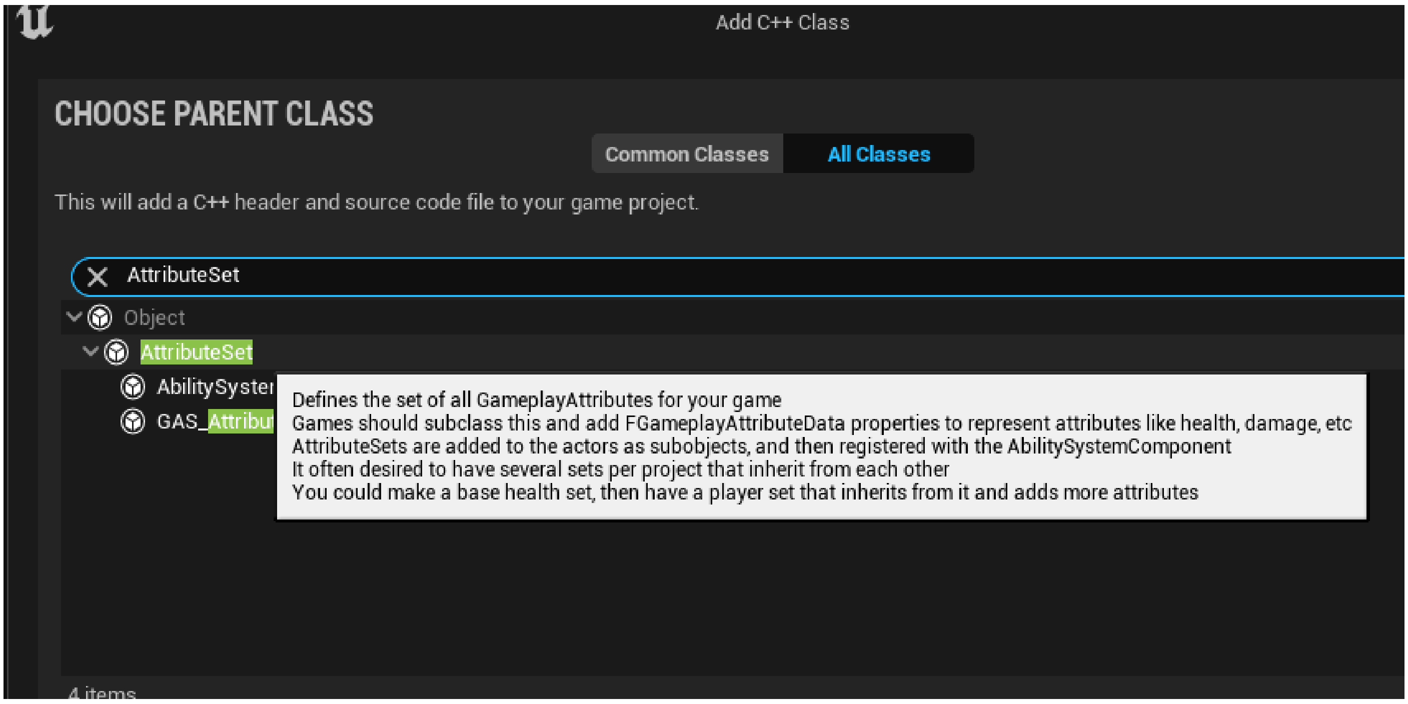
```
// GAS_GameplayAbility.h
// All content (c) Shaun Reed 2021, all rights reserved


#pragma once
```

```
#include "../unrealgame5.h"

#include "CoreMinimal.h"
#include "Abilities/GameplayAbility.h"
#include "GAS_GameplayAbility.generated.h"

UCLASS()
class UNREALGAME5_API UGAS_GameplayAbility : public UGameplayAbility
{
	GENERATED_BODY()

public:
	UGAS_GameplayAbility();

	UPROPERTY(BlueprintReadOnly, EditAnywhere, Category = "Ability")
		EGASAbilityInputID AbilityInputID = EGASAbilityInputID::None;
};
```

```
// All content (c) Shaun Reed 2021, all rights reserved
// GAS_GameplayAbility.cpp

#include "GAS_GameplayAbility.h"

UGAS_GameplayAbility::UGAS_GameplayAbility() { }
```

Attribute Sets

Next, we need to create an `AttributeSet` for our game. Repeat the process of creating a new C++ sourcce file for your ue5 project, but this time inherit from `AttributeSet`



This will add a C++ header and source code file to your game project.

Defines the set of all GameplayAttributes for your game
Games should subclass this and add FGameplayAttributeData properties to represent attributes like health, damage, etc
AttributeSets are added to the actors as subobjects, and then registered with the AbilitySystemComponent
It often desired to have several sets per project that inherit from each other
You could make a base health set, then have a player set that inherits from it and adds more attributes

I named this class `GAS_AttributeSet` and the files genereated are below

```
// GAS_AttributeSet.h
// All content (c) Shaun Reed 2021, all rights reserved


#pragma once


#include "AbilitySystemComponent.h"


#include "CoreMinimal.h"
#include "AttributeSet.h"
#include "GAS_AttributeSet.generated.h"


// Macros to define getters and setters for attributes (AttributeSet.h)
#define ATTRIBUTE_ACCESSORS(ClassName, PropertyName) \
		GAMEPLAYATTRIBUTE_PROPERTY_GETTER(ClassName, PropertyName) \
		GAMEPLAYATTRIBUTE_VALUE_GETTER(PropertyName) \
		GAMEPLAYATTRIBUTE_VALUE_SETTER(PropertyName) \
		GAMEPLAYATTRIBUTE_VALUE_INITTER(PropertyName)


UCLASS()
class UNREALGAME5_API UGAS_AttributeSet : public UAttributeSet
{
	GENERATED_BODY()
```

```
  UGAS_AttributeSet();

public:
    virtual void GetLifetimeReplicatedProps(TArray<FLifetimeProperty>& OutLifetimeProps) const override;

    /*
    * Attribute Definitions
    */

    // Health

    UPROPERTY(BlueprintReadOnly, Category = "Attributes", ReplicatedUsing = OnRep_Health)
    FGameplayAttributeData Health;
    // Use macros we defined from AttributeSet.h to generate getters and setters
    ATTRIBUTE_ACCESSORS(UGAS_AttributeSet, Health);

    UFUNCTION()
        virtual void OnRep_Health(const FGameplayAttributeData& OldHealth);

    // Stamina

    UPROPERTY(BlueprintReadOnly, Category = "Attributes", ReplicatedUsing = OnRep_Stamina)
    FGameplayAttributeData Stamina;
    ATTRIBUTE_ACCESSORS(UGAS_AttributeSet, Stamina);

    UFUNCTION()
        virtual void OnRep_Stamina(const FGameplayAttributeData& OldStamina);

    // Attack Power

    UPROPERTY(BlueprintReadOnly, Category = "Attributes", ReplicatedUsing = OnRep_AttackPower)
    FGameplayAttributeData AttackPower;
    ATTRIBUTE_ACCESSORS(UGAS_AttributeSet, AttackPower);

    UFUNCTION()
        virtual void OnRep_AttackPower(const FGameplayAttributeData& OldAttackPower);
};
```

```
// GAS_AttributeSet.cpp
// All content (c) Shaun Reed 2021, all rights reserved
```

```
#include "Net/UnrealNetwork.h"  // DOREPLIFETIME
#include "GAS_AttributeSet.h"

UGAS_AttributeSet::UGAS_AttributeSet()
{
}

void UGAS_AttributeSet::GetLifetimeReplicatedProps(TArray<FLifetimeProperty>& OutLifetimeProps) const
{
  Super::GetLifetimeReplicatedProps(OutLifetimeProps);

  DOREPLIFETIME_CONDITION_NOTIFY(UGAS_AttributeSet, Health, COND_None, REPNOTIFY_Always);
  DOREPLIFETIME_CONDITION_NOTIFY(UGAS_AttributeSet, Stamina, COND_None, REPNOTIFY_Always);
  DOREPLIFETIME_CONDITION_NOTIFY(UGAS_AttributeSet, AttackPower, COND_None, REPNOTIFY_Always);


}

void UGAS_AttributeSet::OnRep_Health(const FGameplayAttributeData& OldHealth)
{
  GAMEPLAYATTRIBUTE_REPNOTIFY(UGAS_AttributeSet, Health, OldHealth);
}

void UGAS_AttributeSet::OnRep_Stamina(const FGameplayAttributeData& OldStamina)
{
  GAMEPLAYATTRIBUTE_REPNOTIFY(UGAS_AttributeSet, Stamina, OldStamina);
}

void UGAS_AttributeSet::OnRep_AttackPower(const FGameplayAttributeData& OldAttackPower)
{
  GAMEPLAYATTRIBUTE_REPNOTIFY(UGAS_AttributeSet, AttackPower, OldAttackPower);
}
```

Character Setup

In the files below, my character is named `ThirdPersonCharacter`, so any appearances of this string may need to be replaced with your character's name instead. To setup your character that inherits from `ACharacter` base class, make the following changes to your files.

In the `ThirdPersonCharacter.h` file, make sure you're inheriting from `public IABilitySystemInterface`. The start of your class should look like this. Pay attention to the includes.

```
// All content (c) Shaun Reed 2021, all rights reserved

#pragma once

// GAS includes
#include "AbilitySystemInterface.h"
#include <GameplatEffectTypes.h>

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

UCLASS()
class UNREALGAME5_API AThirdPersonCharacter : public ACharacter, public IAbilitySystemInterface
{
    GENERATED_BODY()

public:

// more code....
```

Character Components

Next, we add an instance of our `GAS_AbilitySystem` class using the `UGAS_AbilitySystemComponent` typename, and we also add an instance of our `GAS_AttributeSet` class using the `UGAS_AttributeSet` type.

```
UCLASS()
class UNREALGAME5_API AThirdPersonCharacter : public ACharacter, public IAbilitySystemInterface
{
    GENERATED_BODY()

public:
    // GAS declarations
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "GAS")
    class UGAS_AbilitySystemComponent* AbilitySystemComponent;

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "GAS")
    class UGAS_AttributeSet* Attributes;
// more code....
```

Now we need to modify the character's constructor to add the new components we've declared. I removed the code from my constructor that wasn't related to the GAS. The additions are below.

```
// ThirdPersonCharacter.cpp


AThirdPersonCharacter::AThirdPersonCharacter()
{
    // Initializing any components unrelated to GAS...
    // ...


    // Initialize GAS related components
    AbilitySystemComponent =
CreateDefaultSubobject<UGAS_AbilitySystemComponent>(TEXT("AbilitySystemComponent"));
    AbilitySystemComponent->SetIsReplicated(true);
    AbilitySystemComponent->SetReplicationMode(EGameplayEffectReplicationMode::Minimal);
    Attributes = CreateDefaultSubobject<UGAS_AttributeSet>(TEXT("Attributes"));
}
```

So we have the components we need, and the next step is to provide the required definitions for virtual functions we've inheirted from `IAbilitySystemInterface`

Virtual Functions

To start, we declare the required virtual functions that we will need to define to use the GAS.

```
UCLASS()
class UNREALGAME5_API AThirdPersonCharacter : public ACharacter, public IAbilitySystemInterface
{
    GENERATED_BODY()


public:
    // GAS declarations
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "GAS")
    class UGAS_AbilitySystemComponent* AbilitySystemComponent;


    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "GAS")
    class UGAS_AttributeSet* Attributes;


    virtual class UAbilitySystemComponent* GetAbilitySystemComponent() const override;
```

```
// more code....
```

We implement `GetAbilitySystemComponent`, which is just a simple getter that returns our `UGAS_AbilitySystemComponent` component.

```
// ThirdPersonCharacter.cpp
UAbilitySystemComponent* AThirdPersonCharacter::GetAbilitySystemComponent() const
{
	return AbilitySystemComponent;
}
```

Next we need to overload a virtual function `InitializeAttributes()` to handle initializing the attributes for our game at the start. We also declare the `DefaultAttributeEffect` member variable to help define and apply the default attributes for our character.

```
// ThirdPersonCharacter.h
UCLASS()
class UNREALGAME5_API AThirdPersonCharacter : public ACharacter, public IAbilitySystemInterface
{
	GENERATED_BODY()

public:
	// GAS declarations
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "GAS")
	class UGAS_AbilitySystemComponent* AbilitySystemComponent;

	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "GAS")
    class UGAS_AttributeSet* Attributes;

	virtual class UAbilitySystemComponent* GetAbilitySystemComponent() const override;

    // Add these lines
  	virtual void InitializeAttributes();
    UPROPERTY(BlueprintReadOnly, EditDefaultOnly, Category = "GAS")
    TSubclassOf<class UGameplayEffect> DefaultAttributeEffect;
// more code....
```

Define `InitializeAttributes()`

```
// ThirdPersonCharacter.cpp
void AThirdPersonCharacter::InitializeAttributes()
```

```
{
    // If the ASC and DefaultAttributeEffect objects are valid
    if (AbilitySystemComponent && DefaultAttributeEffect)
    {
        // Create context object for this gameplay effecct
        FGameplayEffectContextHandle EffectContext = AbilitySystemComponent->MakeEffectContext();
        EffectContext.AddSourceObject(this);

        // Create an outgoing effect spec using the effect to apply and the context
        FGameplayEffectSpecHandle SpecHandle = AbilitySystemComponent-
>MakeOutgoingSpec(DefaultAttributeEffect, 1, EffectContext);

        if (SpecHandle.IsValid())
        {
            // Apply the effect using the derived spec
            // + Could be ApplyGameplayEffectToTarget() instead if we were shooting a target
            FActiveGameplayEffectHandle GEHandle = AbilitySystemComponent-
>ApplyGameplayEffectSpecToSelf(*SpecHandle.Data.Get());
        }
    }
}
```

Similar to how we defined default attributes, we define default abilities for our character by overloading the `GiveAbilities()` function. We also add the `DefaultAbilities` array to store the default abilities for the character.

Notice that we use the `UPROPERTY` macro to apply `EditDefaultOnly` to our components. This will later allow us to modify these components in the UE5 editor for our character's blueprint, so we can dynamically add and remove attributes and abilities for our player without modifying the code each time.

```
// ThirdPersonCharacter.h
UCLASS()
class UNREALGAME5_API AThirdPersonCharacter : public ACharacter, public IAbilitySystemInterface
{
    GENERATED_BODY()

public:
    // GAS declarations

    // Define components to store ASC and attributes
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "GAS")
    class UGAS_AbilitySystemComponent* AbilitySystemComponent;
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "GAS")
    class UGAS_AttributeSet* Attributes;

    virtual class UAbilitySystemComponent* GetAbilitySystemComponent() const override;
    // Overload to initialize attributes for GAS, and component to store default attributes
    virtual void InitializeAttributes();
    UPROPERTY(BlueprintReadOnly, EditDefaultOnly, Category = "GAS")
    TSubclassOf<class UGameplayEffect> DefaultAttributeEffect;

    // Overload to initialize abilities for GAS, and component to store default abilities
    virtual void GiveAbilities();
    UPROPERTY(BlueprintReadOnly, EditDefaultOnly, Category = "GAS")
    TArray<TSubclassOf<class UGAS_GameplayAbility>> DefaultAbilities;
public:
// more code....
```

And we define `GiveAbilities` below to handle the allocation of default abilities to our character.

```
// ThirdPersonCharacter.cpp
void AThirdPersonCharacter::GiveAbilities()
{
    // If the server has the authority to grant abilities and there is a valid ASC
    if (HasAuthority() && AbilitySystemComponent)
    {
        // Foreach ability in DefaultAbilities, grant the ability
        for (TSubclassOf<UGAS_GameplayAbility>& StartupAbility : DefaultAbilities)
        {
            // `1` below is the level of the ability, which could later be used to allow abilities to scale with player level
            AbilitySystemComponent->GiveAbility(
                FGameplayAbilitySpec(StartupAbility, 1, static_cast<int32>(StartupAbility.GetDefaultObject()->AbilityInputID)
this));
        }
    }
}
```

For the next step, we need to override `PossessedBy` and `OnRep_PlayerState()` to define how to update the server and client of the player state respectively.

```
// ThirdPersonCharacter.h
UCLASS()
class UNREALGAME5_API AThirdPersonCharacter : public ACharacter, public IAbilitySystemInterface
{
	GENERATED_BODY()

public:
	// GAS declarations

	// Define components to store ASC and attributes
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "GAS")
	class UGAS_AbilitySystemComponent* AbilitySystemComponent;
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "GAS")
  class UGAS_AttributeSet* Attributes;

	virtual class UAbilitySystemComponent* GetAbilitySystemComponent() const override;
	// Overload to initialize attributes for GAS, and component to store default attributes
	virtual void InitializeAttributes();
	UPROPERTY(BlueprintReadOnly, EditDefaultOnly, Category = "GAS")
  TSubclassOf<class UGameplayEffect> DefaultAttributeEffect;

	// Overload to initialize abilities for GAS, and component to store default abilities
	virtual void GiveAbilities();
	UPROPERTY(BlueprintReadOnly, EditDefaultOnly, Category = "GAS")
	TArray<TSubclassOf<class UGAS_GameplayAbility>> DefaultAbilities;
public:	
// more code....
```

And see the definitions for these functions below

```
// ThirdPersonCharacter.cpp
void AThirdPersonCharacter::PossessedBy(AController* NewController)
{
	Super::PossessedBy(NewController);

	// Owner and Avatar are bother this character
	AbilitySystemComponent->InitAbilityActorInfo(this, this);

	InitializeAttributes();
	GiveAbilities();
}

void AThirdPersonCharacter::OnRep_PlayerState()
{
	Super::OnRep_PlayerState();

	AbilitySystemComponent->InitAbilityActorInfo(this, this);
	InitializeAttributes();

	if (AbilitySystemComponent && InputComponent)
	{
		// Where the 3rd parameter is a string equal to enum typename defined in unrealgame5.h
		const FGameplayAbilityInputBinds Binds("Confirm", "Cancel", "EGASAbilityInputID",
static_cast<int32>(EGASAbilityInputID::Confirm), static_cast<int32>(EGASAbilityInputID::Cancel));
		AbilitySystemComponent->BindAbilityActivationToInputComponent(InputComponent, Binds);
	}
}
```

As a final modification, add the following code to the function equivalent to `ThirdPersonCharacter::SetupPlayerInputComponent()` in your project. This is the same code as the last portion of `OnRep_PlayerState`

```
void AThirdPersonCharacter::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
{
	// Code unrelated to GAS...
	// ...

	// Make sure GAS is valid along with player input component
	if (AbilitySystemComponent && InputComponent)
	{
		// Where the 3rd parameter is a string equal to enum typename defined in unrealgame5.h
		const FGameplayAbilityInputBinds Binds("Confirm", "Cancel", "EGASAbilityInputID",
static_cast<int32>(EGASAbilityInputID::Confirm), static_cast<int32>(EGASAbilityInputID::Cancel));
		AbilitySystemComponent->BindAbilityActivationToInputComponent(InputComponent, Binds);
	}
}
```

And just to be sure we have all the headers we need, here are the final includes for my character

```
// ThirdPersonCharacter.h
// GAS includes
```

```
#include "AbilitySystemInterface.h"
#include <GameplayEffectTypes.h>
#include "GAS_AbilitySystemComponent.h"
#include "GAS_GameplayAbility.h"
#include "GAS_AttributeSet.h"

#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "ThirdPersonCharacter.generated.h"
```

```
// ThirdPersonCharacter.cpp
// All content (c) Shaun Reed 2021, all rights reserved

#include "ThirdPersonCharacter.h"

// Custom includes (Not related to GAS)
#include "ActorSpawner.h"  // Fireball spawner object
#include "BallActor.h"  // Fireball object

// Includes for GAS
#include "../unrealgame5.h"

// Engine includes
#include "Kismet/GameplayStatics.h" // For spawning fireball static mesh
#include "Camera/CameraComponent.h"
#include "GameFramework/SpringArmComponent.h"
#include "GameFramework/CharacterMovementComponent.h"
```

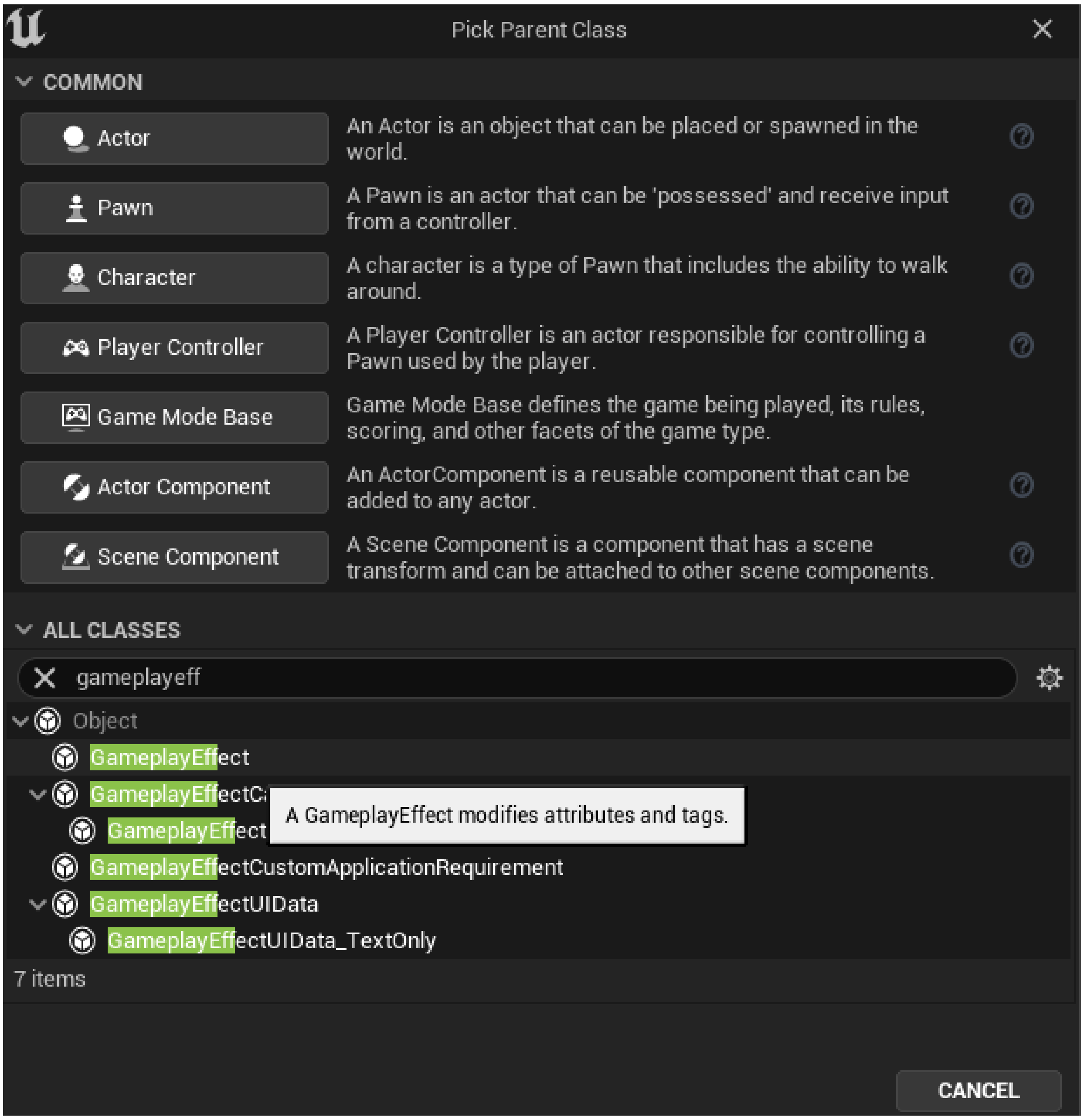
Defining Abilities

At this point we have configured GAS for our project and our character, so we're ready to start defining our abilities!

In my assets folder, I just created an `Abilities` subdirectory and continued with the steps below, creating the assets within this directory.

Default Abilities

First I created a new Blueprint Class using the editor and derived from the `GameplayEffect` class. Applying this effect will result in the player or character being granted a set of default abilities.



An Actor is an object that can be placed or spawned in the world.

A Pawn is an actor that can be 'possessed' and receive input from a controller.

A character is a type of Pawn that includes the ability to walk around.

A Player Controller is an actor responsible for controlling a Pawn used by the player.

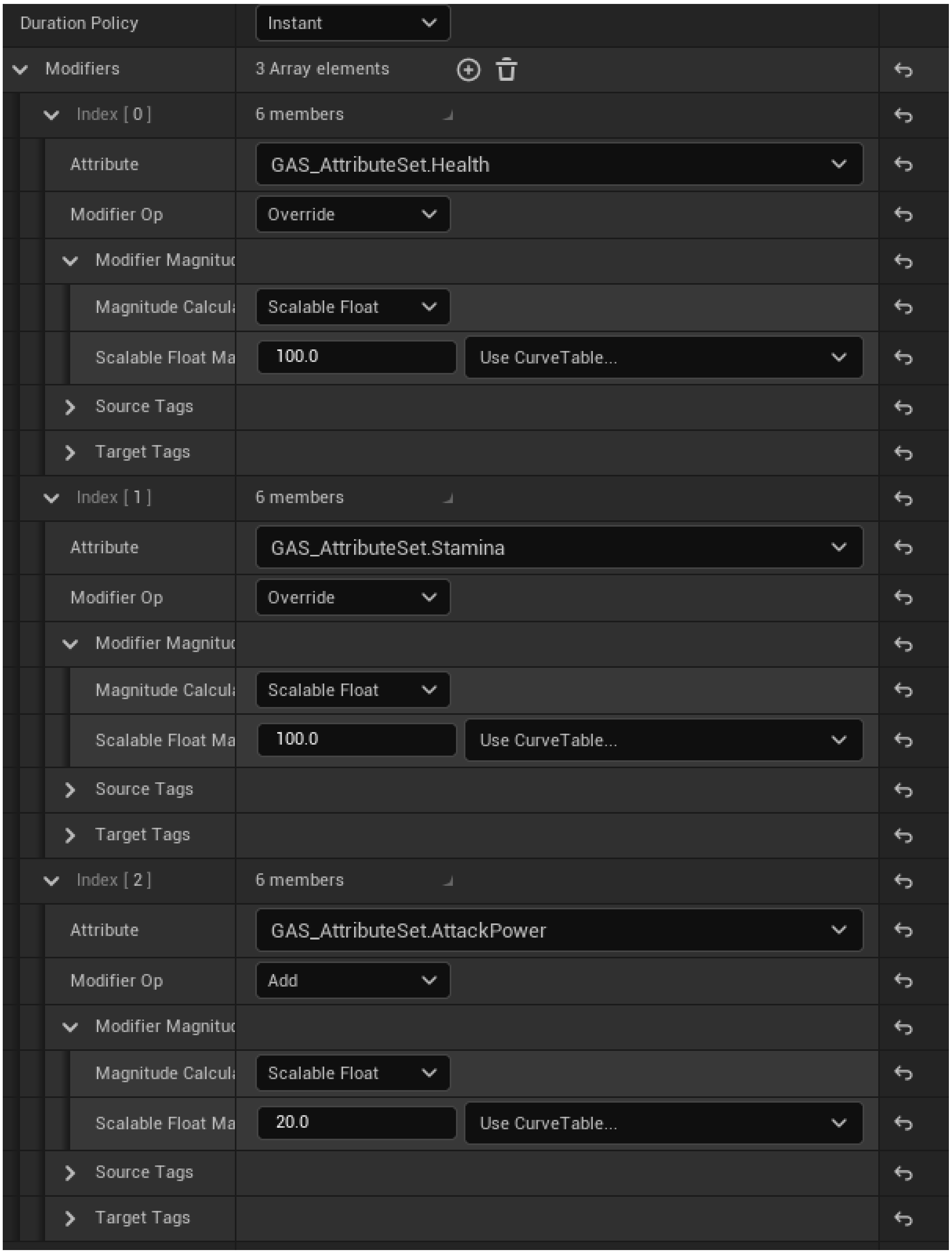
Game Mode Base defines the game being played, its rules, scoring, and other facets of the game type.

An ActorComponent is a reusable component that can be added to any actor.

A Scene Component is a component that has a scene transform and can be attached to other scene components.

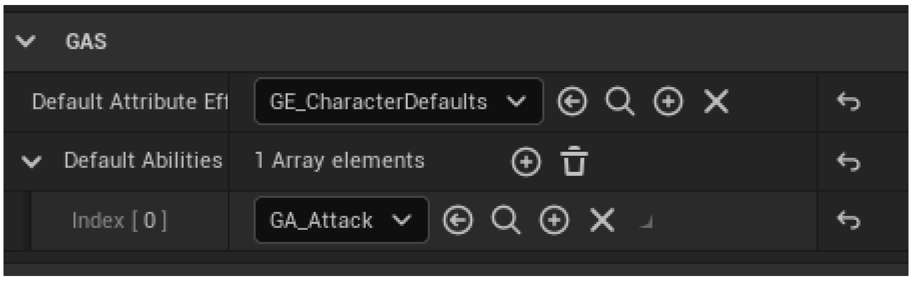
A GameplayEffect modifies attributes and tags.

I named this `GE_CharacterDefaults`, and opened it for editing. The screenshot below contains all settings I modified under the `Class Defaults` panel. If it isn't in this screenshot, I didn't change it.



Duration Policy	Instant
Modifiers	3 Array elements
Index [0]	6 members
Attribute	GAS_AttributeSet.Health
Modifier Op	Override
Modifier Magnituc	
Magnitude Calcul:	Scalable Float
Scalable Float Ma	100.0 / Use CurveTable...
Source Tags	
Target Tags	
Index [1]	6 members
Attribute	GAS_AttributeSet.Stamina
Modifier Op	Override
Modifier Magnituc	
Magnitude Calcul:	Scalable Float
Scalable Float Ma	100.0 / Use CurveTable...
Source Tags	
Target Tags	
Index [2]	6 members
Attribute	GAS_AttributeSet.AttackPower
Modifier Op	Add
Modifier Magnituc	
Magnitude Calcul:	Scalable Float
Scalable Float Ma	20.0 / Use CurveTable...
Source Tags	
Target Tags	

Then I opened my `BP_ThirdPersonCharacter` blueprint for editing and applied the following settings within the `details` panel. You should notice at this point that the `DefaultAttributeEffect` and `DefaultAttributes` in this screenshot are actually the components we exposed to the editor in our `ThirdPersonCharacter.h` file earlier with `UPROPERTY` and `EditDefaultsOnly`.


GAS

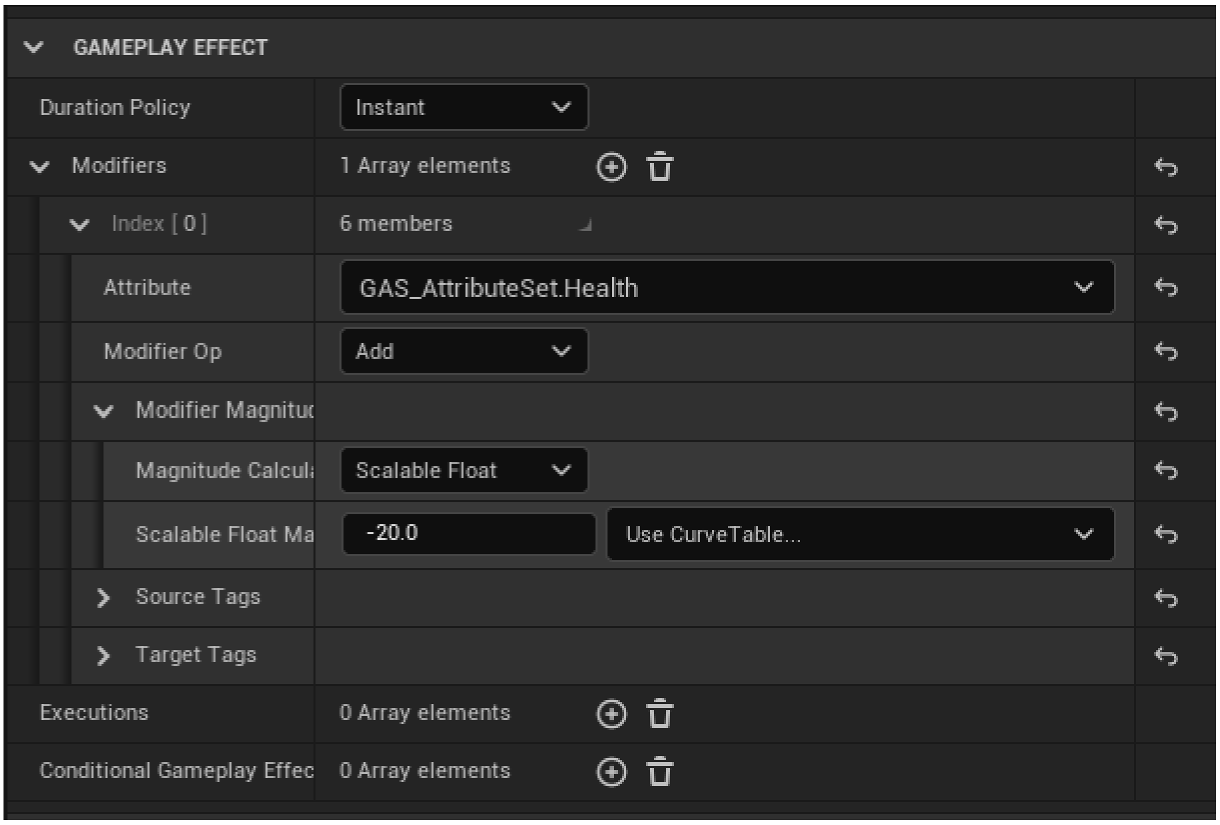
Default Attribute Eff — GE_CharacterDefaults

Default Abilities — 1 Array elements

Index [0] — GA_Attack

Damage Effect

To prove the system is working, create a new blueprint that derives from `GameplayEffect` and apply the settings below


GAMEPLAY EFFECT

Duration Policy — Instant

Modifiers — 1 Array elements

Index [0] — 6 members

Attribute — GAS_AttributeSet.Health

Modifier Op — Add

Modifier Magnituc

Magnitude Calcula — Scalable Float

Scalable Float Ma — -20.0 — Use CurveTable...

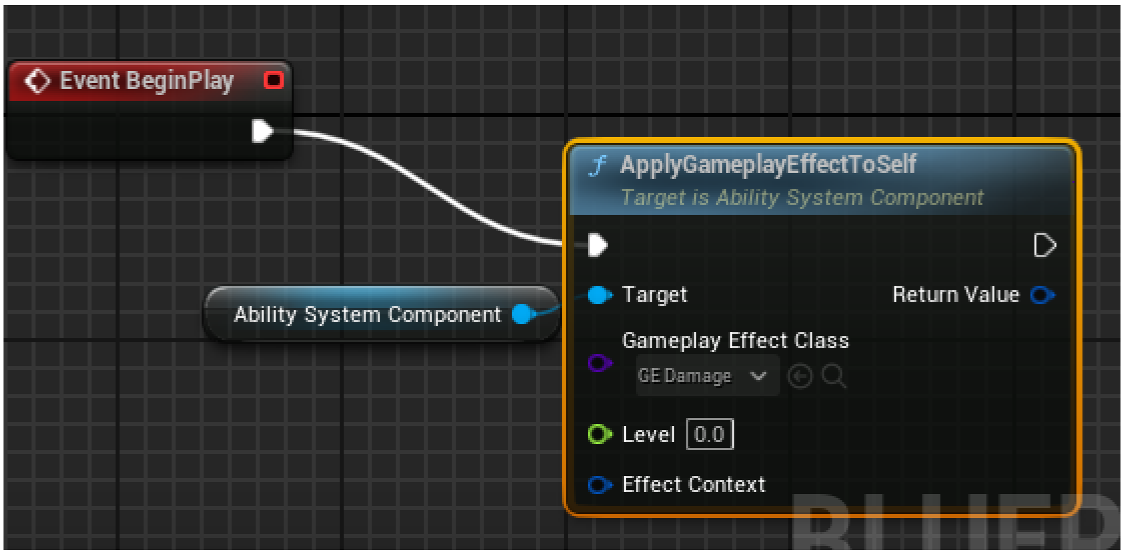
Source Tags

Target Tags

Executions — 0 Array elements

Conditional Gameplay Effec — 0 Array elements

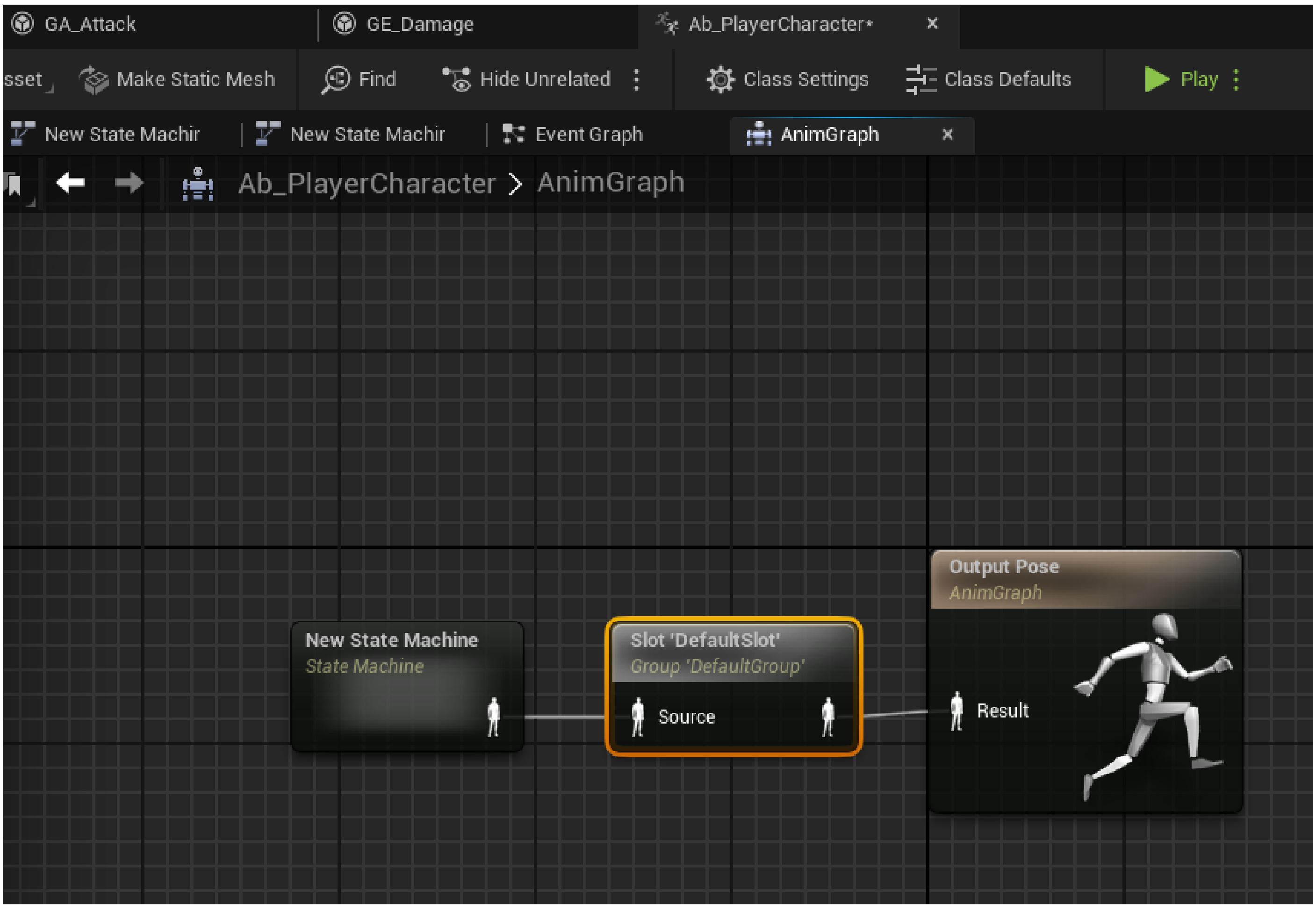
Then, in the event chart for your `BP_ThirdPersonCharacter` add a `BeginPlay` node and apply the damage when the game starts.



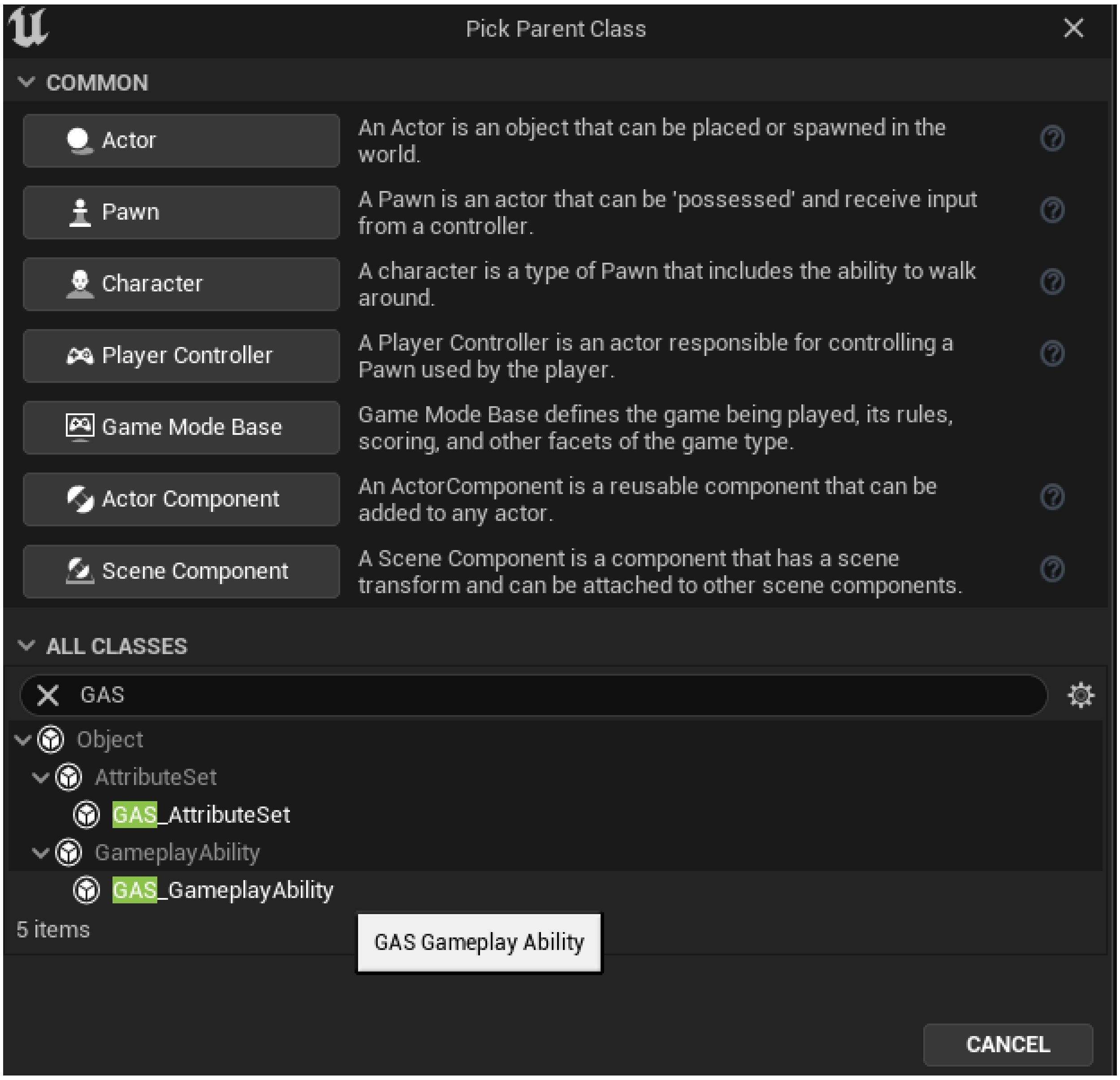
Hit play and then open a console and type `showdebug abilitysystem` to see that your HP should now be `80` on the lefthand side. Remove the damage to your player when youre done testing, but you can keep the `GE_Damage` asset around to use it later.

Attack Ability

First, open the animation blueprint for your character and add a montage `Slot 'DefaultSlot'` to your anim graph. For me, the screen looks like the below after the changes have been made. Make sure to save and apply these changes.



Make a new blueprint deriving from the `GAS_GameplayAbilitiy` class that we defined earlier.



An Actor is an object that can be placed or spawned in the world.

A Pawn is an actor that can be 'possessed' and receive input from a controller.

A character is a type of Pawn that includes the ability to walk around.

A Player Controller is an actor responsible for controlling a Pawn used by the player.

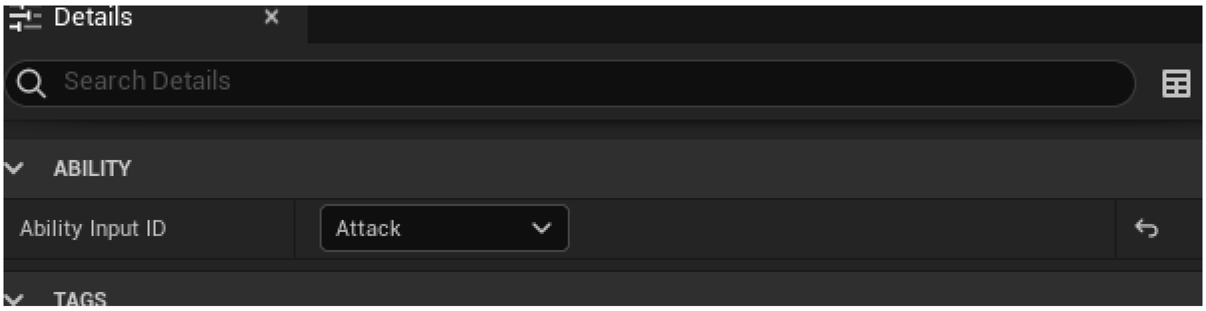
Game Mode Base defines the game being played, its rules, scoring, and other facets of the game type.

An ActorComponent is a reusable component that can be added to any actor.

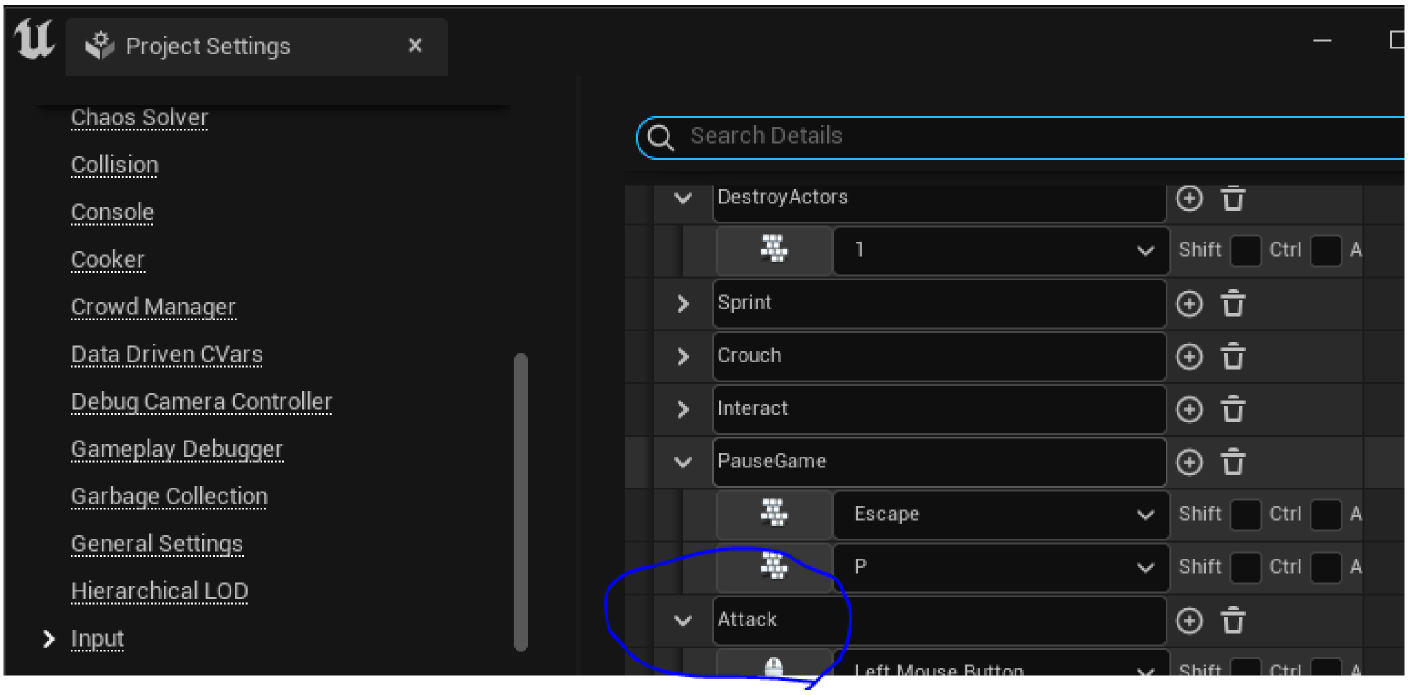
A Scene Component is a component that has a scene transform and can be attached to other scene components.

I named this `GA_Attack` and opened it for editing.

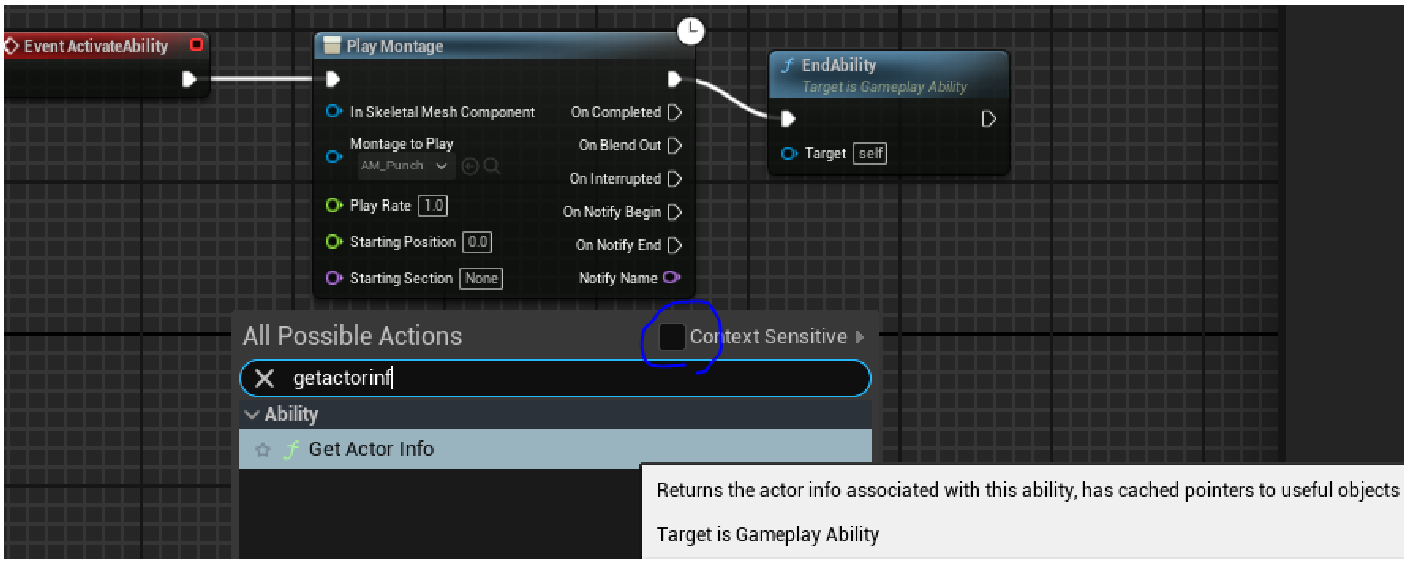
Make sure the `Ability Input ID` matches the input action we want the ability to be mapped to. This is found under the `details` panel.



Ability Input ID: Attack



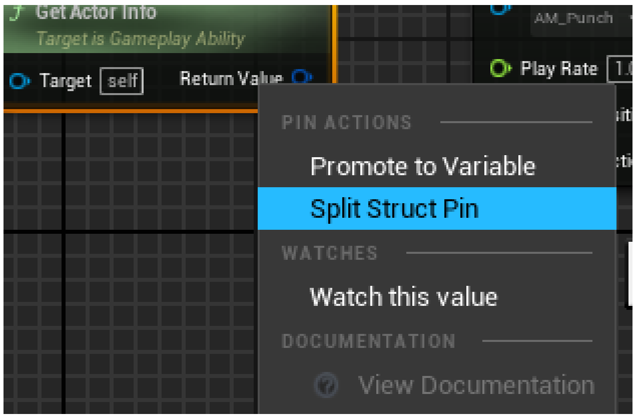
Next, open `GA_Attack` for editing and add the following blueprint nodes. Add the `GetActorInfo` node in the context menu in the screenshot, be sure to uncheck 'Context Sensitive' if it isn't appearing at first.



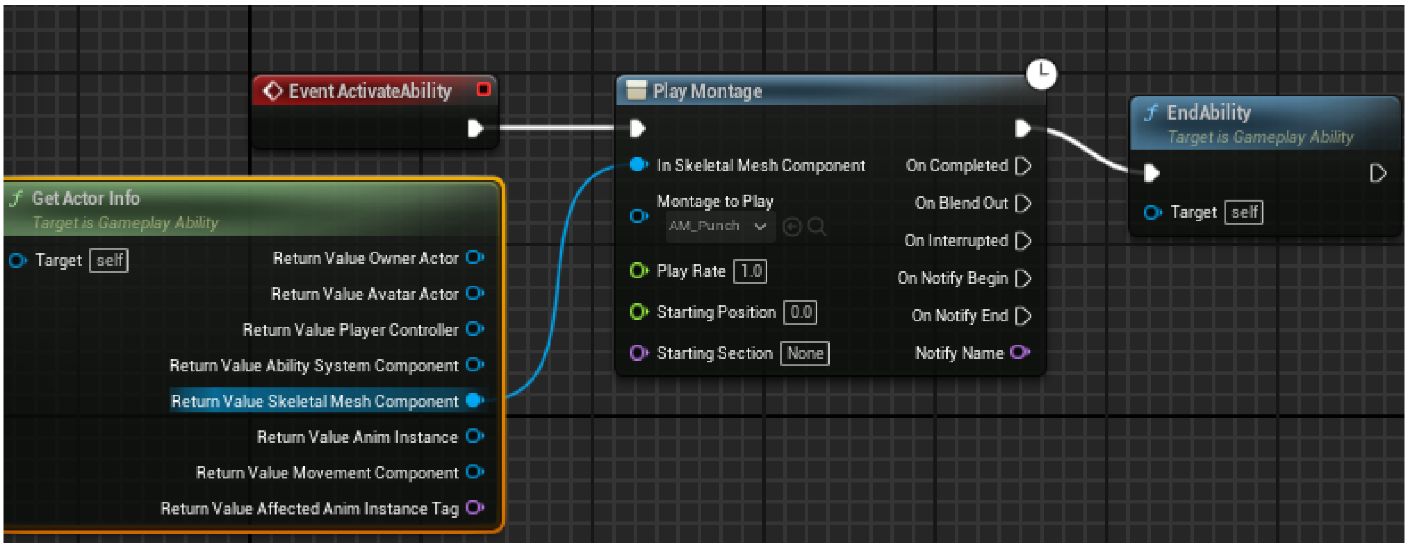
Returns the actor info associated with this ability, has cached pointers to useful objects

Target is Gameplay Ability

Now right click the `GetActorInfo` node and select `Split Struct Pin` to split the actor into its components



And connect the skeletal mesh pins to finish the blueprint for `GA_Attack`


Event ActivateAbility
Play Montage
In Skeletal Mesh Component
Montage to Play
AM_Punch
Play Rate 1.0
Starting Position 0.0
Starting Section None
On Completed
On Blend Out
On Interrupted
On Notify Begin
On Notify End
Notify Name
EndAbility
Target is Gameplay Ability
Target self
Get Actor Info
Target is Gameplay Ability
Target self
Return Value Owner Actor
Return Value Avatar Actor
Return Value Player Controller
Return Value Ability System Component
Return Value Skeletal Mesh Component
Return Value Anim Instance
Return Value Movement Component
Return Value Affected Anim Instance Tag

Under the `Montage To Play` pin on the `Play Montage` node, you may not have a montage available for your skeleton.

If you also don't have an animation, check out Mixamo for a free anmation and see the page on Retargeting Skeleton Animations

Then create a montage by watching this quick youtube video. If you're doing a simple punch animation, you probably just need to create a montage and click and drag the animation into the center of the screen and save. It's pretty simple, but you can use Motages to do some pretty neat things. Maybe for the first montage try making a one-off animation that doesn't loop like punching or a grab motion for interactions.

Once you have the montage made, select it here in this node, and then play the game. You'll now be able to see your character performing the attack!

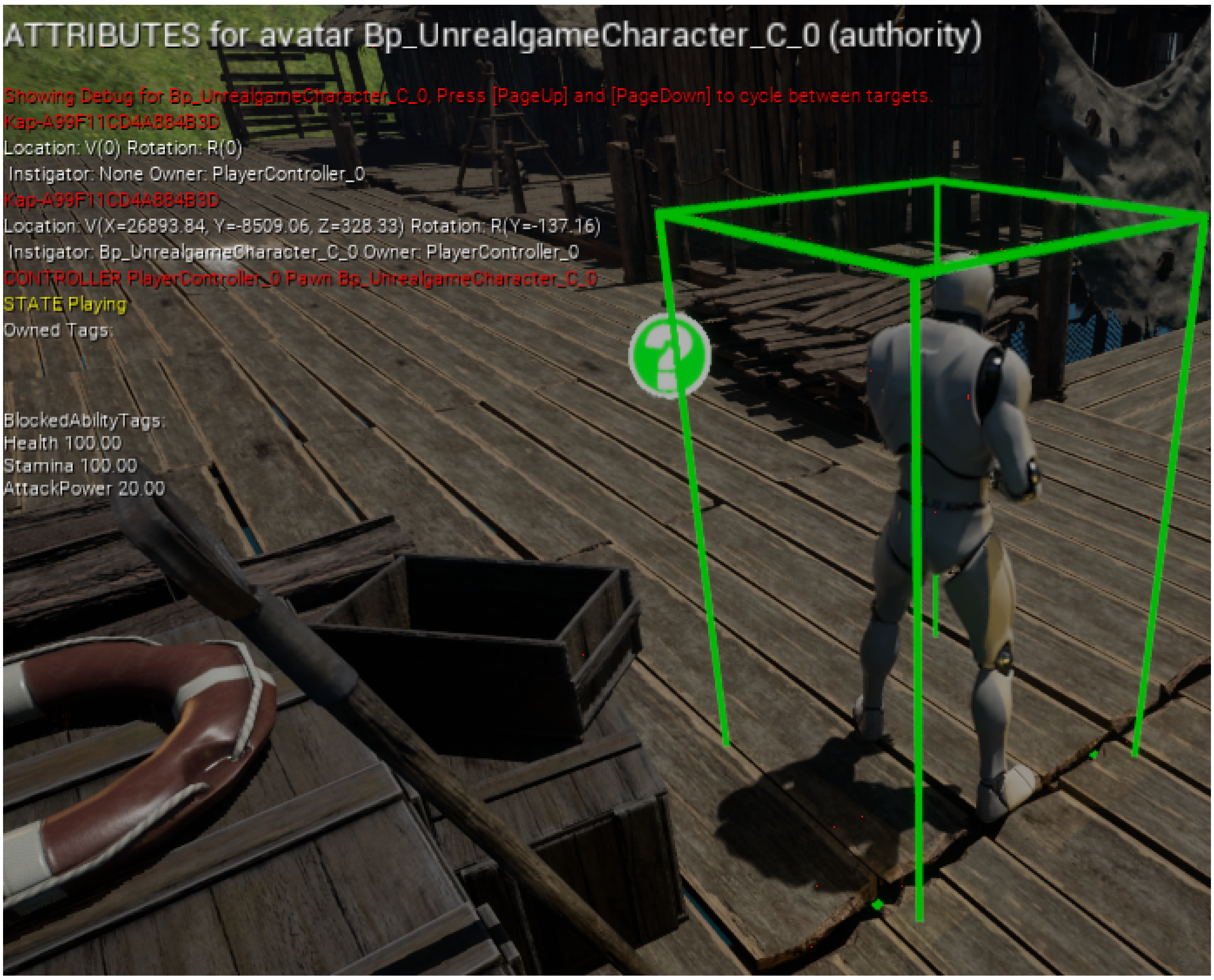
Additional Abilities

At a higher level, the steps for adding a new ability are below

1. Add the input action to the enum defined in `unrealgame5.h` (AKA `<YOUR_PROJECT_NAME>.h`)
2. Recompile in the editor or within the IDE of your choice
3. Restart the editor
4. Create a blueprint deriving from GAS_GameplayAbility
5. Define the `ActivateAbilitiy` event for this new blueprint, and assign an `Ability Input ID` within the `Details` panel
6. Create a new Animation Montage for the ability, if needed (and assign to play on `ActivateAbility` event)
7. Grant the abilitiy under the `Details` tab in the `Default Abilities` section while editing the `BP_ThirdPersonCharacter` blueprint
8. Add the input action to your project under `Edit->Input` if it doesn't already exist

Debugging

To see useful information on the GAS, enter play mode and hit the tilde (~) key to open a console. Then, type `showdebug abilitysystem`, and youll notice you can see your character stats even if there's no UI elements to represent them yet.



```
ATTRIBUTES for avatar Bp_UnrealgameCharacter_C_0 (authority)

Showing Debug for Bp_UnrealgameCharacter_C_0, Press [PageUp] and [PageDown] to cycle between targets.
Kap-A99F11CD4A884B3D
Location: V(0) Rotation: R(0)
 Instigator: None Owner: PlayerController_0
Kap-A99F11CD4A884B3D
Location: V(X=26893.84, Y=-8509.06, Z=328.33) Rotation: R(Y=-137.16)
 Instigator: Bp_UnrealgameCharacter_C_0 Owner: PlayerController_0
CONTROLLER PlayerController_0 Pawn Bp_UnrealgameCharacter_C_0
STATE Playing
Owned Tags:


BlockedAbilityTags:
Health 100.00
Stamina 100.00
AttackPower 20.00
```

Revision #7
Created 19 January 2022 15:57:58 by Shaun Reed
Updated 21 August 2022 22:34:59 by Shaun Reed