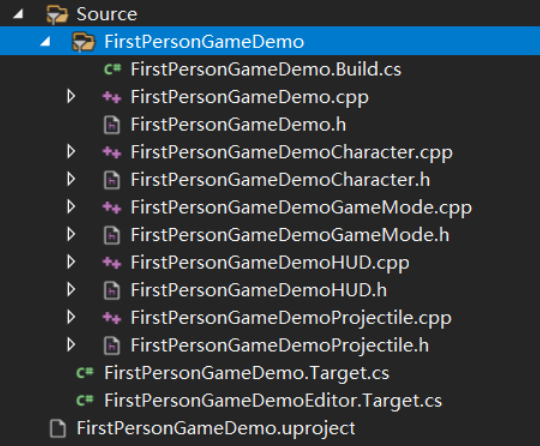
此处详细分析第一人称游戏demo里面的代码：

主要有以下代码文件，逐一进行分析：



FirstPersonGameDemo.Build.cs文件：  
引擎被看作是由许多模块(Modules)搭建起来的，这个游戏工程也维护自己的模块并改变里面的实现，每个模块都包含了一些公用的接口和编译环境，都有一个C#的源文件Bulid.cs文件，这个文件后面的C++文件就是属于这个模块，Build.cs文件主要是被UnrealBuildTool所使用，确定这个模块的环境依赖情况。

参数的具体描述可见ue的官方网站：

<https://docs.unrealengine.com/en-US/ProductionPipelines/UnrealBuildSystem/ModuleFiles/index.html>

关于ModuleRules这个类的源码可见ModuleRules.cs文件

这个demo里的代码如下：

using UnrealBuildTool;

public class FirstPersonGameDemo : ModuleRules

{

public FirstPersonGameDemo(ReadOnlyTargetRules Target) : base(Target)

{

PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;

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

}

}

PCHUsage：设置此模块预编译头文件的用法为显式或者共享，并且支持IWYU(Include-What-You-Use)

PublicDependencyModuleNames：设置此模块需要的一些公共依赖模块的名字，会自动include公有和私有的文件

FirstPersonGameDemo的头文件和源文件：

在源文件里面注册为主模块：IMPLEMENT\_PRIMARY\_GAME\_MODULE

在头文件只进行了：#include "CoreMinimal.h"

FirstPersonGameDemoCharacter的头文件和源文件：

头文件里：

在前面include一些头文件，包括"CoreMinimal.h"和"XXX.generated.h"（这个头文件是自动生成的，只要里面含有UCLASS()，标志着这个类是一个游戏性的类）,然后声明了一些需要用到的类。

开始声明自己的一个类，如果类继承自UObject，就需要在类声明前一行加入：UCLASS()，以及在类体内第一行加入GENERATED\_BODY()或者GENERATED\_UCLASS\_BODY()，前者需要实现无参数的构造函数， 且默认成员是private的。后者需要实现带有const FObjectInitializer&参数的构造函数。

下面就开始声明各种成员和成员函数，如果成员前面使用了UPROPERTY宏，那么这个成员变量就可以被复制、序列化、蓝图访问、GC时的引用计数。UPROPERTY的里面的一些参数设置可见ObjectMacros.h文件里面，比如有VisibleAnywhere/EditAnywhere设置可见性和可编辑性，Category=设置在窗口显示时的分栏，BlueprintReadOnly设置蓝图的操作权限等等

下面详细说一下这个类里面的成员和成员函数，简单起见，把UPROPERTY宏设置都删去了：

UCLASS(config=Game)

class AFirstPersonGameDemoCharacter : public ACharacter

{

GENERATED\_BODY()

USkeletalMeshComponent\* Mesh1P; // 仅自己可见的手臂模型

USkeletalMeshComponent\* FP\_Gun; // 枪的模型，使用USkeletalMeshComponent都是被渲染且有动画的

USceneComponent\* FP\_MuzzleLocation; // 枪口产生的位置，使用USceneComponent类，没有渲染和碰撞功能

UCameraComponent\* FirstPersonCameraComponent; // 相机组件类，设置相机参数

public:

AFirstPersonGameDemoCharacter(); // 无参数的构造函数声明

protected:

virtual void BeginPlay(); // BeginPlay()的消息，从UWorld传到AActor，再dispatch其各个子类，在void UEditorEngine::RequestPlaySession改变PlaySessionRequest（TOptional类型）然后代表开始游戏

public:

float BaseTurnRate; // 设置Yaw方向上的转向速率

float BaseLookUpRate; // 设置Pitch方向上的俯仰速率

FVector GunOffset; // 枪口与角色位置的偏移

TSubclassOf<class AFirstPersonGameDemoProjectile> ProjectileClass; // TSubclassOf的作用是指定这个对象的父类必须是尖括号里这个，也就是在属性窗口里面选择的时候，只会出现括号里面类的派生类的选项，另外还提供一些安全性

USoundBase\* FireSound; // 开火时的音效对象

UAnimMontage\* FireAnimation; // 开火时的动画

protected:

void OnFire();

void MoveForward(float Val);

void MoveRight(float Val);

void TurnAtRate(float Rate);

void LookUpAtRate(float Rate);

struct TouchData // 与触摸控制相关的成员

{

TouchData() { bIsPressed = false;Location=FVector::ZeroVector;}

bool bIsPressed;

ETouchIndex::Type FingerIndex;

FVector Location;

bool bMoved;

};

void BeginTouch(const ETouchIndex::Type FingerIndex, const FVector Location);

void EndTouch(const ETouchIndex::Type FingerIndex, const FVector Location);

void TouchUpdate(const ETouchIndex::Type FingerIndex, const FVector Location);

TouchData TouchItem;

protected:

virtual void SetupPlayerInputComponent(UInputComponent\* InputComponent) override; // APawn的接口

bool EnableTouchscreenMovement(UInputComponent\* InputComponent); // 决定是否开启触摸控制，并进行绑定

public:

USkeletalMeshComponent\* GetMesh1P() const { return Mesh1P; } // 获取自身的模型对象

UCameraComponent\* GetFirstPersonCameraComponent() const { return FirstPersonCameraComponent; } // 获取相机对象

};

源文件里：主要对头文件里这个类的成员函数进行实现：

构造函数：

AFirstPersonGameDemoCharacter()：进行成员初始化相关操作。主要有：  
获取父类Character的CapsuleComponent并初始化其大小；  
设置BaseTurnRate、BaseLookUpRate、GunOffset；  
初始化相机组件，并Attachment到CapsuleComponent上面

初始化自身的模型Mesh1P，并Attachment到相机组件上  
初始化枪的模型FP\_Gun，并Attachment到Actor的RootComponent根节点上面

初始化枪口位置对象，并Attachment到FP\_Gun上

这个类实例的各种骨骼、图元、动画等属于内容包里面的部分，是在蓝图类里面进行指定，避免在C++代码里面对内容产生引用

虚函数：

BeginPlay ：在AActor::DispatchBeginPlay中被调用，里面第一句必须写Super::BeginPlay()；在这里将FP\_Gun Attachment到Mesh1P上面，为什么没在构造函数里面做呢？因为在那时，Mesh1P的骨骼还没有生成

SetupPlayerInputComponent，对传入的UInputComponent\*进行修改，绑定一些输入性的动作：如BindAction，BindAxis，以及在EnableTouchscreenMovement里面的BindTouch，把这些输入名称与相应的成员函数进行绑定，如PlayerInputComponent->BindAction("Jump", IE\_Pressed, this, &ACharacter::Jump);就是把Jump这个名称的输入与ACharacter::Jump函数绑定起来，在检测到相应输入时，就会调用对应函数。而这些带有名字的输入可以在：编辑器-编辑-项目设置-引擎-输入 里面进行相应设置。此demo的一些绑定例子：BindAction：Jump-空格，Fire-鼠标左键；BindAxis：MoveForward-WS，MoveRight-AD，Turn-鼠标X，LookUp-鼠标Y

其他成员函数：

OnFire：里面的核心是World->SpawnActor这个函数，在world里面生成子弹，传入子弹类、生成位置、生成姿态矩阵（即目前的摄像机矩阵，猜想：可以获取子弹方向）、生成配置参数。其他的实现有：是否播放声音、是否播放Mesh1P动画等

MoveForward：传入的值是1或者-1，核心调用父类的函数APawn::AddMovementInput，调用USceneComponent::GetForwardVector()获取当前Scence-to-world的变换矩阵作用在（1，0，0）后的结果，其含义为当前Scene空间的X轴（即前向方向）在世界坐标系下的向量坐标表示，经打LOG后验证是一个三维向量，但是z值为0，xy范围在(-1,1)，将向前的这个向量与1或者-1相乘后再加到APawn:: ControlInputVector上面去，此变量作为一个消耗性的变量，在取用一次后就会被置为0。但是如果在 编辑-项目设置-输入 里面设置的值很大，此处对于速度并没有起到帮助，因为下面这句话，会对Input进行Clamp：

GetMaxAcceleration() \* InputAcceleration.GetClampedToMaxSize(1.0f)

但是如果设置得很小，比如0.1，确实会行走得很慢

MoveRight：原理同MoveForward，只不过调用的是USceneComponent::GetActorRightVector()获取的是此场景变换矩阵作用在Y轴基矢量后的结果，即向右方向在世界坐标系下的向量坐标

TurnAtRate：根据输入的Rate和BaseTurnRate以及GetWorld()->GetDeltaSeconds()的乘积来调用APawn::AddControllerYawInput对Yaw的值APlayerController:: RotationInput.Yaw进行更改，注意此函数与MoveForward的原理有差别，此函数是每个Tick都在执行，并对Yaw进行累加，把累加值给APlayerController:: RotationInput.Yaw。注意此函数实际不是真正与鼠标绑定的函数，而是APawn::AddControllerYawInput与鼠标绑定，并且此绑定的输入值不再是离散的1或者-1，而是连续值，根据鼠标移动情况而定

LookUpAtRate：原理同TurnAtRate一样，只不过改变的是Pitch。同样与鼠标Y实际绑定的是APawn::AddControllerPitchInput

BeginTouch

EndTouch

EnableTouchscreenMovement