# 游戏基本内容

游戏开始玩家控制的为第三人称角色，基本流程为：

（1）踏上电梯后，电梯上升，到达炮台所在房间；（已完成）

（2）靠近大门，门自动开启，玩家进入；（已完成）

（3）在炮台范围内按下“F”键，玩家控制对象变为炮台；（已完成）

（4）点击鼠标左键，发射子弹；（已完成）

（5）界面右上角显示有子弹数量，每发射一次，子弹数量减1，当子弹数量为0后，无法继续发射；（已完成）

（6）在地面上，每隔两秒生成一个NPC，朝炮台方向移动；（已完成）

（7）击中NPC后，NPC变色，击中三次后，NPC销毁；

（8）再次按下“F”，玩家控制对象转回第三人称角色；（已完成）

（9）角色通过上下浮动的平台踏上屋顶；（已完成）

（10）在屋顶放置弹药补充资源，玩家通过拾取该资源重置炮台弹药数。

**目前进度**：

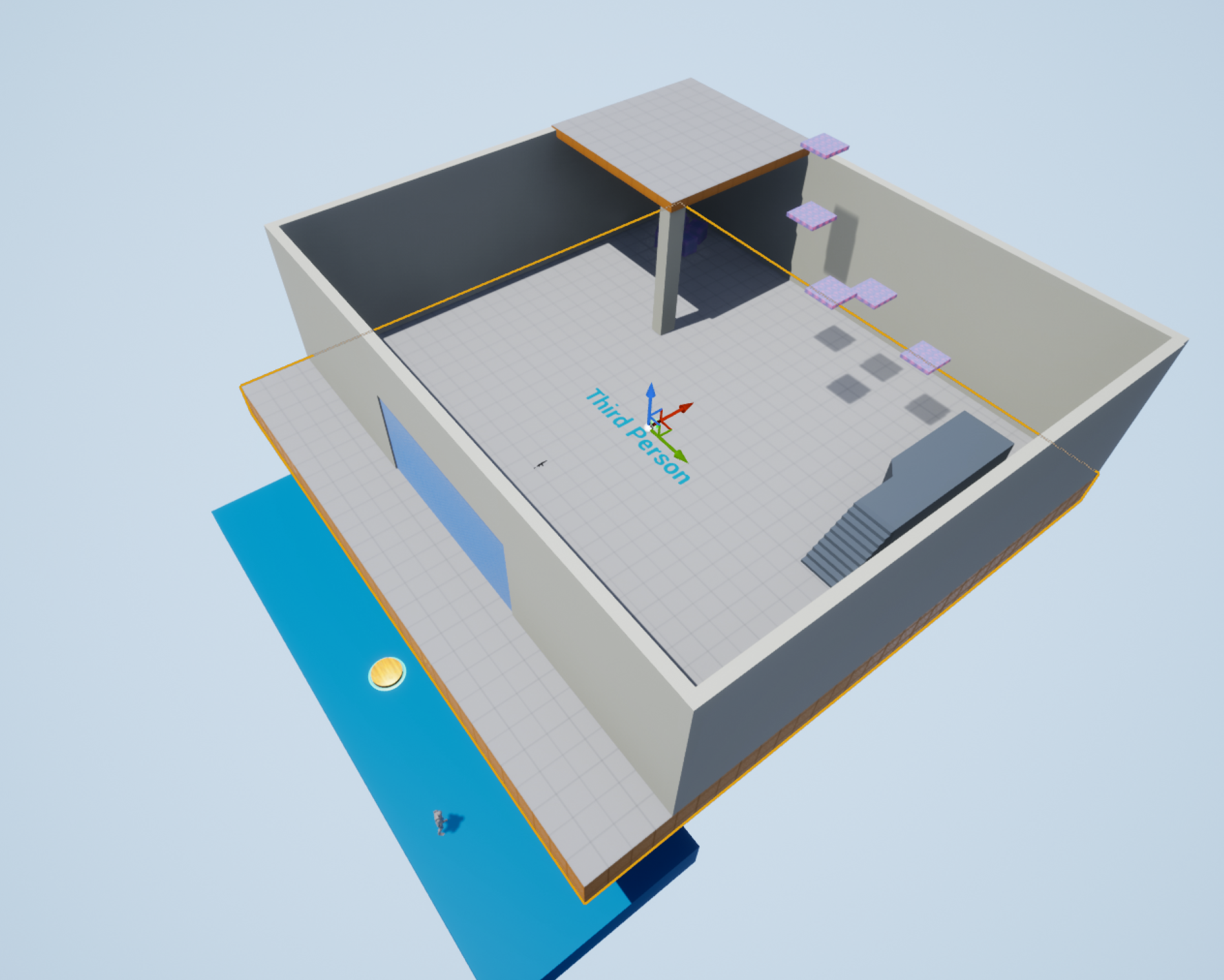
（a）已完成（1）—（6），（7）中NPC的销毁以及（8）、（9）；

（b）（7）中的变色、（10）还未完成。

**下阶段计划：**

（a）完善上述功能，完成（7）与（10）；

（b）在此基础上进行改进完善，构建可联网的多人游戏。



# 电梯设计(Elevator)

## 功能

角色踏上电梯后，电梯自动上升、下降。

## 基本组件

静态网格体组件（UStaticMeshComponent）：用于显示电梯平台；

盒体组件（UBoxComponent）：用于检测角色与电梯的重叠事件；

灯光组件（ULightComponent）：人在电梯上时灯亮；

时间轴相关的组件（UTimelineComponent、UCurveFloat等）

## 实现

（1）使用UBoxComponent组件检测角色是否踏上电梯，通过为其绑定重叠事件，实现电梯上升、下降事件的触发；

（2）电梯的运动通过时间轴组件（UTimelineComponent）实现，通过为其设定变化曲线来调整电梯平台的位置；

（此部分参考官方文档中的：[开门 | 虚幻引擎文档 (unrealengine.com)](https://docs.unrealengine.com/4.27/zh-CN/ProgrammingAndScripting/Blueprints/UserGuide/Timelines/Examples/OpeningDoors/) 案例）

（3）电梯应具有上升、下降两个功能，通过检测“重叠事件”发生时电梯是否位于初始位置，来决定是通过时间轴的Play()实现电梯上升，还是通过时间轴的Reverse()进行反向播放，实现电梯下降。

# 自动门的设计(Door)

## 功能

此处设计的为横向打开的自动门，角色靠近门时，门自动打开；远离门时，自动关闭。

## 基本组件

两个静态网格体组件（UStaticMeshComponent），表示左右两扇门；

盒体组件（UBoxComponent）：用于检测角色是靠近门还是远离。

## 参数

（1）float Opentime; 表示门经过多长时间完全打开，关系到开门的速度，由用户输入。

（2）bool IsDoorOpen; 表示门是要执行打开操作还是关闭操作。

## 实现思路

（1）记录左右两扇门的初始Y坐标；

（2）以两扇门Y方向上的差值作为门打开的最大宽度DoorOpenWidth；以此为依据，得到左右两扇门的极限位置；

（3）求取门打开的速度：DoorOpenWidth/Opentime；

（4）为盒体组件初始化重叠事件，开始重叠时，执行OnBoxOverlap()函数，将IsDoorOpen设置为true，表示门应打开；同时，通过 SetActorTickEnabled (true) 函数开启每帧执行Tick()；

（5）结束重叠时，执行OnBox EndOverlap()函数，将IsDoorOpen设置为false，表示门应关闭；同时，通过SetActorTickEnabled(true) 函数开启每帧执行Tick()；

（6）在Tick()函数中，通过判断IsDoorOpen决定是执行开门函数DoorOpen(float DeltaTime)，还是关门函数DoorClose(float DeltaTime)；

（7）在门完全打开或完全关闭后，执行SetActorTickEnabled(false)关闭每帧执行Tick()。

# 第三人称角色设计(MyFPSProjectCharacter)

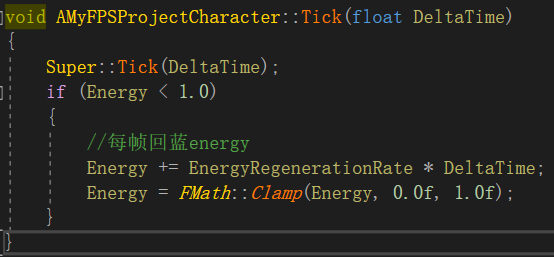
第三人称角色以UE4中“第三人称游戏模板”中自带的角色为基础，进行了以下修改：

## 增加成员变量：Health、Energy、HasAmmo

为第三人称角色设计了血条、蓝条，用float型变量Health表示血量，Energy表示蓝条。布尔变量HasAmmo表示其是否携带弹药包，涉及后续的弹药更新。

（1）**Energy**

进行跳跃、加速跑等操作时，会消耗Energy，当Energy低于相应阈值时，无法跳跃、加速跑。此外，Energy会以一定速率（float EnergyRegenerationRate）自动回复，回复的实现方式为：

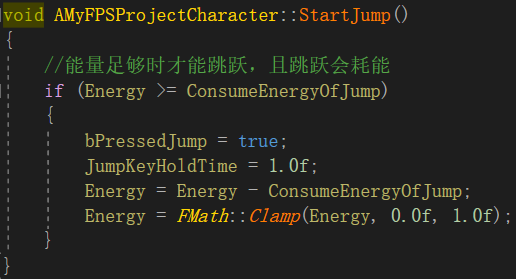


（2）**Health**

Health的销毁方式与回复方式暂时还未设计。

## 跳跃设置

UE4模板中的第三人称角色，跳跃（即空格按下）绑定的事件是其父类ACharacter的Jump函数，为实现“跳跃时Energy”减少这一功能，重新设计了跳跃函数——StartJump，其实现方式为：



其中，“ConsumeEnergyOfJump”是float型变量，表示每跳跃一次所消耗的Energy。

## 加速跑设置

UE4模板中的第三人称角色中继承了ACharacter的角色移动组件：UCharacterMovementComponent。因此，通过修改该组件中的参数MaxWalkSpeed来调整角色前进速度，实现加速跑。

具体实现方式为：

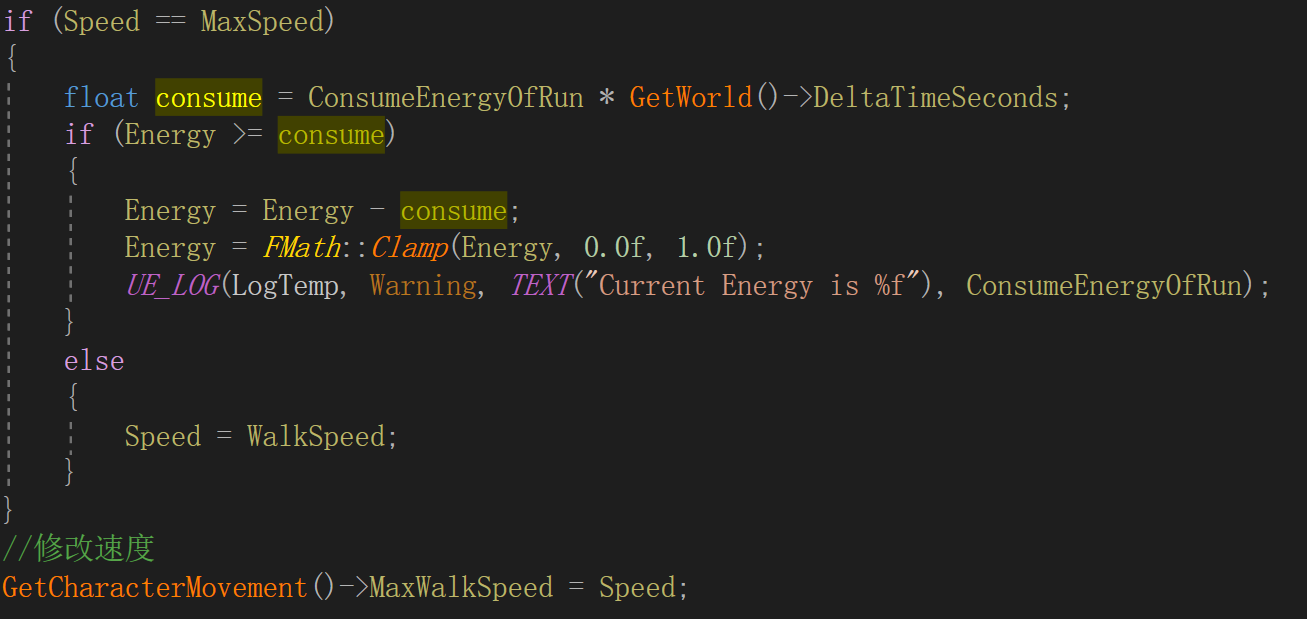
（1）在UE4“项目设置”-“输入”中创建“Run”，其绑定的按键为LeftShift。

（2）为角色添加MaxSpeed、WalkSpeed两个float型变量，分别表示加速跑与正常行走的速度；添加float Speed表示角色当前速度，初始值为WalkSpeed。

（3）设计Run()、StopRun()函数，与“Run”即LeftShift的按下与松开绑定；

（4）Run()中设置**Speed=MaxSpeed**，StopRun()中设置**Speed=WalkSpeed**。

（5）在前进函数MoveForward(float Value)中添加以下代码：

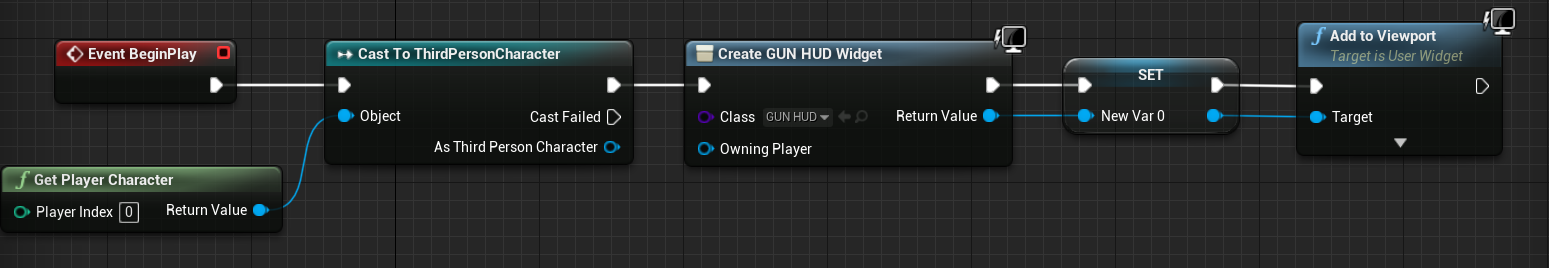


其中，ComsumEnergyOfRun表示加速跑状态下每秒消耗的Energy。

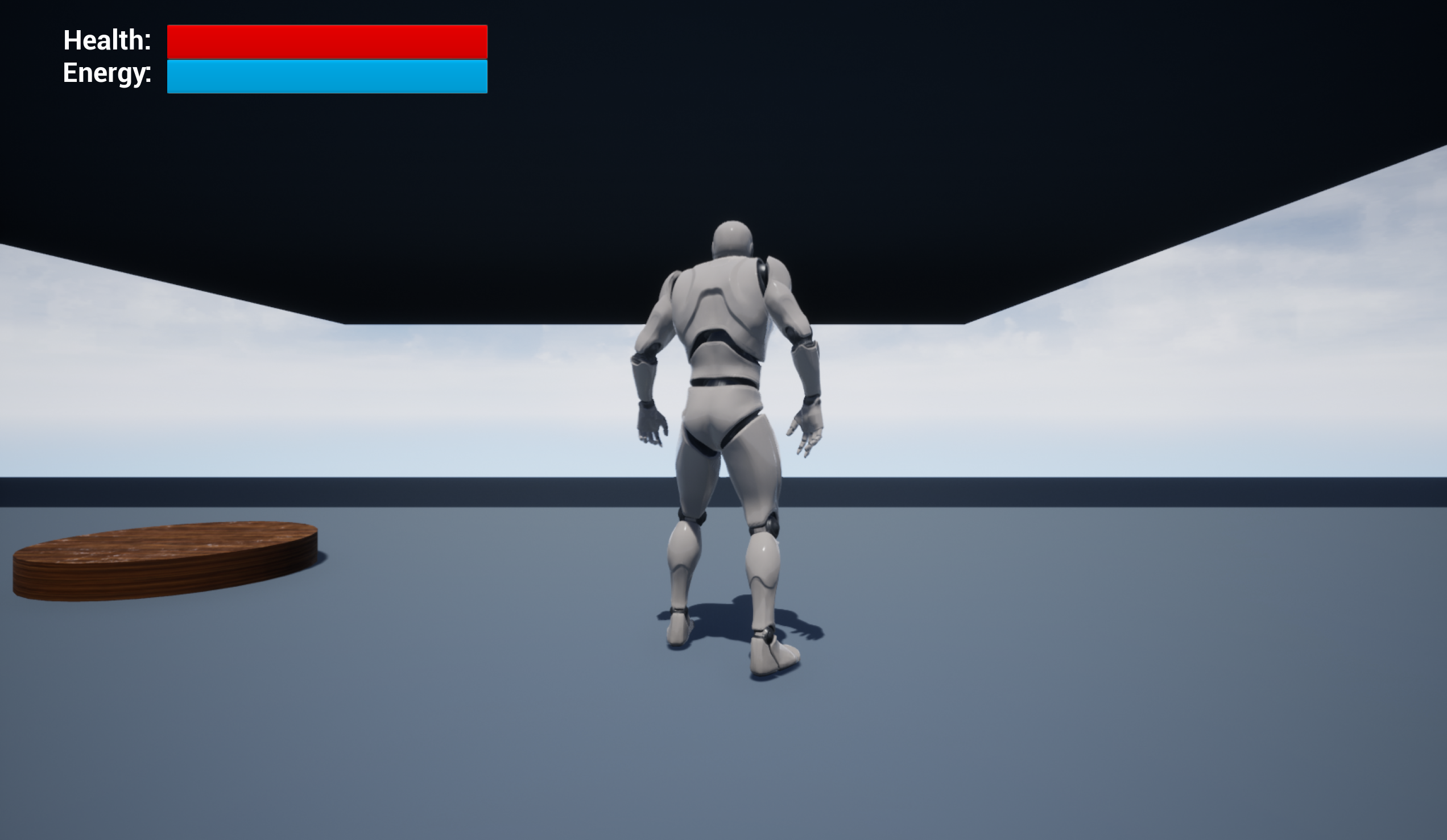
## 血条、蓝条显示

为在游戏中显示角色的Health与Energy，设计了相应的UI界面（UI名称：GUN\_HUD）。UI的设计参考[UMG UI设计器快速入门 | 虚幻引擎文档 (unrealengine.com)](https://docs.unrealengine.com/4.27/zh-CN/InteractiveExperiences/UMG/QuickStart/)。

UI界面的显示是用蓝图实现的，（因为玩家控制的角色会切换为炮台，因此UI界面不是一直显示的，只有在当前控制的角色是第三人称角色时才会显示）对应显示逻辑为：



相应的界面为：



# 炮台设计(FPSGun)

## 功能

炮台应具有的功能为：

（1）开火（Fire）功能：发射炮弹以攻击敌人；

（2）移动方面：炮台只应具有旋转功能，不可移动；

（3）瞄准功能：炮台应具备开镜功能以及准星显示，以瞄准敌人。

## 基本组件

（1）UStaticMeshComponent：用于显示炮台外观；（此处用的枪的外观）

（2）UCameraComponent：摄像机组件，充当玩家在游戏中的“眼睛”；

（3）USpringArmComponent：弹簧臂组件，连接摄像机与角色，使摄像机与角色的距离的变化更加平缓，且可动态调整；

（4）UBoxComponent：用于检测重叠事件（角色是否在附加）；

（5）表示发射物类型的成员：

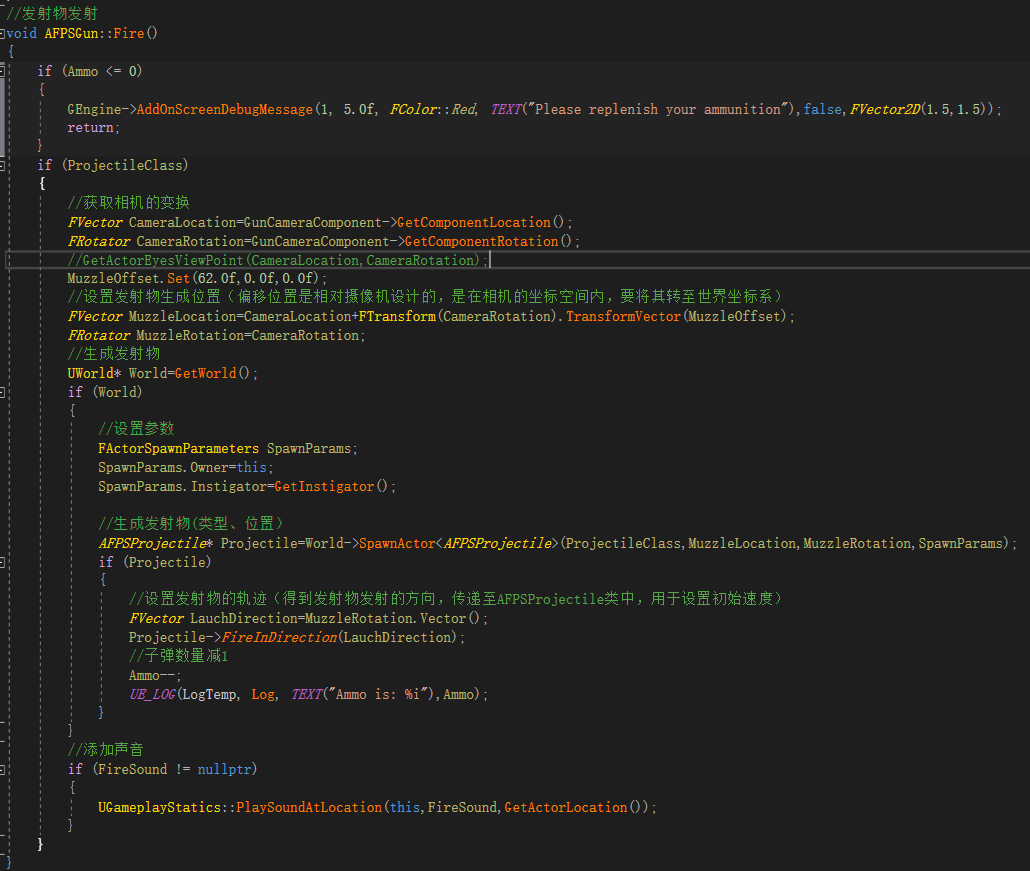
UPROPERTY(EditDefaultsOnly, Category=Projectile)

TSubclassOf<class AFPSProjectile> ProjectileClass;

（6）USoundBase：开火的同时，播放声音（PlaySoundAtLocation()）

## 开火功能

通过Fire()函数实现发射物（炮弹）的生成，Fire()函数与鼠标左键点击事件绑定。具体代码如下图所示。



其中：

（1）Ammo为弹药数，每发射一次炮弹，Ammo要减1；该值等于0时，不再生成炮弹。

（2）FVector MuzzleOffset表示子弹生成位置相对于摄像机位置的偏移量，此偏移量是在摄像机的坐标空间中设定的，而在调用SpawnActor()生成发射物时，应提供世界坐标系下的位置，因此，要通过

FTransform(CameraRotation).TransformVector(MuzzleOffset)

将其转换为世界坐标系下的偏移坐标。

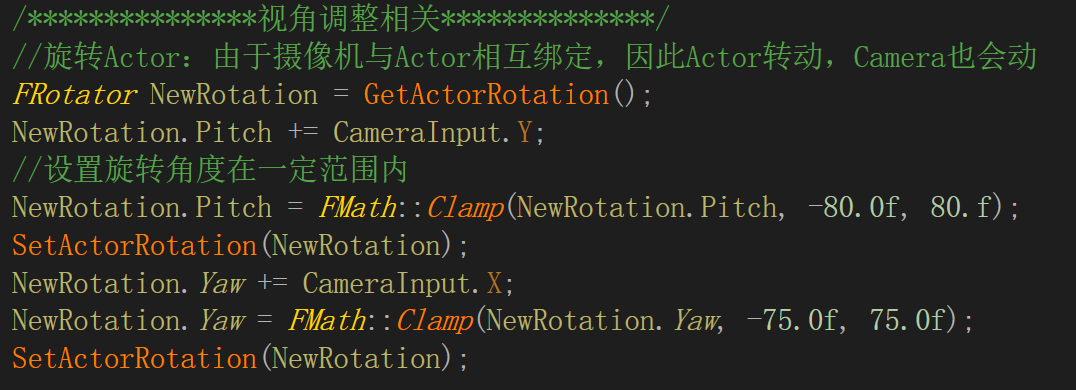
## 炮台与摄像机旋转

旋转事件通过监测鼠标输入：“Mouse X”、“Mouse Y”实现。

在“项目设置”-“输入”中配置输入事件：“Trun”（Mouse X）、“LookUp”（Mouse Y），分别与函数PitchCamera(float AxisValue)、YawCamera(float AxisValue)绑定。

实现思路：

用*FVector2D* CameraInput记录炮台（摄像机）在Yaw、Pitch两个方向应旋转的角度；在PitchCamera(float AxisValue)、YawCamera(float AxisValue)函数中对变量CameraInput进行修改，在Tick()函数中调整炮台与摄像机的角度。Tick()中旋转相关的代码为：



## 开镜

开镜是通过调整摄像机视角范围（FieldOfView）实现的。

点击鼠标右键实现开镜，松开鼠标右键开镜结束。与鼠标右键松开、按下绑定的函数为：ZoomIn()、ZoomOut()。这两个函数通过修改布尔型变量IsZoom来标记是否进行开镜操作，而开镜的具体实现在Tick()函数中。

Tick()函数中的相应代码为：



代码中，Zoomtime为float型变量，表示摄像机视野角度经过多长时间从90°变为30°，该变量使开镜更为平滑。

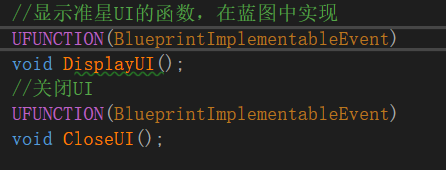
## 准星

用蓝图实现准星与弹药数量的显示。准星是通过制作UI界面的方式创建的。参考：[UMG UI设计器快速入门 | 虚幻引擎文档 (unrealengine.com)](https://docs.unrealengine.com/5.0/zh-CN/umg-ui-designer-quick-start-guide/)



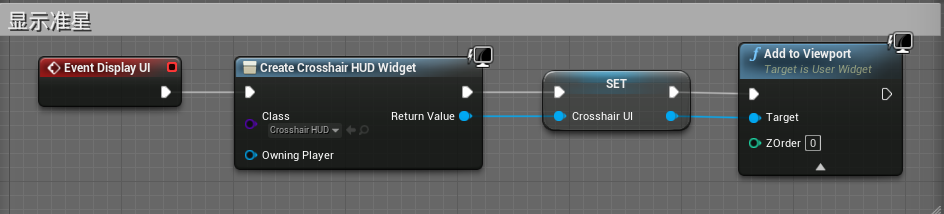
具体显示方式为：

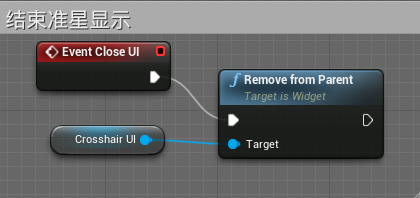
（1）在炮台类（FPSGun）中声明两个在蓝图中实现的函数，用于控制准星、弹药数的显示与隐藏：



（2）在控制器的控制对象由第三人称角色转为炮台时，执行DisplayUI()；控制对象转回第三人称角色时，执行CloseUI()。（具体调用位置在进行控制对象切换时说明）

两个函数在蓝图中的表现形式为：





# 发射物设计（FPSProjectile）

## 基本组件

发射物为炮台发射出去的炮弹，其基本组件包括：

（1）USphereComponent：检测发射物的碰撞；

（2）UProjectileMovementComponent：用于控制发射出去的发射物的运动；

（3）UStaticMeshComponent：显示发射物；

（4）UMaterialInstanceDynamic：发射物的材质；

（5）UParticleSystem：在射中物体时，播放粒子效果。

## 发射

发射相应的函数为：void FireInDirection(const *FVector*& ShootDirection)

由于发射物的发射速度、反弹等状态是由UProjectileMovementComponent组件控制的，因此，在初始化UProjectileMovementComponent组件后，只需设置发射物初始速度（矢量：包括速度大小与方向）即可实现发射物的射出。

## 击中事件

发射物会与其他物体发生碰撞，在击中其他物体后触发碰撞事件，对被击中物体施加冲击力，并播放粒子效果。



# 上下移动的平台（FloatingPlatform）

## 作用

该平台的作用为：玩家通过多个上下移动的平台到达屋顶。

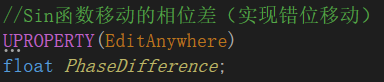
## 实现

具体代码与[编程快速入门 | 虚幻引擎文档 (unrealengine.com)](https://docs.unrealengine.com/4.27/zh-CN/ProgrammingAndScripting/ProgrammingWithCPP/CPPProgrammingQuickStart/)相似。

所作修改为：

（1）改平台以正弦函数的规律在Z方向进行移动；导致的问题是：存在多个平台对象时，他们的运动规律是一样，即同时上升、同时下降；

（2）基于（1），添加了变量：PhaseDifference，表示相位偏移；通过为不同实例化对象赋予不同的PhaseDifference值，在计算Sin()时加上该偏移，可实现不同对象的异步移动。



# 控制对象切换

## 功能

在玩家到达炮台范围内，且按下“F”键后，控制器控制的对象由第三人称角色切换为炮台，再次按下“F”，控制对象切换为第三人称角色。

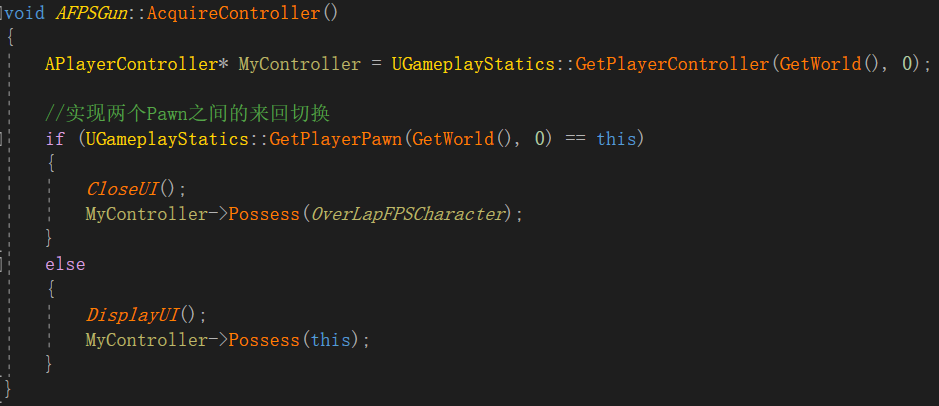
## 实现思路

控制对象的切换主要用到Possess()函数，其基本流程为：

（1）炮台（FPSGun）中有UBoxComponent组件，当检测到开始重叠，且重叠对象是第三人称角色时，将“F”按下这一事件与“调用角色切换”的函数绑定；并更新弹药数量。（同时保存记录此时的第三人称角色，在后续切换回第三人称角色时使用）



（2）玩家按下“F”键后，调用“AcquireController()”函数，该函数的作用是实现的控制对象的切换；（在此部分实现炮台与第三人称角色间的来回切换，即再次按下“F”可停止控制炮台）



（3）玩家控制第三人称角色离开炮台碰撞范围后，解除“F”按下与控制对象切换函数之间的绑定关系，此操作的目的为：防止玩家在远离炮台的位置按下“F”，进而控制炮台。（炮台的控制只能在炮台附加实现）



# NPC设计 (FPSNPC)

## 功能

（1）追踪玩家位置，向靠近玩家的方向移动；

（2）NPC被子弹击中三次后销毁。

## 基本组件与变量

（1）UStaticMeshComponent：显示NPC外观；

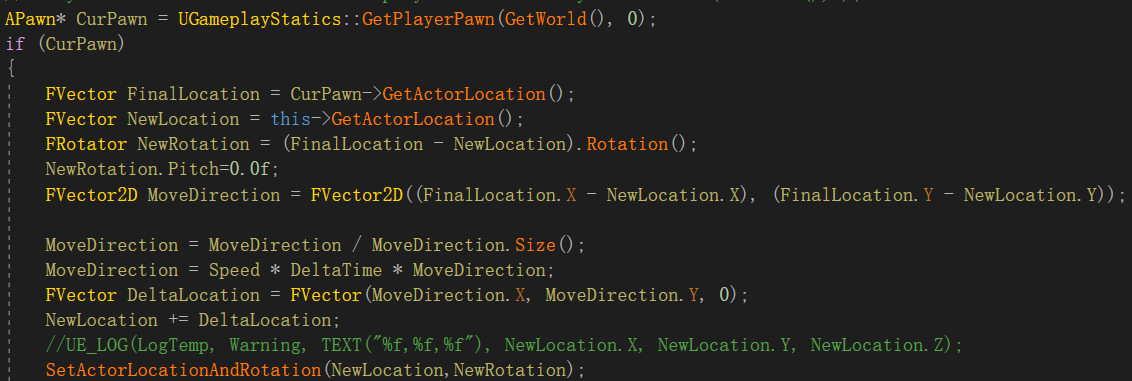
（2）UBoxComponent：检测NPC发生的碰撞；

（3）float *Speed*：NPC移动的速度；

（4）int *NumberOfHit*：NPC被击中几次后销毁（可修改）。

## 追踪玩家

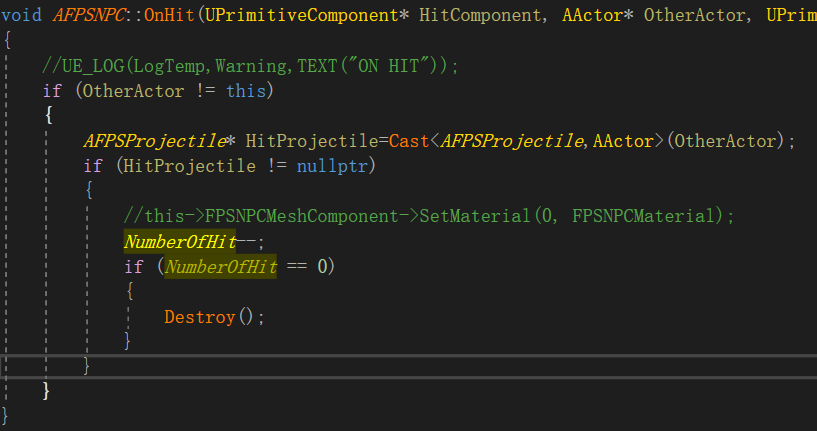
在Tick()函数中添加以下代码，实现NPC向玩家方向的移动。



实现思路为：获取NPC与玩家的位置坐标；由于NPC仅在二维平面移动，故求取玩家在XY平面内相对于NPC的位置（即位置坐标作差），归一化后得到玩家相对NPC的方向向量，与移动速度相乘，得到此帧NPC的位置变化量；以此为基准，设置NPC位置，实现NPC向玩家靠近这一目的。

## NPC销毁

NPC被击中后，检查击中NPC的对象是否是子弹，如果是，将“*NumberOfHit*”减1，当*NumberOfHit*=0时，效果NPC。



# NPC生成器 (SpawnActor)

## 功能

（1）每隔2秒生成一个NPC；

（2）NPC生成的位置具有一定的随机性，即在一定区域内随机生成。

## 基本组件与变量

（1）FTimerHandle：计时器句柄，用于标识该类中的计时器；

（2）float *DeltaTimeOfSpawnActor*：生成NPC的时间间隔，可自由修改；

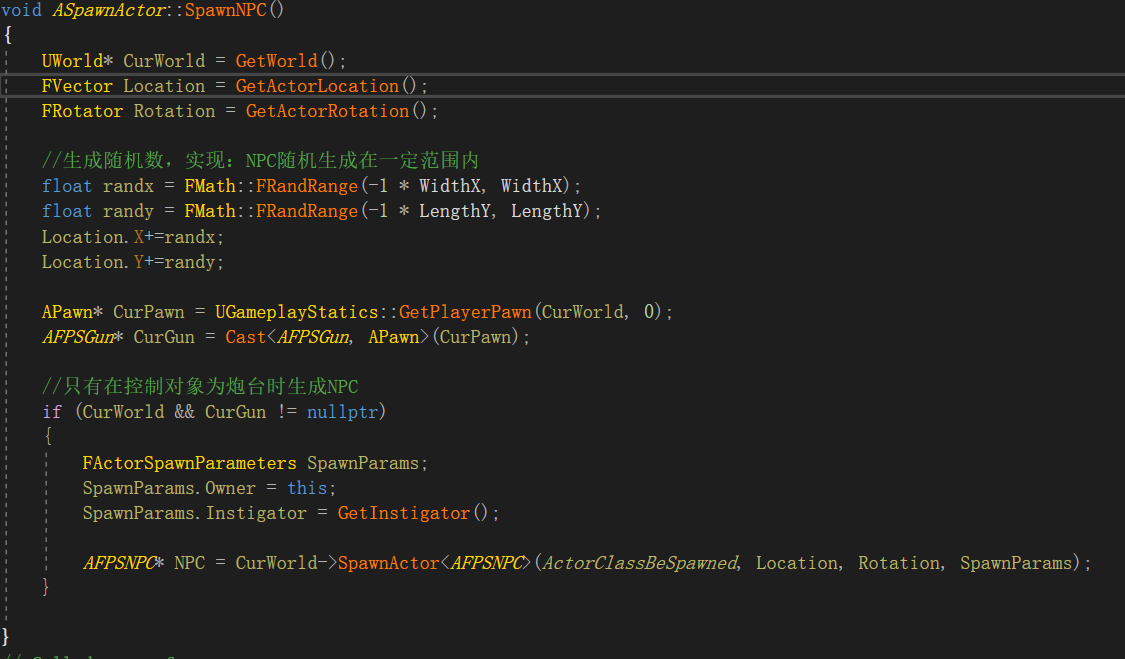
（3）float *WidthX*：设置NPC生成范围的宽度，即X方向的宽度的一半；

（4）float *LengthY*：设置NPC生成范围的长度，即Y方向的长度的一半；

## 实现思路

定时功能由定时器完成，通过GetWorldTimerManager().SetTimer()函数每隔一段时间便调用生成NPC的函数——SpawnNPC()。

SpawnNPC()的实现见下图。通过在(-1 \* *WidthX*, *WidthX*)、(-1 \* *LengthY*, *LengthY*)取随机数实现NPC生成位置的随机化。



# Buff增益（Buff）

由于游戏中可能有多种类型的Buff，；例如加血、回蓝、补充弹药等；

最初的想法是：将Buff类作为基类，具体的增益函数设定为虚函数，将该类设置为其他类型buff的父类，通过重写增益函数实现创建不同的buff类；但引起的问题是，这样的话貌似生成各种buff类时，需分别编写对应的生成函数，因此，后来修改了一下，将增益函数设置为蓝图实现，这样就可以Buff类为基础，衍生出各种buff蓝图类，共用生成器。

## 功能

为玩家提供增益，考虑到游戏中增益可能有很多种，因此，此处的Buff类设置为可在蓝图实现。

## 基本组件与变量

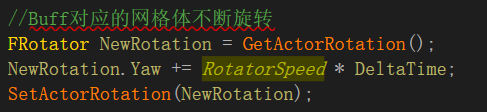
（1）UBoxComponent：检测碰撞；

（2）UStaticMeshComponent：显示Buff外观；

（3）float *RotatorSpeed*：Buff在世界中绕自身旋转的速度。

## 旋转实现

每帧更新角度。



## 增益实现

增益对应的函数*OnBeginOverlap*（）为蓝图可实现的函数，通过不同的蓝图实例可实现不同的增益效果。在发生重叠时调用该函数。