**创建**

class *UParticleSystem*; 向前声明 ~~可用头文件代替 但是会大大加长编译时间~~

*UParticleSystem* 粒子特效 的类型

*USoundBase*\* ObjectvieMissingSound; 声音类型

*UGameplayStatics*::*PlaySound2D*(this, ObjectvieMissingSound);播放声音

UStaticMeshComponent\* MeshComp 静态球体 #include"Components/StaticMeshComponent.h" ！！！**import 让编译器知道我们正在处理那种类型 以及编译器该如何找出这一类型**

*RootComponent* = SphereComp 放到根组件 SphereComp->*SetupAttachment*(MeshComp); 放到MeshComp下

SphereComp ->*SetSphereRadius*(100); 半径设置

OverlapComp->*SetBoxExtent*(*FVector*(20.0f));XYZ均为20.0f

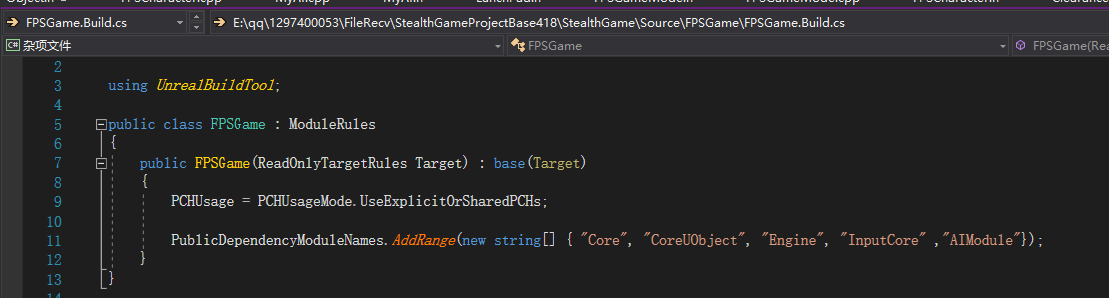
USphereComponent\* SphereComp; 球体碰撞 #include "Components/SphereComponent.h"

SphereComp = *CreateDefaultSubobject*<*USphereComponent*>(*TEXT*("SphereComp")); 实例化

创建默认子对象 将指定类型

AFPSCharacter\* MyPawn = *Cast*<AFPSCharacter>(OtherActor); 类型转换 #include "FPSCharacter.h"

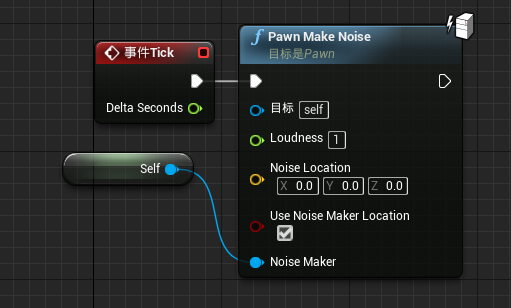
*TArray*<*AActor*\*>ReturnActors;//数组创建方法



**AI**

**需在xxx.build.cs中添加AI模块**

*UPawnSensingComponent*\* PawnSensingComp; #include "DrawDebugHelpers.h" 人形体感应组建 （不是场景组件，也没有层级排序）



*UPawnNoiseEmitterComponent*\* NoiseEmitterComponent; #include "Components/PawnNoiseEmitterComponent.h" 声音发射器组建（目前用于AI侦测）

当视觉机制触发时便不会触发声音事件

**旋转**

**Roll X轴 倾斜**

**Pich y 轴 上下**

**Yaw Z轴 左右**

*FVector* Direction = Location - *GetActorLocation*(); 得到方向向量

Direction.*Normalize*(); 标准化 转为方向矢量

*FRotator* NewLooKat= *FRotationMatrix*::*MakeFromX*(Direction).*Rotator*(); 矩阵旋转

*FRotator* NewLooKat= *FRotationMatrix*::*MakeFromX*(Direction).*Rotator*();

NewLooKat.*Pitch* = 0.0f;

NewLooKat.*Roll* = 0.0f;

*SetActorRotation*(NewLooKat);

**计时器**

*FtimerHandle* TimeHandles;

*GetWorldTimerManager*().*ClearTimer*(TimeHandles);

*GetWorldTimerManager*().*SetTimer*(TimeHandles, this, &AMyAI::ResetOrientation,3.0f,false,-1.f);

循环为假，延迟为-1 表示 3s后开始计时而不是立刻

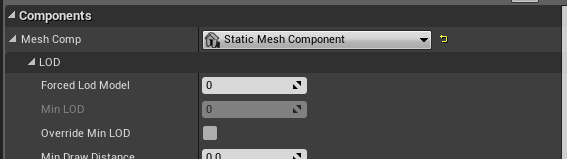
}

void AMyAI::ResetOrientation()

{

*SetActorRotation*(OriginalRotation);

}

**公开**

*UPROPERTY*(EditDefaultsOnly,Category = "FX") 暴露给蓝图**一个**类目FX

*UPROPERTY*(VisibleAnywhere,Category = "Components") 暴漏蓝图**全部**设置选项

*UFUNCTION*(BlueprintImplementableEvent,Category="GameMode") 蓝图可实现事件

MeshComp->*SetCollisionEnabled*(*ECollisionEnabled*::*NoCollision*); 碰撞

OverlapComp->*SetCollisionEnabled*(*ECollisionEnabled*::*QueryOnly*); 仅查询 包括追踪轨迹线或重叠等

OverlapComp->*SetCollisionResponseToAllChannels*(*ECR\_Ignore*); 全为忽略

OverlapComp->*SetCollisionResponseToChannel*(*ECC\_Pawn*, *ECR\_Overlap*); 只对Pawn 设置重叠

*SetCollisionResponseToAllChannels*设置对所有通道的碰撞相应

**向心力 适用于（开物理，开重叠）**

*TArray*<*UPrimitiveComponent*\*> OverlappingComps;

OutSphereCompont->*GetOverlappingComponents*(OverlappingComps); 创建数组将范围内物体存入

for (*int32* i = 0; i < OverlappingComps.*Num*(); i++)

{

*UPrimitiveComponent*\* PrimComp = OverlappingComps[i];

if (PrimComp && PrimComp->*IsSimulatingPhysics*())

{

// the component we are looking for! It needs to be simulating in order to apply forces.

我们正在寻找的组件！为了施加力，需要进行模拟。

const float SphereRadius = OutSphereCompont->*GetScaledSphereRadius*();

const float ForceStrength = -8000; // Negative value to make it pull towards the origin instead of pushing away

负值使它向原点拉而不是推开

PrimComp->*AddRadialForce*(*GetActorLocation*(), SphereRadius, ForceStrength, *ERadialImpulseFalloff*::*RIF\_Constant*, true);

}

}

**其它**

InstigatorPawn->*DisableInput*(nullptr); 禁止玩家进行控制

*UGameplayStatics*::*GetAllActorsOfClass*(this, SpectatingViewpointClass, ReturnActors); 获取SpectatingViewpointClass类型的存放在数组ReturnActors中

*TSubclassOf*<*AActor*>SpectatingViewpointClass; 调用一个Actor

*DrawDebugSphere*(*GetWorld*(), SeenPawn->*GetActorLocation*(), 32.0f, 12, *FColor*::*Yellow*, false, 10.0f); #include "DrawDebugHelpers.h"

半径 分段数 持久谱线 持续时长

**弹射（lunchpad）**

void AFPSLaunchPad::OverlapLaunchPad(*UPrimitiveComponent*\* OverlappedComponent, *AActor*\* OtherActor,

*UPrimitiveComponent*\* OtherComp, *int32* OtherBodyIndex, bool bFromSweep, const *FHitResult*& SweepResult)

{

// Make rotator with our specified 'pitch' and convert to a direction vector \* intensity

用指定的“螺距”旋转转子，并转换成方向矢量\*强度。

*FRotator* LaunchDirection = *GetActorRotation*();

LaunchDirection.*Pitch* += LaunchPitchAngle; 角度

*FVector* LaunchVelocity = LaunchDirection.*Vector*() \* LaunchStrength; 向量

*ACharacter*\* OtherCharacter = *Cast*<*ACharacter*>(OtherActor);

if (OtherCharacter)

{

// Launch Player! Both booleans give consistent launch velocity by ignoring the current player velocity

OtherCharacter->*LaunchCharacter*(LaunchVelocity, true, true);

// Spawn FX

*UGameplayStatics*::*SpawnEmitterAtLocation*(*GetWorld*(), ActivateLaunchPadEffect, *GetActorLocation*());

}

// Did not overlap a player, so check if it's a physics simulating actor we can launch

else if (OtherComp && OtherComp->*IsSimulatingPhysics*())

{

OtherComp->*AddImpulse*(LaunchVelocity, *NAME\_None*, true);

// Spawn FX

*UGameplayStatics*::*SpawnEmitterAtLocation*(*GetWorld*(), ActivateLaunchPadEffect, *GetActorLocation*());

}

}

**碰撞方法**

一（ClearanceActor）

.h

*UFUNCTION*() 绑定函数时，需将其标记，以便让虚幻后端明白该函函数的含义及如何将其与事件绑定

void HandLeOverlap( *UPrimitiveComponent*\* OverlappedComponent, *AActor*\* OtherActor, *UPrimitiveComponent*\* OtherComp, *int32* OtherBodyIndex,

bool bFromSweep, const *FHitResult* & SweepResult);

cpp

….

OverlapComp->*OnComponentBeginOverlap*.*AddDynamic*(this, &AClearanceActor::HandLeOverlap); 绑定

….

void AClearanceActor::HandLeOverlap(*UPrimitiveComponent* \* OverlappedComponent, *AActor* \* OtherActor, *UPrimitiveComponent* \* OtherComp,

*int32* OtherBodyIndex, bool bFromSweep, const *FHitResult* & SweepResult)

{

*UE\_LOG*(LogTemp, *Log*, *TEXT*("overloap")); 输出到日志

}

二（FPSObjectiveActor）

.h

virtual void NotifyActorBeginOverlap(*AActor*\* OtherActor)override; 重载

cpp

void AMyActor::NotifyActorBeginOverlap(*AActor*\* OtherActor)

{

Super::NotifyActorBeginOverlap(OtherActor);

PlayEffects();

Abiu5Character\* character = *Cast*<Abiu5Character>(OtherActor); 类型转换

if (character)

{

character->bIsCarryingObjective = true;

*Destroy*();

}

}