**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

MeshComp = *CreateDefaultSubobject*<*USkeletalMeshComponent*>(*TEXT*("MeshComp")); 骨骼网格体

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

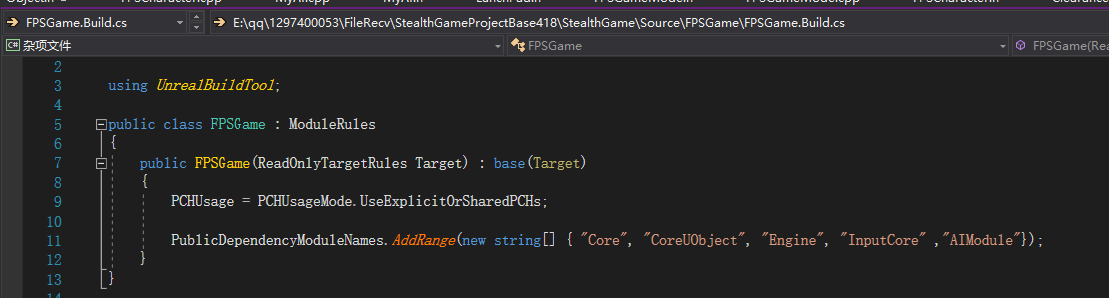
USphereComponent\* SphereComp; 球体碰撞 #include "Components/SphereComponent.h"

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"

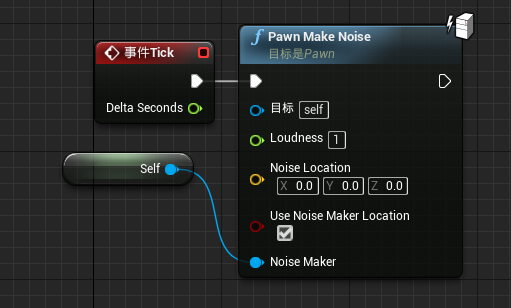
*TArray*<*AActor*\*>ReturnActors;//数组创建方法



**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

*MakeNoise*(1.0f, *Instigator*); *Instigator 为变量存在于每一个Actor中，通常用于伤害。*

*Instigator十分关键，因为MakeNoise函数会从内部使用Instigator，确认其是否具有PawnNoiseEmitterComponent从而发出声音*

**旋转**

**Roll X轴 倾斜**

**Pich y 轴 上下**

**Yaw Z轴 左右**

*FVector* Direction = Location - *GetActorLocation*(); 例： 得到方向向量 OA = a, OB = b, BA = a – b; 即 a-b表示为 从向量b的终点指向向量a的终点

声音位置

Direction.*Normalize*(); 标准化 转为方向矢量

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

基于X的旋转

NewLooKat.*Pitch* = 0.0f;

NewLooKat.*Roll* = 0.0f;

*SetActorRotation*(NewLooKat);

*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);

}

**移动**

void AMyAI::Tick(float DeltaTime)

{

Super::Tick(DeltaTime);

if (CurrentPatrolPoint)

{

*FVector* Delta = *GetActorLocation*() - CurrentPatrolPoint->*GetActorLocation*();

float DistanceToGoal = Delta.*Size*();

if (DistanceToGoal < 100) 与胶囊体大小有关

{

MoveToNextPatrolPoint();

}

}

}

void AMyAI::MoveToNextPatrolPoint()

{

if (CurrentPatrolPoint == nullptr || CurrentPatrolPoint == SecondPatrolPoint)

{

CurrentPatrolPoint = FirstPatrolPoint;

}

else

{

CurrentPatrolPoint = SecondPatrolPoint;

}

*UNavigationSystem*::*SimpleMoveToActor*(*GetController*(), CurrentPatrolPoint);

目标

}

**。。。**

*AController*\* AIController = *GetController*();

if (AIController)

{

AIController->*StopMovement*(); 停止移动

}

**。。。**

**联机**

**Tick函数既在服务器运行又在客户端运行**

*SetReplicates*(true); 能在生成发射物时为客户端复制同样效果

*SetReplicateMovement*(true); 移动，位置，转动等等只要服务器做出改变，都会实时更新至客户端

*UFUNCTION*(Server,Reliable,WithValidation) 每当我们创建带有关键词Server的*UFUNCTION* 时后台就会自动为 *\_Implementation \_Validate* 创建头文件

void SeverFire(); Reliable指能确保连接至服务器或许不会立即连接，比如遇到数据丢失等故障情形时但最终能确保连接 WithValidation 有效性验证

**当我们在客户端调用这一函数，他会运行但不会在客户端执行，而是会发送请求至服务器**

服务器端运行的话，Actor的Role为Role\_Authority

在客户端Actor的Role为Role\_SimulatedProxy

* **ROLE\_None**：
* **ROLE\_SimulatedProxy**：这个actor是其他客户端在本机客户端的一个模拟代理
* **ROLE\_AutonomousProxy**：这个actor是本机客户端的自己控制的角色
* **ROLE\_Authority**：这个actor是服务器上的acto

**数据同步**

**{***UPROPERTY*(ReplicatedUsing = Rep\_OnAIState) **当状态改变时便会调用这一函数 但只在客户端运行** *UPROPERTY*(Replicated,BlueprintReadOnly,Category = "Gameplay")

EAIState AIState; bool bIsCarryingObjective;

*UFUNCTION*()

void Rep\_OnAIState();

#include "Net/UnrealNetwork.h"

void AMyAI::GetLifetimeReplicatedProps(*TArray*< *FLifetimeProperty* > & OutLifetimeProps) const **应用默认规则复制变量，这意味着该变量将复制到所有与我们相连的机器，即所有客户端**

{

Super::GetLifetimeReplicatedProps(OutLifetimeProps);

*DOREPLIFETIME*(AMyAI, AIState);

//DOREPLIFETIME\_CONDITION(AFPSCharacter, bIsCarryingObjective, COND\_OwnerOnly); 优化

}

**}**

**玩家控制器与pawn的不同之处在于玩家控制器在服务器和拥有该玩家控制器的客户端上均存在**

*NetUpdateFrequency* = 66.0f; 网络更新频率

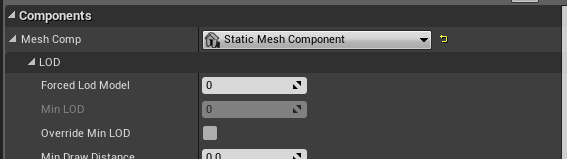
*MinNetUpdateFrequency* = 33.0f; 最小网络更新频率

**Tilde key to open console**

**Open firstPersonExampleMap?listen 主机**

**客户端 open 10.2.4.4：7777**

**主机ip 端口号**



**公开**

*UPROPERTY*(EditDefaultsOnly,Category = "FX") 暴露给蓝图**一个**类目FX，且只能在蓝图中编辑

*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

*UFUNCTION*(BlueprintCallable,Category="GameMode")



***{*** *UPROPERTY*(EditInstanceOnly,Category ="AI") 仅实例化时编辑

bool bPatrol;

*UPROPERTY*(EditInstanceOnly, Category = "AI" , meta = (EditCondition = "bPatrol")) 当bPatrol为真时可编辑

*AActor*\* FirstPatrolPoint;

**}**

*UPROPERTY*(VisibleAnywhere, BlueprintReadOnly, Category = "Components") //TEXT只能看见不能更改AND蓝图中可以调用但不可更改

bool vale;

*UPROPERTY*(VisibleAnywhere, Category = "Components") //TEXT只能看见不能更改不能调用

bool vale;

*UPROPERTY*(*EditDefaultsOnly*,*Category* = "Player",*meta*=(*ClampMin* = 0.0,*ClampMax* = 100)) 限制最大值最小值

float ZoomInterpSpeed;

**自定义事件创建**

*CLARE\_DYNAMIC\_MULTICAST\_DELEGATE\_SixParams*(FOnHealthChangedSignature, USHelthComponent\*,HealthComp,float,*Health*,float,HealthDelta,const class *UDamageType*\*, DamageType, class *AController*\*, InstigatedBy, *AActor*\*, DamageCauser);

名称 触发该事件的组件

OnHealthChanged.Broadcast(this, Healtth, Damage, DamageType, InstigatedBy, DamageCauser);

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[[1]](#footnote-1)

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

**输出日志**

*UE\_LOG*(LogTemp, *Log*, *TEXT*("%f"), DistanceToGoal);

*UE\_LOG*(LogTemp, *Log*, *TEXT*("overloap"));

**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*();

}

}

**角色移动**

PlayerInputComponent->*BindAxis*("Turn", this, &ASCharacter::*AddControllerYawInput*);

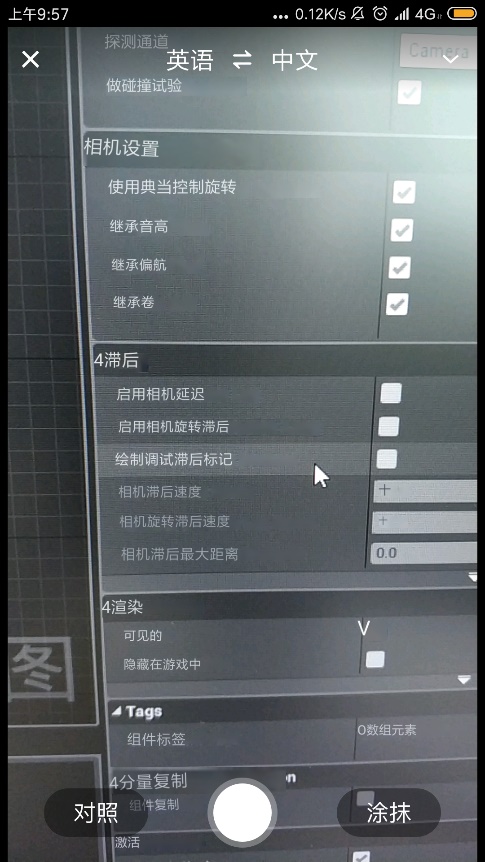
**与项目设置中相同 绑定对应函数**

*GetMovementComponent*()->*GetNavAgentPropertiesRef*().*bCanCrouch* = true;

**蓝图也可设置（如果无效在蓝图中对应查找）**

**相机**

**弹簧臂 相机延迟**



相机前移（开镜）

float TargetFOV = bWantsToZoom ? ZoomedFOV : DefaultFOV;

得到默认焦距 增长范围

float NewFov = *FMath*::*FInterpTo*(CameraComp->*FieldOfView*, TargetFOV, DeltaTime, ZoomInterpSpeed); （焦距值） 使相机缓慢拉近，否则将出现卡顿

CameraComp->*SetFieldOfView*(NewFov); （\*）

相机抖动效果

*TSubclassOf*<*UCameraShake*> FireCamShake;

*APawn*\* MyOwner = *Cast*<*APawn*>(*GetOwner*());

if(MyOwner)

{

*APlayerController*\* PC = *Cast*<*APlayerController*>(MyOwner->*GetController*());

if (PC)

{

PC->*ClientPlayCameraShake*(FireCamShake);

}

}

**创建控制台变量**

static *int32* DebugweapondDrawing = 0;

*FAutoConsoleVariableRef* CVARDebugWeaponDrawing(*TEXT*("COOP.DebugWeapons"), DebugweapondDrawing,*TEXT*("Draw Debug Lines for Weapons"),*ECVF\_Cheat*);

**命名 此变量可在游戏运行时动态变更 方便debug**

**场景组建能提供“转换”，表示着游戏中呈现的特定视觉效果**

1. [↑](#footnote-ref-1)