ChapterOne

Alt+G 跳转

Alt+O 跳转头文件

Destroy 毁坏

.h

class UparticleSystem; //向前声明

UPROPERTY(EditDefaultsOnly, Category = "FX")

（公开） （仅编辑默认） （类目）

UParticleSystem\* ExplosionEffect;

（粒子系统）

.cpp

//生成粒子系统

UGameplayStatics::SpawnEmitterAtLocation(GetWorld(),ExplosionEffect,GetActorLocation());

（在原处生成发射器） （爆炸效应）

ChapterTwo

Shift+Alt+O查找

QueryOnly 只发出询问

BlueprintReadOnly 仅阅读蓝图

EditAnyWhere: 此成员变量在蓝图编辑器中和关卡细节面板中都会被暴露出来

EditDefaultsOnly: 此成员变量只会在蓝图编辑器中被暴露出来

EditInstanceOnly: 此成员变量只会在关卡细节面板中被暴露出来

Virtual: 在某基类中声明为 virtual 并在一个或多个派生类中被重新定义的成员函数，用法格式为：virtual 函数返回类型 函数名（参数表） {函数体}；实现多态性，通过指向派生类的基类指针或引用，访问派生类中同名覆盖成员函数。

.h

公开碰撞组建在蓝图中显示,定义网格体

eg:

UPROPERTY(VisibleAnywhere, Category = "Components")

（随处可见）

UStaticMeshComponent\* MeshComp;

UPROPERTY(VisibleAnywhere, Category = "Components")

USphereComponent\* SphereComp;

UPROPERTY(EditDefaultsOnly,Category="Effects")

（仅在蓝图编辑器中显示） （特效）

UParticleSystem\* PickupFX;

函数重载重写

eg:

virtual void NotifyActorBeginOverlap(AActor\* OtherActor) override;

（虚函数）重写 （在发生重叠时通知Actor） （覆盖）

.cpp

MeshComp = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("MeshComp"));

(创建默认子对象) （实例化类型） （名称）

MeshComp->SetCollisionEnabled(ECollisionEnabled::NoCollision);

（设置碰撞启用）

RootComponent = MeshComp;

（设置根组件）

SphereComp->SetCollisionResponseToAllChannels(ECR\_Ignore);

（设置碰撞为响应所有通道） (忽略)

SphereComp->SetCollisionResponseToChannel(ECC\_Pawn, ECR\_Overlap);

(设置碰撞为响应指定通道) (人形体) (忽略)

SphereComp->SetupAttachment(MeshComp);

（设置附加）

Void （文件名）::自定义事件名（）

{

功能

}

eg：

void AFBSObjectiveActor::PlayEffects()

{

UGameplayStatics::SpawnEmitterAtLocation(this, PickupFX, GetActorLocation());

创建插槽(名字)

}

void AFBSObjectiveActor::NotifyActorBeginOverlap(AActor\* OtherActor)

{

Super::NotifyActorBeginOverlap(OtherActor);

(特级的)

PlayEffects();

类型转换

AFPSCharacter\* MyCharacter = Cast<AFPSCharacter>(OtherActor);

if (MyCharacter)

{

MyCharacter->bIsCarryingObjective = true;

Destroy();

}

}

BlackHole

AddDynamic 添加动态

Generate Overlap(生成重叠事件)

//称每一帧

void AFPSBlackHold::Tick(float DeltaTime)

{

Super::Tick(DeltaTime);

//找到所有可能发生碰撞的重叠组件，并可能进行物理模拟

TArray<UPrimitiveComponent\*>OverlappingComps;//ZXprimitive元组件

//物理作用力的对象

OuterSphereComponent->GetOverlappingComponents(OverlappingComps);

//循环

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

{

UPrimitiveComponent\* PrimComp = OverlappingComps[i];

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

{

//我们正在寻找的组件！它需要模拟才能施加外力。

const float SphereRadius = OuterSphereComponent->GetScaledSphereRadius();

//球体半径

const float ForceStrength = -2000;//负的价值使它向原点而不是推开

//作用力强度

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

//施加作用力（径向力），返回元组件

}

}

}

ChapterThree

EditAnyWhere: 此成员变量在蓝图编辑器中和关卡细节面板中都会被暴露出来

EditDefaultsOnly: 此成员变量只会在蓝图编辑器中被暴露出来

EditInstanceOnly: 此成员变量只会在关卡细节面板中被暴露出来

Intellisense智能编码

Close All But This 关闭除此之外的所用文件

DeveLoper Tools开发者工具

Output Log输出日志

.cpp

//实例化 Ctrl+F 搜索“Size”尺寸

DecalComp = CreateDefaultSubobject<UDecalComponent>(TEXT("DecalComp"));

DecalComp->DecalSize = FVector(200.0f,200.0f,200.0f);

DecalComp->SetupAttachment(RootComponent);

.h

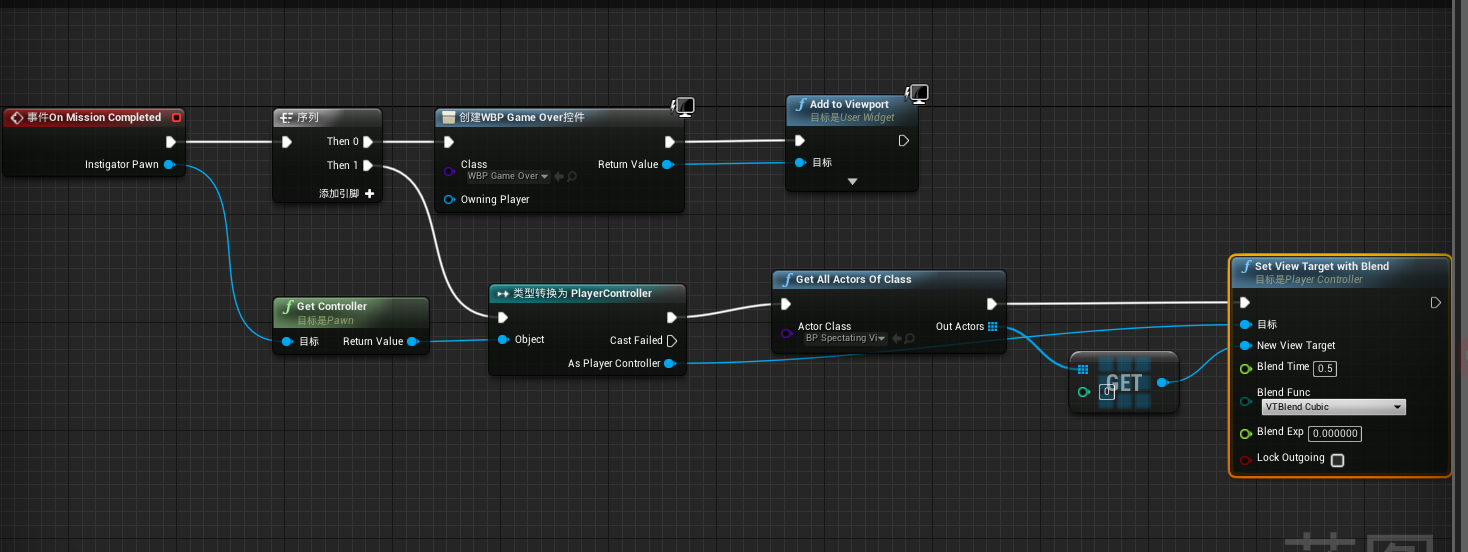
UPROPERTY(VisibleAnywhere, Category = "Components")

UDecalComponent\* DecalComp;

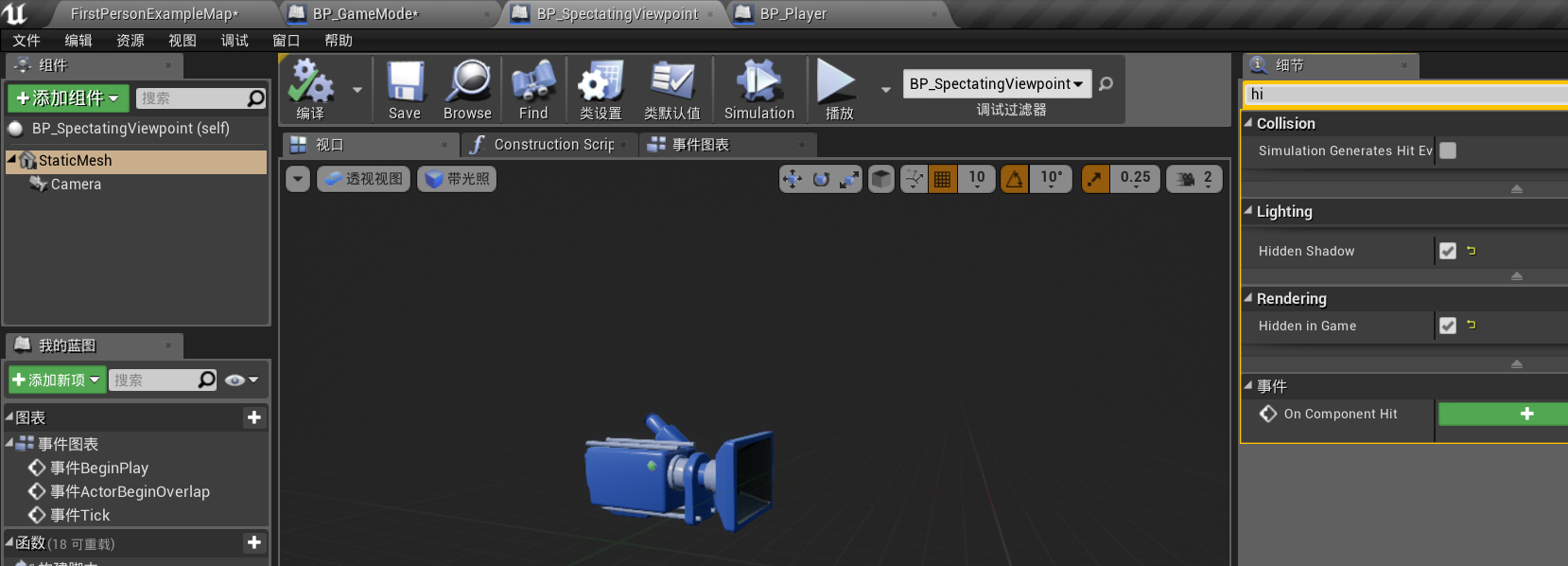
//贴花组件

蓝图第一人称转第三人称视角编辑：

打开BP\_GameMode



新建蓝图添加StaticMesh(静态网格体) 固定在根目录下, 添加Camera（相机）,设置Hidden in Game保证它再游戏中隐藏显示。



.h

UPROPERTY(VisibleAnywhere,Category="Components")

ChapterFour

Use Animation Asset(使用动画资源)

hearing Threshold 听阈；听阀；听力障碍；听觉阈限

LOSHearing Threshold 视距 （指朝发生源望去，其间无遮挡的直线距离）

Peripheral Vision Angle 视野范围

Sight Radius 视野半径

Delegate代理

nullptr 空指针;指针

Create Implementation(建立执行)

Current Project(当前项目)

Expose On Spawn(在生成时显示)

Instance Editable(实力可编辑)

Refresh Node(更新节点）

Guard State(守卫状态)

Draw at Desired Size(自定义尺寸)

OnStateChanged(状态更新时)

CHALLENGE CODE 挑战模式

Nav Mesh Bound Volume(寻路网格体包围体)（p显示）

Select AllMatching Class(选中所有匹配类)

UpawnSensingComponent\* PawnSensingComp;

(人形体感应组建)

PawnSensingComponent是涉及AI编程时候的常用类，实现了很多强大的功能，对于AI感知帮助很大。

1.AI的视觉，是一个锥形感知体。

2.AI的听觉，主要与距离和声音大小有关，与遮挡物也有关。

1.CouldSeePawn仅仅简单地判断你给出的Pawn是否在锥形内。（这个考虑了锥形的覆盖，但是没有考虑遮蔽）

2.HasLineOfSightTo仅仅简单地判断你给出的Pawn是否内被这个PawnSensingComponent看见（这个又没有考虑锥形的覆盖，只考虑了遮挡)

.cpp

视觉绑定，添加动态函数

PawnSensingComp->OnSeePawn(Alt+G).AddDynamic(this, &AEPSAIGuard::OnPawnSeen);

{

if(SeenPawn==nullptr)

{

return;

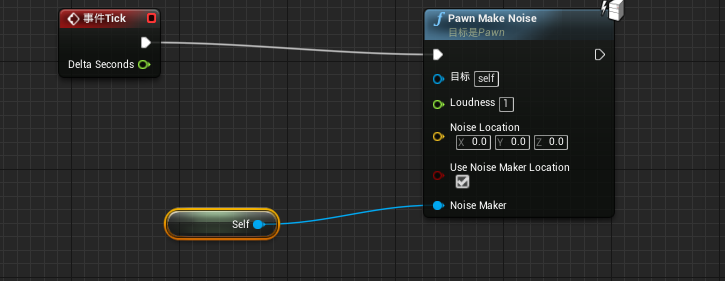
}

DrawDebugSphere(GetWorld(),SeenPawn->GetActorLocation(),32.0f,12,FColor::Red,false, 10.0f);

//设置半径，分段数 ，颜色 , PersistentLine(持久谱线)，持续时长

}

制造声音



LookAtRotation(朝向旋转)

FRotationMatrix(旋转矩阵)

Direction(方向)

GetWorldTimerManager(获取世界场景时钟管理器)

TimeHandle(定时器句柄)

ResetOrientation(重置方向)

ClearTimer(清除定时器)

Pitch上下转动

Yaw左右转动

Roll倾斜转动

OriginalRotation初始方位

.cpp

//在巡逻时停止移动

AController\* Controller = GetController();

if (Controller)

{

Controller->StopMovement();

}

}

//转动调整

NewLookAt.Pitch = 0.0f;

NewLookAt.Roll = 0.0f;

SetActorRotation(NewLookAt);

//用该句柄停止原先的定时器，并重设一个新的定时器，这样就不会同时运行多个定时器了

GetWorldTimerManager().ClearTimer(TimerHandle\_ResetOrientation);

//设置定时器

GetWorldTimerManager().SetTimer(TimerHandle\_ResetOrientation, this, &AFPSAIGuard::ResetOrientation, 3.0f);

// 称每一帧

void AFPSAIGuard::Tick(float DeltaTime)

{

Super::Tick(DeltaTime);

// 巡逻检查目标

if (CurrentPatrolPoint)

{

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

float DistanceToGoal = Delta.Size();

// 距离小于100个单位长度，则寻找下一个巡逻点,ZX实现守卫往返两点之间

if (DistanceToGoal < 100)

{

MoveToNextPatrolPoint();

}

}

}

void AFPSAIGuard::MoveToNextPatrolPoint()

{

//分配下一个巡逻点

// 检查守卫是否不在巡逻点上，当游戏开始时，条件肯定为真，或用于检查当前的巡逻点是否为第二个巡逻点

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

{

//为真靠近第一个巡逻点

CurrentPatrolPoint = FirstPatrolPoint;

}

else

{

//为假靠近第二个巡逻点

CurrentPatrolPoint = SecondPatrolPoint;

}

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

//当前巡逻点

}

UENUM(BlueprintType)

enum class EAIState : uint8

{

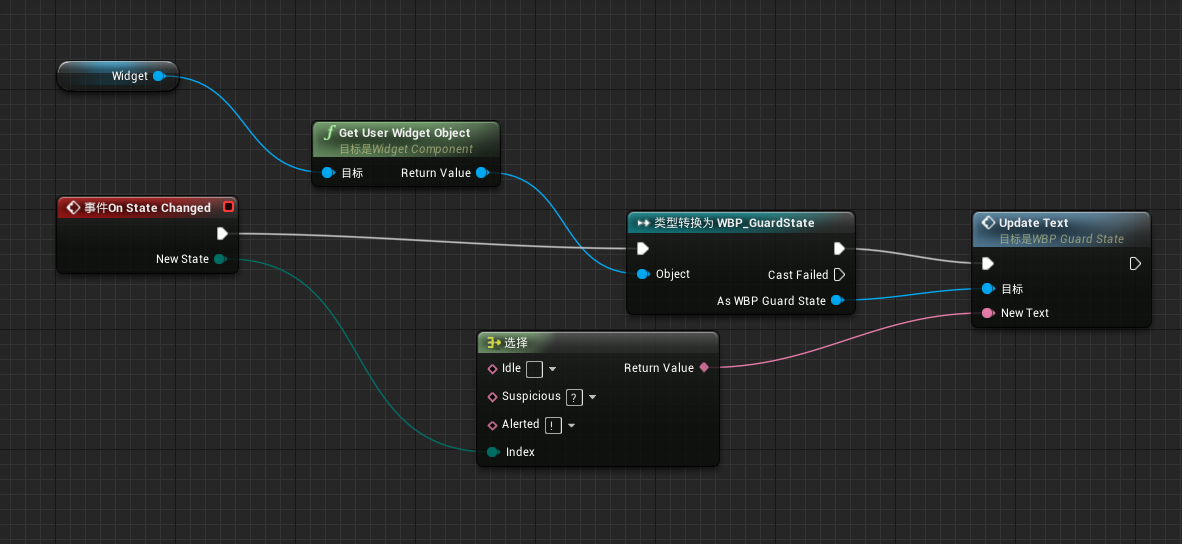
Idle,//闲散

Suspicious,//怀疑

Alerted//警觉

};

状态设置



protected:

//挑战模式

UPROPERTY(EditInstanceOnly, Category = "AI")

bool bPatrol;

//两个巡逻点的第一个巡逻点

UPROPERTY(EditInstanceOnly, Category = "AI", meta = (EditCondition = "bPatrol"))

AActor\* FirstPatrolPoint;

//两个巡逻点的第二个巡逻点

UPROPERTY(EditInstanceOnly, Category = "AI", meta = (EditCondition = "bPatrol"))

AActor\* SecondPatrolPoint;

//目前的观点是，演员要么移动要么站在

AActor\* CurrentPatrolPoint;

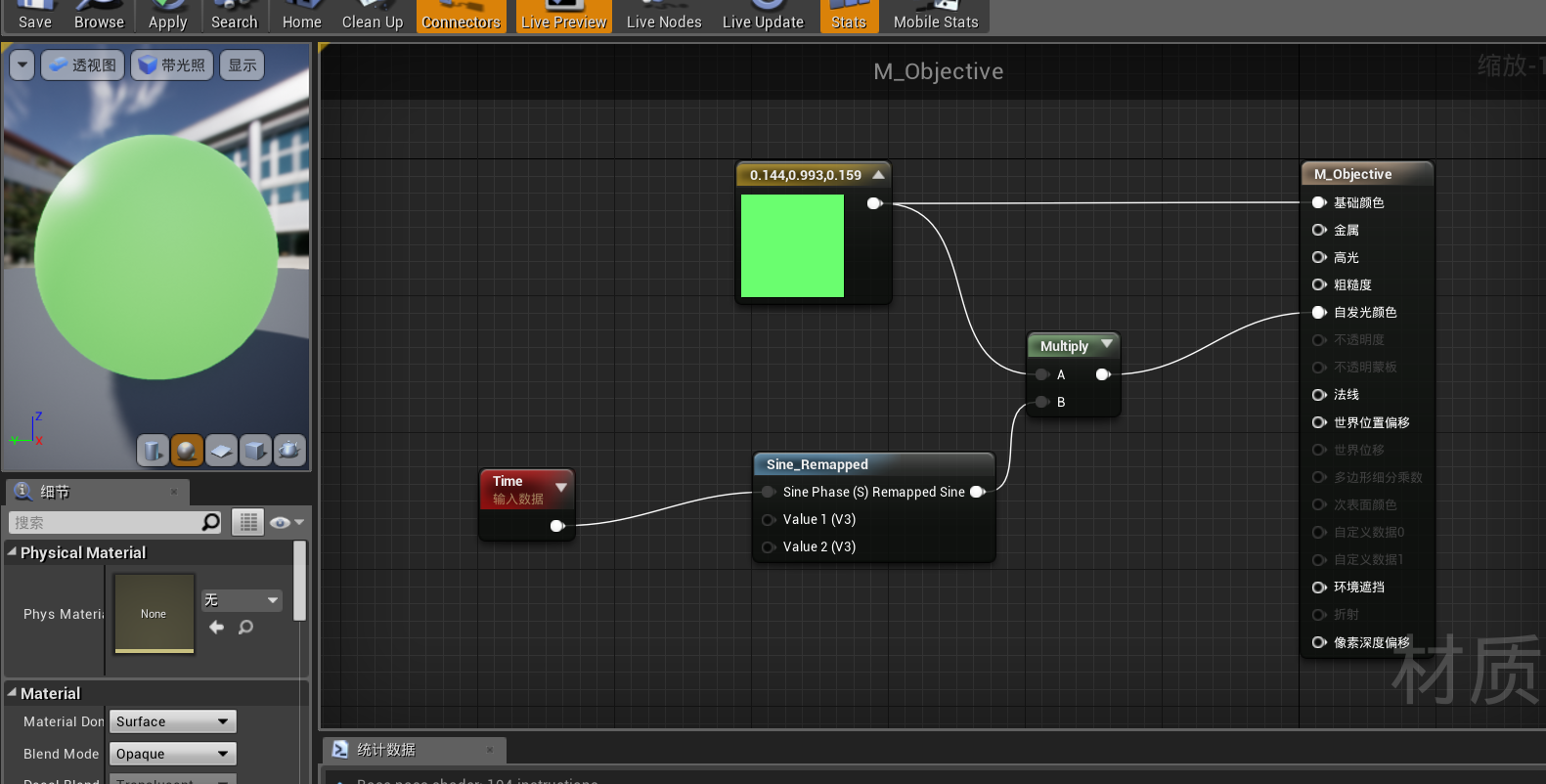
//移动至下一个巡逻点

void MoveToNextPatrolPoint();

ChapterFive

材质：闪烁发光

Sine\_Remapped(正弦\_重新映射)



Reliable 这一关键词指的是能确保链接到服务器或许不会立刻连接，比如遇到数据包丢失等故障情形时但最终能确保链接

\_Implementation（\_实现）

Replication(复制)

Client（客户端）

Server（服务器）

OnStateChanged(状态更改时)

RelativeRotation(相对旋转度)

ISLocallyControlled(玩家是否受本地控制)

RemoteViewPitch(远程视图俯视角)

GetLifetimeRepliicatdProps(在其生命周期内复制)

Game State Base（游戏状态机）

MulticastOnMissionComplete(任务完成时组播)

Shift+F1显示鼠标

发射显示

.cpp

//将SetReplicates(设置复制品/复制品移动(希望发射物能够移动))设置为真 同步

SetReplicates(true);

SetReplicateMovement(true);

.h

void Fire();

//虚拟函数 服务器 可靠链接 进行验证

UFUNCTION(Server,Reliable,WithValidation)

//新建服务器开火函数

void ServerFire();

.cpp

//发送请求

ServerFire();

//执行

void AFPSCharacter::ServerFire\_Implementation()

{

// try and fire a projectile

if (ProjectileClass)

{

FVector MuzzleLocation = GunMeshComponent->GetSocketLocation("Muzzle");

//ZX枪口位置 ZX枪口旋转角度

FRotator MuzzleRotation = GunMeshComponent->GetSocketRotation("Muzzle");

//Set Spawn Collision Handling Override

FActorSpawnParameters ActorSpawnParams;

ActorSpawnParams.SpawnCollisionHandlingOverride = ESpawnActorCollisionHandlingMethod::AdjustIfPossibleButDontSpawnIfColliding;

//Actor人形体参数

ActorSpawnParams.Instigator = this;

// spawn the projectile at the muzzle

GetWorld()->SpawnActor<AFPSProjectile>(ProjectileClass, MuzzleLocation, MuzzleRotation, ActorSpawnParams);

}

}

//它会在服务器端进行完整型检查时用到

bool AFPSCharacter::ServerFire\_Validate()

{

return true;

}

// 角色 角色授权

if (Role==ROLE\_Authority)

{

//检查是否会物理模拟，当击中任意Actor时如if为真，则会对Actor施以推力，且该发射物会自行销毁

MakeNoise(1.0f, Instigator);

Destroy();

}

.cpp

角色视角

void AFPSCharacter::Tick(float DeltaTime)

{

Super::Tick(DeltaTime);

//ZX玩家不受本地控制时也能正常更新网格体

if (!IsLocallyControlled())

{

FRotator NewRot = CameraComponent->RelativeRotation;

//ZX使用RemoteViewPitch来覆盖Pitch,ZX创建新的旋转度

NewRot.Pitch = RemoteViewPitch \* 360.0f / 255.0f;

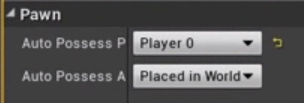
//ZX插入

CameraComponent->SetRelativeRotation(NewRot);

}

Possess(支配) auto（自动）

第一个进入游戏关卡的玩家控制器，会分配至一Pawn上，对于多数玩家而言，并不希望这样的模式



销毁设置

.cpp

//设置复制品为真，也能将其同步到客户端

SetReplicates(true);

//ZX碰撞事件在客户端和服务器上均会发生

void AFBSObjectiveActor::NotifyActorBeginOverlap(AActor\* OtherActor)

{

Super::NotifyActorBeginOverlap(OtherActor);

PlayEffects();

//ZX仅在服务器上运行

if (Role==ROLE\_Authority)

{

AFPSCharacter\* MyCharacter = Cast<AFPSCharacter>(OtherActor);

if (MyCharacter)

{

MyCharacter->bIsCarryingObjective = true;

Destroy();

}

}

}

.cpp

//ZX它能让数据传输至任一控制角色的机器上这样设置也能有效地节省带宽

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

//ZX只用于游戏信息平视显示

}

.h

public:

//ZX网路组播 ZX可靠链接

UFUNCTION(NetMulticast, Reliable)

void MulticastOnMissionComplete(APawn\* InstigatorPawn, bool bMissionSuccess);

.cpp

void AFPSGameState::MulticastOnMissionComplete\_Implementation(APawn\* InstigatorPawn, bool bMissionSuccess)

{

for (FConstPlayerControllerIterator It=GetWorld()->GetPlayerControllerIterator();It;It++)

{

AFPSPlayerController\* PC = Cast<AFPSPlayerController>(It->Get());

//这一过程指，本地机器调用该函数后能找出玩家实际操作的pawn是哪个

if (PC && PC->IsLocalController())

{

PC->OnMissionCompleted(InstigatorPawn, bMissionSuccess);

//ZX禁用输入

APawn\* MyPawn = PC->GetPawn();

if (MyPawn)

{

MyPawn->DisableInput(PC);

}

}

}

}

.cpp

//获取类

GameStateClass = AFPSGameState::StaticClass();

//当玩家完成任务时，我们会获取游戏状态，将其转换为正确类型，然后调用对应的组播函数

AFPSGameState\* GS = GetGameState<AFPSGameState>();

if (GS)

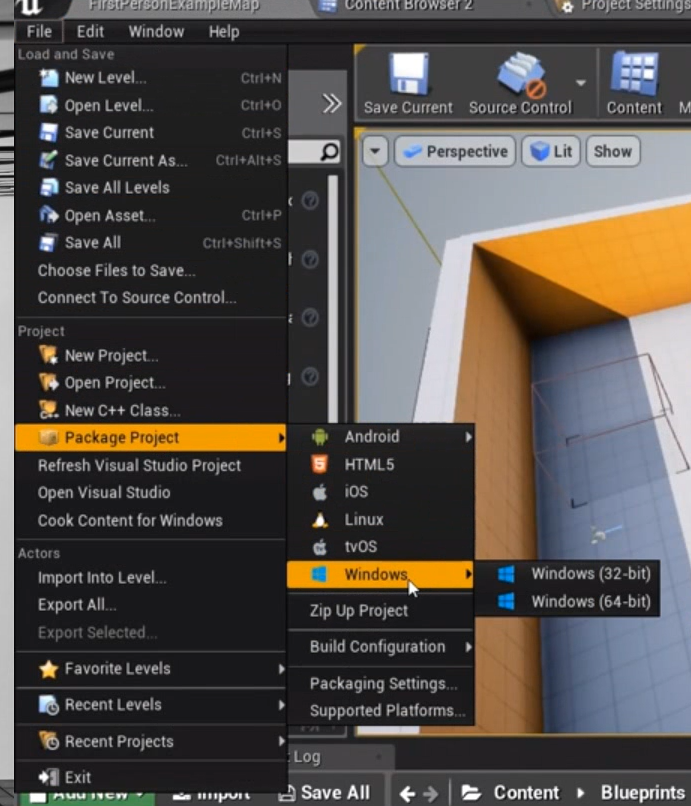
{

GS->MulticastOnMissionComplete(InstigatorPawn, bMissionSuccess);

}

OnMissionCompleted(InstigatorPawn, bMissionSuccess);

打包



Tilde key to open console  
Open firstPersonExampleMap?listen 主机  
客户端  open C:\Users\25476\AppData\Local\Temp\%W@GJ$ACOF(TYDYECOKVDYB.png10.2.4.4：7777  
主机ip   端口号