打印到屏幕

FString str;

str = FString::SanitizeFloat(VectorAngleLevel);

GEngine->AddOnScreenDebugMessage(-1, 5.0f, FColor::Yellow, str );

GEngine->AddOnScreenDebugMessage(-1, 10000.f, FColor::Yellow, TEXT("Begin\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"));

GEngine->AddOnScreenDebugMessage(-1, 10000.f, FColor::Yellow, TEXT("End\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"));

射线距离

LineTrace

获得某个类在场景中的所有实例

GetAllActorsOfClass

类型转换

FString str;

str = FString::SanitizeFloat(TankBarrel->GetComponentLocation().X);

三点求夹角

void ATank::Angle(AActor\* OtherActor)

{

//Self location 3D

FVector SelfLocation3D;

SelfLocation3D = GetActorLocation();

//TankBarrel location 3D

FVector BarrelLocation3D;

BarrelLocation3D = TankBarrel->GetComponentLocation();

//OtherAActor

FVector OtherActorLocation3D;

OtherActorLocation3D = OtherActor->GetActorLocation();

//BarrelVector

FVector BarrelVector;

BarrelVector = BarrelLocation3D - SelfLocation3D;

//ActorVector

FVector ActorVector;

ActorVector = OtherActorLocation3D - SelfLocation3D;

//判断向量夹角,大于零则大于180

if ((BarrelVector.X \* ActorVector.Y - ActorVector.X \* BarrelVector.Y)>0)

{

\_Greater180 = true;

}

else

{

\_Greater180 = false;

}

//VectorAngle

double VectorAngleLevel, VectorAngleVertical;

VectorAngleLevel = acos((BarrelVector.X\*ActorVector.X + BarrelVector.Y\*ActorVector.Y) / ((sqrt(BarrelVector.X\*BarrelVector.X + BarrelVector.Y\*BarrelVector.Y)\*sqrt(ActorVector.X\*ActorVector.X + ActorVector.Y\*ActorVector.Y))))\*(180.0 / 3.141592654);

VectorAngleVertical = acos((BarrelVector.X\*ActorVector.X + BarrelVector.Z\*ActorVector.Z) / ((sqrt(BarrelVector.X\*BarrelVector.X + BarrelVector.Z\*BarrelVector.Z)\*sqrt(ActorVector.X\*ActorVector.X + ActorVector.Z\*ActorVector.Z))))\*(180.0 / 3.141592654);

}

强制类型转换

AFixedActor \* FixedActor = Cast<AFixedActor>(Target);

//缩小放大的动态效果

FVector NewScale = GetActorScale3D();

float Delta = (FMath::Sin(RunningTime + DeltaTime) - FMath::Sin(RunningTime));

FString str;

str = FString::SanitizeFloat(Delta);

GEngine->AddOnScreenDebugMessage(-1, 5.0f, FColor::Yellow, str);

NewScale.X += Delta \* 1.f ;

NewScale.Y += Delta\* 1.f;

NewScale.Z += Delta\* 1.f;

RunningTime += DeltaTime;

SetActorScale3D(NewScale);

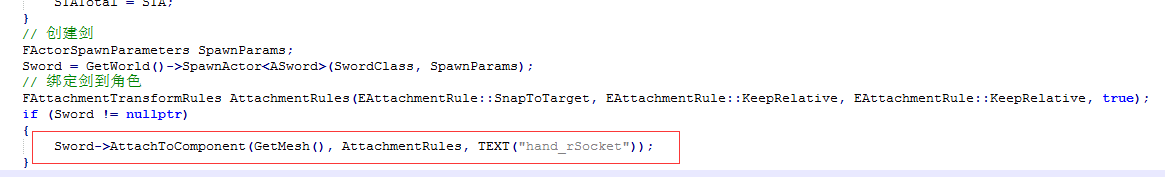
查找战场中所有指定类实例

UGameplayStatics::GetAllActorsOfClass(GetWorld(), ATank::StaticClass(), FoundEnemies);

创建组件

AttachTo只能在构造函数中使用

可以在函数中创建



将函数暴露给蓝图

1. UFUNCTION(BlueprintImplementableEvent)
2. **void** OnDepleted();

将参数暴露给蓝图

UPROPERTY(EditDefaultsOnly, Category = Projectile)

关闭渲染

GEngine->RedrawViewports(false);

当报错物体超出世界范围时，在世界设置中把这个关掉

Enable World Bounds Checks

从蓝图类生成实例

*UClass*\* TEXT1 = *LoadClass*<ATank>(*NULL*, *TEXT*("Blueprint'/Game/StarterContent/Blueprint/MyTank.MyTank\_C'"));

ATank\* SpawnedActor1 = (ATank\*)*GetWorld*()->*SpawnActor*(TEXT1, &\_Location);

当前运行时间

*GetWorld*()->*RealTimeSeconds*;

目标向前移动，方向为向量方向

*GetActorForwardVector*() \* \_MovementInput.*X* \* DeltaTime

打印数据

.h文件的头文件下面

*DECLARE\_LOG\_CATEGORY\_EXTERN*(LogTank, *Log*, *All*);

.cpp文件头文件下面

*DEFINE\_LOG\_CATEGORY*(LogTank);

使用

*UE\_LOG*(LogTank, *Warning*, *TEXT*("33333333，pitch %f, yaw %d, roll %f"), \_BarrelRotation.*Pitch*, \_BarrelRotation.*Yaw*, \_BarrelRotation.*Roll*);



**按键绑定**

InputComponent->BindAction("Jump", IE\_Pressed, **this**, &AFPSCharacter::StartJump);

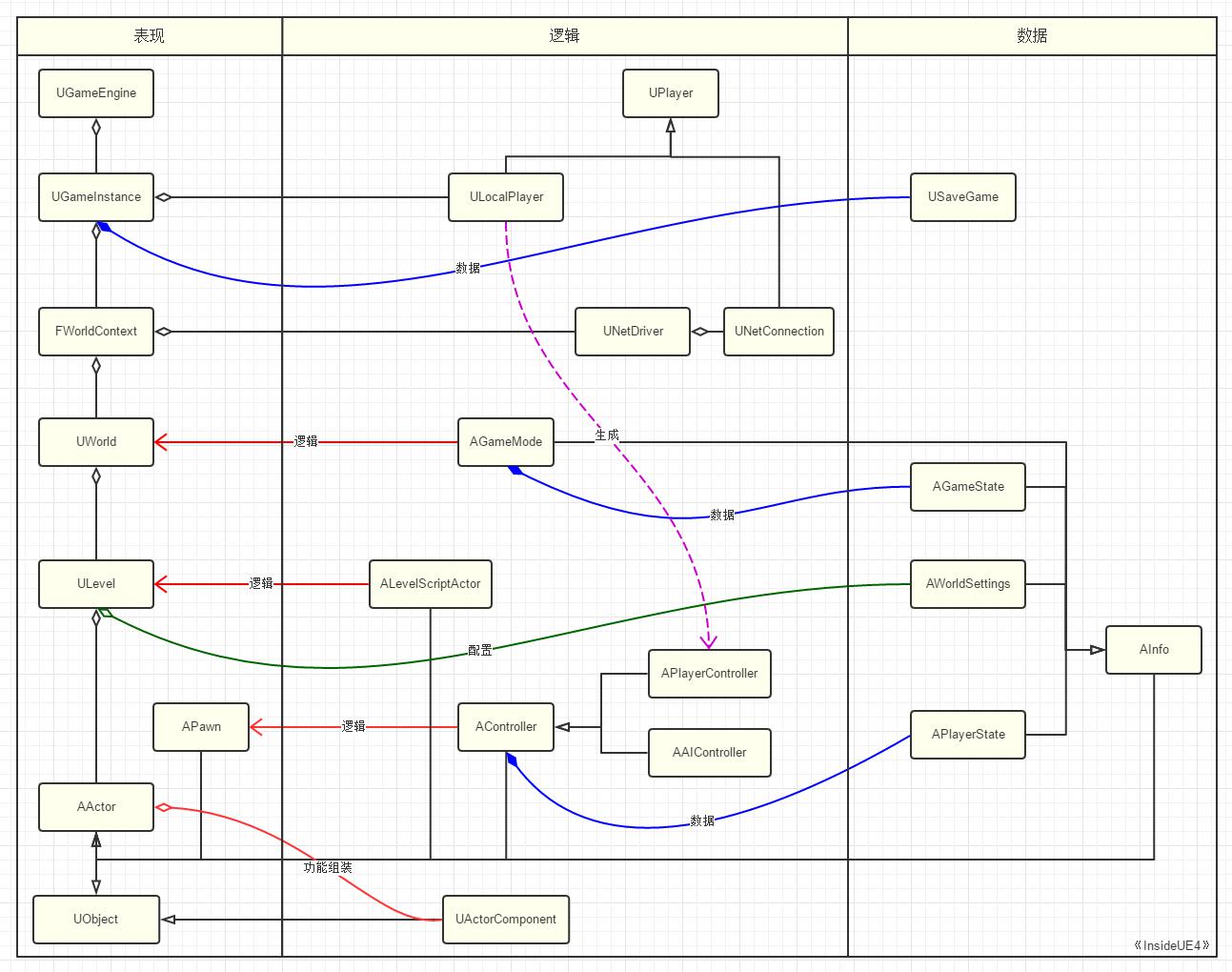
InputComponent->BindAction("Jump", IE\_Released, **this**, &AFPSCharacter::StopJump);

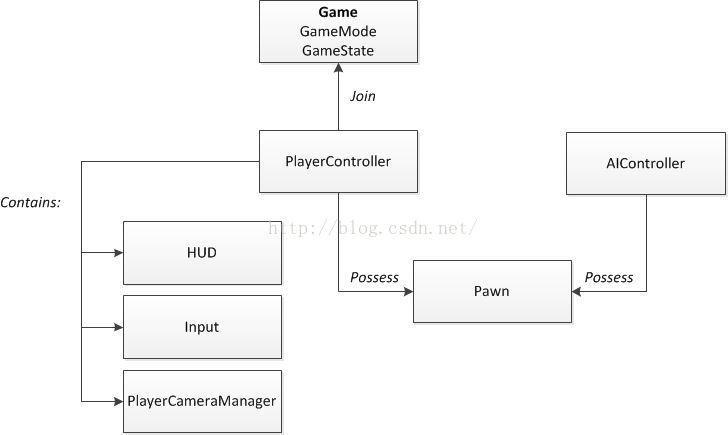
InputComponent->BindAxis("UpDown", this, &ARoamingCamera::CameraUp);

//控制默认玩家

AutoPossessPlayer = EAutoReceiveInput::Player0;

**MVC架构**

****



**判断Pawn是否被控制,可用于AI与Player控制的判断**

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

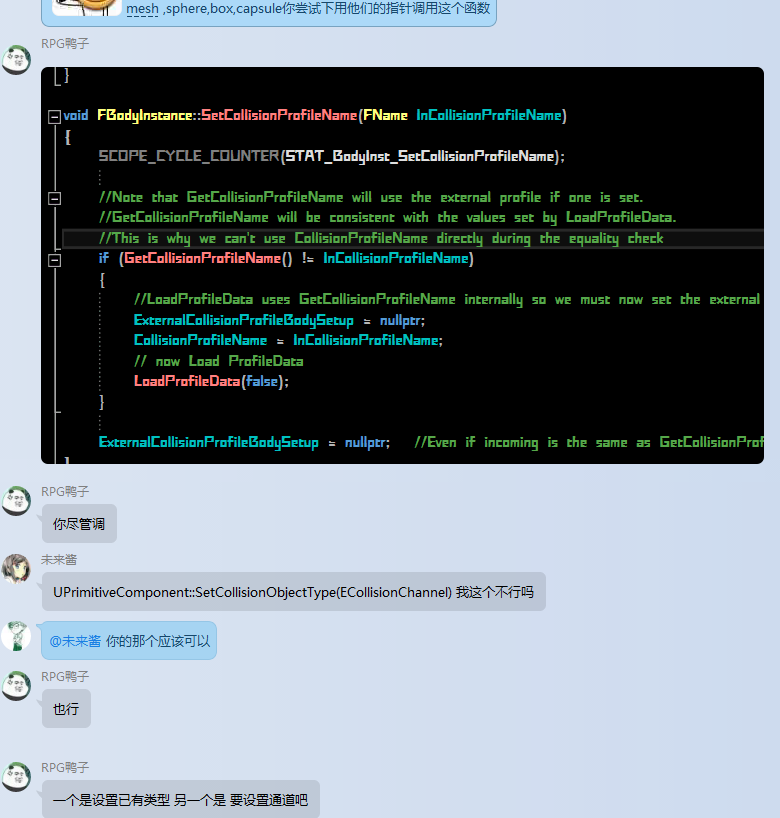
APlayerController\* MyController = Cast<APlayerController>(\*It);

if (!(MyController == Controller))

{

}

动态修改碰撞类型



**UE4：C++动态添加UStaticMeshComponent**

**1. 构造函数内添加**   
xxActor::xxActor()   
{   
//tank mesh   
static ConstructorHelpers::FObjectFinder TankStaticMesh(TEXT(“/Game/Meshs/TwinStick/TwinStickUFO.TwinStickUFO”));   
TankMesh = CreateDefaultSubobject(TEXT(“TankMesh”));   
TankMesh->SetStaticMesh(TankStaticMesh.Object);   
TankMesh->SetCollisionProfileName(UCollisionProfile::Pawn\_ProfileName);   
RootComponent = TankMesh;   
}

**2. 构造函数外添加（一定要添加RegisterComponent(),否则不会显示！）**   
bool AItemActor::InitComponent(FVector loc, FRotator rot)   
{   
bool bRtn = false;   
//mesh   
m\_pMeshComponent = (UStaticMeshComponent\*)NewObject(this, TEXT(“Mesh”));   
m\_pStaticMesh = CreateStaticMesh(m\_ItemType);   
if (m\_pMeshComponent && m\_pStaticMesh)   
{   
m\_pMeshComponent->SetStaticMesh(m\_pStaticMesh);   
m\_pMeshComponent->SetCollisionProfileName(UCollisionProfile::Pawn\_ProfileName);   
m\_pMeshComponent->SetRelativeRotation(rot);   
m\_pMeshComponent->SetRelativeLocation(loc);   
m\_pMeshComponent->RegisterComponent();   
m\_pMeshComponent->OnComponentHit.AddDynamic(this, &AItemActor::OnHit);

RootComponent = m\_pMeshComponent;

bRtn = true;

}

return bRtn;

Actor设置碰撞

TArray < UStaticMesh \*> &AllComponent = OtherActor->GetComponents;

for (int i = 0;i < AllComponent.GetTypeSize;i++)

{

if (AllComponent[i] )

{

\_TankBody->SetCollisionObjectType();

}

}

Actor设置隐藏

OtherActor ->SetActorHiddenInGame(true);

Actor设置无碰撞

OtherActor->SetActorEnableCollision(false);