**获取Pawn**

AActor \*MyActor = GetWorld()->GetFirstPlayerController()->GetPawn();

//获取玩家的Pawn

//若是在Controller里写代码可以直接GetPawn（），若是写在Pawn的蓝图类下的脚本中可以GetOwner（）

//每个Controller都对应一个Pawn，AIController对应AI的Pwan，玩家的Controller对应玩家的Pawn

//例如在AIController里获取自己的Pawn和玩家的Pawn：

ControlledTank = Cast<ATank>(GetPawn());

PlayerTank = Cast< ATank >(GetWorld()->GetFirstPlayerController()->GetPawn());

// Atank继承与Pawn，Pawn继承与Actor，所以获取到Pawn需要cast为Atank，Pawn可直接存储为Actor

**TriggerVolume的重叠事件**

ATriggerVolume \*TriggerVolume;

TArray<AActor\*> OverLappingActors;

TriggerVolume->GetOverlappingActors(OverLappingActors);

//获取发生重叠的所有物体

//要想触发Overlapping事件，需要给物体勾上GenerateOverlapEvents

for (AActor\*Actor: OverLappingActors)

{

float mass = Actor->FindComponentByClass<UPrimitiveComponent>()->GetMass();

//获取UPrimitiveComponent组件，然后获取质量

}

**C++调用蓝图的方法①**

DECLARE\_DYNAMIC\_MULTICAST\_DELEGATE(FOpenDoorEvents);

//UCLASS上方定义多播委托类型

UPROPERTY(BlueprintAssignable) //定义这个蓝图里才找的到

FOpenDoorEvents OpenDoorEvent;

//这里有两种情况

1.这个委托事件写在蓝图类的字组件上或者子c++脚本上

2.这个委托事件写在蓝图类的c++中（就是比如玩家的蓝图，一般是一个继承与Pawn的c++“例如Tank”，然后转为蓝图类，这样这个蓝图就继承与Tank和Pawn）

//1.在蓝图中选中这个C++脚本或组件，然后右键可以添加这个事件

//2.选中蓝图中的最最上层节点，邮件->绑定该事件之后才能使用

OpenDoorEvent.Broadcast();

//开始执行这个事件，事件具体节点在蓝图里

**插曲：C++里绑定自定义事件**

//Tank类里写：

DECLARE\_DYNAMIC\_MULTICAST\_DELEGATE(FTankDead);

//UCLASS上方定义多播委托类型

UPROPERTY(BlueprintAssignable)

FTankDead TankDeadEvent;

//定义具体事件变量

TankDeadEvent.Broadcast();

//当坦克死亡时候Broadcast给所有绑定这个事件的人

//TankAIController类里写：

ControlledTank->TankDeadEvent.AddUniqueDynamic(this, &ATankAIController::OnControlledTankDeathEvent);

//这里是写在TankAIController里的，要在这里绑定自己Tank身上的死亡事件，这样死亡事件Broadcast时候就能调用TankAIController里面的对应的方法

//这个委托事件在Tank类里写的，所以这里要Tank类指针到TankDeadEvent（Tank里定义的事件的名字）

//然后AddUniqueDynamic中第一个参数是绑定到哪个类，第二个参数是绑定到该类的函数地址

void ATankAIController::OnControlledTankDeathEvent()

{//这是死亡绑定的方法}

**C++调用蓝图的方法②**

//这种方式只能调用自己蓝图的函数，就是要将自己C++类转化为蓝图

//然后在蓝图写函数，在C++调用

//不能调用父组件的蓝图的函数

FString cmd = FString::Printf(TEXT("MyBlueprintFunc HeQian"));

FOutputDeviceDebug device;

CallFunctionByNameWithArguments(\*cmd, device, NULL, true);

UCLASS(Blueprintable)

//写了这个这个C++类就能转化为蓝图

**蓝图调用C++的方法**

UFUNCTION(BlueprintCallable,Category="MyCPPFunc")

void RotatePaoGuan();

//这样写蓝图中就能调用C++里的方法了

GetOwner()->SetActorRotation(FRotator(0,90,0));

//设置父组件的旋转

//此代码写在一个C++类里，这个类挂在一个Actor蓝图类物体身上

UPROPERTY(EditAnywhere, Category = "SetUp")

//定义这个可以在编辑器中调整值

UPROPERTY(VisibleAnywhere, Category = "Components")

//定义这个一般是代码刚生成的组件，但是还有些东西要手动设置，所以这样设置可以在编辑器中看到并设置组件右侧面板

//正常代码生成的组件右侧面板是空白的没法调整东西

//有时候写了之后蓝图里点击该组件还是右侧空白，需要重新生成蓝图

AActor \*Owner = GetOwner();

FString Name = Owner->GetName();

FVector Location = Owner->GetActorLocation();

UE\_LOG(LogTemp, Warning, TEXT("Name:%s , Location:%s"),\*Name,\*Location.ToString());

//固定的输出格式，Warning可以换为Error

UCLASS(ClassGroup =(Custom),meta=(BlueprintSpawnableComponent))

//在.h文件写这个可以在蓝图中添加这个C++类

//比如我继承于StaticMeshComponent生成一个C++类

//这样这个C++类添加在蓝图中后可以加模型显示

**绑定鼠标键盘输入按键映射**

//这些写在Controller里面

ATank\* ControlledTank = Cast<ATank>(GetPawn());

//Atank继承与Pwan，且玩家控制的物体蓝图继承与Atank

//蓝图右上角可以改继承

InputComponent = ControlledTank->FindComponentByClass<UInputComponent>();

if (InputComponent == nullptr)

{

UE\_LOG(LogTemp, Warning, TEXT("InputComponent is NULL"));

}

else

{

UE\_LOG(LogTemp, Warning, TEXT("InputComponent is %s"), \*InputComponent->GetName());

InputComponent->BindAction("Fire", EInputEvent::IE\_Pressed, this, &ATankPlayerController::Fire);

//安全校验后绑定我们在“项目设置”里面的“输入”里做的轴映射或者时间映射

//第二个参数是枚举类型表示绑定鼠标的按下还是抬起之类的，第四个参数是要绑定的函数方法的地址

InputComponent->BindAxis("TestAixs", this, &ATankPlayerController::TestFunc);

//绑定轴一样的，少一个参数，而且TestFunc绑定的方法要有输入float参数这样可以接受轴的返回值

}

**绑定撞击事件**

//先代码生成组件，也可以手动生成获取引用

StaticMeshComponent = CreateDefaultSubobject<UStaticMeshComponent>(FName("StaticMeshComponent"));

StaticMeshComponent->SetNotifyRigidBodyCollision(true);

//这个是设置StaticMesh下“Collision”->“SimulationGeneratesHitEvents”为勾选，这样才能产生撞击事件

StaticMeshComponent->OnComponentHit.AddDynamic(this, &ABullet::OnHit);

//绑定该脚本中的函数

//h文件中必须这样写，规矩

UFUNCTION()

void OnHit(UPrimitiveComponent \* HitComponent, AActor \* OtherActor, UPrimitiveComponent \* OtherComponent, FVector NormalImpulse, const FHitResult & Hit);

//cpp文件中实现该方法

void ABullet::OnHit(UPrimitiveComponent \* HitComponent, AActor \* OtherActor, UPrimitiveComponent \* OtherComponent, FVector NormalImpulse, const FHitResult & Hit)

{

StaticMeshComponent->SetNotifyRigidBodyCollision(false);

//这个是设置StaticMesh下“Collision”->“SimulationGeneratesHitEvents”为取消勾选，这样不会重复产生击中事件

}

**绑定重叠事件**

//先找到组件（UsphereComponent类就是SphereCollision组件）

SphereCollision = FindComponentByClass<USphereComponent>();

SphereCollision->SetGenerateOverlapEvents(true); //这个相当于Collision下的“GenerateOverlapEvent”打钩

//绑定该脚本中的函数

SphereCollision->OnComponentBeginOverlap.AddDynamic(this, &APickUpThing::OnOverlap);

//h文件中必须这样写，规矩

UFUNCTION()

void OnOverlap(class UPrimitiveComponent\* HitComp, class AActor\* OtherActor, class UPrimitiveComponent\* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const FHitResult & SweepResult);

//cpp文件中实现该方法

void APickUpThing::OnOverlap(class UPrimitiveComponent\* HitComp, class AActor\* OtherActor, class UPrimitiveComponent\* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const FHitResult & SweepResult)

{

ATank\* ControlledTank = Cast<ATank>(OtherActor); //看看是谁碰到了该物体，不是Tank的话就返回

if (ControlledTank == nullptr)

{

UE\_LOG(LogTemp, Error, TEXT("不是坦克碰到的该物体"));

return;

}

SphereCollision->SetGenerateOverlapEvents(false); //这个相当于GenerateOverlapEvent取消打钩防止反复触发

this->SetActorHiddenInGame(true);

}

**射线检测并抓起物体**

//这个抓取是抓正在物理模拟物的体的

UPhysicsHandleComponent \*PhysicsHandle = GetOwner()->FindComponentByClass<UPhysicsHandleComponent>();

//获取PhysicsHandle组件，这个组件要先被填加在玩家Pawn身上，这个C++脚本也在玩家Pwan身上

FVector PlayerLocation;

FRotator PlayerRotator;

GetWorld()->GetFirstPlayerController()->GetPlayerViewPoint(PlayerLocation, PlayerRotator);

//获取玩家摄像机的位置和旋转

FVector LineEndLocation = PlayerLocation + PlayerRotator.Vector() \* 100;

//向量加法，得到在相机前方的坐标，PlayerRotator.Vector()可以把旋转转为向量，旋转的x轴为向量前方

//测试下和GetForwardVector()发现结果都一样（GetForwardVector是世界坐标向量）

//因为旋转转为向量向量是唯一的，向量转旋转就只能让一个轴为0了

// FRotator.Vector()返回与向量指向的方向相对应的Frotator值。只改变Yaw和Pitch，将Roll为零，因为无法从矢量确定Roll轴。

FCollisionQueryParams QueryParams = FCollisionQueryParams("", false, GetOwner());

//固定写法，第二个参数是是否复杂碰撞，第三个参数是忽略检测谁（忽略自己）

FHitResult HitRes;

GetWorld()->LineTraceSingleByObjectType(

HitRes,

PlayerLocation,

LineEndLocation,

FCollisionObjectQueryParams(ECollisionChannel::ECC\_PhysicsBody),

QueryParams

);

//射线检测ByObjectType，就是给物体勾上SimulatePhysics后物体的ObjType就会是PhysicsBody

AActor \*RayCollisionActor = HitRes.GetActor();//获取射线撞击的Actor

UPrimitiveComponent \*PrimitiveComponent = HitRes.GetComponent();//获取射线撞击物体的PrimitiveComponent组件

if (RayCollisionActor!=nullptr)

{//开始抓取物体的方法，第二参数是骨骼名字，没有，后俩参数是要抓物体的位置和旋转

PhysicsHandle->GrabComponentAtLocationWithRotation(

PrimitiveComponent,

NAME\_None,

RayCollisionActor->GetActorLocation(),

RayCollisionActor->GetActorRotation()

);

UE\_LOG(LogTemp, Warning, TEXT("Grab Obj :%s"), \*RayCollisionActor->GetName());

}

else

{

UE\_LOG(LogTemp, Warning, TEXT("Nothing to Grab"));

}

//每一帧调用，来设置抓取物体的位置

if (PhysicsHandle->GetGrabbedComponent()!=nullptr)

{

PhysicsHandle->SetTargetLocation(LineEndLocation);

}

if (PhysicsHandle->GetGrabbedComponent()!=nullptr)

{

PhysicsHandle->ReleaseComponent();//释放物体

}

**从屏幕上指定点射线检测并旋转指向**

//Controller里的方法，可以获得当前屏幕的宽高

int32 ViewportSizeX, ViewportSizeY;

GetViewportSize(ViewportSizeX, ViewportSizeY);

//获取屏幕指定坐标在世界中的位置和方向向量

FVector WorldLocation;

FVector WorldDirection;

if (DeprojectScreenPositionToWorld(ViewportSizeX\*0.5f, ViewportSizeY\*0.2f, WorldLocation, WorldDirection))

{

//FVector StartPoint = PlayerCameraManager->GetCameraLocation();

//这是获取玩家相机的并没卵用在这里，因为已经获得屏幕点坐标了

FVector EndPoint = WorldLocation + WorldDirection \* RayCollosionLength;

FHitResult HitRes;

if (GetWorld()->LineTraceSingleByChannel(

HitRes,

WorldLocation,

EndPoint,

ECollisionChannel::ECC\_Visibility

))

{

//计算从炮管口指向射线检测点的向量

FVector FireVector = HitRes.Location - (PaoGuan->GetSocketLocation(FName("Fire")));

//然后旋转我们的炮台和炮管

RotatePaoGuan(FireVector); RotatePaoTai(FireVector);

}

}

//下面是物体炮台的旋转的具体方法

void ATankPlayerController::RotatePaoTai(FVector FireVector)

{

FRotator CurrentRotate = (PaoTai->GetForwardVector()).Rotation();

//获取当前世界旋转，GetForwardVector就是世界坐标向量

//也可以通过PaoTai->GetComponentRotation()直接获取

FRotator AimRotate = FireVector.GetSafeNormal().Rotation();//目标旋转

//转向目标所需要的转动的Yaw值

ChangeYaw = AimRotate.Yaw - CurrentRotate.Yaw;

//这里由于Yaw值在旋转过180后会突然变为负的，所以这里修正一下

if (ChangeYaw>180) {

ChangeYaw -= 360;

}

else if(ChangeYaw<-180)

{

ChangeYaw += 360;

}

//开始计算这一帧所要旋转的角度

float RelativeSpeed = FMath::Clamp<float>(ChangeYaw, -1, 1);

float RotateChange = RelativeSpeed \* 50 \* GetWorld()->DeltaTimeSeconds;

//新的旋转等于需要旋转的角度+物体坐标旋转，因为父物体会旋转，这里要是用世界坐标相加就会导致物体body一转炮台转的就不对了

float NewRotateYaw = RotateChange + PaoTai->GetRelativeRotation().Yaw;

//设置给物体的相对坐标旋转

//这里计算需要旋转的角度是用的世界坐标，但是相加和设置用的却是局部坐标

PaoTai->SetRelativeRotation(FRotator(0, NewRotateYaw, 0));

}

**在炮管前画Debug射线**

//这些代码写在Controller类里，所以可以直接GetPawn（），若是在

ATank\* ControlledTank = Cast<ATank>(GetPawn());

PaoGuan = ControlledTank->FindComponentByClass<UTankPaoGuanStaticMesh>();

FVector StartLocation;

FRotator StartRotator;

StartLocation = PaoGuan->GetSocketLocation(FName("Fire"));

StartRotator = PaoGuan->GetSocketRotation(FName("Fire"));

//FVector LineEndPosition = StartLocation + PaoGuan->GetForwardVector()\*1000;

FVector LineEndPosition = StartLocation + StartRotator.Vector() \* 2000;

//输出可以发现PaoGuan->GetSocketRotation(FName("Fire")).Vector()和PaoGuan->GetForwardVector()结果一样

//旋转转向量是以旋转的x轴为向量方向，forword也是以物体x轴朝向为返回向量值

DrawDebugLine(

GetWorld(),

StartLocation,

LineEndPosition,

FColor::Red,

false,

0.0f,

0.0f,

10.0f

);

//false参数是是否一致显示，最后一个参数是线宽度

//如果是要从视口发出射线，就要：

FVector PlayerLocation;

FRotator PlayerRotator;

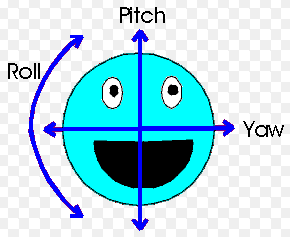
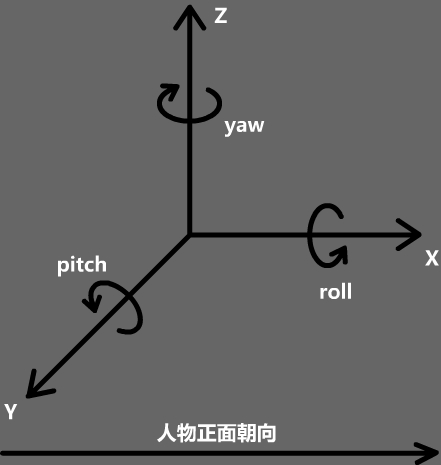
GetWorld()->GetFirstPlayerController()->GetPlayerViewPoint(PlayerLocation, PlayerRotator);

FVector LineEndPosition = PlayerLocation + PlayerRotator.Vector() \* 1000;

//不能用GetForwardVector()了，因为这个脚本写在Atank里Atank就是玩家Pawn蓝图，摄像机只是其中一个组件，GetForwardVector()必须是get的摄像机的前方向量才行

// FRotator.Vector()返回该旋转的向量，有了旋转，向量的话就确定了，不存在啥啥轴为0一说，会以旋转的x轴作为向量的朝前方向

// FVector.Rotation()返回与向量指向的方向相对应的Frotator值。只改变Yaw和Pitch，将Roll为零，因为无法从矢量确定Roll轴。



**int和float转为Fstring**

FString NewString = FString::FromInt(YourInt);

FString VeryCleanString = FString::SanitizeFloat(YourFloat);

UE\_LOG(LogTemp, Warning, TEXT("%s"),\* NewString);

**用蓝图初始化各种变量**

//h文件中这样写，才能在蓝图中调用这个c++方法来设置变量

UFUNCTION(BlueprintCallable, Category = "MyCPPFunc")

void Initialize( UStaticMeshComponent\* iStaticMeshComponent, UParticleSystemComponent\* iLaunchParticle);

void ABullet::Initialize(UStaticMeshComponent\* iStaticMeshComponent, UParticleSystemComponent\* iLaunchParticle)

{

StaticMeshComponent = iStaticMeshComponent;

LaunchParticle = iLaunchParticle;

}

**玩家进入观察者模式**

//以下写在继承与玩家Pawn的Tank类里

//GetWorld()->GetFirstPlayerController()->StartSpectatingOnly();

//这种写法可以直接让玩家进入观察者模式，但是里面设置过多，所以改为以下两个设置，让PlayerController和控制的Pawn脱离并控制新的Pawn

GetWorld()->GetFirstPlayerController()->ChangeState(NAME\_Spectating);

GetWorld()->GetFirstPlayerController()->PlayerState->bIsSpectator = true;

GetWorld()->GetFirstPlayerController()->Possess(this); //这个this是玩家Pawn（Tank类）

//这俩是重新设置刚刚进入观察者模式的设置来恢复

GetWorld()->GetFirstPlayerController()->ChangeState(NAME\_Playing);

GetWorld()->GetFirstPlayerController()->PlayerState->bIsSpectator = false;

//当重新恢复控制玩家Pawn之后，需要重新制定PlayerController里面的变量并且重新绑定事件

**玩敌人Pawn脱离Controller控制**

DetachFromControllerPendingDestroy();

**计时器**

//这是开始计时，5秒后调用绑定方法，handle是个必须传的参数，要在h文件声明，false参数是设置要不要循环调用

GetWorldTimerManager().SetTimer(handle, this, &ATank::TankReLife, 5.0f, false);

void ATank::TankReLife()//这个方法用来重新复活玩家，会恢复生命，重新制定PlayerController的Pawn，并且重新绑定事件等

{

GetWorldTimerManager().ClearTimer(handle);//清除计时器

}

//粒子激活播放，括号里设置false就是停止激活播放，或者直接DeadParticle->SetVisibility(false);

//自动播放就在面板里设置吧

DeadParticle->Activate();

**发出和受到伤害方法**

//第二个参数是伤害值，第三个是伤害的位置，第四个是受到伤害的半径，第五参数是固定写法，第六是忽略受伤害的数组

UGameplayStatics::ApplyRadialDamage(

this,

20,

GetActorLocation(),

RadialForceComponent->Radius,

UDamageType::StaticClass(),

TArray<AActor\*>()

);

//然后在可以要受到伤害的类里写的函数，必须这样写，规定死的

virtual float TakeDamage(float DamageAmount, struct FDamageEvent const& DamageEvent, class AController\* EventInstigator, AActor\* DamageCauser) override;

float ATank::TakeDamage(float DamageAmount, FDamageEvent const & DamageEvent, AController \* EventInstigator, AActor \* DamageCauser)

{ int32 DamagePoint = FPlatformMath::RoundToInt(DamageAmount);

//将伤害(float)通过四舍五入转为int，强制转型会向下取整

int32 DamageToApply = FMath::Clamp<int>(DamagePoint, 0, CurrentHp);

//防止把血打到负数

CurrentHp -= DamageToApply;

if (CurrentHp<=0&& isDead==false)//判断死了没

{

isDead = true;

if (GetController()==GetWorld()->GetWorld()->GetFirstPlayerController())//判断是不是主角死了

{

//GetWorld()->GetFirstPlayerController()->StartSpectatingOnly();

//这种写法可以直接让玩家进入观察者模式，但是里面设置过多，所以改为以下两个设置

GetWorld()->GetFirstPlayerController()->ChangeState(NAME\_Spectating);

GetWorld()->GetFirstPlayerController()->PlayerState->bIsSpectator = true;

//这是开始计时，5秒后调用绑定方法，handle是个必须传的参数，要在h文件声明，false参数是设置要不要循环调用

GetWorldTimerManager().SetTimer(handle, this, &ATank::TankReLife, 5.0f, false);

//PlayerController类里面才有这个开启观察者模式的方法，所以上面都要GetFirstPlayerController

PaoTai->SetVisibility(false);

PaoGuan->SetVisibility(false);

DeadParticle->Activate();

return DamageToApply;//必须返回造成的伤害这个值，方法函数的规矩

}

//不是主角死了，所以将Pwan的控制和Controller相分离，这样Controller就不能控制Pwan了

//然后是删除炮管和炮台播放燃烧特效

DetachFromControllerPendingDestroy();

PaoTai->DestroyComponent();

PaoGuan->DestroyComponent();

DeadParticle->Activate();

return DamageToApply;//必须返回造成的伤害这个值，方法函数的规矩，这个值可以改，比如自定义四舍五入啥的

}

**查找场景中的类**

TArray<AActor\*> PlayerStarts; //声明数组来装过会查找到的

UGameplayStatics::GetAllActorsOfClass(GetWorld(), APlayerStart::StaticClass(), PlayerStarts);

//第二个参数是想要查找的类，第三是装搜寻结果的数组（必须是Actor类型数组）

AActor\* SpawnTarget = PlayerStarts[0]; //场景里只有一个就去数组的第一个

//搜索场景里所有APlayerStart类的对象并存在数组里，然后取第一个就是我们的PlayerStart出生点

//以上写在Tank里，由于Tank类是动态生成的，所以需要这样搜寻，要是Tank蓝图直接放在场景里，那就直接暴露变量然后拖拽就行

**代码生成各种组件**

//这是投射物运动的组件，可以抛物线运动，已创建首先设置不自动激活因为过会要设置速度

ProjectileMovementComponent = CreateDefaultSubobject<UProjectileMovementComponent>(FName("ProjectileMovement"));

ProjectileMovementComponent->bAutoActivate = false;

//这是激活并设置投射速度的代码

ProjectileMovementComponent->SetVelocityInLocalSpace(FVector::ForwardVector \* Speed);

ProjectileMovementComponent->Activate();

//这是最常规的mesh组件

StaticMeshComponent = CreateDefaultSubobject<UStaticMeshComponent>(FName("StaticMeshComponent"));

SetRootComponent(StaticMeshComponent); //设置它为根组件

StaticMeshComponent->SetNotifyRigidBodyCollision(true);

//这个是设置StaticMesh下“Collision”->“SimulationGeneratesHitEvents”为勾选，这样才能产生撞击事件

StaticMeshComponent->SetVisibility(true);

//这个是设置StaticMesh下“Rendering”->“Visible”为勾选，这样才能渲染（这个默认是勾选的这里只是确保万一）

//这个是创造两个空的粒子组件

LaunchParticle = CreateDefaultSubobject<UParticleSystemComponent>(FName("LaunchParticle"));

LaunchParticle->AttachTo(RootComponent);

LaunchParticle->bAutoActivate = true;

//这个是设置“Activation”下的Auto Activate属性

ImpactParticle = CreateDefaultSubobject<UParticleSystemComponent>(FName("ImpactParticle"));

ImpactParticle->AttachTo(RootComponent);

ImpactParticle->bAutoActivate = false;

//这也可以写成ImpactParticle->SetAutoActivate（false）

//这是一个可以施加球体力的组件

RadialForceComponent = CreateDefaultSubobject<URadialForceComponent>(FName("RadialForceComponent"));

RadialForceComponent->AttachTo(RootComponent);

RadialForceComponent->bAutoActivate = false;

//这是调用发出球体力的方法，具体半径力大小在参数面板手动设置吧

RadialForceComponent->FireImpulse();

StaticMeshComponent->OnComponentHit.AddDynamic(this, &ABullet::OnHit);

//给StaticMeshComponent绑定OnHit事件，用它来检测碰撞

UPROPERTY(VisibleAnywhere, Category = "Components")

//代码生成的组件还要定义这个，因为有些东西要手动设置，这样设置可以在编辑器中看到并设置组件右侧面板

//正常代码生成的组件右侧面板是空白的没法调整东西

//有时候写了之后蓝图里点击该组件还是右侧空白，需要重新生成蓝图

**实例化子弹**

//h文件中声明一个在编辑器里可以被指定的类变量

UPROPERTY(EditAnywhere, Category = "SetUp")

TSubclassOf<ABullet> BulletClass;

//开火的方法函数

void ATankPlayerController::Fire()

{

//先判断有没有发弹冷却，FPlatformTime::Seconds()是当前时间

if ((FPlatformTime::Seconds()-LastFireTime)> LoadTime)

{

RadialForceComponent->FireImpulse();//对自己施加一个后坐力

//实例化子弹，第一个参数是要实例化的类，二三参数是实例化的位置旋转

ABullet \*Bullent = GetWorld()->SpawnActor<ABullet>(

BulletClass,

PaoGuan->GetSocketLocation(FName("Fire")),

PaoGuan->GetSocketRotation(FName("Fire"))

);

LastFireTime = FPlatformTime::Seconds();

}

}

**获取开始游戏总时间和上一针渲染时间**

FPlatformTime::Seconds();

GetWorld()->DeltaTimeSeconds;

**控制玩家移动（直接）**

//这个旋转方法写在Controller里

void ATankPlayerController::RotateD(float value)

{

//获取玩家当前旋转

FRotator CurrentRotate = GetPawn()->GetActorRotation();

//直接计算一个值加到现在旋转的Yaw上

float RotateChange = value \* 80 \* GetWorld()->DeltaTimeSeconds;

float NewRotateYaw = RotateChange + CurrentRotate.Yaw;

//设置新旋转

GetPawn()->SetActorRotation(FRotator(0, NewRotateYaw, 0));

}

//这个旋转方法写在玩家Tank蓝图的子组件上

void UTankTrackStaticMesh::MoveW(float value)

{

//利用AddActorLocalOffset方法

ControlledTank->AddActorLocalOffset(FVector(WValue \* 6.0f, 0.0f, 0.0f));

}

**控制玩家移动（物理）**

void ATankPlayerController::PhysicsMoveForward(float value)

{

//控制前后移动，这个方法绑定了w和s两个按键轴，会返回-1到1之间的值

LvDai1Move(value); LvDai2Move(value); FixedFroce();

}

void ATankPlayerController::PhysicsMoveH(float value)

{

//控制左右转弯，这个方法绑定了a和d两个按键轴，会返回-1到1之间的值

//这里通过对左右履带施加相反的力来达到旋转的目的

LvDai1Move(value); LvDai2Move(-value);

}

//给TankBody在其中一个履带的位置施加力

void ATankPlayerController::LvDai1Move(float value)

{

//首先施加力的话是给根组件（TankBodyMesh）的PrimitiveComponent组件加力，所以先获取

UPrimitiveComponent \*PrimitiveComponent = Cast<UPrimitiveComponent>(GetPawn()->GetRootComponent());

if (PrimitiveComponent == nullptr) {

UE\_LOG(LogTemp, Warning, TEXT("PrimitiveComponent is Cast Fail"));

return;

}

//计算出一个要施加力的大小和方向

FVector Force = value \* mul \* (GetPawn()->GetActorForwardVector());

//在履带a的Socket的位置施加力

PrimitiveComponent->AddForceAtLocation(Force, LvDaia->GetSocketLocation(FName("Force")));

}

//给TankBody在另一个履带的位置施加力

void ATankPlayerController::LvDai2Move(float value)

{

UPrimitiveComponent \*PrimitiveComponent = Cast<UPrimitiveComponent>(GetPawn()->GetRootComponent());

if (PrimitiveComponent == nullptr) {

UE\_LOG(LogTemp, Warning, TEXT("PrimitiveComponent is Cast Fail"));

return;

}

FVector Force = value \* mul \* (GetPawn()->GetActorForwardVector());

//在履带a的Socket的位置施加力

PrimitiveComponent->AddForceAtLocation(Force, LvDaib->GetSocketLocation(FName("Force")));

}

//该方法来施加力放置坦克漂移

void ATankPlayerController::FixedFroce()

{

//首先获取根组件的PrimitiveComponent组件

UPrimitiveComponent \*PrimitiveComponent = Cast<UPrimitiveComponent>(GetPawn()->GetRootComponent());

//坦克现在的速度矢量

FVector TankV = PrimitiveComponent->GetComponentVelocity();

//向右的单位矢量

FVector RightV = PrimitiveComponent->GetRightVector();

//点乘得到速度矢量在右方向的投影长度标量

float SideSpeed = FVector::DotProduct(RightV, TankV);

//右方向速度大小乘以右单位矢量得到右方向矢量，然后变成负的（还除了DeltaTime不很懂）

FVector SideSpeedA = SideSpeed / (GetWorld()->DeltaTimeSeconds)\*RightV\*-1;

//F = m a 后面的0.4本来是每个履带都要施加这么一个力，那么每个履带就是0.5，但是这个方法调用是在两个履带施加力之后，那就当他是系数吧

FVector FixedForce = SideSpeedA \* PrimitiveComponent->GetMass()\*0.4f;

PrimitiveComponent->AddForce(FixedForce);

}

**AI自动寻路**

//在AIController里调用这个方法，然后再控制的Pawn的蓝图里的Nav组件里调用另一个方法来接收移动信息

//第一个参数是要朝谁走，第二参数是多远距离停下来

PlayerTank = Cast<ATank>(GetWorld()->GetFirstPlayerController()->GetPawn());

MoveToActor(PlayerTank,5);

//在继承Nav的c++里要重写这个方法

virtual void RequestDirectMove(const FVector& MoveVelocity, bool bForceMaxSpeed) override;

//下面是Nav的cpp文件中具体操作AI移动的代码

void UTankNavMovement::RequestDirectMove(const FVector & MoveVelocity, bool bForceMaxSpeed)

{

//这个函数会传进来需要朝着哪个方向走的MoveVelocity，我们将其标准化

//并且获得自身的前方标准向量和右侧标准向量

FVector MoveVelocityNormal = MoveVelocity.GetSafeNormal();

FVector AIForwardNormal = GetOwner()->GetActorForwardVector().GetSafeNormal();

FVector AIRightNormal = GetOwner()->GetActorRightVector().GetSafeNormal();

//然后通过点乘来获取到我们速度单位矢量在这俩方向上的投影

float ForwardSpeed = FVector::DotProduct(MoveVelocityNormal, AIForwardNormal);

float RightdSpeed = FVector::DotProduct(MoveVelocityNormal, AIRightNormal);

//在正方向上的投影就直接设置为AI向前走的量

//AddActorLocalOffset参数里填的也是局部坐标的位移

ControlledTank->AddActorLocalOffset(FVector(ForwardSpeed \* 6.0f, 0.0f, 0.0f));

//用在侧方向上的投影来设置角度旋转

FRotator CurrentRotate = ControlledTank->GetActorRotation();

float RotateChange = RightdSpeed \* 80 \* GetWorld()->DeltaTimeSeconds;

float NewRotateYaw = RotateChange + CurrentRotate.Yaw;

ControlledTank->SetActorRotation(FRotator(0, NewRotateYaw, 0));

}