PawnTank.h

```
/*
 *****************************************
	Crazy Tank - Driving/shooting game prototype
	By James Romero. Made with Unreal Engine 4
	2021
 *****************************************
 */

#pragma once

#include "CoreMinimal.h"
#include "PawnBase.h"
#include "PawnTank.generated.h"

/*

	Engine classes

*/

class USpringArmComponent;
class UCameraComponent;

/*

	Crazy Tank classes

*/

class AGunBase;

// Delegate to notify suscribed classes when the current Tank's regular
projectiles amount has changed
DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FOnProjectileCountChanged, int32,
ProjectileCount);

// Delegate to notify suscribed classes when the current Tank's homing
projectiles amount has changed
DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FOnHomingProjectileCountChanged,
int32, HomingProjectileCount);

//////////////////////////////////////////////////////////////////////////////
//
// This class handles the Tank's behavior (moving, attacking and destruction)
//
//////////////////////////////////////////////////////////////////////////////
UCLASS()
class CRAZYTANK_API APawnTank : public APawnBase
{
```

```cpp
    GENERATED_BODY()

private:

    /*
        VARIABLES
    */

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
    USpringArmComponent* SpringArm = nullptr;

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
    UCameraComponent* Camera = nullptr;

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
    UParticleSystemComponent* ParticleTrail = nullptr; // Dust trail made by the Tank when moving

    FVector MoveDirection = FVector::ZeroVector;

    FQuat RotationDirection = FQuat::Identity; // The Tank's body rotation direction given by the WASD keys input

    FQuat CounterRotation = FQuat::Identity; // The Tank's turret rotation direction given by the mouse input

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Movement", meta = (AllowPrivateAccess = "true"))
    float MoveSpeed = 100.0f;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Movement", meta = (AllowPrivateAccess = "true"))
    float TurnSpeed = 100.0f;

    APlayerController* PlayerControllerRef = nullptr;

    bool bIsPlayerAlive = true;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Movement", meta = (AllowPrivateAccess = "true"))
    float GroundRayLength = 20.0f; // Raycast length for checking if the Tank is grounded (touching the floor)

    float TankGravity = 10.0f; // Tank's custom down force

    float DragOnGround = 1.5f; // Drag force experimented by the Tank when moving on ground
```

```
	bool bIsGrounded = false;

	UPROPERTY(EditDefaultsOnly)
	TSubclassOf<AGunBase> GunClass; //Blueprint GunActor class to spawn

	UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "Projectile Type", 
meta = (AllowPrivateAccess = "true"))
	TSubclassOf<AProjectileBase> HomingProjectileClass;

	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Components", 
meta = (AllowPrivateAccess = "true"))
	USceneComponent* HomingProjectileSpawnPoint = nullptr; //visual 
representation of where homing projectiles will be spawned from when fired

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Projectile Type", 
meta = (AllowPrivateAccess = "true"))
	int ProjectileAmmoMax = 6;

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Projectile Type", 
meta = (AllowPrivateAccess = "true"))
	int HomingProjectileAmmoMax = 4;

	int ProjectileAmmoCurrent;

	int HomingProjectileAmmoCurrent;

	bool bIsFiringRifle = false; // To know if the Tank is firing its gun

	UPROPERTY();
	AGunBase* Gun = nullptr; // Here we will store the actual Gun instance

	TArray<AActor*> HomingTarget; // Selected targets array that will be 
attacked with a homing projectile

	/*
		METHODS
	*/

	void CalculateMoveInput(float value); // Calculate the Tank's capsule 
component movement from keyboard input and move speed

	// Calculate the Tank's body rotation from keyboard input and turn speed
	void CalculateRotateInput(float value); // Also calculates the counter 
rotation for the Tank's turret from the results of the body rotation

	void RotateView(float value); // Calculates and applies the Tank's turret 
rotation from mouse input and turn speed

	// Raycast down from the Tank's body to know if it's grounded and align 
its body to the surface if that's the case
	void Move(); // Also applies a force to move the Tank if it's grounded or
```

```
a down force (gravity) in case it's not

	// Applies the rotation and counter rotation of the Tank's base and
turret, only if the Tank is moving first, if not it'll not rotate
	void Rotate(); // Also manages a dust particle system when the Tank is
moving

	void FireRifle(); // Activates the firing of the Tank's gun if there's a
Gun Class assigned

	// Sends a raycast to find enemies to target for the Tank's homing
projectile
	void TargetHomingProjectile(); // Also draws an outline to every found
target

	void FireHomingProjectile(); // Spawns and shoots a homing projectile for
every found target

	void DrawTargetOutline(AActor* Target, bool bShouldDraw); // Draws an
outline to every found target mesh

	virtual void Fire() override; // Activates the firing of the Tank's
regular projectiles using the "PawnBase" parent class virtual method

	// Calculates the current ammo of a projectile (homing or regular)
depending on whether the player is shooting or recolecting ammo pick ups
	int ProcessNewAmmo(int CurrentAmmo, int AddedAmount, int MaxAmmo);

public:

	/*
		METHODS
	*/

	APawnTank(); // Sets default values for this pawn's properties

	virtual void Tick(float DeltaTime) override; // Called every frame

	// Called to bind functionality to input
	virtual void SetupPlayerInputComponent(class UInputComponent*
PlayerInputComponent) override;

	virtual void HandleDestruction() override; // Manages this pawn's
behaviour when it's destroyed

	bool GetIsPlayerAlive(); // Getter for the bIsPlayerAlive variable

	// Adds ammo to a specified type of projectile (homing or regular)
	void AddAmmo(int AmmoType, int Amount); // This method is public because
it's used in the Pick Up classes
```

```
	// Delegate to notify suscribed classes when the current Tank's regular projectiles amount has changed
	UPROPERTY(BlueprintAssignable, BlueprintCallable, Category = "Delegates")
		FOnProjectileCountChanged OnProjectileCountChanged;

	// Delegate to notify suscribed classes when the current Tank's homing projectiles amount has changed
	UPROPERTY(BlueprintAssignable, BlueprintCallable, Category = "Delegates")
		FOnHomingProjectileCountChanged OnHomingProjectileCountChanged;

protected:

	/*
		METHODS
	*/

	// Called when the game starts or when spawned
	virtual void BeginPlay() override;

};
```

PawnTank.h

PawnTank.cpp

```cpp
/*
 *****************************************
      Crazy Tank - Driving/shooting game prototype
      By James Romero. Made with Unreal Engine 4
      2021
 *****************************************
 */

#include "PawnTank.h"
#include "GameFramework/SpringArmComponent.h"
#include "Camera/CameraComponent.h"
#include "Components/CapsuleComponent.h"
#include "DrawDebugHelpers.h"
#include "Kismet/KismetMathLibrary.h"
#include "Particles/ParticleSystemComponent.h"
#include "CrazyTank/Actors/GunBase.h"
#include "CrazyTank/Actors/ProjectileBase.h"

////////            Sets default values for this pawn's properties      ////////
APawnTank::APawnTank()
{
      // Set this pawn to call Tick() every frame.  You can turn this off to
improve performance if you don't need it.
      PrimaryActorTick.bCanEverTick = true;

      ParticleTrail =
CreateDefaultSubobject<UParticleSystemComponent>(TEXT("Tank Smoke Trail"));
      ParticleTrail->SetupAttachment(BaseMesh);

      HomingProjectileSpawnPoint =
CreateDefaultSubobject<USceneComponent>(TEXT("Homing Projectile Spawn Point"));
      //We want the HomingProjectileSpawnPoint to inherit the movement and
rotation of the TurretMesh
      HomingProjectileSpawnPoint->SetupAttachment(TurretMesh);

      SpringArm = CreateDefaultSubobject<USpringArmComponent>(TEXT("Spring
Arm"));
      //SpringArm->SetupAttachment(RootComponent);
      SpringArm->SetupAttachment(TurretMesh);

      Camera = CreateDefaultSubobject<UCameraComponent>(TEXT("Camera"));
      Camera->SetupAttachment(SpringArm);

}
////////////////////////////////////////////////////////////////////////////

////////            Called when the game starts or when spawned         ////////
void APawnTank::BeginPlay()
{
	Super::BeginPlay();

	PlayerControllerRef = Cast<APlayerController>(GetController());
	if (PlayerControllerRef)
	{
		PlayerControllerRef->bShowMouseCursor = false;
	}

	if (GunClass)
	{
		//Spawning a blueprint child of the GunActor class
		Gun = GetWorld()->SpawnActor<AGunBase>(GunClass);
		Gun->AttachToComponent(TurretMesh,
FAttachmentTransformRules::KeepRelativeTransform, TEXT("WeaponSocket"));

		//Set up the Gun to have this class as its owner (not in the sense
of transforms, but like, for multiplayer or damaging
		// when you need to know which player have which weapon)
		Gun->SetOwner(this);
	}

	ParticleTrail->DeactivateSystem();

	ProjectileAmmoCurrent = ProjectileAmmoMax;
	HomingProjectileAmmoCurrent = HomingProjectileAmmoMax;

	// Delegate to notify suscribed classes when the current Tank's regular
projectiles amount has changed
	OnProjectileCountChanged.Broadcast(ProjectileAmmoCurrent);

	// Delegate to notify suscribed classes when the current Tank's homing
projectiles amount has changed
	OnHomingProjectileCountChanged.Broadcast(HomingProjectileAmmoCurrent);

}
////////////////////////////////////////////////////////////////////////////

////////		Called every frame		////////
void APawnTank::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

	Rotate();
	Move();
}
/////////////////////////////////////////////

////////		Called to bind functionality to input		////////
void APawnTank::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
{
	Super::SetupPlayerInputComponent(PlayerInputComponent);
	PlayerInputComponent->BindAxis("MoveForward", this,
&APawnTank::CalculateMoveInput);
	PlayerInputComponent->BindAxis("Turn", this,
&APawnTank::CalculateRotateInput);
	PlayerInputComponent->BindAxis("RotateTurret", this,
&APawnTank::RotateView);
	PlayerInputComponent->BindAction("FireProjectile", IE_Pressed, this,
&APawnTank::Fire);
	PlayerInputComponent->BindAction("FireGun", IE_Pressed, this,
&APawnTank::FireRifle);
	PlayerInputComponent->BindAction("TargetHomingProjectile", IE_Pressed,
this, &APawnTank::TargetHomingProjectile);
	PlayerInputComponent->BindAction("FireHomingProjectile", IE_Pressed,
this, &APawnTank::FireHomingProjectile);
}
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////

////////		Calculate the Tank's capsule component movement from
keyboard input and move speed		////////
void APawnTank::CalculateMoveInput(float value)
{
	//Always move forward (where the base of the Tank is front facing)
	MoveDirection = value * BaseMesh->GetForwardVector() * MoveSpeed *
GetWorld()->DeltaTimeSeconds;
}
////////////////////////////////////////////////////////////////////////////////
////////////////////////////

////////		Calculate the Tank's body rotation from keyboard input and
turn speed		////////
////////		Also calculates the counter rotation for the Tank's turret
from the results of the body rotation		////////
void APawnTank::CalculateRotateInput(float value)
{
	// Calculates rotation amount from player input and turn speed
	float RotateAmount = value * TurnSpeed * GetWorld()->DeltaTimeSeconds;

	// saves Tank's base rotation and turret counter rotation around yaw/up
vector
	FRotator Rotation = FRotator(0, RotateAmount, 0);
	FRotator Counter = FRotator(0, -RotateAmount, 0);

	// Converts and saves the results as Quaternions
	RotationDirection = FQuat(Rotation);
	CounterRotation = FQuat(Counter);
}
////////////////////////////////////////////////////////////////////////////////
////////////////////////////

////////		Calculates and applies the Tank's turret rotation from mouse
```

```cpp
input and turn speed             ////////
void APawnTank::RotateView(float value)
{
	float RotateAmount = value * TurnSpeed * GetWorld()->DeltaTimeSeconds;
	FRotator Rotation = FRotator(0, RotateAmount, 0); //rotate around yaw/up 
vector
	TurretMesh->AddLocalRotation(Rotation);
}
////////////////////////////////////////////////////////////////////////////////
////////////////////////////

////////            Raycast down from the Tank's body to know if it's grounded 
and align its body to the surface if that's the case       ////////
////////            Also applies a force to move the Tank if it's grounded or a 
down force (gravity) in case it's not         ////////
void APawnTank::Move()
{
	bIsGrounded = false;
	FHitResult Hit;

	// This parameters are for indicating the Line Trace (Raycast) that we 
don't need complex collision while tracing and that this class is the
	// Trace's owner, so it must ignore it
	FCollisionQueryParams TraceParams(TEXT("LineOfSight_Trace"), false, 
this);

	// Visual representation of the Line Trace for debugging purposes
	DrawDebugLine
	(
		GetWorld(),
		BaseMesh->GetComponentLocation(),
		BaseMesh->GetComponentLocation() + BaseMesh->GetUpVector() * 
-GroundRayLength,
		FColor::Yellow,
		false,
		-1,
		0,
		2.0f
	);

	// Perform the Line Trace down and save its results as a bool
	bool bTraceResult = GetWorld()->LineTraceSingleByChannel
	(
		Hit,
		BaseMesh->GetComponentLocation(),
		BaseMesh->GetComponentLocation() + BaseMesh->GetUpVector() * 
-GroundRayLength,
		ECollisionChannel::ECC_WorldStatic,
		TraceParams,
		FCollisionResponseParams::DefaultResponseParam
	);
```

```
        if (bTraceResult)
        {
                // If the Trace hits something, it means the Tank is grounded
                bIsGrounded = true;

                // Align the Tank to surface using the Trace's hit-point normal
                FRotator SurfaceAlignment =
UKismetMathLibrary::MakeRotFromZX(Hit.ImpactNormal,
BaseMesh->GetForwardVector());
                // Apply the alignment to the Tank's base
                BaseMesh->SetWorldRotation(SurfaceAlignment);
        }

        if (bIsGrounded)
        {
                // If the Tank is grounded, apply a force to its capsule to drive
and some drag for a better feeling of the movement
                CapsuleComp->SetLinearDamping(DragOnGround);
                CapsuleComp->AddForce(MoveDirection * 70000.0f);
        }
        else
        {
                // If the Tank isn't grounded, apply a downward force to its
capsule that'll act as the gravity and some air drag
                CapsuleComp->SetLinearDamping(0.1f);
                CapsuleComp->AddForce(FVector::UpVector * -TankGravity * 20000.0f);
        }
}
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////

////////            Applies the rotation and counter rotation of the Tank's base
and turret,         ////////
////////            only if the Tank is moving first, if not it won't rotate
        ////////
void APawnTank::Rotate()
{
        if (MoveDirection != FVector::ZeroVector && bIsGrounded)
        {
                // If the Tank is moving and is grounded, emit a dust particle
trail
                if (ParticleTrail->bWasDeactivated && !ParticleTrail->bWasActive)
                {
                        ParticleTrail->Activate(true);
                }

                if(MoveDirection.X > 0.0f)
                {
                        // If the Tank is moving forward, it can rotate its base
                        BaseMesh->AddLocalRotation(RotationDirection, true);
```

```
            // For decoupling the turret's rotation from the base's
rotation (so we can move the turret freely with the mouse)
            // We apply a counter rotation to the turret (opposite to
the base's)
            TurretMesh->AddLocalRotation(CounterRotation, true);
        }
        else
        {
            // If the Tank isn't moving forward, stop the rotation of
its base and the counter of the turret
            BaseMesh->AddLocalRotation(RotationDirection * -1, true);
            TurretMesh->AddLocalRotation(CounterRotation * -1, true);
        }
    }
    else
    {
        // If the Tank isn't moving or isn't grounded, deactivate the
emission of the dust particle trail
        if(!ParticleTrail->bWasDeactivated)
        {
            ParticleTrail->bSuppressSpawning = true;
            ParticleTrail->Deactivate();
        }
    }
}
////////////////////////////////////////////////////////////////////////////////
///////////////

////////            Activates the firing of the Tank's gun if there's a Gun
Class assigned              ////////
void APawnTank::FireRifle()
{
    if (GunClass)
    {
        Gun->PullTrigger();
    }
}
////////////////////////////////////////////////////////////////////////////////
///////////////

////////    ////            Manages this pawn's behaviour when it's
destroyed           /////////////////////////
void APawnTank::HandleDestruction()
{
    //Call "PawnBase" class HandleDestruction() to play effects
    Super::HandleDestruction();

    //// Overriding logic in this child class ////

    bIsPlayerAlive = false;
```

```
	//Hide any visual component of the Actor
	SetActorHiddenInGame(true);

	//Stop running Tick functionality to save some performance and also stop
movement and rotation
	SetActorTickEnabled(false);
}
////////////////////////////////////////////////////////////////////////////////
////////////////////

//////////          Getter for the bIsPlayerAlive variable        //////////
bool APawnTank::GetIsPlayerAlive()
{
	return bIsPlayerAlive;
}
//////////////////////////////////////////////////////////////////////

////////////////          Adds ammo to a specified type of projectile (homing
or regular)         ////////////////
void APawnTank::AddAmmo(int AmmoType, int Amount)
{
	switch(AmmoType)
	{
		//Projectile
		case 0:
			ProjectileAmmoCurrent =
ProcessNewAmmo(ProjectileAmmoCurrent, Amount, ProjectileAmmoMax);

			// Notify the subscribed classes that the Tank is changing
its current regular projectile count
			OnProjectileCountChanged.Broadcast(ProjectileAmmoCurrent);

			break;

		//Homing Projectile
		case 1:
			HomingProjectileAmmoCurrent =
ProcessNewAmmo(HomingProjectileAmmoCurrent, Amount, HomingProjectileAmmoMax);

			// Notify the subscribed classes that the Tank is changing
its current homing projectile count

OnHomingProjectileCountChanged.Broadcast(HomingProjectileAmmoCurrent);

			break;

		default:
			return;
			break;
	}
```

```
}
////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////

///////////////////       Sends a raycast to find enemies to target for the
Tank's homing projectile         ///////////////////
////////            Also draws an outline to every found target         ////////
void APawnTank::TargetHomingProjectile()
{
	if (HomingProjectileAmmoCurrent > 0)
	{
		// If the Tank currently have some homing projectiles ammo, it'll
send a forward Line Trace to find enemy targets
		FHitResult HitRes = FHitResult();
		TArray<TEnumAsByte<EObjectTypeQuery>> ObjectsToTarget; // Any
traced enemy will be saved in this array
		ObjectsToTarget.Add(ObjectTypeQuery1); // Static Mesh object type
		FVector EndPointTrace =
projectileSpawnPoint->GetComponentLocation() +
(projectileSpawnPoint->GetForwardVector() * 100000.0f);

		// visual representation of the trace for debbuging purposes
		DrawDebugLine(GetWorld(),
projectileSpawnPoint->GetComponentLocation(), EndPointTrace, FColor::Yellow,
false, 0.5, 0, 2.0f);

		// Perform the Line Trace and save its results as a bool
		bool bTargetFound = GetWorld()->LineTraceSingleByObjectType
		(
			HitRes,
			projectileSpawnPoint->GetComponentLocation(),
			EndPointTrace,
			ObjectsToTarget
		);

		UE_LOG(LogTemp, Error, TEXT("%s"), bTargetFound ? TEXT("true") :
TEXT("false"));

		if (!bTargetFound)
		{
			// If the trace doesn't find anything, exit the function
			return;
		}

		if (!HitRes.GetActor()->IsA(APawn::StaticClass()))
		{
			// If the trace hits something but it's not a Pawn, discard
all previous found targets (if any) and exit the function
			// This could be unnecessary however, it all depends on the
gameplay's goals
			if (HomingTarget.Num() != 0)
```

```
                    {
                            for (int32 index = 0; index != HomingTarget.Num();
index++)
                            {
                                    DrawTargetOutline(HomingTarget[index], false);
                            }
                            HomingTarget.Empty();
                    }
                    return;
            }

            UE_LOG(LogTemp, Error, TEXT("TARGET IS %s"),
*HitRes.GetActor()->GetName());


            if (HomingTarget.Num() >= HomingProjectileAmmoCurrent)
            {
                    // If we're trying to find more targets but the Tank hasn't
enough homing projectile ammo, exit the function
                    UE_LOG(LogTemp, Error, TEXT("No enough Ammo for more Homing
Targets!"));
                    return;
            }

            // if we're trying to target an enemy that is already targeted,
exit the function
            for (int32 index = 0; index != HomingTarget.Num(); index++)
            {
                    if(HitRes.GetActor() == HomingTarget[index])
                    {
                            return;
                    }

            }

            HomingTarget.Add( HitRes.GetActor() );
            DrawTargetOutline(HomingTarget.Last(), true);
    }
    else
    {
            // If we're trying to find targets but the Tank hasn't any homing
projectile ammo, exit the function
            UE_LOG(LogTemp, Error, TEXT("No Ammo for Homing!"));
    }

}
////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////

////////////////////////                Spawns and shoots a homing projectile
for every found target                  ////////////////////////
```

```cpp
void APawnTank::FireHomingProjectile()
{
        if (HomingTarget.Num() == 0)
        {
                // If there're not targets, exit the function
                return;
        }

        if (HomingProjectileClass)
        {
                FVector SpawnLocation =
HomingProjectileSpawnPoint->GetComponentLocation();
                FRotator SpawnRotation = FRotator(0.0f, 0.0f, 0.0f);

                for (int32 index = 0; index < HomingTarget.Num(); index++)
                {
                        // Spawn a homing projectile for every target found
                        AProjectileBase* TempProjectile =
GetWorld()->SpawnActor<AProjectileBase>(HomingProjectileClass,SpawnLocation,
SpawnRotation);
                        TempProjectile->SetOwner(this); // Set the Tank as the
projectile's owner for avoiding unwanted Tank-projectile collisions

                        // Stop drawing the outline in the found targets when the
projectiles are going to be fired
                        DrawTargetOutline(HomingTarget[index], false);

                        // Call the homing projectile's function from its class to
manage its firing, passing every target found
                        TempProjectile->HomingProjectile(HomingTarget[index]);

                        // When a projectile is shot to a target, remove that target
from the found array and rearrange it
                        HomingTarget.RemoveAtSwap(index);

                        // Update the current homing projectile ammo count after
every shot and notify subscribed classes about that change
                        HomingProjectileAmmoCurrent =
ProcessNewAmmo(HomingProjectileAmmoCurrent, -1, HomingProjectileAmmoMax);

OnHomingProjectileCountChanged.Broadcast(HomingProjectileAmmoCurrent);
                }
        }
        else
        {
                // If the Tank hasn't any homing projectile class assigned, discard
every found target and exit the function
                for (int32 index = 0; index != HomingTarget.Num(); index++)
                {
                        DrawTargetOutline(HomingTarget[index], false);
                }
        HomingTarget.Empty();

        UE_LOG
        (
            LogTemp,
            Error,
            TEXT("'HomingProjectileClass' component on Actor %s expects 
it to have a Projectile type set but there isn't any"),
            *GetOwner()->GetName()
        );
        return;
    }

    UE_LOG(LogTemp, Warning, TEXT("Targets: %d"), HomingTarget.Num());

}
////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////

////////////////////////               Draws an outline to every found target 
mesh          ////////////////////////
void APawnTank::DrawTargetOutline(AActor* Target, bool bShouldDraw)
{
    // Get the target's mesh that will be outlined (in this case, it's 
specifically the Enemy Turret's "head")
    UStaticMeshComponent* HomingTargetMesh = Cast<UStaticMeshComponent>
        (

Target->GetRootComponent()->GetChildComponent(0)->GetChildComponent(0)
        );

    if (HomingTargetMesh == nullptr)
    {
        UE_LOG(LogTemp, Error, TEXT("No Target Mesh found to outline!"));
        return;
    }

    // Enable the drawing of the outline in the target's mesh
    HomingTargetMesh->SetRenderCustomDepth(bShouldDraw);
}
////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////

//////       Activates the firing of the Tank's regular projectiles using the 
"PawnBase" parent class virtual method        //////
void APawnTank::Fire()
{
    if (ProjectileAmmoCurrent > 0)
    {
        // If the Tank has regular projectiles ammo, call "PawnBase" class 
Fire() to handle their shooting
```

```
            Super::Fire();

            // Update the current regular projectile ammo count after every
shot and notify subscribed classes about that change
            ProjectileAmmoCurrent = ProcessNewAmmo(ProjectileAmmoCurrent, -1,
ProjectileAmmoMax);
            OnProjectileCountChanged.Broadcast(ProjectileAmmoCurrent);
      }
      else
      {
            // If the Tank hasn't any regular projectile ammo, exit the
function
            UE_LOG(LogTemp, Error, TEXT("No Ammo for Projectile!"));
            return;
      }
}
/////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////

//////      Calculates the current ammo of projectiles (homing or regular)
depending on whether the Tank is shooting or getting ammo pick ups
//////
int APawnTank::ProcessNewAmmo(int CurrentAmmo, int AddedAmount, int MaxAmmo)
{
      CurrentAmmo += AddedAmount;
      // Clamp the ammo calculation so we never get an ammo count less than 0
or greater than the established maximum
      CurrentAmmo = FMath::Clamp(CurrentAmmo, 0, MaxAmmo);
      return CurrentAmmo;
}
/////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
```
PawnTank.cpp