PawnTurret.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 "PawnTurret.generated.h"

 /*

       Crazy Tank classes

 */

class APawnTank;
class APickUpBase;

//////////////////////////////////////////////////////////////////////////////////
//
// This class handles the Enemy Turret's behavior (attacking and destruction)
//
//////////////////////////////////////////////////////////////////////////////////
UCLASS()
class CRAZYTANK_API APawnTurret : public APawnBase
{
      GENERATED_BODY()

private:

      /*
            VARIABLES
      */

      UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Combat", meta =
(AllowPrivateAccess = "true"))
      float FireRange = 500.0f; // Turret's "view" range for checking for the
player and start attacking

      UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Combat", meta =
(AllowPrivateAccess = "true"))
      float FireRate = 2.0f; // If the player is in range, the Turret will fire
every FireRate seconds
```

```
	UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "Pick-Up Type",
meta = (AllowPrivateAccess = "true"))
	TArray< TSubclassOf<APickUpBase> > PickUpClass; // The kind of Pick Up/s
that the Turret will drop when destroyed

	// Timers allow us to trigger events based on elapsed time in the form of
creating asynchronous
	// callbacks to specific function pointers.
	// This Timer is for firing every X amount of seconds based on this fire
rate
	// FTimerHandles: unique handle that can be used to distinguish timers
that have identical delegates
	// FTimerHandle allows us to bind and unbind our timer (control when to
start or stop them)
	FTimerHandle FireRateTimerHandle;

	APawnTank* PlayerPawn = nullptr; // Reference to the Player's Tank

	/*
		METHODS
	*/

	void CheckFireCondition(); // Checking that desired conditions have been
met to allow the firing functionality to be called on the parent class

	float ReturnDistanceToPlayer(); // Calculate the distance to the player's
Tank to see if it's in firing range

public:

	/*
		METHODS
	*/

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

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

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

protected:

	/*
		METHODS
	*/

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

```
};
```

PawnTurret.h

PawnTurret.cpp

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

#include "PawnTurret.h"
#include "Kismet/GameplayStatics.h"
#include "CrazyTank/Actors/PickUpBase.h"
#include "PawnTank.h"


 ////////          Sets default values for this pawn's properties      ////////
APawnTurret::APawnTurret()
{

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

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

      // Cast<DestinyType>(ProvidedType) allows us to convert a provided type
to another using the built-in reflection system of UE
      PlayerPawn = Cast<APawnTank>(UGameplayStatics::GetPlayerPawn(this, 0));

      /* Ensure the timer is created and bound to our CheckFireCondition() as
soon as the game begins.
         GetTimerManager() is a kind of global timer manager for the game, so
you can have multiple timers and this kinds of
         handles them in the background.
         SetTimer() is telling the TimerManager to create a new timer to track,
and we'll be using our timer handle
         to bind and control this during gameplay. */
      // This means that whenever the fire condition is met, the Turret will
fire every X amount of seconds based on its fire rate
      GetWorld()->GetTimerManager().SetTimer(FireRateTimerHandle, this,
&APawnTurret::CheckFireCondition, FireRate, true);
}
//////////////////////////////////////////////////////////////////////////////

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

```
	if (!PlayerPawn || ReturnDistanceToPlayer() > FireRange)
	{
		// If there isn't any player Tank or he's out of the Turret's
firing range, exit this function
		return;
	}
	else
	{
		// Call RotateTurret() from parent class "PawnBase.h" to get a
"look-at" rotation to the Player's Tank when it's in range
		RotateTurret(PlayerPawn->GetActorLocation());
	}
}
//////////////////////////////////////////////////////////////////////

////// Checking that the desired conditions have been met to allow the firing
functionality to be called on the parent class	//////
void APawnTurret::CheckFireCondition()
{
	if(!PlayerPawn || !PlayerPawn->GetIsPlayerAlive())
	{
		// If there isn't any player Tank or it's dead, stop the tick and
exit the function
		SetActorTickEnabled(false);
		return;
	}

	if(ReturnDistanceToPlayer() <= FireRange)
	{
		// If the player's Tank is in range,
		// call the firing logic from parent class "PawnBase"
		Fire();
	}
}
//////////////////////////////////////////////////////////////////////

////////		Calculate the distance to the player's Tank to see if it's
in firing range		////////
float APawnTurret::ReturnDistanceToPlayer()
{
	if (!PlayerPawn)
	{
		// If there isn't any player Tank, return 0
		return 0.0f;
	}

	//Dist() Returns the distance between two FVectors as a float value
	return FVector::Dist(PlayerPawn->GetActorLocation(), GetActorLocation());
}
//////////////////////////////////////////////////////////////////////////////
```

```
//////

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

	/*

		Overriding logic in this child class

	*/

	// Get a random number for enabling the spawning of Pick Ups when this
Turret is going to be destroyed
	int SpawnPickUp = FMath::RandRange(0, 10);
	if (SpawnPickUp >= 5)
	{
		if (PickUpClass.Num() != 0)
		{
			// If the random number is greater than some value and the
Turret has any Pick Up class assigned
			// Spawn a random Pick Up at the same location of this
Turret before it gets destroyed
			int32 RandomIndex = FMath::RandRange(0, PickUpClass.Num() -
1);
			FVector SpawnLocation =
RootComponent->GetComponentLocation();
			APickUpBase* TempPickUp =
GetWorld()->SpawnActor<APickUpBase>(PickUpClass[RandomIndex], SpawnLocation,
FRotator::ZeroRotator);
		}
		else
		{
			UE_LOG
			(
				LogTemp,
				Error,
				TEXT("'PickUpClass' component on Actor %s expects it
to have a PickUp type set but there isn't any"),
				*GetOwner()->GetName()
			);
			return;
		}
	}

	Destroy();
}
//////////////////////////////////////////////////////////////////////////////
```

```
//////
```

PawnTurret.cpp