```cpp
// Fill out your copyright notice in the Description page of Project Settings.


#include "enemy1.h"
#include "Components/SphereComponent.h"
#include "AIController.h"
#include "Maincharacter.h"
#include "Components/BoxComponent.h"
#include "Components/SkeletalMeshComponent.h"
#include"Weapon1.h"
#include"Kismet/GameplayStatics.h"
#include "Engine/SkeletalMeshSocket.h"
#include"Sound/SoundCue.h"
#include"Animation/AnimInstance.h"
#include "Components/CapsuleComponent.h"
#include "MainPlayerController.h"
#include"TimerManager.h"
#include"Components/CapsuleComponent.h"
// Sets default values
Aenemy1::Aenemy1()
{
    // Set this character to call Tick() every frame.  You can turn this off to improve performance if you don't need it.
    PrimaryActorTick.bCanEverTick = true;
    EnemyMovementStatus = EEnemyMovementStatus::EMS_Idle;
    Agrosphere = CreateDefaultSubobject<USphereComponent>(TEXT("Agrosphere"));
    Agrosphere->SetupAttachment(GetRootComponent());
    Agrosphere->InitSphereRadius(600.f);

    Combatsphere = CreateDefaultSubobject<USphereComponent>(TEXT("CombatSphere"));
    Combatsphere->SetupAttachment(GetRootComponent());
    Combatsphere->InitSphereRadius(75.f);


    CombatCollision = CreateDefaultSubobject<UBoxComponent>(TEXT("CombatCollision"));
    CombatCollision->SetupAttachment(GetMesh(), FName("enemysocket"));
    bOverlappingCombatSphere = false;
    Health = 75.f;
    MaxHealth = 100.f;
    Damage = 10.f;

    DeathDelay = 3;


    AttackMinTime = 0.5f;
    AttackMaxTime = 3.5f;

}

```

```cpp
// Called when the game starts or when spawned
void Aenemy1::BeginPlay()
{
    Super::BeginPlay();
    Aicontroller = Cast<AAIController>(GetController());
    Agrosphere->OnComponentBeginOverlap.AddDynamic(this, &Aenemy1::AgroSphereOnOverLapbegin);
    Agrosphere->OnComponentEndOverlap.AddDynamic(this, &Aenemy1::AgroSphereOnOverLapend);
    Combatsphere->OnComponentBeginOverlap.AddDynamic(this, &Aenemy1::CombatSphereOnOverLapbegin);
    Combatsphere->OnComponentEndOverlap.AddDynamic(this, &Aenemy1::CombatSphereOnOverLapend);

    CombatCollision->OnComponentBeginOverlap.AddDynamic(this, &Aenemy1::combatOnOverLapbegin);
    CombatCollision->OnComponentEndOverlap.AddDynamic(this, &Aenemy1::combatOnOverLapend);
    CombatCollision->SetCollisionEnabled(ECollisionEnabled::NoCollision);
    CombatCollision->SetCollisionObjectType(ECollisionChannel::ECC_WorldDynamic);
    CombatCollision->SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Ignore);
    CombatCollision->SetCollisionResponseToChannel(ECollisionChannel::ECC_Pawn, ECollisionResponse::ECR_Overlap);



}

// Called every frame
void Aenemy1::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);

}

// Called to bind functionality to input
void Aenemy1::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
{
    Super::SetupPlayerInputComponent(PlayerInputComponent);

}

void Aenemy1::AgroSphereOnOverLapbegin(UPrimitiveComponent* OverlappedComponent, AActor* OtherActor, UPrimitiveComponent* OtherComp, int32 otherbodyindex, bool bFromSweep, const FHitResult& SweepResult)
{
    if (OtherActor && alive())
    {
        AMaincharacter* Main = Cast<AMaincharacter>(OtherActor);
        if (Main)
        {
            MoveToTarget(Main);
        }
    }
 }

void Aenemy1::AgroSphereOnOverLapend(UPrimitiveComponent* OverlappedComponent, AActor* OtherActor, UPrimitiveComponent* OtherComp, int32 otherbodyindex)
{
    if (OtherActor)
    {
        AMaincharacter* Main = Cast<AMaincharacter>(OtherActor);
        if (Main)
        {
            bhasvalidtarget = false;
            Main->SetCombatTarget(nullptr);
            Main->SetCombatTarget(false);
            Main->UpdateCombatTarget();
            SetEnemyMovementStatus(EEnemyMovementStatus::EMS_Idle);
            if (Aicontroller)
            {
                Aicontroller->StopMovement();
            }


        }
    }
 }


void Aenemy1::CombatSphereOnOverLapbegin(UPrimitiveComponent* OverlappedComponent, AActor* OtherActor, UPrimitiveComponent* OtherComp, int32 otherbodyindex, bool bFromSweep, const FHitResult& SweepResult)
{

    if (OtherActor && alive() )
    {
        AMaincharacter* Main = Cast<AMaincharacter>(OtherActor);
        if (Main )
        {
            bhasvalidtarget = true;

            Main->SetCombatTarget(this);
            Main ->SetHasCombatTarget(true);
            Main->UpdateCombatTarget();
            CombatTarget = Main;
            bOverlappingCombatSphere = true;

            attack();

        }

    }

}

void Aenemy1::CombatSphereOnOverLapend(UPrimitiveComponent* OverlappedComponent, AActor* OtherActor, UPrimitiveComponent* OtherComp, int32 otherbodyindex)
{

    if (OtherActor && OtherComp)
    {
        AMaincharacter* Main = Cast<AMaincharacter>(OtherActor);
        if (Main)
        {


            bOverlappingCombatSphere = false;
            MoveToTarget(Main);
            CombatTarget = nullptr;
            if (Main->CombatTarget == this)
            {
                Main->SetCombatTarget(nullptr);
                Main->bHasCombatTarget = false;
                Main->UpdateCombatTarget();
            }
            if (Main->MainPlayerController)
            {
                USkeletalMeshComponent* MainMesh = Cast<USkeletalMeshComponent>(OtherComp);

            }
            GetWorldTimerManager().ClearTimer(AttackTimer);
        }

    }

}
void Aenemy1::MoveToTarget(class AMaincharacter* Target)
{
    SetEnemyMovementStatus(EEnemyMovementStatus::EMS_MoveToTarget);

    if (Aicontroller)
    {
        FAIMoveRequest Moverequest;
        Moverequest.SetGoalActor(Target);
        Moverequest.SetAcceptanceRadius(5.0f);

        FNavPathSharedPtr Navpath;
        Aicontroller->MoveTo(Moverequest, &Navpath);

        bhasvalidtarget = false;

    }
}
void Aenemy1::combatOnOverLapbegin(UPrimitiveComponent* OverlappedComponent, AActor* OtherActor, UPrimitiveComponent* OtherComp, int32 otherbodyindex, bool bFromSweep, const FHitResult& SweepResult)
{
    if (OtherActor)
    {
        AMaincharacter* Main = Cast<AMaincharacter>(OtherActor);
        if (Main && Main->MovementStatus!=EMovementStatus::EMS_Dead)
        {
            if (Main->HitParticles)
            {
                const USkeletalMeshSocket* tipsocket = GetMesh()->GetSocketByName("tipsocket");
                if (tipsocket)
                {
                    FVector SocketLocation = tipsocket->GetSocketLocation(GetMesh());

                    UGameplayStatics::SpawnEmitterAtLocation(GetWorld(), Main->HitParticles, SocketLocation, FRotator(0.f), false);
                }

            }
            if (Main->HitSound)
            {
                UGameplayStatics::PlaySound2D(this, Main->HitSound);
            }
            if (DamageTypeclass)
            {
                UGameplayStatics::ApplyDamage(Main, Damage, Aicontroller, this, DamageTypeclass);

            }
        }
    }
}

void Aenemy1::combatOnOverLapend(UPrimitiveComponent* OverlappedComponent, AActor* OtherActor, UPrimitiveComponent* OtherComp, int32 otherbodyindex)
{

}
void Aenemy1::activatecollision()
{
    CombatCollision->SetCollisionEnabled(ECollisionEnabled::QueryOnly);

```

```cpp
}

void Aenemy1::deactivatecollision()
{
    CombatCollision->SetCollisionEnabled(ECollisionEnabled::NoCollision);
}

void Aenemy1::attack()
{
    if (alive() && bhasvalidtarget )
    {
        if (Aicontroller)
        {

            SetEnemyMovementStatus(EEnemyMovementStatus::EMS_Attacking);
        }
        if (!bAttacking)
        {
            bAttacking = true;
            UAnimInstance* AnimInstance = GetMesh()->GetAnimInstance();
            if (AnimInstance)
            {
                AnimInstance->Montage_Play(CombatMontage, 1.35f);
                AnimInstance->Montage_JumpToSection(FName("attacknew"), CombatMontage);
            }

        }
    }

}
void Aenemy1::attackend()
{
    bAttacking = false;
    if (bOverlappingCombatSphere )
    {
        float AttackTime = FMath::FRandRange(AttackMinTime, AttackMaxTime);
        GetWorldTimerManager().SetTimer(AttackTimer, this, &Aenemy1::attack, AttackTime);
    }
}
float Aenemy1::TakeDamage(float DamageAmount, struct FDamageEvent const& DamageEvent, class AController* EventInstigator, AActor* DamageCauser)
{
    if (Health - DamageAmount <= 0.f)
    {
        Health -= DamageAmount;
        die(DamageCauser);
    }
    else
    {
        Health -= DamageAmount;
    }
    return DamageAmount;
}
void Aenemy1::die(AActor* Causer)

{

    UAnimInstance* AnimInstance = GetMesh()->GetAnimInstance();
    if (AnimInstance)
    {
        AnimInstance->Montage_Play(CombatMontage, 1.35f);
        AnimInstance->Montage_JumpToSection(FName("death"),
          CombatMontage);
    }
    SetEnemyMovementStatus(EEnemyMovementStatus::EMS_Dead);
    CombatCollision->SetCollisionEnabled
      (ECollisionEnabled::NoCollision);
    Agrosphere->SetCollisionEnabled(ECollisionEnabled::NoCollision);
    Combatsphere->SetCollisionEnabled(ECollisionEnabled::NoCollision);
    GetCapsuleComponent()->SetCollisionEnabled
      (ECollisionEnabled::NoCollision);

    AMaincharacter* Main = Cast<AMaincharacter>(Causer);
    if (Main)
    {
        Main->UpdateCombatTarget();
    }
}
void Aenemy1::Deathend()
{
    GetMesh() ->bPauseAnims = true;
    GetMesh()->bNoSkeletonUpdate = true;
    GetWorldTimerManager().SetTimer(DeathTimer, this,
      &Aenemy1::disappear, DeathDelay);
}
bool Aenemy1::alive()
{
    return GetEnemyMovementStatus() != EEnemyMovementStatus::EMS_Dead;
}
void Aenemy1::disappear()
{

    Destroy();
}
```