

```
1 // Fill out your copyright notice in the Description page of Project Settings.
2
3
4 #include "enemy1.h"
5 #include "Components/SphereComponent.h"
6 #include "AIController.h"
7 #include "Maincharacter.h"
8 #include "Components/BoxComponent.h"
9 #include "Components/SkeletalMeshComponent.h"
10 #include "Weapon1.h"
11 #include "Kismet/GameplayStatics.h"
12 #include "Engine/SkeletalMeshSocket.h"
13 #include "Sound/SoundCue.h"
14 #include "Animation/AnimInstance.h"
15 #include "Components/CapsuleComponent.h"
16 #include "MainPlayerController.h"
17 #include "TimerManager.h"
18 #include "Components/CapsuleComponent.h"
19 // Sets default values
20 Aenemy1::Aenemy1()
21 {
22     // Set this character to call Tick() every frame. You can turn
23     // this off to improve performance if you don't need it.
24     PrimaryActorTick.bCanEverTick = true;
25     EnemyMovementStatus = EEnemyMovementStatus::EMS_Idle;
26     Agrosphere = CreateDefaultSubobject<USphereComponent>(TEXT
27     ("Agrosphere"));
28     Agrosphere->SetupAttachment(GetRootComponent());
29     Agrosphere->InitSphereRadius(600.f);
30
31     Combatsphere = CreateDefaultSubobject<USphereComponent>(TEXT
32     ("CombatSphere"));
33     Combatsphere->SetupAttachment(GetRootComponent());
34     Combatsphere->InitSphereRadius(75.f);
35
36     CombatCollision = CreateDefaultSubobject<UBoxComponent>(TEXT
37     ("CombatCollision"));
38     CombatCollision->SetupAttachment(GetMesh(), FName("enemysocket"));
39     bOverlappingCombatSphere = false;
40     Health = 75.f;
41     MaxHealth = 100.f;
42     Damage = 10.f;
43
44     DeathDelay = 3;
45
46     AttackMinTime = 0.5f;
47     AttackMaxTime = 3.5f;
48 }
```

```
49 // Called when the game starts or when spawned
50 void Aenemy1::BeginPlay()
51 {
52     Super::BeginPlay();
53     Aicontroller = Cast<AAIController>(GetController());
54     Agrosphere->OnComponentBeginOverlap.AddDynamic(this,           ↗
        &Aenemy1::AgroSphereOnOverlapbegin);
55     Agrosphere->OnComponentEndOverlap.AddDynamic(this,           ↗
        &Aenemy1::AgroSphereOnOverlapend);
56     Combatsphere->OnComponentBeginOverlap.AddDynamic(this,       ↗
        &Aenemy1::CombatSphereOnOverlapbegin);
57     Combatsphere->OnComponentEndOverlap.AddDynamic(this,         ↗
        &Aenemy1::CombatSphereOnOverlapend);
58
59     CombatCollision->OnComponentBeginOverlap.AddDynamic(this,    ↗
        &Aenemy1::combatOnOverlapbegin);
60     CombatCollision->OnComponentEndOverlap.AddDynamic(this,      ↗
        &Aenemy1::combatOnOverlapend);
61     CombatCollision->SetCollisionEnabled
        (ECollisionEnabled::NoCollision);
62     CombatCollision->SetCollisionObjectType
        (ECollisionChannel::ECC_WorldDynamic);
63     CombatCollision->SetCollisionResponseToAllChannels
        (ECollisionResponse::ECR_Ignore);
64     CombatCollision->SetCollisionResponseToChannel
        (ECollisionChannel::ECC_Pawn, ECollisionResponse::ECR_Overlap);
65
66
67
68 }
69
70 // Called every frame
71 void Aenemy1::Tick(float DeltaTime)
72 {
73     Super::Tick(DeltaTime);
74
75 }
76
77 // Called to bind functionality to input
78 void Aenemy1::SetupPlayerInputComponent(UInputComponent*       ↗
    PlayerInputComponent)
79 {
80     Super::SetupPlayerInputComponent(PlayerInputComponent);
81
82 }
83
84 void Aenemy1::AgroSphereOnOverlapbegin(UPrimitiveComponent*   ↗
    OverlappedComponent, AActor* OtherActor, UPrimitiveComponent* ↗
    OtherComp, int32 otherbodyindex, bool bFromSweep, const FHitResult& ↗
    SweepResult)
85 {
86     if (OtherActor && alive())
87     {
```

```
88     AMaincharacter* Main = Cast<AMaincharacter>(OtherActor);
89     if (Main)
90     {
91         MoveToTarget(Main);
92     }
93 }
94 }
95
96 void Aenemy1::AgroSphereOnOverLapend(UPrimitiveComponent*      ↗
    OverlappedComponent, AActor* OtherActor, UPrimitiveComponent* ↗
    OtherComp, int32 otherbodyindex)
97 {
98     if (OtherActor)
99     {
100         AMaincharacter* Main = Cast<AMaincharacter>(OtherActor);
101         if (Main)
102         {
103             bhasvalidtarget = false;
104             Main->SetCombatTarget(nullptr);
105             Main->SetCombatTarget(false);
106             Main->UpdateCombatTarget();
107             SetEnemyMovementStatus(EEnemyMovementStatus::EMS_Idle);
108             if (Aicontroller)
109             {
110                 Aicontroller->StopMovement();
111             }
112
113
114         }
115     }
116 }
117
118
119 void Aenemy1::CombatSphereOnOverLapbegin(UPrimitiveComponent* ↗
    OverlappedComponent, AActor* OtherActor, UPrimitiveComponent* ↗
    OtherComp, int32 otherbodyindex, bool bFromSweep, const FHitResult& ↗
    SweepResult)
120 {
121
122     if (OtherActor && alive() )
123     {
124         AMaincharacter* Main = Cast<AMaincharacter>(OtherActor);
125         if (Main )
126         {
127             bhasvalidtarget = true;
128
129             Main->SetCombatTarget(this);
130             Main ->SetHasCombatTarget(true);
131             Main->UpdateCombatTarget();
132             CombatTarget = Main;
133             bOverlappingCombatSphere = true;
134
135             attack();
```

```
136
137     }
138
139     }
140
141 }
142
143 void Aenemy1::CombatSphereOnOverLapend(UPrimitiveComponent*           ↗
    OverlappedComponent, AActor* OtherActor, UPrimitiveComponent*       ↗
    OtherComp, int32 otherbodyindex)
144 {
145
146     if (OtherActor && OtherComp)
147     {
148         AMaincharacter* Main = Cast<AMaincharacter>(OtherActor);
149         if (Main)
150         {
151
152
153             bOverlappingCombatSphere = false;
154             MoveToTarget(Main);
155             CombatTarget = nullptr;
156             if (Main->CombatTarget == this)
157             {
158                 Main->SetCombatTarget(nullptr);
159                 Main->bHasCombatTarget = false;
160                 Main->UpdateCombatTarget();
161             }
162             if (Main->MainPlayerController)
163             {
164                 USkeletalMeshComponent* MainMesh =                ↗
                    Cast<USkeletalMeshComponent>(OtherComp);
165
166             }
167             GetWorldTimerManager().ClearTimer(AttackTimer);
168         }
169     }
170 }
171
172 }
173 void Aenemy1::MoveToTarget(class AMaincharacter* Target)
174 {
175     SetEnemyMovementStatus(EEnemyMovementStatus::EMS_MoveToTarget);
176
177     if (Aicontroller)
178     {
179         FAIMoveRequest Moverequest;
180         Moverequest.SetGoalActor(Target);
181         Moverequest.SetAcceptanceRadius(5.0f);
182
183         FNavPathSharedPtr Navpath;
184         Aicontroller->MoveTo(Moverequest, &Navpath);
185     }
```

```
186         bhasvalidtarget = false;
187
188     }
189 }
190 void Aenemy1::combatOnOverLapbegin(UPrimitiveComponent*           ↗
    OverlappedComponent, AActor* OtherActor, UPrimitiveComponent*   ↗
    OtherComp, int32 otherbodyindex, bool bFromSweep, const FHitResult& ↗
    SweepResult)
191 {
192     if (OtherActor)
193     {
194         AMaincharacter* Main = Cast<AMaincharacter>(OtherActor);
195         if (Main && Main->MovementStatus!=EMovementStatus::EMS_Dead)
196         {
197             if (Main->HitParticles)
198             {
199                 const USkeletalMeshSocket* tipsocket = GetMesh()- ↗
                    >GetSocketByName("tipsocket");
200                 if (tipsocket)
201                 {
202                     FVector SocketLocation = tipsocket- ↗
                        >GetSocketLocation(GetMesh());
203
204                     UGameplayStatics::SpawnEmitterAtLocation(GetWorld() ↗
                        (), Main->HitParticles, SocketLocation, FRotator ↗
                        (0.f), false);
205                 }
206
207             }
208             if (Main->HitSound)
209             {
210                 UGameplayStatics::PlaySound2D(this, Main->HitSound);
211             }
212             if (DamageTypeclass)
213             {
214                 UGameplayStatics::ApplyDamage(Main, Damage, ↗
                    Aicontroller, this, DamageTypeclass);
215
216             }
217         }
218     }
219 }
220
221 void Aenemy1::combatOnOverLapend(UPrimitiveComponent*           ↗
    OverlappedComponent, AActor* OtherActor, UPrimitiveComponent*   ↗
    OtherComp, int32 otherbodyindex)
222 {
223
224 }
225 void Aenemy1::activatecollision()
226 {
227     CombatCollision->SetCollisionEnabled(ECollisionEnabled::QueryOnly);
228 }
```

```
229 }
230
231 void Aenemy1::deactivatecollision()
232 {
233     CombatCollision->SetCollisionEnabled
234         (ECollisionEnabled::NoCollision);
235 }
236
237 void Aenemy1::attack()
238 {
239     if (alive() && bhasvalidtarget )
240     {
241         if (Aicontroller)
242         {
243             SetEnemyMovementStatus
244                 (EEEnemyMovementStatus::EMS_Attacking);
245         }
246         if (!bAttacking)
247         {
248             bAttacking = true;
249             UAnimInstance* AnimInstance = GetMesh()->GetAnimInstance();
250             if (AnimInstance)
251             {
252                 AnimInstance->Montage_Play(CombatMontage, 1.35f);
253                 AnimInstance->Montage_JumpToSection(FName("attacknew"),
254                     CombatMontage);
255             }
256         }
257     }
258 }
259
260 void Aenemy1::attackend()
261 {
262     bAttacking = false;
263     if (bOverlappingCombatSphere )
264     {
265         float AttackTime = FMath::FRandRange(AttackMinTime,
266             AttackMaxTime);
267         GetWorldTimerManager().SetTimer(AttackTimer, this,
268             &Aenemy1::attack, AttackTime);
269     }
270 }
271
272 float Aenemy1::TakeDamage(float DamageAmount, struct FDamageEvent
273     const& DamageEvent, class AController* EventInstigator, AActor*
274     DamageCauser)
275 {
276     if (Health - DamageAmount <= 0.f)
277     {
278         Health -= DamageAmount;
279         die(DamageCauser);
280     }
281 }
```

```
275     else
276     {
277         Health -= DamageAmount;
278     }
279     return DamageAmount;
280 }
281 void Aenemy1::die(AActor* Causer)
282 {
283 {
284     UAnimInstance* AnimInstance = GetMesh()->GetAnimInstance();
285     if (AnimInstance)
286     {
287         AnimInstance->Montage_Play(CombatMontage, 1.35f);
288         AnimInstance->Montage_JumpToSection(FName("death"),
289             CombatMontage);
290     }
291     SetEnemyMovementStatus(EEnemyMovementStatus::EMS_Dead);
292     CombatCollision->SetCollisionEnabled
293         (ECollisionEnabled::NoCollision);
294     Agrosphere->SetCollisionEnabled(ECollisionEnabled::NoCollision);
295     Combatsphere->SetCollisionEnabled(ECollisionEnabled::NoCollision);
296     GetCapsuleComponent()->SetCollisionEnabled
297         (ECollisionEnabled::NoCollision);
298     AMaincharacter* Main = Cast<AMaincharacter>(Causer);
299     if (Main)
300     {
301         Main->UpdateCombatTarget();
302     }
303 }
304 void Aenemy1::Deathend()
305 {
306     GetMesh() ->bPauseAnims = true;
307     GetMesh()->bNoSkeletonUpdate = true;
308     GetWorldTimerManager().SetTimer(DeathTimer, this,
309         &Aenemy1::disappear, DeathDelay);
310 }
311 bool Aenemy1::alive()
312 {
313     return GetEnemyMovementStatus() != EEnemyMovementStatus::EMS_Dead;
314 }
315 void Aenemy1::disappear()
316 {
317     Destroy();
318 }
```