AbilityInfo

#pragma once

#include "CoreMinimal.h"

#include "GameFramework/Actor.h"

#include "AbilityInfo.generated.h"

class UGameplayAbilityBase;

UENUM(Blueprintable, BlueprintType)

enum class ECostType :uint8 {

Health,

Mana,

Strength

};

USTRUCT(Blueprintable,BlueprintType)

struct FGameplayAbilityInfo {

GENERATED\_BODY();

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "AbilityInfo")

float Duration;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "AbilityInfo")

float Cost;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "AbilityInfo")

class UMaterialInstance\* mats;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "AbilityInfo")

ECostType type;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "AbilityInfo")

class TSubclassOf<UGameplayAbilityBase> abilityClass;

FGameplayAbilityInfo();

FGameplayAbilityInfo(float duration, float cost, UMaterialInstance\* m, ECostType t, TSubclassOf<UGameplayAbilityBase> abClass);

};

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

#include "AbilityInfo.h"

FGameplayAbilityInfo::FGameplayAbilityInfo() {

Duration = 0;

Cost = 0;

mats = nullptr;

type = ECostType::Health;

abilityClass = nullptr;

}

FGameplayAbilityInfo::FGameplayAbilityInfo(float duration, float cost, UMaterialInstance\* m, ECostType t, TSubclassOf<UGameplayAbilityBase> abClass) {

Duration = duration;

Cost = cost;

mats = m;

type = t;

abilityClass = abClass;

}

AIControllerBase

#pragma once

#include "CoreMinimal.h"

#include "AIController.h"

#include "BehaviorTree/BehaviorTreeComponent.h"

#include "BehaviorTree/BlackboardComponent.h"

#include "AIControllerBase.generated.h"

/\*\*

\*

\*/

UCLASS()

class SHINBIFORC\_API AAIControllerBase : public AAIController

{

GENERATED\_BODY()

public:

AAIControllerBase();

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "AIControllerProperties")

class UBehaviorTreeComponent\* btComp;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "AIControllerProperties")

class UBlackboardComponent\* bbComp;

UFUNCTION(BlueprintCallable,Category = "AIControllerProperties")

void OnPossess(APawn\* inPawn) override;

UFUNCTION(BlueprintCallable, Category = "AIControllerProperties")

void OnUnPossess() override;

};

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

#include "AIControllerBase.h"

#include "CharacterBase.h"

AAIControllerBase::AAIControllerBase() {

PrimaryActorTick.bCanEverTick = true;

btComp = CreateDefaultSubobject<UBehaviorTreeComponent>("btComp");

bbComp = CreateDefaultSubobject<UBlackboardComponent>("bbComp");

}

void AAIControllerBase::OnPossess(APawn\* inPawn)

{

Super::OnPossess(inPawn);

ACharacterBase\* enemy = Cast<ACharacterBase>(inPawn);

if (enemy) {

bbComp->InitializeBlackboard(\*enemy->tree->GetBlackboardAsset());

btComp->StartTree(\*enemy->tree);

}

}

void AAIControllerBase::OnUnPossess()

{

Super::OnUnPossess();

btComp->StopTree(EBTStopMode::Safe);

}

AttributeSetBase

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

#pragma once

#include "CoreMinimal.h"

#include "AttributeSet.h"

#include "GameplayEffect.h"

#include "GameplayEffectExtension.h"

#include "AttributeSetBase.generated.h"

/\*\*

\*

\*/

DECLARE\_DYNAMIC\_MULTICAST\_DELEGATE\_TwoParams(FOnPropertyDelegate,float, min,float,max);

UCLASS()

class SHINBIFORC\_API UAttributeSetBase : public UAttributeSet

{

GENERATED\_BODY()

public:

UAttributeSetBase();

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "AttributeSetBaseProperties")

FGameplayAttributeData Health;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "AttributeSetBaseProperties")

FGameplayAttributeData MaxHealth;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "AttributeSetBaseProperties")

FGameplayAttributeData Mana;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "AttributeSetBaseProperties")

FGameplayAttributeData MaxMana;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "AttributeSetBaseProperties")

FGameplayAttributeData Strength;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "AttributeSetBaseProperties")

FGameplayAttributeData MaxStrength;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "AttributeSetBaseProperties")

FGameplayAttributeData DamageAttr;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "AttributeSetBaseProperties")

FGameplayAttributeData ArmorAtrr;

void PostGameplayEffectExecute(const struct FGameplayEffectModCallbackData& Data)override;

FOnPropertyDelegate healthDelegate;

FOnPropertyDelegate manaDelegate;

FOnPropertyDelegate strengthDelegate;

};

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

#include "AttributeSetBase.h"

#include "CharacterBase.h"

UAttributeSetBase::UAttributeSetBase():Health(100),MaxHealth(100),Mana(100),MaxMana(100),Strength(100),MaxStrength(100), ArmorAtrr(100),DamageAttr(100){

}

void UAttributeSetBase::PostGameplayEffectExecute(const FGameplayEffectModCallbackData& Data)

{

if (Data.EvaluatedData.Attribute.GetUProperty() == FindFieldChecked<FProperty>(UAttributeSetBase::StaticClass(),GET\_MEMBER\_NAME\_CHECKED(UAttributeSetBase,Health))) {

Health.SetCurrentValue(FMath::Clamp<float>(Health.GetCurrentValue(),0, MaxHealth.GetBaseValue()));

healthDelegate.Broadcast(Health.GetCurrentValue(), MaxHealth.GetBaseValue());

ACharacterBase\* shinbi = Cast<ACharacterBase>(GetOwningActor());

if (Health.GetCurrentValue() == MaxHealth.GetBaseValue()) {

if (shinbi) {

shinbi->AddGameplayTag(shinbi->fullHealthTag);

}

else {

if (shinbi) {

shinbi->RemoveGameplayTag(shinbi->fullHealthTag);

}

}

}

}

if (Data.EvaluatedData.Attribute.GetUProperty() == FindFieldChecked<FProperty>(UAttributeSetBase::StaticClass(),GET\_MEMBER\_NAME\_CHECKED(UAttributeSetBase, Mana))) {

Mana.SetCurrentValue(FMath::Clamp<float>(Mana.GetCurrentValue(),0,MaxMana.GetBaseValue()));

manaDelegate.Broadcast(Mana.GetCurrentValue(),MaxMana.GetBaseValue());

}

if (Data.EvaluatedData.Attribute.GetUProperty() == FindFieldChecked<FProperty>(UAttributeSetBase::StaticClass(),GET\_MEMBER\_NAME\_CHECKED(UAttributeSetBase,Strength))) {

Strength.SetCurrentValue(FMath::Clamp<float>(Strength.GetCurrentValue(),0,MaxStrength.GetBaseValue()));

strengthDelegate.Broadcast(Strength.GetCurrentValue(),MaxStrength.GetBaseValue());

}

}

BTServiceBase

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

#pragma once

#include "CoreMinimal.h"

#include "BehaviorTree/BTService.h"

#include "BehaviorTree/BehaviorTreeComponent.h"

#include "BTServiceBase.generated.h"

/\*\*

\*

\*/

UCLASS()

class SHINBIFORC\_API UBTServiceBase : public UBTService

{

GENERATED\_BODY()

public:

UBTServiceBase();

virtual void TickNode(UBehaviorTreeComponent& ownerComp,uint8\* nodememory,float deltatime);

};

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

#include "BTServiceBase.h"

#include "AIControllerBase.h"

#include "CharacterBase.h"

UBTServiceBase::UBTServiceBase() {

bCreateNodeInstance = true;

}

void UBTServiceBase::TickNode(UBehaviorTreeComponent& ownerComp, uint8\* nodememory, float deltatime) {

AAIControllerBase\* enemyController = Cast<AAIControllerBase>(ownerComp.GetAIOwner());

if (enemyController) {

ACharacterBase\* shinbi = Cast<ACharacterBase>(GetWorld()->GetFirstPlayerController()->GetPawn());

enemyController->bbComp->SetValueAsObject("Shinbi",shinbi);

}

}

BTTask\_Attack

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

#pragma once

#include "AIControllerBase.h"

#include "CharacterBase.h"

#include "CoreMinimal.h"

#include "BehaviorTree/BehaviorTreeComponent.h"

#include "BehaviorTree/Tasks/BTTask\_BlackboardBase.h"

#include "BTTask\_Attack.generated.h"

/\*\*

\*

\*/

UCLASS()

class SHINBIFORC\_API UBTTask\_Attack : public UBTTask\_BlackboardBase

{

GENERATED\_BODY()

public:

UBTTask\_Attack();

virtual EBTNodeResult::Type ExecuteTask(UBehaviorTreeComponent& ownerComp, uint8\* nodememory) override;

};

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

#include "BTTask\_Attack.h"

UBTTask\_Attack::UBTTask\_Attack() {

bCreateNodeInstance = true;

}

EBTNodeResult::Type UBTTask\_Attack::ExecuteTask(UBehaviorTreeComponent& ownerComp, uint8\* nodememory)

{

AAIControllerBase\* enemyController = Cast<AAIControllerBase>(ownerComp.GetAIOwner());

ACharacterBase\* enemy = Cast<ACharacterBase>(enemyController->GetPawn());

UAnimMontage\* montage = Cast<UAnimMontage>(LoadObject<UAnimMontage>(NULL,TEXT("AnimMontage'/Game/Blueprint/AM\_Melee\_Enemy.AM\_Melee\_Enemy'")));

if (enemy && montage) {

enemy->PlayAnimMontage(montage);

/\*GEngine->AddOnScreenDebugMessage(-1, 10, FColor::Blue, TEXT("shinbi Found"));\*/

return EBTNodeResult::Succeeded;

}

else {

return EBTNodeResult::Failed;

}

return EBTNodeResult::Failed;

}

BTTask\_MoveToShinbi

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

#pragma once

#include "CoreMinimal.h"

#include "BehaviorTree/Tasks/BTTask\_BlackboardBase.h"

#include "BTTask\_MoveToShinbi.generated.h"

/\*\*

\*

\*/

UCLASS()

class SHINBIFORC\_API UBTTask\_MoveToShinbi : public UBTTask\_BlackboardBase

{

GENERATED\_BODY()

public:

UBTTask\_MoveToShinbi();

virtual EBTNodeResult::Type ExecuteTask(UBehaviorTreeComponent& ownerComp,uint8\* nodememory) override;

};

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

#include "BTTask\_MoveToShinbi.h"

#include "AIControllerBase.h"

#include "CharacterBase.h"

UBTTask\_MoveToShinbi::UBTTask\_MoveToShinbi() {

bCreateNodeInstance = true;

}

EBTNodeResult::Type UBTTask\_MoveToShinbi::ExecuteTask(UBehaviorTreeComponent& ownerComp, uint8\* nodememory)

{

AAIControllerBase\* enemyController = Cast<AAIControllerBase>(ownerComp.GetAIOwner());

ACharacterBase\* shinbi = Cast<ACharacterBase>(enemyController->bbComp->GetValueAsObject("Shinbi"));

if (enemyController && shinbi) {

enemyController->MoveToActor(shinbi, 50.0f);

return EBTNodeResult::Succeeded;

}

else {

return EBTNodeResult::Failed;

}

return EBTNodeResult::Failed;

}

ACharacterBase

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

#pragma once

#include "CoreMinimal.h"

#include "GameFramework/CharacterMovementComponent.h"

#include "GameFramework/SpringArmComponent.h"

#include "Camera/CameraComponent.h"

#include "Components/SkeletalMeshComponent.h"

#include "AbilitySystemComponent.h"

#include "AbilitySystemInterface.h"

#include "Abilities/GameplayAbility.h"

#include "BehaviorTree/BehaviorTree.h"

#include "GameplayEffect.h"

#include "AttributeSetBase.h"

#include "GameplayAbilityBase.h"

#include "AIControllerBase.h"

#include "AbilityInfo.h"

#include "GameFramework/Character.h"

#include "CharacterBase.generated.h"

UCLASS()

class SHINBIFORC\_API ACharacterBase : public ACharacter, public IAbilitySystemInterface

{

GENERATED\_BODY()

public:

// Sets default values for this character's properties

ACharacterBase();

protected:

// Called when the game starts or when spawned

virtual void BeginPlay() override;

public:

// Called every frame

virtual void Tick(float DeltaTime) override;

// Called to bind functionality to input

virtual void SetupPlayerInputComponent(class UInputComponent\* PlayerInputComponent) override;

public:

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "ShinbiProperty")

FVector moveToAnimation;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "ShinbiProperty")

class USkeletalMeshComponent\* skeletalMeshComp;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "ShinbiProperty")

class USpringArmComponent\* springArmComp;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "ShinbiProperty")

class UCameraComponent\* cameraComp;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "ShinbiProperty")

class UCharacterMovementComponent\* moveComp;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "ShinbiProperty")

class UAbilitySystemComponent\* abilityComp;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "ShinbiAttributeSet")

class UAttributeSetBase\* attributeComp;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "ShinbiAIForEnemy")

class UBehaviorTree\* tree;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "ShinbiProperty")

FGameplayTag fullHealthTag;

UFUNCTION(BlueprintCallable, Category = "ShinbiMovement")

void lookUp(float value);

UFUNCTION(BlueprintCallable, Category = "ShinbiMovement")

void lookRight(float value);

UFUNCTION(BlueprintCallable, Category = "ShinbiMovement")

void moveForward(float value);

UFUNCTION(BlueprintCallable, Category = "ShinbiMovement")

void moveRight(float value);

UFUNCTION(BlueprintCallable, Category = "ShinbiMovement")

void Jumps();

UFUNCTION(BlueprintCallable, Category = "ShinbiAbilities")

void AquireAbility(TSubclassOf<UGameplayAbility> abilityToAquire);

UFUNCTION(BlueprintCallable, Category = "ShinbiAttributes")

void AquireAbilities(TArray<TSubclassOf<UGameplayAbility>> abilityToAquireArray);

UFUNCTION(BlueprintCallable, Category = "ShinbiAttributes")

void AddToUI(TSubclassOf<UGameplayAbilityBase> abilityBase);

UFUNCTION(BlueprintImplementableEvent, Category = "ShinbiAttributes")

void K2\_AddToUI(FGameplayAbilityInfo abilityInfo);

UFUNCTION(BlueprintCallable, Category = "ShinbiAttributes")

void HealthChange(float health, float maxhealth);

UFUNCTION(BlueprintImplementableEvent, Category = "ShinbiAttributes")

void K2\_HealthChange(float health, float maxhealth);

UFUNCTION(BlueprintCallable, Category = "ShinbiAttributes")

void ManaChange(float mana, float maxmana);

UFUNCTION(BlueprintImplementableEvent, Category = "ShinbiAttributes")

void K2\_ManaChange(float mana, float maxmana);

UFUNCTION(BlueprintCallable, Category = "ShinbiAttributes")

void StrengthChange(float strength, float maxstrength);

UFUNCTION(BlueprintImplementableEvent, Category = "ShinbiAttributes")

void K2\_StrengthChange(float strength, float maxstrength);

UFUNCTION(BlueprintImplementableEvent, Category = "ShinbiAttributes")

void Die();

UFUNCTION(BlueprintCallable, Category = "ShinbiAttributes")

void InitializeTheTeamID();

UFUNCTION(BlueprintCallable, Category = "ShinbiAttributes")

bool isHostilePawn(ACharacterBase\* other);

UFUNCTION(BlueprintCallable, Category = "ShinbiAttributes")

int GetTeamID();

UFUNCTION(BlueprintCallable, Category = "ShinbiAttributes")

void Dead();

UFUNCTION(BlueprintCallable, Category = "ShinbiAttributes")

void AddGameplayTag(FGameplayTag tagtoadd);

UFUNCTION(BlueprintCallable, Category = "ShinbiAttributes")

void RemoveGameplayTag(FGameplayTag tagtoremove);

UFUNCTION(BlueprintCallable, Category = "ShinbiAttributes")

void HitStun(float duration);

UFUNCTION(BlueprintCallable, Category = "ShinbiAttributes")

void ApplyGameplayEffectSpecHanlde(const FGameplayEffectSpecHandle& effectHandle, const FGameplayAbilityTargetDataHandle& abilityHandle);

virtual UAbilitySystemComponent\* GetAbilitySystemComponent() const;

bool isDead;

int TeamID;

FScriptDelegate healthDelegateHandle;

FScriptDelegate manaDelegateHandle;

FScriptDelegate strengthDelegateHandle;

FTimerHandle timer;

void DisableInputControl();

void EnableInputControl();

};

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

#include "CharacterBase.h"

// Sets default values

ACharacterBase::ACharacterBase()

{

// 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;

moveToAnimation = FVector::ZeroVector;

skeletalMeshComp = GetMesh();

springArmComp = CreateDefaultSubobject<USpringArmComponent>("springArmComp");

springArmComp->SetupAttachment(skeletalMeshComp);

springArmComp->TargetArmLength = 150.0f;

springArmComp->bEnableCameraLag = true;

springArmComp->bUsePawnControlRotation = true;

springArmComp->SetRelativeLocation(FVector(0,0,150));

cameraComp = CreateDefaultSubobject<UCameraComponent>("cameraComp");

cameraComp->SetupAttachment(springArmComp);

cameraComp->bUsePawnControlRotation = false;

bUseControllerRotationPitch = false;

bUseControllerRotationRoll = false;

bUseControllerRotationYaw = true;

moveComp = GetCharacterMovement();

moveComp->bOrientRotationToMovement = true;

auto skeletalMeshAsset = ConstructorHelpers::FObjectFinder<USkeletalMesh>(TEXT("SkeletalMesh'/Game/ParagonShinbi/Characters/Heroes/Shinbi/Meshes/Shinbi.Shinbi'"));

if (skeletalMeshAsset.Succeeded()) {

skeletalMeshComp->SetSkeletalMesh(skeletalMeshAsset.Object);

skeletalMeshComp->SetRelativeLocation(FVector(0, 0, -90));

skeletalMeshComp->SetRelativeRotation(FRotator(0, -90, 0));

}

else {

GEngine->AddOnScreenDebugMessage(-1, 5, FColor::Red, TEXT("SkeletalMesh Failed!"));

}

abilityComp = CreateDefaultSubobject<UAbilitySystemComponent>("abilityComp");

attributeComp = CreateDefaultSubobject<UAttributeSetBase>("attributeComp");

healthDelegateHandle.BindUFunction(this,"HealthChange");

attributeComp->healthDelegate.Add(healthDelegateHandle);

manaDelegateHandle.BindUFunction(this,"ManaChange");

attributeComp->manaDelegate.Add(manaDelegateHandle);

strengthDelegateHandle.BindUFunction(this,"StrengthChange");

attributeComp->strengthDelegate.Add(strengthDelegateHandle);

TeamID = 100;

isDead = false;

}

// Called when the game starts or when spawned

void ACharacterBase::BeginPlay()

{

Super::BeginPlay();

AddGameplayTag(fullHealthTag);

}

// Called every frame

void ACharacterBase::Tick(float DeltaTime)

{

Super::Tick(DeltaTime);

}

// Called to bind functionality to input

void ACharacterBase::SetupPlayerInputComponent(UInputComponent\* PlayerInputComponent)

{

Super::SetupPlayerInputComponent(PlayerInputComponent);

PlayerInputComponent->BindAxis("lookUp", this, &ACharacterBase::lookUp);

PlayerInputComponent->BindAxis("lookRight", this, &ACharacterBase::lookRight);

PlayerInputComponent->BindAxis("moveForward", this, &ACharacterBase::moveForward);

PlayerInputComponent->BindAxis("moveRight", this, &ACharacterBase::moveRight);

PlayerInputComponent->BindAction("Jumps", IE\_Pressed, this, &ACharacterBase::Jumps);

}

void ACharacterBase::lookUp(float value)

{

AddControllerPitchInput(value);

}

void ACharacterBase::lookRight(float value)

{

AddControllerYawInput(value);

}

void ACharacterBase::moveForward(float value)

{

FRotator controllerRotation = Controller->GetControlRotation();

FRotator YawRotation = FRotator(0, controllerRotation.Yaw, 0);

FVector forwardVector = FRotationMatrix(YawRotation).GetUnitAxis(EAxis::X);

AddMovementInput(forwardVector \* value);

moveToAnimation.X = value;

}

void ACharacterBase::moveRight(float value)

{

FRotator controllerRotation = Controller->GetControlRotation();

FRotator YawRotation = FRotator(0, controllerRotation.Yaw, 0);

FVector rightVector = FRotationMatrix(YawRotation).GetUnitAxis(EAxis::Y);

AddMovementInput(rightVector \* value);

moveToAnimation.Y = value;

}

void ACharacterBase::Jumps()

{

ACharacter::Jump();

}

void ACharacterBase::AquireAbility(TSubclassOf<UGameplayAbility> abilityToAquire)

{

if (abilityComp) {

if (HasAuthority() && abilityToAquire) {

abilityComp->GiveAbility(FGameplayAbilitySpec(abilityToAquire, 1, 0));

abilityComp->InitAbilityActorInfo(this, this);

}

}

}

void ACharacterBase::AquireAbilities(TArray<TSubclassOf<UGameplayAbility>> abilityToAquireArray)

{

for (TSubclassOf<UGameplayAbility> ability : abilityToAquireArray) {

if (ability) {

AquireAbility(ability);

if (ability->IsChildOf(UGameplayAbilityBase::StaticClass())) {

TSubclassOf<UGameplayAbilityBase> abilitybase = \*ability;

AddToUI(abilitybase);

}

}

}

}

void ACharacterBase::AddToUI(TSubclassOf<UGameplayAbilityBase> abilityBase)

{

if (abilityBase) {

UGameplayAbilityBase\* abilityBasefromTSub = abilityBase->GetDefaultObject<UGameplayAbilityBase>();

FGameplayAbilityInfo abilityInfo = abilityBasefromTSub->GetAbilityInfo();

K2\_AddToUI(abilityInfo);

}

}

void ACharacterBase::HealthChange(float health, float maxhealth)

{

if (health <= 0.0f && isDead == false) {

isDead = true;

Die();

Dead();

}

else {

K2\_HealthChange(health, maxhealth);

}

}

void ACharacterBase::ManaChange(float mana, float maxmana)

{

K2\_ManaChange(mana,maxmana);

}

void ACharacterBase::StrengthChange(float strength, float maxstrength)

{

K2\_StrengthChange(strength, maxstrength);

}

void ACharacterBase::InitializeTheTeamID()

{

AAIControllerBase\* enemyController = Cast<AAIControllerBase>(Controller);

if (enemyController) {

ACharacterBase\* enemy = Cast<ACharacterBase>(enemyController->GetPawn());

enemy->TeamID = 1;

}

}

bool ACharacterBase::isHostilePawn(ACharacterBase\* other)

{

return TeamID != other->GetTeamID();

}

int ACharacterBase::GetTeamID()

{

return TeamID;

}

void ACharacterBase::Dead()

{

APlayerController\* playerController = Cast<APlayerController>(Controller);

if (playerController) {

playerController->DisableInput(playerController);

}

AAIControllerBase\* enemyController = Cast<AAIControllerBase>(Controller);

if (enemyController) {

enemyController->GetBrainComponent()->StopLogic("Safe");

}

}

void ACharacterBase::AddGameplayTag(FGameplayTag tagtoadd)

{

abilityComp->AddLooseGameplayTag(tagtoadd);

abilityComp->SetTagMapCount(tagtoadd,1);

}

void ACharacterBase::RemoveGameplayTag(FGameplayTag tagtoremove) {

abilityComp->RemoveLooseGameplayTag(tagtoremove);

}

void ACharacterBase::HitStun(float duration)

{

DisableInputControl();

GetWorld()->GetTimerManager().SetTimer(timer,this,&ACharacterBase::EnableInputControl,duration,false);

}

void ACharacterBase::ApplyGameplayEffectSpecHanlde(const FGameplayEffectSpecHandle& effectHandle, const FGameplayAbilityTargetDataHandle& abilityHandle)

{

for (TSharedPtr<FGameplayAbilityTargetData> Data : abilityHandle.Data)

{

Data->ApplyGameplayEffectSpec(\*effectHandle.Data.Get());

}

}

UAbilitySystemComponent\* ACharacterBase::GetAbilitySystemComponent() const

{

return abilityComp;

}

void ACharacterBase::DisableInputControl() {

if (!isDead) {

APlayerController\* playerController = Cast<APlayerController>(Controller);

if (playerController) {

playerController->DisableInput(playerController);

}

AAIControllerBase\* enemyController = Cast<AAIControllerBase>(Controller);

if (enemyController) {

enemyController->GetBrainComponent()->StopLogic("Safe");

}

}

}

void ACharacterBase::EnableInputControl() {

if (!isDead) {

APlayerController\* playerController = Cast<APlayerController>(Controller);

if (playerController) {

playerController->EnableInput(playerController);

}

AAIControllerBase\* enemyController = Cast<AAIControllerBase>(Controller);

if (enemyController) {

enemyController->GetBrainComponent()->RestartLogic();

}

}

}

UGameplayAbilityBase

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

#pragma once

#include "CoreMinimal.h"

#include "AbilityInfo.h"

#include "Abilities/GameplayAbility.h"

#include "GameplayAbilityBase.generated.h"

/\*\*

\*

\*/

UCLASS()

class SHINBIFORC\_API UGameplayAbilityBase : public UGameplayAbility

{

GENERATED\_BODY()

public:

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "GameplayAbilityBase")

class UMaterialInstance\* matInstance;

UFUNCTION(BlueprintCallable, Category = "GameplayAbilityBase")

FGameplayAbilityInfo GetAbilityInfo();

};

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

#include "GameplayAbilityBase.h"

FGameplayAbilityInfo UGameplayAbilityBase::GetAbilityInfo()

{

UGameplayEffect\* costEffect = GetCostGameplayEffect();

UGameplayEffect\* cooldownEffect = GetCooldownGameplayEffect();

ECostType costType;

if (costEffect && cooldownEffect) {

float duration = 0;

cooldownEffect->DurationMagnitude.GetStaticMagnitudeIfPossible(1,duration);

float cost = 0;

if (costEffect->Modifiers.Num() > 0) {

costEffect->Modifiers[0].ModifierMagnitude.GetStaticMagnitudeIfPossible(1,cost);

}

FString type = costEffect->Modifiers[0].Attribute.AttributeName;

if (type == "Health") {

costType = ECostType::Health;

}

else if (type == "Mana") {

costType = ECostType::Mana;

}

else {

costType = ECostType::Strength;

}

return FGameplayAbilityInfo(duration,cost, matInstance,costType,GetClass());

}

else {

return FGameplayAbilityInfo();

}

}

AFireBlastTargetActor

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

#pragma once

#include "CoreMinimal.h"

#include "Abilities/GameplayAbility.h"

#include "Abilities/GameplayAbilityTargetActor.h"

#include "FireBlastTargetActor.generated.h"

/\*\*

\*

\*/

UCLASS()

class SHINBIFORC\_API AFireBlastTargetActor : public AGameplayAbilityTargetActor

{

GENERATED\_BODY()

public:

AFireBlastTargetActor();

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "FireBlastTargetActor")

float Radius;

virtual void StartTargeting(UGameplayAbility\* Ability) override;

virtual void ConfirmTargetingAndContinue() override;

};

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

#include "FireBlastTargetActor.h"

AFireBlastTargetActor::AFireBlastTargetActor(){

PrimaryActorTick.bCanEverTick = true;

Radius = 100.0f;

}

void AFireBlastTargetActor::StartTargeting(UGameplayAbility\* Ability)

{

OwningAbility = Ability;

MasterPC=Cast<APlayerController>(OwningAbility->GetOwningActorFromActorInfo()->GetInstigatorController());

}

void AFireBlastTargetActor::ConfirmTargetingAndContinue()

{

APawn\* pawn = MasterPC->GetPawn();

FVector lookAtLocation = pawn->GetActorLocation();

FCollisionQueryParams params;

params.bTraceComplex = true;

if (pawn) {

params.AddIgnoredActor(pawn->GetUniqueID());

}

TArray<FOverlapResult> overlapped;

TArray<TWeakObjectPtr<AActor>> overlappedActor;

bool traceResult = GetWorld()->OverlapMultiByObjectType(overlapped, lookAtLocation,FQuat::Identity,FCollisionObjectQueryParams(ECC\_Pawn),FCollisionShape::MakeSphere(Radius),params);

if (traceResult) {

for (int i = 0; i < overlapped.Num();i++) {

APawn\* enemy = Cast<APawn>(overlapped[i].GetActor());

if (enemy && !overlappedActor.Contains(enemy)) {

overlappedActor.Add(enemy);

}

}

}

if (overlappedActor.Num() > 0) {

FGameplayAbilityTargetDataHandle handleDelegate = StartLocation.MakeTargetDataHandleFromActors(overlappedActor);

TargetDataReadyDelegate.Broadcast(handleDelegate);

}

else {

TargetDataReadyDelegate.Broadcast(FGameplayAbilityTargetDataHandle());

}

//FGameplayAbilityTargetData\_LocationInfo\* locationInfo = new FGameplayAbilityTargetData\_LocationInfo();

//locationInfo->TargetLocation.LiteralTransform = MasterPC->GetPawn()->GetTransform();

//locationInfo->TargetLocation.LocationType = EGameplayAbilityTargetingLocationType::LiteralTransform;

//handleDelegate.Add(locationInfo);

}

AGATargeActorGround

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

#pragma once

#include "CoreMinimal.h"

#include "Abilities/GameplayAbilityTargetActor.h"

#include "Components/DecalComponent.h"

#include "Components/SceneComponent.h"

#include "GATargeActorGround.generated.h"

/\*\*

\*

\*/

UCLASS()

class SHINBIFORC\_API AGATargeActorGround : public AGameplayAbilityTargetActor

{

GENERATED\_BODY()

public:

virtual void StartTargeting(UGameplayAbility\* Ability) override;

virtual void ConfirmTargetingAndContinue() override;

virtual void Tick(float DeltaSeconds) override;

AGATargeActorGround();

UFUNCTION(BlueprintCallable,Category = "GroundBlastTargetData")

FVector GetLookAtLocation();

UPROPERTY(EditAnywhere, BlueprintReadWrite, meta = (ExposeOnSpawn = true),Category = "GroundBlastTargetData")

float Radius;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "GroundBlastTargetData")

class UDecalComponent\* decalComp;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "GroundBlastTargetData")

class USceneComponent\* rootComp;

};

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

#include "GATargeActorGround.h"

#include "Abilities/GameplayAbility.h"

AGATargeActorGround::AGATargeActorGround() {

Radius = 500.0f;

PrimaryActorTick.bCanEverTick = true;

decalComp = CreateDefaultSubobject<UDecalComponent>("decalComp");

rootComp = CreateDefaultSubobject<USceneComponent>("rootComp");

RootComponent = rootComp;

decalComp->SetRelativeRotation(FRotator(0,90,0));

decalComp->SetupAttachment(GetRootComponent());

decalComp->DecalSize = FVector(Radius);

}

void AGATargeActorGround::Tick(float DeltaSeconds)

{

Super::Tick(DeltaSeconds);

decalComp->SetWorldLocation(GetLookAtLocation());

}

void AGATargeActorGround::StartTargeting(UGameplayAbility\* Ability) {

OwningAbility = Ability;

MasterPC = Cast<APlayerController>(OwningAbility->GetOwningActorFromActorInfo()->GetInstigatorController());

}

void AGATargeActorGround::ConfirmTargetingAndContinue() {

FVector LookAtLocation = FVector::ZeroVector;

LookAtLocation = GetLookAtLocation();

FCollisionQueryParams queryParams;

queryParams.bTraceComplex = true;

queryParams.bReturnPhysicalMaterial = false;

APawn\* pawn = MasterPC->GetPawn();

if (pawn) {

queryParams.AddIgnoredActor(pawn->GetUniqueID());

}

TArray<FOverlapResult> overlapResult;

TArray<TWeakObjectPtr<AActor>> overlapActors;

bool overlapTrace = GetWorld()->OverlapMultiByObjectType(overlapResult, LookAtLocation,FQuat::Identity,FCollisionObjectQueryParams(ECC\_Pawn),FCollisionShape::MakeSphere(Radius), queryParams);

if (overlapTrace) {

for (int i = 0; i < overlapResult.Num(); i++) {

APawn\* pawntobeAdd = Cast<APawn>(overlapResult[i].GetActor());

if (pawntobeAdd && !overlapActors.Contains(pawntobeAdd)) {

overlapActors.Add(pawntobeAdd);

}

}

}

FGameplayAbilityTargetData\_LocationInfo\* locationInfo = new FGameplayAbilityTargetData\_LocationInfo();

if (locationInfo) {

locationInfo->TargetLocation.LiteralTransform = decalComp->GetComponentTransform();

locationInfo->TargetLocation.LocationType = EGameplayAbilityTargetingLocationType::LiteralTransform;

}

if (overlapActors.Num() > 0) {

FGameplayAbilityTargetDataHandle targetDataHandle = StartLocation.MakeTargetDataHandleFromActors(overlapActors);

targetDataHandle.Add(locationInfo);

TargetDataReadyDelegate.Broadcast(targetDataHandle);

}

else {

TargetDataReadyDelegate.Broadcast(FGameplayAbilityTargetDataHandle());

}

}

FVector AGATargeActorGround::GetLookAtLocation()

{

FVector ViewLocation;

FRotator ViewRotator;

MasterPC->GetPlayerViewPoint(ViewLocation, ViewRotator);

FHitResult hitResult;

FCollisionQueryParams params;

params.bTraceComplex = true;

APawn\* pawn = MasterPC->GetPawn();

if (pawn != nullptr) {

params.AddIgnoredActor(pawn->GetUniqueID());

}

bool TraceResult = GetWorld()->LineTraceSingleByChannel(hitResult, ViewLocation, ViewLocation + ViewRotator.Vector() \* 1000.0,ECollisionChannel::ECC\_Pawn,params);

//FCollisionObjectQueryParams params1;

//params1.AddObjectTypesToQuery(ECC\_Pawn);

//bool TraceResult = GetWorld()->LineTraceSingleByObjectType(hitResult, ViewLocation, ViewLocation + ViewRotator.Vector() \* 1000.0, params1, params);

FVector LookAtLocation = FVector::ZeroVector;

if (TraceResult) {

LookAtLocation = hitResult.ImpactPoint;

return LookAtLocation;

}

else {

return FVector::ZeroVector;

}

}

UGameplayEffectExecutionCalculation

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

#pragma once

#include "CoreMinimal.h"

#include "AttributeSetBase.h"

#include "GameplayEffectExecutionCalculation.h"

#include "MGEC.generated.h"

/\*\*

\*

\*/

UCLASS()

class SHINBIFORC\_API UMGEC : public UGameplayEffectExecutionCalculation

{

GENERATED\_BODY()

public:

UMGEC();

FProperty\* DamagePro;

FProperty\* HealthPro;

FProperty\* ArmorPro;

FGameplayEffectAttributeCaptureDefinition HealthDef;

FGameplayEffectAttributeCaptureDefinition DamageDef;

FGameplayEffectAttributeCaptureDefinition ArmorDef;

void Execute\_Implementation(const FGameplayEffectCustomExecutionParameters& parameters,FGameplayEffectCustomExecutionOutput& output) const;

};

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

#include "MGEC.h"

UMGEC::UMGEC() {

DamagePro = FindFieldChecked<FProperty>(UAttributeSetBase::StaticClass(), GET\_MEMBER\_NAME\_CHECKED(UAttributeSetBase, DamageAttr));

DamageDef = FGameplayEffectAttributeCaptureDefinition(DamagePro,EGameplayEffectAttributeCaptureSource::Source,true);

HealthPro = FindFieldChecked<FProperty>(UAttributeSetBase::StaticClass(), GET\_MEMBER\_NAME\_CHECKED(UAttributeSetBase,Health));

HealthDef = FGameplayEffectAttributeCaptureDefinition(HealthPro,EGameplayEffectAttributeCaptureSource::Target,true);

ArmorPro = FindFieldChecked<FProperty>(UAttributeSetBase::StaticClass(), GET\_MEMBER\_NAME\_CHECKED(UAttributeSetBase, ArmorAtrr));

ArmorDef = FGameplayEffectAttributeCaptureDefinition(ArmorPro,EGameplayEffectAttributeCaptureSource::Target,true);

RelevantAttributesToCapture.Add(DamageDef);

RelevantAttributesToCapture.Add(HealthDef);

RelevantAttributesToCapture.Add(ArmorDef);

}

void UMGEC::Execute\_Implementation(const FGameplayEffectCustomExecutionParameters& parameters, FGameplayEffectCustomExecutionOutput& output) const {

float damagevalue = 0.0;

float healthvalue = 0.0;

float armorvalue = 0.0;

parameters.AttemptCalculateCapturedAttributeMagnitude(DamageDef,FAggregatorEvaluateParameters(),damagevalue);

parameters.AttemptCalculateCapturedAttributeMagnitude(HealthDef,FAggregatorEvaluateParameters(),healthvalue);

parameters.AttemptCalculateCapturedAttributeMagnitude(ArmorDef,FAggregatorEvaluateParameters(), armorvalue);

float finalDamage = damagevalue - (healthvalue + armorvalue);

finalDamage = FMath::Clamp<float>(finalDamage,0, finalDamage);

if (finalDamage > 0) {

output.AddOutputModifier(FGameplayModifierEvaluatedData(HealthPro, EGameplayModOp::Additive, -finalDamage));

}else{

output.AddOutputModifier(FGameplayModifierEvaluatedData(HealthPro, EGameplayModOp::Additive, finalDamage));

}

}