|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **주차** | 20주차 | **기간** | 11.7 - 11.13 | **지도교수** | (서명) | |
| 이번주 한일 요약 | 유데미 강의 수강 | | | | |

<상세 수행내용>

이번주에는 중간고사와 중간 숙제 밑 실습이 너무 많아서 계획 했던 것의 절반밖에 하지 못했습니다.

모델링은 수행하지 못했고, 그동안 들었던 언리얼 강의를 마무리했습니다.

계획을 조금 변경했는데 4번쨰 캐릭터를 모델링 하기 전에 첫번째 캐릭터의 모델링을 다듬고, 리깅 작업을 하고 간단한 걷기 동작을 만들어 언리얼 에디터에 가져와 간단한 움직임을 만들어보며, 앞으로의 모델링과 언리얼 구현 방향성을 잡아볼 생각입니다.

이번에 강의를 보며 공부한 내용은 AI를 구현할 때 필요한 요소들을 많이 다루었는데 Behavior tree와 그에 상호작용을 하는 c++클래스를 만들어 보았습니다. 세부적인 행동들의 기준을 정해주는 BTService를 만들고, 적AI와 전투를 하며, 그를 판단해주는 게임 모드와, 타이머, UI를 공부하며 따라 만들었습니다.

◎ BTTasks 실행하기

ExecuteTask함수 만들기

ExecuteTask 함수에서는 블랙보드에서 선택된 키의 값을 지우는 동작을 수행한 후

작업이 성공적으로 완료되었음을 알리는 결과를 반환

== BTTask\_ClearBlackboardValue.h

#pragma once

#include "CoreMinimal.h"

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

#include "BTTask\_ClearBlackboardValue.generated.h"

UCLASS()

class SIMPLESHOOTER\_API UBTTask\_ClearBlackboardValue : public UBTTask\_BlackboardBase

{

GENERATED\_BODY()

public:

UBTTask\_ClearBlackboardValue();

protected:

virtual EBTNodeResult::Type ExecuteTask(UBehaviorTreeComponent &OwnerComp, uint8 \*NodeMemory) override;

};

== BTTask\_ClearBlackboardValue.cpp

#include "BTTask\_ClearBlackboardValue.h"

#include "BehaviorTree/BlackboardComponent.h"

UBTTask\_ClearBlackboardValue::UBTTask\_ClearBlackboardValue()

{

NodeName = "Clear Blackboard Value";

}

EBTNodeResult::Type UBTTask\_ClearBlackboardValue::ExecuteTask(UBehaviorTreeComponent &OwnerComp, uint8 \*NodeMemory)

{

Super::ExecuteTask(OwnerComp, NodeMemory);

OwnerComp.GetBlackboardComponent()->ClearValue(GetSelectedBlackboardKey());

return EBTNodeResult::Succeeded;

}

◎ Pawn을 사용하는 BTTask

BT에서 왼쪽의 MoveTo 설정 observe 선택하면 계속 우리가 향하는 방향 경로 업데이트

반경 500으로

C++ 파일 생성 (BTTaskNode)

== BTTask\_Shoot.h

#pragma once

#include "CoreMinimal.h"

#include "BehaviorTree/BTTaskNode.h"

#include "BTTask\_Shoot.generated.h"

UCLASS()

class SIMPLESHOOTER\_API UBTTask\_Shoot : public UBTTaskNode

{

GENERATED\_BODY()

public:

UBTTask\_Shoot();

protected:

virtual EBTNodeResult::Type ExecuteTask(UBehaviorTreeComponent &OwnerComp, uint8 \*NodeMemory) override;

};

=== BTTask\_Shoot.cpp

#include "BTTask\_Shoot.h"

#include "AIController.h"

#include "ShooterCharacter.h"

UBTTask\_Shoot::UBTTask\_Shoot()

{

NodeName = "Shoot";

}

EBTNodeResult::Type UBTTask\_Shoot::ExecuteTask(UBehaviorTreeComponent &OwnerComp, uint8 \*NodeMemory)

{

Super::ExecuteTask(OwnerComp, NodeMemory);

if (OwnerComp.GetAIOwner() == nullptr)

{

return EBTNodeResult::Failed;

}

AShooterCharacter\* Character = Cast<AShooterCharacter>(OwnerComp.GetAIOwner()->GetPawn());

if (Character == nullptr)

{

return EBTNodeResult::Failed;

}

Character->Shoot();

return EBTNodeResult::Succeeded;

}

== MyCharacter.h

public:

void Shoot();

BT에서 shoot 추가

적이 총을 계속 발사하게 됨

루프로 shoot와 wait 담아주기

주기적으로 총 발사

◎ C++에서 BTServices

BT에서 can see player 노드 밑에 우클릭으로 add service

set default focus하면 나를 향해 회전한다.

C++ 클래스로 BTService\_BlackboardBase 만들기

==BTService\_PlayerLocation.h

#pragma once

#include "CoreMinimal.h"

#include "BehaviorTree/Services/BTService\_BlackboardBase.h"

#include "BTService\_PlayerLocation.generated.h"

UCLASS()

class SIMPLESHOOTER\_API UBTService\_PlayerLocation : public UBTService\_BlackboardBase

{

GENERATED\_BODY()

public:

UBTService\_PlayerLocation();

protected:

virtual void TickNode(UBehaviorTreeComponent &OwnerComp, uint8 \*NodeMemory, float DeltaSeconds) override;

};

==BTService\_PlayerLocation.cpp

#include "BTService\_PlayerLocation.h"

#include "BehaviorTree/BlackboardComponent.h"

#include "Kismet/GameplayStatics.h"

#include "GameFramework/Pawn.h"

UBTService\_PlayerLocation::UBTService\_PlayerLocation()

{

NodeName = "Update Player Location";

}

void UBTService\_PlayerLocation::TickNode(UBehaviorTreeComponent &OwnerComp, uint8 \*NodeMemory, float DeltaSeconds)

{

Super::TickNode(OwnerComp, NodeMemory, DeltaSeconds);

APawn \*PlayerPawn = UGameplayStatics::GetPlayerPawn(GetWorld(), 0);

if (PlayerPawn == nullptr)

{

return;

}

OwnerComp.GetBlackboardComponent()->SetValueAsVector(GetSelectedBlackboardKey(), PlayerPawn->GetActorLocation());

}

추가하고 can see 노드에 우클릭 서비스에서 업데이트 플레이어 로케이션추가

하나 더 만들고

== BTService\_PlayerLocationIfSeen.h

#pragma once

#include "CoreMinimal.h"

#include "BehaviorTree/Services/BTService\_BlackboardBase.h"

#include "BTService\_PlayerLocationIfSeen.generated.h"

UCLASS()

class SIMPLESHOOTER\_API UBTService\_PlayerLocationIfSeen : public UBTService\_BlackboardBase

{

GENERATED\_BODY()

public:

UBTService\_PlayerLocationIfSeen();

protected:

virtual void TickNode(UBehaviorTreeComponent &OwnerComp, uint8 \*NodeMemory, float DeltaSeconds) override;

};

== BTService\_PlayerLocationIfSeen.cpp

#include "BTService\_PlayerLocationIfSeen.h"

#include "BehaviorTree/BlackboardComponent.h"

#include "Kismet/GameplayStatics.h"

#include "GameFramework/Pawn.h"

#include "AIController.h"

UBTService\_PlayerLocationIfSeen::UBTService\_PlayerLocationIfSeen()

{

NodeName = "Update Player Location If Seen";

}

void UBTService\_PlayerLocationIfSeen::TickNode(UBehaviorTreeComponent &OwnerComp, uint8 \*NodeMemory, float DeltaSeconds)

{

Super::TickNode(OwnerComp, NodeMemory, DeltaSeconds);

APawn \*PlayerPawn = UGameplayStatics::GetPlayerPawn(GetWorld(), 0);

if (PlayerPawn == nullptr)

{

return;

}

if (OwnerComp.GetAIOwner() == nullptr)

{

return;

}

if (OwnerComp.GetAIOwner()->LineOfSightTo(PlayerPawn))

{

OwnerComp.GetBlackboardComponent()->SetValueAsVector(GetSelectedBlackboardKey(), PlayerPawn->GetActorLocation());

}

else

{

OwnerComp.GetBlackboardComponent()->ClearValue(GetSelectedBlackboardKey());

}

}

◎ 라인트레이스에서 엑터 무시

== Gun.cpp

void AGun::PullTrigger()

{

UGameplayStatics::SpawnEmitterAttached(MuzzleFlash, Mesh, TEXT("MuzzleFlashSocket"));

APawn\* OwnerPawn = Cast<APawn>(GetOwner());

if (OwnerPawn == nullptr) return;

AController\* OwnerController = OwnerPawn->GetController();

if (OwnerController == nullptr) return;

FVector Location;

FRotator Rotation;

OwnerController->GetPlayerViewPoint(Location, Rotation);

FVector End = Location + Rotation.Vector() \* MaxRange;

// TODO: LineTrace

FHitResult Hit;

FCollisionQueryParams Params;

Params.AddIgnoredActor(this);

Params.AddIgnoredActor(GetOwner());

bool bSuccess = GetWorld()->LineTraceSingleByChannel(Hit, Location, End, ECollisionChannel::ECC\_GameTraceChannel1, Params);

if (bSuccess)

{

FVector ShotDirection = -Rotation.Vector();

UGameplayStatics::SpawnEmitterAtLocation(GetWorld(), ImpactEffect, Hit.Location, ShotDirection.Rotation());

AActor\* HitActor = Hit.GetActor();

if (HitActor != nullptr)

{

FPointDamageEvent DamageEvent(Damage, Hit, ShotDirection, nullptr);

HitActor->TakeDamage(Damage, DamageEvent, OwnerController, this);

}

}

}

== MyCharacter.cpp

#include "Components/CapsuleComponent.h"

float AShooterCharacter::TakeDamage(float DamageAmount, struct FDamageEvent const &DamageEvent, class AController \*EventInstigator, AActor \*DamageCauser)

{

float DamageToApply = Super::TakeDamage(DamageAmount, DamageEvent, EventInstigator, DamageCauser);

DamageToApply = FMath::Min(Health, DamageToApply);

Health -= DamageToApply;

UE\_LOG(LogTemp, Warning, TEXT("Health left %f"), Health);

if (IsDead())

{

DetachFromControllerPendingDestroy();

GetCapsuleComponent()->SetCollisionEnabled(ECollisionEnabled::NoCollision);

}

return DamageToApply;

}

◎ 게임 종료하기

== Character.cpp

#include "SimpleShooterGameModeBase.h"

if (IsDead())

{

DetachFromControllerPendingDestroy();

GetCapsuleComponent()->SetCollisionEnabled(ECollisionEnabled::NoCollision);

ASimpleShooterGameModeBase\* GameMode = GetWorld()->GetAuthGameMode<ASimpleShooterGameModeBase>();

if (GameMode != nullptr)

{

GameMode->PawnKilled(this);

}

}

만들기

== SimpleShooterGameModeBase.h

class SIMPLESHOOTER\_API ASimpleShooterGameModeBase : public AGameModeBase

{

GENERATED\_BODY()

public:

virtual void PawnKilled(APawn\* PawnKilled);

};

== SimpleShooterGameModeBase.cpp

#include "SimpleShooterGameModeBase.h"

void ASimpleShooterGameModeBase::PawnKilled(APawn\* PawnKilled)

{

}

SimpleShooterGameModeBase를 상속받는 c++ 클래스 만들기

== KillEmAllGameMode.h

#pragma once

#include "CoreMinimal.h"

#include "SimpleShooterGameModeBase.h"

#include "KillEmAllGameMode.generated.h"

UCLASS()

class SIMPLESHOOTER\_API AKillEmAllGameMode : public ASimpleShooterGameModeBase

{

GENERATED\_BODY()

public:

virtual void PawnKilled(APawn\* PawnKilled) override;

};

== KillEmAllGameMode.cpp

#include "KillEmAllGameMode.h"

void AKillEmAllGameMode::PawnKilled(APawn\* PawnKilled)

{

Super::PawnKilled(PawnKilled);

UE\_LOG(LogTemp, Warning, TEXT("A pawn was killed!"));

}

게임 모드를 BP\_KillEmAllGameMode로 이름 바꾸고 설정 -> 상속받는 클래스도 KillEmAllGameMode로

◎ 타이머 설정

playercontroller 상속받는 c++파일 만들고

그를 상속받는 BP도 만든다.

게임모드에서 -> 컨트롤러 연결

== KillEmAllGameMode.cpp

void AKillEmAllGameMode::PawnKilled(APawn\* PawnKilled)

{

Super::PawnKilled(PawnKilled);

APlayerController\* PlayerController = Cast<APlayerController>(PawnKilled->GetController());

if (PlayerController != nullptr)

{

PlayerController->GameHasEnded(nullptr, false);

}

}

== Character.cpp

if (IsDead())

{

ASimpleShooterGameModeBase \*GameMode = GetWorld()->GetAuthGameMode<ASimpleShooterGameModeBase>();

if (GameMode != nullptr)

{

GameMode->PawnKilled(this);

}

DetachFromControllerPendingDestroy();

GetCapsuleComponent()->SetCollisionEnabled(ECollisionEnabled::NoCollision);

}

== ShooterPlayerController.h

#pragma once

#include "CoreMinimal.h"

#include "GameFramework/PlayerController.h"

#include "ShooterPlayerController.generated.h"

UCLASS()

class SIMPLESHOOTER\_API AShooterPlayerController : public APlayerController

{

GENERATED\_BODY()

public:

virtual void GameHasEnded(class AActor \*EndGameFocus = nullptr, bool bIsWinner = false) override;

private:

UPROPERTY(EditAnywhere)

float RestartDelay = 5;

FTimerHandle RestartTimer;

};

== ShooterPlayerController.cpp

#include "ShooterPlayerController.h"

#include "TimerManager.h"

void AShooterPlayerController::GameHasEnded(class AActor \*EndGameFocus, bool bIsWinner)

{

Super::GameHasEnded(EndGameFocus, bIsWinner);

GetWorldTimerManager().SetTimer(RestartTimer, this, &APlayerController::RestartLevel, RestartDelay);

}

|  |  |  |  |
| --- | --- | --- | --- |
| **문제점 정리** |  | | |
| **해결방안** |  | | |
| **다음주차** |  | **다음기간** |  |
| **다음주 할일** | 캐릭터 1 만든 것 언리얼환경에서 다루어 보고 기본 동작 만들기 | | |
| **지도 교수**  **Comment** |  | | |