# Project N

| | |
|---|---|
| develop period | @2024년 11월 17일 → 2024년 12월 23일 |
| roll | 프로그래밍 |
| category | 개인프로젝트 언리얼 게임 |
| size | Large |
| FrameWork | Unreal |
| 언어 | C++ |



# 개요

전체 코드는 다음 깃에 올라가 있습니다.

https://github.com/kimkyungjae1112/Project_N

# 게임 설명

## P의 거짓의 전투 시스템 클론 구현

P의 거짓을 플레이 해본 후 전투 시스템이 마음에 들어 해당 프로젝트를 시작하게 되었습니다.

## 게임 테마 및 스토리

게임의 장르는 소울라이크이다. 베일 왕국이란 가상의 배경은 깊고 음울한 다크 판타지 세계입니다. 몬스터들에게 왕국(성)을 빼앗긴 주인공은 다시 왕국을 찾기 위해 위험한 여정을 시작합니다. 적들을 물리치고 장애물을 넘어 끝끝내 왕국을 빼앗고 왕좌를 차지하고 있던 적의 우두머리와 만나게 되며, 우두머리를 물리치고 왕국을 다시 찾게 됩니다.

## 게임 규칙과 절차

- 퀘스트 트리거

  게임이 시작되고 플레이어가 특정 위치에 도달하면 퀘스트를 시작하게 됩니다.

  예) 일반 몬스터를 일정 수 이상 제거

- 보스 전투

  퀘스트 완료 시, 보스로 가는 길이 열리고 보스와의 대결이 가능해집니다.

## 게임 전투 시스템

- 스테미나 관리

  공격, 회피, 특정 이동 시 스테미나를 소모하며, 스테미나가 바닥나면 다른 행동이 불가능합니다.

- 콤보 시스템

  사용할 수 있는 공격은 총 4가지로, 현재 상태에 따라 다양한 콤보를 만들어 낼 수 있습니다.

  좌클릭만 누를 시, 약공격 실행 (총 5콤보)

  좌클릭 1번 클릭 후 우클릭 시, 총 3콤보의 강공격 실행 (총 3콤보)

  달리면서 좌클릭 시 대쉬 공격

  좌클릭을 2초 동안 누르고 있을 시 차지 공격

  좌클릭 없이 우클릭만 누르면 방어

## 게임 플랫폼 및 조작법

PC 윈도우 플랫폼에서 실행되며, 마우스 및 키보드를 이용하며 게임을 진행합니다.

이동 – WASD
달리기 – Shift
앉기 – Ctrl
구르기 - C
무기 넣기/빼기 - Tab
시점 – 마우스 XY축

공격 – 마우스 좌클릭 및 우클릭
방어 – 마우스 우클릭

## 플레이 영상

https://prod-files-secure.s3.us-west-2.amazonaws.com/7ebf7000-f855-492a-973c-2e002d905ac7/2bae0125-6b3b-4b75-ac66-f92ecd39a64f/ProjectN_%ED%94%8C%EB%A0%88%EC%9D%B4_%EC%98%81%EC%83%81.mp4

https://drive.google.com/file/d/16cSb3JgFYUaeaxbVX6jHcfQzHpt080LG/view?usp=sharing

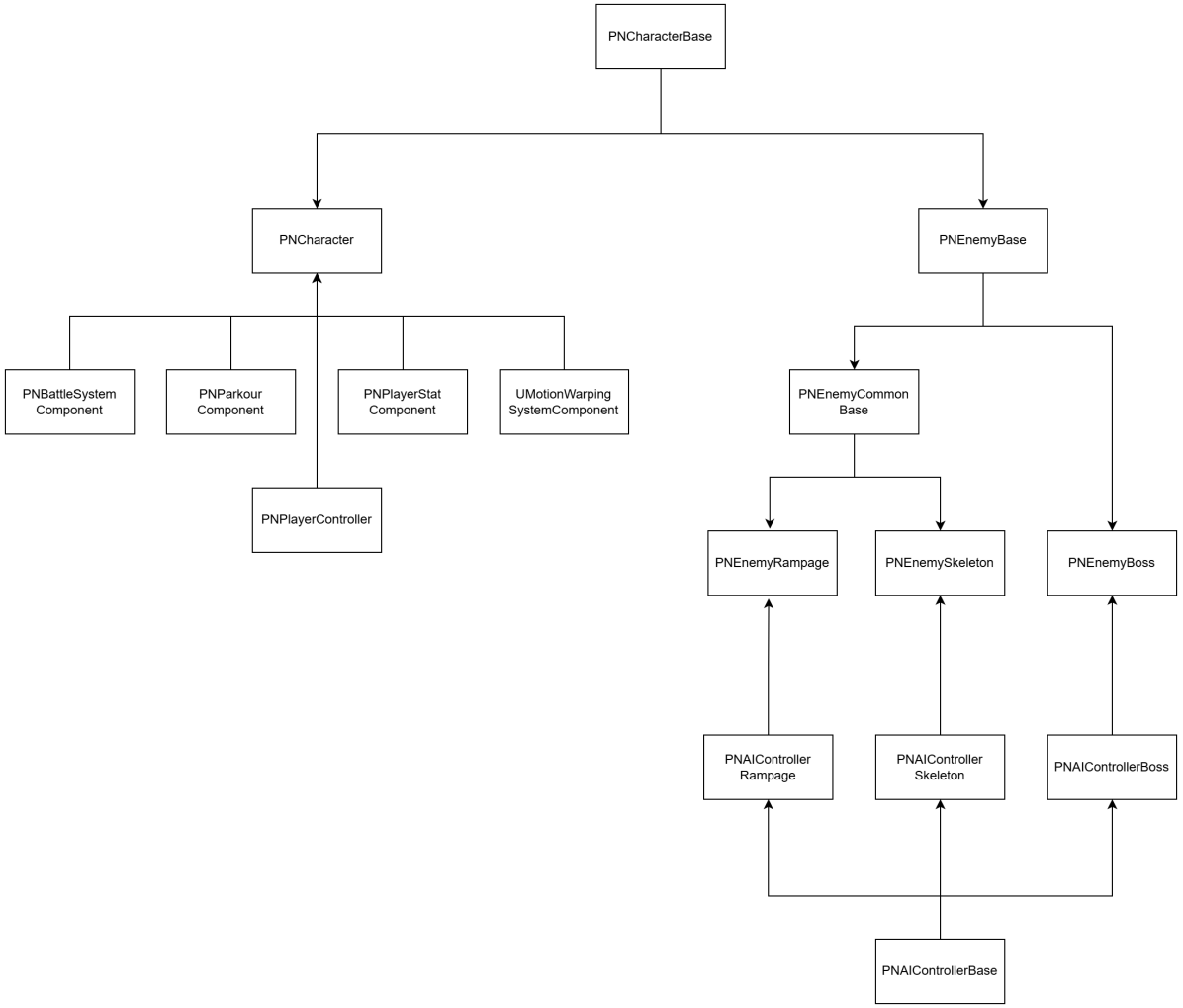
# 개발

현 프로젝트를 기획하며 중요한 개발 포인트를 총 3가지로 생각했습니다.

1. 플레이어의 다양한 콤보
2. 플레이어의 정밀한 공격 판정
3. 몬스터들의 똑똑한 AI

마우스만으로 다양한 콤보 공격을 구현하기 위해선 플레이어의 상태에 따라 다른 공격 함수들이 호출되도록 만드는 것이 가장 이상적일 것이라 생각했습니다.

### 코드 클래스 구조




PNCharacterBase라는 상위 클래스를 두고 플레이어와 적으로 나뉨

플레이어 캐릭터는 PNCharacter로 구현되어 있고 기능은 컴포넌트로 모두 구현

적은 자신의 AIController에 빙의되며,  BehaviorTree와 Blackboard, AI Perception를 사용해서 행동 패턴을 결정

그 외의 Notify, UI, Gimmick 등 게임 요소에 필요한 것들 또한 C++ 클래스로 구현되어 있음

▼ **해당 화살표를 누르면 자세한 설명을 보실 수 있습니다.**

**코드 및 설정에 대한 설명입니다.**

## 전투 시스템

- 공격
  - 플레이어 캐릭터의 공격은 UPNBattleSystemComponent 에서 구현 및 관리합니다.
  - 공격의 기본 설정

    1. 마우스 좌클릭으로 플레이어가 바인딩된 공격 함수를 실행합니다.

```
void APNCharacter::MouseLeftAttack()
{
    IsCharge = true;

    BattleSystemComp→Attack();
}
```

    2. 공격 함수 실행

       현재 어떤 공격을 실행 중인지 enum class로 구분 중입니다.

       처음 좌클릭을 이용해 공격을 실행하면 Charge() 함수를 호출하게 됩니다.

       Charge() 함수는 Charge 모션을 취하는 애니메이션 몽타주를 실행하는 함수입니다.

```
UENUM()
enum class EAttackState : uint8
{
    ASIdle,
    ASLight,
    ASHeavy,
    ASCharge,
    ASDash
};

void UPNBattleSystemComponent::Attack()
{
    switch (CurrentAttackState)
    {
    case EAttackState::ASIdle:
        Charge();
```

```
        break;
    case EAttackState::ASLight:
        if (!LightAttackTimer.IsValid())
            HasNextLightAttack = false;
        else
            HasNextLightAttack = true;
        break;
    }
}

void UPNBattleSystemComponent::Charge()
{
    if (!Anim->Montage_IsPlaying(ChargeMontage))
    {
        Anim->Montage_Play(ChargeMontage);
    }

    FOnMontageEnded MontageEnd;
    MontageEnd.BindUObject(this, &UPNBattleSystemComponent::End
    Anim->Montage_SetEndDelegate(MontageEnd, ChargeMontage);
}
```

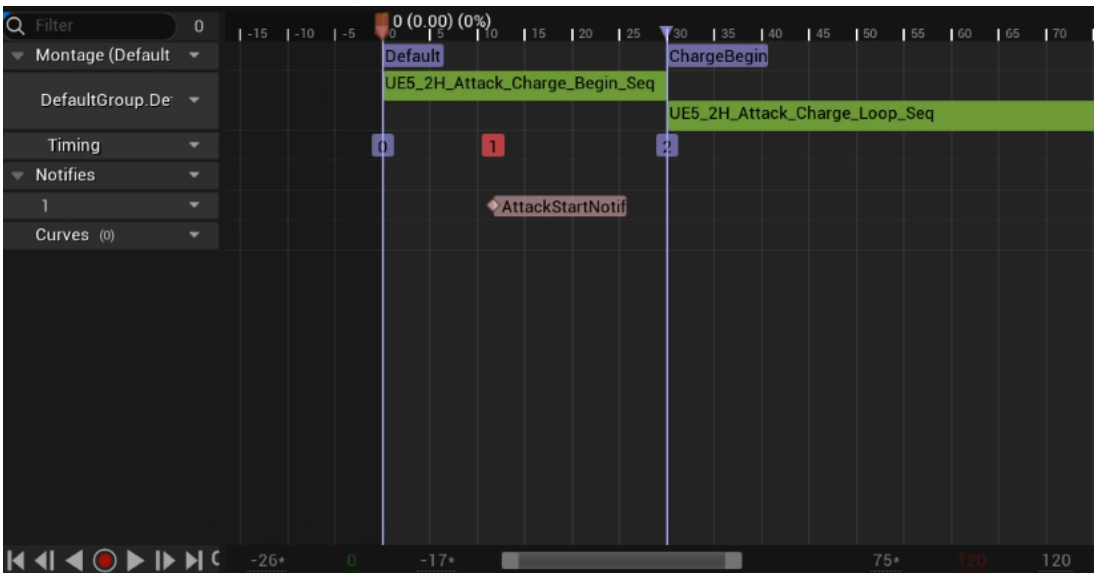
3. IsCharge의 값에 따라 공격 방식이 분기됩니다.

   이때 IsCharge라는 bool 형식의 값이 true로 바뀌고 일정 프레임 이상 true 상태라면 Charge 행동을 취하고, false 상태라면 약공격으로 전환됩니다.

   IsCharge는 플레이어가 좌클릭을 누르고 있으면 true, 좌클릭을 때면 false로 값이 바뀝니다.

```
void UAttackStartNotify::Notify(USkeletalMeshComponent* MeshCom
UAnimSequenceBase* Animation, const FAnimNotifyEventReference&
{
    Super::Notify(MeshComp, Animation, EventReference);

    APNCharacter* Player = Cast<APNCharacter>(MeshComp->GetOw
    if (Player)
    {
```

```
        if (Player->GetIsCharge())
        {
            UPNBattleSystemComponent* BSComp = Cast<UPNBattleSyst
                        (Player->GetComponentByClass<UPNBattleSystem

            BSComp->SuccessCharge();
        }
        else
        {
            UPNBattleSystemComponent* BSComp = Cast<UPNBattleSyst
                        (Player->GetComponentByClass<UPNBattleSystem

            BSComp->FailCharge();
        }

        Player->SetAfterAttack(true);
    }
}
```




- 약공격 : 최대 콤보 5회

▼ **코드 흐름**

1. 약공격

IsCharge가 false가 되면 FailCharge() 함수를 호출합니다.

Charge 몽타주를 멈추고 LightAttack() 함수를 호출하여 약공격을 실행합니다.

현재 공격의 상태는 Light로 바뀌며, 플레이어가 다음번 좌클릭을 클릭 시 현재 약공격 콤보 공격이 가능한지 검사 후 약공격을 계속할지, 중단할지 결정합니다.

```
void UPNBattleSystemComponent::FailCharge()
{
    CurrentAttackState = EAttackState::ASLight;

    Anim->Montage_Stop(0.1f, ChargeMontage);
    LightAttack();
}

void UPNBattleSystemComponent::LightAttack()
{
    CurrentLightAttackCombo = 1;
    StatComp->ApplyEnergy(StatComp->UseLightAttackEnergy());
    Anim->Montage_Play(AttackMontage);

    FOnMontageEnded MontageEnd;
    MontageEnd.BindUObject(this, &UPNBattleSystemComponent::EndL
    Anim->Montage_SetEndDelegate(MontageEnd, AttackMontage);

    LightAttackTimer.Invalidate();
    SetTimerLightAttack();
}
```

- 강공격 : 최대 콤보 3회

▼ **코드 흐름**

1. 마우스 우클릭으로 플레이어가 바인딩된 공격 함수를 실행합니다.

```
void APNCharacter::MouseRightAttack()
{
```

```
    BattleSystemComp→HeavyAttack();
}
```

2. 약공격의 콤보가 1일 때 강공격을 실행할 수 있습니다.

   처음 강공격을 실행하기 위해서 약공격의 콤보가 1일 때 진행될 수 있도록, 그 후 공격 상태를 Heavy(강공격)으로 바꿔 진행합니다.

   최대 3콤보까지 강공격을 진행할 수 있습니다.

```
void UPNBattleSystemComponent::HeavyAttack()
{
    if (CurrentLightAttackCombo == 1 || CurrentAttackState == EAttackS
    {

        CurrentAttackState = EAttackState::ASHeavy;

        if (CurrentHeavyAttackCombo == 0)
        {
            BeginHeavyAttack();
            return;
        }

        if (!HeavyAttackTimer.IsValid())
            HasNextHeavyAttack = false;
        else
            HasNextHeavyAttack = true;
    }
}
```

- 차지 공격
  - Input Action의 Triggers 옵션을 이용해 구현
  - Hold 옵션으로 1초 이상 누르고 있을 시 공격 실행
- 대쉬 공격
  - 현재 플레이어 캐릭터 상태가 Dash 상태일 때 좌클릭 시 공격 실행

- 공격 판정
  - 실제 칼을 휘두르는 궤적에 적이 부딪히거나, 적이 휘두르는 궤적에 플레이어가 닿았을 때, 대미지를 주고 받도록 설계
  - AnimNotifyState 클래스를 사용해 구현

  ▼ **코드 흐름**

  플레이어 캐릭터의 공격 판정을 예시로 들겠습니다.

  1. AnimNotifyState 클래스 설계

     상위 클래스의 Begin, Tick End 가상 함수를 오버라이딩 합니다.

     지금 적에게 행하려는 공격이 어떤 타입의 공격인지 검사합니다.

     라인 트레이스를 만들어 해당 라인에 접촉하면 대미지를 전달합니다.

     한 번 피격 당한 적이 더 당하지 않도록 TSet 컨테이너를 활용합니다.

```
UCLASS()
파생된 블루프린트 클래스 0개
class PROJECTN_API USwordAttackHitNotifyState : public UAnimNotifyState
{
    GENERATED_BODY()

public:
    USwordAttackHitNotifyState();

    virtual void NotifyBegin(USkeletalMeshComponent* MeshComp, UAnimSequenceBase*
    virtual void NotifyTick(USkeletalMeshComponent* MeshComp, UAnimSequenceBase* 
    virtual void NotifyEnd(USkeletalMeshComponent* MeshComp, UAnimSequenceBase* A

private:
    void DecideAttackType(const EAttackState& AttackState);
    void MakeLineTrace(AActor* Owner);

    UPROPERTY()
    0 Blueprints에서 변경됨
    TSet<AActor*> HitEnemys;

    float Damage;
    FName DamageType;

};
```

  2. AnimNotifyState 클래스 구현

     AnimNotifyState가 트리거 되면 UPNBattleSystemComponent 클래스에서 현재 공격 타입을 가져옵니다.

그 후 MakeLineTrace를 통해 무기 메쉬의 칼날 부분에 라인 트레이스를 생성합니다.

라인 트레이스에 닿은 적들은 TSet 컨테이너에 저장 후 대미지를 적용합니다.

몬스터들은 대미지를 받는 함수를 따로 구현하였으며, 대미지 타입, 피격 위치에 따라 피격 모션이 다릅니다.

AnimNotifyState가 종료되면 TSet 컨테이너를 비웁니다.

```
void USwordAttackHitNotifyState::NotifyBegin(...)
{
   Super::NotifyBegin(MeshComp, Animation, TotalDuration, EventRefe

   if (UPNBattleSystemComponent* BSComp =
            MeshComp→GetOwner()→GetComponentByClass<UPNBattl
   {
      DecideAttackType(BSComp→GetAttackState());
   }
}

void USwordAttackHitNotifyState::NotifyTick(...)
{
   Super::NotifyTick(MeshComp, Animation, FrameDeltaTime, EventRe

   MakeLineTrace(MeshComp→GetOwner());
}

void USwordAttackHitNotifyState::NotifyEnd(...)
{
   Super::NotifyEnd(MeshComp, Animation, EventReference);

   HitEnemys.Empty();
}

void USwordAttackHitNotifyState::DecideAttackType(const EAttackSta
{
   switch (AttackState)
   {
   case EAttackState::ASLight:
```

```
        Damage = 300.f;
        DamageType = TEXT("Light");
        break;
    case EAttackState::ASHeavy:
        Damage = 700.f;
        DamageType = TEXT("Heavy");
        break;
    case EAttackState::ASDash:
        Damage = 400.f;
        DamageType = TEXT("Dash");
        break;
    case EAttackState::ASCharge:
        Damage = 800.f;
        DamageType = TEXT("Charge");
        break;
    default:
        Damage = -1.f;
        DamageType = TEXT("Error");
        UE_LOG(LogTemp, Error, TEXT("Player Sword Hit Error"));
        break;
    }
}

void USwordAttackHitNotifyState::MakeLineTrace(AActor* Owner)
{
    IWeaponSocketCarryInterface* WeaponInterface =
            Cast<IWeaponSocketCarryInterface>(Owner);
    if (WeaponInterface)
    {
        USkeletalMeshComponent* SwordMesh = WeaponInterface->Get
        if (SwordMesh)
        {
            FVector StartLoc = SwordMesh->GetSocketLocation(TEXT("Sw
            FVector EndLoc = SwordMesh->GetSocketLocation(TEXT("Swo

            FHitResult HitResult;
            FCollisionQueryParams Params(NAME_None, true, Owner);
```

```
            bool bHit = Owner→GetWorld()→LineTraceSingleByChannel
                    (HitResult, StartLoc, EndLoc, ECC_GameTraceChannel2,
            if (bHit && !HitEnemys.Contains(HitResult.GetActor()))
            {
                HitEnemys.Add(HitResult.GetActor());

                if (IEnemyApplyDamageInterface* Enemy =
                        Cast<IEnemyApplyDamageInterface>(HitResult.GetActo
                {
                    Enemy→ApplyDamage(Damage, Owner, DamageType, HitI
                    return;
                }
            }
        }
    }
}
```

## 플레이어 상태

- 상태 변화가 감지되면 델리게이트를 통해 바인딩 된 함수 호출로 상태 변경
- 이때 해당 상태에 맞는 **InputMappingContext** 변경

▼ **코드 흐름**

1. 플레이어 캐릭터 상태 변화에 관여하는 멤버 함수 및 멤버 변수입니다.

```
/*** 플레이어 캐릭터 상태 enum class ***/
UENUM(BlueprintType)
enum class EBehaviorState : uint8
{
    EWalk = 0,
    ERun,
    ECrouch,
    EJump,
    ENonCombat
};
```

```
/*** 플레이어 캐릭터 상태 변화 델리게이트 Wrapper 구조체 ***/
DECLARE_DELEGATE(FChangeBehaviorState)
USTRUCT()
struct FChangeBehaviorStateWarpper
{
    GENERATED_BODY()

    FChangeBehaviorStateWarpper() {}
    FChangeBehaviorStateWarpper(const FChangeBehaviorState& ChagneI
        ChangeBehaviorState(ChagneBehaviorState) { }

    FChangeBehaviorState ChangeBehaviorState;
};


/*** 상태값에 따라 상태 변화 델리게이트 Wrapper 구조체 관리 ***/
UPROPERTY(VisibleAnywhere, Category = "State")
TMap<EBehaviorState, FChangeBehaviorStateWarpper> ChangeBehavior

void SetBehaviorState(const EBehaviorState& BehaviorState);
void SetBehaviorStateWalk();
void SetBehaviorStateRun();
void SetBehaviorStateCrouch();
void SetBehaviorStateJump();
void SetBehaviorStateNonCombat();


/*** 상태값에 따라 InputMappingContext 관리 ***/
UPROPERTY(VisibleAnywhere, Category = "Input")
TMap<EBehaviorState, class UInputMappingContext*> IMC;
```

2. 상태 변화 함수 호출

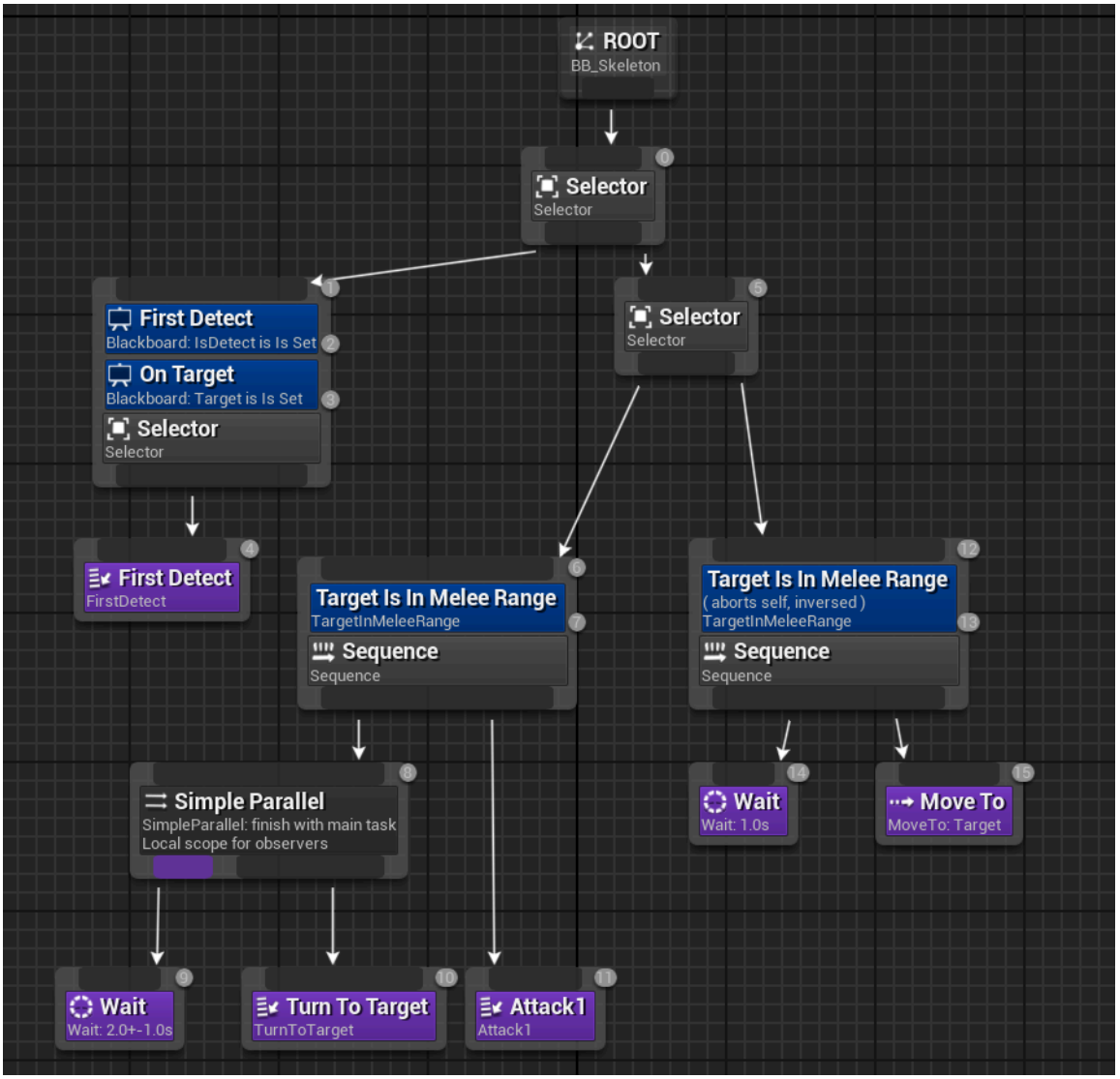
   해당 함수를 호출하면 현재 상태에 맞는 함수를 호출해줍니다.

```
void APNCharacter::SetBehaviorState(const EBehaviorState& BehaviorSta
{
```

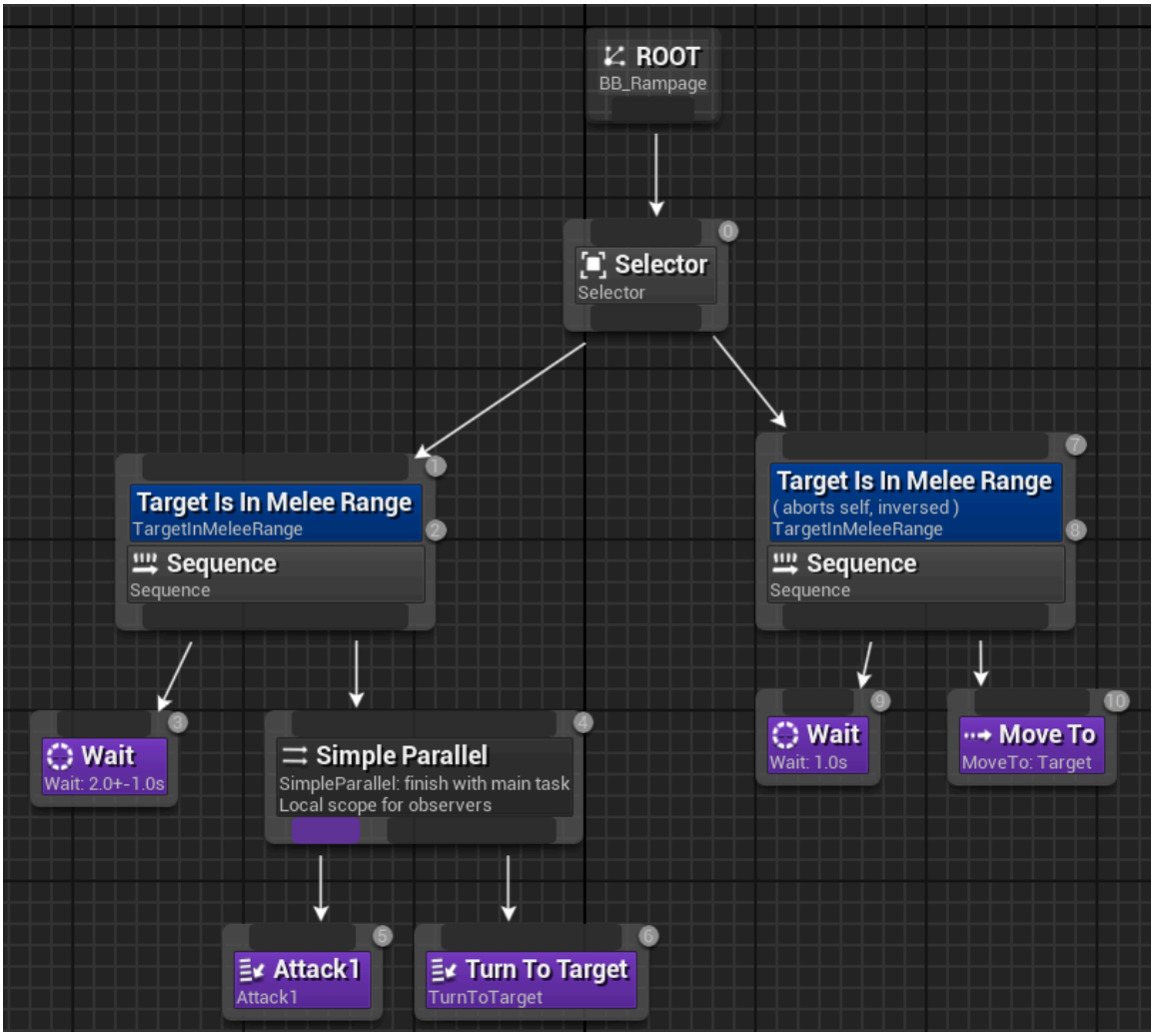
```
    ChangeBehaviorStateMap[BehaviorState].ChangeBehaviorState.Execut
}

void APNCharacter::SetBehaviorStateWalk()
{
    if (UEnhancedInputLocalPlayerSubsystem* Subsystem =
                    ULocalPlayer::GetSubsystem<UEnhancedInputLocalPlaye
                    (GetMyController()->GetLocalPlayer()))
    {
        Subsystem->ClearAllMappings();
        Subsystem->AddMappingContext(IMC[CurrentBehaviorState], 0);
    }
}
```

## 몬스터

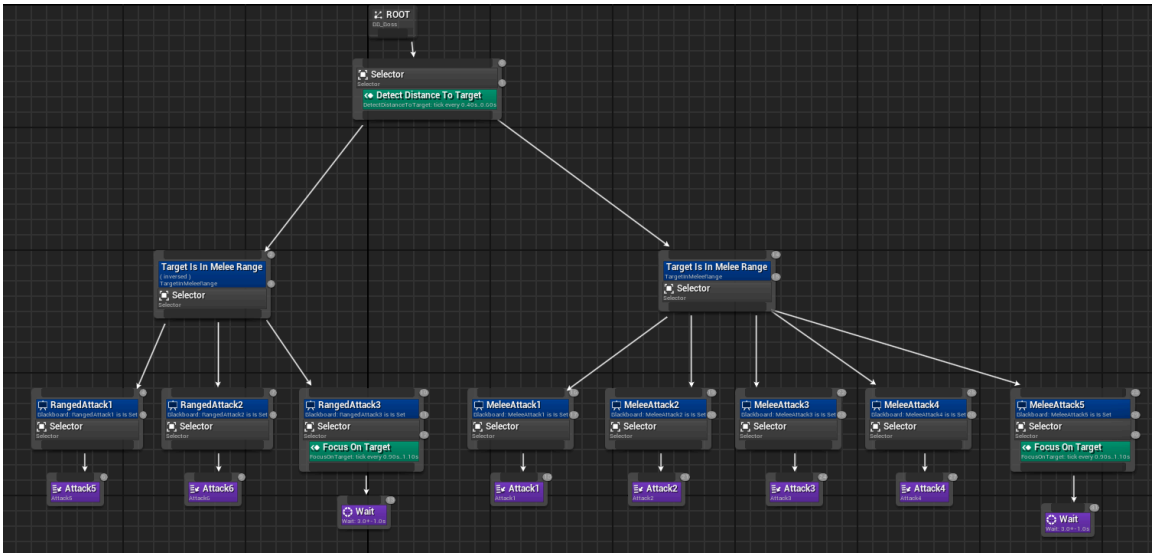
- 스켈레톤 몬스터
    - 플레이어 캐릭터가 최초 탐지 범위에 들어오지 않으면 AIController가 실행되지 않음
    - 플레이어 캐릭터가 최초 탐지 범위에 들어오면 애니메이션과 AIController 실행
    - 공격 범위에 플레이어 캐릭터가 들어오면 일정 시간 공격 쿨타임을 가진 후 공격 진행
    - 공격 범위를 벗어나면 플레이어 캐릭터를 쫓아감




- Rampage 몬스터
  - 플레이어 캐릭터가 최초 탐지 범위에 들어오지 않으면 AIController가 실행되지 않음
  - 공격 범위에 플레이어 캐릭터가 들어오면 일정 시간 공격 쿨타임을 가진 후 공격 진행
  - 공격 범위를 벗어나면 플레이어 캐릭터를 쫓아감




- 보스 몬스터
  - 근접 공격 4가지 패턴과 원거리 공격 2가지 패턴 존재
  - Detect Distance To Target Service 노드를 사용해서 어떤 공격을 할지 확률적으로 결정




# 회고

## 새로 알게된 점

1. NotifyState를 통해 애니메이션 몽타주가 실행되는 기간을 정해서 로직을 실행시킬 수 있음을 알게됨
2. 퀘스트와 시네마틱, 몬스터의 스폰 애니메이션 등 지금까지는 구현해보지 않았던 기능들을 구현해봄
3. Input Action에도 인풋을 핸들링 할 옵션들이 있었다는 것을 알게됨

## 아쉬운점

1. 퀘스트를 싱글톤 클래스를 통해 관리하는게 맞는지 잘 모르겠다.

   → 정적 데이터를 최대한 줄이는 것이 나은 방향 같아보임.