2단계_06. 캐릭터 공격 판정

수강일	@2023/07/10
이름	전영재
멘토	Min-Kang Song,현웅 최
멘티	
작성 상태	Done
단계	2단계
강의 시청 여부	☑
이수 여부	☐

Contents

캐릭터 공격 판정

💡 **강의 목표**

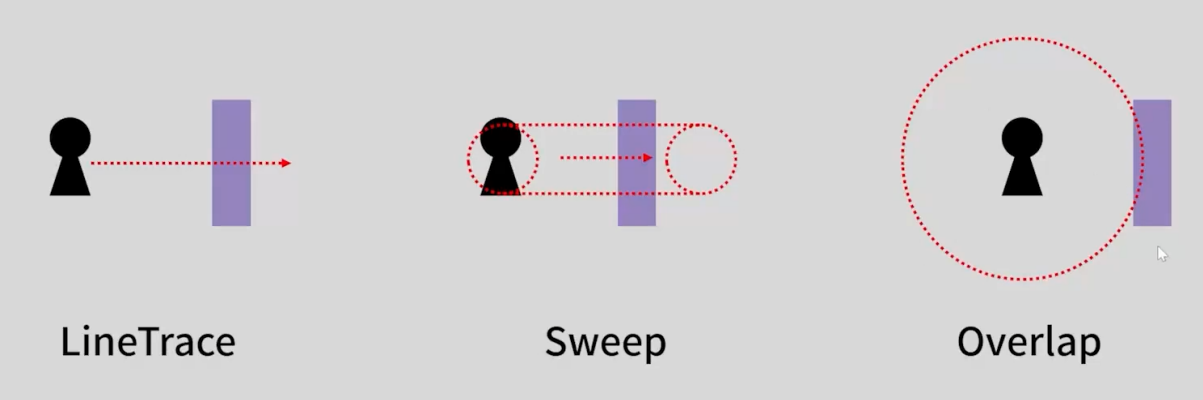
- 애니메이션 이벤트 발생을 위한 Notify 설정 방법의 이해
- 물리엔진의 트레이스 채널 설정과 이를 시각적으로 디버깅
- 데미지 프레임워크를 화용해 데미지 전달과 Dead 구현

---

[1] 충돌 채널의 설정

언리얼 엔진은 크게 3가지 **충돌 판정 서비스**를 제공한다.

Line Trace	지정한 방향으로 선분을 투사하여 충돌을 확인
Sweep	선분이 아닌 도형을 투사해 충돌을 확인
Overlap	지정한 영역 내에 다른 물체와 충돌이 존재했는지 여부를 확인

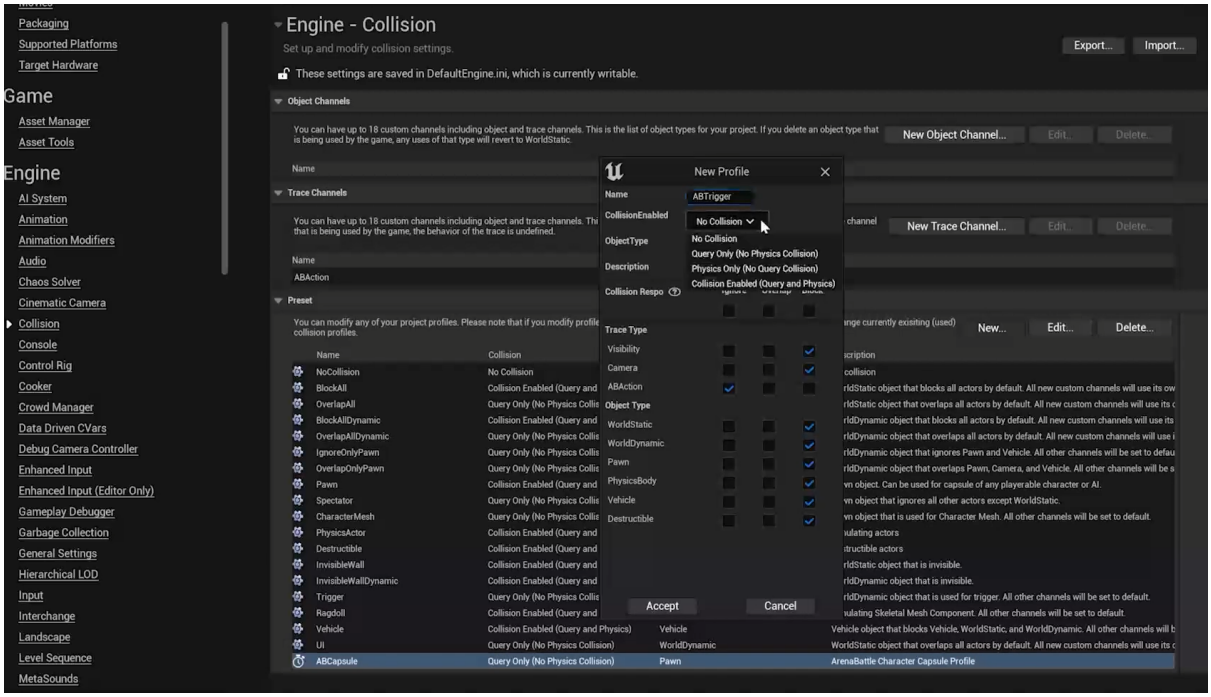


앞으로 위 기능과 게임의 충돌 프로필을 사용해 물리 판정 여부를 결정할 것인데, 이 때 캐릭터는 **Capsule Collision 컴포넌트**와 **Skeletal Mesh 컴포넌트 2가지 경우를 각각 생각**해 주어야 한다.

> 💡 구분해주지 않을 경우, 2번의 이벤트가 발생할 수 있다

해당 사진처럼 커스텀 Trace채널을 만들어주고, 그 Trace채널을 활용할 Preset프로필을 만들어 사용하면 된다.

주로 Tace Type은 Ignore과 Block만을 사용하는 경우가 많고, Object Type의 경우에는 상황에 맞는 collision반응을 설정해준다.

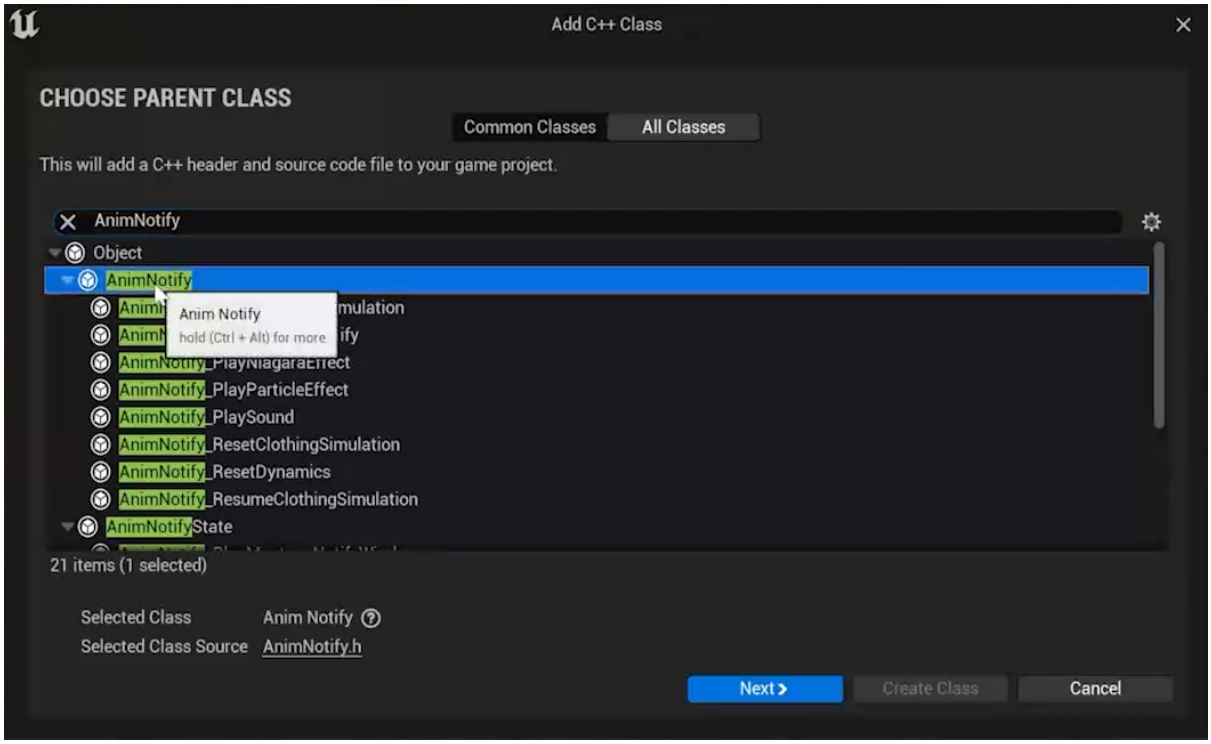


> 💡 설정해준 충돌 채널에 대한 정보는 **Project/Config/DefaultEngine.ini** 내에 기록된다.
> 이때 우리가 설정해준 이름이 아닌 **다른 이름(열거형)으로 저장**되기에, 이를 꼭 확인하고 사용하자.
> ex) ABAction → ECC_GameTraceChannel5

---

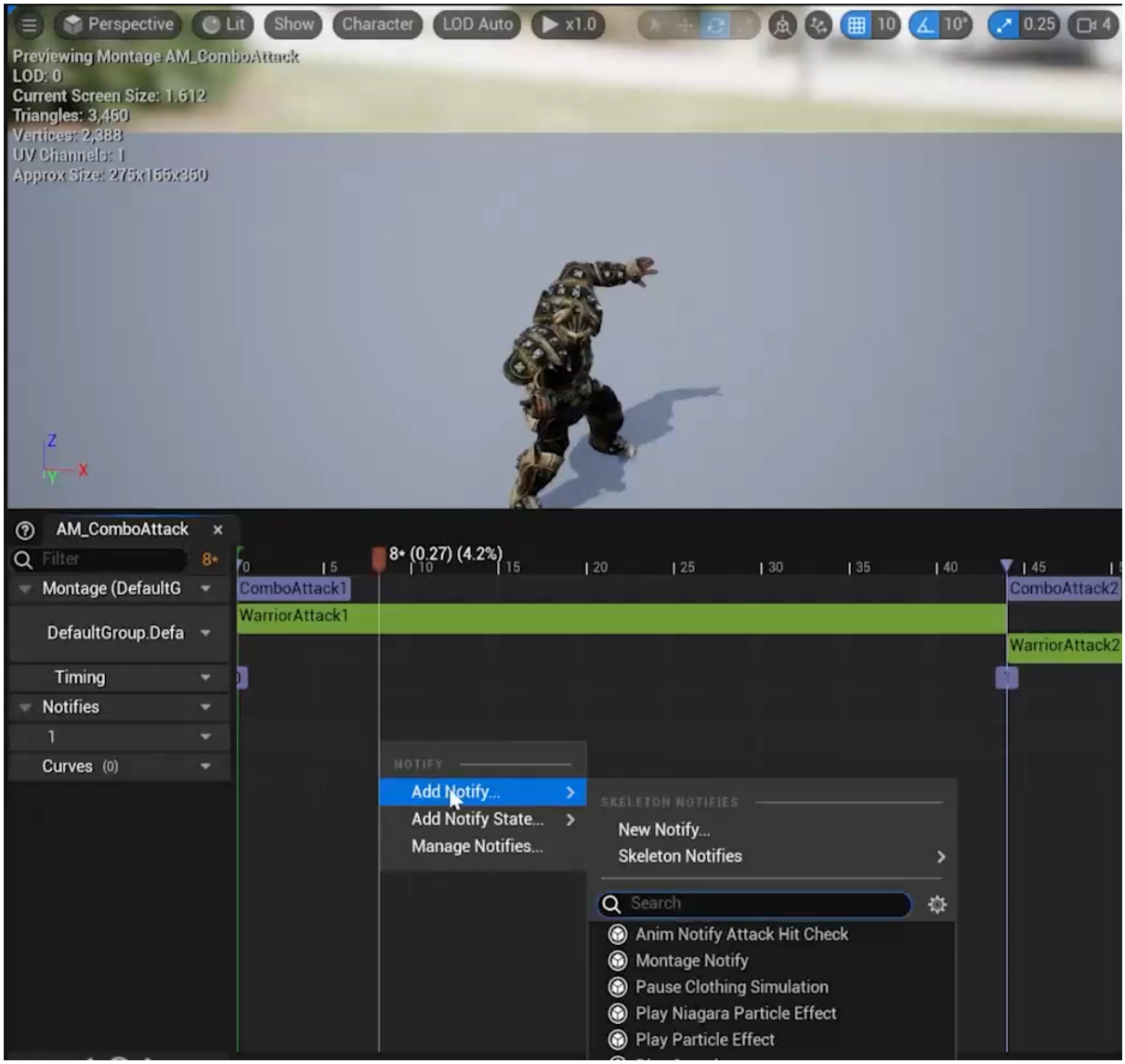
[2] 애니메이션 내 Notify 설정

애니메이션 애셋 내에서, **특정 프레임에 맞춰 다양한 이벤트를 발생**시킬수 있다.



내장된 다른 AnimNotify와 같은 형식으로 만들어주는게 후에 정렬하기 편하다. ex) AnimNotify_기능

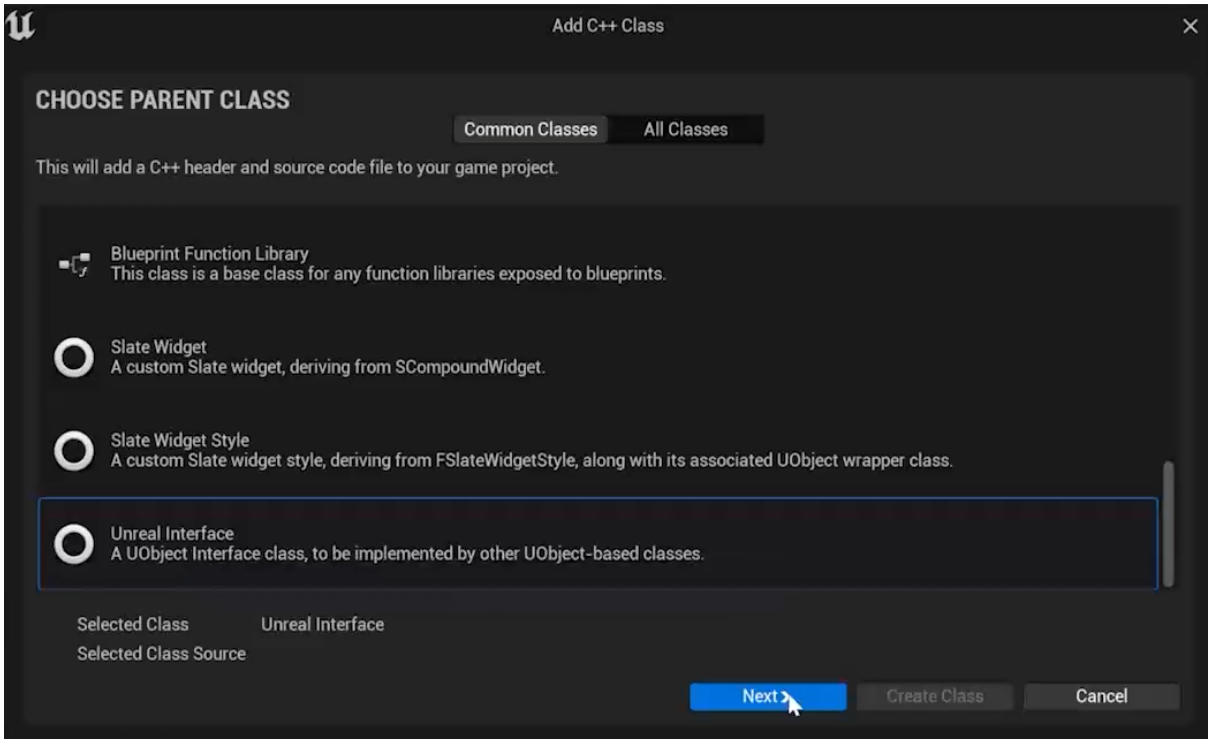
언리얼엔진에서 제공하는 기본 AnimNotify를 상속받는 커스텀 클래스를 생성해 애니메이션 애셋 내에서 사용한다.



헤더를 사용하지 않기 위한 인터페이스

위에서 생성한 AnimNotify클래스에서기능을 구현하려면 `Notify( )`를 오버라이드 하여 구현하면 된다. 그러나 특정 클래스의 정보를 사용하려면 해당 헤더가 필요로 함은 타 클래스들과 똑같다.

그때마다 헤더를 추가하면 **의존성이 생기므로 Interface**를 사용하는 것이 바람직하다.



필요한 클래스 내에서 해당 Interface를 소지하게 하고 내용을 만든 뒤, AnimNotify에서는 해당 클래스에서 Interface기능을 소지하고 있는가에 대한 여부를 사용하는 것이다.

즉, 직접적으로 header를 통해 연결하는 것이 아닌, Interface라는 **중간 매개체를 사용**하는 것이다.

▼ Interface 사용예시 코드

```
#include "CoreMinimal.h"
#include "UObject/Interface.h"
#include "ABAnimationAttackInterface.generated.h"

// This class does not need to be modified.
UINTERFACE(MinimalAPI)
class UABAnimationAttackInterface : public UInterface
{
  GENERATED_BODY()
};

class ARENABATTLE_API IABAnimationAttackInterface
{
  GENERATED_BODY()

  // Add interface functions to this class. This is the class that will be inherited to implement this interface.
public:
  //Interface기능 구현부. 여기서 구현할 필요없다.
  virtual void AttackHitCheck() = 0;
};
```

```
#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "Interface/ABAnimationAttackInterface.h"
#include "ABCharacterBase.generated.h"
...
UCLASS()
class ARENABATTLE_API AABCharacterBase : public ACharacter, public IABAnimationAttackInterface
{
  GENERATED_BODY()
  ...
// Attack Hit Section
protected:
  virtual void AttackHitCheck() override;
  virtual float TakeDamage(float DamageAmount, struct FDamageEvent const& DamageEvent, class AController* EventInstigator, AActor*
  ...
}
```

```
#include "Animation/AnimNotify_AttackHitCheck.h"
#include "Interface/ABAnimationAttackInterface.h"

void UAnimNotify_AttackHitCheck::Notify(USkeletalMeshComponent* MeshComp, UAnimSequenceBase* Animation, const FAnimNotifyEventRefe
{
  Super::Notify(MeshComp, Animation, EventReference);

  if (MeshComp)
  {
    //Interface를 통해 호출
    IABAnimationAttackInterface* AttackPawn = Cast<IABAnimationAttackInterface>(MeshComp->GetOwner());
    if (AttackPawn)
    {
      AttackPawn->AttackHitCheck();
    }
  }
}
```

💡 결국 AnimNotify클래스에서는 Interface를 헤더를 추가하니 그게 그거 아닌가? 라고 생각할 수 있지만 전혀 다르다.
1. Interface 내의 기능 변화는 극히 드물다.
2. Interface를 소지하는 여러 클래스에서 세부적인 기능을 변화할 수 있으므로(override), 이에 따른 추가적인 코드 변화에 대해 대응할(헤더 추가라던지) 필요가 없다.

---

[3] 공격 방법의 구현

이제 준비된 `AttackHitCheck( )` 함수 내에서 트레이스를 통해 히트 여부를 판단할 것이다.

내용을 구현하기에 앞서, 우리의 **커스텀 채널은 열거형으로 변해 다른 이름으로 사용**해야 한다고 말했다. 그러나 이 경우 코드의 직관성이 떨어지기에 간단한 헤더 파일을 제작해 **매크로를 설정**해주면 편리하게 사용 가능하다.

```
#include "CoreMinimal.h"

// ABCapsule프로필은 CPROFILE_ABCAPSULE이라 쓸거다
#define CPROFILE_ABCAPSULE TEXT("ABCapsule")
#define CPROFILE_ABTRIGGER TEXT("ABTrigger")
//ECC_GameTraceChannel1 (=TraceChannel)은 CCHANNEL_ABACTION이라 쓸거다
#define CCHANNEL_ABACTION ECC_GameTraceChannel1
```

전방으로 Sweep을 해서 Hit에 성공했다면 DamageEvent를 건네준다.

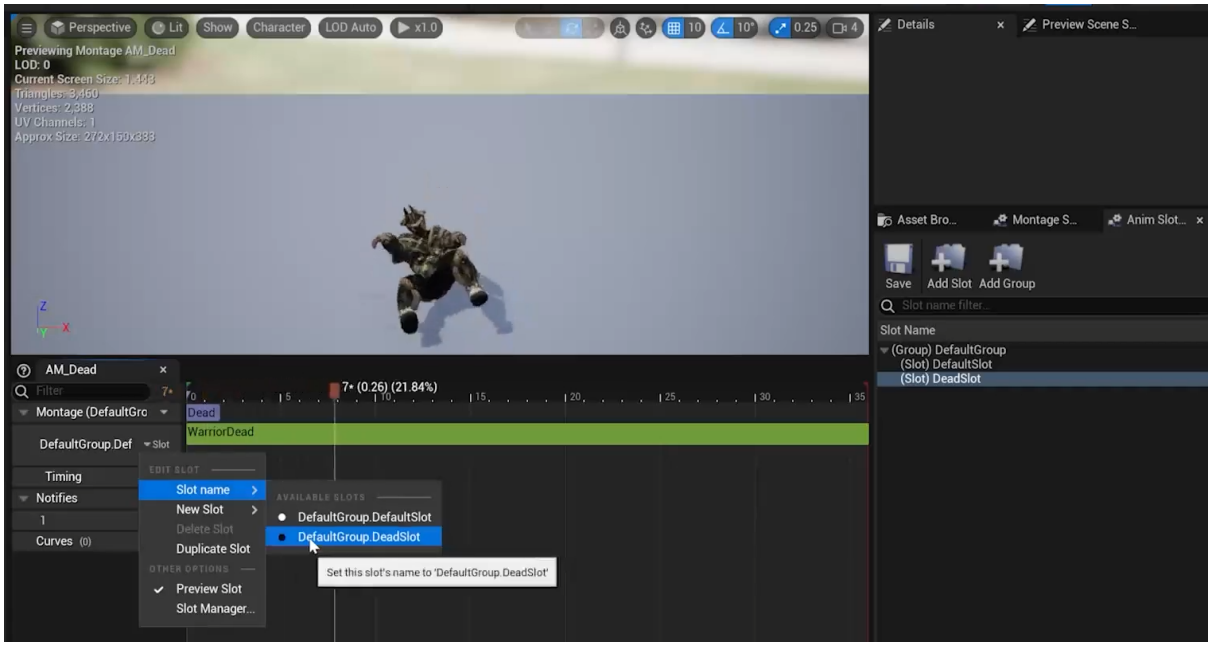
```
void AABCharacterBase::AttackHitCheck()
{
  FHitResult OutHitResult;
  FCollisionQueryParams Params(SCENE_QUERY_STAT(Attack), false, this);
  ... //float와 vector 생략
  bool HitDetected = GetWorld()->SweepSingleByChannel(
    OutHitResult,
    Start,
    End,
    FQuat::Identity,
    CCHANNEL_ABACTION,
    FCollisionShape::MakeSphere(AttackRadius),
    Params
  );
  if (HitDetected)
  {
    FDamageEvent DamageEvent;
    OutHitResult.GetActor()->TakeDamage(AttackDamage, DamageEvent, GetController(), this);
  }
}
```

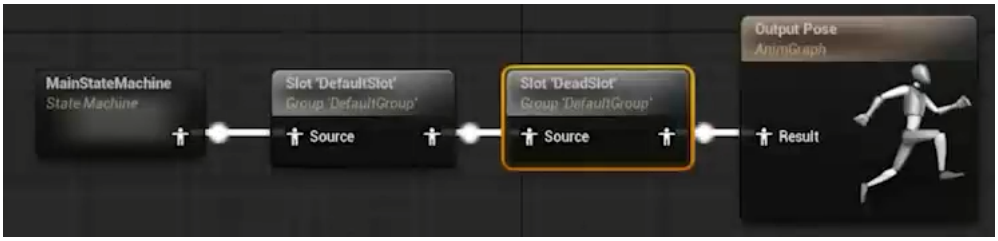
---

[4] Dead 구현

사망 애니메이션 몽타주를 만들어 캐릭터가 사망하면 해당 몽타주를 재생해주자.

이때, 몽타주 내의 슬롯 기능을 사용하여, 공격모션과 다른 슬롯에서 사망 애니메이션을 재생할 수 있다. 이 방법을 통해 슬롯간의 우선순위를 지정할 수 있다.





몽타주를 플레이 한다는 것은 몽타주 슬롯을 플레이 하는 것이고, 몽타주 애셋에서는 각각의 슬롯에 각각의 애니메이션을 담는 것이라 생각하면 이해하기 편할것이다.

람다 함수의 사용

간단한 1회성 함수의 경우에는 람다식을 사용하여 구현해줄 수 있다.

타이머를 설정하여 일정시간 후에 본인을 `Destroy( )`하는 함수를 구현한다고 하면, `Destroy( )`를 호출하기위해 새로운 함수를 만드는 것은 번거로운 작업일 것이다.

```
void AABCharacterNonPlayer::SetDead()
{
  Super::SetDead();

  FTimerHandle DeadTimerHandle;
  GetWorld()->GetTimerManager().SetTimer(DeadTimerHandle, FTimerDelegate::CreateLambda(
    [&]()
    {
      Destroy();
    }
  ), DeadEventDelayTime, false);
}
```

람다식의 사용법은 항상 헷갈려서 아래 조금더 추가해봤다.

```
// Lambda([주소](함수스타일){내용}) 의 형태로 쓰는거다

Lambda(
  [&]() //지금 이 Lambda함수의 주소값을 알린다.
  {
    //기능 구현부
  }
)
```

📌 **Summary**

- 언리얼엔진 내 충돌설정의 사용법
- 애니메이션 내 AnimNotify의 사용법
  - Interface를 활용한 코드 내 의존성 개선 방법
- 언리얼엔진의 Trace 기법과 데미지 프레임워크 사용법

0n. 캐릭터 공격 판정 강의 과제

❓ **Q1. 자신의 게임에서 활용할 채널과 프로필 설정 및 사용할 트레이스 기능에 대해 정리하시오.**

내 게임의 공격로직을 다시 점검해보자.

포탑은 적을 감지할 경우, 포구 전방에서 공격함수를 호출한다.

즉, 포구 전방에 상대가 존재하는지 확인하기 위해 트레이스를 사용해야 하며, 이 과정에서 전방 다수의 상대방을 색적해야(범위공격) 하고, 트레이스 대상 중 몬스터만을 감지하고 캐릭터 및 타 액터들은 감지하지 말아야 한다.

그렇다면 이 내용을 채널과 프로필을 이용해 만들려면 어떻게 해야할까?

1. 일단 포탑의 트레이스를 위한 개별 Trace Channel이 필요할 것이다.
2. 또한 이 채널과 상호작용하기 위한 커스텀 프로필이 필요하다.
3. 전방의 넓은 범위의 다수의 적을 트레이스 할 것이기에 LineTrace보다는 Sweep이 적합하고 Multi기능을 사용해야 하며, 앞서 커스텀 프로필을 설정해주었기에 SweepMultiByProfile을 사용하면 되겠다.

❓ **Q2. 자신의 게임에서 물리 오브젝트 타입을 추가할 계획이 있는지, 없다면 왜 그런지 이유를 설명하시오.**

건물의 충돌을 담당하는 오브젝트 타입을 추가할 계획이다.

내 게임에서는 건물은 단순히 공격할 뿐만 아니라, 그 자리에서 움직이지 않고 적들의 경로를 방해하는 역할도 해야한다. 하지만 WorldStatic으로 단순히 취급할 수는 없는것이 특정 액터들은 건물을 통과할 수 있어야 하기 때문이다.

EnemyCharacter들은 건물액터를 뚫고 이동할 수 없지만, 플레이어나 Projectile의 경우에는 경로를 방해받지 않고 뚫고 이동할 수 있어야 한다는 특수성이 있기 때문에 이것을 위한 특별한 오브젝트 타입이 필요하다.