전영재) 충돌설정과 데미지전달

생성일	@2023년 3월 30일 오후 4:29
태그	

충돌 설정과 데미지 전달

콜리전 설정

충돌(Collision)영역은 크게 3가지 방법으로 제작할 수 있다

스태틱메시 애셋	스태틱메시에 콜리전 영역을 심는다.	StaticMesh 컴포넌트에서 비주얼과 충돌 두가지 기능을 모두 설정할 수 있기에 관리가 편함.
기본도형 컴포넌트	구체, 박스, 캡슐 등의 기본도형을 사용해 스태틱메시와 별도의 충돌 영역을 제작한다.	스켈레탈 메시를 움직일 때 주로 사용되며, 기본 Character클래스에서 사용하는 방식이다.
피직스 애셋	캐릭터의 각 부위에 기본 도형으로 충돌 영역을 설정하고 이를 연결해 캐릭터의 물리를 설정한다.	관절이 흐느적 거리는 헝겊인형(RagDoll) 효과를 구현할 때 사용한다. 스켈레탈 메시에만 사용할 수 있다.

위 방법으로 제작한 충돌 영역의 다음 3가지 설정을 조정함으로써, 게임내의 물리 충돌을 구현할 수 있다.

1. 콜리전 채널과 기본 반응
2. 콜리전 채널의 용도
3. 다른 콜리전 채널과의 반응

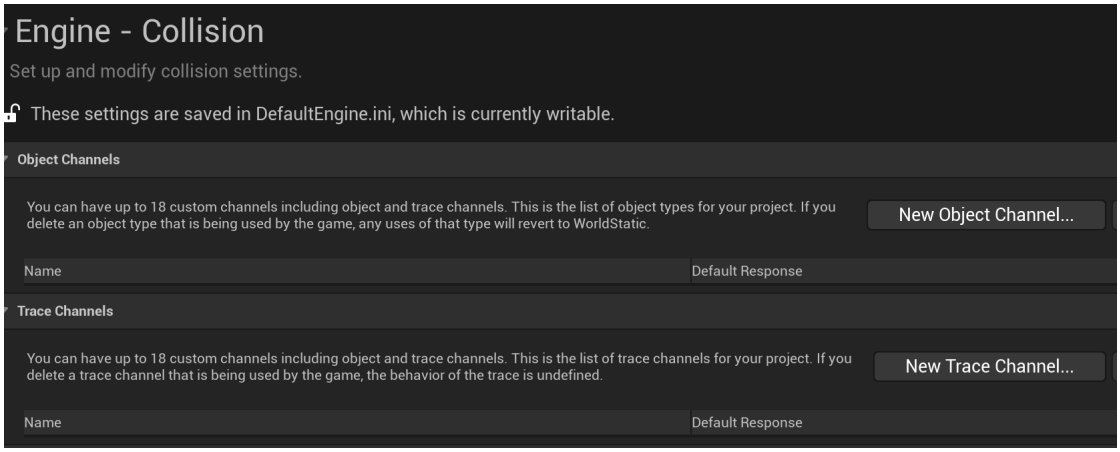
콜리전 채널과 기본 반응

- **WorldStatic** : 움직이지 않는 액터에 사용하는 콜리전 채널
- **WorldDynamic** : 움직이는 액터에 사용하는 콜리전 채널
- **Pawn** : 컨트롤러가 조종하는 물체에 주로 사용한다.
- **PhysicsBody** : 물리 시뮬레이션으로 움직이는 컴포넌트에 설정한다.
- **Visibility** : 시각적으로 보이는지 탐지하는데 사용한다.
  - 탐지 시, Pawn은 제외된다.
  - 마우스로 물체를 선택하는 피킹(Picking) 기능에 사용된다.
- **Camera** : 카메라와 목표물 간에 장애물이 있는지 사용한다.
- **Custom Collision Channel** : 엔진에서 기본적으로 제공하는 채널 외에도 사용자의 정의에 따라 새로운 콜리전 채널을 만들 수 있다.

  ▼ Custom Collision Channel 만드는법

  ProjectSettings→Engine→Collision 에서 Object 혹은 Trace 채널을 만들 수 있다.

Object **채널**	오브젝트와 오브젝트 간의 충돌을 지정
Trace **채널**	레벨을 뻗어나가는 가상의 선과 오브젝트 간의 충돌을 지정



Visibilty와 **Camera**의 차이점이 뭘까? 어쩌피 둘다 시각적으로 보이는지에 대한 콜리전을 지정하는 채널 아닌가?
→ 아니다! 이것을 이해하기 위해서는 **유리**와 **안개**를 연상하면 좋을 것이다.

유리 : 유리를 뚫고 갈 수 없지만, 건너편을 볼 수 있다.

- Visibility = Collision::Ignore
- Camera = Collision::Block

안개 : 안개를 뚫고 갈 수 있지만, 건너편을 볼 수 없다.

- Visibility = Collision::Block
- Camera = Collision::Ignore

(Camera 채널은 시각Collision이 아닌, CameraComponent라는 오브젝트와 다른 오브젝트간의 콜리전이라 보는게 좋을 것 같다.)

Visibility vs Camera trace channels in UE4 confusion
When you perform a collision test in UE4 e.g. via line trace or sphere overlap there are two built in trace channels: Visibility and Camera...
https://zompidev.blogspot.com/2021/08/visibility-vs-camera-trace-channels-in.html

콜리전 채널의 용도

오브젝트에 콜리전 채널을 설정해준다면, **ObjectType** → **CollisionEnabled**를 통해 해당 컴포넌트에서 물리 기능을 어떻게 사용할지 지정할 수 있다.

Query	두 오브젝트의 충돌 영역이 겹치는지 테스트하는 설정.
Physics	물리적 시뮬레이션을 사용할 때 설정한다.
Query and Physics	위 두 기능을 모두 사용

💡 기본적으로 충돌로 인한 이벤트 트리거 형식으로 사용하고 싶다면 **Query**를, 래그돌과 같이 물리 시뮬레이션이 필요하다면 **Physics**를 사용하면 되겠다!

다른 콜리전 채널과의 반응

설정된 콜리전 채널이 상대방 컴포넌트의 콜리전 채널과 어떻게 반응을 할 지 지정하면 콜리전채널의 설정이 전부 끝난다.

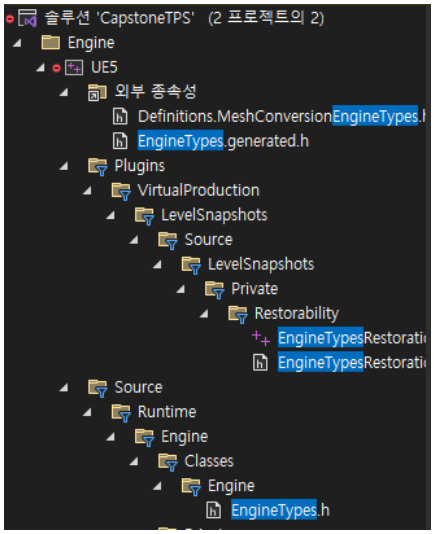
반응은 3가지로 지정할 수 있다.

Ignore	콜리전이 있어도 충돌이 일어나지 않는다.
Overlap	Ignore과 동일하게 충돌이 일어나진 않지만, Overlap 이벤트를 발생시킨다.
Block	충돌이 일어나 뚫고 갈 수 없다.

트레이스 채널의 활용

제공되는 혹은 새롭게 생성한 콜리전 채널을 C++코드 내에서 사용하려면 약간의 복잡함이 있다.

생성된 커스텀 콜리전 채널은 엔진 내의 EngineTypes.h 파일 내의 GameTraceChannel 1~18번 중 하나를 배정받는다. 어떤 값을 배정받았는지는 프로젝트 Config → DefaultEngine.ini 내에서 확인 후 코드에서 사용하면 된다.



EngineTypes.h 파일 경로 (여기가… 어딥니까..?)

```
[/Script/Engine.CollisionProfile]
...
+DefaultChannelResponses=(Channel=ECC_GameT raceChannell,Name="ABCh
DefaultResponse=ECR_Block,
bTraceType=False,bStaticObject=False)
+DefaultChannelResponses=(Channel=ECC_GameT raceChannel2,Name="Atta
DefaultResponse=ECR_Ignore,
bTraceType=True,bStaticObject=False)
...
```

교재 같은 경우는 Trace채널을 통해 공격을 구현했다. AttackCheck() 함수 호출시, 캐릭터 로케이션 전방으로 Sweep트레이스를 호출하고 Block이 발생한다면 해당 객체를 공격하는 방식이다.

```
ABCharacter.cpp
void AABCharacter::PostInitializeComponents()
{...
  ABAnim->OnAttackHitCheck.AddUObject(this, &AABCharacter::AttackCheck);
//AnimNotify와 연결한 것으로 추정
}
...
```

```
void AABCharacter:; AttackCheck()
{
  FHitResult HitResult;
  FCollistonQueryParams Params(NAME_None, false, this);
  bool bResult = GetWorld()->SweepSingleByChannel(
  HitResult,
  GetActorLocation(),
  GetActorLocation() + GetActorForwardVectorf) * 200.f,
  FQuat::Identity, //탐색에 사용될 회전
  ECollisionChannel::ECC_GameTraceChannel2, //Trace채널 사용
  FCollisionShape::MakeSphere(50.0f),
  Params);

}
```

`AddDynamic` 은 봤는데 `AddUObejct` 는 뭐지?
⇒ **UObject**기반의 델리게이트로 오브젝트에 대한 약(Weak) 레퍼런스를 유지한다.
(전투에 의해 소멸이 발생할 확률이 높고, 그럴때 오브젝트 참조여부로 인해 가비지 컬렉터가 작동하지 않을 수 있어 약 포인터 방식을 사용한것이다!)

> 멀티캐스트 델리게이트
> 여러 함수에 바인딩시켜 동시에 실행시킬수 있는 델리게이트 입니다.
> Ⓤ https://docs.unrealengine.com/4.26/ko/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/Delegates/Multicast/

▼ 이런방법으로 공격할 수도 있다!

무기에 **활성화 되지 않은 Collision박스**들을 생성하고, **Anim Notify**를 통해 해당 박스를 순간적으로 활성화시켜, Overlap이벤트가 발생하면 상대에게 데미지를 가하는 식으로 작성할 수도 있다.

디버그 드로잉

DrawDebugHelpers.h를 추가해 그리기 함수들을 선언해 디버그 도형들을 그릴 수 있다.

데미지 프레임워크

앞선 내용들을 통해 공격 영역을 설정하고, 공격 대상을 감지하는 부분까지 구현할 수 있었다. 이제 공격 행동을 완성하려면 감지된 액터에 데미지를 전달해야 한다.

다행히 언리얼엔진은 이미 **데미지 프레임워크**를 제공하고 있기에, 이를 사용하면 여러 기능을 간편하게 처리할 수 있다.

ex) 타입 FDamageEvent, 함수 TakeDamage()

이 부분은 데미지 프레임워크에 대한 설명이라기보다는 데미지를 전달하고 이로 인해 사망한 캐릭터의 애니메이션 처리에 관한 부분이 많다 생각하여 정리를 마치겠습니다.