17_ 콜리전

<제목 차례>

인천대학교 컴퓨터공학부 박종승

1. 개요

콜리전(collision; 충돌)은 상호작용 구현의 가장 기본적인 방법이다. 콜리전을 다루기 위한 방법으로 트러거 액터를 사용할 수도 있고 또는 커스텀 액터 내에 콜리전 컴포넌트를 추가하여 사용할 수도 있다. 액터나 컴포넌트의 콜리전 발생을 처리하기 위해서는 액터의 이벤트 **OnActorBegin(End)Overlap** 이나 컴포넌트의 이벤트 **OnComponentBegin(End)Overlap**을 추가하여 이벤트 그래프를 작성하면 된다.

레벨에 배치된 많은 액터들에 대해서 콜리전을 실제로 어떻게 구현할지에 대해서 생각해보자. 레벨에 다양한 종류의 데이터가 복합적으로 배치되어 있다. 높이맵으로 구축한 랜드스케이프, 레벨 에디터에서 구축한 브러시, 외부에서 가져와서 배치한 메시, 움직이는 객체 등 다양하다. 이러한 다양한 종류의 데이터에 대해서 모든 상호 관계를 고려하여 콜리전에 적용되어야 하므로 구현이 매우 복잡해진다.
각 객체에 대해서 다른 객체와의 충돌 시에 블록으로 처리할 지 또는 겹침으로 처리할 지 또는 무시할 지의 콜리전 반응을 지정해주어야 한다. 배치된 모든 객체 관계에 대하여 이러한 상호 콜리전 반응 형태를 지정하는 일에도 많은 시간이 걸린다. 이러한 경우를 모두 고려하여 레벨에 배치된 모든 데이터에 대하여 정확히 동작하는 콜리전 반응을 구현하는 것은 매우 어려운 일이고 또한 필요한 계산량도 매우 크다. 성능을 고려하면 이보다는 유연하고 간략화된 더 효율적인 콜리전 처리 방법이 필요하다.
UE에서는 효율적인 콜리전 처리를 위하여 오브젝트 채널 개념을 도입하였다. 각 객체마다 콜리전 반응을 명시하는 대신에 성질이 동일한 여러 객체를 그룹으로 묶고 그룹 단위로 콜리전 반응을 명시하도록 한다. 이러한 그룹을 오브젝트 채널이라고 한다. 이제부터 UE에서의 콜리전 처리 방법에 대해서 자세히 학습해보자.

먼저, 콜리전 반응에 대해서 알아보자.
두 물체가 콜리전하게 되면 서로 막혀서 통과하지 못할 수도 있고, 상대 물체가 없는 것처럼 그대로 뚫고 지나갈 수도 있다. 엔진에서는 콜리전 후의 반응에 따라서 **무시**(Ignore), **겹침**(Overlap,오버랩), **블록**(Block, 막음)의 세 가지로 두고 있다. **블록**은 두 물체 간의 모든 상호 침투나 겹침이 금지된다. **무시**나 **겹침**은 서로를 그대로 관통하여 통과한다.
무시나 **겹침**은 모두 관통하는 것은 같지만 통과할 때에 통지 여부에 따라서 달라진다. **무시**는 아무 통지 없이 통과하지만 **겹침**은 통지를 하면서 통과한다. 마치 출입문에 센서가 있는 것처럼 감지가 필요한 경우에는 겹침을 통지하도록 **겹침**을 사용하면 된다. 겹침이 시작할 때에 **겹침 시작** 이벤트를 통지하고 겹침이 종료될 때에 **겹침 끝** 이벤트를 통지한다.
두 객체의 콜리전 시에 두 객체의 콜리전 반응이 서로 다르게 지정된 경우에는 **무시**, **겹침**, **블록**의 우선순위로 결정된다. 예를 들어, 한 객체는 **무시**로 되어 있고 다른 객체는 **겹침**으로 되어 있다면 콜리전 반응은 **무시**로 결정된다. 정리하면, 둘 중 하나라도 콜리전 반응이 **무시**로 명시되어 있으면 항상 **무시**로 결정된다. 그 외의 경우에 대해서, 둘 중 하나로도 **겹침**이면 **겹침**으로 결정된다. 따라서 둘 다 모두 **블록**인 경우에만 **블록**으로 결정된다.

이제, **오브젝트 채널**에 대해서 알아보자.
한 객체가 항상 하나의 콜리전 반응으로 행동하도록 지정하면 편리하겠지만 현실에서는 대상 객체에 따라서 콜리전 반응이 달라질 수 있다. 예를 들어서 방충망이 있다면 총알은 대해서는 관통이

가능하지만 새에 대해서는 관통이 불가능하다. 한편, 레벨에 배치된 각 액터나 컴포넌트에 대해서 서로의 콜리전 반응을 모두 일일이 명시하는 것은 거의 불가능한 일이다. 이를 해결하기 위해서 게임 엔진에서는 콜리전 반응이 동일한 여러 객체를 하나의 그룹으로 묶어서 그룹 단위로 고려한다. 동일한 콜리전 반응을 가지는 객체 그룹을 **오브젝트 채널**(object channel)이라고 한다.
엔진은 디폴트로 6개의 **오브젝트 채널**은 제공한다. 이는 **WorldStatic**, **WorldDynamic**, **Pawn**, **PhysicsBody**, **Vehicle**, **Destructible**이다. 개발자는 자신의 새로운 커스텀 **오브젝트 채널**을 프로젝트 세팅에서 추가할 수 있다.

<참고> 엔진은 디폴트로 제공하는 6개의 **오브젝트 채널**인 **WorldStatic**, **WorldDynamic**, **Pawn**, **PhysicsBody**, **Vehicle**, **Destructible**에 대해서 알아보자.
먼저, **WorldStatic** 채널은 레벨에 배치하여 움직이지 않는 정적인 배경으로 사용되는 스태틱 메시나 브러시 객체가 사용한다.
그다음, **WorldDynamic** 채널은 모빌리티 설정이 무버블로 되어있어 움직임이 가능한 스태틱 메시나 액터가 사용한다.
그다음, **Pawn** 채널은 플레이어 또는 폰이 사용한다.
그다음, **PhysicsBody** 채널은 물리 엔진이 움직임을 시뮬레이션하는 강체가 사용한다.
그다음, **Vehicle** 채널은 Pawn이 탑승하는 이동 기구에 사용한다.
그다음, **Destructible** 채널은 물리 엔진이 파괴할 수 있는 객체에 사용한다.
어디에서 속하지 않는 객체인 경우에는 **WorldDynamic** 채널을 사용하면 된다.

이제, **콜리전 프리셋**에 대해서 알아보자.
에디터에서 각 **오브젝트 채널** 간의 충돌 반응 형태를 미리 표 형태로 표현해둔다. 이를 **콜리전 프리셋**(collision preset)이라고 한다. 또는 **콜리전 프로파일**(collision profile)이라고도 한다. 이제 우리는 레벨에 배치된 각 액터 인스턴스에 대해서는 해당하는 **콜리전 프리셋**만 지정하면 된다. 엔진은 디폴트로 18개의 콜리전 프리셋을 제공한다. 우리는 우리의 커스텀 콜리전 프리셋을 만들어 추가할 수 있다.

용어에 대해서 정리해보자. 여러 용어가 혼합 사용되고 있어서 혼란스러울 때가 있다. 우리는 혼란을 줄이기 위해서 다음과 같이 정리하자.
먼저, **콜리전**(collision)은 **충돌**로 언급하기도 한다. 충돌은 물리적인 충돌뿐만 아니라 겹침 충돌을 포함하는 개념으로 포괄적으로 사용된다.
그다음, **오브젝트 채널**(object channel)은 **오브젝트 유형**(object type), 객체 채널로 언급하기도 한다. 특히 **오브젝트 유형**이 흔히 등장한다.
그다음, **트레이스 채널**(trace channel)은 **트레이스 유형**으로 언급하기도 한다.
그다음, **콜리전 프리셋**(collision preset)은 **콜리전 프로파일**로 언급하기도 한다.
우리는 앞으로 가급적이면 일관적으로 **콜리전**, **오브젝트 채널**, **트레이스 채널**의 용어로 언급할 것이다.

이제부터 실습을 진행하면서 콜리전에 대해서 학습해보자.

<참고> 콜리전에 대한 설명 문서로 다음을 참조하자.
https://www.unrealengine.com/ko/blog/collision-filtering

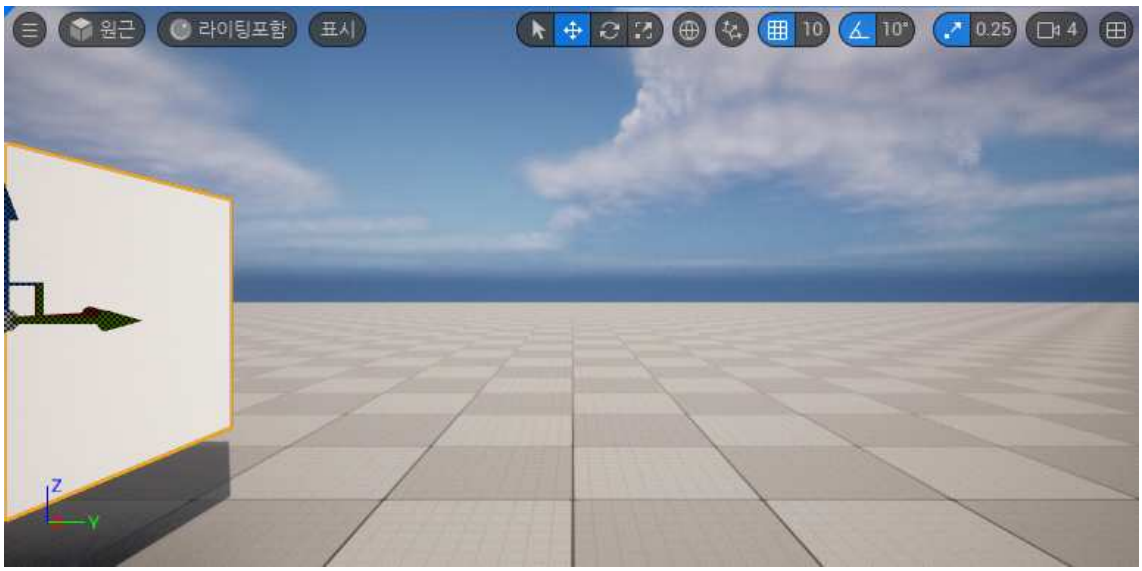
2. 콜리전 프리셋으로 콜리전 반응 명시하기

이 절에서는 콜리전 프리셋으로 콜리전 반응을 명시하는 방법에 대해서 학습한다. 콜리전 프리셋을 명시하여 콜리전 시에 관통하지 못하도록 하는 방법을 배워본다. 또한, 콜리전 시에 무시하고 관통하도록 하는 방법도 배워본다.

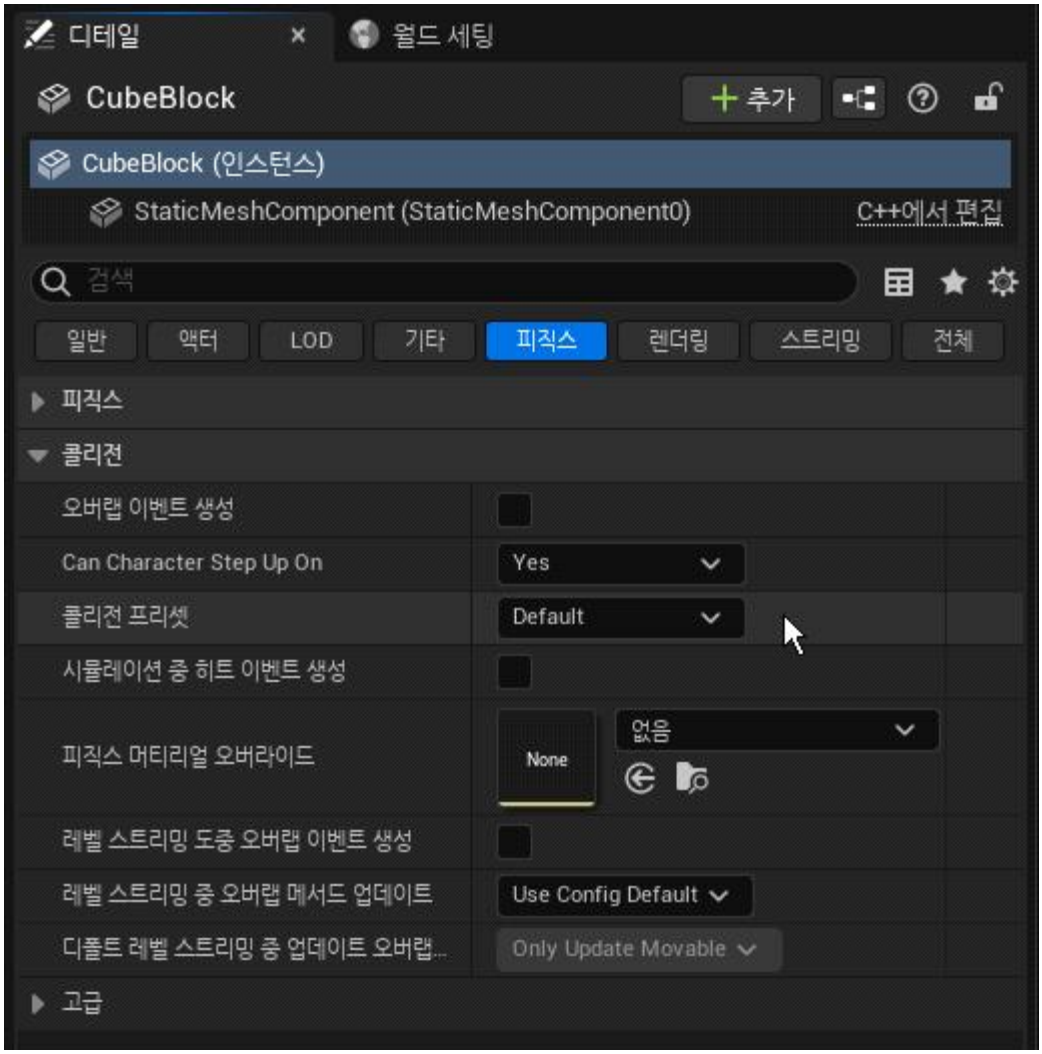
---

1. 새 프로젝트 Pcollisioncube를 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 Pcollisioncube로 입력하고 **생성** 버튼을 클릭하자. 프로젝트가 생성되고 언리얼 에디터 창이 뜰 것이다.
창이 뜨면, 메뉴바에서 **파일** » **새 레벨**을 선택하고 **Basic**을 선택하여 기본 템플릿 레벨을 생성하자. 그리고, 레벨 에디터 툴바의 저장 버튼을 클릭하여 현재의 레벨을 MyMap으로 저장하자.
그리고, **프로젝트 세팅** 창에서 **맵&모드** 탭에서의 **에디터 시작 맵** 속성값을 MyMap으로 수정하자.
그리고, 툴바의 **액터 배치** 아이콘을 클릭하고 **기본** 아래의 **플레이어 스타트** 액터를 드래그하여 레벨에 배치하자. 위치를 (0,0,112)로 지정하자.

2. 툴바의 **액터 배치** 아이콘을 클릭하고 **셰이프** 아래의 **큐브**를 드래그하여 레벨에 배치하자. 배치된 액터의 인스턴스 이름은 **CubeBlock**으로 수정하자. 위치는 (200,-200,100)에 배치하자.
플레이해보자. 플레이어를 움직여서 큐브에 부딪혀보자. 큐브에 닿은 후에는 블록되어 더 이상 전진하지 못할 것이다.



3. 레벨에 배치된 **CubeBlock** 액터를 선택하고 **디테일** 탭을 살펴보자.
콜리전 영역을 찾아보자. **콜리전** 영역이 보이지 않으면 **피직스** 탭을 클릭하자.
콜리전 영역에 **콜리전 프리셋** 속성이 있다. 이 속성이 바로 콜리전 반응을 명시하는 속성이다.

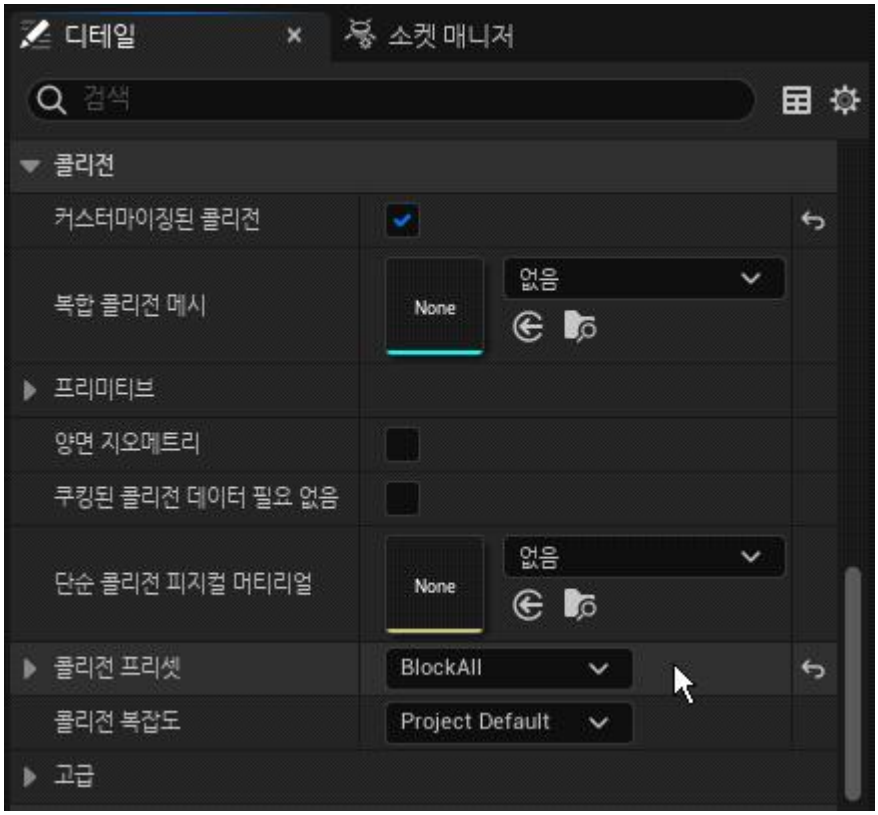


이제부터 **콜리전 프리셋** 속성에 대해서 자세히 알아보자.

컴포넌트의 **디테일 탭에서 콜리전 프리셋** 속성에 지정되어 있는 프리셋이 이 컴포넌트의 모든 콜리전을 결정한다. 엔진에서는 디폴트로 18개의 **콜리전 프리셋**을 제공한다. 개발자는 자신의 새로운 **콜리전 프리셋**을 **프로젝트 세팅**에서 추가할 수도 있다.

콜리전 프리셋의 오른쪽 입력 상자를 클릭하여 다른 것을 선택할 수 있다. 클릭하면 **Default**, **Custom..**, 18개의 엔진 제공 프리셋, 개발자가 추가한 프리셋이 나열되면 이 중에서 하나를 선택하면 된다. **콜리전 프리셋**이 **Default**로 되어 있는 것은, 스태틱 메시 애셋에 지정되어 있는 디폴트 **콜리전 프리셋**을 사용하겠다는 의미이다. 디폴트 **콜리전 프리셋**은 다른 모든 **오브젝트 채널**과 콜리전 반응이 **Block**으로 처리되도록 설정되어 있다.

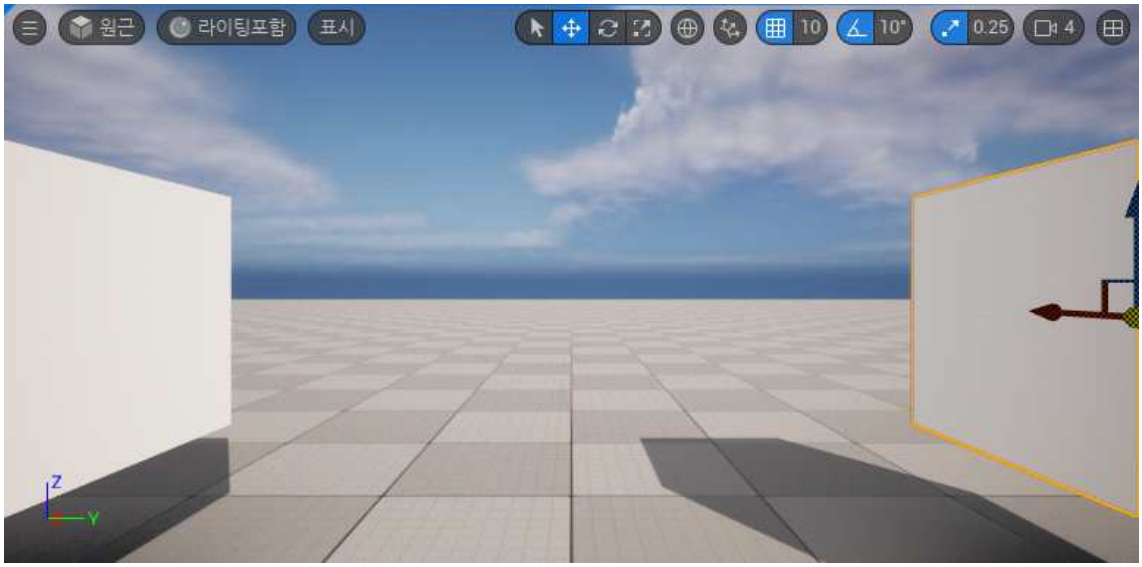
4. 스태틱 메시 애셋에 지정되어 있는 디폴트 **콜리전 프리셋**이 어떻게 되어 있는지 확인해보자. **아웃라이너**에서 **CubeBlock** 위에서 우클릭하고 팝업메뉴창에서 **Cube 편집**을 선택하자. **스태틱 메시 에디터** 창이 열린다. **Ctrl+E**를 클릭해도 바로 창이 열린다. **스태틱 메시 에디터** 창의 **디테일** 탭에서 **콜리전** 영역의 **콜리전 프리셋** 속성을 찾아보자.



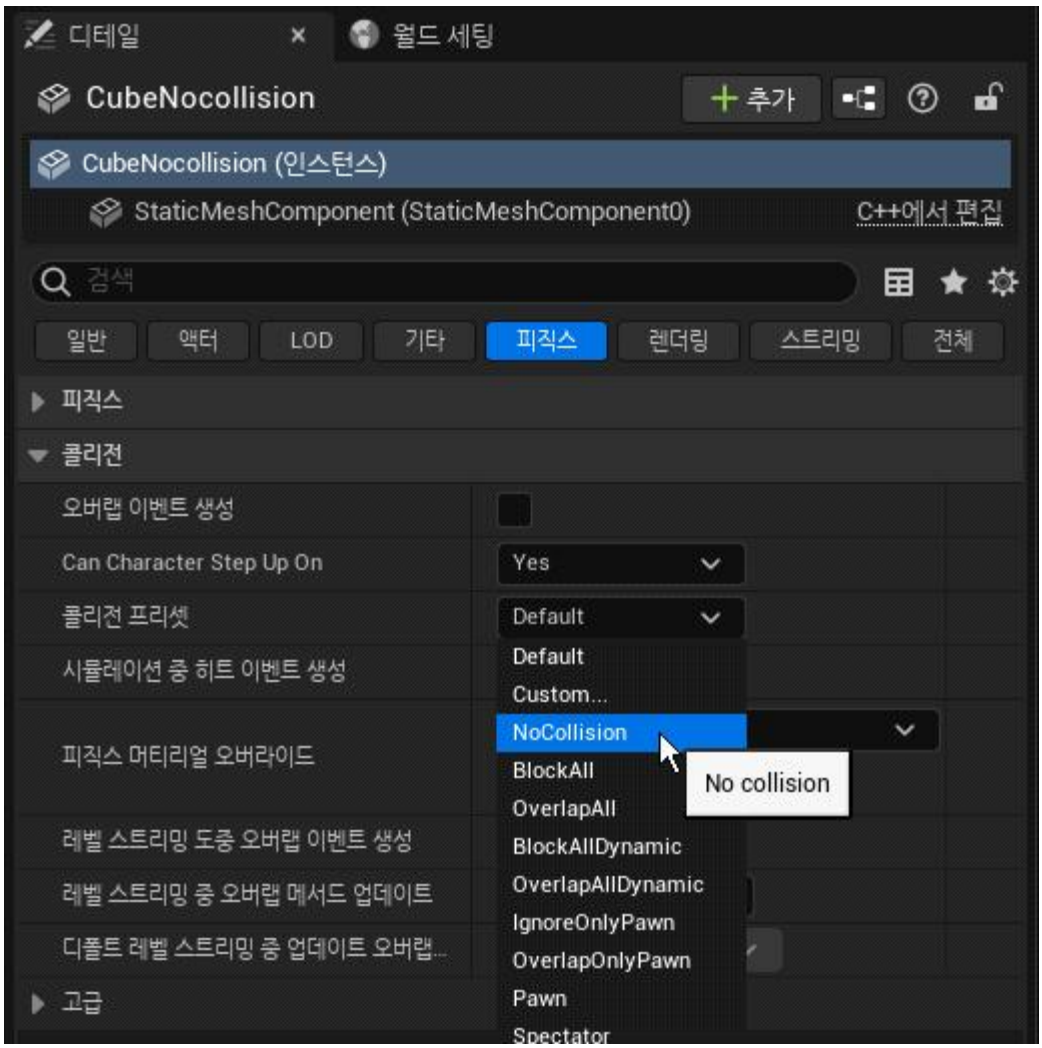
스태틱 메시 애셋에는 **콜리전 프리셋**에 **BlockAll**이 지정되어 있다. 이 프리셋은 모든 다른 **오브젝트 채널**과 **Block**하는 것으로 설정된 프리셋이다. 따라서 **Default**로 두면 다른 모든 **오브젝트 채널**과 콜리전 반응이 **Block**으로 처리될 것이다.
플레이어가 큐브를 관통하지 못하는 이유가 바로 이처럼 다른 모든 **오브젝트 채널**과의 콜리전 반응이 **Block**으로 되어 있기 때문이다.

5. 콜리전 반응에 대해서 더 자세히 알아보자. **CubeBlock**의 콜리전 반응만 **블록**이라고 해서 **블록**으로 반응하는 것이 아니라, **CubeBlock**와 충돌하는 객체의 콜리전 반응도 **블록**이어야 **블록**으로 반응하게 된다. 따라서 플레이어 폰에 명시된 콜리전 반응도 확인해 보아야 한다.
플레이어 폰은 **DefaultPawn** 클래스의 인스턴스이다. **DefaultPawn** 액터는 미리 배치되어 있지 않고 플레이 시에 생성되는 액터이므로 플레이해보자. 게임을 플레이한 후에 **아웃라이너**를 살펴보면 현재 생성된 액터의 목록이 보인다. 생성된 **DefaultPawn** 인스턴스를 클릭하고 **디테일** 탭의 **콜리전** 영역에서 **CollisionComponent**의 **콜리전 프리셋** 속성을 살펴보면 **Pawn**으로 지정되어 있음을 알 수 있다. **Pawn** 프리셋은 모든 **오브젝트 채널**에 대해서 **블록**으로 콜리전 반응하도록 설정되어 있다. 따라서 **CubeBlock**와 플레이어 폰의 콜리전 반응이 모두 **블록**이므로 두 객체 간의 콜리전 반응은 **블록**으로 결정된다.

6. 이제, 충돌 시에 상대 객체가 없는 것처럼 무시하고 관통하도록 설정해보자.
먼저 상자 메시를 배치하자. 툴바의 **액터 배치** 아이콘을 클릭하고 **셰이프** 아래의 **큐브**를 드래그해서 레벨에 배치하자. 배치 후 인스턴스 이름은 **CubeNocollision**으로 수정하자. 위치는 (200,200,100)에 배치하자.



7. 추가된 인스턴스가 선택된 상태에서 **디테일** 탭을 살펴보자. **콜리전** 카테고리에 있는 **콜리전 프리셋** 속성이 이전과 같이 **Default**로 되어 있을 것이다. 이것을 **NoCollision**으로 수정하자.



8. 플레이해보자. 플레이어를 움직여서 배치된 **CubeNocollision**로 향하여 이동해보자. **CubeNocollision**에 닿은 후도 계속해서 상자를 통과하여 전진할 것이다. 이는 **CubeNocollision**가 충돌 없음으로 설정되었기 때문이다.

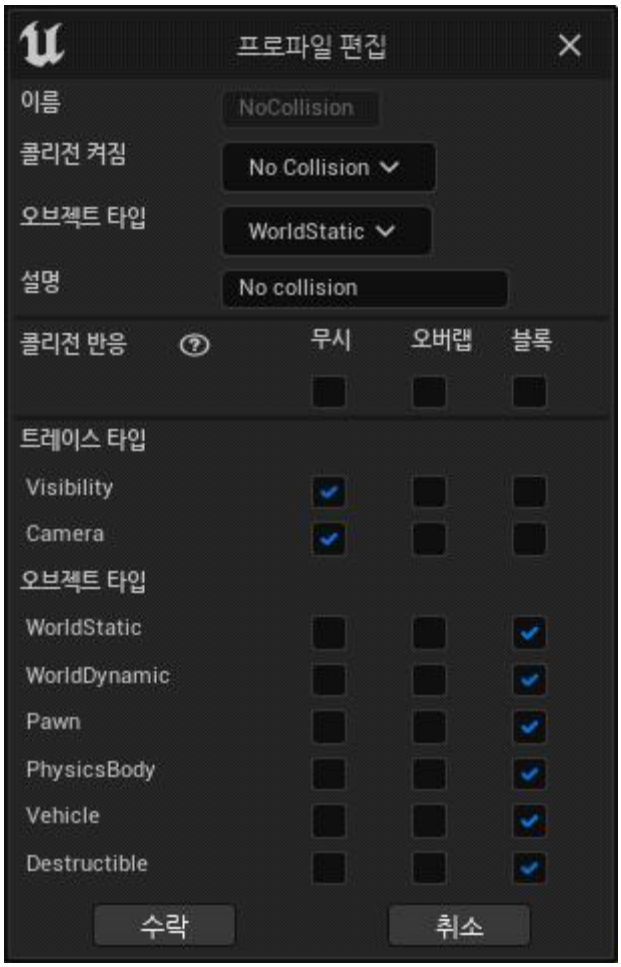
9. 이제부터, **콜리전 프리셋**의 구조에 대해서 살펴보자.
프로젝트 세팅 창에서 왼쪽 탭에서 **엔진** » **콜리전**을 클릭하면, 오른쪽 탭에서 **Object Channels**, **Trace Channels**, **Preset**의 세 목록이 표시된다. 이 중에서 **Preset** 아래에는 시스템에서 디폴트로 제공하는 **콜리전 프리셋**이 표시된다.

엔진에서는 디폴트로 6개의 오브젝트 유형을 제공한다. 이들은 **WorldStatic**, **WorldDynamic**, **Pawn**, **PhysicsBody**, **Vehicle**, **Destructible**이다.
그리고 2개의 트레이스 유형을 제공한다. 이들은 **Visibility**와 **Camera**이다.

그리고 이들과 관련된 18개의 **콜리전 프리셋**을 제공한다.

<참고> 엔진에서 지원하는 18개의 **콜리전 프리셋**을 적절하게 사용하면 매우 편리하다. 예를 들어보자.
박스 트리거 액터나 박스 콜리전 컴포넌트 등과 같이 겹침 이벤트를 목적으로 하는 객체는 **OverlapAll** 프리셋을 사용한다.
그리고, 다른 객체를 블록하는데 사용되는 투명 볼륨인 블로킹 볼륨 객체는 보이지 않는 벽으로 처리하는 **InvisibleWall** 프리셋을 사용한다.
그리고, 스태틱 매시 액터를 레벨에 배치하면 디폴트로 **모빌리티**가 **스태틱**으로 배치된다. 이 경우 콜리전 프리셋이 **BlockAll**로 지정된다. 나중에 **모빌리티**를 **무버블**로 바꾸게 되면 프리셋도 자동으로 **BlockAllDynamic**으로 바꾸어준다.
그리그, 스태틱 매시 액터에서 **SimulatePhysics**에 체크하여 물리 엔진이 적용되도록 하면 **SimulatePhysics** 프리셋으로 바꾸어준다.
이러한 방법의 프리셋 활용을 통해서 콜리전 설정을 매우 간편하게 바꿀 수 있다.

10. **콜리전 프리셋** 중에서 **NoCollision**을 더블클릭하자. **프로파일 편집** 창이 뜰 것이다.



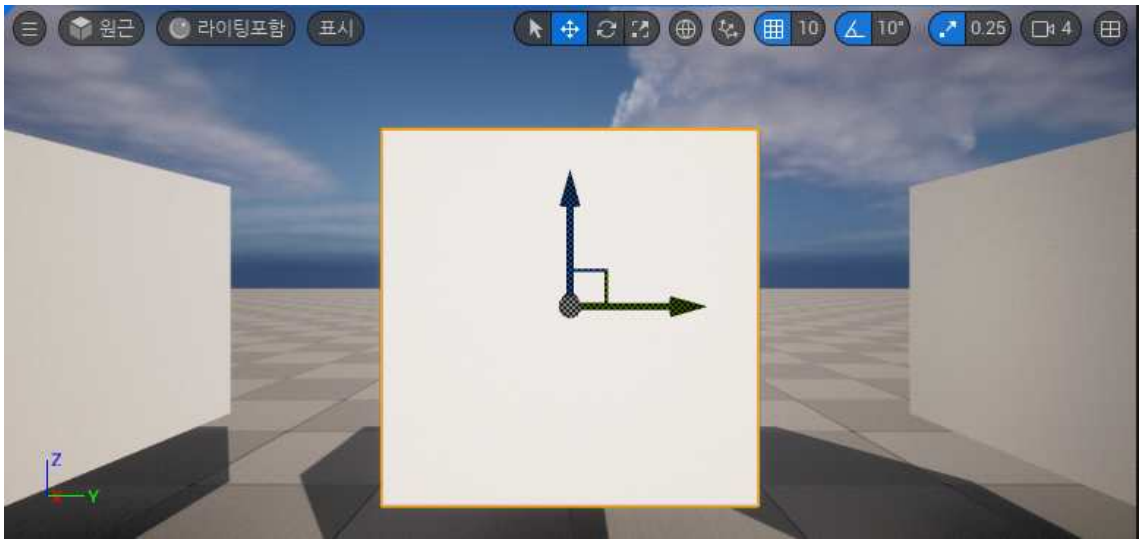
창에서 **콜리전 켜짐** 속성이 있다. 속성값이 **No Collision**으로 되어 있다. 이는 모든 콜리전을 무시하겠다는 것이다. 따라서 모든 충돌이 무시되므로 그대로 관통하게 되고 통지도 일어나지 않는다.
우리는 **CubeNocollision**의 **콜리전 프리셋**에 모든 콜리전을 무시하는 **콜리전 프리셋**인 **NoCollision**을 지정하여 충돌 자체가 무시되도록 설정하였다.

11. **콜리전 프리셋**의 프로파일 편집 창에 대해서 더 자세히 알아보자.

프로파일 편집 창에는 프로파일 이름 다음에 **콜리전 켜짐**, **오브젝트 유형**(**오브젝트 채널**을 의미함), **설명**, **콜리전 반응**의 네 속성이 순서대로 나열되어 있다. 각 속성에 대해서 조금 더 자세히 알아보자.
첫번째, **콜리전 켜짐**은 **No Collision**, **Query Only (No Physics Collision)**, **Physics Only (No Query Collision)**, **Collision Enabled (Query and Physics)**의 네 개 중에서 하나를 선택한다. 콜리전은 쿼리 타입의 콜리전과 피직스 타입의 콜리전의 두 부류로 나누어진다. 피직스 타입의 콜리전은 물리 시뮬레이션으로 인한 움직임에 대한 콜리전을 의미한다. 쿼리 타입의 콜리전은 물리적인 움직임이 아닌 인위적인 제어에 의한 움직임에 대한 콜리전을 의미한다. 예를 들어, 무엇이 존재하는 지를 확인하기 위해 직선을 쏘아서 충돌하는 지를 검사하거나 객체가 겹치는 지를 확인하기 위해서 객체 간의 충돌을 검사하는 경우이다. **No Collision**은 모든 콜리전을 고려하지 않다. **Query Only**는 쿼리 타입의 콜리전만 고려한다. **Physics Only**는 피직스 타입의 콜리전만 고려한다. **Collision Enabled**는 두 경우를 모두 고려한다.
두번째, **오브젝트 유형**은 이 프리셋이 지정되는 객체의 **오브젝트 채널**을 명시한다. 엔진이 6개의 기본 **오브젝트 채널**과 개발자가 추가한 **오브젝트 채널** 중에서 선택하면 된다. 개발자가 추가한 **오브젝트 채널**은 **프로젝트 세팅** 창의 **엔진** » **콜리전** 탭에서 **Object Channels** 항목에 나열된다.
세번째, **설명**은 툴팁을 위한 문자열이다.
네번째 **콜리전 반응**은 이 프리셋이 지정되는 객체가 다른 **오브젝트 채널**과 어떻게 콜리전 반응할지를 표 형태로 명시한다. 가장 중요한 부분이고 모든 콜리전 반응은 이 정보로부터 결정된다. 각 줄은 각 **오브젝트 채널**에 대한 콜리전 반응을 **무시**, **겹침**, **블록** 중 하나에 선택하는 방법으로 명시한다.
참고로, **오브젝트 유형** 이외에 **트레이스 유형**(**트레이스 채널**을 의미함)이라는 것이 있는데, 이는 직접 움직여서 충돌하는 것이 아니라 시야 등을 체크하기 위한 광선을 쏘아서 충돌을 검사하는 용도의 트레이스 기법에서의 유형을 의미한다.

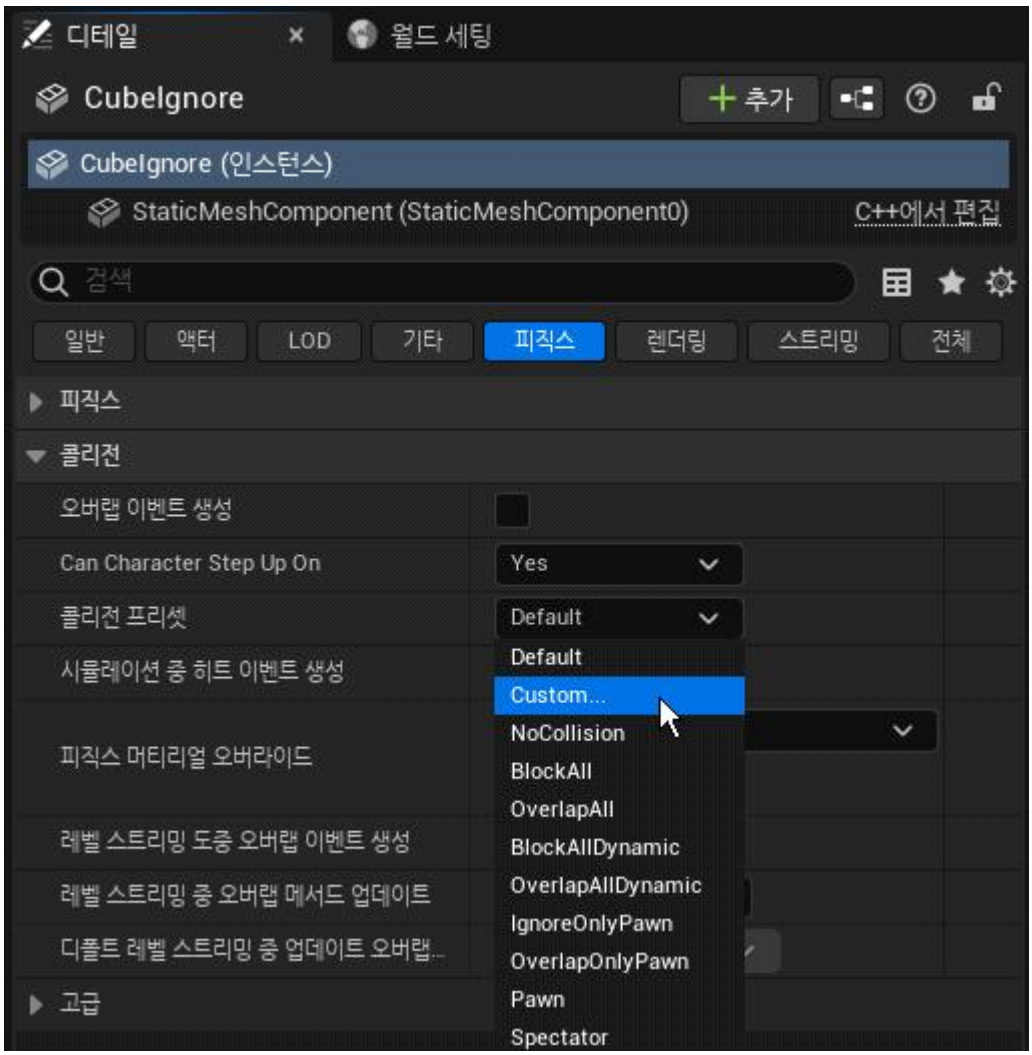
<참고> **콜리전 프리셋**의 자세한 설명은 아래의 링크를 참조하자.
https://docs.unrealengine.com/collision-response-reference-in-unreal-engine/

12. 이제, 다른 방법으로 충돌 시에 관통하도록 설정해보자.
먼저, 상자 메시를 배치하자. 툴바의 **액터 배치** 아이콘을 클릭하고 **셰이프** 아래의 **큐브**를 드래그해서 레벨에 배치하자. 배치 후 인스턴스 이름은 **CubeIgnore**로 수정하자. 위치는 (200,0,100)에 배치하자.

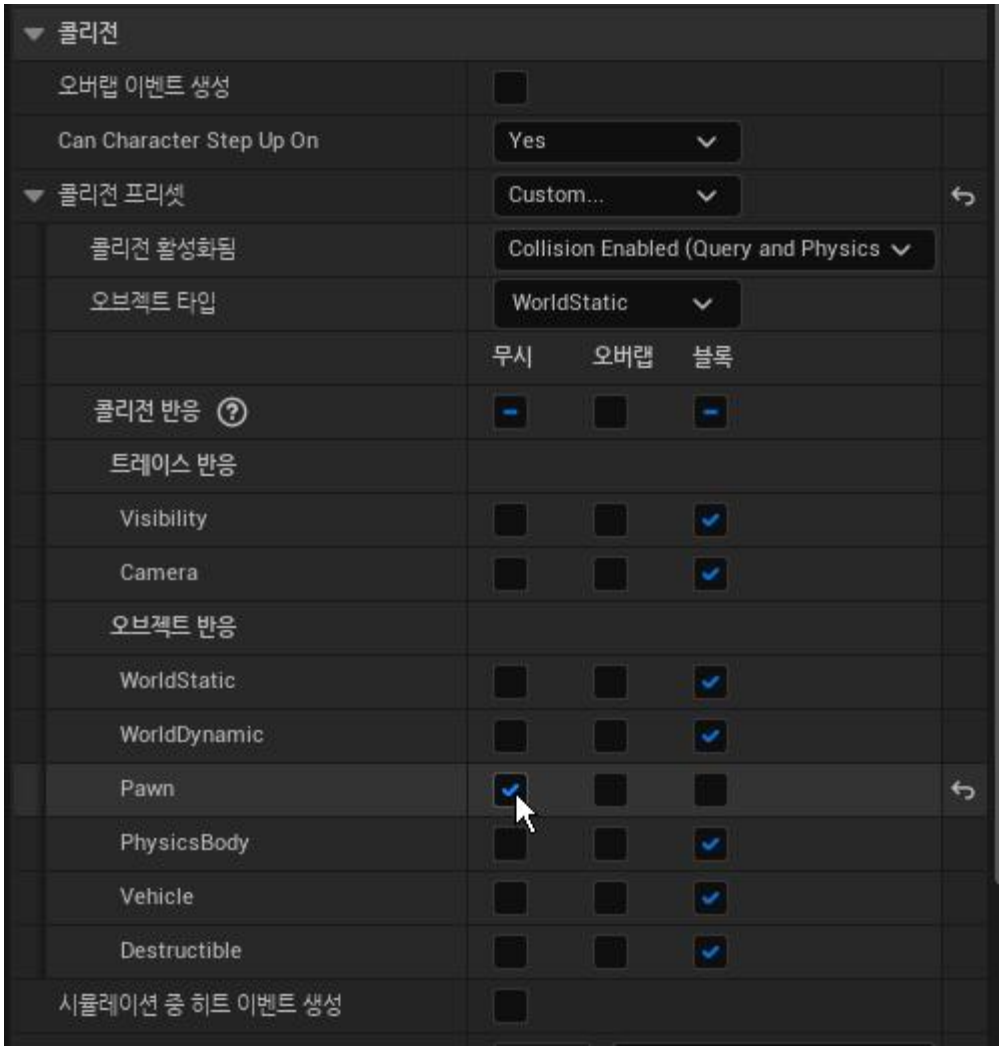


13. 이미 만들어진 **콜리전 프리셋** 중에서 원하는 것이 없다면, **Custom…**을 선택하고 이 인스턴스만을 위한 콜리전 반응을 명시할 수 있다.

CubeIgnore의 **디테일** 탭에서 **콜리전 프리셋**을 클릭하고 **Custom…**을 선택하자.



14. 그다음, **콜리전 프리셋**을 펼쳐서 내용을 살펴보자. **콜리전 반응**이 모두 **블록**으로 되어 있을 것이다.



다른 것은 그대로 두고 **Pawn**에 대해서만 **무시**에 체크하여 수정하자. 플레이어 폰과 충돌이 일어나지 않도록 지정한 것이다.

참고로, 플레이어가 키보드를 조작해서 폰이 움직이는 경우에 발생하는 충돌은 피직스 타입의 콜리전이 아니라 쿼리 타입의 콜리전에 해당한다. 따라서 **Collision Enabled** 속성은 **Collision Enabled (Query and Physics)**로 되어 있는 값을 그대로 두어도 되고 **Query Only (No Physics Collision)**로 바꾸어도 된다.

15. 플레이해보자. 플레이어를 움직여서 배치된 **CubeIgnore**로 향하여 이동해보자. **CubeIgnore**에 닿은 후도 계속해서 상자를 통과하여 전진할 것이다.

충돌하지 않고 통과하는 이유를 다시 정리해보자. 플레이어 폰에서는 **콜리전 프리셋**이 **Pawn**으로 지정되어 있고, **Pawn** 프리셋은 모든 **오브젝트 채널**에 대해서 **블록**으로 콜리전 반응하도록 설정되어 있다. 그러나 **CubeIgnore**에서는 **Pawn 오브젝트 채널**에 대해서는 **무시**로 설정하였으므로 두 객체의 콜리전 반응은 **무시**로 결정된다.

지금까지, 콜리전 프리셋으로 콜리전 반응 명시하는 방법에 대해서 학습하였다.

3. 오버랩 이벤트 사용하기

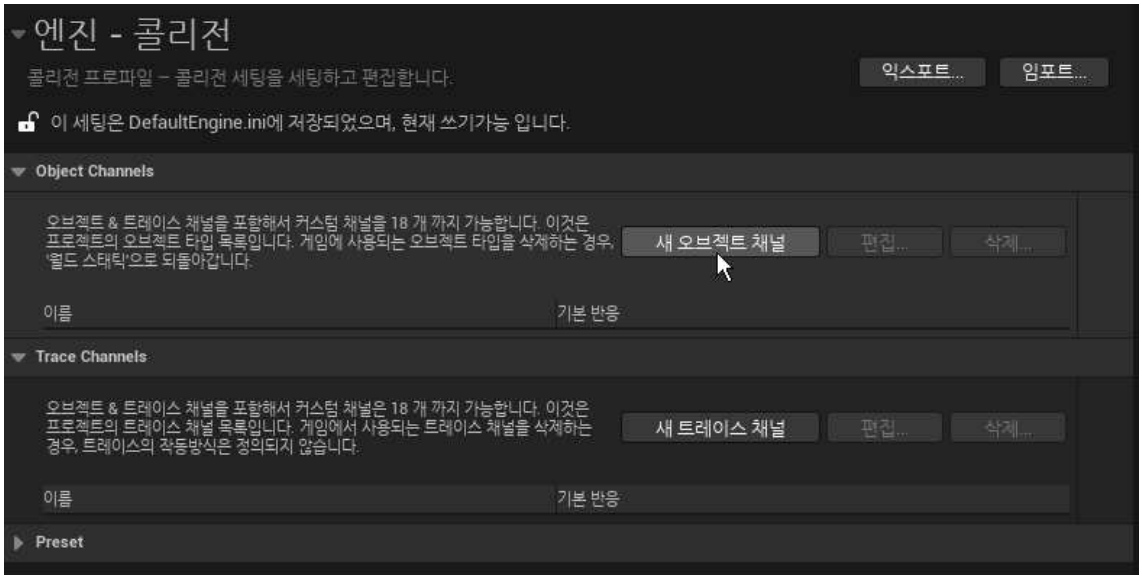
이 절에서는 오버랩 이벤트에 대해서 학습한다.
두 객체가 관통하는 경우에 물체가 겹치기 시작할 때와 겹치는 것이 종료될 때에 오버랩 이벤트가 발생되도록 한다. 이벤트가 발생했을 때에는 우리의 이벤트 그래프가 호출하도록 한다. 이 방법을 활용하면, 게임에서 픽업 아이템을 취득하는 것을 구현할 수 있다.

---

1. 새 프로젝트 **Poverlappickupitem**를 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 Poverlappickupitem로 입력하고 **생성** 버튼을 클릭하자. 프로젝트가 생성되고 언리얼 에디터 창이 뜰 것이다.
창이 뜨면, 메뉴바에서 **파일** » **새 레벨**을 선택하고 **Basic**을 선택하여 기본 템플릿 레벨을 생성하자. 그리고, 레벨 에디터 툴바의 저장 버튼을 클릭하여 현재의 레벨을 MyMap으로 저장하자.
그리고, **프로젝트 세팅** 창에서 **맵&모드** 탭에서의 **에디터 시작 맵** 속성값을 MyMap으로 수정하자.
그리고, 툴바의 **액터 배치** 아이콘을 클릭하고 **기본** 아래의 **플레이어 스타트** 액터를 드래그하여 레벨에 배치하자. 위치를 (0,0,112)로 지정하자.

2. 게임에서 흔히 등장하는 파워업 아이템을 생각해보자. 아이템은 실제 물건에 해당하는 스태틱 메시 액터로 기본적인 콜리전 반응이 **블록**일 것이다. 한편 플레이어는 아이템에 다가가서 이를 획득할 수 있어야 한다. 따라서 플레이어에 대해서는 아이템과 겹침 반응이 되도록 하여 충돌을 통보받을 수 있도록 하고 획득할 때에 아이템이 사라지도록 해야 한다.
이제부터, 이러한 픽업 아이템 액터를 위한 **오브젝트 채널**과 **콜리전 프리셋**을 만들어보자.

3. 픽업 아이템을 위한 새 **오브젝트 채널**을 만들자. **프로젝트 세팅** 창에서 왼쪽 탭에서 **엔진** » **콜리전**을 클릭하면, 오른쪽 탭에서 **Object Channels**, **Trace Channels**, **Preset**의 세 목록이 표시된다. 이 중에서 **Object Channels** 아래에는 커스텀 오브젝트 채널이 표시되고 **Trace Channels** 아래에는 커스텀 트레이스 채널이 표시된다. 우리는 **새 오브젝트 채널** 버튼을 클릭하여 픽업 아이템을 위한 커스텀 오브젝트 채널을 만들자.



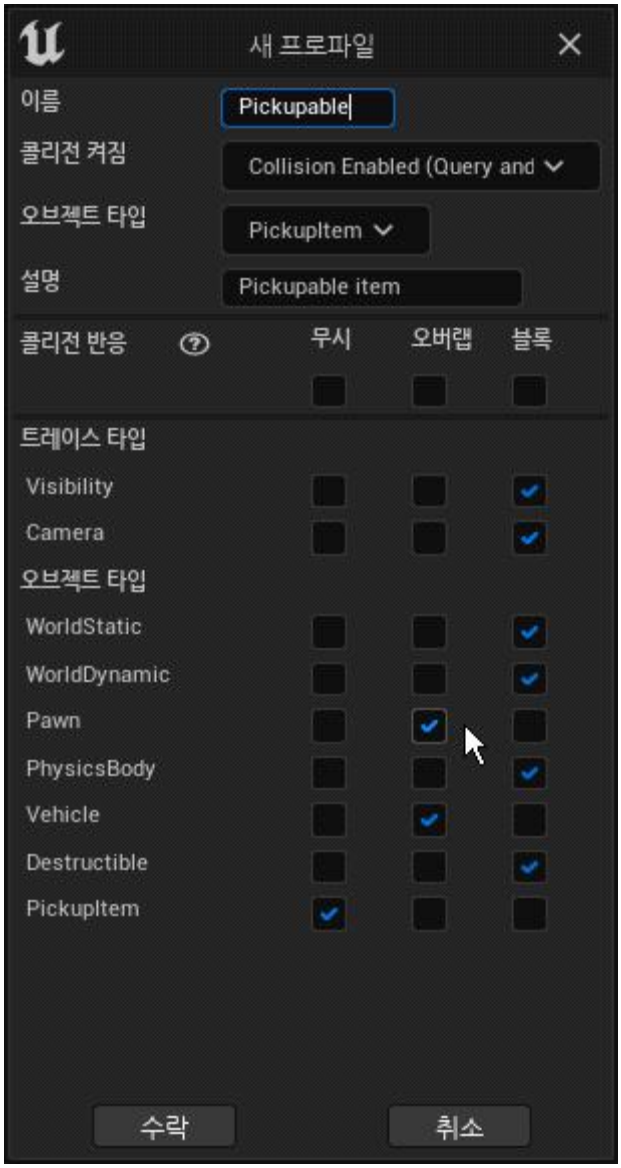
4. 새 채널 생성을 위한 창이 뜬다. **이름**에 **PickupItem**으로 입력하고 **기본 반응**에 **블록**으로 지정하자. **기본 반응**을 **블록**으로 지정하면, 기존에 있던 18개의 시스템 제공 프리셋과 이미 만들어졌거나 앞으로 새로 만들어지는 커스텀 프리셋에 대해서, 이 **PickupItem**에 대한 충돌 응답의 디폴트 값을 **블록**으로 지정한다.

수락 버튼을 클릭하자. 이제 **Object Channels** 아래에 우리가 생성한 **오브젝트 채널**인 **PickupItem**이 나열될 것이다.

<참고> 새로운 **오브젝트 채널**이 생성되면 엔진은 기존의 모든 프리셋에 대해서 새로운 **오브젝트 채널**과의 반응을 자동으로 추가해준다. 이 때에 모두 기본 반응으로 지정한 **Block**으로 추가해준다.
새로운 **오브젝트 채널**을 생성한 후에는 기존의 **프리셋**에 대해서 수정할 사항이 있는지를 전체적으로 체크하자. 우리의 예에서, **PickupItem**의 **기본 반응**으로 설정한 **블록**으로 명시되어 있을 것인데, 이것이 부적절한 **프리셋**도 있을 것이다. 예를 들어 **OverlapAll** 프리셋에서는 모두 **겹침**이 되는 것이 적절하므로 **PickupItem**에 대해서도 **겹침**으로 수정하는 것이 좋다. **Spectator** 프리셋에서는 **PickupItem**에 대해서 **무시**로 수정하는 것이 좋다. 이 과정은 실제 게임 제작 시에는 반드시 필요한 과정이지만, 우리는 그냥 넘어가도록 하자.

5. 픽업 아이템의 콜리전 반응을 표현하기 위한 새 **프리셋**을 생성하자. **Preset** 아래의 **새 프로파일…** 버튼을 클릭하자.

6. 새 **프리셋**을 생성하기 위한 창이 뜨면, **이름**에 **Pickupable**을 입력하자. **콜리전 켜짐**은 **No Collision**에서 **Collision Enabled (Query and Physics)**으로 바꾸자. **오브젝트 유형**은 **WorldStatic**에서 **PickupItem**으로 바꾸자. **설명**에는 **Pickupable item**와 같이 설명을 입력하자.
그 아래에는 **콜리전 반응**이 나열된다. 우리가 이전에서 **기본 반응**을 **Block**으로 하였기 때문에 모든 **콜리전 반응**이 모두 **블록**에 체크되어 있을 것이다.
Pawn이나 **Vehicle**에서는 픽업 아이템을 취득할 수 있도록 **겹침**으로 바꾸자. 같은 픽업 아이템끼리는 서로 충돌하지 않도록 **PickupItem**은 **무시**로 바꾸자.

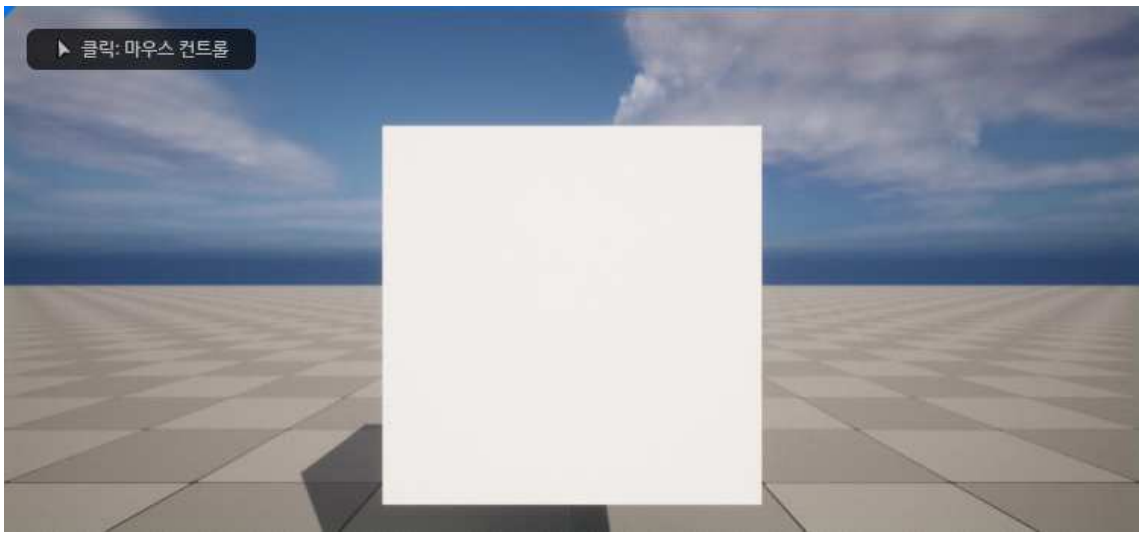


이제, **수락** 버튼을 클릭하자. 이제 **Preset** 아래에 우리가 생성한 커스컴 **프리셋**인 **Pickupable**이 나열될 것이다.
이제 모든 설정이 완료되었다.

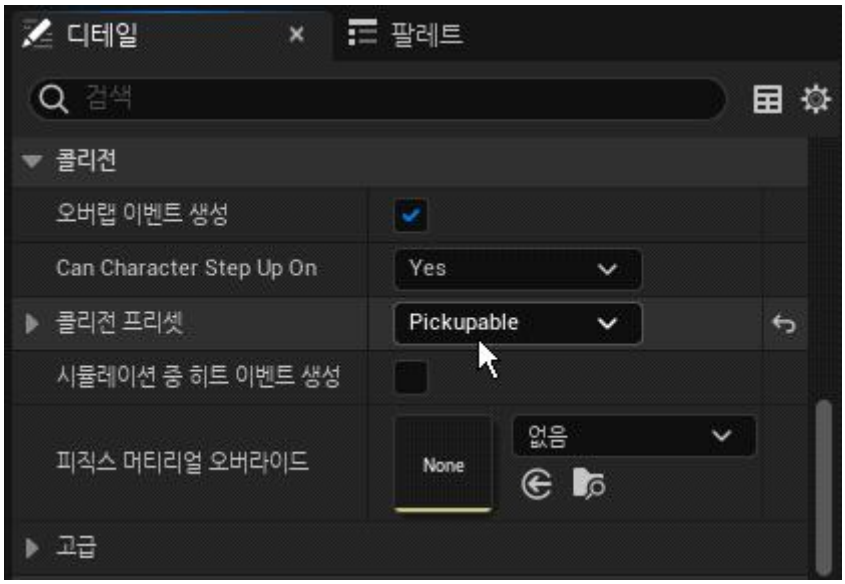
7. 이제, 픽업 아이템을 하나 만들어보자.
이전 예제에서와 같이 큐브 블루프린트 클래스를 만들어보자.
먼저, **콘텐츠 브라우저**에서 툴바에서 **+추가**를 클릭하고, 드롭다운 메뉴에서 **블루프린트 클래스**를 선택하고, **부모 클래스 선택** 창에서 **Actor**를 부모 클래스로 선택하자. 이름은 **BP_MyPickupItem**으로 수정하자.
그다음, **BP_MyPickupItem**을 더블클릭하여 **블루프린트 에디터**를 열자. **컴포넌트** 탭에서 **+추가**를 클릭하고 **큐브**를 선택하자. 큐브 스태틱 메시 컴포넌트가 추가될 것이다. 디폴트로 **Cube**라는 이름으로 그대로 두자. 컴파일하고 저장하자.

8. 레벨 에디터에서 큐브를 드래그해서 레벨에 배치하자. 위치는 (200,0,100)에 배치하자.

플레이해보자. 플레이어를 움직여서 큐브에 부딪혀보자. 이전처럼 블록될 것이다.



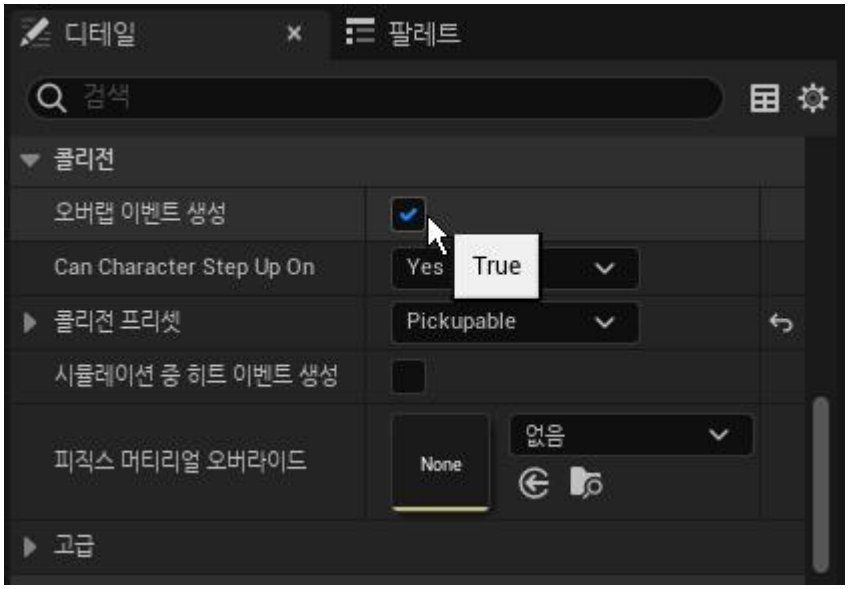
9. **BP_MyPickupItem 블루프린트 에디터**로 가자. **컴포넌트** 탭에서 **Cube** 컴포넌트를 선택하자.
이 컴포넌트의 **디테일** 탭에서 **콜리전** 영역의 **콜리전 프리셋** 속성을 찾자. 속성값이 **BlockAllDynamic**으로 되어 있을 것이다. 이것을 **Pickupable**로 수정하자.
이제 이 **MyBoxPickupItem**와 플레이어 폰과는 콜리전 반응이 **겹침**으로 된다.



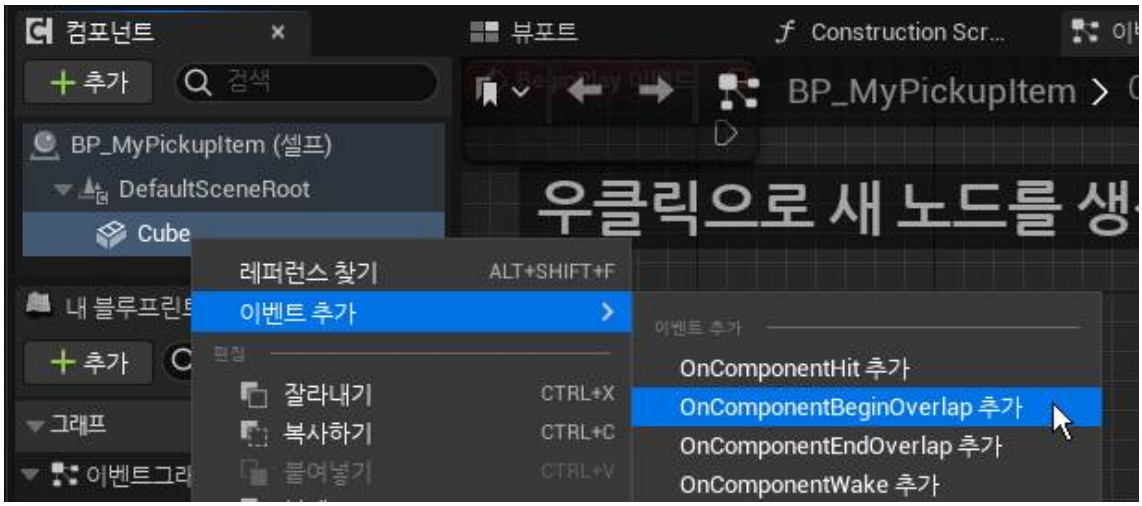
컴파일하고 저장하자.
플레이해보자. 이제는 플레이어가 다가가서 관통할 수 있을 것이다.

<참고> **Pickupable** 프리셋은 **오브젝트 유형**이 **PickupItem**이고 **Pawn**에 대해서 **겹침**으로 되어 있고, **Pawn** 프리셋은 **오브젝트 유형**이 **Pawn**이고 **PickupItem**에 대해서 **블록**으로 되어 있다. 하나는 **블록**이지만 다른 하나가 **겹침**이므로 최종 반응은 **겹침**으로 결정된다.

10. **디테일** 탭에서, **콜리전** 카테고리에 있는 **오버랩 이벤트 생성**(**Generate Overlap Events**)속성이 체크되어 있는지 확인하자. 디폴트로 체크되어 있을 것이므로 손대지 않아도 되지만, 이 속성에 체크되어 있어야만 겹침이 발생할 때에 이벤트가 통지된다는 것을 기억하자.

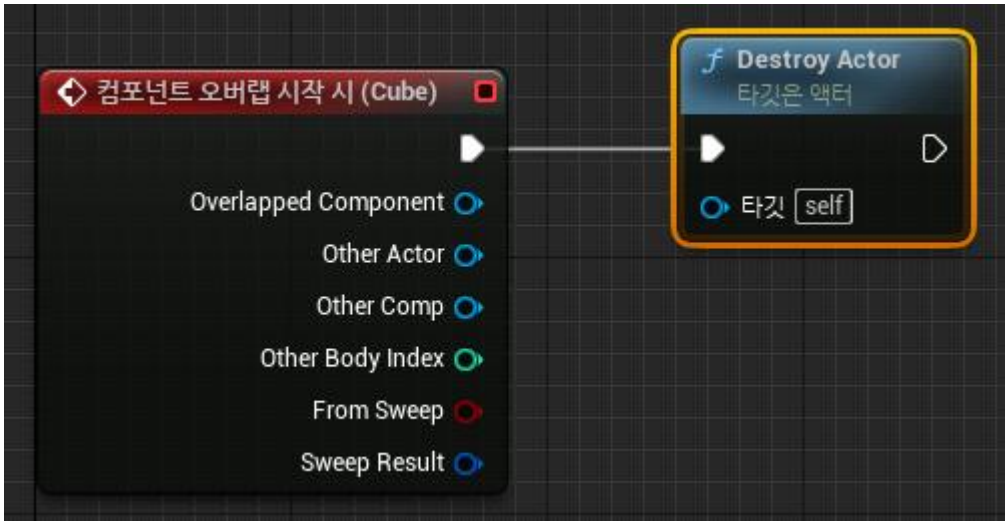


11. **컴포넌트** 탭에서 **Cube** 위에서 우클릭하고 **이벤트 추가** » **OnComponentBeginOvelap 추가**를 선택하자.



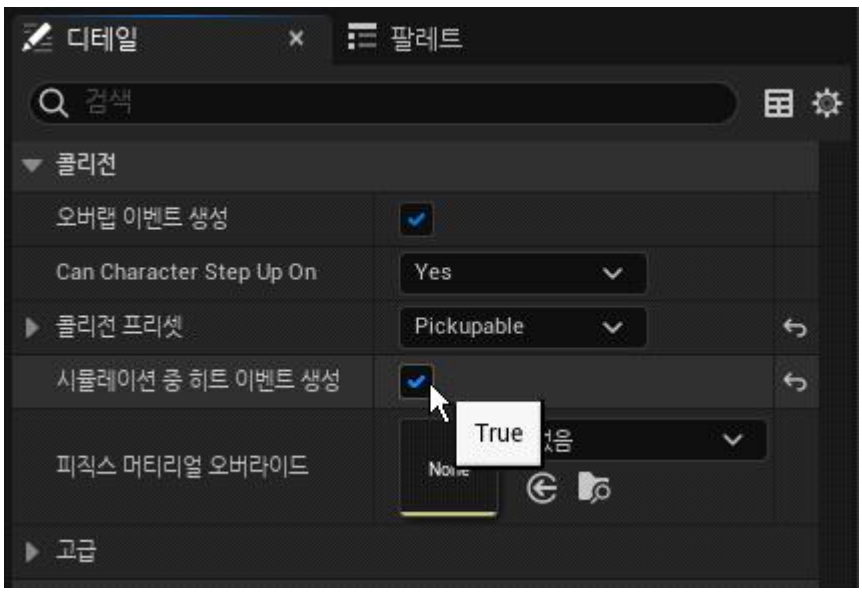
<참고> 위의 **Cube** 컴포넌트에는 큐브 모양의 스태틱 메시(`StaticMesh'/Engine/BasicShapes/Cube.Cube'`)가 포함되어 있다. 이 스태틱 메시는 내부에 콜리전 볼륨을 포함하고 있다. 따라서 **Cube** 컴포넌트에 대해서 콜리전 이벤트 노드를 추가할 수 있다.
만약 콜리전 볼륨이 포함되어 있지 않다면 우리가 직접 콜리전 컴포넌트 하나를 액터에 추가해주어야 한다. **컴포넌트** 탭에서 **+추가**를 클릭하고 **BoxCollision**, **CapsuleCollision**, **SphereCollision** 중 하나를 추가하면 된다. 그리고 그 컴포넌트에 대해서 이벤트 노드를 추가하면 된다.

12. 이벤트 그래프에 **OnComponentBeginOvelap (Cube)** 이벤트 노드가 추가될 배치될 것이다. 우리는 이벤트 그래프를 적절하게 작성하면 된다. 다음과 같이 **DestroyActor** 노드를 배치하자.



플레이해보자. 이제는 플레이어가 다가가면 큐브가 사라진다.
이제 모든 실습이 완료되었다.

13. 실습은 모두 완료되었다.
지금부터는 콜리전과 관련된 추가적인 지식에 대해서 학습해보자.
먼저, **Hit** 이벤트에 대해서 알아보자.
우리는 지금까지 콜리전 반응이 **겹침**인 경우의 이벤트 처리에 대해서 알아보았다. 콜리전 반응에는 **무시**(Ignore), **겹침**(Overlap), **블록**(Block)의 세 종류가 있다. 이 중에서 이벤트를 통지하는 목적으로 사용되는 콜리전 반응은 **겹침**이다. 겹침시작 이벤트와 겹침끝 이벤트에 대해서 처리할 수 있다. 한편, **블록**인 경우에도 이벤트 통지가 가능하도록 할 수 있다. 물리 시뮬레이션 중에서 블록된 두 객체가 서로 부딪치는 경우에 **Hit** 이벤트가 발생되도록 할 수 있다. **디테일** 탭에서 **콜리전** 아래에 **시뮬레이션 중 히트 이벤트 생성**(**Simulation Generates Hit Event**) 속성이 있다. 디폴트로 체크되어 있지 않지만 이곳에 체크하면 충돌 시에 **Hit** 이벤트가 발생된다. 이때 특정 기능을 수행하려면 `OnActorHit` 함수나 `OnComponentHit` 함수를 재정의하면 된다.



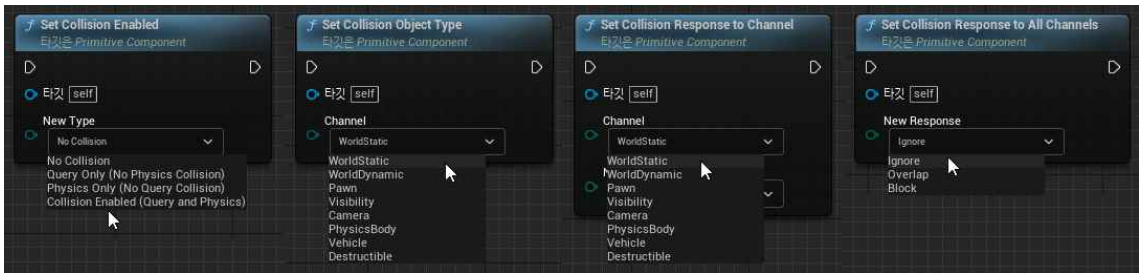
참고해두고 나중에 필요할 경우에 사용하자.

14. 지금부터는, 동적으로 콜리전을 제어하는 방법에 대해서 알아보자.
액터 컴포넌트의 **콜리전 프리셋**은 블루프린트 에디터의 **디테일** 탭에서 **콜리전** 영역의 **콜리전 프리셋** 속성에서 지정한다. 여기서 지정된 콜리전 프리셋이 디폴트로 적용되어 동작할 것이다. 한편, 플레이 도중에서 콜리전 프리셋을 동적으로 변경할 수도 있다. **SetCollisionProfileName** 함수를 사용하면 동적으로 콜리전 프리셋을 변경할 수 있다.

15. 콜리전 프리셋을 변경하면 모든 콜리전 관련 설정이 바뀌게 된다. 한편, 현재의 프리셋 상태에서 일부 수정을 희망하는 속성만을 변경하는 방법도 있다.
먼저, 콜리전 관련 설정에 대해서 알아보자. 아래의 콜리전 프리셋을 보자. 콜리전 프리셋 내부에는 **콜리전 켜짐(활성화됨)** (**Collision Enabled**) 속성, **오브젝트 타입** (**Object Type**) 속성, 그 아래에 각 트레이스 채널 및 객체 채널에 대해서 **채널별 콜리전 반응** 속성이 있다.

이들 각각의 속성을 개별적으로 수정할 수 있다.
콜리전 켜짐 속성을 수정할 수 있고, **오브젝트 유형** 속성을 변경할 수 있고, 특정 **오브젝트 유형**에 대해서 콜리전 반응을 변경할 수 있고, 모든 **오브젝트 유형**에 대해서 콜리전 반응을 일괄적으로 변경할 수 있다.

16. 일부 속성을 변경하는 함수들에 대해서 알아보자.



먼저, **콜리전 켜짐** 속성을 변경하는 방법으로, **SetCollisionEnable**를 호출하면 새로 명시된 값으로 수정된다.
그다음, 컴포넌트의 **오브젝트 유형**을 바꾸는 방법으로, **SetCollisonObjectType** 함수를 호출하면 명시된 **오브젝트 채널**로 바꾼다.
그다음, 한 오브젝트 채널에 대해서 콜리전 반응을 바꾸는 방법으로, **SetCollisionResponse to Channel** 함수를 호출하면 명시된 특정 객체 채널에 대해서 콜리전 반응을 재지정한다.
그다음, 모든 오브젝트 채널에 대해서 콜리전 반응을 하나로 바꾸는 방법으로, **SetCollisionResponse to All Channels** 함수를 호출하면 모든 채널에 대해서 지정된 콜리전 반응으로 바꾼다.

이 절에서는 오버랩 이벤트를 활용하는 방법을 학습하였다.

4. 트레이스 채널 추가하고 라인 트레이스 사용하기

이 절에서는 트레이스 채널을 추가하고 라인 트레이스를 사용하는 방법을 학습한다.

먼저, 트레이스 채널에 대해서 알아보자.
우리는 이전에서 **오브젝트 채널**에 대해서 학습하였다. 엔진에서는 콜리전 반응이 동일한 객체들을 하나의 그룹으로 묶어서 그룹 단위로 관리하기 위해서 **오브젝트 채널**의 개념을 도입하였다. 즉 동일한 콜리전 반응을 가지는 객체 그룹을 **오브젝트 채널**이라고 한다.
한편, 오브젝트 채널과 별도로 콜리전 반응을 처리해야 할 경우가 있다. 레이저 광선을 쏘는 경우가 대표적이다. 한 액터가 레이저 광선을 쏘고 광선에 부딪치는 다른 액터를 체크하는 경우를 생각해보자. 광선이 다른 액터와 충돌했는지의 콜리전 반응에 대한 분류가 필요하다. 광선은 나무판은 통과하지 못하지만 유리판은 충돌하지 않고 통과한다.
엔진에서의 트레이스 기능은 그래픽에서의 레이 캐스트(ray cast) 또는 레이 트레이스(ray trace) 기법과 동일하다. 여러 그래픽 응용에서 중요하게 사용되는 기법이다.

트레이스 채널의 활용에 대해서 알아보자.
대표적으로, 시야 확인이다. 한 캐릭터의 시야에 다른 캐릭터나 특정 지점이 들어왔는지의 여부를 확인하기 위해서 사용된다. 캐릭터에서 출발한 광선의 첫 콜리전 위치가 다른 캐릭터라면 시야에 들어왔다고 판정하면 된다.
또다른 예로, 착탄 지점 계산이다. 발사한 총알이 맞춘 목표물을 계산하기 위해서 사용된다. 발사 위치에서 출발한 광선의 첫 콜리전 위치가 바로 착탄 지점에 해당한다. 총알을 따라가면서 총알과 물체와의 콜리전을 계산하지는 않는다. 총알은 시각적인 효과를 제공할 뿐이고 발사 시점에서 모든 것이 결정된다.
또다른 예로, 차폐 여부 확인이다. 폭탄은 일정범위 내에서 모든 방향으로 타격을 준다. 폭탄의 타격이 플레이어에게 도달하는지를 확인하기 위해서 사용된다. 폭탄에서 플레이어 방향으로 출발한 광선의 첫 콜리전 위치가 플레이어라면 플레이어는 폭탄의 타격을 받게 된다.

엔진에서 제공하는 트레이스의 종류에 대해사 알아보자.
트레이스의 가장 대표적인 종류는 **라인 트레이스**이다. 라인 트레이스는 두 지점을 직선으로 연결하여 그 중간에서의 콜리전을 확인하는 방식이다.
만약 두 지점의 위치가 정확하게 정해진 경우라면 라인 트레이스로 충분할 것이지만, 한쪽 지점의 위치가 정확하지 않거나 또는 방향이 정확하지 않은 경우가 있다. 이런 경우에는 볼륨 모양의 광선을 쏘아서 콜리전을 확인하는 방식을 사용하면 된다. 이들 트레이스의 종류에는 볼륨의 모양에 따라 **박스 트레이스**, **캡슐 트레이스**, **구체 트레이스**가 있다.

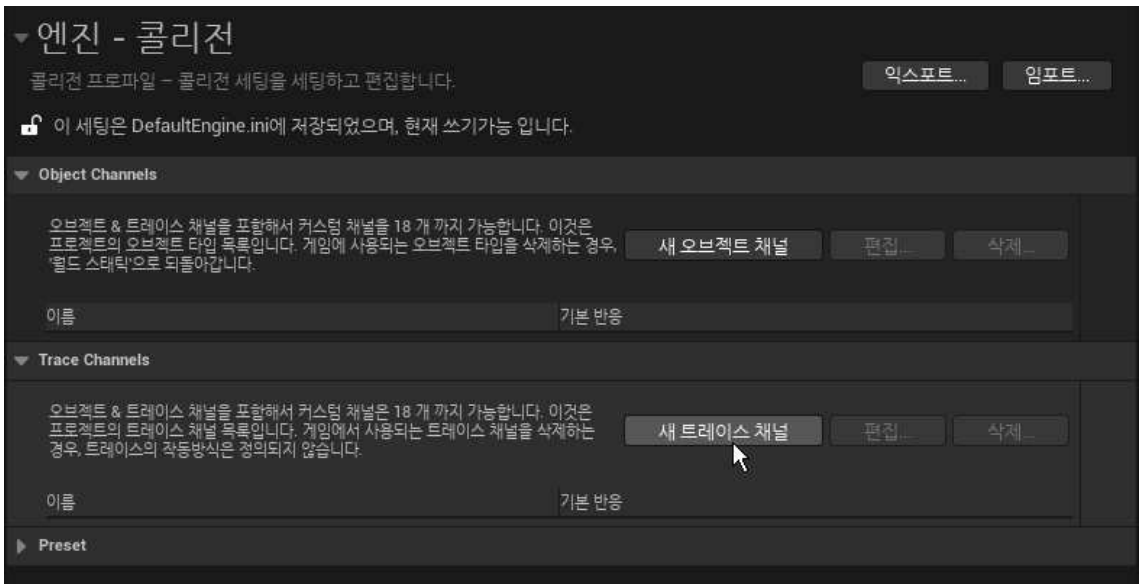
<참고> 트레이스에 대한 자세한 내용은 다음의 문서를 참조하자.
https://docs.unrealengine.com/traces-with-raycasts-in-unreal-engine/

---

1. 새 프로젝트 **Plinetrace**를 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 Plinetrace로 입력하고 **생성** 버튼을 클릭하자. 프로젝트가 생성되고 언리얼 에디터 창이 뜰 것이다.

창이 뜨면, 메뉴바에서 **파일** » **새 레벨**을 선택하고 **Basic**을 선택하여 기본 템플릿 레벨을 생성하자.
그리고, 레벨 에디터 툴바의 저장 버튼을 클릭하여 현재의 레벨을 MyMap으로 저장하자.
그리고, **프로젝트 세팅** 창에서 **맵&모드** 탭에서의 **에디터 시작 맵** 속성값을 MyMap으로 수정하자.
그리고, 툴바의 **액터 배치** 아이콘을 클릭하고 **기본** 아래의 **플레이어 스타트** 액터를 드래그하여 레벨에 배치하자. 위치를 (0,0,112)로 지정하자.

2. 엔진에서는 디폴트로 2개의 트레이스 채널을 제공한다.
그러나, 우리는 우리의 커스텀 트레이스 채널을 만들어보자.
프로젝트 세팅 창을 열고 **엔진** » **콜리전** 탭으로 이동하자.
새 트레이스 채널 버튼을 클릭하자.



<참고> 엔진에서는 디폴트로 제공하는 2개의 트레이스 채널인 **Visibility**와 **Camera**에 대해서 알아보자.
Visibility 채널의 트레이스는 보이지 않는 객체들은 관통하여 지나간다. 따라서 보이지 않는 객체들은 **Visibility** 채널을 무시로 설정한다. **Pawn**, **CharacerMesh**, **Ragdoll**은 보이는 객체이지만 **Visibility** 채널을 **무시**로 설정해서 폰 뒤의 물체가 보이도록 한다. 그 이유는 폰의 다리 사이 등으로 그 뒤의 물체가 보이는 경우가 흔하기 때문이다.
Camera는 카메라 물건 자체에 대한 채널이다. 따라서 대부분의 물체는 **Camera** 채널을 블록으로 설정한다. 다른 액터를 블록하는데 사용되는 투명 볼륨인 **블로킹 볼륨** 객체에 대해서 **Visibility** 채널의 트레이스는 통과할 수 있으나 **Camera** 채널의 트레이스는 통과할 수 없다.

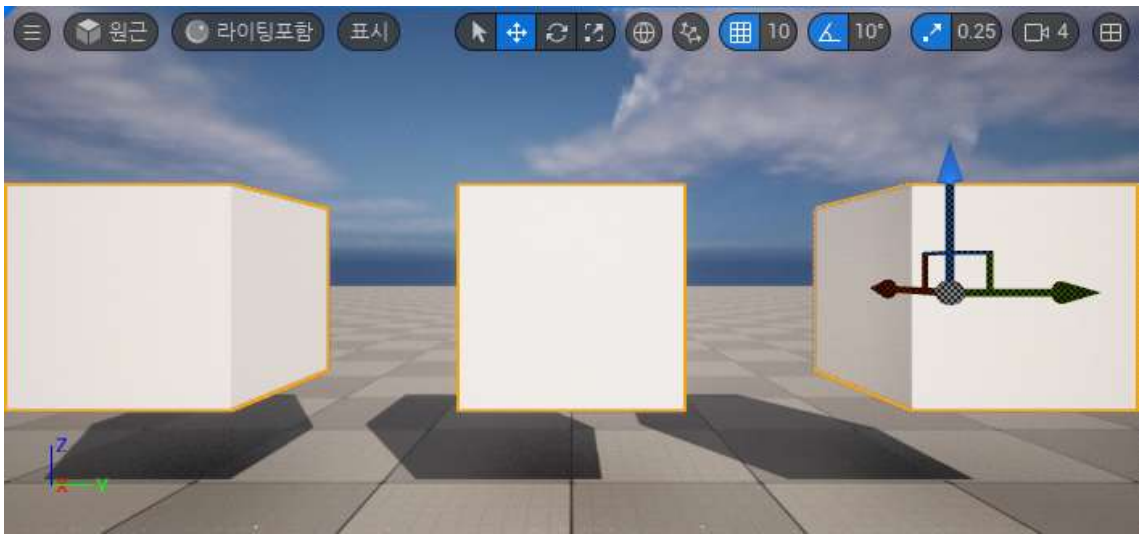
3. 엔진에 새 채널의 이름은 **Sight**라고 하고 기본 반응은 **Block**으로 두고 수락을 클릭하자.

<참고> 새로운 **트레이스 채널**이 생성될 때에도 **오브젝트 채널**이 생성될 때와 마찬가지로 엔진은 기존의 모든 프리셋에 대해서 새로운 **트레이스 채널**과의 반응을 자동으로 추가해준다. 이 때에 모두 기본 반응으로 지정한

Block으로 추가해준다.
새로운 **트레이스 채널**을 생성한 후에는 기존의 **프리셋**에 대해서 수정할 사항이 있는지를 전체적으로 체크하자. 우리의 예에서, **Sight**의 **기본 반응**으로 설정한 **블록**으로 명시되어 있을 것인데, 이것이 부적절한 **프리셋**도 있을 것이다. 예를 들어 **InvisibleWall** 프리셋에서는 시야를 막지 않도록 **Sight**를 **무시**로 바꾸어 주어야 한다. 그 외에도 **NoCollision**, **OverlapAll**, **OverlapAllDynamic**, **OverlapOnlyPawn**, **Trigger** 프리셋에서 **Sight**를 **무시**로 바꾸어 주어야 한자. 이 과정은 실제 게임 제작 시에는 반드시 필요한 과정이지만, 우리는 그냥 넘어가도록 하자.

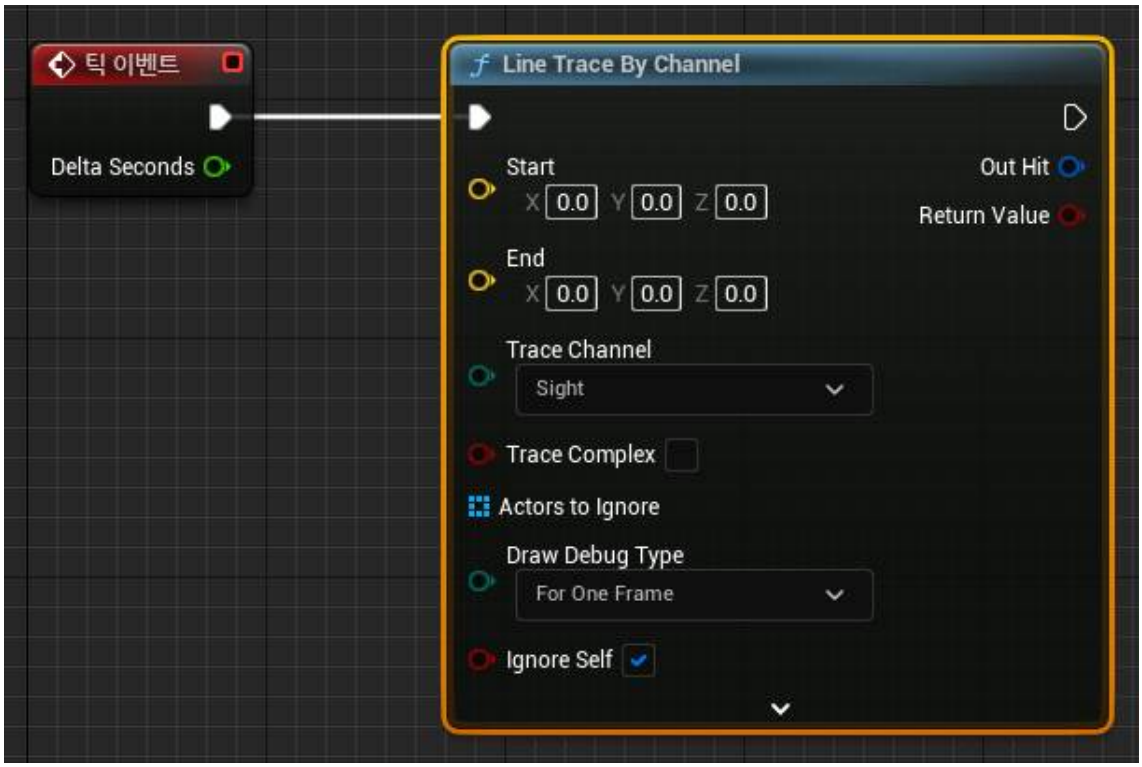
4. 먼저, 이전 프로젝트에서의 작업을 가져오는 일을 진행하자.
첫 번째로, 이전 프로젝트해서 생성해둔 블루프린트 클래스를 가져오자. 윈도우 파일 탐색기에서 이전 `Pinputevent` 프로젝트의 `Content` 폴더로 이동하자. 그 아래에 있는 3개의 애셋 파일(`BP_MyCharacter`, `BP_MyGameMode`, `BP_MyPlayerController`)을 **Ctrl+C**로 복사하고 새 프로젝트의 `Content` 폴더 아래로 이동하여 **Ctrl+V**로 붙여넣자.
두 번째로, 이전 프로젝트해서 설정한 입력 매핑을 가져오자. 우선, 새 프로젝트를 종료하자. 그다음, 이전의 프로젝트 `Pinputevent` 프로젝트의 `Config` 폴더로 이동하자. 그 아래에 있는 `DefaultInput.ini` 파일을 **Ctrl+C**로 복사하고 새 프로젝트의 `Config` 폴더 아래로 이동하여 **Ctrl+V**로 붙여넣자. 그다음, 새 프로젝트를 다시 로드하자.
세 번째로, 새 프로젝트의 **월드 세팅** 탭에서 **게임모드 오버라이드** 속성에 **BP_MyGameMode**를 지정하자.
위의 과정에 대한 상세한 설명은 `Pinputevent` 프로젝트에 대한 학습 내용을 참조하자.
이제, 이전 프로젝트에서의 모든 작업이 포함되도록 준비과정을 완료하였다.

5. 레벨에 상자를 배치하자.
툴바의 **액터 배치** 아이콘을 클릭하고 **셰이프** 아래의 **큐브**를 드래그해서 배치하자. 세 개의 **큐브**를 (300,-200,100), (300,0,100), (300,200,100)에 배치하자. 배치된 각 **큐브** 인스턴스의 이름을 **CubeLeft**, **CubeMiddle**, **CuberRight**라고 하자.



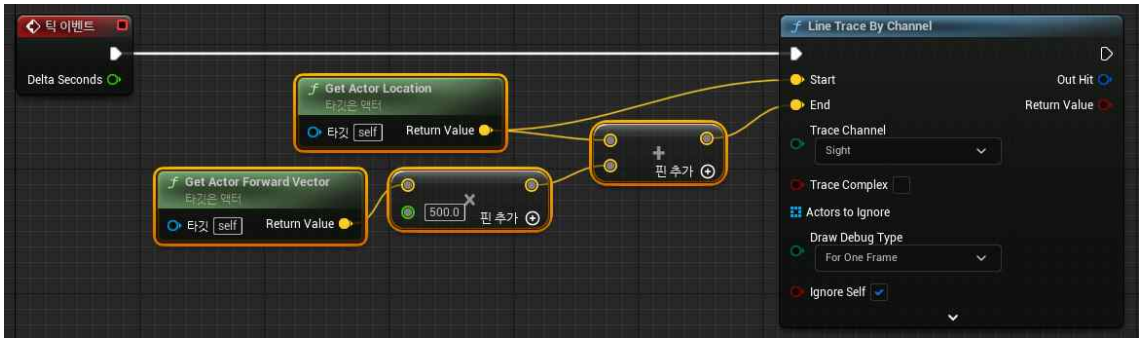
6. 이제 그래프를 작성해보자.
BP_MyCharacter를 더블클릭하여 블루프린트 에디터를 열자.
이벤트 그래프 탭에서 **Tick 이벤트** 노드의 실행핀을 당기고 라인 트레이스 노드인 **LineTraceByChannel** 노드를 배치하자.

이 노드는 **Start** 위치와 **End** 위치를 연결하는 선을 따라 콜리전 트레이스를 수행한다. 그 중간에 발생한 첫 번째 블로킹 히트를 반환한다. 모든 객체에 대해서 콜리전을 체크하는 것이 아니라 **LineTraceByChannel** 입력핀에 명시한 **트레이스 채널**에 반응하도록 설정된 객체들과만 콜리전을 체크한다.
그다음, **TraceChannel** 입력핀의 값을 **Visibility**에서 **Sight**로 바꾸자.
그다음, **DrawDebugType** 입력핀의 값을 **None**에서 **ForOneFrame**으로 바꾸자. 이것은 라인 트레이스를 눈으로 쉽게 확인할 수 있도록 하는 디버그 목적의 표시 기능이다.



<참고> **DrawDebugType** 입력핀에는 **None**, **For One Frame**, **For Duration**, **Persistent** 중의 하나를 선택할 수 있다. 디폴트는 광선을 표시하지 않는 **None**이다. **For One Frame**으로 지정하면 현재 프레임에서의 광선이 표시된다. **For Duration**으로 지정하면 현재 프레임에서의 광선과 더불어 일정 시간 이전 프레임에서 표시된 광선이 함께 표시된다. **Persistent**로 지정하면 모든 프레임에서의 광선이 계속 누적되어 표시된다.

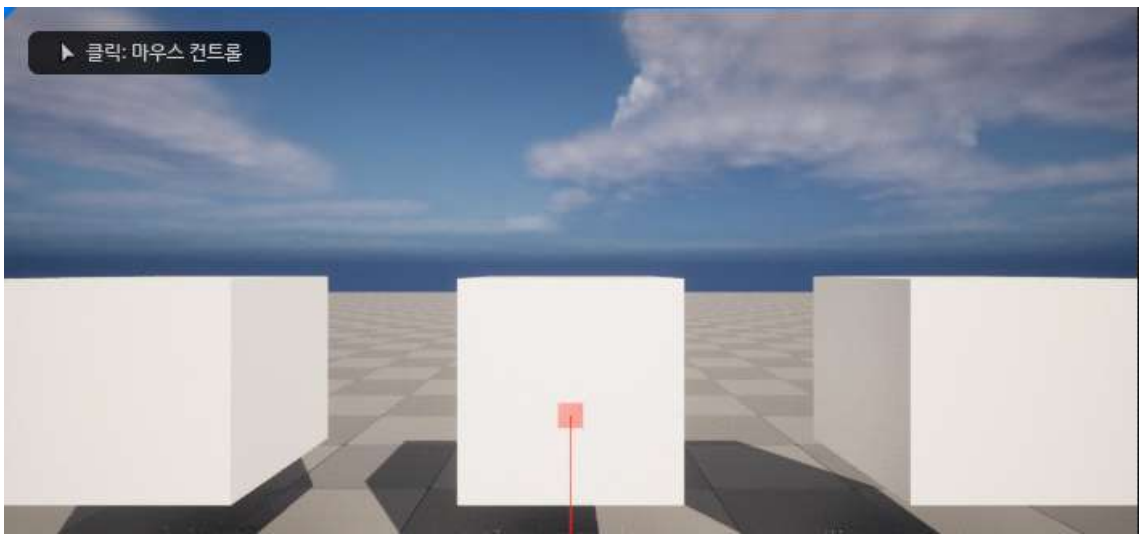
7. 그다음, **Start** 핀과 **End** 핀에 연결해주어야 한다.
라인 트레이스를 사용하려면 시작 지점과 끝 지점의 위치 정보가 필요하다. 시작 지점은 플레이어 폰의 눈의 위치로 하면 된다. 이를 위해서 먼저 액터의 위치를 추출하는 액션 노드인 **GetActorLocation** 노드를 배치하자. 시작 지점은 **GetActorLocation**의 출력 위치 벡터를 **Start** 핀으로 연결하면 된다. 끝 지점은 시선 방향으로의 일정 거리만큼 떨어진 위치의 지점으로 하자. 시선 방향은 액터의 전방 벡터를 구하여 시선 방향을 얻고 이 벡터에 시야 거리만큼을 곱해서 벡터를 늘인 후에 시작 지점 위치 벡터를 더해주면 끝 지점 위치 벡터가 완성된다. 이를 위해서 먼저 전방 벡터를 추출하는 액션 노드인 **GetActorForwardVector** 노드를 배치하자. 끝 지점은 **GetActorForwardVector**의 방향 벡터를 5m 만큼 스케일한 후에 시작 지점에 더해서 **End** 핀으로 연결하면 된다.



<참고> **곱하기** 노드의 사용법에 대해서 학습해보자.
위의 **곱하기** 노드를 보면 두 입력핀의 타입이 서로 다르다. 한 입력핀이 벡터이고 다른 입력핀이 플로트인 경우에는 플로트 값을 벡터의 각 요소에 곱하여 출력값을 결정한다.
곱하기 노드를 배치하는 방법은 먼저 액션선택 창에서 '*' 또는 multiply 또는 곱하기를 검색하여 배치한다. 그다음, 배치된 **곱하기** 노드의 입력핀에서 우클릭하고 팝업메뉴에서 **핀 변환...**으로 이동하면 **플로트**, **벡터** 등 많은 핀 변환 메뉴가 나열된다. 이 중에서 원하는 유형을 선택하면 된다.

<참고> 액터의 위치에 관련된 다음과 같은 노드가 있다. **GetActorLocation**는 액터의 위치를 추출하는 노드이고, **SetActorLocation**는 액터를 주어진 위치로 이동시키는 노드이다. 액터의 배치 방향을 리턴하는 노드는 다음과 같은 노드가 있다. **GetActorForwardVector**는 액터의 전방 벡터를 추출하는 노드이다. 액터의 우향 벡터를 추출하는 **GetActorRightVector**가 있고, 액터의 상향 벡터를 추출하는 **GetActorUpVector**가 있다. 방향은 월드 공간에서 정의되는 단위 벡터로 리턴한다.

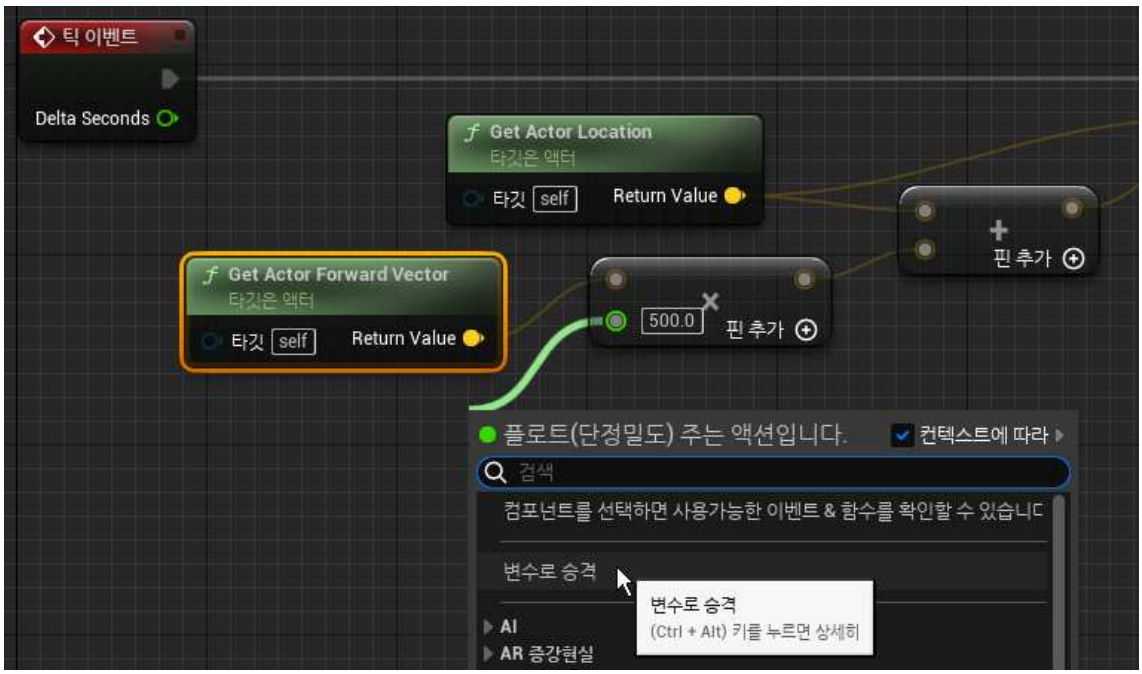
8. 플레이해보자. 플레이어의 전방으로 붉은색 광선이 나갈 것이다. 5m의 길이의 광선이 나가고 그 중에게 콜리전이 발생하면 그 지점이 붉은색으로 표시될 것이다.



<참고> 광선은 5m의 길이이다. 히트가 되지 않은다면 광선은 모두 붉은색으로 표시된다. 히트가 되면 히트되는 부분까지 붉은색으로 표시되고 히트 지점도 붉은색으로 표시된다. 한편, 히트된 이후의 남은 광선 부분은 녹색으로 표시된다. 1인칭 시점에서는 이를 확인할 수가 없다. 이 때에는 실행한 후에 **F8**을 클릭하여 플레이어에서 벗어난 후에 카메라를 제어하면서 이동해보자. 관통한 후의 녹색 부분을 확인할 수 있을 것이다.

9. 이제 그래프를 정리해보자.
곱하기 노드의 입력핀에 상수값 500으로 지정되어 있다. 이 입력핀을 왼쪽으로 드래그하고 팝업메뉴

에서 **변수로 승격**을 선택하자.

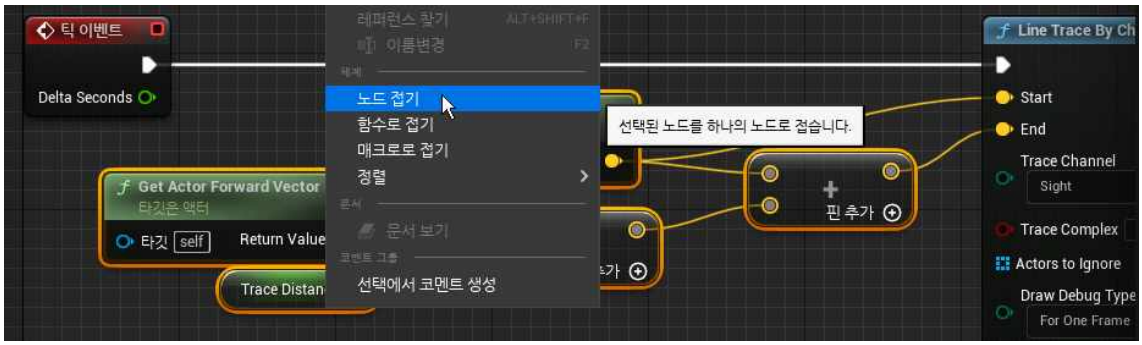


새로운 변수가 추가될 것이다. 추가된 새 변수의 **Get** 노드가 배치될 것이다. 변수의 이름을 **TraceDistance**로 수정하자.

컴파일하고 변수의 **디테일** 탭을 확인해보면 기본값이 500으로 설정되어 있을 것이다.

또한, 디테일 탭에서 **인스턴스 편집기능**에도 체크해두자. 나중에 레벨 에디터에서의 디테일 탭에서도 기본값을 수정할 수 있게 된다. 현재는 미리 배치해두는 것이 아니어서 사용할 수는 없지만 나중에 미리 배치해두는 경우가 생기면 활용할 수 있을 것이다.

10. 이제, **Tick 이벤트** 노드와 **LineTraceByChannel** 노드의 사이에 있는 5개의 노드를 모두 선택한 후에 우클릭하고 **노드 접기**를 선택하자.

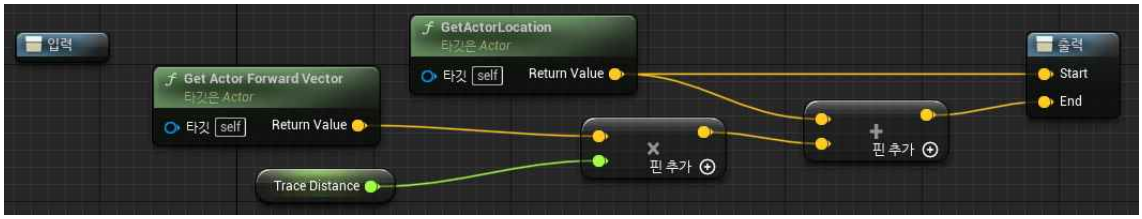


선택된 노드들이 하나의 노드로 대치될 것이다. 이 노드의 이름을 **GetTraceStartEndPosition**으로 하자.

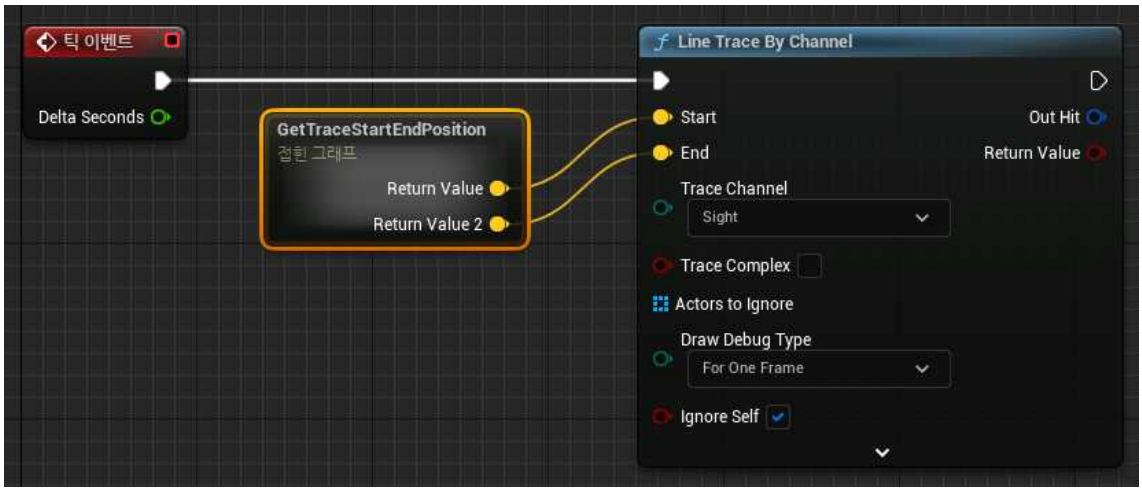
11. 접힌 **GetTraceStartEndPosition** 노드를 더블크릭하자. 접힌 노드의 이벤트 그래프 탭이 열릴 것이다.

매크로처럼 보이지만 매크로가 아니고 노드들을 단순히 접은 것이다.

두 개의 출력 노드를 클릭하고 오른쪽 디테일 탭에서 이름을 각각 **Start**와 **End**로 수정하자. 아래와 같은 그래프 모양이 될 것이다.



12. 이벤트 그래프 탭으로 다시 돌아오자. 다음과 같이 바뀌어 보일 것다.



접힌 **GetTraceStartEndPosition** 노드를 더블클릭하면 접힌 그래프의 노드 네트워크를 수정할 수 있다. 또한, 내 블루프린트 탭에서 이벤트 그래프 영역에도 나열되어 있다. 이것을 클릭해도 접힌 그래프를 편집할 수 있다.

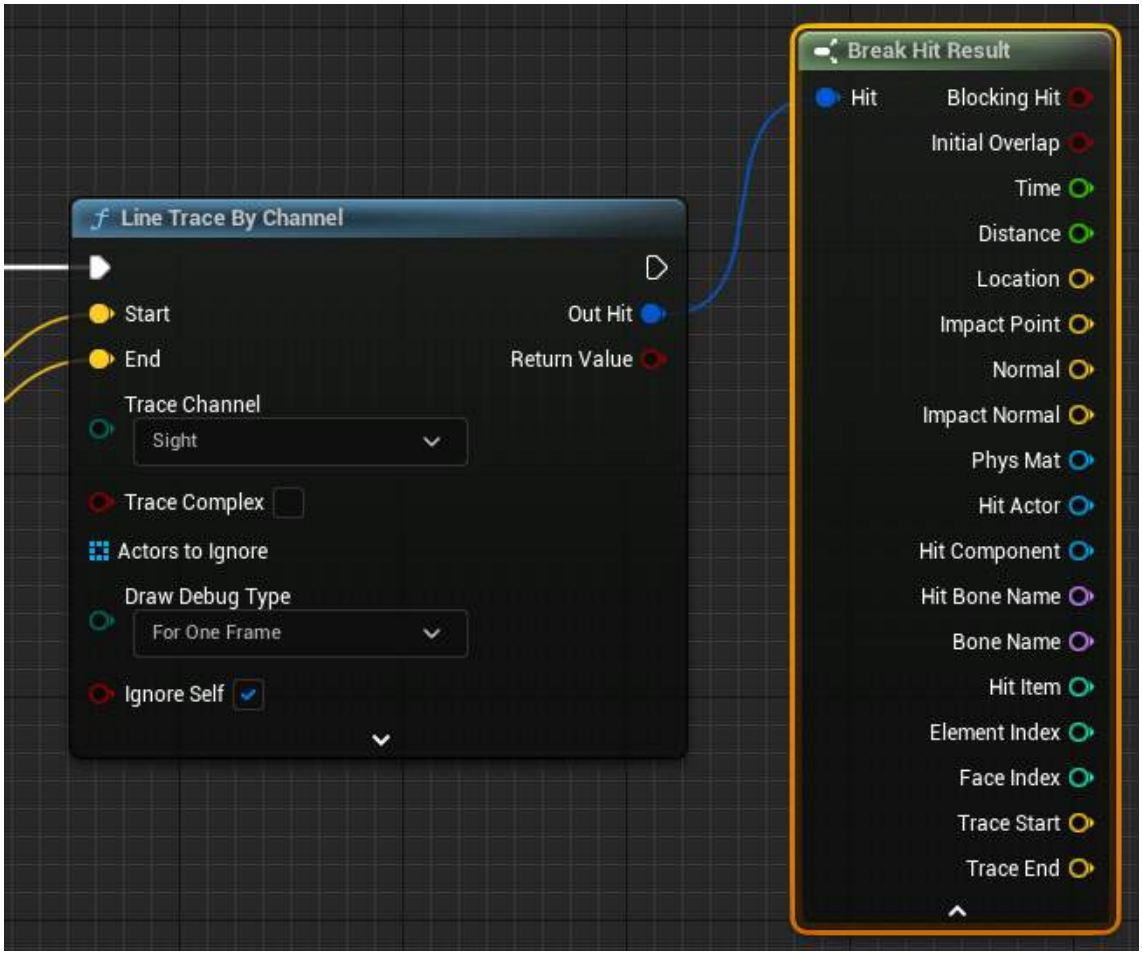
<참고> 접힌 노드를 원래대로 복원하려면 접힌 노드 위에서 우클릭한 후에 **노드 펼침**을 선택하면 된다. 접힌 노드가 다시 펼쳐진 이후에는 이전의 접힌 노드 네트워크로 다시 돌아갈 수 없다. 다시 노드 접기를 수행하고 정리 작업을 모두 다시 해야 한다.

13. 이제부터는, **LineTraceByChannel** 노드의 출력핀에 대해서 알아보자.

출력핀에는 **OutHit**와 **ReturnValue**의 두 개의 출력핀이 있다. **ReturnValue**는 **Boolean** 타입으로 히트가 있는 경우는 **true**이고 없는 경우는 **false**이다. **OutHit**는 **HitResult** 구조체 타입으로 자세한 히트 정보를 리턴한다.

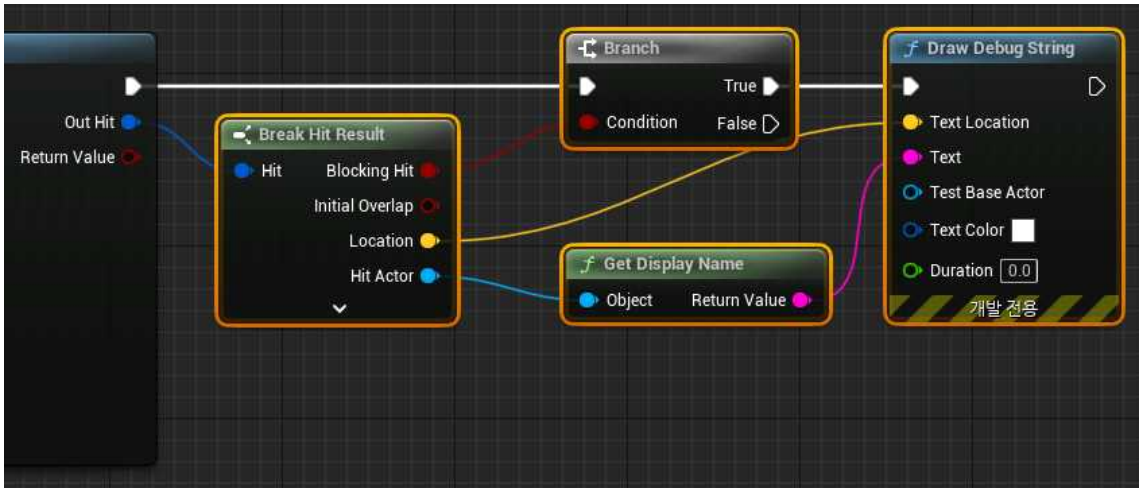
이제, **OutHit** 출력값을 확인해보자. 엔진에서 제공하는 대부분의 구조체에 대해서, **Make**와 **Break**가 구조체명 앞에 붙은 편리 함수가 제공된다. **BreakHitResult** 함수를 사용하면 **HitResult** 구조체를 분해하여 구조체 내의 각 요소들을 접근할 수 있다.

LineTraceByChannel 노드의 **OutHit** 출력핀을 드래그하고 **BreakHitResult** 노드를 배치하자.



BreakHitResult 노드는 많은 출력핀을 제공한다. 먼저, **BlockingHit**는 블로킹 히트가 있으면 **true**를 리턴한다. 이 값이 **true**인 경우에만 그 아래의 출력핀을 사용하면 된다. 그리고, **Location**은 월드 공간에서의 히트 위치이다. 그리고, **HitActor**는 히트된 액터이다.

14. **LineTraceByChannel** 노드 다음에 이어서 노드 네트워크를 작성해보자.
먼저, **Branch** 노드를 배치하자. 그리고, **BreakHitResult** 노드의 **BlockingHit** 출력핀이 **true**일 경우에만 다음으로 진행되도록 **BlockingHit** 출력핀을 **Branch** 노드의 **Condition** 입력핀에 연결하자.
그다음, **DrawDebugString** 노드를 검색하여 배치하자. 이 노드는 주어진 3D 월드 위치에 디버그 문자열을 표시하는 함수이다.
그다음, **BreakHitResult** 노드의 **Location**과 **HitActor** 출력핀을 **DrawDebugString** 노드의 **TextLocation**과 **Text**에 연결하자. **Text**에 연결되는 연결선에 대해서는 중간에 형변환 노드가 자동으로 추가된다.
LineTraceByChannel 노드와 **BreakHitResult** 노드와 **DrawDebugString** 노드의 실행핀을 연결하자.
그리고, **BreakHitResult** 노드의 연결되지 않은 출력핀이 많다. 노드의 가장 아래의 접기 화살표 아이콘을 클릭하면 연결된 출력핀만 표시되도록 노드 크기를 줄여준다.
이제 다음과 같은 모습의 그래프가 될 것이다.



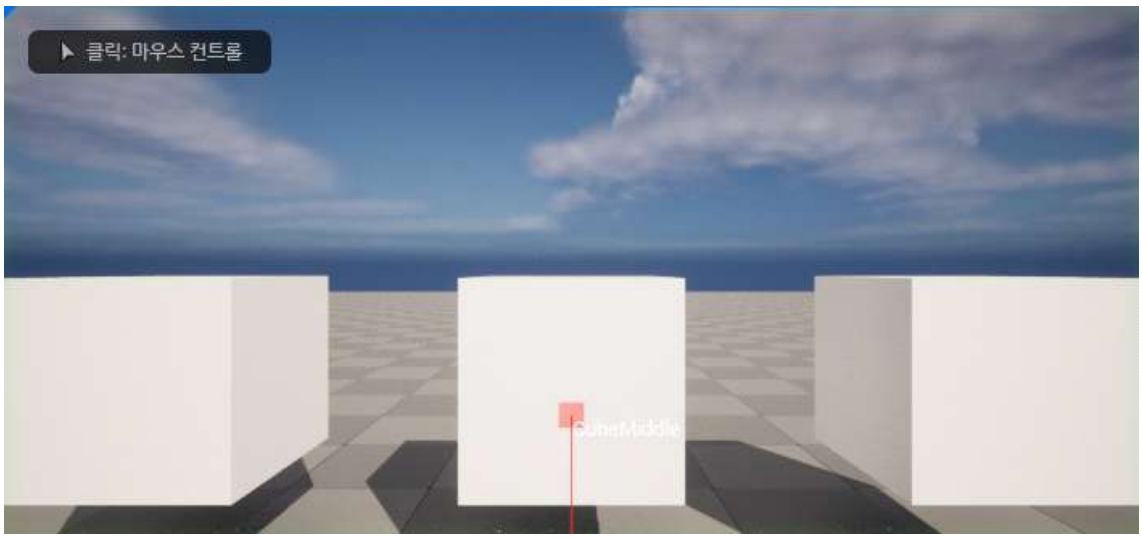
<참고> **LineTraceByChannel** 함수에 대한 자세한 내용은 다음의 문서를 참조하자.

https://docs.unrealengine.com/BlueprintAPI/Collision/LineTraceByChannel/

HitResult 구조체에 대해서는 다음의 문서를 참조하자.

https://docs.unrealengine.com/BlueprintAPI/Collision/BreakHitResult/

15. 플레이해보자. 마우스를 움직여보자. 히트가 발생하면 액터 인스턴스의 이름이 히트 위치에 표시될 것이다.



이 절에서는 트레이스 채널을 추가하고 라인 트레이스를 사용하는 방법을 학습하였다.

□