18_ 물리 엔진

<제목 차례>

인천대학교 컴퓨터공학부 박종승

1. 개요

이 장에서는 물리 엔진에 대해서 학습한다.

게임 세계에서 객체를 움직이는 방법에는 여러 가지가 있다.
먼저, Tick 이벤트 노드를 사용하여 매번 위치를 변경하는 방법이 있다. 매번 위치를 변경해야 하므로 세밀한 수작업이 많을 수 있다.
그다음에는, 타임라인을 사용하는 방법이 있다. 타임라인에서 특정 위치나 방향을 가지는 키 프레임을 특정 시간 위치에 설치한다. 키 프레임 사이에서는 값이 자동으로 보간된다. 이 방법은 게임에서 사용자에 반응하지 않는 액터의 움직임에 사용하는 방법이다. 사용자의 반응에 따라서 정해진 움직임을 바꿀 수 없고 사용자 반응과 무관하게 진행되어야 한다.
그다음에는, 무브먼트 컴포넌트를 사용하는 방법이 있다. 액터나 캐릭터 내부에 무브먼트 컴포넌트를 추가할 수 있다. 무브먼트 컴포넌트는 자신이 속한 액터나 캐릭터에 이동 기능을 제공하는 컴포넌트이다. 예를 들어 캐릭터 무브먼트 컴포넌트에는 캐릭터의 걷기 이동 등의 기능이 포함되어 있다. 속성값을 적절히 설정하고 이동 함수를 호출하면 된다.
그다음에는, 물리 엔진을 사용하는 방법이 있다. 객체에 피지컬 파라메터를 지정한 후에 물리 엔진에게 모든 처리를 맡기는 방법이다. 속성값을 적절히 설정하고 이동을 맡긴다는 측면에서 무브먼트 컴포넌트와 사용 방법에서 유사한 측면이 있다. 그러나 물리 엔진은 무브먼트 컴포넌트와는 다르게 현실 세계의 물리 현상을 시뮬레이션하는 것에 집중한다.

물리 엔진의 한계를 알아보자.
물리 엔진의 사용을 긍정적으로만 생각하면 안된다. 물리 엔진의 사용에는 여러 한계점이 있기 때문이다.
물리 엔진은 현실 세계의 물리 시뮬레이션을 수행한다. 현실 세계의 물리 시뮬레이션은 결과를 미리 예측하기가 힘들고 원하는 목표를 재현하기도 힘들다. 물리 엔진의 이런 한계점에 대해서 더 자세히 알아보자.
먼저, 물리 엔진에는 재현성이 없다. 동일한 움직임을 반복해서 실행할 때에 매번 결과가 달라질 수 있다. 랜덤값의 사용이나 초기화의 순서가 하드웨어의 부하 정도 등의 실행할 때마다 달라질 수 있고 사용자가 제어할 수 없는 환경적 요소가 많기 때문이다.
다음으로, 힘을 가하게 된 이후에 어떻게 움직일 지의 예측이 어렵다. 물리 엔진에서의 피직스 시뮬레이션은 힘과 토크를 사용해서 강체를 움직인다. 힘과 토크가 가해지면 강체의 모양, 밀도, 무게중심 등의 관련된 여러 속성에 따라서 움직임이 결정된다. 원하는 운동을 얻기 위한 힘과 토크를 결정하는 것은 복잡한 계산을 풀어서 구할 수는 있겠지만 매우 어렵다.

물리 엔진의 활용 방법에 대해사 알아보자.
물리 엔진의 예측할 수 없는 특성으로 인하여 게임에서 시나리오 진행에서 물리 엔진을 적용하는 것이 꺼려질 수 있다. 따라서 게임에서 객체를 움직일 때에는 여러 경우를 고려하면서 가변적으로 물리 엔진을 활용해야 한다. 어떤 경우에는 물리 엔진에 전적으로 맡겨두고 어떤 경우에는 다른 방법으로 움직이도록 하고 어떤 경우에는 혼합하여 움직이도록 하는 식으로 해야 한다.
한편, 게임의 진행에 민감하지 않은 부분에 대해서는 물리 엔진을 적용하면 좋은 결과를 얻을 수 있다. 깨져서 부딪치며 떨어지는 장면이나 비탈길을 굴러 떨어지는 장면 등은 게임의 진행과 무관하

므로 물리 엔진을 적용하기에 좋은 활용이다. 물리 엔진을 사용하면 약간의 속성값 설정 만으로 멋진 장면을 만들 수 있다. 이러한 세부적인 부분을 물리 엔진을 사용하지 않고 구현하는 것이 오히려 매우 부담스러운 일이다.
물리 엔진의 한계점에도 불구하고 게임 개발에서 물리 엔진의 사용은 권장된다. 게임의 진행에 영향을 주지 않는 많은 요소들이 있고 이들을 물리 엔진에 맡김으로써 개발이 쉬워지고 실감을 높일 수 있기 때문이다. 최근의 게임에서는 물리 엔진이 널리 사용되고 있다.

언리얼 엔진는 엔비디어(NVIDIA)사의 물리 엔진인 PhysX가 내장되어 있다. 약간의 설정만으로 PhysX의 멋진 기능을 사용할 수 있다.

<참고> 무브먼트 컴포넌트에 대해서는 다음의 문서를 참조하자.
https://docs.unrealengine.com/movement-components-in-unreal-engine/

<참고> 언리얼에서의 피직스와 관련된 모든 문서는 다음의 페이지에 정리되어 있으니 참조하자.
https://docs.unrealengine.com/physics-in-unreal-engine/

2. 물리 엔진 처음으로 사용해보기

이 절에서는 물리 엔진을 사용하는 방법에 대해서 학습한다.

물리 엔진은 메시에 적용하는 것이 아니라 콜리전 볼륨에 적용한다. 따라서 물리 엔진을 적용한 액터에는 반드시 콜리전 볼륨이 있어야 한다. 콜리전 볼륨이 복잡할수록 물리 엔진의 부하가 증가한다. 따라서 단순한 형태의 콜리전 볼륨이 되도록 하자. 자동 생성 기능을 사용하면 물리 엔진의 적용을 고려하여 적절한 수준의 콜리전 볼륨을 생성해준다.

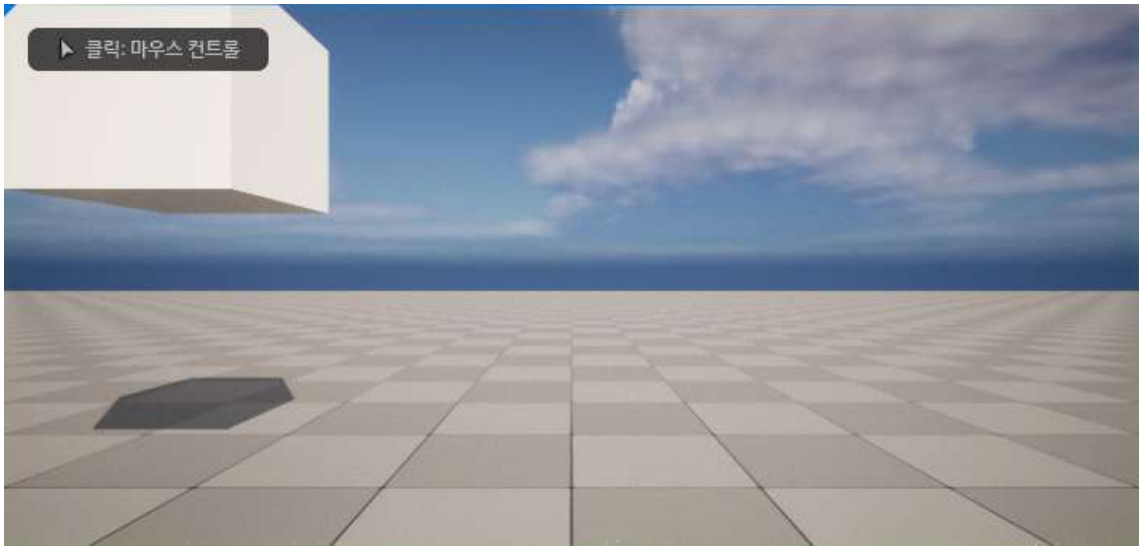
특정 액터에 대해 물리 엔진을 적용하기 위해서는 레벨 에디터에서 레벨에 배치된 액터 인스턴스의 디테일 탭에서 관련된 속성을 지정해주어야 한다. 가장 중요한 속성으로 **피직스** 영역에 있는 **피직스 시뮬레이트(Simulate Physics)** 속성을 체크해서 활성화해 주어야 한다. 또한 액터의 **모빌리티** 속성도 **무버블**로 되어 있어야 물리 엔진이 액터의 위치를 바꿀 수 있다.

이제부터 물리 엔진을 사용하는 방법에 대해서 학습해보자

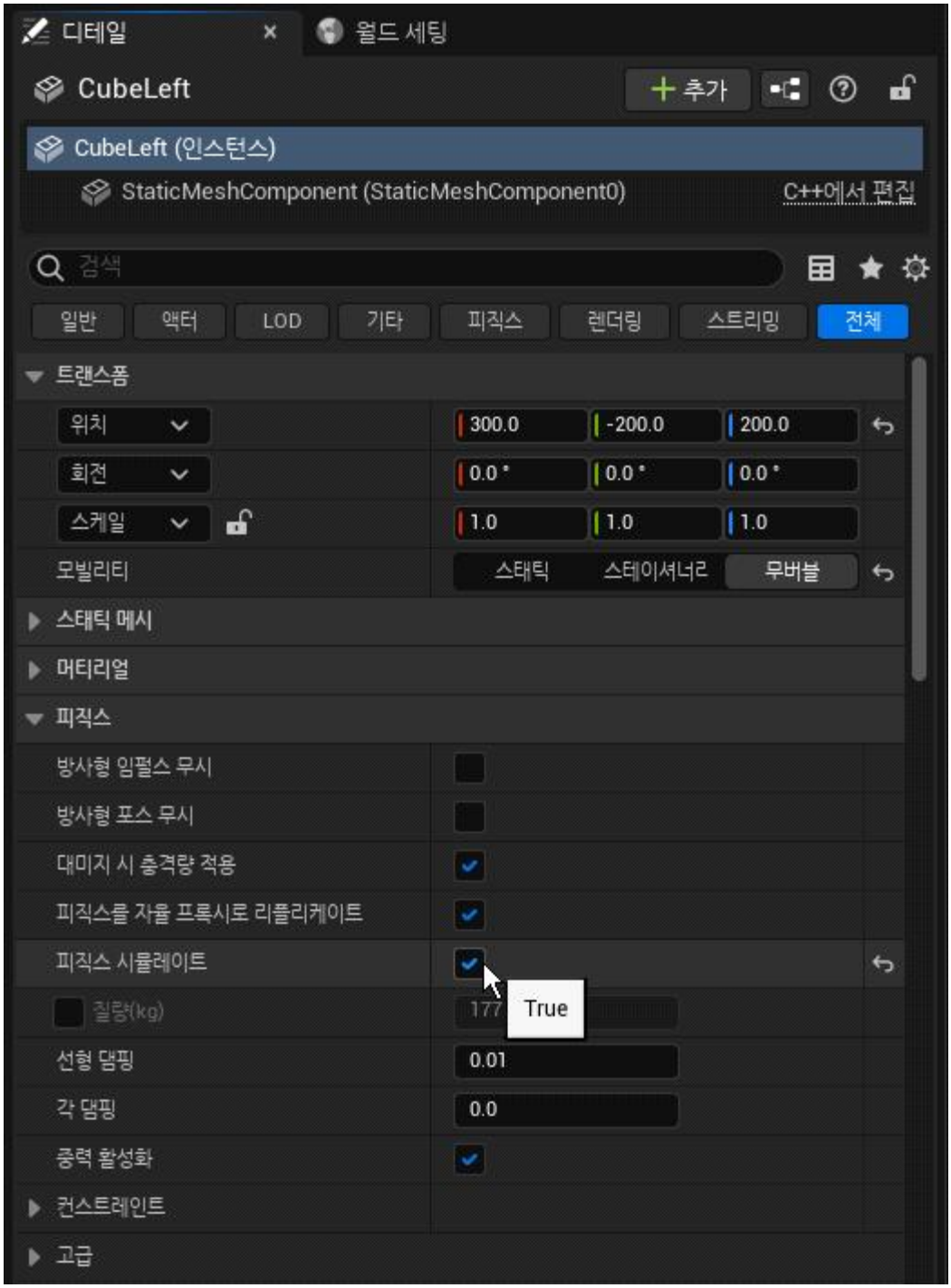
---

1. 새 프로젝트 `Pphysicsfirst`를 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 `Pphysicsfirst`로 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 `MyMap`으로 저장하자.
그리고, **프로젝트 세팅** 창에서 **맵&모드** 탭에서의 **에디터 시작 맵** 속성값을 `MyMap`으로 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 **CubeLeft**로 수정하자. 위치는 공중에 떠 있도록 (300,-200,200)에 배치하자. 플레이해보자. 공중에 뜬 채로 그대로 있을 것이다.



3. **CubeLeft**를 선택한 상태에서 **디테일** 탭을 살펴보자. **트랜스폼**의 **모빌리티**가 **스태틱**으로 되어 있을 것이다. 스태틱으로 되어 있는 액터는 절대 움직이지 못한다. 일단 그대로 두자.
디테일 탭에서 **피직스** 영역에 있는 **피직스 시뮬레이트**(**Simulate Physics**) 속성을 찾아보자. 디폴트로 체크해제되어있을 것이다. 이곳을 체크하자. 이 속성을 체크하면 물리 엔진에 이 액터를 제어한다. **모빌리티** 속성도 동시에 **무버블**로 바뀐다.



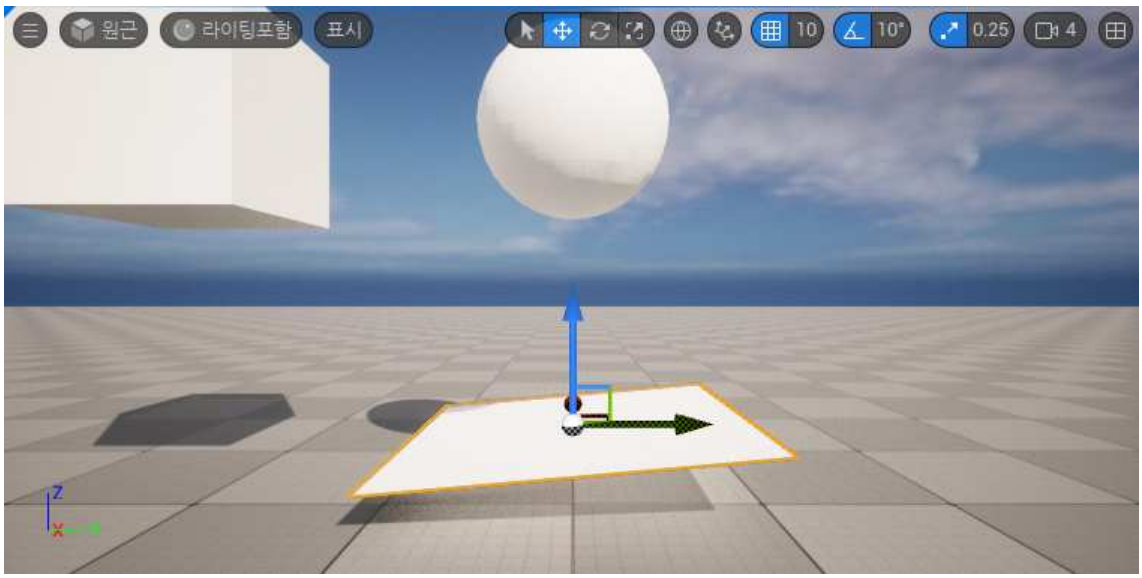
플레이해보자. 공중에 떠있는 큐브가 바닥으로 떨어질 것이다.

<참고> **피직스 시뮬레이트(Simulate Physics)** 속성에 체크하면, 몇가지 다른 속성도 동시에 바뀐다. 먼저 모빌리티 속성이 **스태틱**에서 **무버블**로 바뀐다. 물리 엔진의 영향을 받아 이동될 수 있도록 하기 위함이다. 또한, 콜리전 프리셋이 **Default**에서 **PhysicsActor**로 바뀐다. **Default**는 스태틱 메시 애셋에 지정된 기본 콜리전 프리셋을 사용하라는 의미이다. 스태틱 메시 애셋에서는 일반적으로 **BlockAll**로 지정하고 있다.
BlockAll 프리셋과 **PhysicsActor** 프리셋은 모두 **블록**으로 다른 채널 객체에 콜리전 응답하는 것은 동일하지만 **BlockAll** 프리셋을 사용하는 오브젝트 채널은 **WorldStatic**이지만 **PhysicsActor**의 오브젝트 채널은 **PhysicsBody**인 점이 다르다.

또한, 액터 영역의 속성 **Can be Damanged**도 체크해제 상태에서 체크 상태로 된다. 이 액터가 물리적은 충격으로부터 데미지 이벤트를 받을 수 있게 된다.

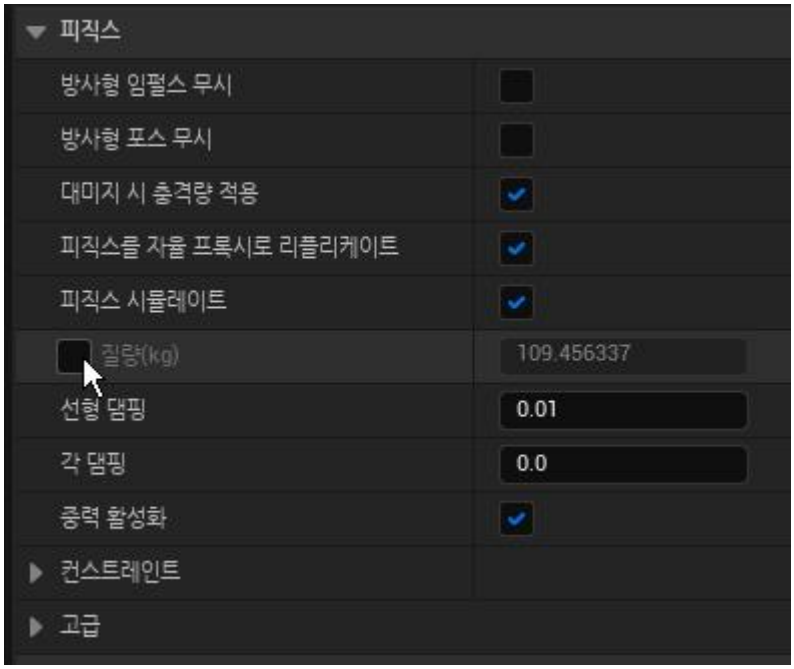
4. 툴바의 **액터 배치** 아이콘을 클릭하고 **셰이프** 아래의 **스피어**를 드래그하여 레벨에 배치하자. 배치된 액터의 인스턴스 이름은 **SphereMiddle**로 수정하자. **위치**는 공중에 떠 있도록 (300,0,200)에 배치하자. 그리고, 물리 엔진이 적용되도록 **피직스** 영역에 있는 **피직스 시뮬레이트**(**Simulate Physics**) 속성에 체크하자.

그다음, 툴바의 **액터 배치** 아이콘을 클릭하고 **셰이프** 아래의 **평면**을 드래그하여 레벨에 배치하자. 이름은 **PlaneLower**로 수정하자. 위치는 (350,0,30)로 하고, 회전은 (-5,0,0)으로 하고, 스케일은 (2,2,2)로 하자.



플레이해보자. 구체가 경사면을 내려온 후에 큐브에 부딪히며 정지하는데 이때, 큐브가 충돌에 반응함을 알 수 있다.

5. **아웃라이너**에서 **SphereMiddle** 액터를 선택하고 **디테일** 탭에서 **피직스** 영역을 살펴보자.



속성 **질량(kg)**이 회색으로 표시되어 있을 것이다. 이 물체의 질량이 자동으로 계산되었고 그 값이 약 109kg임을 의미한다. 물리 엔진이 물체의 질량을 자동으로 계산한 것이다. 질량은 **피지컬 머티리얼**에서 설정하는 **Density** 속성값과 메시의 부피를 기반으로 자동 계산된다. **피지컬 머티리얼**에 대해서는 다음 절에서 다룬다.

한편, 우리는 이 값을 직접 지정해줄 수 있다. 체크박스에 체크한 후에 입력 상자에 값을 입력하면 된다. 우리는 매우 큰 값인 10000의 값을 입력해보자.

플레이해보자. 왼쪽 상자에 충돌한 이후헤 상자가 많이 밀려나는 것을 알 수 있다.

이제, 다시 **질량(kg)**에 체크하여 원래대로 되돌려놓자.

6. **피직스** 영역에는 **질량(kg)** 외에도 많은 물리 속성들이 있다.

대표적으로, **중력 활성화**(**EnableGravity**) 속성이 있다.

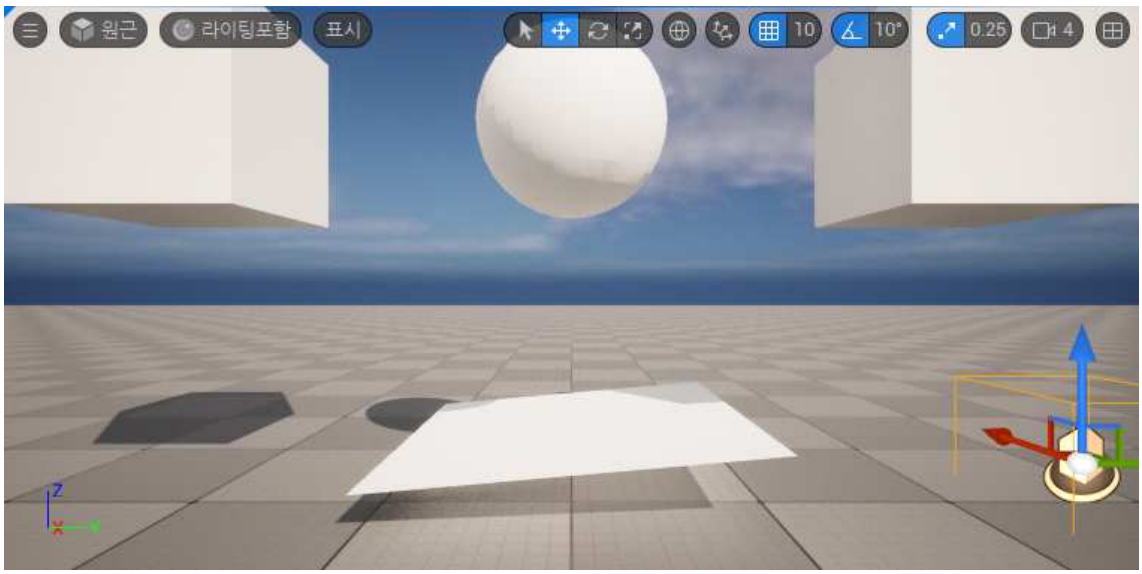
이 속성은 중력을 적용할 지를 결정한다. 디폴트로 체크되어 있다. 만약 체크되어 있지 않다면 공중에 떠 있는 물체도 떨어지지 않는다. 우주공간을 무대로 하는 게임일 경우에는 체크해제하면 된다.

피직스 영역 아래의 **고급** 속성을 펼쳐보자. 많은 물리 속성이 있다.

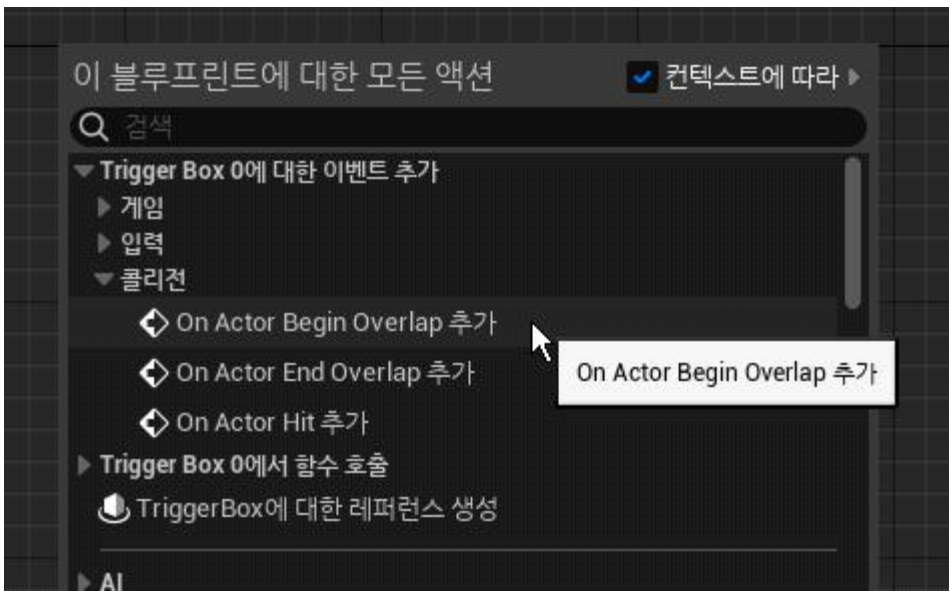
그 중의 하나로, **깨우기 시작**(**Start Awake**) 속성이 있다.
이 속성에 체크되어 있으면 레벨이 시작될 때에 또는 액터가 스폰될 때에 피직스 시뮬레이션을 바로 시작한다. 디폴트로 체크되어 있다. 따라서 공중에 떠 있는 물체나 굴러다니고 있는 물체는 플레이 시작과 동시에 떨어지거나 구르기 시작한다.
만약 체크되어 있지 않다면 스폰된 이후에 그대로 움직이지 않고 있으면서 피직스 시뮬레이션이 적용되지 않고 대기 상태로 있게 된다. 이후에 액터에 추가적인 물리적인 힘을 가해야 피직스 시뮬레이션이 작동하기 시작한다. 게임 시작 시점에서 마찰이 있는 표면 상에서 정지하고 있어야 하는 물체에 대해서는 체크해제해두어야 한다.
그리고, **Center Of Mass Offset** 속성이 있다.
이 속성은 액터의 무게 중심을 지정한다. 액터의 무게 중심은 메시의 모양으로부터 자동으로 계산되므로 대부분의 경우는 그대로 두면 된다.
한편 밀도가 균일하지 않은 물체인 경우라면 무게 중심을 임의로 이동시킬 수 있다. 위에는 비어있고 아래쪽에는 무겁게 차이는 볼륨의 경우에 무게중심을 아래로 내릴 수 있다. 오뚝이의 움직임처럼 무게중심이 바뀌면 그에 따라 움직임도 바뀌게 된다. 움직임이 이상하게 보일 수도 있으니 꼭 필요한 경우에만 수정하자.

7. 이제부터, 물리 엔진이 아닌 콜리전 이벤트에 대해서 학습해보자.
물리 엔진으로 움직이는 물체도 콜리전으로 인한 이벤트를 일으킬 수 있다.
툴바의 **액터 배치** 아이콘을 클릭하고 **셰이프** 아래의 **큐브**를 드래그하여 레벨에 배치하자. 배치된 액터의 인스턴스 이름은 **CubeRight**로 수정하자. 위치는 공중에 떠 있도록 (300,200,200)에 배치하자.
그리고, **피직스** 영역의 **피직스 시뮬레이트**(**Simulate Physics**) 속성에 체크하자.
그다음, 툴바의 **액터 배치** 아이콘을 클릭하고 **기본** 아래의 **트리거 박스**를 드래그하여 배치하자. 이름은 **TriggerBox**로 그대로 두자. 위치는 (300,270,20)에 배치하자.

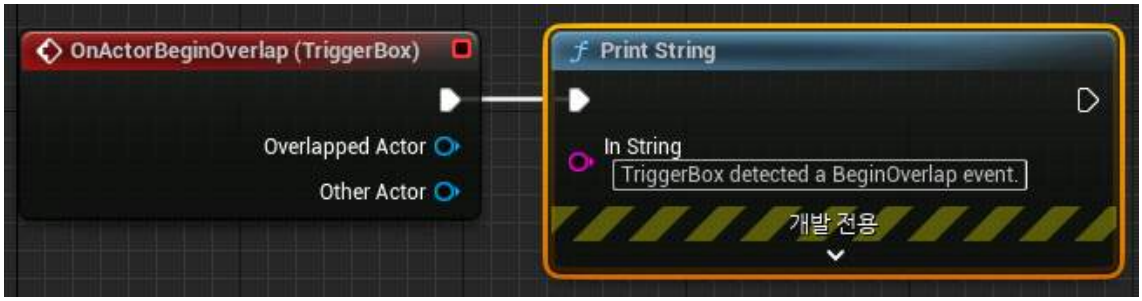
그리고, **디테일** 탭에서 **렌더링** 영역의 **액터 게임에서 숨김(Actor Hidden In Game)**에 체크되어 있는 것을 해제하자. 해제되면 플레이 도중에도 트리거 박스의 윤곽이 보이게 된다.
이제, 레벨을 살펴보자. **TriggerBox**가 더 오른쪽에 치우쳐서 배치되어 있어서 **CubeRight**가 떨어지면 **TriggerBox**의 왼쪽 일부로만 겹칠 것이다.



8. **아웃라이너**에서 **TriggerBox**를 클릭하여 선택하자.
그다음, 레벨 블루프린트 에디터를 열자.
이벤트 그래프 탭의 격자판의 빈 곳에서 우클릭하고 액션선택 창에서 **TriggerBox0에 대한 이벤트 추가** 아래의 **콜리전** 아래의 **OnActorBeginOverlap 추가**를 선택하자. 레벨 블루프린트에 **OnActorBeginOverlap (TriggerBox)** 이벤트 노드가 추가될 것이다.

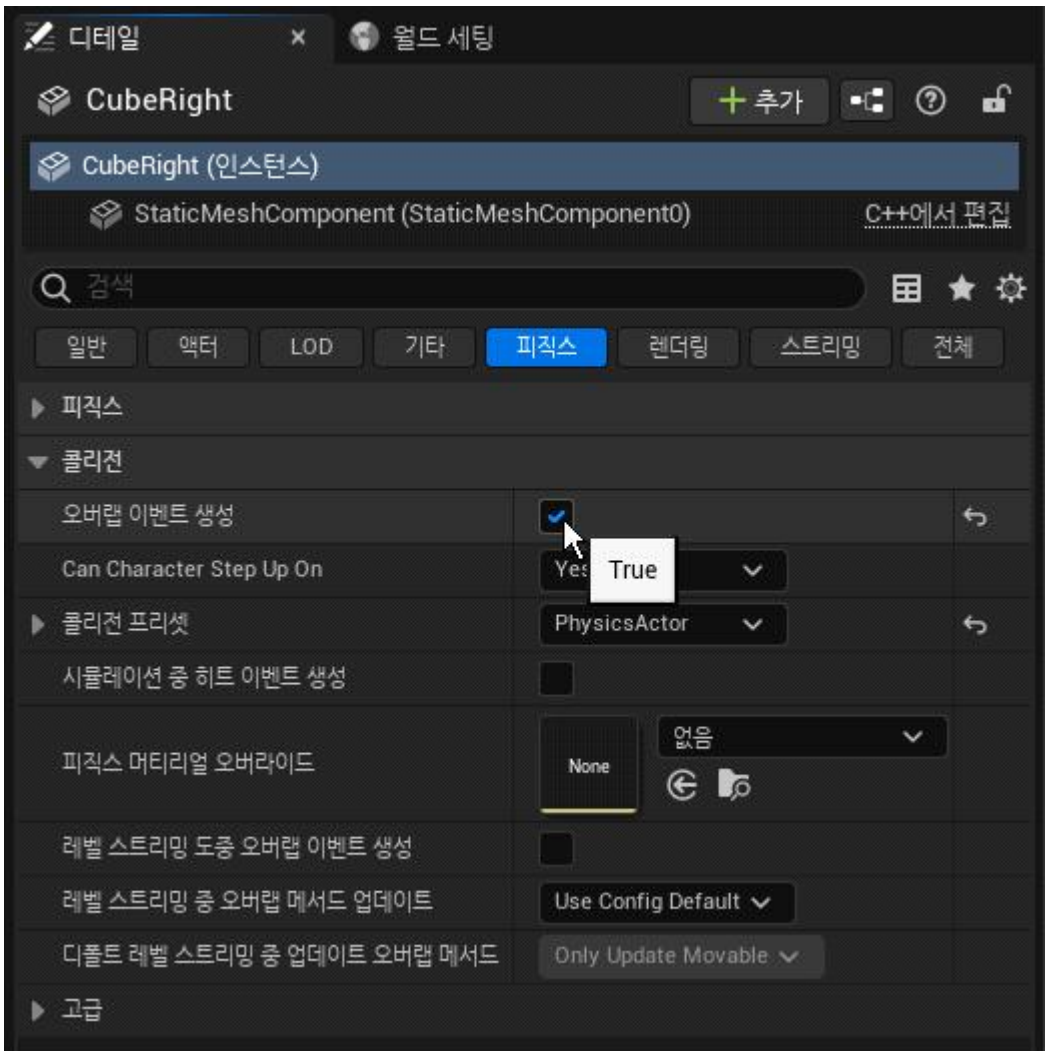


9. 배치된 이벤트 노드에서 **PrintString** 노드를 추가하여 '`TriggerBox detected a BeginOverlap event.`' 문자열을 출력해보자.



플레이해보자. 문자열이 출력되지 않을 것이다.

10. 문자열이 출력되지 않는 이유를 알아보자. 엔진은 계산량을 줄이기 위해서 레벨에 배치된 일부 액터에 대해서만 겹침 발생 여부를 검사한다. 즉 겹침 이벤트를 발생시킬 수 있는 액터들을 따로 모아두고 그들에 대해서만 겹침 이벤트가 발생했는지의 여부를 확인한다.
아웃라이너에서 **CubeRight** 액터를 선택하고 디테일 탭에서 콜리전 영역의 **오버랩 이벤트 생성(Generate Overlap Events)** 속성을 찾아보자. 체크되어 있지 않을 것이다.

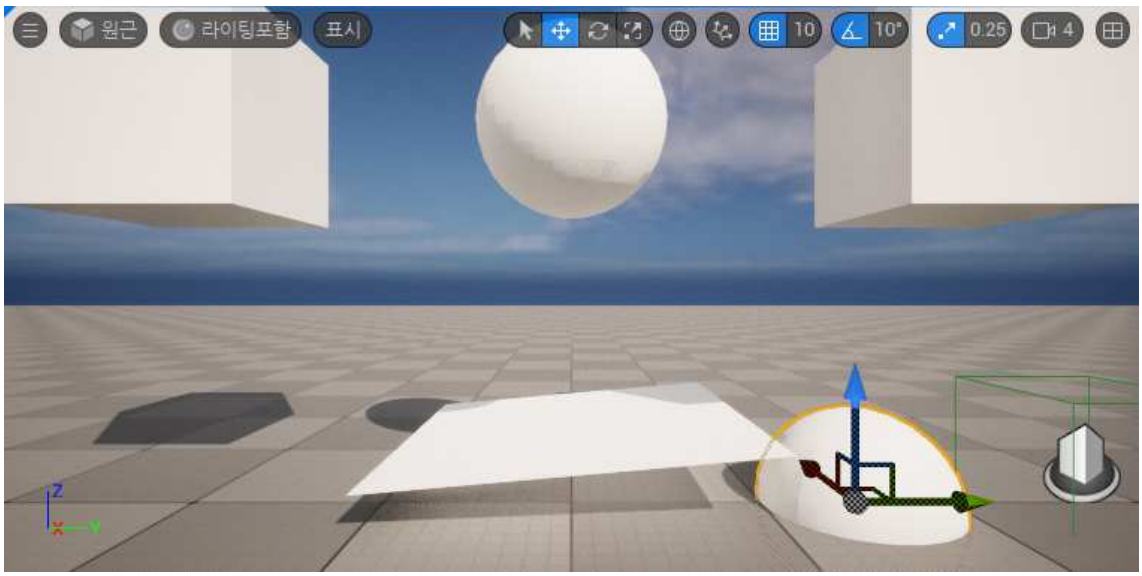


스태틱 메시 액터를 레벨에 배치하면 에디터는 디폴트로 **오버랩 이벤트 생성(Generate Overlap Events)** 속성이 비활성되도록 하여 배치한다. 엔진은 스태틱 메시 액터를 상호작용성이 없는 배경용으로

생각하는 것이다.
우리는 **오버랩 이벤트 생성**(**Generate Overlap Events**) 속성에 체크하여 활성으로 바꾸자.

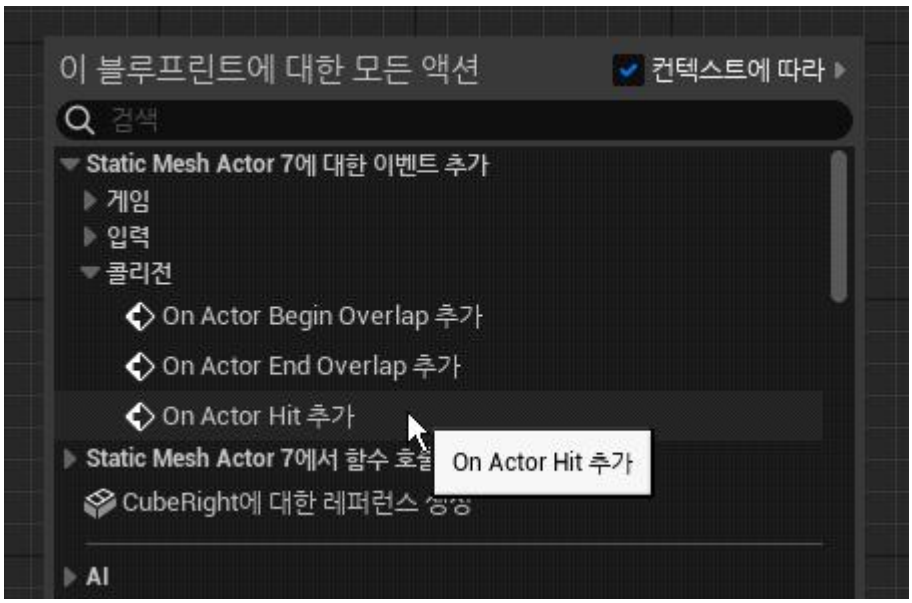
11. 다시 플레이해보자. 이제, 문자열이 출력될 것이다.

12. 이제부터, **히트** 이벤트에 대해서도 알아보자.
툴바의 **액터 배치** 아이콘을 클릭하고 **셰이프** 아래의 **스피어**를 드래그하여 레벨에 배치하자. 배치된 액터의 인스턴스 이름은 **SphereLower**로 수정하자. 위치는 절반만 바닥위에 노출되도록 (300,150,0)에 배치하자.
다른 것은 그대로 두자. 즉, 이 액터는 고정되어 있는 스태틱 메시이고 물리 엔진의 영향을 받지 않는 메시로 생각하자.



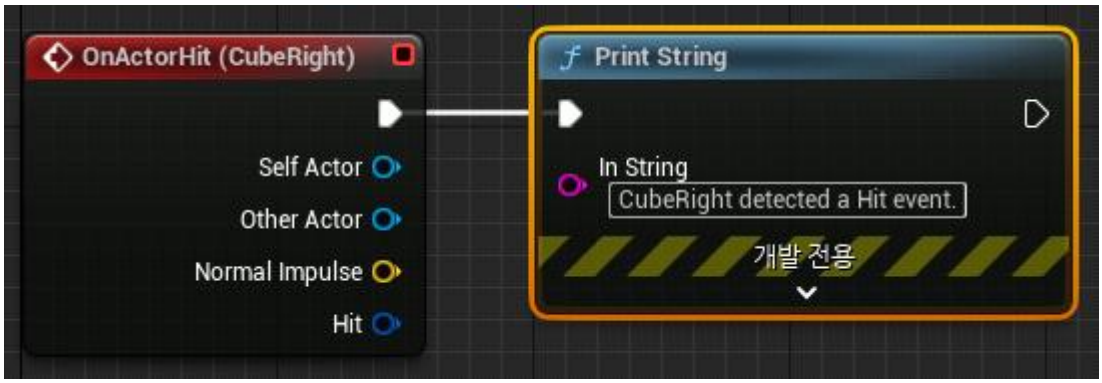
13. **아웃라이너**에서 **CubeRight**를 클릭하여 선택하자.
그다음, 레벨 블루프린트 에디터로 가자.
그리고, 이벤트 그래프 탭의 격자판의 빈 곳에서 우클릭하고 액션선택 창에서 **StaticMeshActor7에**

대한 이벤트 추가 아래의 **콜리전** 아래의 **OnActorHit 추가**를 선택하자.



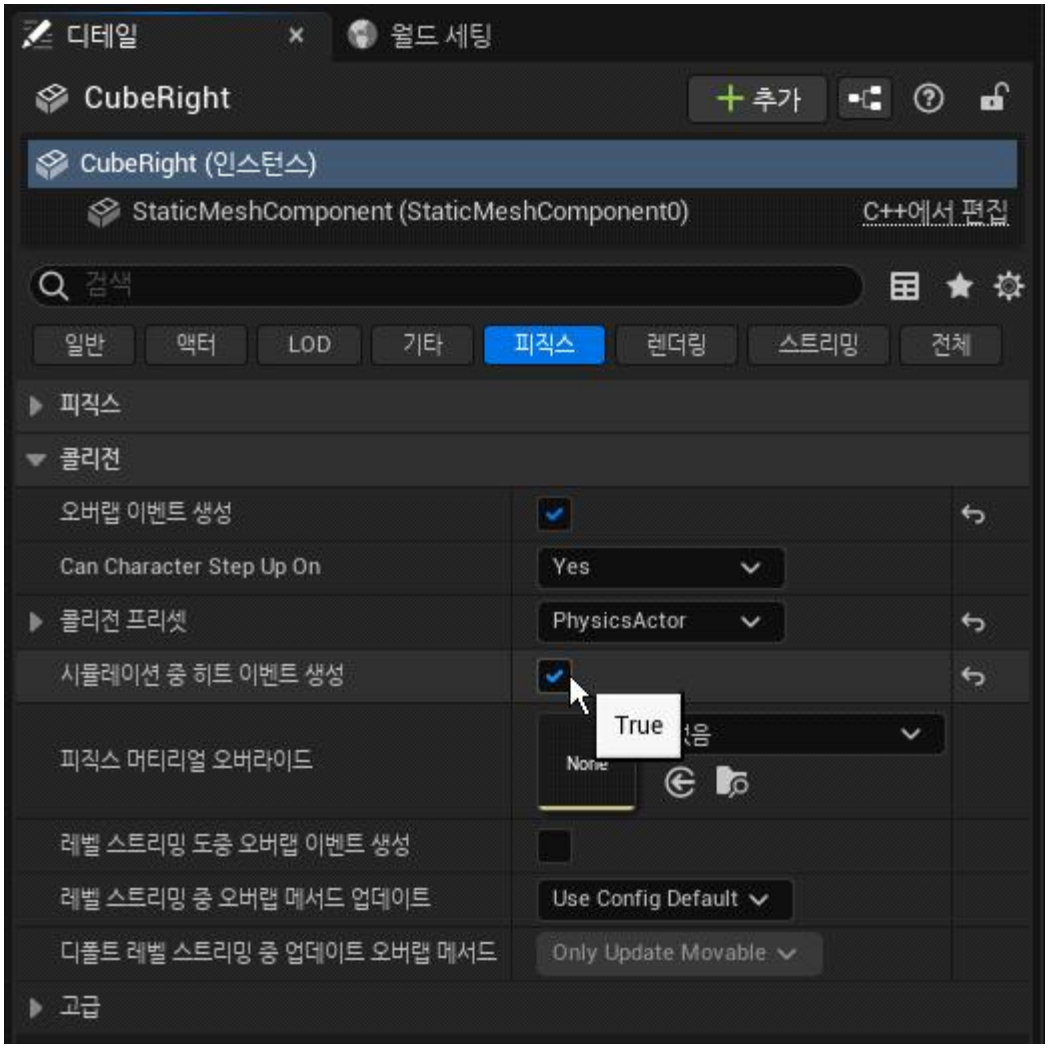
레벨 블루프린트에 **OnActorHit (CubeRight)** 이벤트 노드가 추가될 것이다.

14. 배치된 이벤트 노드에서 **PrintString** 노드를 추가하여 '`CubeRight detected a Hit event.`' 문자열을 출력해보자.



플레이해보자. 문자열이 출력되지 않을 것이다.

15. 문자열이 출력되지 않는 이유는 이전의 겹침 이벤트의 경우와 동일하다. 대부분의 경우 히트 이벤트는 디폴트로 비활성으로 되어 있다. 따라서 히트 이벤트의 발생을 원하는 액터에 대해서 명시적으로 활성으로 바꾸어주어야 한다.
아웃라이너에서 **CubeRight** 액터를 선택하고 디테일 탭에서 콜리전 영역의 **시뮬레이션 중 히트 이벤트 생성**(**Simulation Generates Hit Events**) 속성을 찾아보자. 체크되어 있지 않을 것이다. 우리는 체크하여 활성으로 바꾸자.

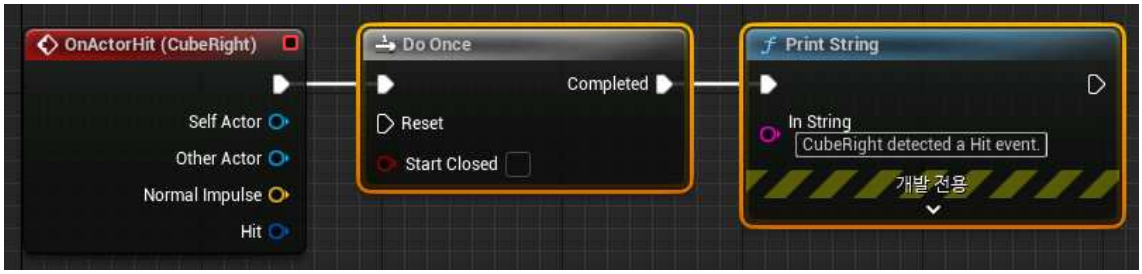


플레이해보자. **OnActorHit (CubeRight)** 이벤트 그래프가 호출되어 문자열이 출력되는 것을 확인할 수 있다.

한편, 겹침 이벤트에서와는 다른 차이점이 있다.

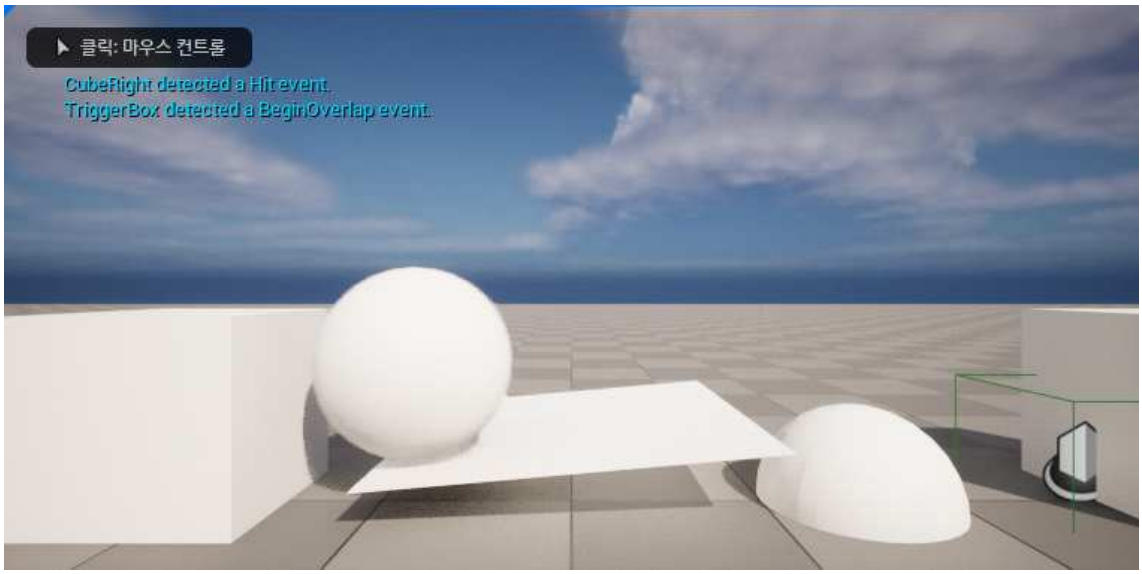
자신의 **OnActorHit** 이벤트 그래프가 호출되도록 하려면, 자신의 **시뮬레이션 중 히트 이벤트 생성(Simulation Generates Hit Events)**를 활성화해야한다. 이것을 활성화하면 이 액터는 물리 시뮬레이션으로 발생되는 히트 이벤트를 받을 수 있는 구독자 명단에 추가된다. 즉 자신이 다른 액터와 충돌하거나 충돌당한 경우에 자신의 이벤트 함수가 호출된다. 다만 트랜스폼이 스태틱으로 되어 있는 경우에는 **시뮬레이션 중 히트 이벤트 생성(Simulation Generates Hit Events)**를 활성화해도 무시된다.

16. 물리 시뮬레이션으로 움직이는 **CubeRight** 액터에 대해서는 많은 히트 이벤트가 발생한다. 출력도 그만큼 많이 반복될 것이다. 알아보기 쉽게 하기 위해서 다음과 같이 한번만 출력하도록 수정하자.

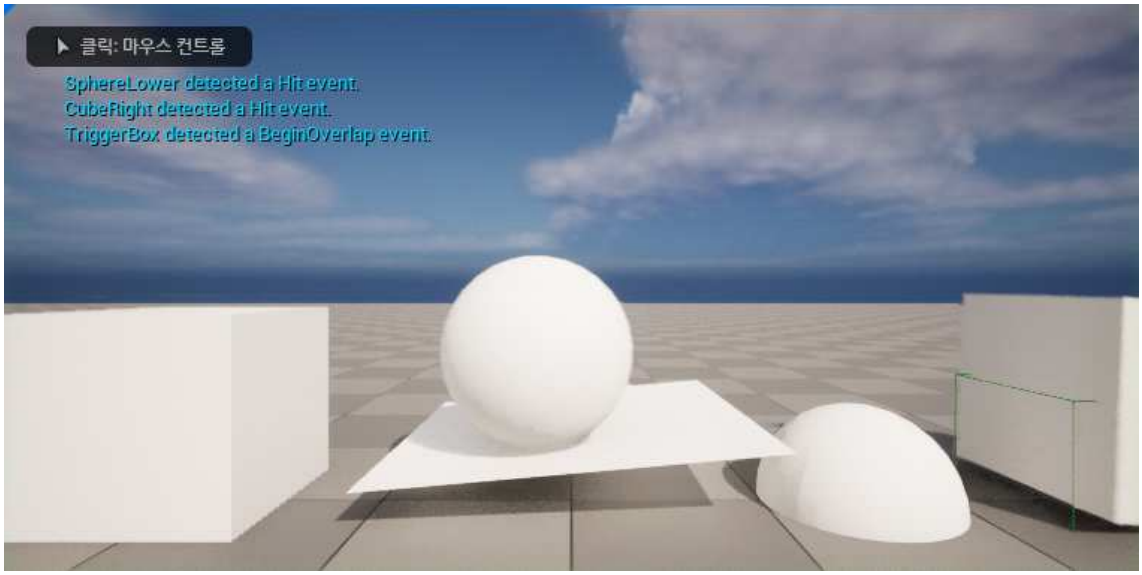


DoOnce 노드는 실행핀의 흐름이 한 번만 흐르도록 하고 그 이후에는 흐름이 전달되지 못하도록 막는다.

17. 플레이해보자. 이제 한 번만 출력된다.



18. 이제 물리 시뮬레이션의 적용을 받지 않는 액터에 대해서도 자신의 **OnActorHit** 이벤트 그래프가 호출되도록 해보자.
아웃라이너에서 **SphereLower**를 클릭하여 선택하자.
그다음, 레벨 블루프린트 에디터로 가자.
그리고, 이벤트 그래프 탭의 격자판의 빈 곳에서 우클릭하고 액션선택 창에서 **StaticMeshActor8에 대한 이벤트 추가** 아래의 **콜리전** 아래의 **OnActorHit 추가**를 선택하자.
레벨 블루프린트에 **OnActorHit (SphereLower)** 이벤트 노드가 추가될 것이다.
그다음, 배치된 이벤트 노드에서 **PrintString** 노드를 추가하여 '`SphereLower detected a Hit event.`' 문자열을 출력해보자. **DoOnce** 노드도 이전처럼 **PrintString** 노드 앞에 추가하자.
그다음, **SphereLower**의 **디테일** 탭으로 가자.
그리고, **콜리전** 영역의 **시뮬레이션 중 히트 이벤트 생성**(**Simulation Generates Hit Events**)에 체크하자.
플레이해보자. **OnActorHit (SphereLower)** 이벤트 그래프가 호출되어 문자열이 출력되는 것을 확인할 수 있다.



이 절에서는 물리 엔진을 처음으로 사용해보는 방법에 대해서 학습하였다.

3. 피지컬 머티리얼을 만들고 사용하기

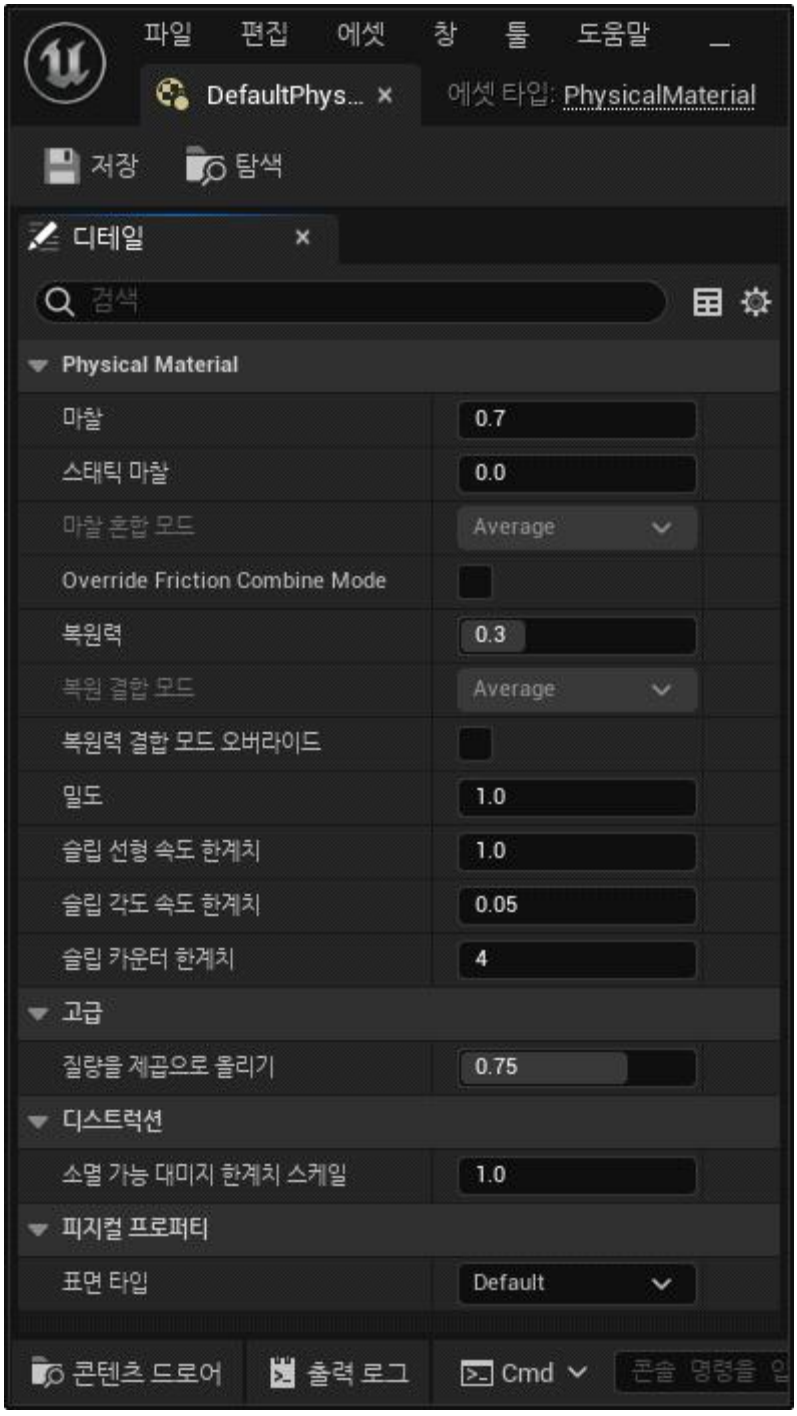
이 절에서 피지컬 머티리얼에 대해서 학습한다.

이번 절에서는 **피지컬 머티리얼**에 대해서 학습한다.
피지컬 머티리얼(physical material)은 물리 시뮬레이션에 필요한 물리 속성을 정의하는 전용 애셋이다. **피지컬 머티리얼** 애셋을 스태틱 메시 등의 프리미티프 컴포넌트에 적용하면 해당 애셋은 **피지컬 머티리얼** 애셋에서 정의하는 물리 속성을 가지게 된다.
또한 **피지컬 머티리얼**을 머티리얼에 지정해두고 해당 머티리얼을 컴포넌트에 적용하는 방법으로도 사용할 수 있다.
피지컬 머티리얼은 독립된 애셋이다. 따라서 **피지컬 머티리얼**을 사용하면 각 액터마다 일일이 물리 속성을 지정해야 할 필요 없이 여러 유사한 물리 속성을 가지는 액터에 해당 **피지컬 머티리얼**을 적용하면 된다.

물리 엔진이 액터의 움직임을 제어할 때에 액터의 물리 속성을 이용한다. 여러 액터의 물리 속성은 **피지컬 머티리얼**에서 정의되어 있다. 이전 절에서 액터의 **피직스** 속성 중에서 **질량(kg)**은 **Density** 속성값을 사용하여 자동으로 계산된다고 하였다. **Density** 속성값은 **피지컬 머티리얼**에서 정의하는 물리 속성 중의 하나이다.

<참고> 피지컬 머티리얼에 대한 자세한 내용은 다음의 문서를 참조하자.
https://docs.unrealengine.com/tutorials-about-physical-materials-in-unreal-engine/
다음의 문서도 도움이 될 것이다.
https://hub.packtpub.com/overview-physics-bodies-and-physics-materials/

피지컬 머티리얼 애셋의 에디터를 살펴보자. **피지컬 머티리얼** 애셋은 단순히 물리 시뮬레이션을 위한 속성값들을 정의하는 애셋이다.
아래의 **피지컬 머티리얼**은 **엔진** » **콘텐츠** » **EngineMaterials** 아래에 있는 **DefaultPhysicalMaterial** 애셋의 모습이다. 특별히 지정하지 않은 경우에 사용되는 디폴트 애셋이다.



피지컬 머티리얼 애셋에서 정의하는 각 속성에 대해서 잘 이해해야 한다. 우리는 이제부터 물리에서의 마찰력, 복원력, 질량에 대해서 각 속성과 연관지어서 학습해보자.

먼저, 마찰력에 대해서 알아보자. 마찰력은 두 물체의 접촉면 사이에서 물체의 운동을 방해하는 힘이다.

피지컬 머티리얼 애셋에는 속성 **Friction**이 있다. 이 속성은 마찰 계수(coefficient of friction)이다. 보통 [0,1]의 값을 지정한다. 0이면 마찰력이 없고 1이면 마찰력이 매우 크다. 마찰력이 작을수록 강체를 움직이기가 쉬우지며 한 번 움직이면 잘 멈추지 않는다. 반대로, 마찰력이 클수록 강체를 움직이기가

어려워지며 움직이다가도 쉽게 멈춘다.
이 값은 또한, 바운스 뒤에 탄젠트 방향으로의 속도가 유지되는 비율에도 영향을 준다.
속성값은 보통은 1을 넘기지 않지만 1을 초과하는 실수값을 지정하여 마찰력을 더 크게 할 수도 있다.
그다음, 속성 **Static Friction**이 있다. 이 속성은 표면에서 강체가 얼마나 잘 미끄러지는 지를 제어한다. 0은 마찰력이 없음을 의미하고 1은 큰 마찰력을 의미하고 1보다 더 큰 실수값을 지정할 수도 있다.

<참고> 물리학에서의 마찰력은 정지 마찰력과 운동 마찰력의 두 경우를 나누어 고려하고 둘 중 하나만 적용되는 것으로 생각하지만 게임 엔진에서는 이와 다르다. **Friction**와 **Static Friction**의 두 속성값을 함께 사용하고 있다. 보통은 **Friction**만을 지정하고 이 값이 정지 마찰력과 운동 마찰력 모두에게 사용된다. **Static Friction**의 디폴트 값은 0이어서 디폴트는 **Friction**에만 의존하여 마찰력을 결정한다. 만약 경사면에 대해서 잘 미끄러지지 않도록 경사면의 마찰력을 높이고자 하는 경우에는 속성 **Static Friction**을 사용하면 된다.

그다음, **Override Friction Combine Mode**가 있다. 이 속성은 **Friction Combine Mode**를 재정의한다.
Override Friction Combine Mode 체크박스에 체크하면 바로 위의 **Friction Combine Mode** 속성을 수정할 수 있다.
Friction Combine Mode란 두 마찰면의 마찰력을 조합하는 방법을 명시한다.
마찰은 두 마찰면이 접촉할 경우에 발생하는 힘이다. 따라서 두 마찰면의 특성에 모두 영향을 받으면서 최종 마찰력이 결정된다. 물리 엔진에서는 두 마찰면의 마찰 계수를 간단하게 조합하는 방법을 제공한다. 조합 방법으로 **Average**, **Multiply**, **Min**, **Max**가 있다.
Average는 두 강체의 마찰 계수의 평균값을 사용한다. 디폴트로 사용되는 일반적인 방법이다.
Multiply는 두 강체의 마찰 계수를 곱한 값을 사용한다. 마찰 계수는 보통 1보다 작으므로 곱하면 더 작은 값이 된다. 따라서 이 방법을 사용하면 마찰력이 더 작게 된다. 보통, 마찰력을 0으로 하든지 또는 1보다 더 큰 값으로 하여 마찰력을 없애든지 또는 더 크게 하는 게임에서의 특별한 공간을 만드는 목적으로 사용한다.
Min은 두 강체의 마찰 계수의 작은 값을 사용한다. 마찰력을 가급적 줄이는 것을 원할 때에 사용한다. 예를 들어 진흙길에서도 자동차가 움직일 수 있도록 하기 위해서 사용할 수 있다.
Max는 큰 값을 사용한다. 마찰력을 가급적 크게 하는 것을 원할 때에 사용한다. 빙판길에서 자동차가 너무 쉽게 움직이지 않도록 하기 위해서 사용할 수 있다.
Override Friction Combine Mode의 디폴트 값은 **Average**이다. 디폴트는 프로젝트 속성에서 지정할 수 있다.
프로젝트 세팅 창을 열고 **엔진** » **피직스** 탭에 가면 **시뮬레이션** 영역에 **마찰 혼합 모드**가 **Average**로 되어 있을 것이다. 이 값이 디폴트 값으로 사용된다.

이제부터, 반발력에 대해서 알아보자. 반발력은 충돌이 발생할 때에 두 물체가 서로 반발하는 힘에 해당하며 표면의 특성에 따른 속도의 변화와 관련된다.
피지컬 머티리얼 애셋에는 **Restitution** 속성이 있다. 이 속성은 표면의 **반발력**을 제어하는 **반발 계수**(coefficient of restitution)이다. 반발력을 **복원력**이라고도 한다. 탁구공이 탁구대 표면에 떨어질 때에는 잘 튀어오르지만 진흙바닥에 떨어진다면 잘 튀어오르지 않을 것이다. 두 표면의 반발 계수가 다르게 때문이다.
Restitution 값은 0에서 1 사이의 값을 가지고 1을 초과할 수 없다.
0은 충돌 후에 다시 역방향으로 바운스되지 않고 표면에 달라붙어서 움직이게 된다.

1은 입력 속도와 동일한 출력 속도로 역방향으로 바운스된다.
Restitition Combine Mode나 **Override Restitution Combine Mode**는 마찰력의 경우와 동일하게 사용하면 된다. 복원 결합 모드의 디폴트 값도 **Average**이다.

이제부터, 물체의 질량에 대해서 알아보자.
물체의 질량은 물리 시뮬레이션에서 중요하게 작용한다.
한 예로, 마찰력은 마찰 계수에 물체의 질량을 곱하여 결정된다. 질량이 다른 두 상자가 동일 표면위에 있다면 가벼운 상자는 마찰력이 작아서 잘 미끄러지고 무거운 상자의 마찰력이 커서 잘 미끄러지지 않을 것이다.
따라서 물리 시뮬레이션을 위해서는 각 물체에 대해서 물체의 질량을 알아야 한다. 한편, 레벨에 배치된 많은 물체에 대해서 각각마다 질량을 별도로 입력하는 것은 매우 번거로운 일이다. 이를 위해서 게임 엔진은 각 물체의 질량을 자동으로 계산해주고 있다.

피지컬 머티리얼에는 **밀도**(density) 속성이 있다. **밀도**는 단위 부피당 질량을 정의한다. **밀도**가 1인 경우는 1cm^{3}당 1g에 해당한다. 이 값을 사용하면 물체의 질량을 계산할 수 있다. 물체의 질량은 밀도와 부피를 곱한 값이다. 밀도값이 높을수록 질량이 무겁게 되도록 한다.
게임 엔진은 물체의 부피를 계산하고 이 밀도 속성값을 곱하여 질량을 자동으로 계산한다.

<참고> 한편, 보통은 물체의 내부가 꽉 차있는 경우가 드물기 때문에 부피가 커질수록 질량이 덜 증가하게 조절하는 것이 자연스럽다. **고급** 영역에 **Raise Mass to Power**라는 속성이 있다. 이 값은 밀도를 반영하는 비율을 나타내며 0.1에서 1 사이의 값을 가진다. 1이라면 최대 증가를 의미하며 **밀도** 속성값을 그대로 반영하여 수식에 의한 질량이 그대로 되도록 한다. 0.1에 가까울수록 물체의 내부가 덜 차있는것처럼 밀도 속성값의 일부만 반영되도록 하여 질량이 낮아지도록 한다.
우리의 예제에서 사용한 큐브의 각 변은 100cm이다. 따라서 부피는 1,000,000cm3이고 **밀도**가 1이라면 1,000kg에 해당한다. 만약 **Raise Mass to Power** 속성값이 1이라면 밀도가 그대로 반영되어 1,000kg이 될 것이다. 그러나 디폴트 값이 0.75로 되어 있으며 이 경우 질량은 대략 178kg으로 계산된다.

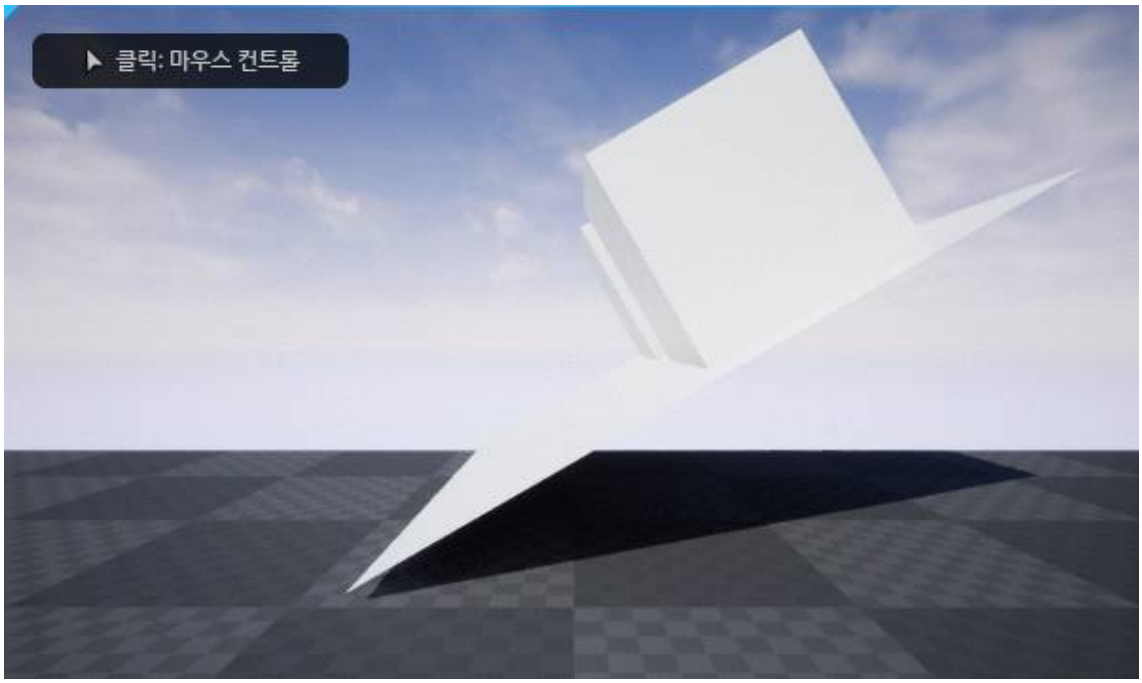
이제부터 피지컬 머티리얼에 대해서 학습해보자

---

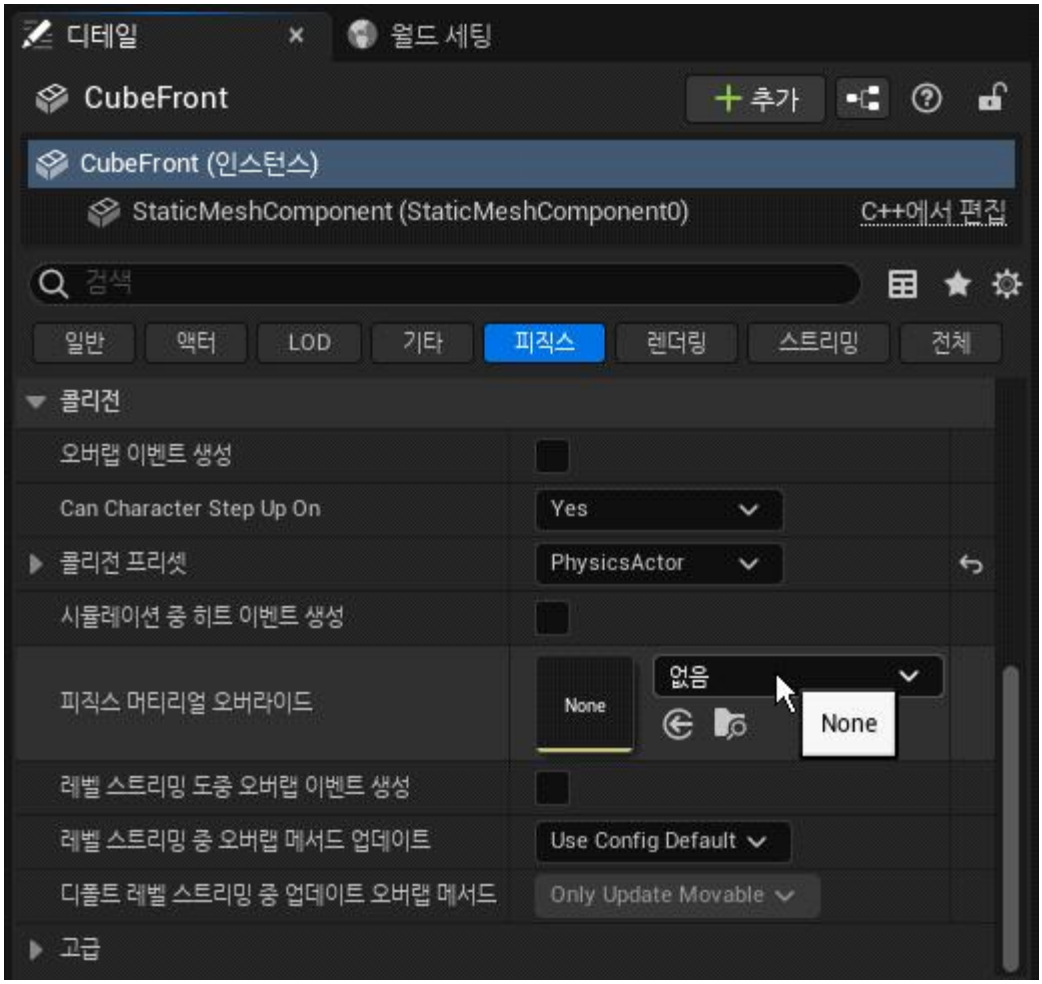
1. 새 프로젝트 `Pphysicsmaterial`를 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 `Pphysicsmaterial`로 입력하고 **생성** 버튼을 클릭하자. 프로젝트가 생성되고 언리얼 에디터 창이 뜰 것이다.
창이 뜨면, 메뉴바에서 **파일** » **새 레벨**을 선택하고 **Basic**을 선택하여 기본 템플릿 레벨을 생성하자. 그리고, 레벨 에디터 툴바의 저장 버튼을 클릭하여 현재의 레벨을 `MyMap`으로 저장하자.
그리고, **프로젝트 세팅** 창에서 **맵&모드** 탭에서의 **에디터 시작 맵** 속성값을 `MyMap`으로 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 **평면**을 드래그하여 레벨에 배치하자. 이름은 **PlaneInclined**으로 수정하자. 위치는 (350,50,100)로 하고, 회전은 (-20,0,0)으로 하고, 스케일은 (3,3,3)로 하자.

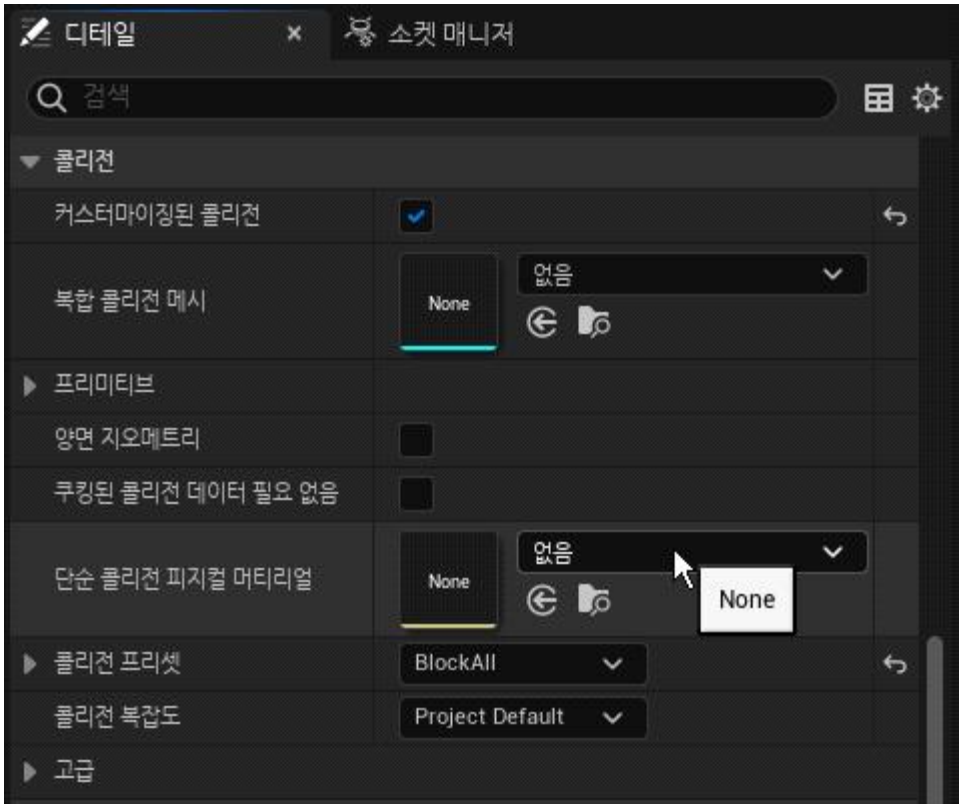
그다음, 툴바의 **액터 배치** 아이콘을 클릭하고 **셰이프** 아래의 **큐브**를 드래그하여 레벨에 배치하자. 배치된 액터의 인스턴스 이름은 **CubeFront**로 하자. 위치는 공중에 떠 있도록 (280,100,250)에 배치하자. 회전은 (-20,0,0)으로 하자.
그리고, 물리 엔진이 적용되도록 **피직스** 영역에 있는 **피직스 시뮬레이트(Simulate Physics)** 속성에 체크하자.
그다음, 툴바의 **액터 배치** 아이콘을 클릭하고 **셰이프** 아래의 **큐브**를 드래그하여 레벨에 배치하자. 배치된 액터의 인스턴스 이름은 **CubeRear**로 하자. 위치는 공중에 떠 있도록 (420,100,250)에 배치하자. 회전은 (-20,0,0)으로 하자.
그리고, 물리 엔진이 적용되도록 **피직스** 영역에 있는 **피직스 시뮬레이트(Simulate Physics)** 속성에 체크하자.
플레이해보자. 상자가 경사면에 떨어진 후에 미끄러지지 않고 멈추어 있을 것이다.



3. 이제부터 하나의 상자만 미끄러지도록 수정해보자.
CubeFront를 선택하고 **디테일** 탭의 **콜리전** 영역으로 가보자. **피직스 머티리얼 오버라이드(Phys Material Override)** 속성이 있고 값이 **없음**으로 되어 있을 것이다. 레벨에 배치된 액터 인스턴스의 속성이 가장 우선 순위가 높지만 **없음**일 경우에는 그다음의 우선 순위인 **스태틱 메시 에디터**에서 지정한 **피지컬 머티리얼**을 사용하는 것을 의미한다.



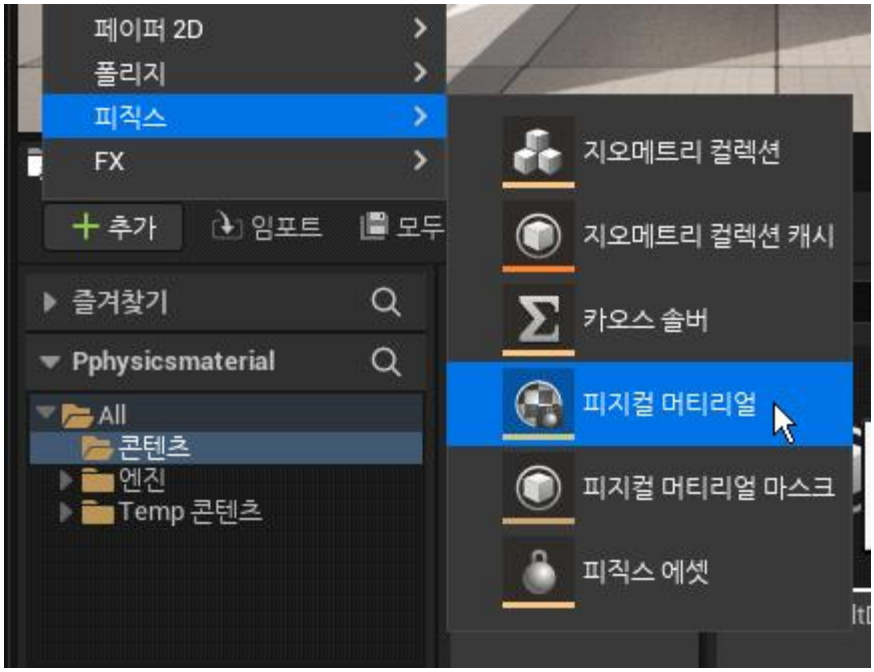
4. **아웃라이너**에 가서 **CubeFront**를 선택하고 **Ctrl+E**를 입력해보자. **스태틱 메시 에디터**가 열릴 것이다. 디테일 탭에서 콜리전 영역에서 **단순 콜리전 피지컬 머티리얼**(**Simple Collision Physical Material**) 속성을 찾아보자. 여기서도 **없음**으로 되어있을 것이다.



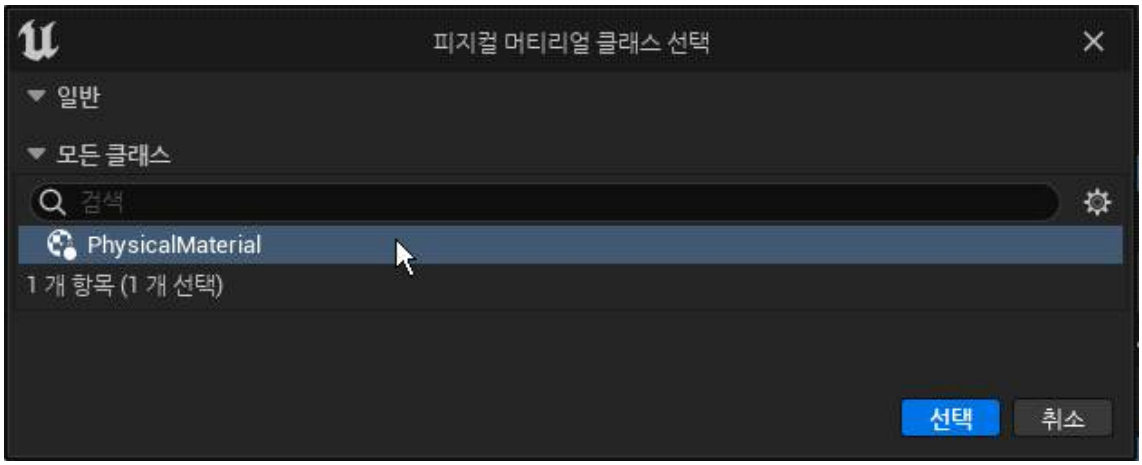
이럴 경우에는 엔진의 설정 파일에서 명시한 디폴트 **피지컬 머티리얼**을 사용한다.

<참고> 엔진의 설정 파일은 엔진의 설치 폴더 아래에서 다음과 같은 위치에 있다.
C:/Program Files/Epic Games/UE_5.0/Engine/Config/BaseEngine.ini
설정 파일의 내부에는 다음과 같은 속성이 있을 것이다.
DefaultPhysMaterialName=/Engine/EngineMaterials/DefaultPhysicalMaterial.DefaultPhysicalMaterial
즉 디폴트 **피지컬 머티리얼**로 **DefaultPhysicalMaterial** 애셋을 사용함을 알 수 있다. 이 애셋은 콘텐츠 브라우저에서 **엔진** » **콘텐츠** » **EngineMaterials** 아래로 가면 찾을 수 있다.

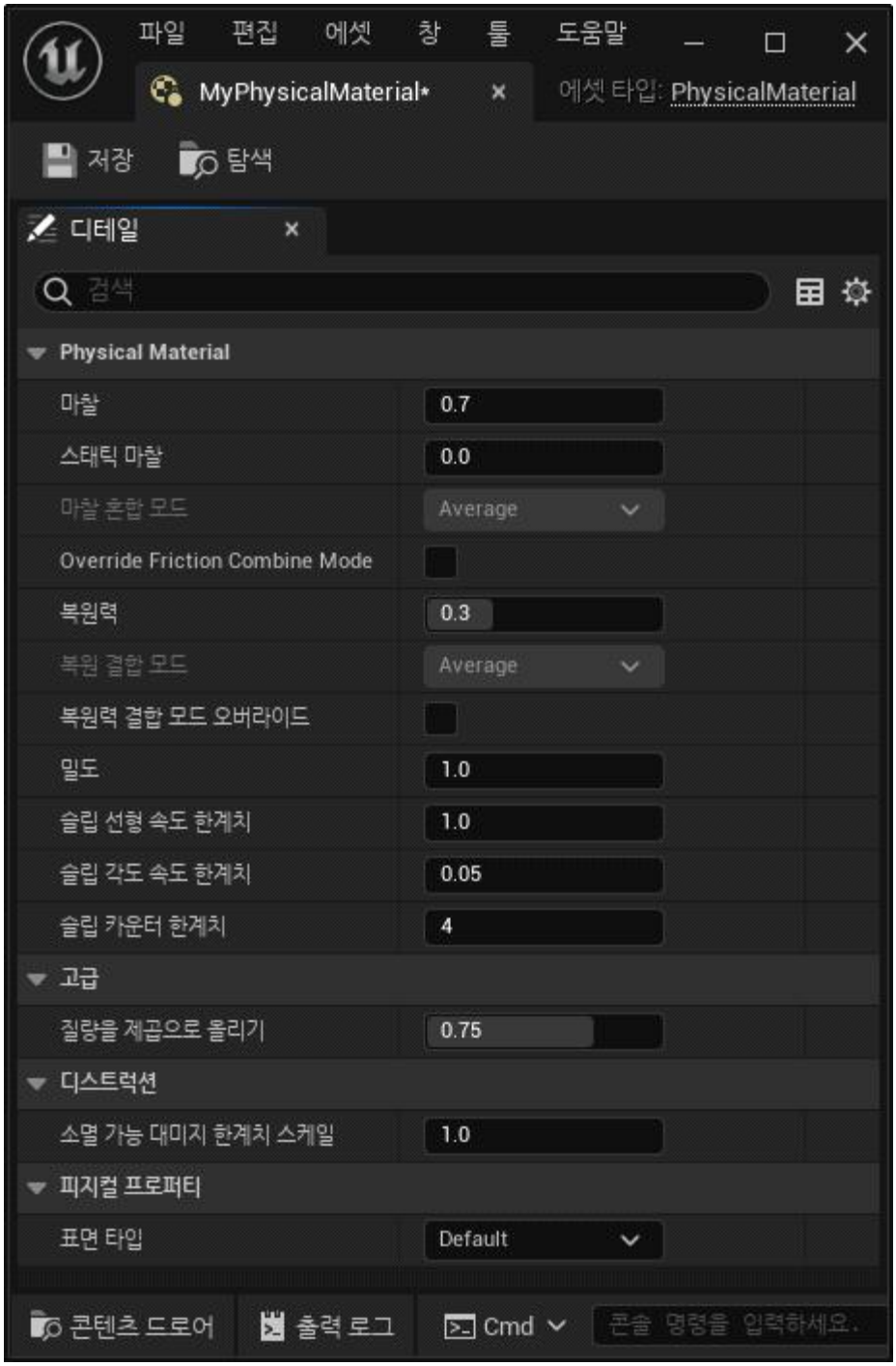
5. 이제, **피지컬 머티리얼** 애셋을 생성해보자.
콘텐츠 브라우저에서 **+추가**를 클릭하고 팝업메뉴에서 **피직스** » **피지컬 머티리얼**을 선택하자.



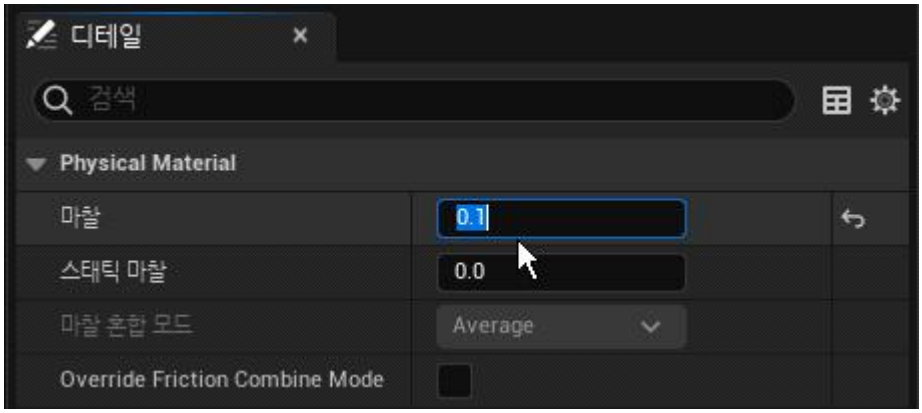
6. **피지컬 머티리얼 클래스 선택** 창이 뜬다. **PhysicalMaterial**을 선택하고 **선택**을 클릭하자. 애셋이 생성될 것이다. 생성된 애셋의 이름을 **MyPhysicalMaterial**로 지정하자.



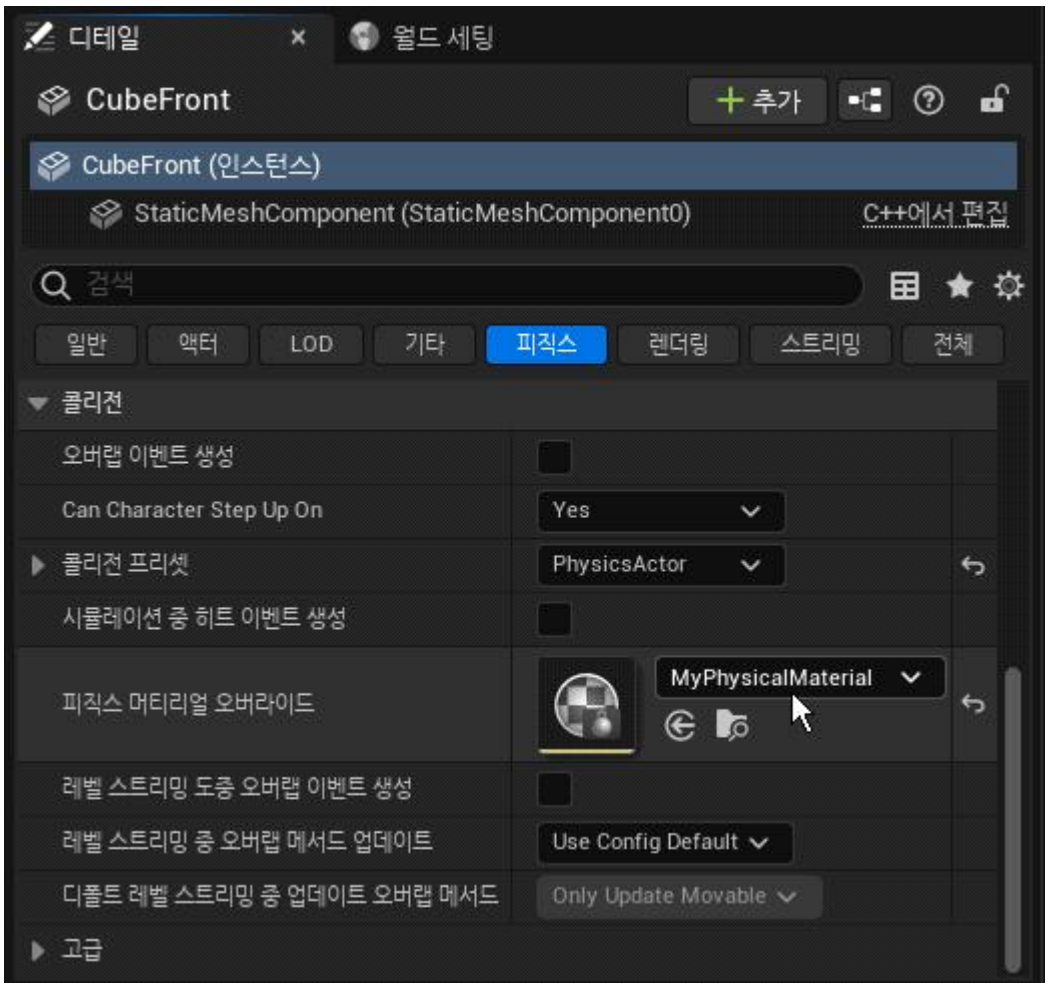
7. 생성된 **피지컬 머티리얼** 애셋인 **MyPhysicalMaterial**를 더블클릭하자. 다음과 같은 모습의 **피지컬 머티리얼** 에디터가 열릴 것이다.



8. 속성 **마찰(Friction)**은 마찰력을 제어하는 계수이다. 이 속성값을 0.7에서 0.1로 수정하여 마찰이 작아지도록 해보자. 그리고 저장하자.



9. 레벨 에디터에서 **CubeFront**를 선택하고 디테일 탭으로 가자.
콜리전 영역의 **Phys Material Override** 속성값을 **없음**에서 **MyPhysicalMaterial**로 바꾸자.



10. 플레이해보자.
이제 **CubeFront**는 더 잘 미끄러져 내려올 것이다.

이 절에서는 피지컬 머티리얼에 대해서 학습하였다.

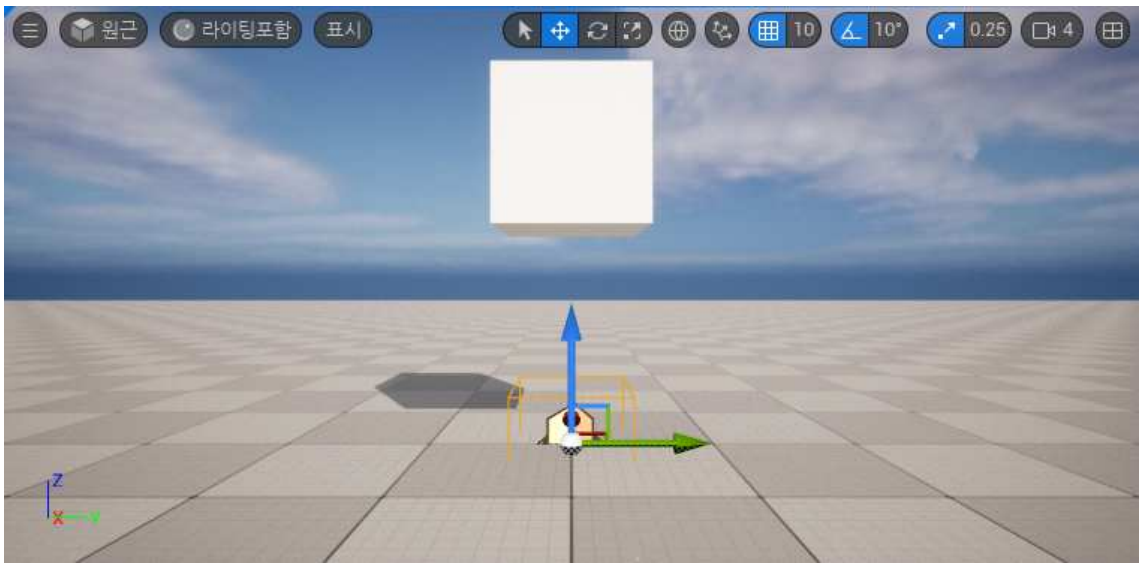
4. 동적으로 물리 상태 변경하고 힘 가하기

이 절에서 동적으로 물리 상태를 변경하고 힘을 가하는 방법을 학습한다.
이제부터 예제를 통해서 학습해보자

---

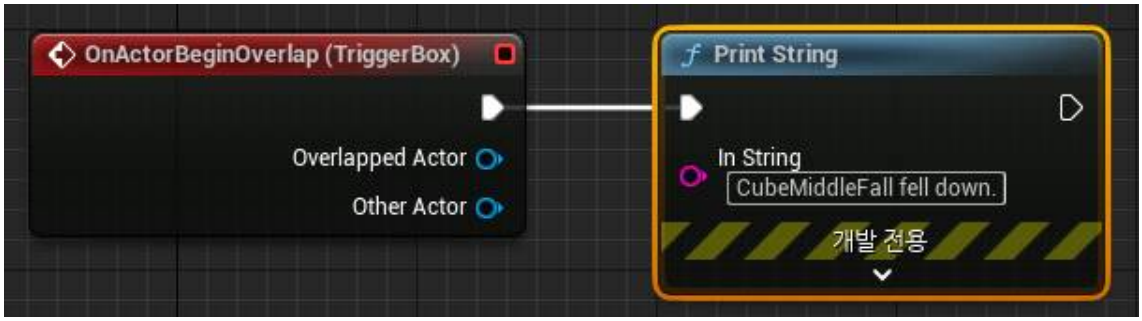
1. 새 프로젝트 **Pimpluse**를 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 Pimpulse로 입력하고 **생성** 버튼을 클릭하자. 프로젝트가 생성되고 언리얼 에디터 창이 뜰 것이다.
창이 뜨면, 메뉴바에서 **파일** » **새 레벨**을 선택하고 **Basic**을 선택하여 기본 템플릿 레벨을 생성하자.
그리고, 레벨 에디터 툴바의 저장 버튼을 클릭하여 현재의 레벨을 MyMap으로 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. 이름은 **CubeMiddleFall**로 수정하자. 위치는 (400,0,200)로 하자.
그리고, **디테일** 탭의 **피직스** 영역에 있는 **피직스 시뮬레이트(Simulate Physics)** 속성에 체크하자. 이제 물리 엔진이 적용될 것이다.
그다음, **디테일** 탭의 **콜리전** 영역에 있는 **오버랩 이벤트 생성(Generate Overlap Events)**에 체크하자. 이제 겹침 콜리전을 발생시키는 액터 부류에 속하게 되어서 겹침 이벤트가 발생될 수 있도록 하였다.
그다음, 툴바의 **액터 배치** 아이콘을 클릭하고 **셰이프** 아래의 **트리거 박스**를 드래그하여 배치하자. 이름은 **TriggerBox**로 그대로 두자.
위치를 (400,0,0)에 배치하자. **CubeMiddleFall**이 떨어지면 바로 아래에 충돌하도록 배치된다.



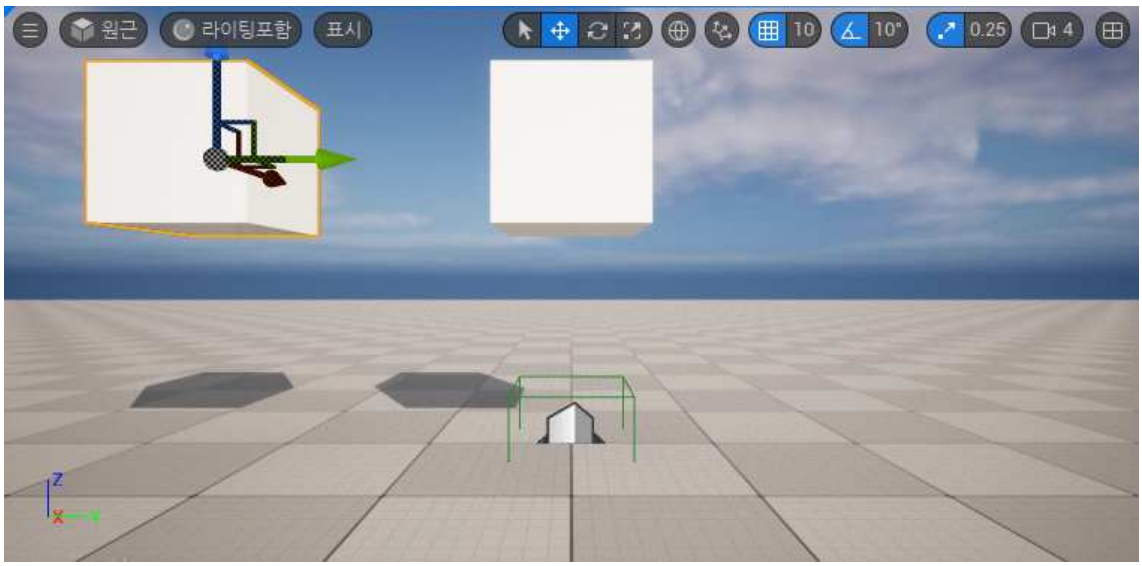
3. **아웃라이너**에서 **TriggerBox**를 클릭하여 선택하자.
그다음, 레벨 블루프린트 에디터로 가자.
그리고, 이벤트 그래프 탭의 격자판의 빈 곳에서 우클릭하고 액션선택 창에서 **TriggerBox0에 대한 이벤트 추가** 아래의 **콜리전** 아래의 **OnActorBeginOverlap 추가**를 선택하자.

레벨 블루프린트에 **OnActorBeginOverlap (TriggerBox)** 이벤트 노드가 추가될 것이다.
배치된 이벤트 노드에서 **PrintString** 노드를 추가하자.

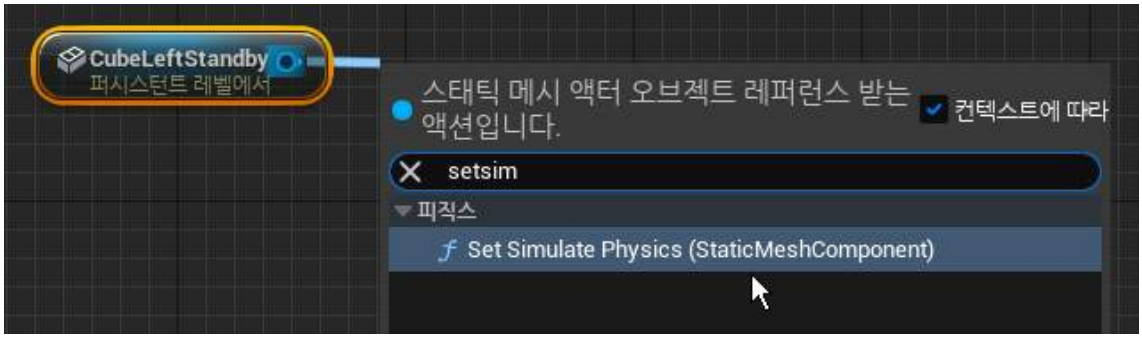


플레이해보자. 문자열이 출력될 것이다.

4. 이제부터, 겹침 이벤트가 발생하면 멈추어 있던 물체가 떨어지도록 해보자.
먼저, 떨어질 물체를 배치하자.
툴바의 **액터 배치** 아이콘을 클릭하고 **셰이프** 아래의 **큐브**를 드래그하여 레벨에 배치하자. 이름은 **CubeLeftStandby**로 수정하자. 위치는 (400,-250,200)로 하자.
그리고, **디테일** 탭의 **트랜스폼** 영역에 있는 **모빌리티** 속성을 **스태틱**에서 **무버블**로 바꾸자. 이제, 움직임이 가능하도록 하였다. 그러나, **피직스 시뮬레이트**(**Simulate Physics**) 속성에 체크하지 않았으므로 물리 엔진이 적용되는 대상은 아니다.



5. **아웃라이너**에서 **CubeLeftStandby**을 선택한 상태에서 레벨 블루프린트 에디터로 가자.
이벤트 그래프에서 우클릭고 액션선택 창에서 **CubeLeftStandby에 대한 레퍼런스 생성**을 클릭하자. 그다음, **CubeLeftStandby** 레퍼런스 노드의 출력핀을 드래그해서 **SetSimulatePhysics (StaticMeshComponent)** 노드를 배치하자.

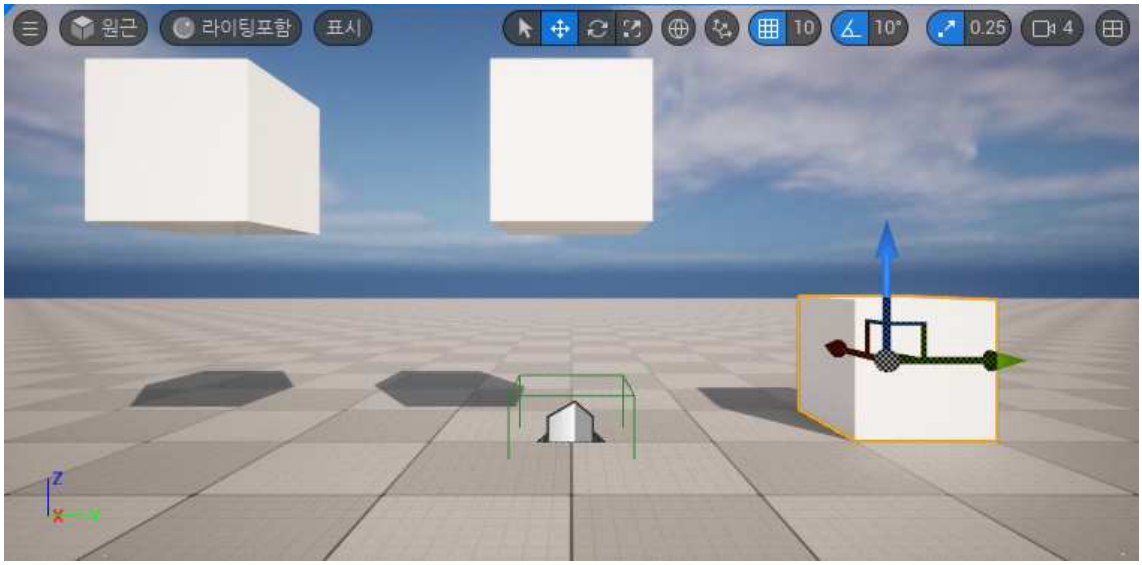


CubeLeftStandby 액터 내에 있는 **StaticMeshComponent**의 **SetSimulatePhysics** 함수 노드가 배치된다. 이 함수는 **디테일** 탭의 **피직스** 영역에 있는 **피직스 시뮬레이트(Simulate Physics)** 속성에 체크하는것과 동일한 효과를 만든다. 즉 함수 호출을 통해 물리 시뮬레이션이 시작되도록 한다.
참고로, 노드가 배치될 때에 타입 변환을 위한 추가적인 노드도 함께 배치된다.

6. 배치된 **SetSimulatePhysics** 노드의 **Simulate** 입력 핀에 체크하여 활성되도록 하자. 그리고 실행핀을 연결하여 그래프를 완성하자.

컴파일하자.
플레이해보자. 겹침 이벤트가 발생하면 **CubeLeftStandby**가 떨어지기 시작할 것이다.

7. 지금까지 물리 시뮬레이션에서의 모든 움직임은 중력에 기반하여 발생하였다.
한편, 중력 이외에도 외부에서 임의로 힘을 가하여 움직임을 일으킬 수 있다.
순간적인 힘을 **충격량** 또는 **임펄스**(impulse)라고 한다. 엔진에서는 액터에 임펄스를 가하는 함수인 **AddImpulse** 함수를 제공한다. 이제부터 임펄스를 적용하는 방법에 대해서 알아보자.
먼저, 임펄스를 가할 물체를 배치하자.
툴바의 **액터 배치** 아이콘을 클릭하고 **셰이프** 아래의 **큐브**를 드래그하여 레벨에 배치하자. 이름은 **CubeRightThrow**로 수정하자. 위치는 (450,250,50)로 하자. 바닥에 붙어있게 된다.
그다음, **디테일** 탭의 **트랜스폼** 영역에 있는 **모빌리티** 속성을 **스태틱**에서 **무버블**로 바꾸자.
그다음, **피직스** 영역에 있는 **피직스 시뮬레이트(Simulate Physics)** 속성에 체크하자. 충격량을 가해서 움직이도록 하기 위해서는 **무버블**로 되어 있어야 하고 또한 **피직스 시뮬레이트(Simulate Physics)**도 활성으로 되어 있어야 한다.
그다음, **피직스** 영역에 있는 **질량(kg)**의 체크박스를 체크하고 입력상자에 1을 입력하자. 액터의 질량은 대개는 메시의 부피를 기반으로 자동으로 계산하여 사용하지만 **질량(kg)** 속성에 체크하고 킬로그램 단위의 무게를 직접 지정할 수 있다. 자동계산된 무게가 178kg로 너무 무겁게 되어 있어서 이를 1kg로 아주 가볍게 만들어주었다.
이제, 임펄스를 받아서 움직임이 가능하도록 준비를 완료하였다.



8. **아웃라이너**에서 **CubeRightThrow**를 선택한 상태에서 레벨 블루프린트 에디터로 가자.
이벤트 그래프에서 우클릭고 액션선택 창에서 **CubeRightThrow에 대한 레퍼런스 생성**을 클릭하자.
그다음, **CubeRightThrow** 레퍼런스 노드의 출력핀을 드래그해서 **AddImpulse (StaticMeshComponent)** 노드를 배치하자. **CubeRightThrow** 액터 내에 있는 **StaticMeshComponent**의 **AddImpulse** 함수 노드가 배치된다.
AddImpulse 노드의 **Impluse** 입력핀은 적용할 임펄스의 방향과 크기를 명시한다.
우리는 **Impluse** 입력핀의 값을 (0,0,0)에서 (0,0,500)으로 바꾸자.

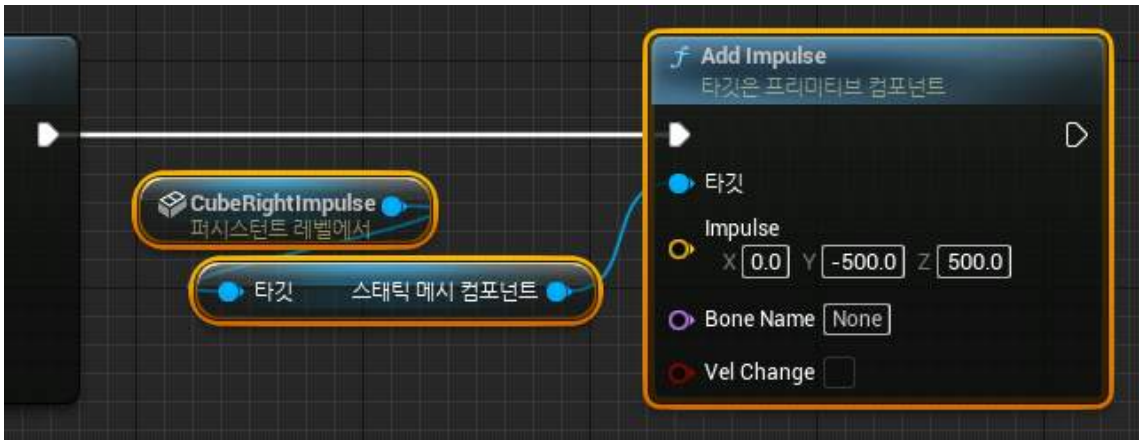
컴파일하자.
플레이해보자. 겹침 이벤트가 발생하면 **CubeRightThrow**는 임펄스를 받아서 튀어오를 것이다.

<참고> **AddImpluse** 노드에는 **VelChange** 입력핀이 있다. 이 입력핀에 체크하여 **true**가 되면 엔진은 **Impluse** 입력값을 힘(충격량)의 개념이 아닌 속도의 개념으로 처리해준다.
Impluse 입력값을 힘의 개념으로 처리한다면, 힘은 질량과 가속도의 곱이므로 질량이 클수록 발생하는 가속도가 작아진다. 따라서 우리는 이전에서 질량을 1로 작게 하여서 힘이 그대로 가속도로 연결되도록 하였다.
한편, **Impluse** 입력값을 속도의 개념으로 처리한다면, 질량은 1로 가정하는것과 같이 질량을 무시하고 바로 해당 속도가 발생하는 미지의 힘이 가해진 것으로 처리해준다. 즉 **VelChange** 입력핀에 체크하면 질량의 효과가 없어진다. 따라서, 우리는 이전에서 질량을 1로 작게 하지말고 원래대로 그대로 두고 **Impluse** 입력핀에 체크하여도 동일한 결과를 얻는다.

9. 뷰포트에서 뷰 모드를 상단 뷰로 바꾸자.
CubeRightThrow를 선택하고 **Alt**+**좌클릭**+**드래그**해서 왼쪽에 복제하자. 그 왼쪽에 또하나를 다시 복제

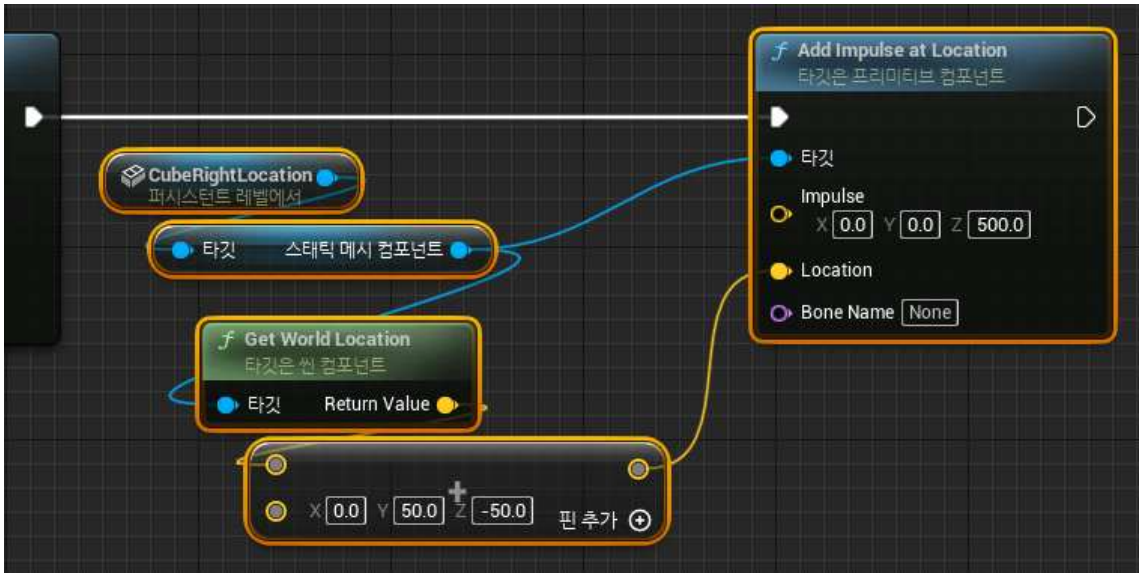
하자.
각각의 이름을 CubeRightImpluse와 CubeRightLocation으로 하자.
위치는 각각 (280,250,70)과 (160,250,70)으로 하자.
이렇게 하면 CubeRightThrow는서 설정한 동일한 속성값이 모두 동일하게 CubeRightImpluse와 CubeRightLocation에 복제되어 편리하다.
이제 다시 뷰모드를 원근 뷰로 바꾸자.
그다음, 플레이할 때에 잘 보이도록 아웃라이너에서 PlayerStart 액터를 선택하고 위치를 (0,0,112)에서 (-200,0,112)로 수정하자. 약간 뒤에서 시작하므로 레벨을 잘 볼 수 있는 위치가 된다.

10. 아웃라이너에서 CubeRightImpluse와 CubeRightLocation를 레벨 블루프린트 에디터의 이벤트그래피 탭의 격자공간으로 드래그해서 각각의 레퍼런스 노드를 배치하자.
그다음, CubeRightImpluse 레퍼런스 노드의 출력핀을 드래그해서 AddImpulse (StaticMeshComponent) 노드를 배치하자. AddImpulse 노드의 Impluse 입력핀의 값을 (0,-500,500)으로 바꾸자.
그다음, 이전의 AddImpulse 노드의 실행핀을 이어주자.



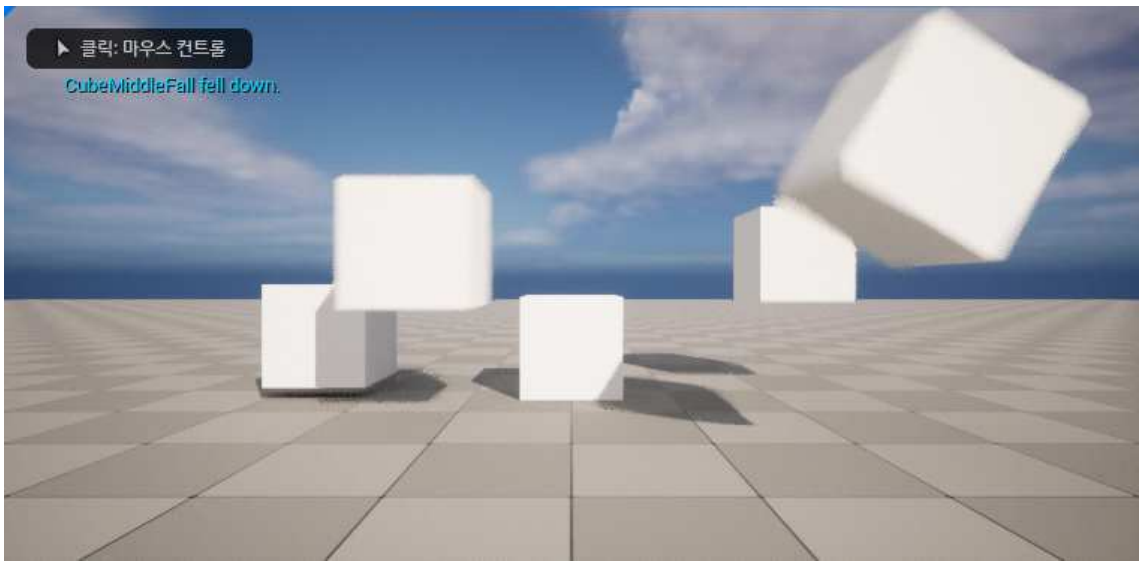
플레이해보자.
CubeRightImpluse 액터가 왼쪽으로 발사될 것이다. 이런 방식으로 Impluse 입력핀을 통해서 가하는 힘의 방향을 조절할 수 있다.

11. 이제, CubeRightLocation 레퍼런스 노드의 출력핀을 드래그해서 AddImpulse at Location (StaticMeshComponent) 노드를 배치하자. AddImpulse at Location 노드의 Impluse 입력핀의 값을 (0,0,500)으로 바꾸자.
한편, StaticMeshComponent의 피봇은 메시의 중심이 있다. 우리는 중심이 아닌 오른쪽으로 끝부분에 힘을 가하도록 해보자. 이를 위해서 먼저, StaticMeshComponent의 GetWorldLocation 함수를 호출하여 월드 공간에서의 메시의 피봇의 위치를 가져오자. 메시의 피봇에서 오른쪽 끝부분으로의 상대적인 이동 거리를 (0,50,-50)라고 하자. 피봇 위치에 (0,50,-50)만큼 더한 위치 벡터를 구하자. 그다음, 구한 위치 벡터를 AddImpulse at Location 노드의 Location 입력핀에 연결하자.
이런 방식으로 Impluse 입력핀과 Location 입력핀을 함께 사용하여 가하는 힘의 방향뿐만 아니라 가해지는 위치를 조절할 수 있다.
그다음, 이전의 AddImpulse 노드의 실행핀을 이어주자.



12. 플레이해보자.

CubeRightLocation 액터의 오른쪽 부분에 힘을 가했으므로 스핀하면서 수직 위로 발사될 것이다.



이 절에서는 동적으로 물리 상태를 변경하고 힘을 가하는 방법을 학습하였다.

□