12_ 액터 생성과 소멸

<제목 차례>

인천대학교 컴퓨터공학부 박종승

1. 개요

이 장에서는 액터를 생성하고 소멸하는 방법에 대해서 학습한다.

레벨에 배치된 모든 액터는 레벨이 플레이되면 자동으로 생성된다. 게임에는 레벨에 배치된 정적인 액터 이외에도 플레이 도중에 나타났다가 사라지는 동적인 액터도 있다. 동적인 액터는 게임플레이에 있어서 매우 중요한 역할을 한다.
동적으로 나타났다가 사라지는 객체의 예를 살펴보자. 첫 번째 예로, 픽업 가능한 아이템은 동적으로 나타날 수 있고 캐릭터가 습득하게 되면 아이템은 사라진다. 두 번째 예로, 대포를 발사하면 포탄 객체가 생성되어 멀리 날아가고 그 후에는 포탄이 사라진다. 세 번째 예로, 적은 계속해서 다시 생성되고 데미지를 받으면 죽어서 사라진다.
스폰과 리스폰 용어에 대해서 알아보자. 캐릭터나 아이템 등의 객체를 동적으로 생성하는 행위를 **스폰**(spawn)이라고 표현한다. 한편, 죽거나 수명이 소멸된 캐릭터나 객체를 다시 생성하는 행위를 **리스폰**(respawn)이라고 표현한다. 리스폰 과정은 스폰 과정을 수행하는 것에 추가적으로 이전의 객체가 다시 부활한 것으로 간주되도록 처리해주어야 한다.
예를 들어, 무한히 생성되는 적 몬스터는 이전의 죽은 적 몬스터와 독립적이므로 스폰에 해당한다. 반면, 플레이어 캐릭터가 죽은 후에 다시 생성하는 것은 하나만 있는 것을 부활시키는 것이므로 리스폰에 해당한다.

스폰할 위치를 지정하는 방법에 대해서 알아보자. 스폰할 때에 스폰할 위치 좌표를 상수로 명시하는 것은 불편하고 융통적이지 않은 방법이다. 이보다 더 좋은 방법은 위치 정보를 가지는 액터를 레벨에 미리 배치해두고 스폰할 때에 이 액터의 위치에 스폰되도록 하는 방법이다.
이러한 방법을 사용하면 많은 장점이 있다. 스폰할 위치를 마우스로 움직여서 시각적으로 편리하게 조정할 수 있다. 또한 나중에 수시로 쉽게 바꿀 수도 있다.
위치 정보를 나타내주는 역할을 하도록 사용되는 대표적인 액터로 **TargetPoint**라는 액터가 있다. 또한 플레이어 폰이 스폰될 때에 사용되는 **PlayerStart** 액터도 이러한 역할을 하는 액터이다. 이러한 액터를 레벨에 배치해두고 실행 시에 그 액터의 위치에 스폰되도록 하면 된다. 이러한 액터는 스폰 지점으로 활용하는 경우 외에도 이동할 때의 거쳐야 하는 중계 지점의 위치 정보를 제공하는 기능 등의 다양한 활용이 가능하다.

이제부터 액터를 소멸시키는 방법에 대해서 알아보자.
액터를 즉시 소멸시키고자 할 때에는 액터의 **DestroyActor** 함수를 호출하면 된다. 노드의 실행을 위해 필요한 인자는 없다. 더이상 필요없는 액터들은 컴퓨터의 자원이 낭비되지 않도록 바로 제거해 주는 것이 좋다.
한편, 액터를 동적으로 생성하고 소멸시키는 일을 반복하다보면 더이상 불필요하지만 소멸되지 않고 계속 남아있는 액터가 있을 수 있다. 예를 들어 공중에 발사한 포탄은 계속 날아갈 것이다. 어느 지점에 충돌하는 경우에는 충돌 검사를 통해 포탄이 소멸되도록 조치할 수 있을 것이지만 충돌하지 않으면 소멸되지 않고 계속 날아가기만 할 것이다.
게임에서 이러한 경우가 생기면 필요없는 액터로 인해서 하드웨어 자원이 계속 소모된다. 장시간 누적되면 심각한 속도저하로 이어진다. 따라서 필요없는 액터는 반드시 제거되도록 조치해주어야 한다.

이러한 문제의 해결 방법의 하나는 수명(lifespan)이다. 수명은 게임에서 적절하게 사용해야 하는 중요한 속성이다. 발사체, 총알, 깨진 파편, 낙하물 등은 반드시 수명을 명시해주도록 하자.
각 액터에 LifeSpan 속성이 있다. 이 속성에 액터의 최대 수명을 설정할 수 있다. 설정된 수명을 경과하면 엔진이 해당 액터를 파괴한다. 수명의 디폴트 값은 0이다. 0은 수명 기능을 사용하지 않는다는 의미이다. 이 경우에는 액터는 수명 속성으로 인해서 엔진에 의해 파괴되지 않는다.
액터의 수명을 명시하는 방법을 알아보자. 블루프린트 에디터에서 **InitialLifeSpan** 속성에 값을 명시할 수 있다. 명시된 값은 액터의 수명값을 초기화하는데 사용된다. 값을 바꾸지 않는 한 명시된 수명값이 계속 사용된다. 수명을 실행 시간에 동적으로 지정하기 위해서는 **SetLifeSpan** 함수를 호출하면 된다.

한편, 수명으로 해결할 수 없는 경우도 있다. 이동 물체나 발사된 투사체나 파괴된 조각 등이 낙하하는 경우를 생각해보자. 이들 물체는 바닥에 닿을 때까지 계속 낙하할 것이다. 만약 바닥이 뚫려있는 곳이거나 바닥을 피해서 그 아래로 내려가는 경우라면 물체는 낙하만 계속할 것이다. 바닥 아래로 내려가게 되면 시야에서 사라지지만 레벨에 계속 존재하면서 계속 낙하하게 된다. 이러한 불필요한 액터가 장시간 누적되면 시스템 성능이 저하된다.
이런 경우의 문제를 해결을하기 위해서 게임 엔진은 무한 낙하를 방지하는 기능을 제공한다. 엔진이 액터를 소멸시키는 또다른 방법으로 **킬Z** (Kill Z) 방법을 제공하고 있다. 이 방법은 액터가 설정한 Z 위치보다 더 아래로 떨어질 경우에 그 액터를 파괴시킨다.
킬Z 기능의 사용 방법을 알아보자. **킬Z**는 각 액터 단위로 설정하는 속성이 아니라 레벨 단위로 설정하는 속성이다. **월드 세팅** 탭에서 **월드** 영역에 **킬Z** 속성이 있다. 이 속성값에 적당한 값을 지정하면 된다. 실행 시에 지정된 높이보다 더 아래로 내려가는 액터가 생기는 경우에, 엔진은 그 액터를 즉시 소멸시킨다.
지금까지 액터를 소멸시키는 방법에 대해서 알아보았다. 액터의 **DestroyActor**를 호출하여 액터를 즉시 소멸시키는 방법이 있고, 액터의 **SetLifeSpan**으로 수명을 지정해서 일정 시간 후에 자동 소멸되도록 하는 방법이 있고, 설정한 Z 위치보다 더 아래로 떨어질 경우 액터를 소멸시키는 **킬Z** 방법이 있다.

2. 액터 생성과 소멸

이 절에서는 액터를 생성하고 소멸하는 방법에 대해서 학습한다.

액터를 스폰하는 함수 노드를 알아보자. 액션선택 창에서 **SpawnActor fromClass** 노드를 검색하여 배치하면 된다. 생성할 액터 클래스 타입을 노드의 입력핀인 **Class** 입력핀에 명시해주어야 한다. 또한 생성할 위치를 **SpawnTransform** 입력핀에 명시해주어야 한다. 이 노드는 액터를 생성하고 생성된 액터 객체를 출력핀으로 리턴해준다.

SpawnTransform 입력핀에는 **Transform** 타입의 값을 지정해주어야 한다. **Transform** 타입에 대해서 더 알아보자. **Transform** 타입은 **Location**, **Rotation**, **Scale**을 세 요소값을 모두 포함하는 행렬 타입으로 생각하면 된다. **Transform** 타입에 대해서도 **Make** 함수와 **Break** 함수를 제공한다. 즉, **MakeTransform**은 **Location**, **Rotation**, **Scale**의 세 요소값으로부터 **Transform**을 만들어준다. 그리고 **BreakTransform**은 **Transform**으로부터 **Location**, **Rotation**, **Scale**의 세 요소값을 분리해준다.

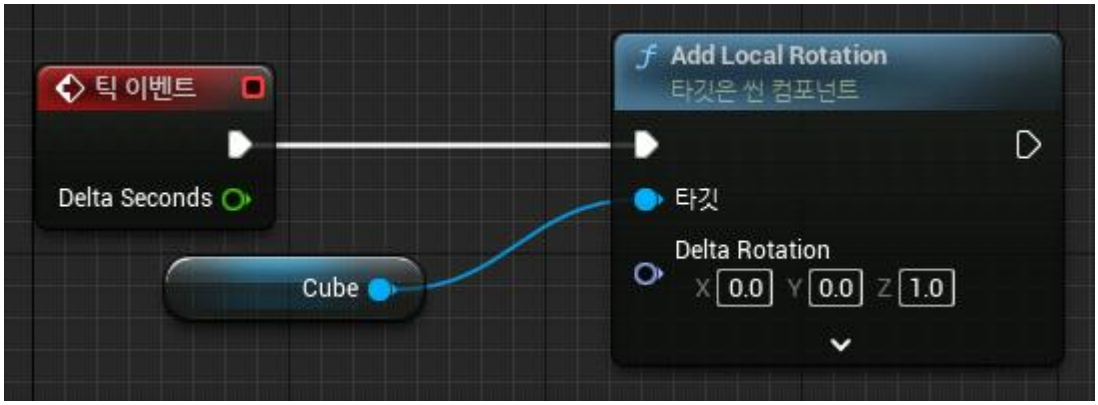
레벨에 배치된 액터로부터 그 액터의 현재 상태의 **Transform** 정보를 바로 얻을 수 있다. 이를 위해서는 해당 액터의 **GetActorTransform** 함수를 호출하면 된다. 이 함수를 사용하면 **TargetPoint** 액터를 레벨에 배치하고 이 액터에 스폰되도록 할 수 있다. 즉, **TargetPoint** 액터의 **GetActorTransform** 함수를 호출하고 그 결과를 **SpawnActor** 노드의 **SpawnTransform** 입력핀에 연결해주면 된다.

이제부터 액터를 스폰하는 방법에 대해서 알아보자.

---

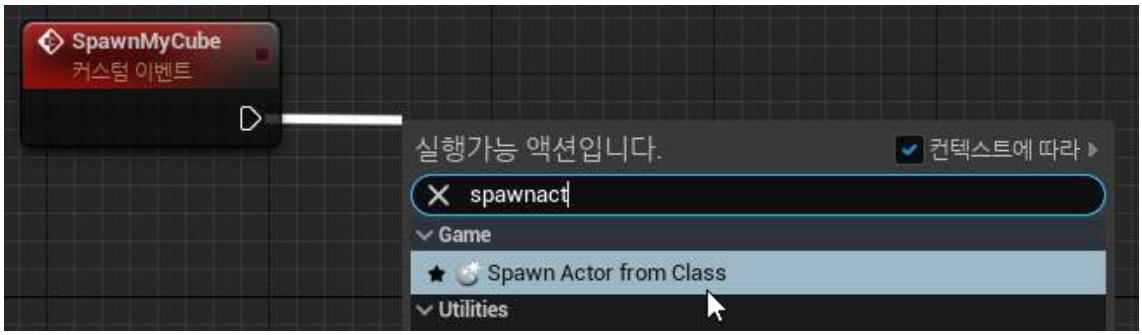
1. 새 프로젝트 `Pspawnactor`를 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 `Pspawnactor`로 입력하고 **생성** 버튼을 클릭하자. 프로젝트가 생성되고 언리얼 에디터 창이 뜰 것이다. 창이 뜨면, 메뉴바에서 **파일** » **새 레벨**을 선택하고 **Basic**을 선택하여 기본 템플릿 레벨을 생성하자. 그리고, 레벨 에디터 툴바의 저장 버튼을 클릭하여 현재의 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. 동적으로 액터를 생성하는데 사용될 액터를 블루프린트 클래스로 만들자.
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_MyCube**으로 수정하자.
그다음, **BP_MyCube**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.
그다음, **Cube**가 선택된 상태에서 **이벤트 그래프** 탭으로 이동하고, 빈 격자판에서 우클릭하고 **AddLocal Rotation (Cube)**를 선택하여 배치하자. 그리고, 노드의 **DeltaRotation** 입력핀의 **Z**에 1을 입력하자. 그리고, **Tick** 이벤트 노드의 실행핀에 연결하자. 다음과 같은 모습이 될 것이다.

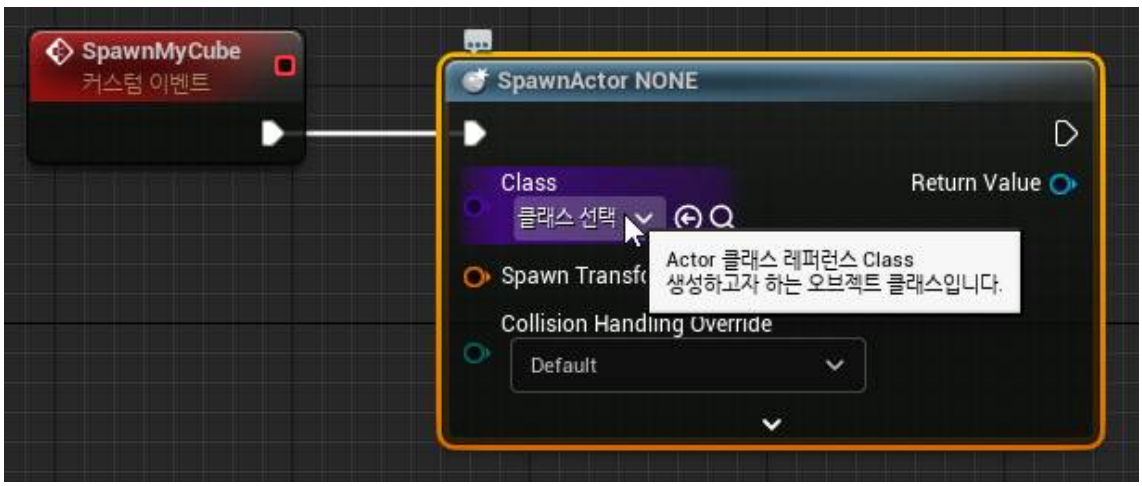


이제, 예제에 사용할 액터인 **BP_MyCube**를 완성하였다.
저장하고 컴파일하자.

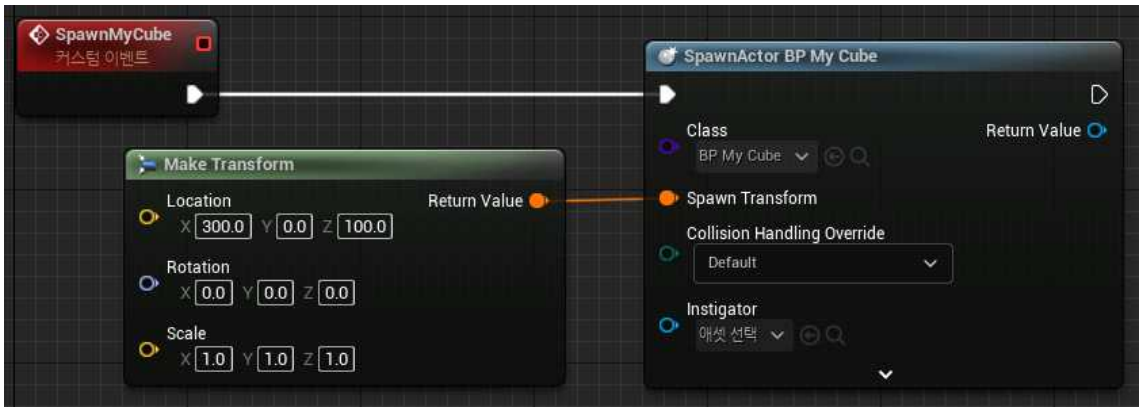
3. 레벨 에디터에서 툴바에서 **블루프린트**를 클릭하고 **레벨 블루프린트 열기**를 선택하여 레벨 블루프린트 에디터를 열자.
그다음, **이벤트 그래프**의 격자판의 빈 곳에서 우클릭하고 **커스텀 이벤트 추가**를 선택하여 커스텀 이벤트 노드를 추가하자. 이름은 **SpawnMyCube**라고 하자.
SpawnMyCube 이벤트 노드의 실행핀을 드래그하고 액션선택 창에서 **SpawnActor fromClass**를 검색하여 배치하자.



4. 노드의 제목이 **SpawnActor NONE**인 노드가 배치될 것이다. 아직 어떤 클래스 타입의 액터를 스폰할 것인지가 명시되어 있지 않아서 **NONE**으로 되어 있다. **Class** 입력핀의 **클래스 선택**을 클릭하고 **BP_MyCube**를 선택하자.



5. SpawnActor 노드는 생성할 액터 클래스 타입을 명시하는 Class 입력핀과 더불어 생성할 위치를 명시하는 SpawnTransform 입력핀에 반드시 입력을 연결해주어야 한다.
SpawnTransform 입력핀에 상수값을 입력할 수 없으므로 Transform 타입을 만드는 함수인 MakeTransform을 배치하자. MakeTransform 노드의 Location 입력값이 (300,0,100)이 되도록 하고 ReturnValue 출력핀을 SpawnTransform 입력핀에 연결하자.



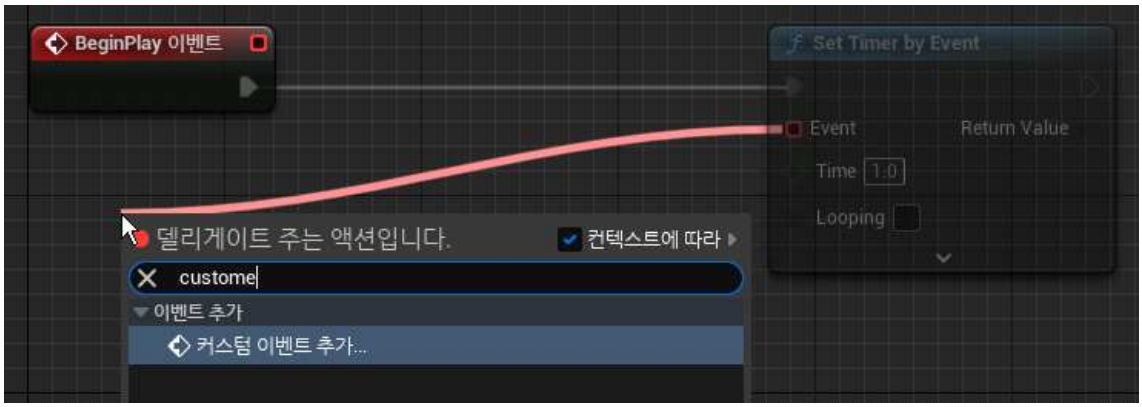
이제 액터를 스폰하는 SpawnMyCube 이벤트 그래프를 완성하였다.

6. 이제, BeginPlay 이벤트로 이동하여 실행핀을 당기고 SetTimerByEvent 노드를 배치하자.
SetTimerByEvent 노드는 타이머를 지정하는 노드이다. Time 입력핀의 값으로 1을 입력하자.

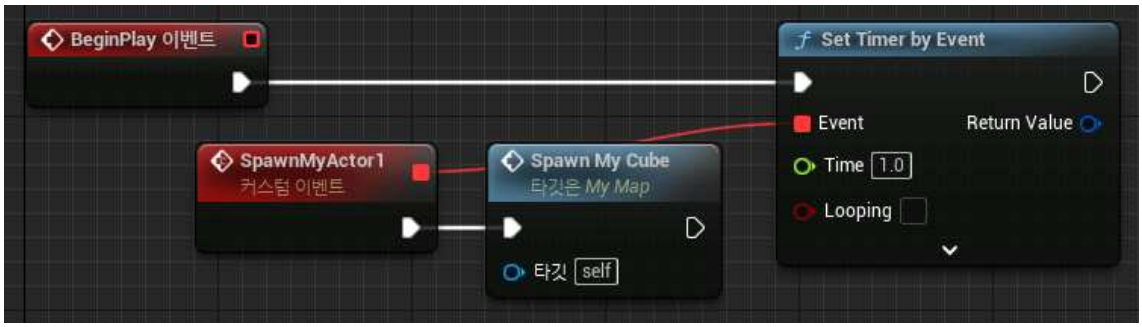
여기서, 타이머에 대해서 더 알아보자.
SetTimerbyEvent 노드는 Time 입력핀에 지정된 초 단위의 시간이 경과한 후에 Event 입력핀에 지정된 이벤트 노드를 실행한다. Event 입력핀은 실행할 이벤트 노드를 지정하는 핀이므로 이벤트 디스패처에서 학습한 델리게이트 핀에 해당한다. 즉, 나중에 실행할 이벤트 노드를 미리 바인드해두는 것이다.
타이머 노드의 Looping 입력핀은 디폴트로 체크해제되어 있다. 이 입력핀에 체크하면 Time에 설정한 시간 간격으로 계속해서 Evnet 입력핀에 바인드된 이벤트 노드를 실행한다.
타이머 노드는 ReturnValue 출력핀으로 타이머 핸들을 리턴한다. 이 핸들을 사용하여 나중에 타이머 관련 함수를 호출할 수 있다.
주요한 타이머 관련 함수를 예를 들면 다음과 같다.
먼저, Clear and Invalidate Timer by Handle 노드는 설정된 타이머를 제거한다. 무한 반복 중인 타이머를 제거하는데 사용된다.
그리고, Get Timer Elapsed Time by Handle과 Get Timer Remaining Time by Handle는 현재 반복에서의 타이머가 시작된 이후 경과된 시간과 남아있는 시간을 리턴한다.

그리고, **Pause Timer by Handle**과 **Unpause Timer by Handle**은 설정된 타이머를 일시정지하고, 일시정지된 타이머를 재개한다.
그리고, **Does Timer Exist by Handle**과 **Is Timer Active by Handle**과 **Is Timer Paused by Handle**은 주어진 핸들의 타이머가 존재하는지를 체크하고, 주어진 핸들의 타이머가 존재하고 또 활성상태인지를 체크하고, 주어진 핸들의 타이머가 존재하고 또 일시정지 상태인지를 체크한다.
한편, **byHandle** 대신 **byFunctionName**이 붙은 타이머 관련 함수들이 있다.
SetTimerbyEvent 함수 외에 **SetTimerbyFunctionName** 함수가 있고 그 외의 대부분의 타이머 함수들에 대해서도 대응 함수가 있다. 이들 **byFunctionName** 대응 함수는 핸들을 입력하는 대신 호출될 함수 이름을 문자열로 직접 입력한다. 이들 함수는 문자열 오류 등으로 인해 문제가 발생할 여지가 있으므로 **byFunctionName**이 붙은 함수보다는 **byHandle**이 붙은 함수를 사용하는 것을 권장한다.
지금까지 타이머에 대해서 알아보았다.

7. 이제 **SetTimerbyEvent** 노드의 **Event** 델리게이트 핀을 드래그하고 액션선택 창에서 **커스텀 이벤트 추가**를 선택하여 커스텀 이벤트 노드를 배치하자. 이름은 **SpawnMyActor1**으로 하자.



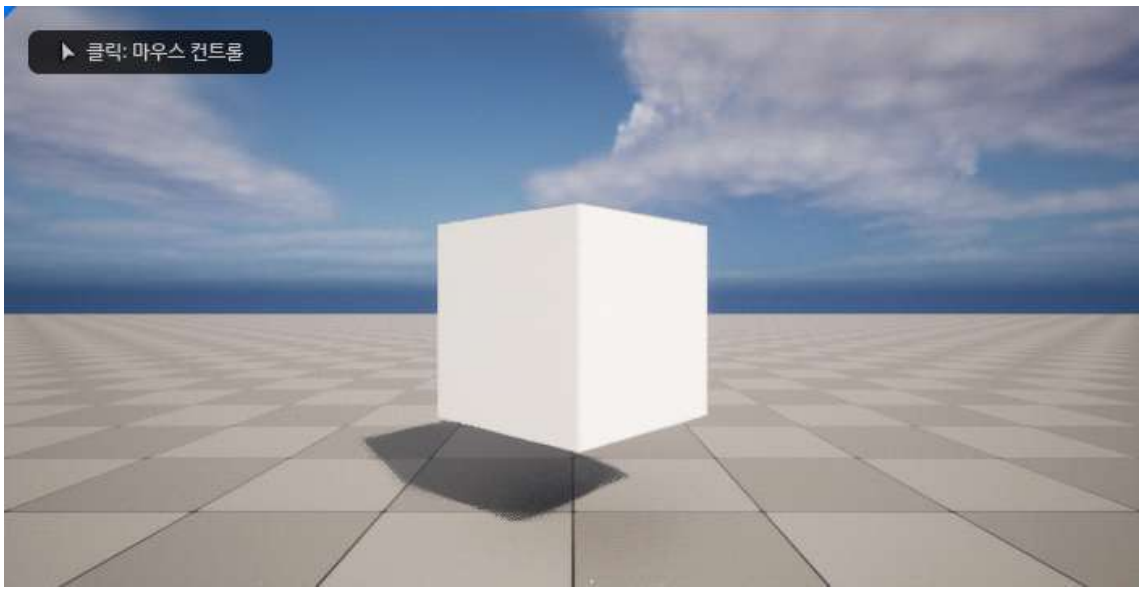
8. **SpawnMyActor1** 노드의 실행핀을 드래그하고 **SpawnMyCube** 함수호출 노드를 배치하자.
다음과 같은 모습이 될 것이다.



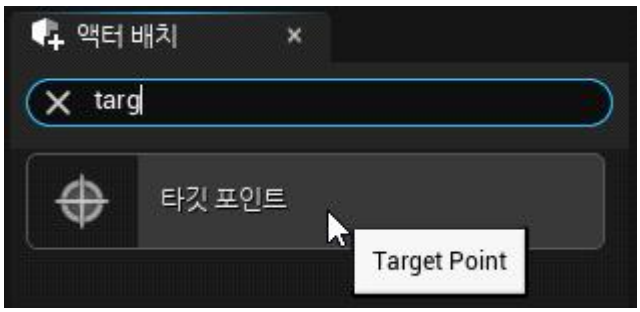
한편, **SpawnMyActor1** 노드를 새로 만드는 대신에, **SetTimerbyEvent** 노드의 **Event** 델리게이트 입력핀과 **SpawnMyCube** 이벤트 노드의 델리게이트 출력핀을 바로 연결해도 된다. 그러나 이렇게 델리게이트 핀을 자주 연결하다보면 나중에 노드그래프가 서로 얽혀서 분리되지 못하게 되는 단점이 있다. 우리는 앞으로 **SpawnMyActor1**과 같은 커스텀 이벤트 노드를 추가로 하나 더 만들더라도 노드 그래프 분리를 위한 방법을 택하기로 하자.
컴파일하고 저장하자.

9. 레벨 에디터에서 플레이해보자. 1초 후에 스핀하는 큐브 모양의 액터인 **BP_MyCube**가 화면에

나타날 것이다.

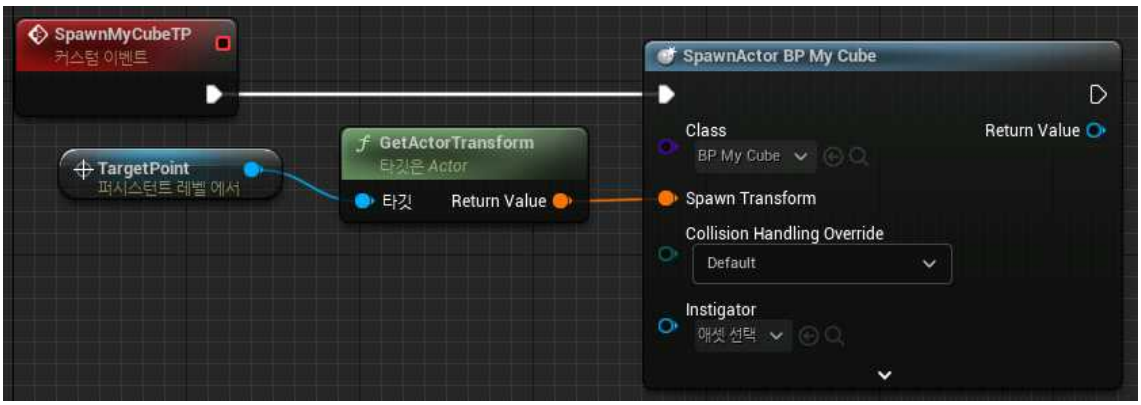


10. 액터를 생성하는 함수인 **SpawnActor** 노드는 **SpawnTransform** 입력핀에 반드시 생성할 위치를 명시해주어야 한다. 이전 예제에서는 **MakeTransform** 노드를 사용하여 **Transform** 타입의 값을 상수값으로 생성하여 입력해주었다. 그러나 이러한 스크립트 코드에서 생성할 특정 위치를 직접 지정하는 방법으로 유연하지 못한 하드코딩 방법이어서 바람직하지 않다. 이보다 더 나은 방법은 레벨 에디터에서 생성할 위치를 지정할 수 있도록 하는 방법이다.
엔진은 **TargetPoint**라는 액터를 제공한다. 이 액터는 트랜스폼 정보를 가진 간단한 액터이다. 이 액터는 시각적인 실체가 없고 단지 레벨에서 위치를 표시하기 위한 용도로 사용된다. 이 액터를 사용하면 생성할 위치를 좌표로 직접 입력하는 대신 액터의 위치를 레벨에서 마우스로 조절하여 표시할 수 있다.
레벨 에디터의 **액터 배치** 탭에서 **TargetPoint** 액터를 검색하여 찾아보자.

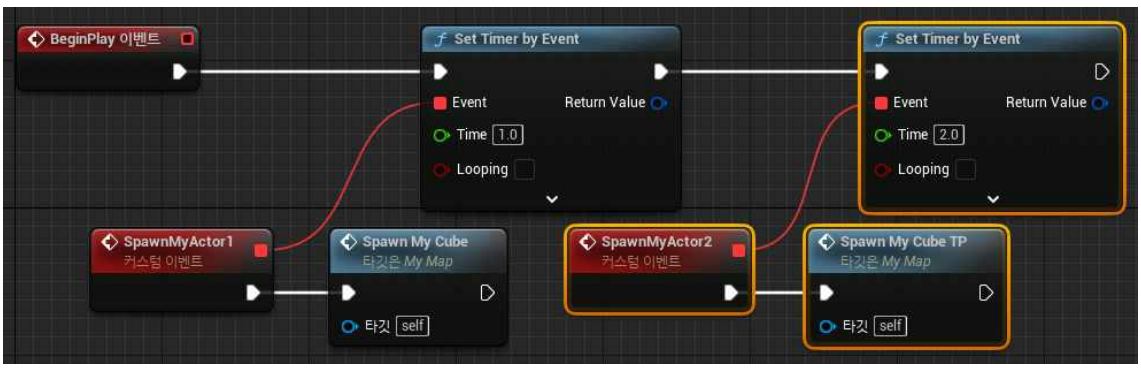


11. **액터 배치** 탭에서 **TargetPoint** 액터를 드래그해서 레벨에 배치하자.
위치를 (300,-200,100)으로 지정하자.
그다음, 레벨에서 **TargetPoint** 액터가 선택된 채로 레벨 블루프린트로 이동하자.
그다음, 커스텀 이벤트 노드를 추가하고 이름을 **SpawnMyCubeTP**라고 하자.
그다음, **SpawnMyCubeTP** 이벤트 노드의 실행핀을 드래그하고 **SpawnActor fromClass** 노드를 추가하자. **Class** 입력핀에는 **BP_MyCube**를 지정하자.
그다음, 격자판의 빈 곳에서 우클릭하고 **GetActorTransform**을 검색하자. **TargetPoint1에서 함수 호출** 카테고리 아래에 있는 함수를 선택하자. **TargetPoint** 액터의 레퍼런스 노드와 함수 노드가 함께 배치될

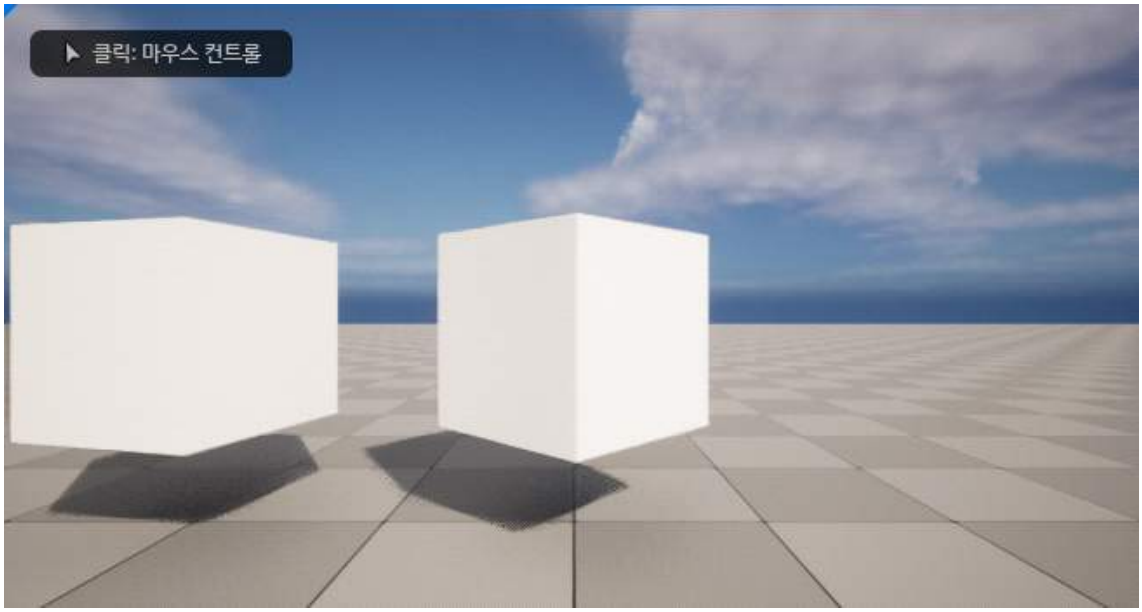
것이다. 여기서 **TargetPoint1**은 배치된 **TargetPoint** 액터의 ID이며 배치할때마다 달라질 수 있다.
그다음, **GetActorTransform** 노드의 **ReturnValue** 출력핀을 **SpawnActor** 노드의 **SpawnTransform** 입력핀에 연결하자.
이제 다음과 같은 모습이 될 것이다.



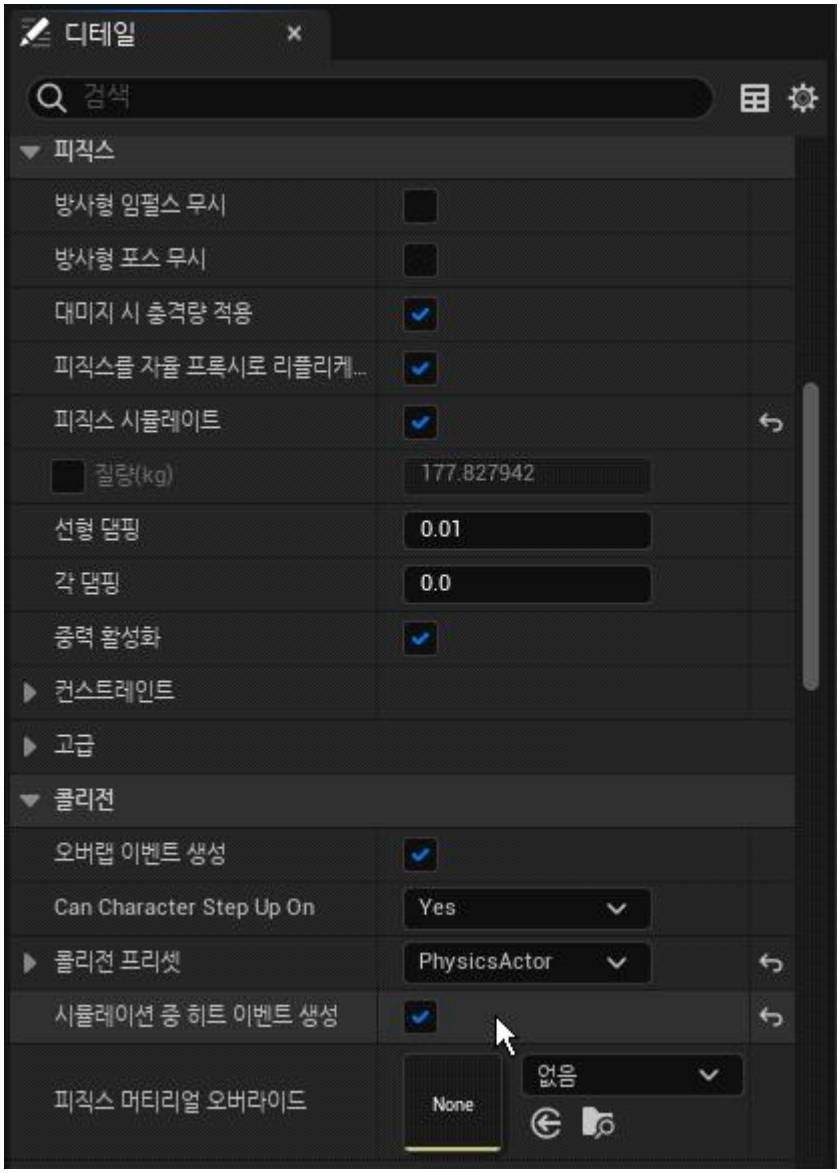
12. 이제, **BeginPlay** 이벤트 노드의 그래프에서 **SpawnMyCubeTP**가 호출되도록 하자.
먼저, 노드 네트워크의 가장 뒤에 **SetTimerbyEvent** 노드를 추가하자. **Time** 입력핀에는 2를 입력하자.
그다음, 추가된 **SetTimerbyEvent** 노드의 **Event** 델리게이트 입력핀을 왼쪽으로 드래그하고
커스텀 이벤트 추가를 선택하여 새 커스텀 이벤트 노드를 배치하자. 이름은 **SpawnMyActor2**라고 하자.
SpawnMyActor2라고 노드의 출력핀을 드래그하여 **SpawnMyCubeTP** 함수 호출 노드를 배치하자.
이제 다음과 같은 모습이 될 것이다.



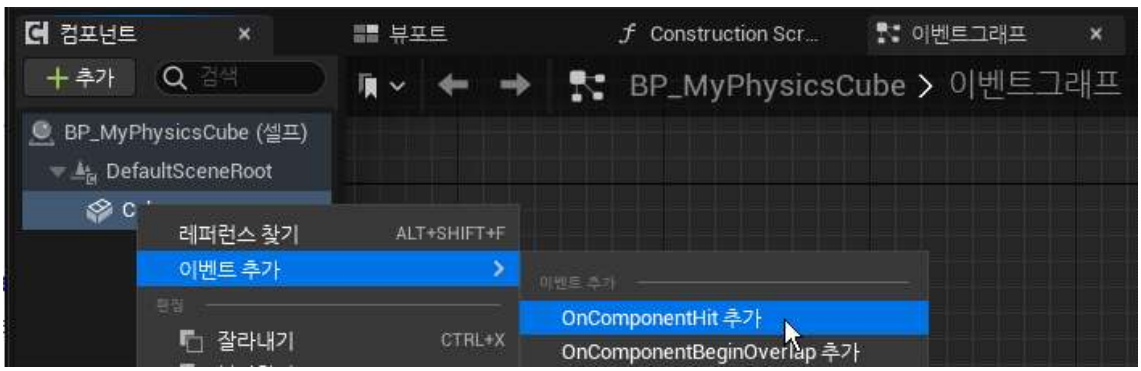
13. 레벨 에디터에서 플레이해보자. 1초 후에 스핀하는 큐브 액터가 나타나고 2초 후에 또다른 스핀 큐브 액터가 나타날 것이다.



14. 지금까지는 액터 생성에 대해서 학습하였다.
이제부터는 액터 소멸에 대해서 학습해보자.
먼저, 액터 소멸 예제를 적용해볼 액터를 만들자.
우선, 콘텐츠 브라우저에서 **BP_MyCube**를 선택하고 **Ctrl+C**를 누르고 **Ctrl+V**를 눌러 액터를 복제하자. 복제된 액터의 이름을 **BP_MyPhysicsCube**라고 하자. 그다음, **BP_MyPhysicsCube**를 더블클랙하여 블루프린트 에디터를 열자. 그다음, **컴포넌트** 탭에서 **Cube**를 선택하자. 그다음, **디테일** 탭에서 **피직스** 영역의 **피직스 시뮬레이트(Simulate Physics)**를 체크하고, **콜리전** 영역의 **시뮬레이션 중 히트 이벤트 생성(Simulation Generates Hit Events)**을 체크하자. 이 두 속성은 계산량의 최소화를 위해서 디폴트로 체크해제되어 있다.
Simulate Physics를 체크하면 이 컴포넌트가 물리 엔진의 적용을 받도록 한다. **Simulation Generates Hit Events**를 체크하면 이 컴포넌트에 대해서 겹침이 아닌 블록 유형의 충돌 시에 **Hit** 이벤트를 발생시킨다.



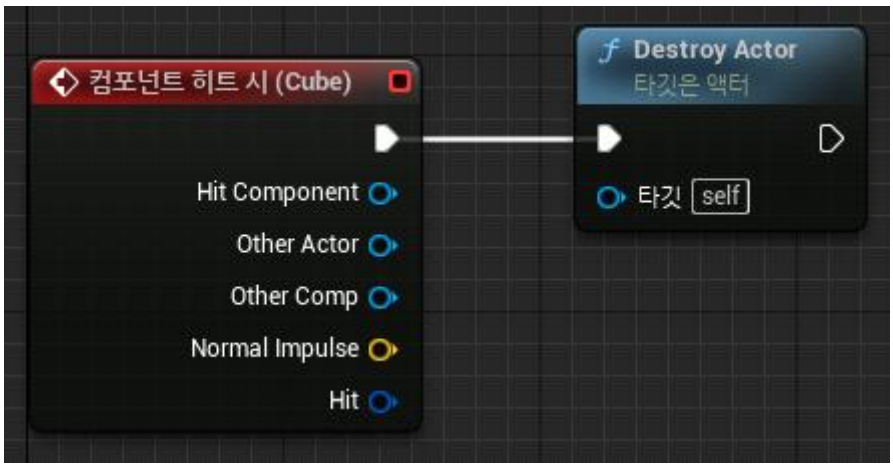
15. **컴포넌트** 탭에서 **Cube** 위에서 우클릭하고, 팝업메뉴창에서 **이벤트 추가** » **OnComponentHit 추가**를 선택하자.



OnComponentHit (Cube) 이벤트 노드가 배치된다.

16. **OnComponentHit (Cube)** 이벤트 노드의 실행핀을 드래그하고 **DestroyActor** 노드를 검색하여 배치하

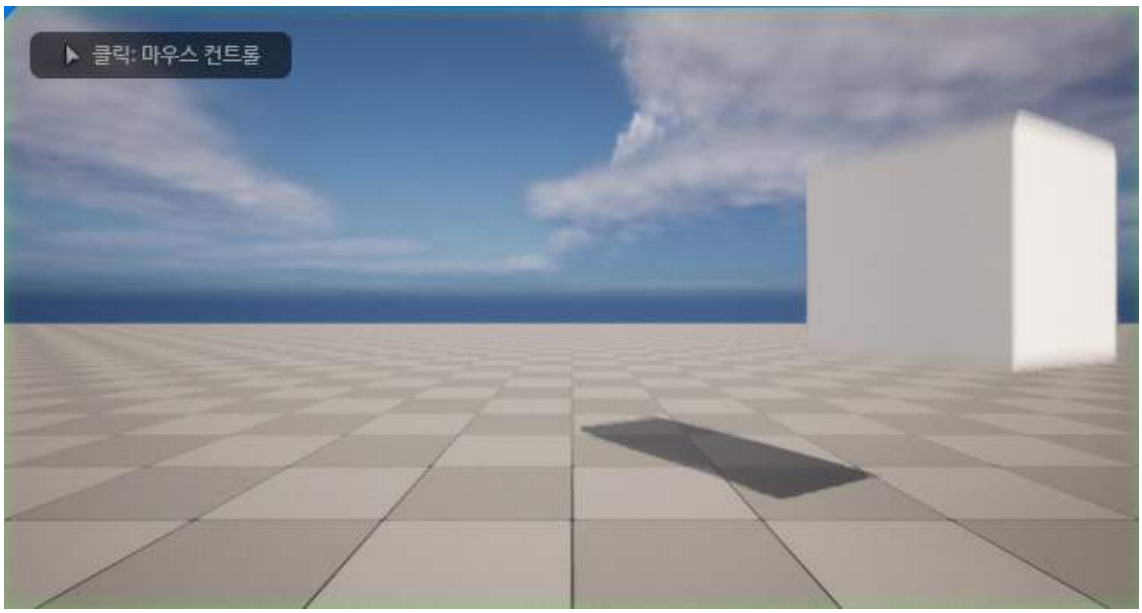
자. 이 노드는 맵에서 액터를 소멸시킨다. 아래와 같은 그래프가 완성된다.



DestroyActor 노드의 **타깃** 입력핀은 그대로 두면 디폴트 값인 **self**가 적용된다. **self**는 자기 자신 **Actor**를 의미하므로 자신을 파괴하게 된다.
이제 예제로 사용할 액터인 **BP_MyPhysicsCube**를 완성하였다.
컴파일하고 저장하자.

<참고> 레벨 블루프린트에서는 액션 노드의 타겟 입력에 지정하기 위해서 명령의 타겟이 되는 액터의 레퍼런스를 그래프에 추가하는 과정이 흔히 필요하다. 그러나, 클래스 블루프린트에서는 타겟이 되는 액터가 자기 자신인 경우가 많으며 이 경우에는 명령의 타켓을 지정하지 않고 디폴트인 자기 자신(self)으로 그대로 두면 된다.

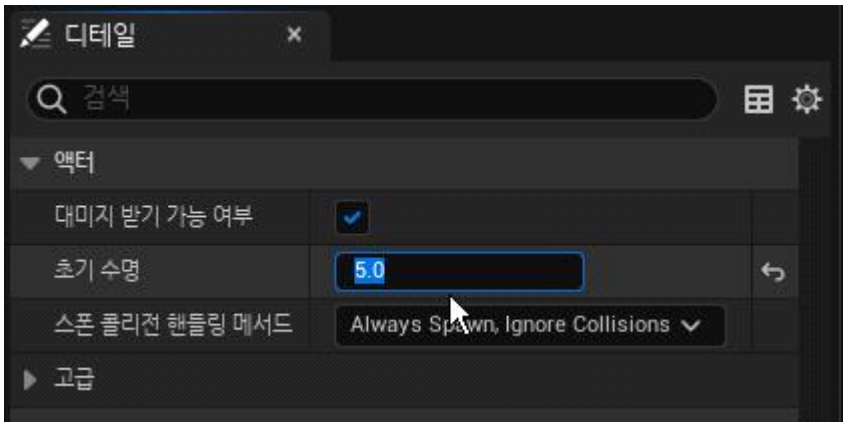
17. **콘텐트 브라우저**에서 **BP_MyPhysicsCube**를 드래그해서 레벨에 배치하자. 배치된 인스턴스의 이름을 **BP_MyPhysicsCube1**로 수정하자. 위치는 (300,200,300)에 배치하자.
레벨 에디터에서 플레이해보자. 오른쪽에 **BP_MyPhysicsCube1**가 낙하하기 시작할 것이다. 지면에 부딪히는 순간 사라질 것이다.
플레이 도중에 **아웃라이너** 탭을 관찰해보자. 액터가 생성되고 소멸되는 것을 관찰할 수 있을 것이다.



18. 또다시, **콘텐트 브라우저**에서 **BP_MyPhysicsCube**를 드래그해서 레벨에 배치하자. 배치된 인스턴스

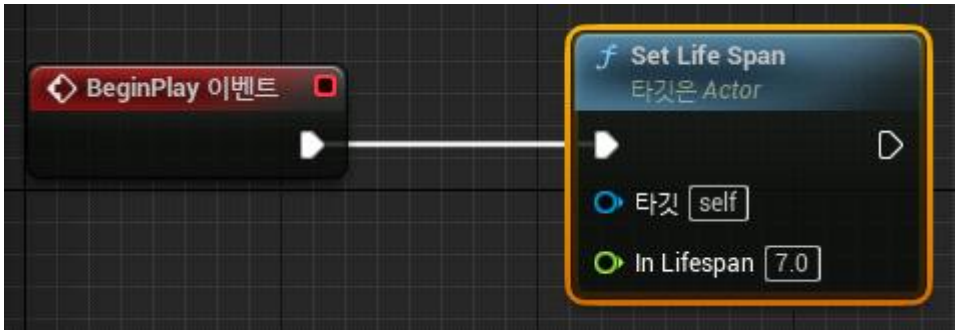
의 이름을 **BP_MyPhysicsCube2**로 수정하자. 이번에는 높고 아주 멀리 배치해보자. 위치는 (5000,1000,2000)에 배치하자. 바닥 영역을 벗어나서 배치된다. 레벨 에디터에서 플레이해보자. 오른쪽에 **BP_MyPhysicsCube2**가 멀리서 낙하하는 것이 보일 것이다.
그러나 이 액터는 바닥 영역과 멀리 떨어져 있기 때문에 바닥면에 부딪히지 않고 **Hit** 이벤트도 발생하지 않는다. 따라서 소멸되지 않고 계속 낙하한다. 플레이 도중에 **아웃라이너** 탭을 관찰해보자. **BP_MyPhysicsCube2**가 소멸되지 않고 계속 남아있는 것을 확인할 수 있다.

19. 액터의 수명을 명시해서 일정시간 후에 자동으로 소멸되도록 해보자. 수명을 명시하기 위해서는 **InitialLifeSpan** 속성에 원하는 값을 지정하면 된다.
먼저, **콘텐트 브라우저**에서 **BP_MyPhysicsCube**를 더블클릭하여 블루프린트 에디터를 열자. 툴바에서 **클래스 디폴트** 아이콘을 선택하고 **디테일** 탭에서 **액터** 영역에 있는 **초기 수명**(**InitialLifeSpan**) 속성값을 0에서 5로 수정하자.



만약 **디테일** 탭에서 속성이 보이지 않는다면, **컴포넌트** 탭에서 **BP_MyPhysicsCube**를 선택하거나 또는 툴바에서 **클래스 디폴트** 버튼을 클릭한 후에 다시 **디테일** 탭에서 찾아보자. 컴포넌트가 선택되어 있는 경우에는 클래스 전체의 속성이 아니라 컴포넌트 속성이 디테일 탭에 나타나기 때문이다. 수명은 컴포넌트의 개별 속성이 아니라 액터 수준의 클래스 속성이다.
이제, 레벨 블루프린트에서 플레이해보자.
아웃라이너 탭을 관찰해보면 5초 후에 **BP_MyPhysicsCube2**가 소멸되는 것을 확인할 수 있다.

20. 이번에는 수명을 동적으로 지정하는 방법을 알아보자.
먼저, **BP_MyCube**를 더블클릭하여 블루프린트 에디터를 열자.
그다음, 이벤트 그래프에서 **BeginPlay 이벤트** 노드의 실행핀을 드래그하고 **SetLifeSpan** 함수를 배치하자. 그다음, **InLifespan** 입력핀에 7을 입력하자.



컴파일하고 저장하자.

이제, 플레이해보자. **아웃라이너** 목록을 확인해보자. 중간과 왼쪽에 배치되어 있던 두 큐브가 7초 뒤에 모두 소멸할 것이다.

지금까지, 액터를 생성하고 소멸하는 방법에 대해서 학습하였다.

3. 폰의 리스폰

이 절에서는 폰을 리스폰하는 방법에 대해서 학습한다.
레벨이 시작하면 플레이어 폰은 최초에 자동으로 스폰된다. 이후에 플레이어 폰이 죽으면 부활하여 게임에 다시 참여하게 된다. 이것을 리스폰(respawn)이라고 한다. 이전 절에서의 액터와는 달리 폰은 단순히 다시 스폰해서 모든 문제가 해결되는 것은 아니다. 플레이어 폰은 플레이어 컨트롤러와 연관되어 있기 때문이다.

플레이어 폰과 플레이어 컨트롤러의 관계를 알아보자.
플레이어 컨트롤러와 플레이어 폰은 서로 일대일로 연결되어 있다.
플레이어 폰은 게임 플레이 시에 화면에 보이는 플레이어의 물리적인 모습을 표현하는 역할을 한다.
플레이어 컨트롤러는 게임을 제어하는 사람 플레이어와 플레이어 폰 사이의 인터페이스 역할을 한다. 즉, 플레이어 컨트롤러는 사람 플레이어의 의지를 나타내는 액터에 해당한다.
플레이어 폰과 플레이어 컨트롤러의 역할을 어떻게 나눌 것인지에 대해서는 개발자가 고민해야 할 부분이다. 간단한 입력의 경우에는 플레이어 컨트롤러를 거치지 않고 바로 플레이어 폰에서 모든 입력 처리하도록 구현할 수 있다. 이 경우에는 플레이어 컨트롤러는 가장 간단한 형태로 그대로 두면 된다. 한편, 플레이어 컨트롤러에서 구현하는 것이 유용한 경우가 있다. 하나의 클라이언트 컴퓨터에서 멀티 플레이어를 처리하는 경우나 플레이 도중에 캐릭터 폰을 바꾸는 경우 등이 해당된다.
가장 근본적인 차이는, 플레이어 컨트롤러는 레벨이 플레이되는 동안에 영구적으로 존재하지만 플레이어 폰은 중간에 소멸되고 다시 스폰될 수 있는 존재라는 점이다. 따라서 플레이어가 현재 점수 정보는 반드시 플레이어 컨트롤러에 저장해야 플레이어가 리스폰되더라도 유지될 수 있을 것이다.
이제, 플레이어 폰과 플레이어 컨트롤러의 연결에 대해서 알아보자.
게임이 시작되면 플레이어 컨트롤러가 플레이어 폰을 **빙의**(possess)하여 연결이 이루어진다. 만약 플레이어 폰이 소멸되는 경우에는 **빙의 해제**(unpossess)되어 연결이 끊어진다. 따라서 만약 플레이어 폰이 소멸되어 다시 플레이어 폰을 리스폰한다면, 새로 플레이어 폰이 생성된 후에 기존의 플레이어 컨트롤러가 새 플레이어 폰을 **빙의**하여 연결되도록 조치해야 한다.

이제, 플레이어 폰의 리스폰 과정을 실습을 통해서 학습해보자. 우리는 이전의 입력 처리를 다룬 장에서의 프로젝트 `Pinputevent`에서 시작할 것이다. 그러나 새 프로젝트를 만들고 그 프로젝트에서의 작업을 빠르게 옮긴 후에 시작할 것이다.

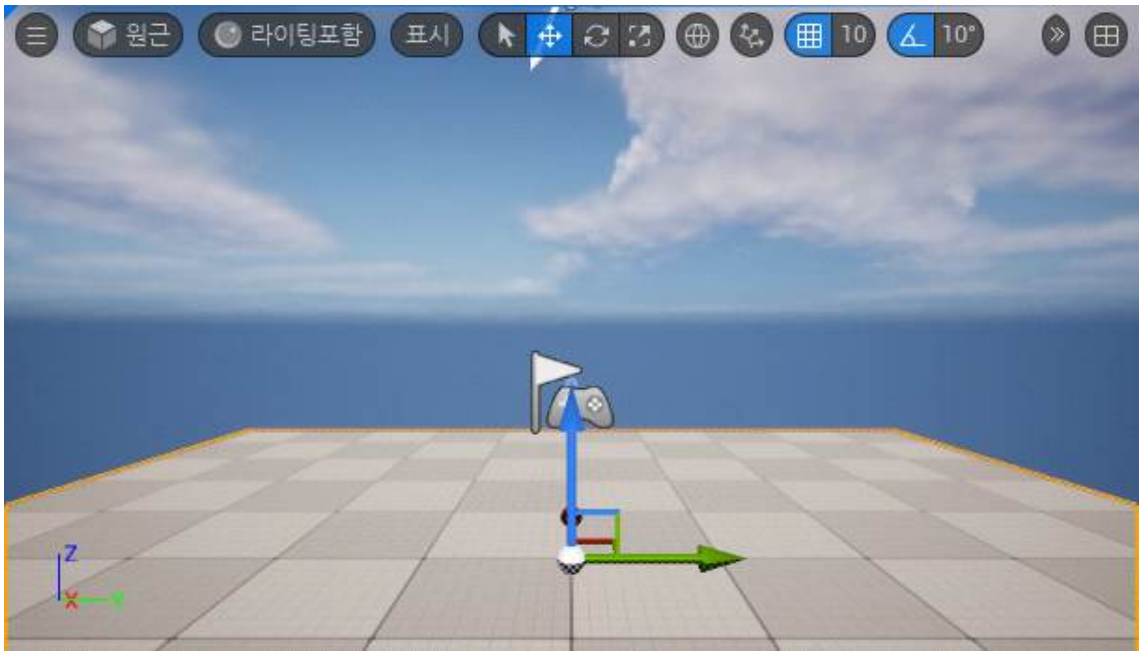
---

1. 새 프로젝트 `Prespawnpawn`를 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 `Prespawnpawn`으로 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.

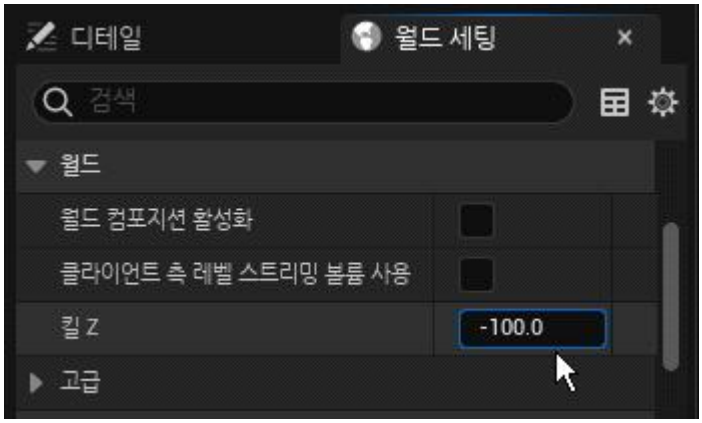
그리고, 툴바의 **액터 배치** 아이콘을 클릭하고 **기본** 아래의 **플레이어 스타트** 액터를 드래그하여 레벨에 배치하자. 위치를 (0,0,112)로 지정하자.

2. 이제 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 프로젝트을 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.

3. 바닥 메시가 너무 커서 테스트하기에 불편하다. 바닥 메시를 작게 줄이자. **아웃라이너** 탭에서 **Floor** 액터를 선택하자. **디테일** 탭에서 **스케일** 속성을 (8,8,8)에서 (1,1,1)로 수정하자.



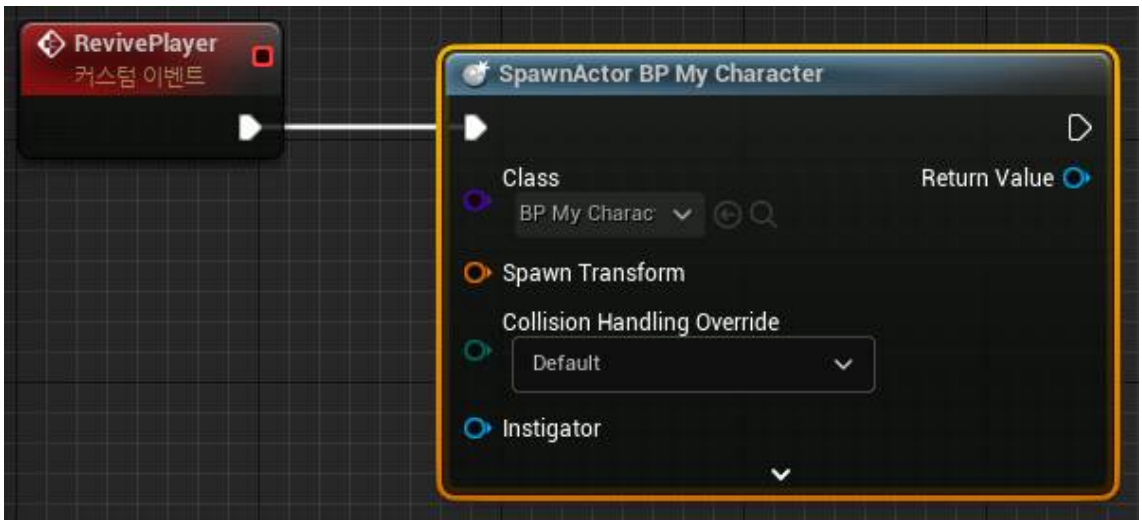
4. 먼저 플레이어 폰이 죽도록 해보자. 액터를 소멸시키는 또다른 방법인 **킬Z** (Kill Z) 방법을 사용해 보자. **월드 세팅** 탭에서 **월드** 영역에 **킬Z** 속성이 있다. 디폴트 값이 -1048575이다. 이것은 1단위가 1cm에 해당하므로 -10km에 해당하는 높이이다. 이는 너무 낮은 값이어서 효과적이지가 않다. 우리는 이 값을 -100으로 지정하자.



이제, 레벨을 플레이해보자.
플레이어 폰을 바닥 끝으로 이동하여 아래로 떨어지도록 해보자. 바닥 아래로 1m 떨어지는 지점에서 화면이 멈출 것이다. 엔진이 플레이어 폰을 강제로 소멸시켰음을 알 수 있다. **아웃라이너**를 살펴보면 플레이어 폰인 **BP_MyCharacter**가 소멸된 것을 확인할 수 있다.

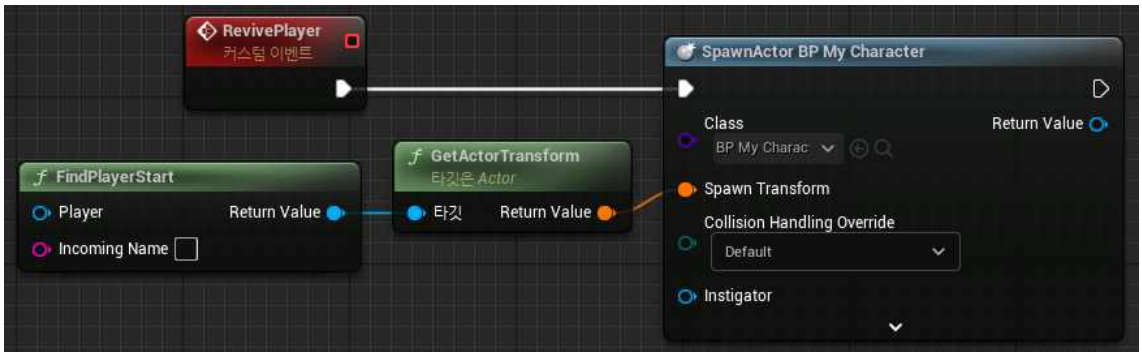
5. 플레이어가 바닥 끝으로 떨어질 때에 **아웃라이너**를 살펴보자. 플레이어 폰인 **BP_MyCharacter**는 소멸되지만 플레이어 컨트롤러인 **BP_MyPlayerController**은 소멸하지 않고 그대로 있는 것을 확인할 수 있다.
이제부터, 플레이어 폰을 리스폰하는 과정을 진행해보자.

6. 게임 모드에서 플레이어 폰을 리스폰하는 과정을 구현해보자.
먼저, **BP_MyGameMode**를 더블클릭하여 블루프린트 에디터를 열자.
그다음, 이벤트 그래프의 격자판의 빈 곳에서 우클릭하고 **커스텀 이벤트 추가**를 선택하여 커스텀 이벤트 노드를 배치하자. 노드 이름을 **RevivePlayer**이라고 하자.
그다음, **RevivePlayer** 이벤트 노드의 실행핀을 당기고 **SpawnActor fromClass**를 선택하여 **SpawnActor** 노드를 배치하자.
그다음, **SpawnActor** 노드의 Class 입력핀에서 **BP_MyCharacter**를 선택하자.

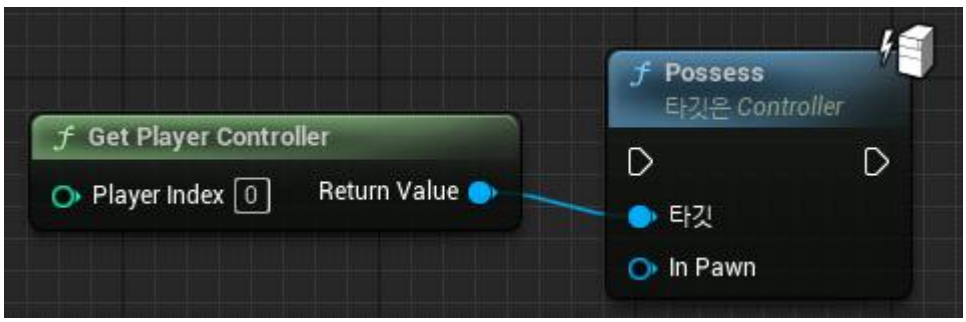


7. **SpawnActor** 노드의 **SpawnTransform** 입력핀에 스폰될 위치를 나타내는 트랜스폼을 넣어주어야 한다.

먼저, 격자판의 빈 곳에서 우클릭하고 **FindPlayerStart** 노드를 배치하자. 이 함수는 플레이어 폰이 스폰될 때에 사용되는 **PlayerStart** 노드를 리턴한다.
그다음, **FindPlayerStart** 노드의 **ReturnValue** 출력핀을 당기고 **GetActorTransform**을 배치하자. 이 함수는 액터의 트랜스폼(Transform)을 추출하는 함수이다.
그다음, **GetActorTransform**의 **ReturnValue** 출력핀을 **SpawnActor** 노드의 **SpawnTransform** 입력핀에 연결하자.

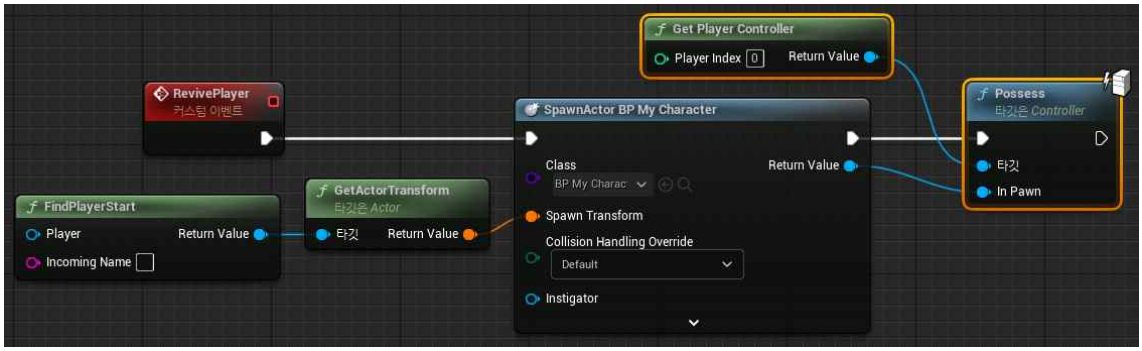


8. 이제, 새로 만든 플레이어 폰을 기존의 플레이어 컨트롤러에 연결하는 빙의(possess) 절차를 진행하자.
위의 **RevivePlayer** 그래프에서 작업을 계속한다.
먼저, 격자판의 빈 공간에서 우클릭하고 **GetPlayerController**를 배치하자. 이 함수는 현재 레벨의 플레이어 컨트롤러를 리턴한다.
그다음, **GetPlayerController**를 노드의 **ReturnValue** 출력핀을 드래그하고 **Possess** 함수 노드를 배치하자.



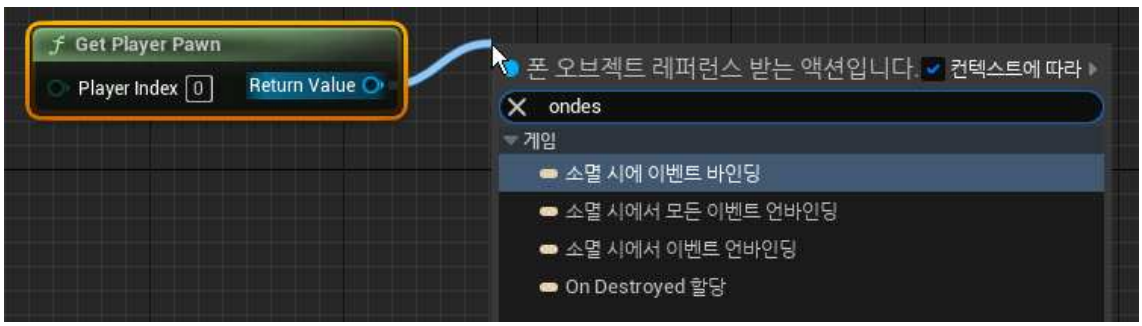
Possesss 노드의 **타깃** 입력핀에는 현재 레벨의 플레이어 컨트롤러가 연결되어 있다. **Possesss** 노드는 현재 플레이어 컨트롤러가 **InPawn** 입력핀에 연결되는 플레이어 폰을 **빙의**하도록 해준다. 하나의 컨트롤러는 하나의 폰만 빙의할 수 있다. 만약 컨트롤러가 이미 다른 폰을 빙의하고 있는 상태라면 그 폰을 **빙의 해제**(unpossess)한 후에 빙의한다.

9. 이제, **SpawnActor** 노드의 출력 실행핀을 **Possess** 노드의 입력 실행핀에 연결하자.
그다음, **SpawnActor** 노드의 **ReturnValue** 출력핀을 **Possess** 노드의 **InPawn** 입력핀에 연결하자.

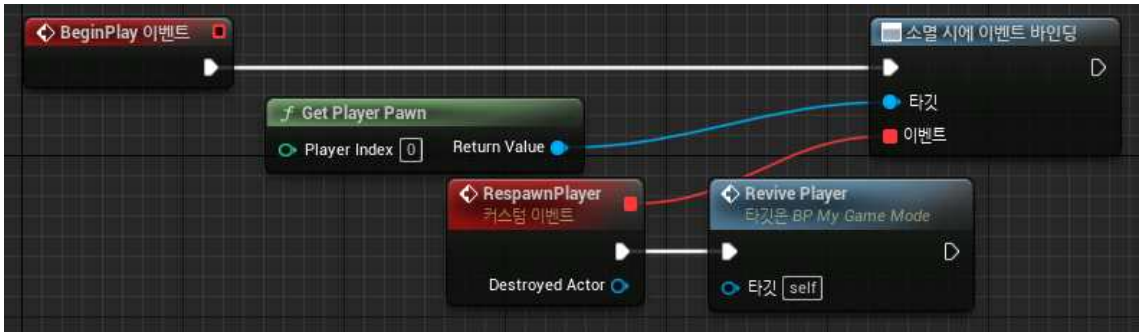


이제, **RevivePlayer** 이벤트 노드 그래프를 모두 완성하였다.

10. 이제, 플레이어 폰이 소멸될 때에, 부활하도록 **RevivePlayer** 이벤트 노드를 호출해주면 된다. **RevivePlayer** 이벤트 노드를 누가 어떻게 호출할 것인지에 대해서 생각해보자.
폰에는 **OnDestroyed** 이벤트 디스패처가 기본으로 포함되어 있다. 이 **OnDestroyed** 이벤트 디스패처는 폰 자신이 소멸될 때에 소멸 사실을 바인드된 함수들에 통지해준다. 따라서 우리는 이 이벤트 디스패처에 우리가 작성한 **RevivePlayer** 이벤트 노드가 호출되도록 바인드해주면 된다. 사실 이 기능은 폰뿐만 아니라 모든 액터에 대해 지원되는 기능이다.
따라서, 우리는 폰을 접근해서 **OnDestroyed** 이벤트 디스패처에 접근해서 바인드해 주어야 한다. 그런데, 플레이어 폰은 이미 레벨에 배치되어 있는 액터가 아니라, 플레이될 때에 스폰되는 액터이다. 따라서 게임이 시작된 이후에 스폰된 플레이어 폰을 가져와서 바인드 작업을 해주어야 한다.
우리는 이 작업을 게임 모드의 **BeginPlay 이벤트** 그래프에서 진행해보자.
먼저, **BeginPlay 이벤트** 노드의 근처에서 격자판의 빈 곳에 우클릭하고 **GetPlayerPawn**을 배치하자. 그다음, **GetPlayerPawn** 노드의 **ReturnValue** 출력핀을 드래그하고 **소멸 시(OnDestroyed)에 이벤트 바인딩**을 선택하자.

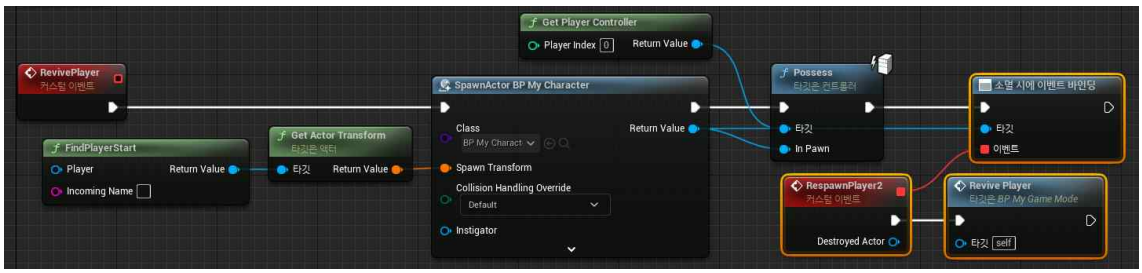


11. 이제, **BeginPlay 이벤트** 노드의 실행핀과 **OnDestroyed에 이벤트 바인딩** 노드의 실행핀을 연결하자. 그다음, **OnDestroyed에 이벤트 바인딩** 노드의 **이벤트** 델리게이트 핀을 왼쪽으로 당기고 **커스텀 이벤트 추가**를 선택하여 커스텀 이벤트 노드를 배치하자. 노드의 이름을 **RespawnPlayer**라고 하자.
RespawnPlayer 이벤트 노드의 출력 실행핀을 드래그하고 **RevivePlayer** 함수호출 노드를 배치하자.



이제 함수 호출 부분을 완료하였다.

12. 플레이하여 최초로 생성되는 플레이어 폰이 죽게 되면 **RevivePlayer** 함수가 호출되도록 바인드 되었다. 그러나, 만약 다시 생성된 플레이어 폰이 죽게 된다면 그 때에는 **RevivePlayer** 함수가 호출되지 않을 것이다. 왜냐하면 다시 생성된 플레이어 폰에는 바인드 작업이 없었기 때문이다.
따라서 다시 생성되는 플레이어 폰에도 바인드 작업을 해주어야 한다.
먼저, **RevivePlayer** 이벤트 그래프로 가자. 그래프 안에 **SpawnActor** 노드가 있을 것이다. 그 노드의 **ReturnValue** 출력핀을 드래그하고 **소멸 시(OnDestroyed)에 이벤트 바인딩** 노드를 선택하여 배치하자. **OnDestroyed에 이벤트 바인딩** 노드의 **이벤트** 델리게이트 핀을 왼쪽을 당기고 **커스텀 이벤트 추가**를 선택하여 커스텀 이벤트 노드를 배치하자. 노드의 이름을 **RespawnPlayer2**라고 하자.
RespawnPlayer2 이벤트 노드의 출력 실행핀을 드래그하고 **RevivePlayer** 함수호출 노드를 배치하자.
그리고, **Possess** 노드의 출력 실행핀을 **OnDestroyed에 이벤트 바인딩** 노드의 입력 실행핀에 연결하자.



13. 플레이해보자.
바닥에서 떨어져서 죽도록 해보자. 계속 리스폰될 것이다.

지금까지, 플레이어 폰을 리스폰하는 방법을 학습하였다.

□