11_ 커스텀 이벤트와 이벤트 디스패처

<제목 차례>

인천대학교 컴퓨터공학부 박종승

1. 개요

우리는 이전 장에서 이벤트 노드에 대해서 다루었다.
먼저, **BeginPlay** 노드, **Tick** 노드에 대해서 다루어보았다. 이 노드들은 객체가 생성될 때나 매 프레임 갱신 틱 신호 시에 호출되는 이벤트 노드이다.
또한, 입력 이벤트 노드에 대해서 다루어보았다. 이 노드들은 입력장치로부터의 입력 이벤트가 발생될 때에 호출되는 이벤트 노드이다.
또한, **OnActorBeginOverlap** 노드와 **OnActorEndOverlap** 노드에 대해서도 다루어보았다. 이 노드들은 액터에 대한 겹침시작 이벤트나 겹침끝 이벤트가 발생될 때에 호출되는 이벤트 노드이다.
또한, **OnComponentBeginOverlap** 노드와 **OnComponentEndOverlap** 노드에 대해서도 다루어보았다. 이 노드들은 액터 내의 컴포넌트에 대한 겹침 이벤트가 발생될 때에 호출되는 이벤트 노드이다.
이러한 이벤트 노드는 모두 스크립트 코딩에서 흐름도의 시작 지점에 해당하는 노드이다. 즉, 이벤트 노드는 노드 네트워크의 첫 번째 노드에 해당하는 특별한 노드이다. 노드의 컬러도 특별하게 붉은색으로 표시된다.
블루프린트 스크립트 코딩은 관심있는 각 이벤트 노드에 대해서 노드 네트워크를 작성하는 것에 해당한다.

한편, 게임 엔진에서 기본으로 제공하는 이벤트 노드 이외에도 우리가 직접 이벤트 노드를 정의할 수도 있다. 게임 엔진에서 모든 이벤트 노드를 제공하는 것이 아니기 때문에 우리가 원하는 게임을 만들기 위해서는 자신이 필요한 이벤트 노드를 직접 만들 수 있어야 한다. 이러한 사용자가 정의한 이벤트 노드를 **커스텀 이벤트** 노드라고 한다.
커스텀 이벤트 노드를 정의하는 것은 일반적인 함수를 정의하는 것과 거의 유사한 개념이다. 이들은 약간의 차이가 있고 그 차이점에 대해서는 나중에 학습한다. 그 전까지 우리는 커스텀 이벤트 노드 그래프를 작성하는 것을 함수를 작성하는 것과 동일한 개념으로 이해하자.
커스텀 이벤트 노드 그래프를 실행시키기 위해서는 함수를 호출하는 것과 동일한 방법으로 **커스텀 이벤트** 노드를 호출하는 위치에 배치하면 된다. 액션선택 창에서 **함수 호출** 카테고리 아래에 있는 이벤트 이름을 선택하면 된다. 노드 네트워크의 시작인 이벤트 노드는 붉은색으로 표시되지만 이벤트 그래프를 호출하는 함수 호출 노드는 파란색으로 표시된다.
이 장에서는 커스텀 이벤트 노드를 생성하고 사용하는 방법에 대해서 학습한다.

2. 커스텀 이벤트 노드 생성하고 호출하기

이 절에서는 커스텀 이벤트 노드를 생성하고 호출하는 방법에 대해서 학습한다.

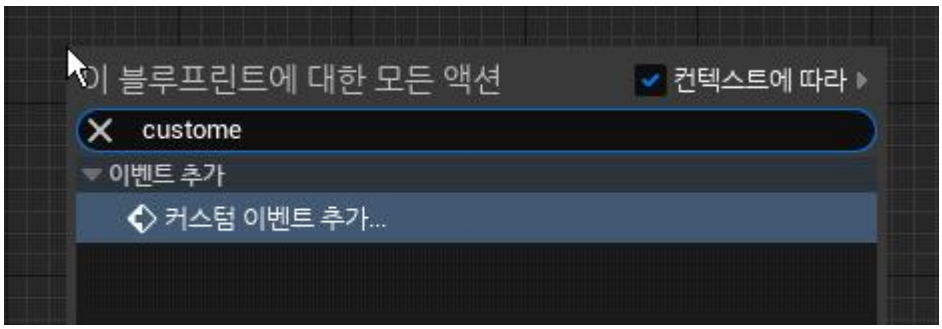
---

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

2. 액터의 파생 블루프린트 클래스를 생성하자.
콘텐츠 브라우저에서 **+추가**를 클릭하고 **블루프린트 클래스**를 선택하자. **부모 클래스 선택** 창에서 **Actor**를 선택하자. 이름을 **BP_MyCube**로 입력하자.

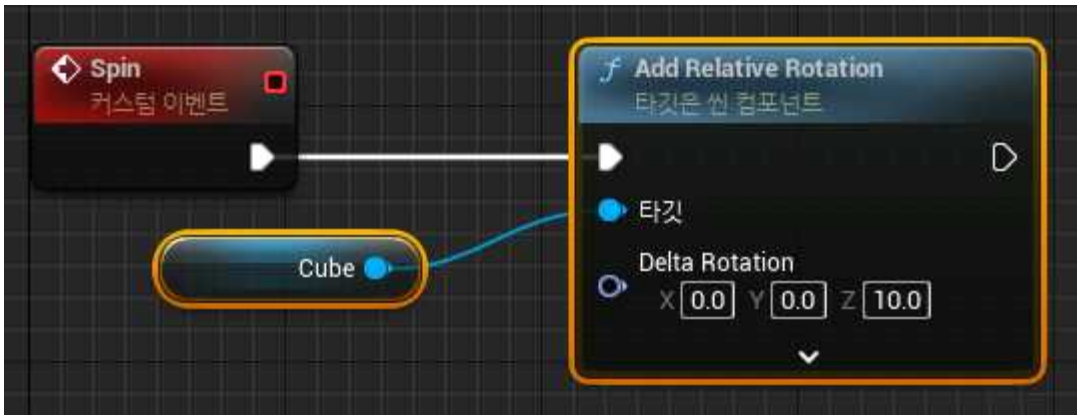
3. **BP_MyCube**를 더블클릭하여 블루프린트 에디터를 열자. 왼쪽의 **컴포넌트** 탭에서 **+추가**를 클릭하고 **큐브**를 선택하자. 큐브 모양의 스태틱 메시 컴포넌트인 **Cube**가 추가될 것이다.

4. 이제, 커스텀 이벤트 노드를 추가해보자.
먼저, 이벤트 그래프 탭을 클릭하자.
격자판의 빈 곳에서 **AddCustomEvent**를 검색하여 **AddEvent** 아래에 있는 **커스텀 이벤트 추가**를 선택하자. 붉은색의 **커스텀 이벤트** 노드가 배치될 것이다.



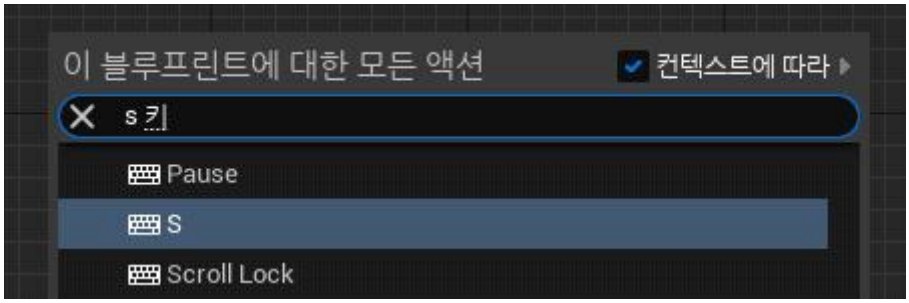
5. 이제, **커스텀이벤트_0**의 식으로 자동으로 정해지는 이름을 **Spin**으로 수정하자. 내 블루프린트 탭의 이벤트 그래프 영역에서의 나열에도 추가될 것이다.

6. 이제, **Spin** 이벤트 노드의 출력 실행핀을 당기고 검색하여 액션선택 창에서 **AddRelativeRotation (Cube)**를 선택하자. **Cube** 컴포넌트의 레퍼런스 노드와 **AddRelativeRotation** 함수 노드가 함께 배치될 것이다. **AddRelativeRotation** 함수 노드의 **DeltaRotation** 입력핀의 **Z**값을 10으로 수정하자.



7. 이제, 레벨 에디터에서 **BP_MyCube**를 레벨에 배치하자. 위치는 (0,0,100)에 배치하자.

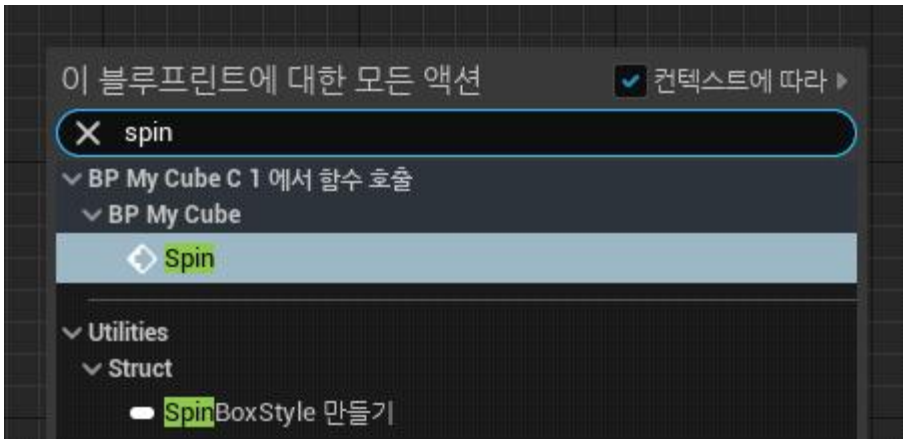
8. 이제, 레벨 에디터의 툴바에서 **블루프린트**를 클릭하고 **레벨 블루프린트 열기**를 선택하고 레벨 블루프린트 에디터를 열자.
이벤트 그래프 탭의 격자판의 빈 곳에서 우클릭하고 액션선택 창에서 **Input** » **키보드 이벤트** 아래에 있는 **S**를 선택하자. 키보드 **S** 키 입력 이벤트 노드가 배치될 것이다.
참고로, 액션선택 창에서 검색 기능을 사용하는 경우에는 너무 많은 목록이 표시될 것이다. 이때에는 카테고리 단어의 일부를 추가로 입력해주면 범위가 좁아진다.



9. 이제, **아웃라이너**에서 배치된 **BP_MyCube** 액터를 드래그해서 **레벨 블루프린트 에디터**의 이벤트그래픽 격자판에 드롭하여 레퍼런스 노드를 배치하자. 그다음, 레퍼런스 노드의 출력핀을 드래그하고 액션선택 창에서 **Spin** 함수 호출 노드를 검색하여 배치하자.

우리는 빨간색의 **Spin** 이벤트 노드와 파란색의 **Spin** 함수 호출 노드를 확실하게 구분해야 한다. 빨간색의 이벤트 노드는 이벤트가 발생할 때에 실행할 노드 그래프를 정의하는 노드이다. 즉, 실행할 작업 흐름도의 시작 위치에 해당하는 노드이다. 반면, 파란색의 함수 호출 노드는 노드 그래프를 실행하라는 호출 노드이다.

10. 위의 방법 외에도 **Spin** 이벤트 노드를 배치하는 다른 방법이 있다. 먼저, **레벨 에디터**의 **아웃라이너**나 뷰포트에서 **BP_MyCube** 액터를 선택하자. 그다음, **레벨 블루프린트 에디터**의 이벤트 그래프 격자판에서 우클릭하고 액션선택 창에서 **Spin** 이벤트 노드를 검색하면 **BP_MyCube_C_1에서 함수 호출 » BP_MyCube** 아래에 **Spin** 이벤트 노드가 나열된다. 이것을 선택하면 레퍼런스 노드와 함수 노드가 연결된 채로 함께 추가된다. 여기서 **BP_MyCube_C_1**는 레벨에 배치된 **BP_MyCube** 노드의 인스턴스 ID 이름으로 현재 선택된 액터를 의미한다.

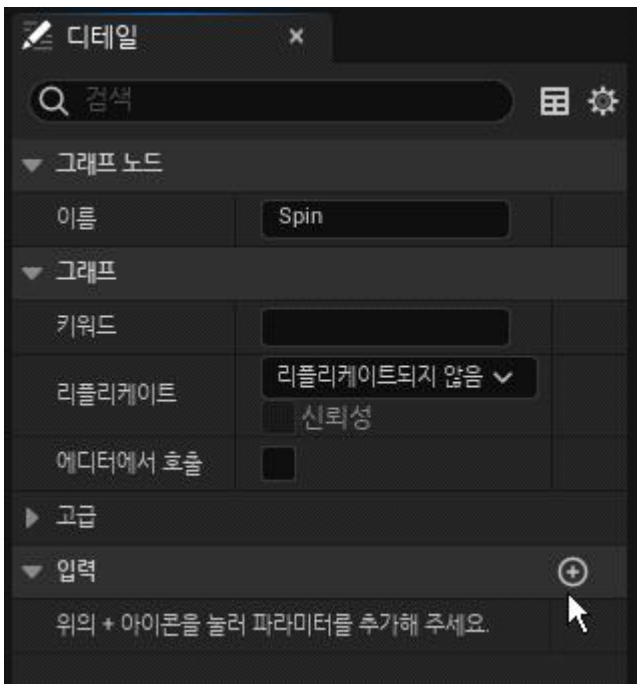


11. 이제, **S** 키 입력 이벤트 노드의 **Pressed** 출력 실행핀에 **Spin** 함수 호출 노드를 연결하자.

키보드 **S** 키 입력 이벤트 노드에는 출력 실행핀이 두 개가 있다. 키보드가 눌러질 때에는 **Pressed** 출력 실행핀에서 실행 신호가 발생하고, 키보드 키가 떼질 때에는 **Released** 출력 실행핀에서 실행 신호가 발생한다.

12. 레벨을 플레이해보자. **S** 키를 눌러보자. 상자가 10도 회전할 것이다. 여러 번 눌러보자. 연속해서 10도 단위로 회전할 것이다. 위에서 내려다보았을 때에 시계방향으로 회전한다.

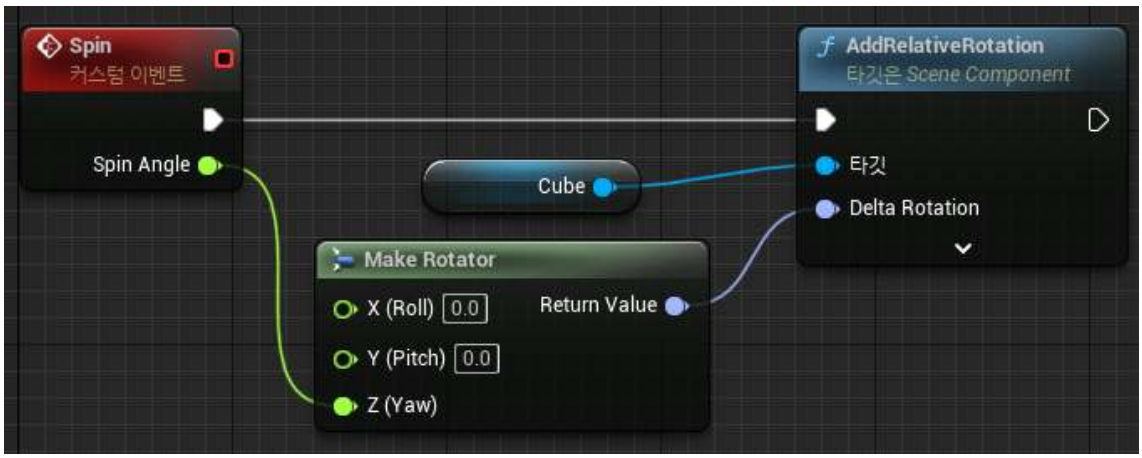
13. 한편, 이벤트 노드에 입력 인자를 추가할 수 있다. 스핀 각도를 10도로 고정하였는데 이를 입력 인자로 받아서 그 각도만큼 스핀하도록 해보자.
BP_MyCube 블루프린트 에디터로 가자. **Spin** 커스텀 이벤트 노드를 선택하자. 오른쪽의 **디테일** 탭을 보면 **입력** 영역이 있다. 그 오른쪽에 + 아이콘을 클릭하자.



14. 하나의 인자가 추가된다. 인자 이름은 디폴트인 **NewParam**을 **SpinAngle**로 수정하자. 그 오른쪽의 타입은 디폴트인 **부울(Boolean)**을 **플로트**로 수정하자. 그 아래를 펼치면 **디폴트 값** 속성이 있다. **디폴트 값**은 0에서 10으로 수정하자.

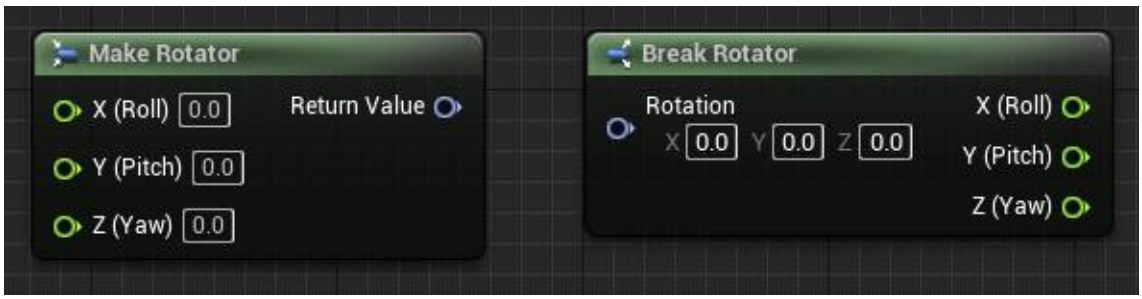
컴파일하자. 이벤트 그래프에서 **Spin** 커스텀 이벤트 노드를 살펴보자. **SpinAngle** 이름의 출력핀이 생겼을 것이다.

15. 격자판의 빈 곳에서 우클릭하고 검색하여 **MakeRotator** 노드를 추가하자.
Spin 이벤트 노드의 **SpinAngle** 출력핀을 **MakeRotator** 노드의 **Z(Yaw)** 입력핀에 연결하고 **ReturnValue** 출력핀을 **AddRelativeRotation** 노드의 **DeltaRotation** 입력핀에 연결하여 다음과 같이 그래프를 구성하자.



컴파일하자.

16. 여기서, **MakeRotator**에 대해서 더 자세히 알아보자.
위의 그래프에서 **AddRelativeRotation** 함수의 입력 인자 **DeltaRotation**의 타입은 **Rotator**라는 구조체 타입이다. 이 구조체의 멤버에는 X,Y,Z의 세 축에 대한 회전값을 가지는 실수가 있다. 모든 회전값은 라디안(radian)이 아닌 도(degree) 단위로 표현된다.
세 실수값으로부터 구조체를 만들어주는 편리 함수가 **MakeRotator**이다. 즉 **MakeRotator**의 입력은 X,Y,Z축에 대한 회전값이고 출력은 **Rotator** 구조체이다.



한편, 이와 반대로 구조체를 입력으로 하고 그 구조체의 멤버 변수들을 출력으로 하는 **BreakRotator**도 있다. 즉, **BreakRotator**는 입력이 **Rotator** 구조체이고 출력이 X,Y,Z축에 대한 회전값이다. 이런식으로, 대부분의 구조체에 대해서, **Make**와 **Break**가 구조체명 앞에 붙은 편리 함수가 있다.

17. 레벨 블루프린트 에디터로 가자. **Spin** 함수 호출 노드를 살펴보자. 입력핀 **SpinAngle**이 생겼고 디폴트 값이 10으로 되어 있음을 확인할 수 있다. 우리는 이번에는 10을 -10으로 수정해보자.

컴파일하자.

18. 레벨을 플레이해보자. **S** 키를 눌러보자. 여러 번 눌러보자. 이번에는 음수 각도를 지정하였으므로 위에서 내려다보았을 때에 반시계방향으로 회전할 것이다.

지금까지, 커스텀 이벤트 노드를 생성하고 이를 호출하는 방법을 학습하였다.

3. 이벤트 디스패처 생성하기

이 절에서는 이벤트 디스패처에 대해서 학습한다.

<참고> 먼저 디스패처라는 단어에 대해서 알아보자. 디스패처(dispatcher)의 사전적 의미는 항공기 관제사나 '지령 요원'의 역할을 하는 사람을 의미한다. 이들은 항공관제실, 경찰서, 소방서, 병원 응급실, 택시 호출 사무소, 기차역 등에서 근무하면서 특정 사건 정보나 서비스 요청을 수신하고 이를 처리하거나 해결할 수 있도록 관련 부서나 담당자에게 연락하는 역할을 한다.
디스패치(dispatch)라는 단어에 대해서도 알아보자. 디스패치의 사전적인 의미는 고객의 서비스 요청에 대해 담당자나 차량을 배정하는 절차를 의미한다. 담당자나 차량을 출동시키거나 파견하거나 물건을 발송하는 의미도 있다.
코딩에서의 이벤트 디스패처와 디스패치의 의미도 알아보자. 이벤트 디스패치는 새로운 이벤트를 '발송'하는 의미로 해석하면 된다. 즉 새 이벤트를 발생시킨다는 의미이다. 이벤트 디스패처는 발생된 이벤트가 감지되면 감지되었다는 사실을 해당 이벤트를 담당하는 모든 관련자들에게 알려준다.

이벤트 디스패처(event dispatcher)는 액터간의 통신하는 방법이다.
한 액터는 이벤트를 발송하고, 그 이벤트에 귀를 기울이는 다른 여러 액터는 발송된 이벤트를 통지받는다. 그 절차를 더 자세히 알아보자. 먼저, 이벤트를 발송하는 액터는 **이벤트 디스패처**(event dispatcher)라는 객체를 만든다. 여기서, 발송(dispatch)을 다른 말로 통지(notify)라고도 한다. 그다음, 이벤트 디스패처에 귀를 기울이고자(listen) 하는 액터들은 그 이벤트 디스패처에 자신을 등록(subscribe)한다. 등록 과정을 바인드(bind)라고 한다. 등록한 액터들을 구독자(subscriber)라고도 한다.
이후에, 액터가 이벤트 디스패처에게 발송을 지시하면 이벤트 디스패처는 자신에 등록된 여러 액터에게 통지한다. 이벤트 디스패처는 이러한 방식으로 하나의 액터가 여러 액터들에게 정보를 전달하는 통신 방식이다.

이번 절에서 다루는 예제에 대해서 미리 살펴보자.
먼저 예제의 시나리오는 다음과 같다. 플레이어가 깃발을 정복하게 되면 승리하는 게임을 생각해보자. 플레이어가 깃발 영역에 도달하면 깃발을 담당하는 액터는 다른 액터들에게 플레이어가 승리하였음을 알린다. 다른 액터들 중에서 다음 레벨로 통하는 문에 대한 액터는 승리 소식을 듣게 되면 다음 레벨로 진입할 수 있도록 문이 열리도록 한다. 또다른 액터인 스테이지 주변장식에 대한 액터는 승리 소식을 듣게 되면 축하하는 의미의 장식 움직임을 일으킨다.
예제의 시나리오를 구현하기 위한 방법을 정리하면 다음과 같다.
우리는 **BP_Flagpole**, **BP_DoorToNextLevel**, **BP_StageOrnament**의 세 블루프린트 클래스를 만들 것이다. 이 중에서 **BP_Flagpole** 액터는 이벤트 디스패처를 가지고 있는 클래스이다. 이벤트 디스패처의 이름은 **OnFlagCaptured**이다. 그리고 **BP_DoorToNextLevel**와 **BP_StageOrnament**는 이벤트 디스패처 **OnFlagCaptured**의 구독자 클래스에 해당한다.
깃발을 담당하는 **BP_Flagpole** 액터에서는 겹침시작 사건이 발생하면 플레이어가 승리했다는 의미이므로 **OnFlagCaptured** 이벤트 디스패처의 호출 노드를 실행하여 승리 소식을 구독자에게 통지한다. 통지하는 방법은 구독자가 미리 바인드해둔 이벤트 함수를 모두 실행해주는 것으로 수행된다.
모든 구독자 클래스는 내부에 커스텀 이벤트인 **Win**을 정의하고 있다. 각 클래스마다 자유로운 이름으로 정해도 되지만 우리는 편의상 모두 **Win**으로 하였다. 그리고, 구독자의 **BeginPlay 이벤트** 노드의 그래프에서 **Win** 이벤트 노드를 **OnFlagCaptured** 이벤트 디스패처에 바인드해둔다.
각 구독자 클래스에서 **Win** 이벤트 그래프가 하는 일은 각각 다르다. **BP_DoorToNextLevel**에서는 **Win**

이벤트 노드는 자신의 **OpenDoor** 이벤트 함수를 호출하여 다음 스테이지로 가는 문을 열도록 한다. **BP_StageOrnament**에서는 **Win** 이벤트 노드는 자신의 **CelebrateVictory** 이벤트 함수를 호출하여 승리를 축하하는 주변장식의 움직임을 발생시킨다.
한편, 구독자 클래스에서 **BP_Flagpole** 클래스에 있는 **OnFlagCaptured** 이벤트 디스패처에 접근하기 위한 방법이 필요하다. 이를 위해서 각 구독자 클래스는 **BP_Flagpole** 타입의 변수 **MyVictoryMessenger**를 선언하고 블루프린트 에디터에서 속성값을 지정할 수 있도록 허용하고 있다. 우리는 레벨 에디터에서 레벨에 **BP_Flagpole** 액터와 구독자 클래스 액터를 모두 레벨에 배치한 후에 속성값 지정 작업을 해 주어야 한다. 즉, 배치된 각 구독자 클래스의 인스턴스에 대해서 디테일 탭에서 **MyVictoryMessenger** 속성값에 레벨에 배치된 **BP_Flagpole** 인스턴스를 명시해주어야 한다.
이제 예제를 통해서 하나씩 구현해보자.

<참고> 이벤트 디스패처에 대해서는 다음의 문서를 참조하자.
https://docs.unrealengine.com/event-dispatchers-in-unreal-engine/

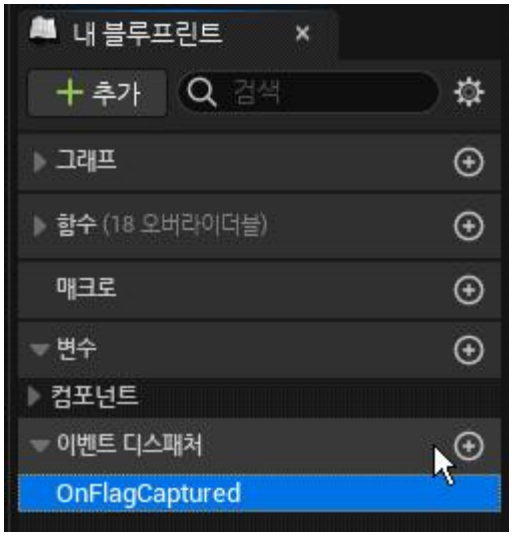
<참고> 블루프린트에서의 이벤트 디스패처는 C++ 클래스에서의 델리게이트(delegate)에 해당한다.

---

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

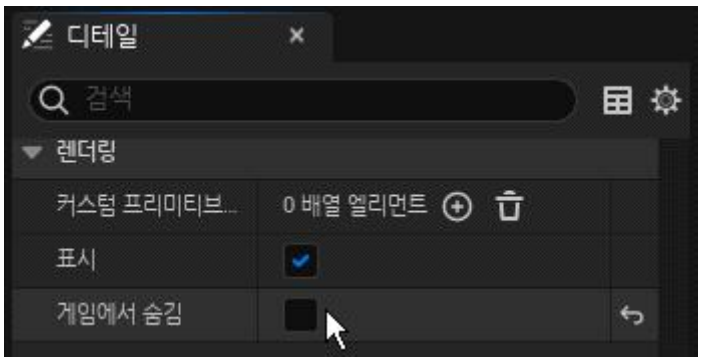
2. 먼저, 블루프린트 클래스를 만들자. 이 블루프린트 클래스에서는 이벤트 디스패처를 만들고 또한 이벤트가 발생하면 이를 구독자에게 통지할 것이다.
콘텐츠 브라우저에서 툴바에서 **+추가**를 클릭하고, 드롭다운메뉴에서 **블루프린트 클래스**를 선택하고, **부모 클래스 선택** 창에서 **Actor**를 부모 클래스로 선택하자. 이름은 **BP_Flagpole**로 수정하자.

3. **BP_Flagpole**을 더블클릭하여 블루프린트 에디터를 열자.
이제 **이벤트 디스패처**를 추가해보자. **내 블루프린트** 탭에서 가장 아래에 **이벤드 디스패처** 영역이 있다. 오른쪽의 + 버튼을 클릭하여 **이벤트 디스패처**를 추가하자. 이름은 **OnFlagCaptured**로 하자.

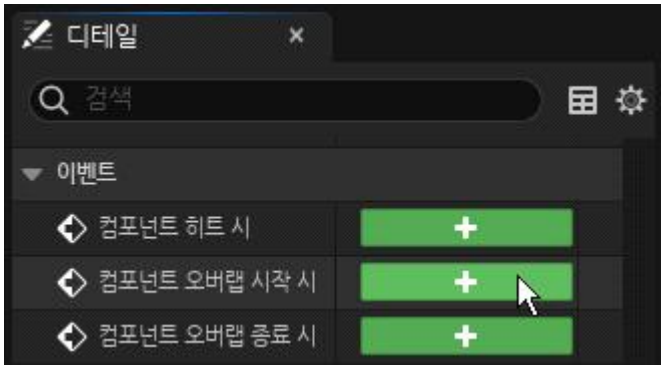


이제 **OnFlagCaptured**라는 **이벤트 디스패처**가 준비되었다.

4. 이제부터, 이벤트 디스패처를 호출하여 발송을 지시하는 과정을 진행해보자.
우리는 플레이어가 깃발을 획득하면 이기는 것으로 가정하자. 이를 위해서, 이 블루프린트에서 한 콜리전 박스를 두고 여기서 겹침시작 이벤트가 발생하면 이벤드 디스패처를 호출하자.
먼저, **컴포넌트** 탭에서 **+추가**를 클릭하고 **BoxCollision** 컴포넌트를 추가하자. 추가된 컴포넌트 이름은 디폴트로 **Box**로 지정되며 그대로 두자.
그다음, **디테일** 탭에서 **렌더링** 영역의 **게임에서 숨김(Hidden in Game)** 속성을 체크해제하자. 이 속성이 체크해제되면 게임 플레이 시에 콜리전 박스가 보이게 된다.

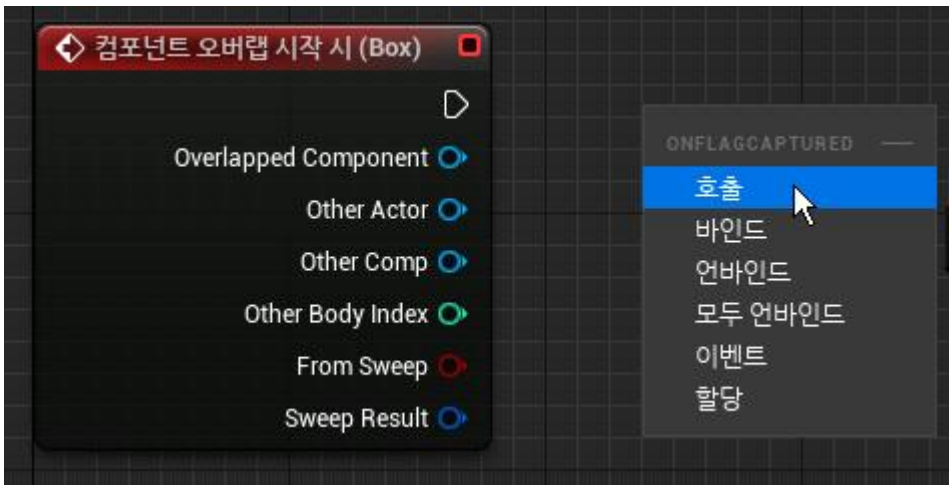


5. 그다음, **컴포넌트** 탭에서 **Box**를 우클릭하고 **이벤트 추가** » **OnComponentBeginOverlap** 추가를 선택하자. 또다른 방법으로, **디테일** 탭에서 이벤트 영역의 **컴포넌트 오버랩 시작 시(OnComponentBeginOverlap)**을 클릭해도 된다.



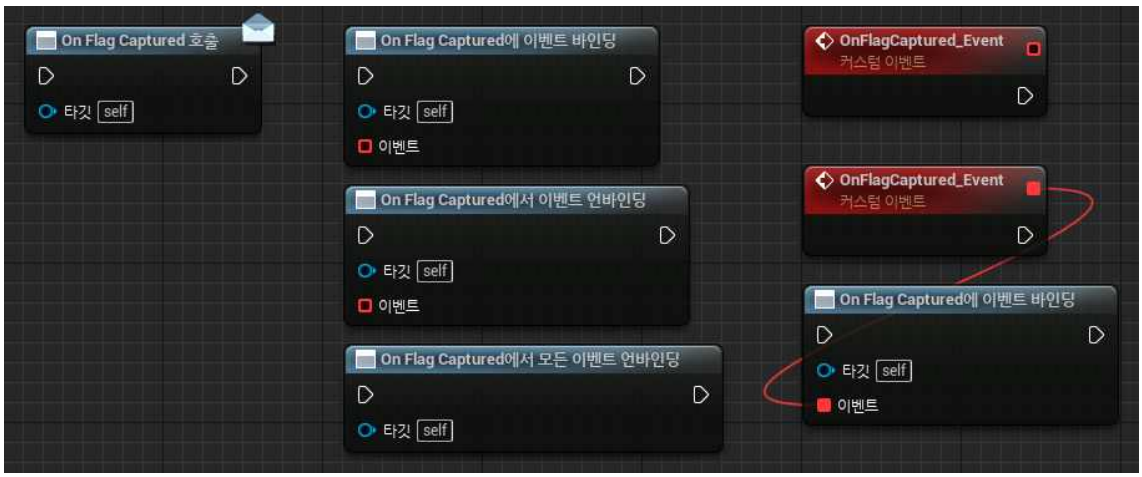
이제, **이벤트 그래프**에 **OnComponentBeginOverlap** 이벤트 노드가 추가되어 있을 것이다.

6. 그다음, **내 블루프린트** 탭에서 이벤트 디스패처 **OnFlagCaptured**를 드래그해서 격자판에 드롭하자. 드롭하면 여러 옵션 중 하나를 선택하도록 선택창이 뜬다. 여기서 **호출**을 선택하자. **OnFlagCaptured 호출** 노드가 배치될 것이다.



이벤트 디스패처 호출 노드가 실행되면 구독자에게 통지하도록 지시한다.

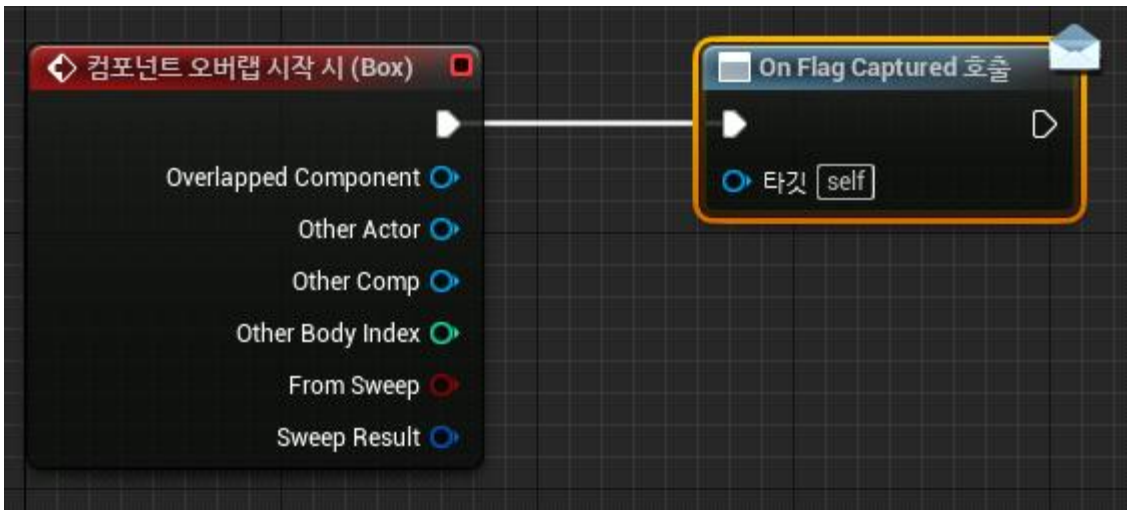
7. 계속 진행하기 전에, 이벤트 디스패처 배치와 관련된 다른 노드에 대해서도 알아보자. 아래의 그림은 참고용이므로 실제로 배치하지 말고 참고만 하자.



먼저, **호출**(call)을 선택하면 이벤트 디스패처 호출 노드를 배치된다. 이 노드는 이벤트 디스패처에

바인드된 모든 이벤트 노드를 호출하여 실행되도록 한다. 이 과정은 구독자에게 배달하는 과정에 해당한다.
그다음, **바인드**(bind)는 이벤트 바인딩 노드를 배치한다. 이 노드는 입력되는 이벤트 노드를 이벤트 디스패처에 바인딩한다. 이 과정은 구독 신청에 해당한다.
그다음, **언바인드**(unbind)는 이벤트 언바인드 노드를 배치한다. 이 노드는 입력되는 이벤트 노드를 언바인딩한다. 이 과정은 구독 취소에 해당한다.
그다음, **모두 언바인드**(unbind all)는 바인드된 모든 구독자를 취소하는 노드를 배치한다.
이제, 구독자가 바인드하는 이벤트 노드의 형식에 대해서 생각해보자.
각 이벤트 노드는 입출력 인자를 가질 수 있다. 입출력 인자의 개수와 각 인자의 타입 형식을 **시그니처**(signature)라고 한다. 시그니처는 C++에서 함수 원형과 유사한 의미이다. 한편, 모든 시그니처의 이벤트 노드를 이벤트 디스패처에 바인드할 수 있는 것이 아니다. 이벤트 디스패처에서 요구하는 시그니처와 동일한 이벤트 노드만 이벤트 디스패처에 바인드할 수 있다.
옵션 선택창의 그다음에 있는, **이벤트**(event)는 이벤트 디스패처에서 요구하는 시그니처와 동일한 이벤트 노드를 생성하여 배치한다.
그다음에 있는, **할당**(assign)은 **이벤트**와 **바인드**를 한번에 실행하는 것과 같다. 또한 배치되는 두 노드의 이벤트 핀을 연결된 채로 배치해주므로 편리하다.

8. 그다음, 실행핀을 연결하자. 이제 겹침시작 이벤트가 발생하면 이벤트 디스패처에 바인드된 모든 구독자의 이벤트 노드들을 실행해준다. 컴파일하고 저장하자.



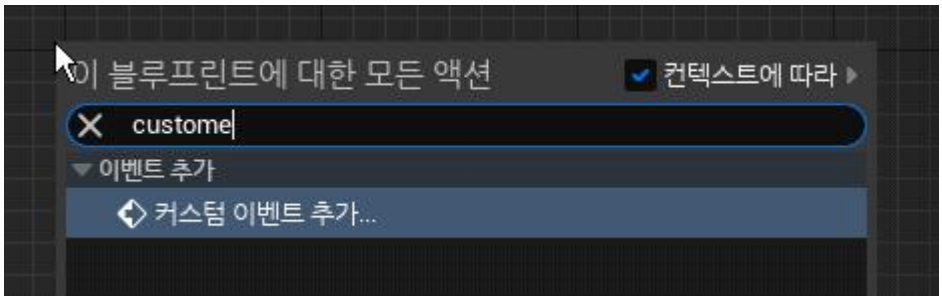
9. 레벨 에디터에서 콘텐츠 브라우저에서 **BP_Flagpole**를 드래그해서 레벨에 배치하자. 위치는 (200,0, 60)에 배치하자.

10. 이제, 첫 번째 구독자 액터를 블루프린트 클래스로 만들어보자.
콘텐츠 브라우저에서 툴바에서 **+추가**를 클릭하고, 드롭다운 메뉴에서 **블루프린트 클래스**를 선택하고, **부모 클래스 선택** 창에서 **Actor**를 부모 클래스로 선택하자. 이름은 **BP_DoorToNextLevel**로 수정하자.

11. **BP_DoorToNextLevel**를 더블클릭하여 블루프린트 에디터를 열자.
컴포넌트 탭에서 **+추가**를 클릭하고 **큐브**를 선택하여 큐브 스태틱 메시 컴포넌트를 추가하자. 이름은 디폴트로 되는 **Cube**로 그대로 두자.
그다음, 문 모양이 되도록 **디테일** 탭에서 **트랜스폼**에서 **스케일**의 **X**값을 0.1로 수정하고 **Z**값을 2로

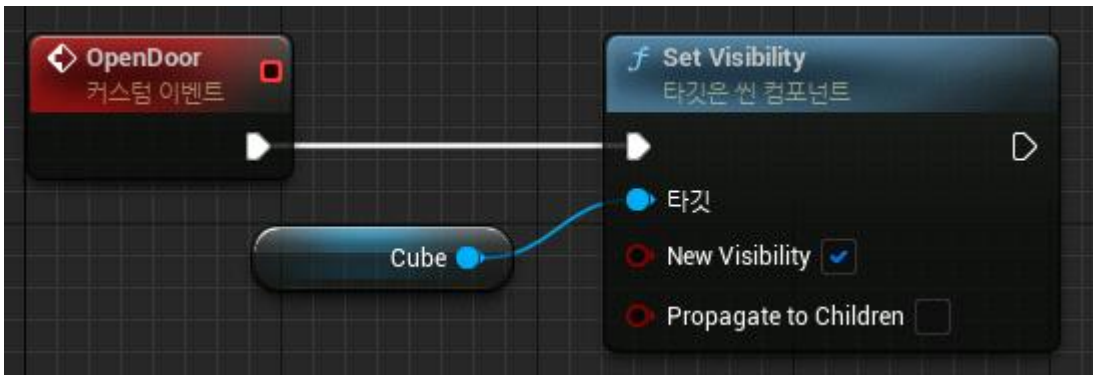
수정하자.

12. 이벤트 그래프 탭에서 격자판의 빈 곳에서 우클릭하고 액션선택 창에서 검색하여 **커스텀 이벤트**를 추가하자.

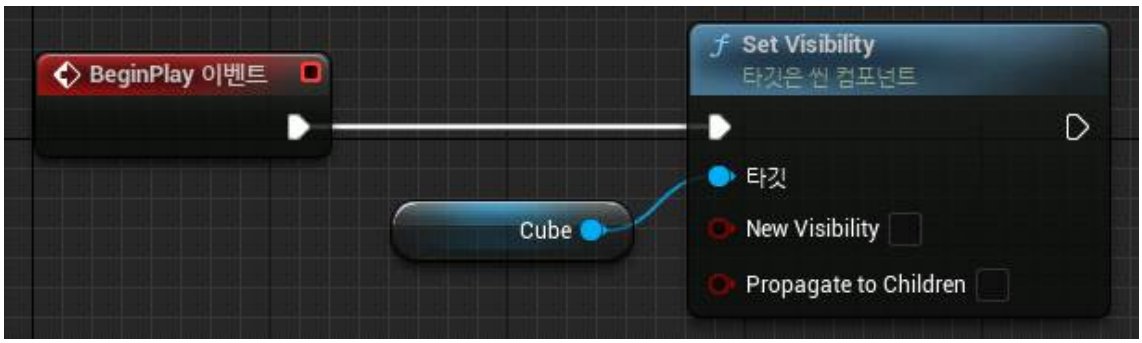


추가된 커스텀 이벤트의 이름을 **OpenDoor**로 수정하자.

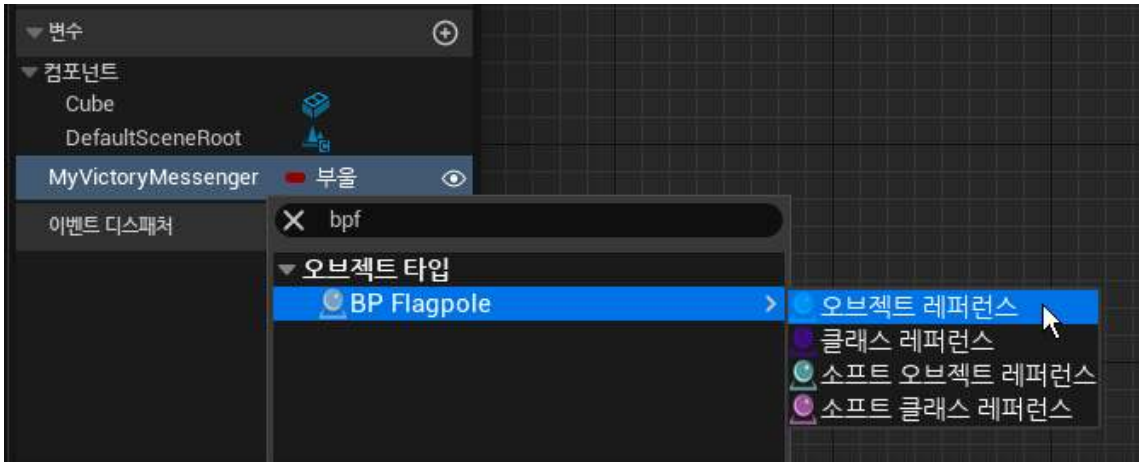
13. **OpenDoor** 이벤트 노드의 출력 실행핀을 당기고 **SetVisibility (Cube)** 노드를 검색하여 배치하자. 그리고 노드의 **NewVisibility** 입력핀을 체크하여 활성되도록 하자.



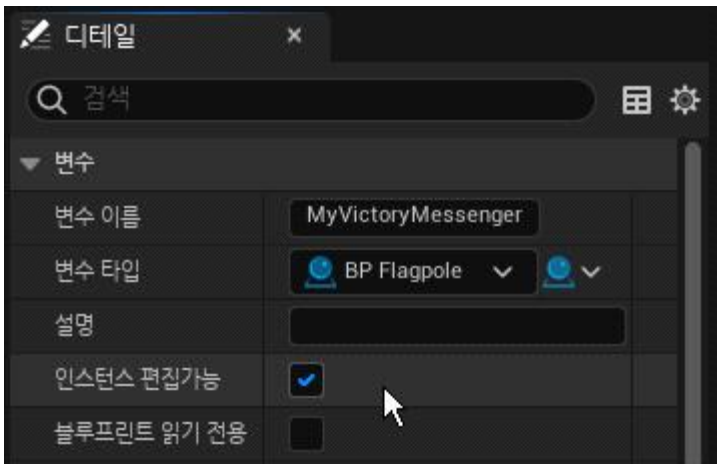
14. **BeginPlay 이벤트** 노드의 출력 실행핀을 당기고 **SetVisibility (Cube)** 노드를 검색하여 배치하자. 그리고 노드의 **NewVisibility** 입력핀을 그대로 두어 체크해제되도록 두자.



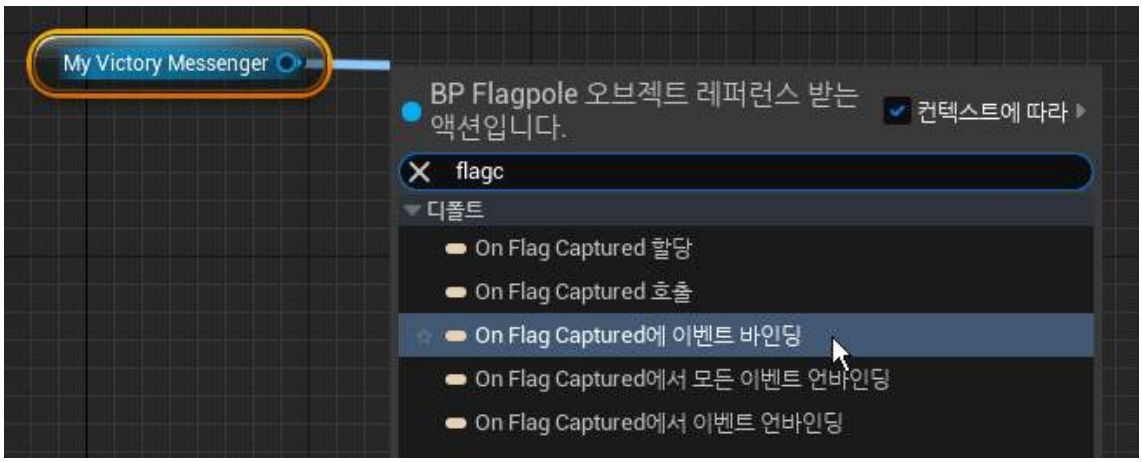
15. **내 블루프린트** 탭에서 **변수** 영역의 오른쪽 **+** 버튼을 클릭하자.
디폴트로 **NewVar_0**으로 되는 변수 이름을 **MyVictoryMessenger**로 바꾸자.
디폴트로 **부울(Boolean)**로 되어 있는 변수 타입을 클릭하고 팝업메뉴창에서 검색하여 **BP_Flagpole**의 **오브젝트 레퍼런스** 타입으로 바꾸자.



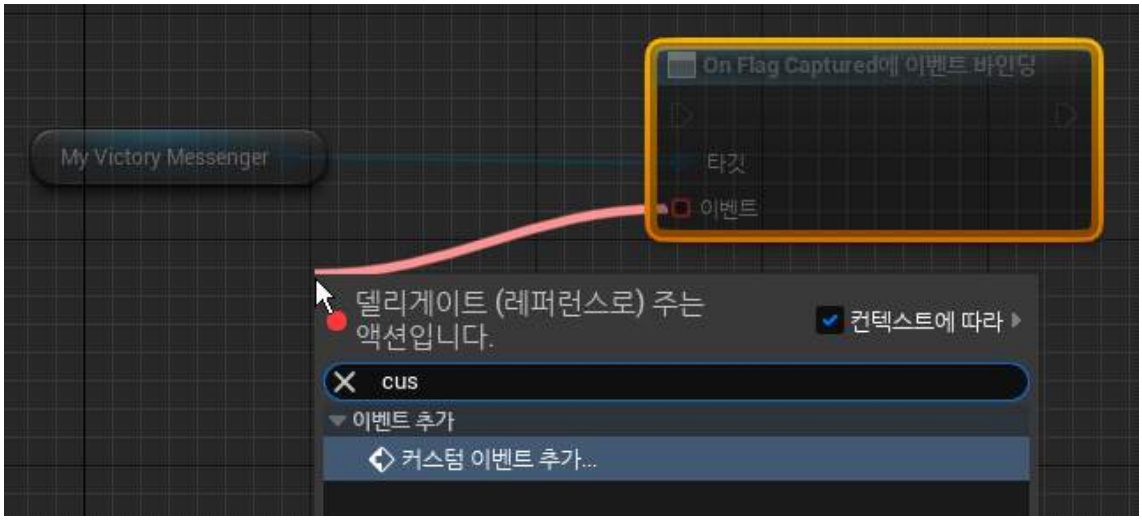
16. **디테일** 탭에서 **인스턴스 편집가능**에 체크되어 있는지 확인하자. 체크되어 있지 않다면 체크해서 활성 상태가 되도록 하자.



17. 이제, 내 블루프린트 탭에서 변수 **MyVictoryMessenger**를 드래그해서 이벤트 그래프 탭의 격자판에 드롭하자. 드롭 시의 선택창에서 **Get MyVictoryMessenger**를 선택하자.
MyVictoryMessenger를 노드의 출력핀을 드래그하고 액션선택 창에서 **OnFlagCaptured에 이벤트 바인딩**을 선택하자.



18. 이제, 배치된 이벤트 바인딩 노드의 **이벤트** 입력핀을 왼쪽으로 드래그하고 액션선택 창에서 **커스텀 이벤트 추가**를 선택하여 새 커스텀 이벤트 노드를 추가하자. 추가된 커스텀 이벤트 노드의 이름을 **Win**으로 수정하자.



19. 이제, **Win** 이벤트 노드의 출력 실행핀을 드래그하고 검색하여 **OpenDoor** 함수호출 노드를 배치하자.

그다음, **BeginPlay** 이벤트 노드 뒤의 **SetVisibility** 노드의 출력 실행핀을 이벤트 바인딩 노드의 입력 실행핀에 연결하자.

BeginPlay 이벤트 노드는 액터 인스턴스 생성 시에 바로 실행되므로 시작 시점에서 **BP_Flagpole**의 **OnFlagCaptured** 이벤트 디스패처에 **Win** 커스텀 이벤트 노드가 바인드된다.

컴파일하고 저장하자.

20. 위의 노드 네트워크를 다시한번 살펴보자.

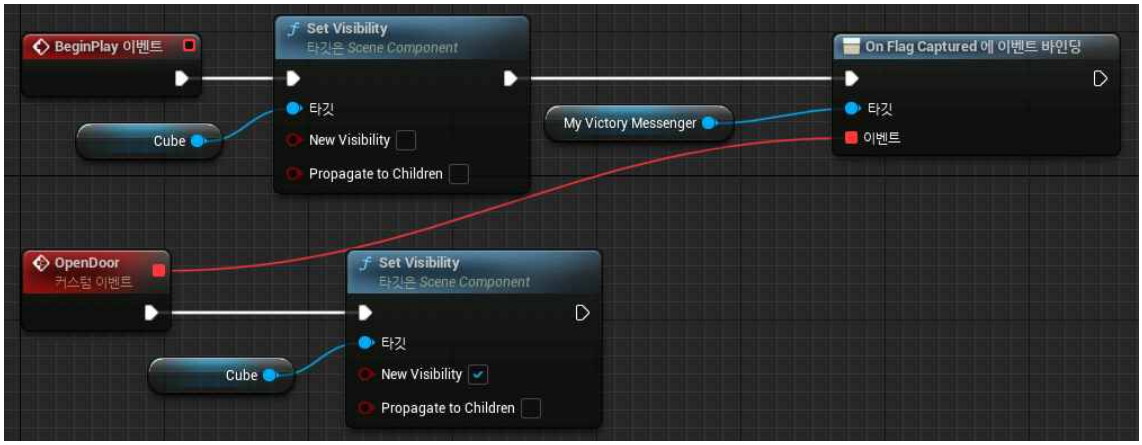
OnFlagCaptured에 이벤트 바인딩 노드의 **이벤트** 입력핀에는 **OnFlagCaptured** 이벤트 디스패처에 바인드할 이벤트 노드를 지정한다. 위의 노드 네트워크에서는 **Win** 이벤트 노드를 지정하였다. 모든 이벤트 노드에는 오른쪽 위에 붉은색 네모 모양의 출력핀에 있다. 이 핀을 **델리게이트** 핀이라고 한다. 보통의 입출력 핀은 핀 아이콘이 원형이지만 **델리게이트** 핀은 핀 아이콘이 붉은색 테두리의 네모 모양으로 다르게 표시되므로 쉽게 구분된다.

델리게이트라는 용어는 이벤트 디스패처라는 용어와 동일한 의미로 이해하면 된다. 특정 사건이 발생할 때에 그 사건의 발생에 관심있는 모든 구독자에게 알리는 기능을 수행하는 것을 델리게이트 또는 이벤트 디스패처라고 한다. 두 용어가 있어서 혼란스럽겠지만 블루프린트에서는 이벤트 디스패처라고 하고 C++에서는 델리게이트라고 하자.

델리게이트 핀의 용도는 해당 이벤트 노드를 이벤트 디스패처에 바인드하는 용도이다. 즉, 이벤트

노드의 **델리게이트** 출력핀으로 구독자의 호출 함수 정보가 전달되고 이벤트 바인딩 노드는 그 정보를 받아서 이벤트 디스패처의 구독자 목록에 보관하게 된다. 나중에 이벤트 디스패처의 호출 노드가 실행되면 이벤트 디스패처의 구독자 목록에 있는 모든 이벤트 노드를 실행하여 구독자에게 알려준다.

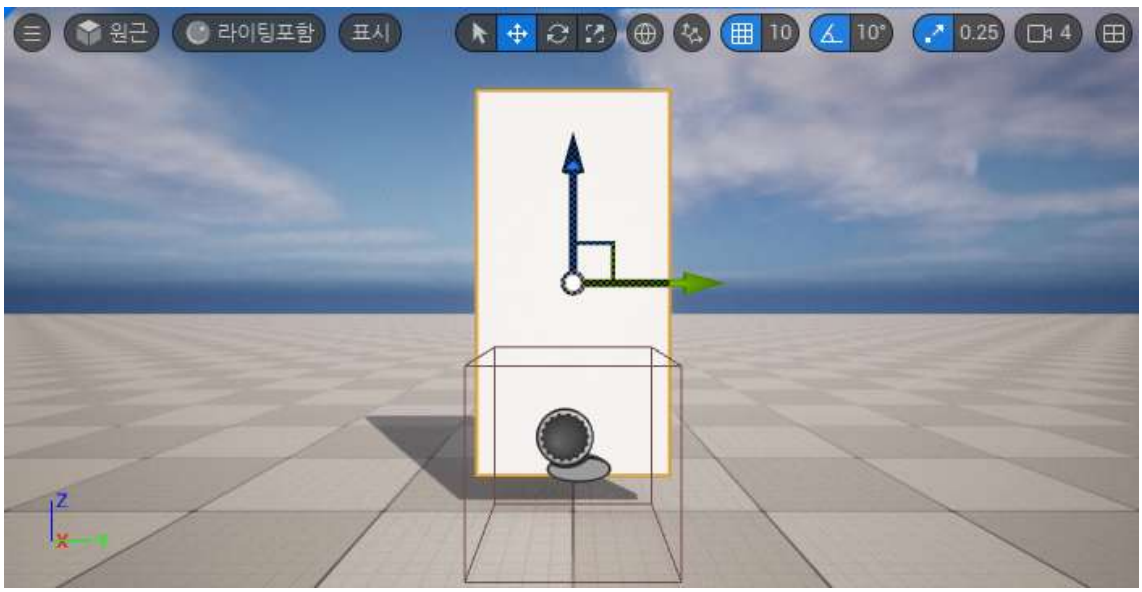
한편, 위의 노드 네트워크를 보면 **Win** 이벤트 노드는 **OpenDoor**를 호출하는 것이 전부이다. 따라서 아래 그림과 같이 **OpenDoor** 이벤트 노드를 바로 이벤트 바인딩 노드의 **이벤트** 입력핀에 연결해도 된다.



그러나, 위의 그림처럼 구현하게 되면 **델리게이트** 핀이 너무 멀리 연결되어서 노드 네트워크가 자칫 혼란스러울 수 있다. 따라서 이 경우 바인딩으로만 사용될 목적의 이벤트 노드인 **Win** 이벤트 노드를 추가하여 연결이 길게 이어지지 않도록 하였다.

21. 이제 레벨 에디터로 가자.

첫 번째 구독자 액터 **BP_DoorToNextLevel**를 레벨 에디터에 배치하자. 위치는 (300,0,120)에 배치하자.



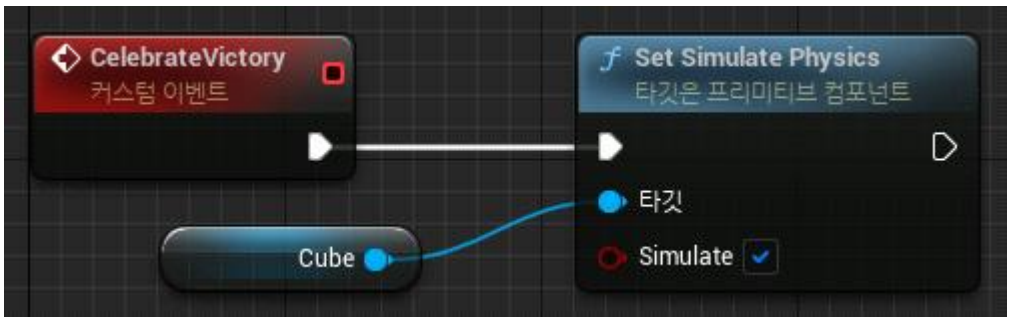
22. 그다음, **디테일** 탭의 **MyVictoryMessenger** 속성값이 디폴트로 **없음**으로 되어 있는데 이것을 **BP_Flagpole**로 바꾸자.

23. 이제, 두 번째 구독자 액터를 블루프린트 클래스로 만들어보자.
콘텐츠 브라우저에서 툴바에서 **+추가**를 클릭하고, 드롭다운 메뉴에서 **블루프린트 클래스**를 선택하고, **부모 클래스 선택** 창에서 **Actor**를 부모 클래스로 선택하자. 이름은 **BP_StageOrnament**로 수정하자.

24. **BP_StageOrnament**를 더블클릭하여 블루프린트 에디터를 열자.
컴포넌트 탭에서 **+추가**를 클릭하고 **큐브**를 선택하여 큐브 스태틱 메시 컴포넌트를 추가하자. 이름은 디폴트로 되는 **Cube**로 그대로 두자.

25. 이벤트 그래프 탭에서 격자판의 빈 곳에서 우클릭하고 액션선택 창에서 검색하여 **커스텀 이벤트**를 추가하자. 추가된 커스텀 이벤트의 이름을 **CelebrateVictory**로 수정하자.

26. **CelebrateVictory** 이벤트 노드의 출력 실행핀을 당기고 **SetSimulatePhysics (Cube)** 노드를 검색하여 배치하자. 그리고 노드의 **Simulate** 입력핀을 체크하여 활성되도록 하자.

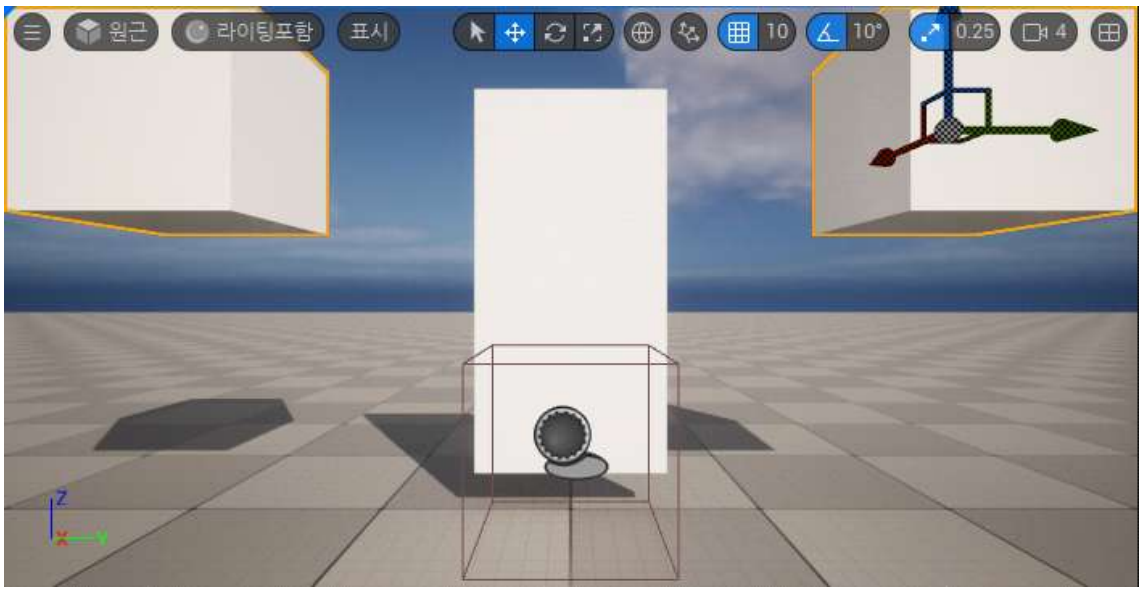


27. 첫 번째 구독자에서의 과정을 반복하자.
먼저, **내 블루프린트** 탭에서 **변수** 영역의 오른쪽 **+** 버튼을 클릭하자.
디폴트로 **NewVar_0**으로 되는 변수 이름을 **MyVictoryMessenger**로 바꾸자.
디폴트로 **Boolean**으로 되어 있는 변수 타입을 클릭하고 팝업메뉴창에서 검색하여 **BP_Flagpole**의 **오브젝트 레퍼런스** 타입으로 바꾸자.
그다음, **디테일** 탭에서 **인스턴스 편집가능**에 체크해서 활성 상태가 되도록 하자.
그다음, 내 블루프린트 탭에서 변수 **MyVictoryMessenger**를 드래그해서 이벤트 그래프 탭의 격자판에 드롭하자. 드롭 시의 선택창에서 **Get MyVictoryMessenger**를 선택하자.
MyVictoryMessenger를 노드의 출력핀을 드래그하고 액션선택 창에서 **OnFlagCaptured에 이벤트 바인딩**을 선택하자.
그다음, 배치된 이벤트 바인딩 노드의 **이벤트** 입력핀을 왼쪽으로 드래그하고 액션선택 창에서 **커스텀 이벤트 추가**를 선택하여 새 커스텀 이벤트 노드를 추가하자. 추가된 커스텀 이벤트 노드의 이름을 **Win**으로 수정하자.
그다음, **Win** 이벤트 노드의 출력 실행핀을 드래그하고 검색하여 **CelebrateVictory** 함수호출 노드를 배치하자.
그다음, **BeginPlay** 이벤트 노드의 출력 실행핀을 이벤트 바인딩 노드의 입력 실행핀에 연결하자.

컴파일하고 저장하자.

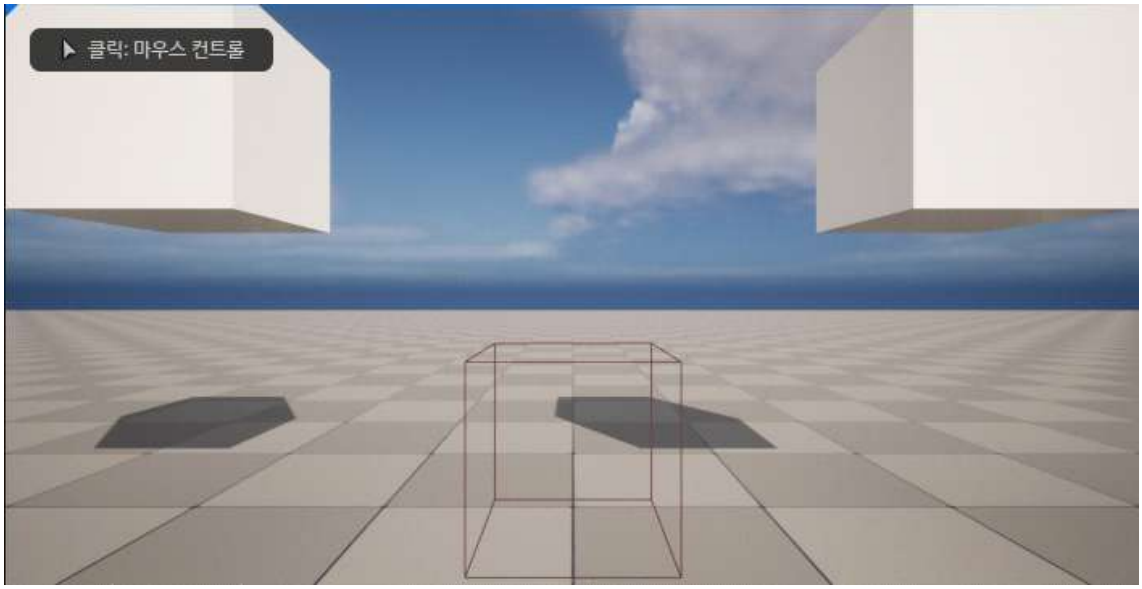
28. 레벨 에디터로 가자.

두 번째 구독자 액터 **BP_StageOrnament**를 레벨 에디터에 배치하자. 두 개를 배치하자. 첫 번째는 위치 (300,-200,200)에 배치하고, 두 번째는 위치 (300,200,200)에 배치하자.



그다음, 두 인스턴스 모두에 대해서, 디테일 탭의 **MyVictoryMessenger** 속성값을 **BP_Flagpole**로 바꾸자.

29. 플레이해보자. 조금씩 앞으로 가보자. 콜리전 볼륨과 겹침시작 이벤트가 발생하면 보이지 않았던 중간의 문이 보일 것이다. 그리고 좌우에 떠 있던 상자가 떨어질 것이다.



지금까지, 이벤트 디스패처에 대해서 학습하였다

우리는 지금까지 커스텀 이벤트 디스패처에 대해서 학습하였다. 한편, 액터에서는 이미 정의된 이벤트 디스패처들이 있다. 이들에 대해서도 유용하게 사용할 수 있어야 한다. 사용 방법은 예제에서 학습한 방법과 동일하게 사용하면 된다. 시스템 제공 이벤트 디스패처들은 다음과 같다.
게임 » **데미지** 카테고리에서 **OnTakeAnyDamage**, **OnTakePointDamage**, **OnTakeRadialDamage**가 있다.
게임 카테고리에서 **OnDestroyed**, **OnEndPlay**가 있다.
입력 » **마우스 입력** 카테고리에서 **OnBeginCursorOver**, **OnClicked**, **OnEndCursorOver**, **OnReleased**가 있다.
입력 » **터치 입력** 카테고리에서 **OnInputTouchBegin**, **OnInputTouchEnd**, **OnInputTouchEnter**, **OnInputTouchLeave**가 있다.
콜리전 카테고리에 **OnActorBeginOverlap**, **OnActorEndOverlap**, **OnActorHit**가 있다.
이들은 C++에서 델리게이트로 불리지만 블루프린트에서 이벤트 디스패처에 해당하고 사용방법도 우리가 예제에서 다루었던 방법과 동일하다.

□