08_ 블루프린트 스크립트 작성하기

<제목 차례>

인천대학교 컴퓨터공학부 박종승

1. 개요

이 장에서는 블루프린트 스크립트 작성에 대해서 학습한다.

블루프린트는 언리얼 엔진에서의 시각 스크립트 시스템(visual scripting system)이다. C++ 언어는 텍스트 기반의 스크립트로 코딩을 진행하는데 반해서, 블루프린트에서는 노드 기반의 시각 스크립트로 코딩을 진행한다. 즉, 블루프린트는 텍스트를 입력하는 방식이 아니라 그래픽적인 프로그램 요소를 조작하여 프로그래밍하는 비주얼 프로그래밍 언어이다. 대표적으로 스크래치 언어를 예를 들 수 있다.

우리는 이전에서 블루프린트 클래스에 대해서 다루어보았다. 블루프린트 클래스에 대해서 다시 더 정확하게 정리해보자. C++, Java, C# 등의 언어에서는 객체를 클래스의 형태로 구현한다. 언리얼에서도 이와 동일한 개념으로, 객체를 C++ 언어로 구현한 것을 **C++ 클래스**라고 하고 블루프린트 스크립트로 구현한 것을 **블루프린트 클래스**라고 한다. 즉, **블루프린트 클래스**(blueprint class)는 한 객체를 블루프린트 시각 스크립트 언어로 구현한 실체이다.
구현된 블루프린트 클래스는 하나의 애셋에 해당하고 독립된 애셋 파일로 저장된다. 따라서 블루프린트 클래스는 여러 레벨에서 재사용될 수 있다. 또한 블루프린트 클래스는 그 특성이 C++ 클래스와 거의 동일하다. 예를 들어, 한 블루프린트 클래스 또는 C++ 클래스를 상속하여 자식 블루프린트 클래스를 만들 수도 있고, 멤버 변수나 멤버 함수 등을 내부에 정의할 수도 있다.

우리는 이전에서 블루프린트 스크립트로 구현한 실체를 **블루프린트 클래스**라고 하였다. 한편, 블루프린트 클래스 이외에 블루프린트 스크립트로 구현한 실체가 하나 더 있다. 바로 레벨 블루프린트이다. 이제부터 레벨 블루프린트에 대해서 알아보자. **레벨 블루프린트**(level blueprint)는 클래스가 아닌 특별한 형태의 블루프린트이다. 레벨 블루프린트는 한 레벨 내에서 그 레벨의 모든 것을 바로 접근할 수 있는 이벤트 그래프이다.
각 레벨마다 하나의 유일한 레벨 블루프린트가 있다. 따로 생성하지 않아도 레벨이 생성되면 레벨 내에 포함되도록 함께 생성된다. 레벨 블루프린트는 별도의 애셋으로 존재하지 않고 레벨 애셋의 내부에 포함된 요소로 간주된다.
레벨 블루프린트는 레벨에의 배치된 액터들을 직접 접근하여 레벨 고유의 시나리오를 표현할 수 있다. 따라서 특정 레벨에 종속되어 있고 다른 레벨에서는 사용할 수 없다.
보다 쉽게 이해하기 위해서, 레벨 블루프린트는 특정 레벨 내에서만 실행되도록 작성해둔 블루프린트 스크립트 묶음으로 생각하면 된다. 또한, 레벨을 특별하고 유일한 하나의 클래스라고 한다면 그 레벨 클래스에 대한 블루프린트 클래스로 생각해도 된다.

우리는 앞으로 블루프린트 에디터에서의 블루프린트 코딩에 대해서 조금씩 배워갈 것이다.

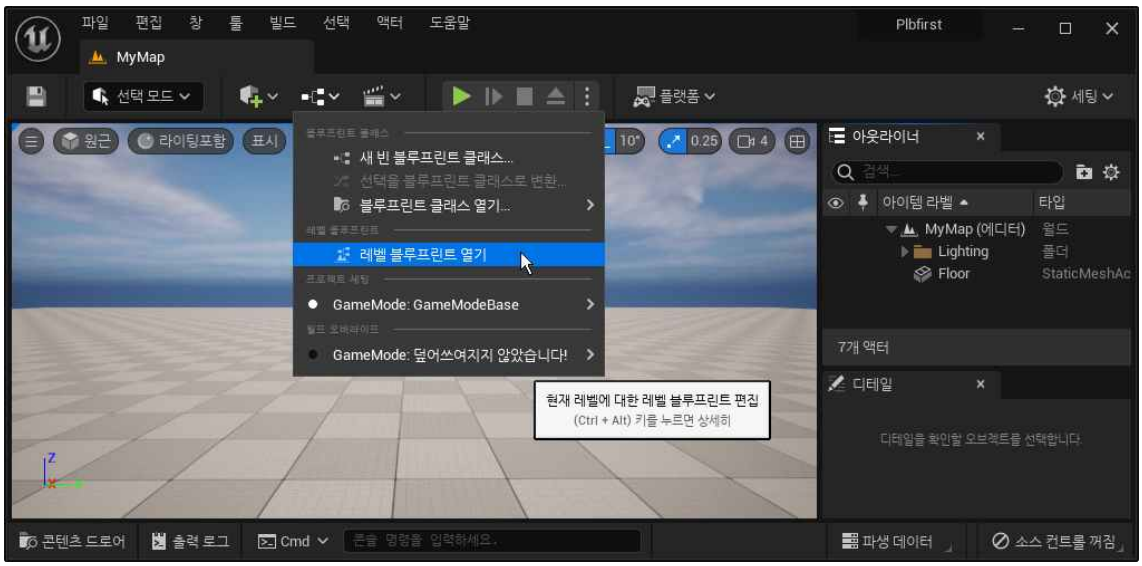
2. 레벨 블루프린트 스크립트 처음으로 작성해보기

레벨 블루프린트 스크립트를 작성하는 방법을 학습해보자.
블루프린트 클래스 에디터와 레벨 블루프린트 에디터의 모습도 약간 다르다. 레벨 블루프린트 에디터에는 컴포넌트 탭이나 뷰포트 탭은 없다. 레벨 블루프린트에는 격자판 하단에 **블루프린트** 대신 **레벨 블루프린트**라고 크게 적혀있다.
이 절에서는 레벨 블루프린트 스크립트를 처음으로 작성해본다.

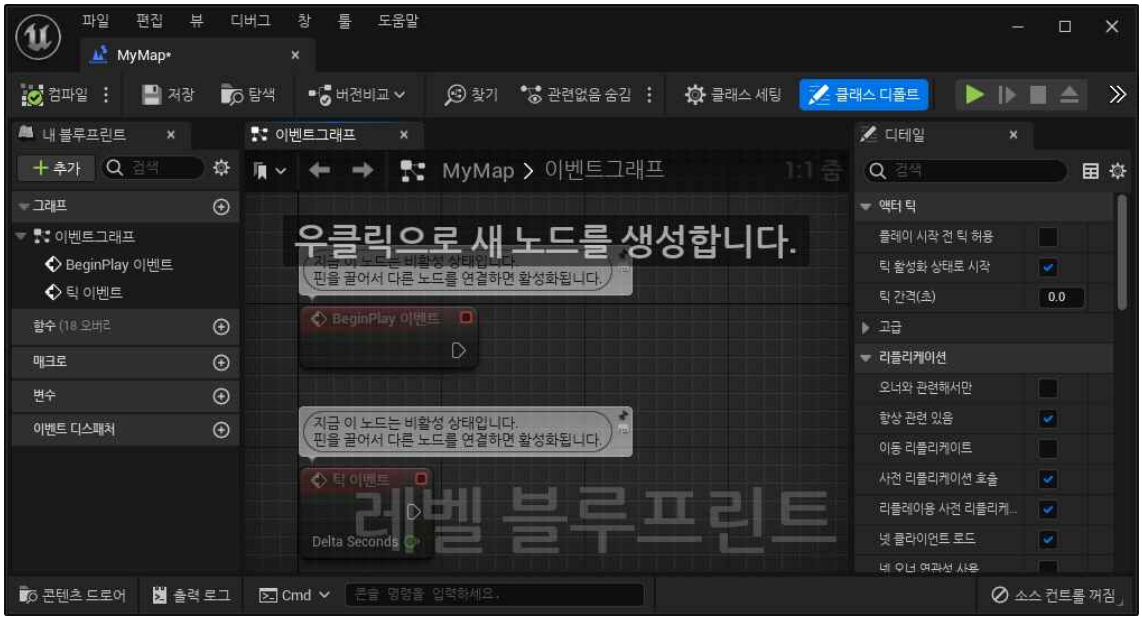
---

1. 새 프로젝트 **Plbfirst**를 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 Plbfirst로 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. 이제부터, 레벨 블루프린트 스크립트 작업을 시작해보자. **레벨 에디터**의 툴바에서 **블루프린트** 버튼을 클릭하자. 그리고, 드롭다운 메뉴에서 **레벨 블루프린트 열기**를 선택하자.



3. 레벨 블루프린트 에디터가 다음과 같은 모습으로 나타난다.



먼저, 레벨 블루프린트 에디터의 모습을 살펴보자.

대부분의 일반적인 에디터에서와 유사하게 메뉴바가 있고 툴바가 있다.

툴바 아래에는 중간에는 **이벤트 그래프** 탭이 있다. 그 왼쪽에는 **내 블루프린트** 탭이 있고 오른쪽에는 **디테일** 탭이 있다.

이벤트 그래프 탭은 노드 네트워크를 편집하는 곳으로 실제 블루프린트 스크립트를 작성하는 곳이다.

왼쪽의 **내 블루프린트** 탭은 블루프린트에 포함된 그래프, 함수, 변수 등에 대한 리스트를 표시한다. 블루프린트를 한번에 파악할 수 있고 기존 요소를 확인하거나 새 요소를 추가할 수 있다.

오른쪽의 **디테일** 탭은 블루프린트 함수나 변수 등 선택된 항목의 속성을 편집할 수 있도록 한다. 블루프린트 내의 특정 요소를 선택하면 디테일 탭에는 그 요소의 속성이 표시되는 반면, 아무 요소도 선택하지 않은 초기 상태이거나 또는 툴바에서 **클래스 디폴트** 버튼을 선택하면 디테일 탭에는 블루프린트 전체의 속성에 대한 디폴트 값이 표시된다.

<참고> 레벨 블루프린트 에디터의 UI에 대해서는 다음의 문서를 참조하자.
https://docs.unrealengine.com/blueprints-visual-scripting-editor-user-interface-for-level-blueprints-in-unreal-engine/

4. 레벨 블루프린트 에디터의 모습을 계속 살펴보자. 에디터의 왼쪽 아래에 **내 블루프린트** 탭이 있다.

이 탭은 블루프린트로 코딩을 하는 매우 중요한 탭이다. 여기에는 **그래프**, **함수**, **매크로**, **변수**, **이벤트 디스패처**의 5개 영역이 있다. 앞으로 각각에 대해서 학습할 것이다.
우리는 우선 가장 위의 **그래프** 영역에 대해서 살펴보자. 여기에서 **그래프**는 단지 여러 **이벤트 그래프**를 묶은 그룹명이며 더 이상의 의미는 없다. **그래프**라는 이름이 다소 혼란스러울 수 있는데 우리는 여기에서의 **그래프**라는 용어가 **이벤트 그래프 그룹**이라는 이름의 줄임말이라고 생각하자.

5. 이제, 블루프린트에서 **그래프**의 개념에 대해서 알아보자.
블루프린트에서는 노드와 노드를 서로 연결하는 네트워크를 작성하는 방법으로 코딩을 수행한다. 작성된 결과를 **노드 네트워크**라고 한다. 이 노드 네트워크를 언리얼에서는 **그래프**라고도 한다. 즉 그래프는 노드 네트워크를 의미하는 명칭이다.
하나의 노드 네트워크 또는 그래프는 하나의 함수 또는 프로시져 또는 매크로 등에 해당한다. 그래프의 가장 왼쪽은 작업이 시작되는 부분이고 가장 오른쪽은 작업이 종료되는 부분이다. 함수를 예를 들어 설명하면 그래프 자체가 함수의 구현에 해당하고 네트워크 흐름의 왼쪽 시작이 함수의 시작 부분이고 오른쪽 끝이 함수의 종료 부분에 해당한다.
그래프의 세부적인 형태에는 **이벤트 그래프**(event graph), **컨스트럭션 스크립트**(construction script), **함수**(function), **매크로**(macro)의 네 종류가 있다. **이벤트 그래프**는 블루프린트에서의 가장 중심이 되는 그래프이다. **컨스트럭션 스크립트**는 블루프린트 클래스마다 하나씩만 있는 것으로 초기화를 위해 사용되는 그래프이다. **함수**와 **매크로**는 다른 언어에서의 함수와 매크로의 개념과 유사하다.
지금까지 그래프의 개념에 대해서 알아보았다.
이제, **이벤트 그래프**의 개념에 대해서 알아보자.
한 블루프린트에는 다수개의 노드 네트워크가 포함되어 있다. 한 노드 네트워크 내의 모든 노드는 링크로 서로 연결되어 있지만, 각 노드 네트워크 사이에는 명시적인 링크 연결이 없이 서로 분리되어 있다. 각 노드 네트워크는 독립적인 단일 작업을 수행하는 태스크 단위에 해당한다. 게임 플레이에서 특정 이벤트가 발생하면 특정 노드 네트워크의 시작 부분이 발동된다. 각 노드 네트워크의 시작 지점을 **이벤트** 노드라고 하고 붉은색 노드로 표시된다. 이벤트 노드로 시작하는 노드 네트워크를 **이벤트 그래프**라고 한다. 즉, **이벤트 그래프**는 이벤트가 발생할 때에 호출되는 콜백 함수라고 생각할 수 있다.

한편, 자주 사용되는 노드 네트워크의 일부를 함수나 매크로의 형태로 별도의 분리해서 독립적인 노드 네트워크로 구현할 수도 있다. 이러한 노드 네트워크는 기존의 **이벤트 그래프**와는 약간 다른 특성을 가진다. 예를 들어 함수에 해당하는 노드 네트워크는 리턴을 할 수 있어야 한다. 따라서 모든 노드 네트워크를 이벤트 그래프라고 하지 않고 이벤트 그래프가 아닌 함수나 매크로 등의 **이벤트 그래프**와는 다른 형태의 노드 네트워크가 있는 것이다.
지금까지 **이벤트 그래프**에 대해서 알아보았다.
모든 종류의 노드 네트워크는 그 형태나 사용 방법이 거의 동일하다. 노드 네트워크의 시작 노드의 색상으로 쉽게 구분할 수 있다. **이벤트 그래프**의 시작 노드는 빨간색이다. **컨스트럭션 스크립트**와 **함수**는 보라색이다. **매크로**는 남색이다. 이들에 대해서는 추후에 하나씩 학습할 것이다.
한편, 그래프 중에서 **이벤트 그래프**와 **함수**를 구분하는 것이 어렵게 느껴질 수 있다. 대부분의 사용 방법이 동일하게 때문이다. 이들의 차이점에 대해서는 이후에 **함수**를 학습할 때에 자세하게 알아보자. 그 전까지는 **이벤트 그래프**와 **함수**는 거의 동일한 그래프로 생각하자.

<참고> 노드 그래프의 편집 방법에 대해서는 다음의 문서를 참조하자.
https://docs.unrealengine.com/specialized-blueprint-visual-scripting-node-groups-in-unreal-engine/

6. 이제 다시, **내 블루프린트** 탭을 살펴보자. **그래프** 아래에 **이벤트 그래프**가 있다. 이 **이벤트 그래프**는 이벤트 그래프 노드 네트워크를 구현하는 코딩 격자판에 해당한다. 코딩이 점점 늘어나서 하나의 코딩 격자판에 담기가 복잡해지면 **그래프** 영역에서 새 **이벤트 그래프**를 추가하면 새로운 코딩 격자판이 추가되는 것이다. 여러 **이벤트 그래프**가 있다고 해서 서로 분리된 것은 아니며 단지 시각적인 편의를 위해서 제공되는 기능이다.

7. 디폴트로 추가되어 있는 **이벤트 그래프**를 클릭해보자.
BeginPlay 이벤트와 **Tick 이벤트**의 두 이벤트 노드가 나열되어 있다. 더블클릭하면 이벤트 그래프 격자판에서 해당 이벤트 노드 위치로 이동한다. 이 두 노드는 매우 빈번하게 사용되는 이벤트 노드로 편의를 위해서 엔진이 미리 추가해준 것이다. 블루프린트 코딩은 한 이벤트 노드를 배치하고 그 이벤트 노드를 시작으로 노드 네트워크를 만들어가면 된다.

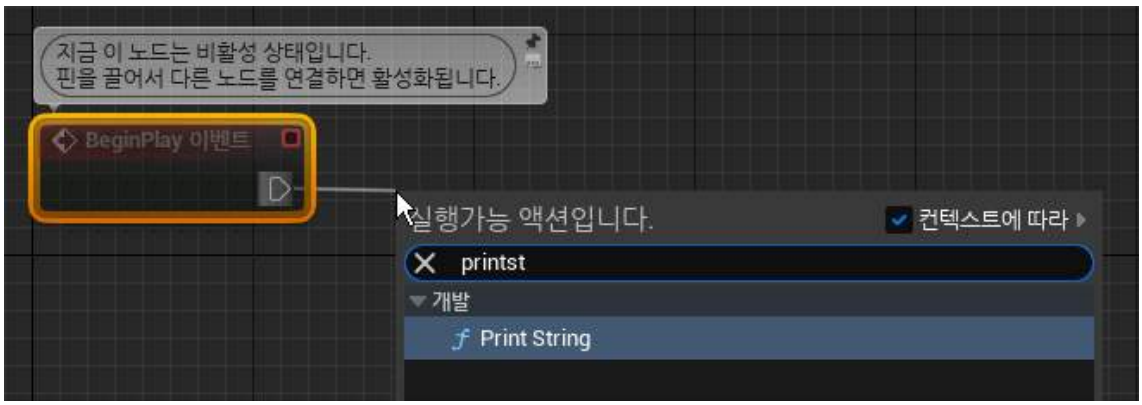
위의 그림에서는 이미 배치되어 있는 **BeingPlay 이벤트** 노드를 보여준다. 노드의 오른쪽의 흰색 출력 핀을 오른쪽으로 당겨서 새로운 노드를 배치하는 식으로 노드 네트워크를 작성해나가면 된다. 해당

이벤트가 발생하면 흰색 핀이 연결되어 있는 노드가 순차적으로 하나씩 실행된다.

8. 이제 **이벤트 그래프** 탭의 격자판에서의 조작에 대해서 알아보자.
노드 네트워크를 작성하기 위한 마우스 조작은 직관적이고 쉽다.
격자판 위에서 **우클릭+드래그**하면 격자판의 현재 위치를 이동할 수 있다.
그리고, **휠버튼+스크롤** 조작으로 격자판을 12단계로 축소할 수도 있다. 격자판은 원본 크기인 **1:1 줌**이고 작게 축소하는 단계는 **-1 줌**부터 **-12 줌**까지 가능하다.
그리고, 빈 공간에서 **우클릭**하면 배치 가능한 노드를 나열해주고 배치할 노드를 선택할 수 있도록 하는 **액션선택** 창이 뜬다. **액션선택** 창에서 특정 노드 하나를 선택하면 선택된 노드가 배치된다.
그리고, 노드의 한 핀과 다른 노드의 핀을 잇기 위해서는 핀에서 핀으로 **좌클릭+드래그**하면 된다.
그리고, 노드의 한 핀에서 **좌클릭+드래그**를 시작해서 빈 곳에서 버튼을 떼면 **액션선택** 창이 뜬다. 노드를 선택하면 선택된 노드가 배치되고 핀도 함께 연결된다.
그리고, 노드를 이동하기 위해서는 노드 위에서 **좌클릭+드래그**하여 이동하면 된다.
그리고, 두 노드의 핀 연결을 끊기 위해서는 한쪽 핀 위에서 **Alt+좌클릭**하면 연결이 끊어진다.
그리고, 노드를 삭제하기 위해서는 노드를 클릭하여 선택한 후에 **Delete** 키를 누르면 된다.
그리고, 여러 노드를 선택하고 이를 한꺼번에 삭제하거나 이동할 수도 있다.
여러 노드를 선택하는 방법으로, 마우스로 사각 영역을 드래그하여 범위 내의 여러 노드를 선택할 수 있다. 또한, **Shift+좌클릭** 또는 **Ctrl+좌클릭**으로 기존 선택에 노드를 추가할 수도 있다.
직접 조작해보면서 익숙해지도록 익혀두자.

9. **BeginPlay 이벤트** 노드를 살펴보자.
BeginPlay 이벤트 노드는 이 레벨이 실행되면 처음 한번 호출되는 노드이다. **BeginPlay 이벤트** 노드가 호출되면 실행 신호가 이벤트 노드의 출력 실행핀으로 전달된다. 실행핀은 기울어진 흰색 오각형 모양으로 표시된다. 위의 노드 그림에서 오른쪽의 흰색 핀이 실행핀이다. 그래프의 노드가 서로 실행핀으로 연결되어 있으면 각 노드가 순차적으로 실행된다. 노드의 왼쪽의 핀은 입력 핀이고 오른쪽의 핀은 출력 핀이다. 단, 이벤트 노드의 경우에는 그래프의 시작 지점이므로 입력 핀이 없다. 흰색 핀은 실행핀에 해당하고 흰색 실행핀 이외의 핀은 입력 인자값이나 출력 리턴값에 해당한다. 핀의 컬러는 타입에 따라서 결정된다.

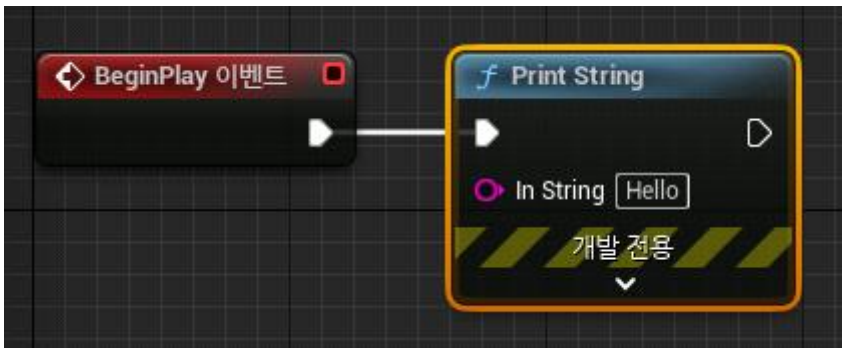
10. 이제 최초의 블루프린트 스크립트를 작성해보자.
BeginPlay 이벤트 노드의 흰색 출력 실행핀 위로 마우스를 옮기면 핀의 배경이 흰색으로 바뀔 것이다. 이때부터 **좌클릭+드래그**를 시작해서 약간 오른쪽의 빈 곳으로 이동한 후에 버튼을 떼자.



위의 그림과 같이 액션선택 창이 뜰 것이다. **PrintString** 노드를 검색해서 배치하자.

<참고> 액션선택 창의 오른쪽에 '컨텍스트에 따라'라는 체크박스가 있다. 체크되어 있는 경우에는 현재 맥락을 자동으로 파악하고 해당 액터에서 사용할 수 있는 액션만 필터링하여 리스트로 보여준다. 이 체크박스는 특별한 경우가 아니면 체크된 채로 사용하는 것이 좋다.

11. 노드가 배치될 것이다. 배치된 노드를 **좌클릭+드래그**하면 연결을 그대로 유지하면서 노드의 위치를 이동할 수 있다. 각 노드를 더 보기좋은 위치로 옮기면서 노드 네트워크를 수시로 정리하도록 하자.



이벤트 노드의 출력 실행핀이 **PrintString** 노드의 입력 실행핀에 연결되어 있다. 따라서, 이벤트 실행 신호가 발생하면 실행 신호가 이벤트 노드에서 **PrintString** 노드로 전달되어 **PrintString** 노드가 실행될 것이다. 이런 방식으로 실행할 노드를 계속해서 실행핀으로 연결시키면 연결된 노드들이 순차적으로 실행된다.

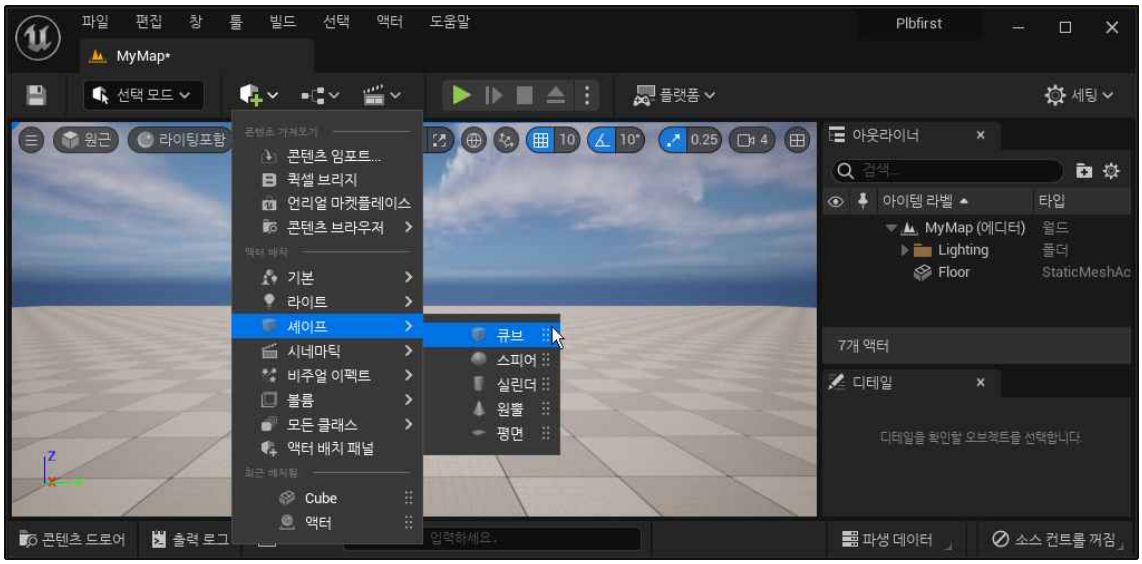
12. **PrintString** 노드는 디버그 용도의 문자열을 출력하는 노드이다.
이 노드의 입력 인자를 살펴보자.
왼쪽의 자홍색 핀은 함수의 입력 인자이다. **InString**은 입력 인자명이다. **PrintString** 함수 노드는 **InString** 인자값을 화면에 출력하는 기능을 수행한다. 입력핀이 자홍색(magenta)이면 **String** 타입을 의미한다.
현재 인자명 **InString** 오른쪽에 흰색 상자가 있다. 이 상자는 입력될 상수값을 직접 명시하는 입력상자이다. 현재 `Hello`가 입력되어 있으므로 `Hello`라는 상수값이 입력 인자 **InString**으로 전달된다. 이 흰색 입력상자를 클릭하여 상수값을 직접 입력할 수 있다.
한편, 다른 노드의 출력핀을 **InString** 핀에 연결할 수도 있다. 이 경우는 다른 노드의 출력핀이 함수의 리턴값이나 변수값일 수 있는데 이를 인자값으로 넣어주는 것을 의미한다. 핀이 연결되면 흰식 상수값 상자는 사라지고 핀의 빈 원모양이 채워진 원모양으로 바뀐다.

13. 이제 블루프린트가 실행되도록 툴바의 플레이 버튼을 클릭하여 플레이해보자.
툴바의 플레이 버튼이나 레벨 에디터의 플레이 버튼은 동일한 버튼이다. 어느 것을 사용해도 좋다.
툴바의 플레이 버튼을 클릭하여 플레이하면 레벨 블루프린트가 실행되면서 화면 상단에 Hello라는 문자열이 출력되는 것을 확인할 수 있다.

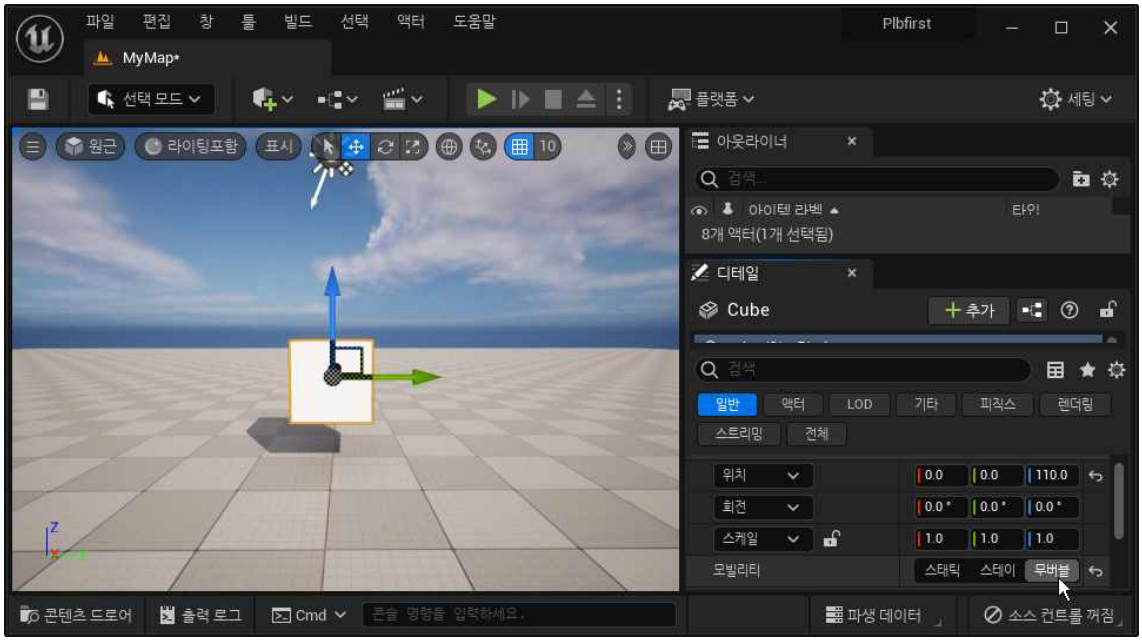
한편, 블루프린트를 실행하기에 앞서 수정된 블루프린트가 컴파일되어야 한다. 컴파일은 툴바의

가장 앞에 컴파일 버튼을 누르면 된다. 그러나 **플레이** 버튼을 누르면 수정된 내용을 컴파일한 후에 실행해주기 때문에 컴파일 버튼을 누르지 않아도 된다.

14. 조금 더 복잡한 예를 작성해보자.
우선 액터를 하나 배치하자.
우리는 지금까지 액터를 배치할 때에 메뉴바에서 **창** » **액터 배치**를 실행하고 **액터 배치** 탭이 보이도록 하였다. 그다음에, **액터 배치** 탭에서 **셰이프** 탭을 선택하고 목록에서 **큐브**를 드래그하여 레벨에 배치하였다.
한편, **액터 배치** 탭을 사용하지 않고 툴바의 아이콘을 사용하여 더 간편하게 액터를 배치할 수도 있다. 이번에는 툴바의 아이콘을 사용하여 배치해보자. 레벨 에디터의 툴바에 **액터 배치** 아이콘이 있다. 아이콘을 클릭하면 드롭다운 메뉴가 나타나는데 **셰이프** » **큐브**를 클릭하면 레벨의 중앙에 배치된다. 클릭하는 대신에 드래그하여 레벨의 원하는 장소에 배치할 수도 있다.



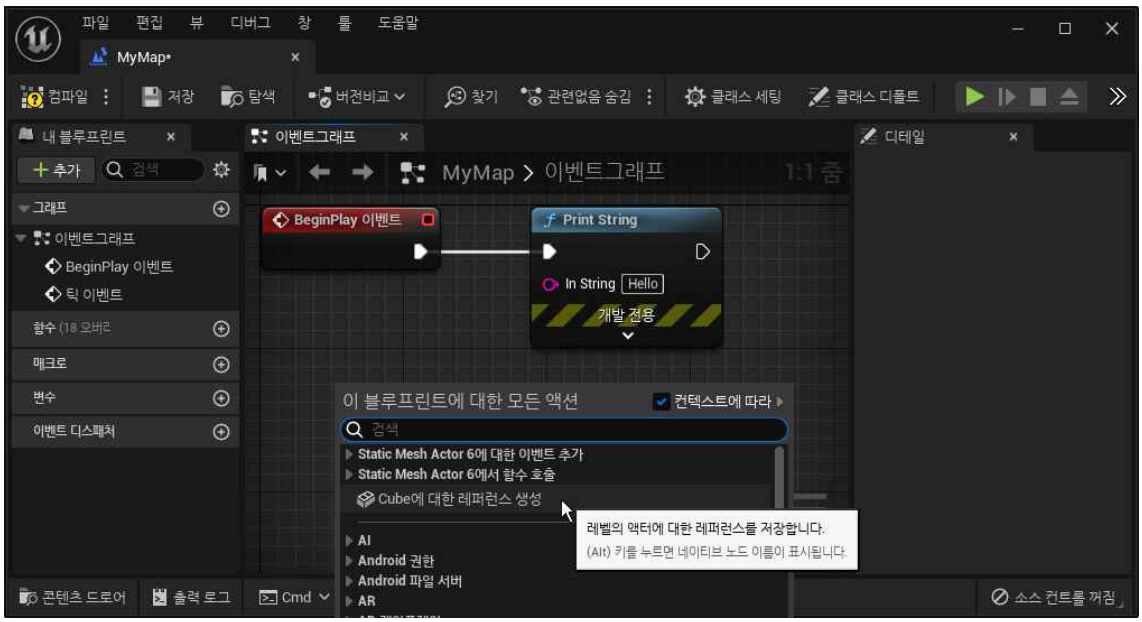
15. 큐브를 클릭하여 배치한 후에 배치된 큐브의 위치는 약간 공중에 있도록 큐브의 위치를 (0,0,110)로 수정하자.
또한 **디테일** 탭에서 **트랜스폼** 속성 아래에 있는 **모빌리티** 속성을 찾아보자. 이 속성은 배치된 액터가 게임 내부에서 움직이지 않는 것인지 또는 움직이는 것인지를 결정한다. 디폴트로 **스태틱**으로 되어 있는 것을 **무버블**로 변경하자. 이렇게 해야 배치된 큐브가 움직일 수 있게 된다.



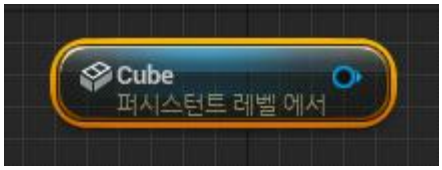
<참고> 액터의 모빌리티는 가급적이면 스태틱으로 지정하는 것이 좋다. 그 이유는 라이팅 및 쉐도잉 등에 있어서의 계산의 효율성 때문이다. 게임에서 실시간으로 빛과 그림자를 계산하는 것인 매우 부하가 높은 일이다. 광원과 물체가 움직이지 않는다면 고정된 빛과 그림자를 미리 계산해두고 실시간 계산을 하지 않아도 되므로 부하를 크게 줄일 수 있다.

16. 이제, 레벨 에디터에서 **Cube**가 선택된 상태로 유지하고, 다시 **레벨 블루프린트 에디터**로 가자. 에디터 창을 다시 열려면 툴바의 **블루프린트** 버튼을 클릭하고 드롭다운 메뉴에서 **레벨 블루프린트 열기**를 선택하면 된다.

중간의 이벤트 그래프 탭의 격자판의 빈 곳에서 우클릭하자. 그리고, 나타나는 액션선택 창에서 **Cube에 대한 레퍼런스 생성**을 클릭하자. 레벨에 배치된 **Cube** 액터에 대한 레퍼런스 노드가 생성된다.

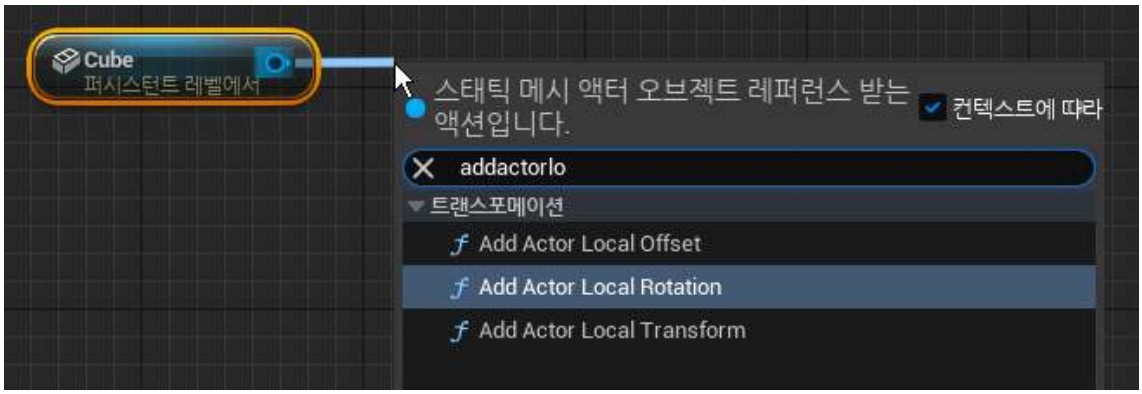


17. 배치된 **Cube** 액터에 대한 레퍼런스 노드는 다음과 같은 모습으로 보인다.

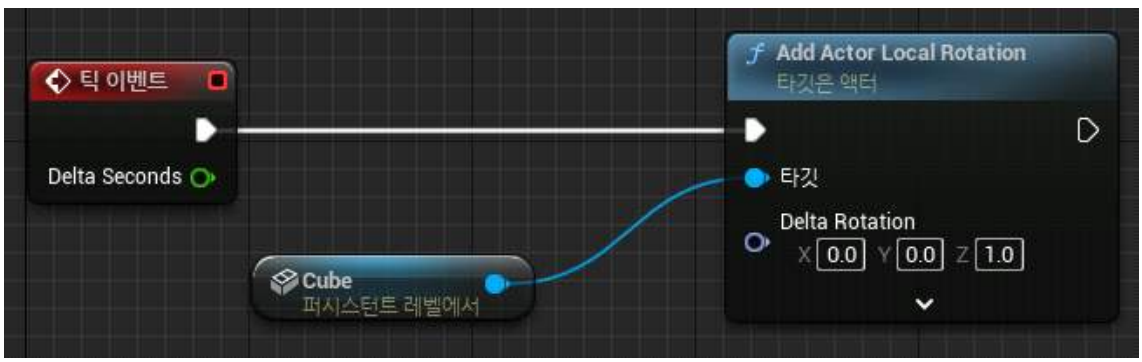


레퍼런스 노드를 생성하는 또다른 방법이 있다. **레벨 에디터**의 **아웃라이너**에서 **Cube**를 드래그하기 시작하여 레벨 블루프린트 에디터의 격자판에 드롭해도 된다. 편리한 방법을 사용하면 된다.

18. 이제, **Cube** 레퍼런스 노드의 출력핀을 드래그해서 **AddActorLocalRotation** 노드를 검색해서 배치하자.

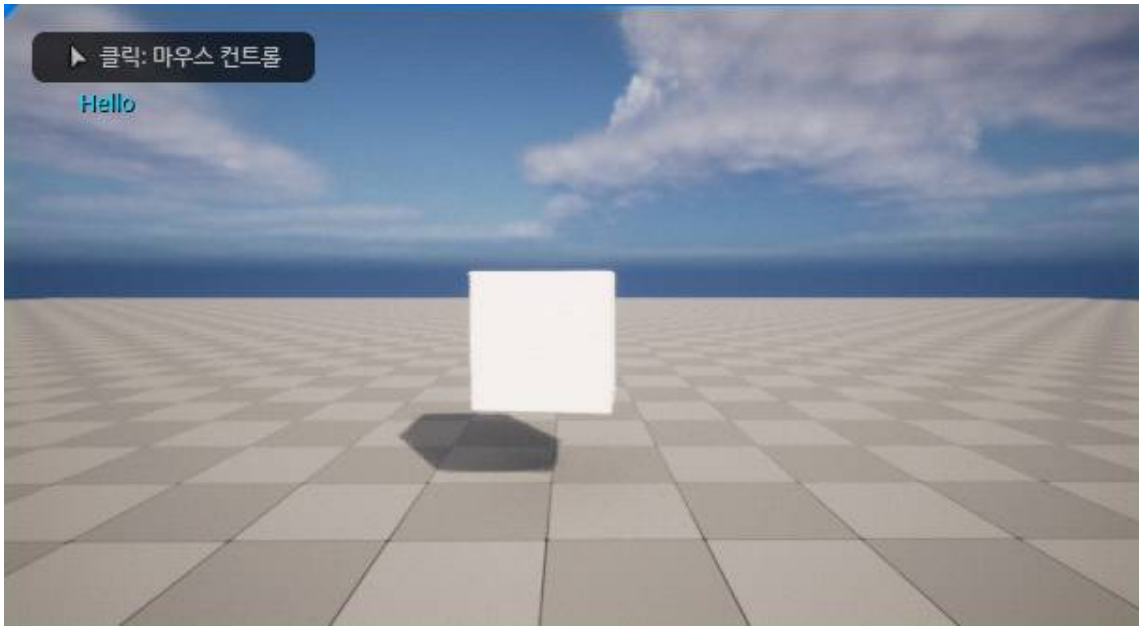


19. 배치된 **AddActorLocalRotation** 노드의 입력핀 **DeltaRotation**의 **Z**값에 1을 입력하자.
그다음, **Tick 이벤트** 노드의 출력 실행핀을 드래그하여 **AddActorLocalRotation** 노드의 입력 실행핀에 드롭하자. 두 노드의 실행핀이 연결될 것이다. 노드가 서로 멀리 있다면 노드를 드래그하여 서로 가깝게 되도록 이동하자. 범위 선택으로 여러 노드를 쉽게 선택하여 이동할 수도 있다. 완성된 모습은 아래와 같다.



Tick 이벤트 노드는 매 프레임마다 틱 신호가 발생하며 이때마다 호출되는 이벤트 노드이다. 매 프레임마다 호출되므로 게임플레이 화면을 갱신하는 용도로 사용되는 이벤트 노드이다.
AddActorLocalRotation 노드는 로컬좌표계에서 입력값만큼 회전값을 증가시킨다. Z값에 1을 입력하였으므로 매 틱 신호마다 1도씩 증가하게 된다.

20. 플레이해보자. 상자가 스핀할 것이다.



지금까지 처음으로 레벨 블루프린트에서 스크립트를 작성하는 방법을 학습하였다.

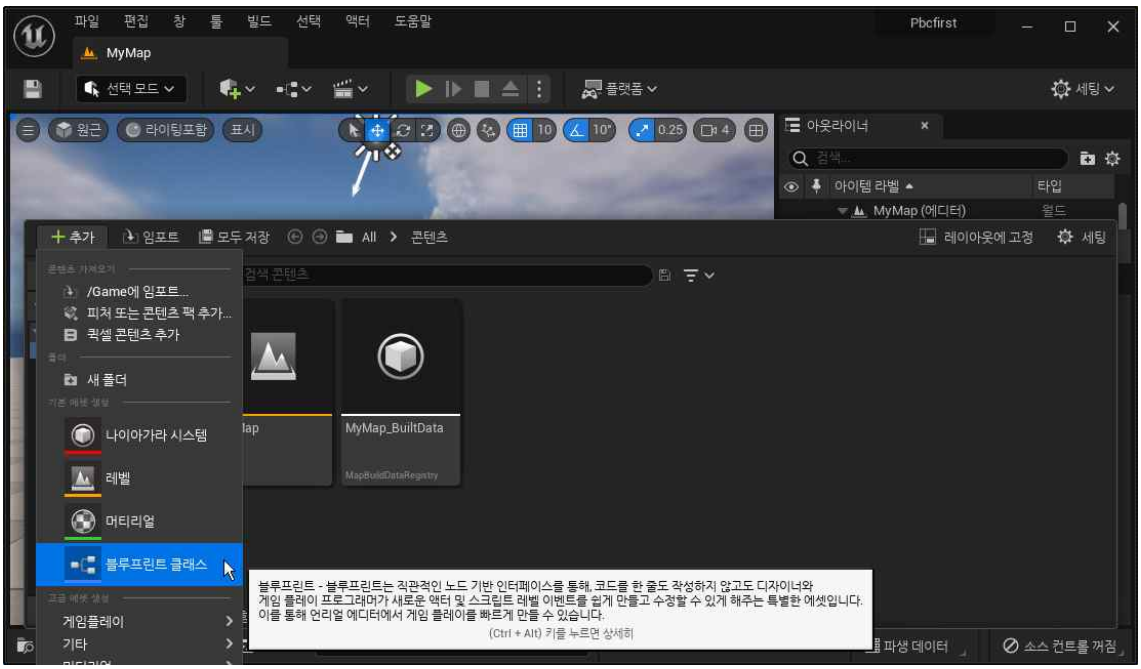
3. 블루프린트 클래스 스크립트 처음으로 작성해보기

이 절에서는 블루프린트 클래스 스크립트를 처음으로 작성해본다.

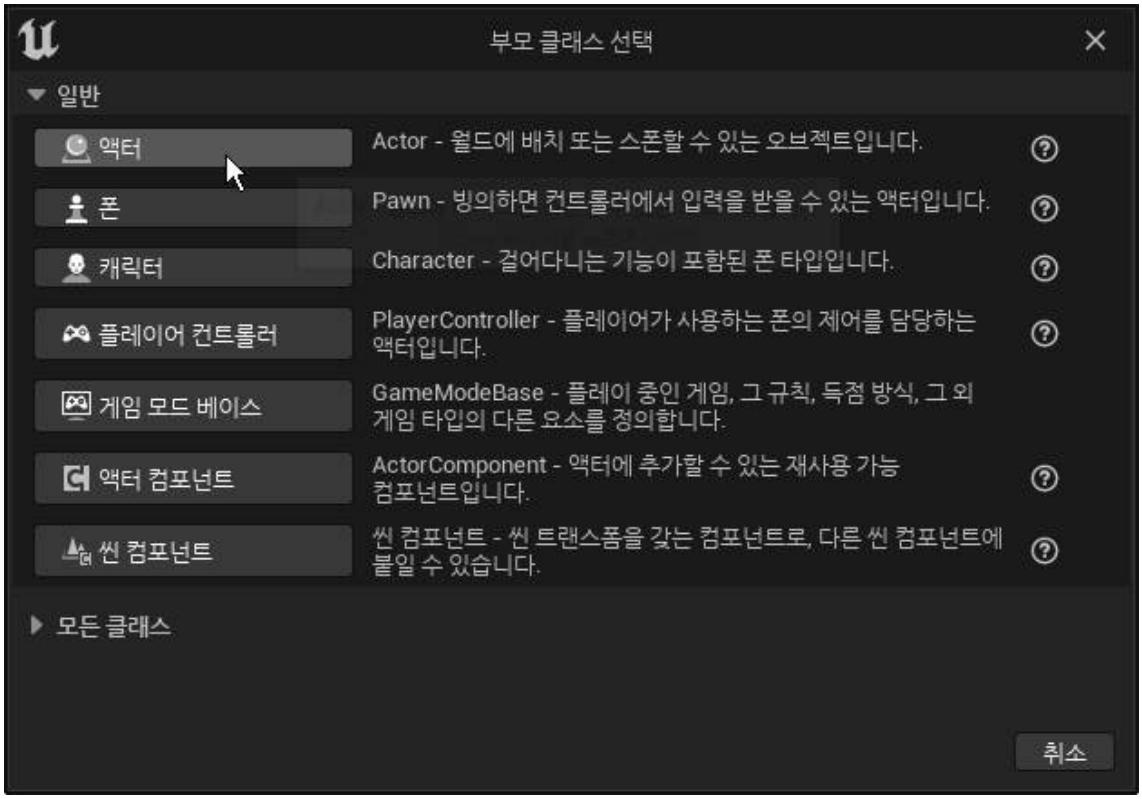
---

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

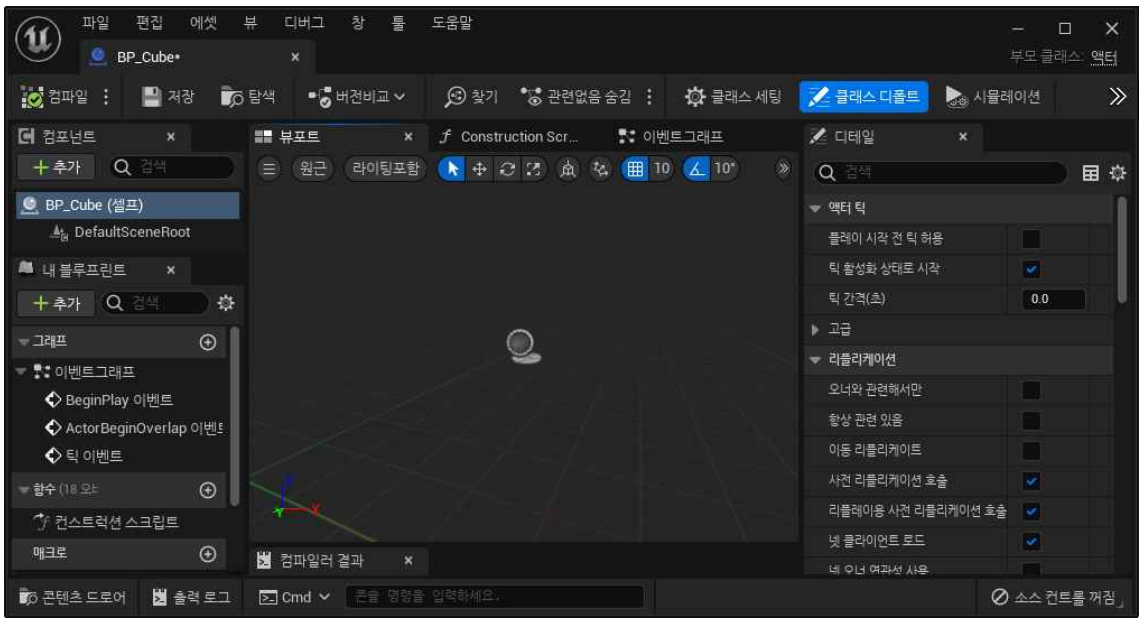
2. 이제부터, 블루프린트 클래스를 만들어보자. 레벨 에디터의 하단바의 가장 왼쪽에 있는 **콘텐츠 드로어**를 클릭하여 **콘텐츠 브라우저**가 표시되도록 하자.
왼쪽 폴더 구조에서 디폴트로 **콘텐츠** 폴더가 선택되어 있을 것이다. 애셋을 생성하면 선택된 폴더 위치에 애셋이 저장될 것이다.
콘텐츠 브라우저의 왼쪽 위의 **+추가** 버튼을 클릭하고 **블루프린트 클래스**를 선택하자.



3. **부모 클래스 선택** 창에서 **액터**를 선택하자.



4. 디폴트로 선택되어 있던 **콘텐츠** 폴더에 애셋이 생성될 것이다. 생성된 블루프린트 클래스의 이름을 **BP_Cube**로 지정하자. **BP_Cube**를 더블클릭하여 블루프린트 에디터를 열자. 다음과 같은 모습일 것이다.

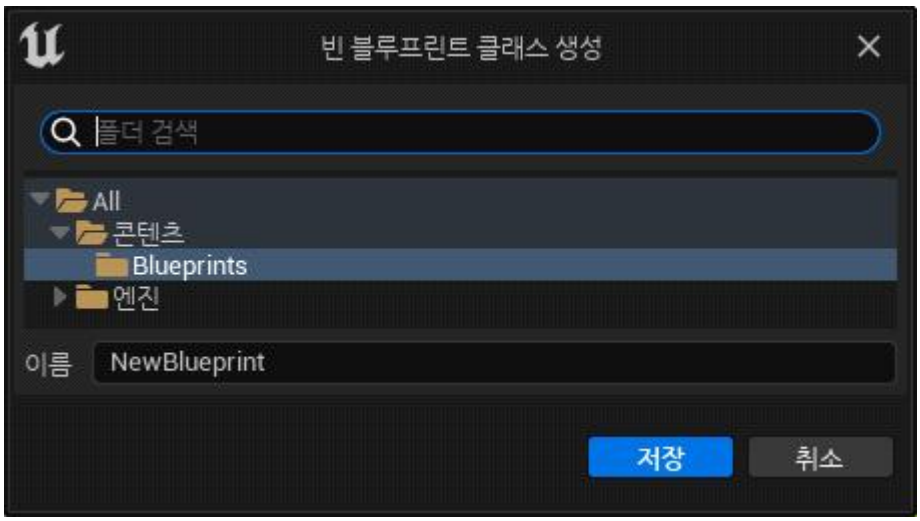


5. 블루프린트 클래스를 생성하는 또다른 방법이 있다.
먼저 블루프린트 에디터를 닫고 **콘텐츠 브라우저**에서 이미 생성해둔 **BP_Cube** 애셋을 **Del** 키를 눌러 삭제하자.

이제 다른 방법으로 블루프린트 클래스를 생성해보자.
툴바의 **블루프린트** 아이콘을 클릭하자. 그리고 드롭다운 메뉴에서 **새 빈 블루프린트 클래스...**를 클릭하자.

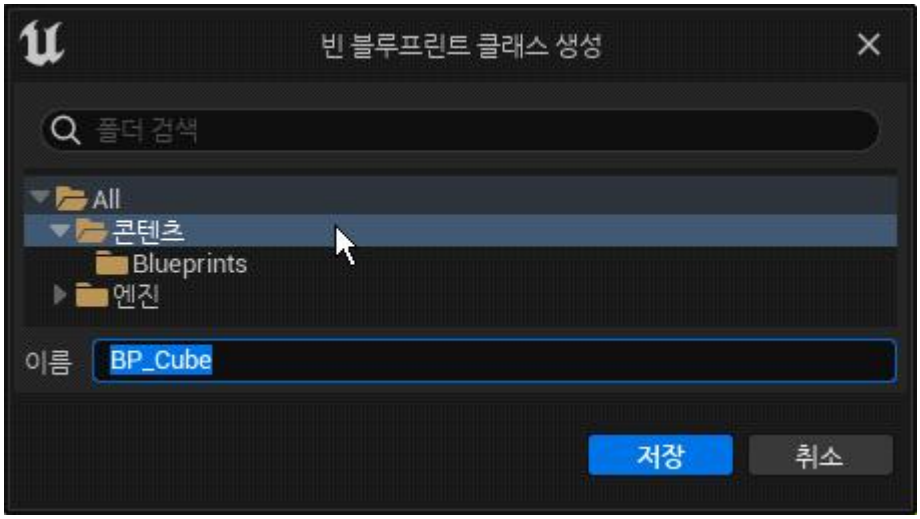
이전과 동일하게 **부모 클래스 선택** 창이 뜰 것이다. 이전과 동일하게 **액터**를 선택하자.

6. 그다음에, **빈 블루프린트 클래스 생성** 창이 뜰 것이다. 아래와 같이 보일 것이다.



만약 그대로 저장 버튼을 클릭한다면 콘텐츠 폴더 아래에 새로운 **Blueprints** 폴더가 생기게 되고 그 아래에 **NewBlueprint.uasset** 애셋 파일이 생길 것이다.

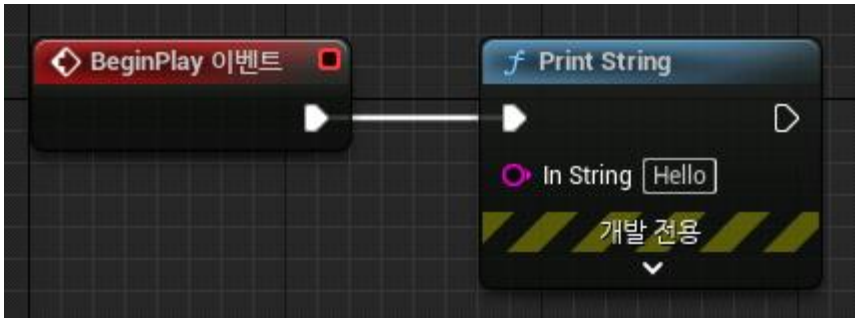
7. 우리는 폴더 구조가 이전과 동일하게 되도록 **콘텐츠** 폴더를 선택하고 **이름**에는 **BP_Cube**를 입력하자.



저장을 클릭하면 블루프린트 클래스가 생성된다. 또한 블루프린트 에디터도 자동으로 띄워준다. 지금까지 블루프린트 클래스를 생성하는 방법을 학습하였다.

8. 이제부터, 블루프린트 에디터에서 그래프를 작성해보자.
먼저, 블루프린트 에디터의 레이아웃을 살펴보자.
블루프린트 에디터는 **레벨 블루프린트 에디터**와 거의 유사하지만 약간의 차이가 있다. 먼저, **블루프린트 에디터**에는 **컴포넌트** 탭이 추가되어 있다. 한 액터 클래스는 여러 컴포넌트의 컨테이너이고 각 컴포넌트에 대한 편집 기능이 있어야 하기 때문이다. 반면에 **레벨 블루프린트**는 특정 액터의 내부를 명시하는 것이 아니라 레벨에 대한 스크립트를 명시하는 것이기 때문에 **컴포넌트** 탭이 없다.
그리고, 에디터의 레이아웃 중간에도 차이가 있다. **레벨 블루프린트 에디터**에서는 **이벤트 그래프** 탭만 있었는데, **블루프린트 에디터**에서 **뷰포트** 탭과 **Construction Script** 탭이 추가되어 있다. 뷰포트 탭에는 액터 내부의 씬 컴포넌트가 가질 수 있는 메시의 시각적인 모습을 보여주고 **Construction Script** 탭에서는 액터의 생성 시에 초기화 과정을 실행하는 스크립트를 명시한다. **레벨 블루프린트**에서는 개별 액터의 내부 요소를 명세하는 것이 아니므로 **뷰포트** 탭과 **Construction Script** 탭은 필요없다.
또한, **내 블루프린트** 탭에서 **이벤트 그래프**에서의 초기 설정에도 차이가 있다. **레벨 블루프린트 에디터**에서는 **BeginPlay 이벤트**와 **Tick 이벤트**의 두 이벤트 노드만 있었는데 **블루프린트 에디터**에서는 **ActorBeginOverlap 이벤트**가 추가되어 있다. 이것은 액터의 겹침시작 이벤트 노드이므로 블루프린트 에디터에만 있다.

9. 이벤트 그래프 탭을 클릭하자.
BeginPlay 이벤트 노드, **Tick 이벤트** 노드, **ActorBeginOverlap 이벤트** 노드의 세 노드가 이미 배치되어 있을 것이다. **BeginPlay 이벤트** 노드는 이 액터 인스턴스가 생성될 때에 최초도 한번 호출된다. **Tick 이벤트** 노드는 매 프레임 틱 신호마다 한번씩 호출된다. **ActorBeginOverlap 이벤트** 노드는 이 액터에 겹침시작 이벤트가 발생할 때에 호출된다. 자주 사용되는 이벤트 노드이어서 디폴트로 배치해두고 있다.
우리는 이전에서와 같이 **BeginPlay 이벤트** 노드 뒤에 **PrintString** 노드를 추가하여 연결하자.



컴파일하고 저장하자.

10. 플레이 버튼을 클릭하여 실행해보자. `Hello`가 출력되지 않을 것이다.
그 이유는 **BP_Cube** 액터 인스턴스가 생성되지 않았으므로 **BeginPlay 이벤트** 노드도 호출되지 않기 때문이다.
레벨에 배치된 모든 액터는 레벨을 플레이하면 자동으로 모두 생성된다.
따라서 **콘텐츠 브라우저**의 **콘텐츠** 폴더에서 **BP_Cube**를 드래그해서 레벨에 배치하자.

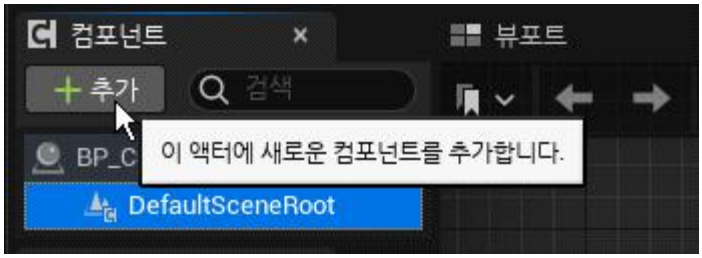
BP_Cube 엑터는 내부에 비어있는 씬 컴포넌트만 하나 가지고 있는 형체가 없는 액터이므로 아무 위치에나 배치해도 상관없다.
그러나, 나중을 위해서 이전과 동일하게 약간 공중에 있도록 (0,0,110)에 배치하자.

11. 플레이 버튼을 클릭하여 실행해보자. 이제 `Hello`가 출력될 것이다. 예상대로 **BP_Cube** 액터 인스턴스가 생성되고 **BeginPlay 이벤트** 노드가 호출되는 것이다.

12. 조금 더 복잡한 예를 작성해보자.

BP_Cube 블루프린트 에디터로 돌아가자. 이제 큐브 모양의 스태틱 메시 컴포넌트를 추가해보자.

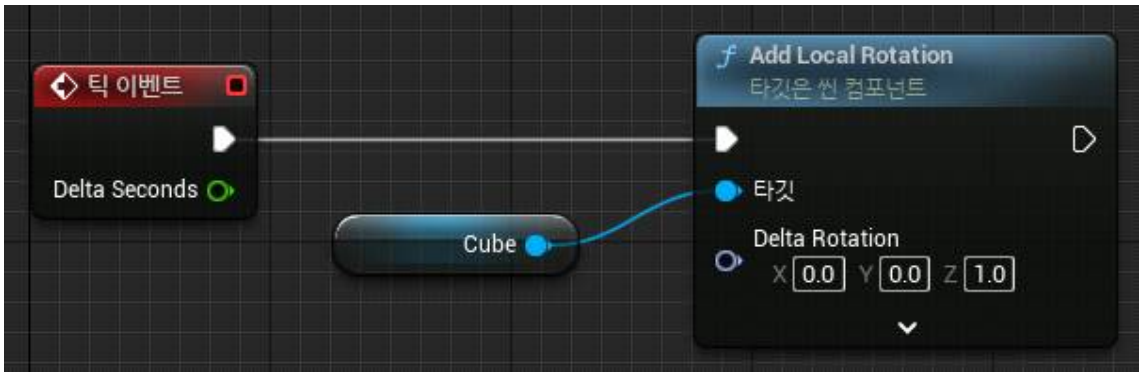
컴포넌트 탭에서 **+추가** 버튼을 클릭하자.



13. 그다음, 드롭다운 메뉴에서 큐브를 선택하자.

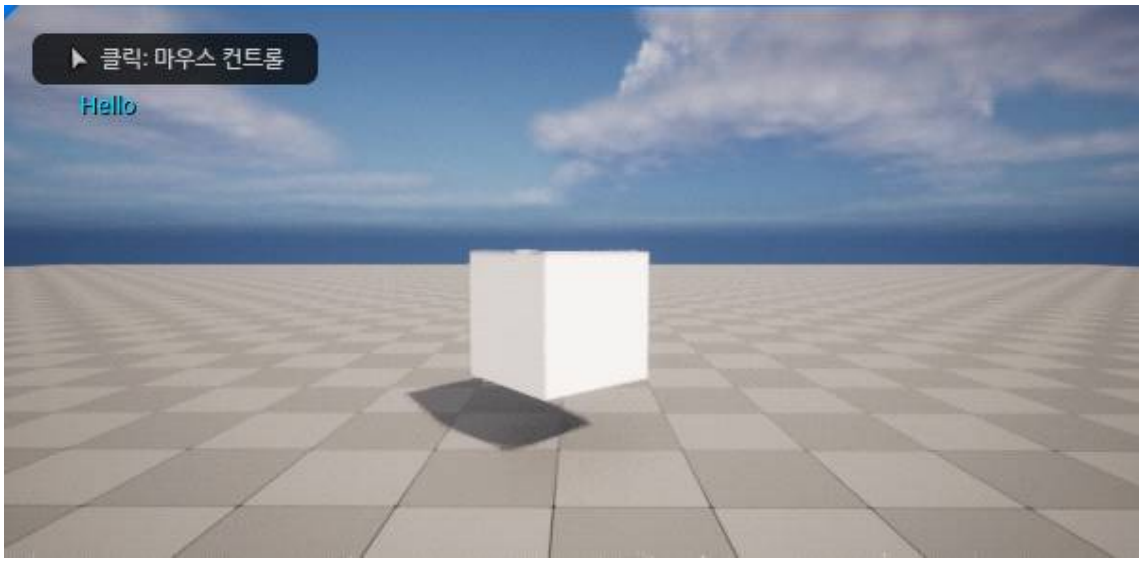
14. 큐브 스태틱 메시 컴포넌트가 추가될 것이다.

15. 그다음, **컴포넌트** 탭에서 **Cube**를 드래그해서 이벤트 그래프 탭의 격자판에 배치하자. **Cube** 컴포넌트의 레퍼런스 노드가 배치될 것이다.
이제, **Cube** 레퍼런스 노드의 출력핀을 드래그해서 **AddLocalRotation** 노드를 검색해서 배치하자. 그리고 노드의 입력핀 **DeltaRotation**의 **Z**값에 1을 입력하자.
그다음, **Tick 이벤트** 노드의 출력 실행핀을 **AddLocalRotation** 노드의 입력 실행핀에 연결하자. 완성된 모습은 아래와 같다.



BP_Cube 내의 컴포넌트들을 **트랜스폼**의 **모빌리티** 속성이 디폴트로 **무버블**로 되어있을 것이다. 따라서 따로 수정해주지 않아도 된다.
컴파일하고 저장하자.

16. 플레이해보자. 이전과 동일하게 상자가 스핀할 것이다.



17. 이제, **BP_Cube**를 두 개 정도를 드래그해서 추가로 배치해보자. 첫 번째는 (0,-200,110)에 배치하고 두 번째는 (0,200,110)에 배치하자.
플레이해보자. 세 상자가 모두 스핀할 것이다. 또한 `Hello`도 세 번 출력될 것이다. 레벨에 배치된 세 액터 인스턴스가 모두 각각 독립적으로 실행되고 있기 때문이다.
만약 이후에 블루프린트 클래스 애셋의 스크립트를 수정하게 되면, 레벨에 배치된 동일 타입 애셋의 다른 모든 액터들에도 자동으로 수정된 내용이 반영된다.

지금까지 처음으로 블루프린트 클래스에서 스크립트를 작성하는 방법을 학습하였다.

□