13_ 변수와 함수 기초

<제목 차례>

인천대학교 컴퓨터공학부 박종승

1. 개요

이 장에서는 변수와 함수에 대해서 학습한다.
스크립트를 작성하기 위해서는 반드시 변수와 함수를 사용할 수 있어야 한다. 변수는 정하지 않은 값을 보관할 수 있는 저장소이다. 함수는 입력으로부터 주어진 특정 작업을 수행하여 출력을 생성하는 스크립트 묶음이다. 이러한 변수와 함수의 개념은 프로그래밍에서 가장 기본적인 개념이다. 한편, 프로그래밍의 경험이 없는 경우라고 하더라도 간단한 수준의 지식만으로도 언리얼의 블루프린트 스크립트를 작성할 수 있다. 이 장에서는 프로그래밍에 전문적이지 않은 블루프린트 입문자를 대상으로 변수와 함수의 개념을 설명한다.

2. 변수의 생성과 사용

이 절에서는 변수의 생성과 사용에 대해서 학습한다.
변수 유형에 대해서 알아보자. 선택 가능한 변수 유형은 아래의 그림과 같다.

각 유형에 대해서 알아보자.

분류	유형명	컬러	설명	C++ 유형
기본 유형	부울(Boolean)		참 또는 거짓의 두 값만을 가짐	bool
	바이트(Byte)		[0,255] 범위의 정수를 가짐	unsigned char
	인티저(Integer)		$[-2^{31},2^{31}-1]$ 범위의 정수를 가짐	int
	인티저64(Integer64)		$[-2^{63},2^{63}-1]$ 범위의 정수를 가짐	long
	플로트(Float)		소숫점 숫자값을 가짐	double
문자열	네임(Name)		명칭으로 사용되는 문자열	
	스트링(String)		일반적인 문자열	string
	텍스트(Text)		화면에 출력하기 위한 문자열	
빈번한 구조체	벡터(Vector)		세 숫자 (X,Y,Z)를 가지는 구조체	
	로테이터(Rotator)		3D 공간에서 회전을 정의하는 구조체	
	트랜스폼(Transform)		트랜스폼 구조체	
구조체	구조체		구조체	struct
인터페이스	인터페이스		인터페이스	
객체	오브젝트 타입		블루프린트 객체	
열거형	열거형		열거형	enum

먼저, **부울**(Boolean), **바이트**(Byte), **인티저**(Integer), **인티저64**(Integer64), **플로트**(Float)는 C++에서의 기본 유형 **bool**, **unsigned char**, **int**, **long**, **double**에 해당한다. **바이트**는 1 바이트 크기이며, **인티저**는 4바이트 크기이며, **플로트**와 **인티저64**는 8바이트 크기이다. 참고로, 부동소숫점 실수 유형의 경우에 UE4에서는 32비트 **플로트**와 64비트 **더블**이 함께 존재했으나 UE5에서는 64비트로 하나로 통일되고 이름을 **플로트**로 하였다.
다음으로, 문자열는 **네임**(Name), **스트링**(String), **텍스트**(Text)의 세 종류가 있다.

먼저, **스트링**은 일반적인 문자열에 해당한다. 내부에 문자의 배열로 저장하고 있다. 값을 검색하거나 수정하거나 비교하는 모든 연산이 가능하다. 가장 무거운 표현 방식이다. 한편, **네임**이나 **텍스트**는 문자 배열을 공유 사용하며 연산이 인덱스를 사용하여 가볍게 수행되도록 하고 있다. **네임**은 애셋 등의 명칭으로 사용되는 가벼운 버전의 문자열이다. 대소문자 구분이 없고, 값이 생성된 후에는 수정할 수 없는 불변(immutable) 특성을 가진다. **텍스트**는 화면에 출력하여 사용자에게 보여주기 위한 문자열로 사용된다. 텍스트의 현지화(localization) 기능을 제공한다.
다음으로, 빈번하게 사용되는 구조체로 **벡터**(Vector), **로테이터**(Rotator), **트랜스폼**(Transform)이 있다.
벡터는 **X**,**Y**,**Z**의 세 **플로트** 값을 가진다. 3D 좌표나 RGB 컬러를 표시할 때 사용된다.
로테이터는 **X(Roll)**, **Y(Pitch)**, **Z(Yaw)**의 세 **플로트** 값을 가진다. 각 축에 대한 회전값을 각도단위의 실수값으로 표현한다.
트랜스폼은 내부에 **Location**, **Rotation**, **Scale**의 세 구조체를 가진다. **Location**과 **Scale**은 **벡터** 유형이고 **Rotation**은 **로테이터** 유형이다. 이들은 **위치**와 **스케일**과 **회전**을 표현한다.
다음으로, **구조체**와 **열거형**에는 엔진에서 제공하는 많은 구조체 유형과 열거형 유형이 있다.
다음으로, **오브젝트 타입**에는 엔진에서 제공하는 객체 유형이 있다.
다음으로, **인터페이스**에는 특별한 유형인 인터페이스 유형이 있다.

<참고> 변수에 대한 자세한 내용은 다음의 문서를 참조하자.
https://docs.unrealengine.com/blueprint-variables-in-unreal-engine/

<참고> 문자열에 대한 자세한 내용은 다음의 문서를 참조하자.
https://docs.unrealengine.com/4.27/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/StringHandling/

이제부터, 변수의 노드에 대해서 알아보자.
변수의 노드에는 **Get** 노드와 **Set** 노드가 있다.
Get 노드는 변수로부터 값을 추출하는 노드이고 **Set** 노드는 변수에 값을 할당하는 노드이다. **Get** 노드와 **Set** 노드를 getter와 setter라고도 하며, 객체 지향 코딩에서 내부 데이터를 외부의 직접 접근으로부터 보호하면서 동시에 내부 데이터의 무결성을 유지하기 위한 방법이다.
아래의 두 노드는 **부울** 유형의 변수 **NewVar_0**의 **Get** 노드와 **Set** 노드를 보여주고 있다.

두 노드의 가장 큰 차이는 실행핀이다. **Get** 노드에는 실행핀이 없고 **Set** 노드에는 실행핀이 있다. **Get** 노드에는 실행핀이 없는 이유는 **Get** 노드에서 값을 특정 순간에 미리 읽어오는 것이 아니라 **Get** 노드의 출력값이 실제로 사용될 때에 그 순간에 읽어오겠다는 의미이다. 따라서 **Get** 노드의 실행 여부 및 실행 순서는 **Get** 노드의 출력핀이 연결된 다른 노드의 실행 여부 및 순서에 의존된다. **Set** 노드에는 변수에 저장할 값을 입력하는 입력핀이 있다. 상수값을 명시할 수도 있다. 또한 출력핀도 있다. 이 출력핀은 **Get** 노드의 기능으로 활용할 수 있도록 한다.

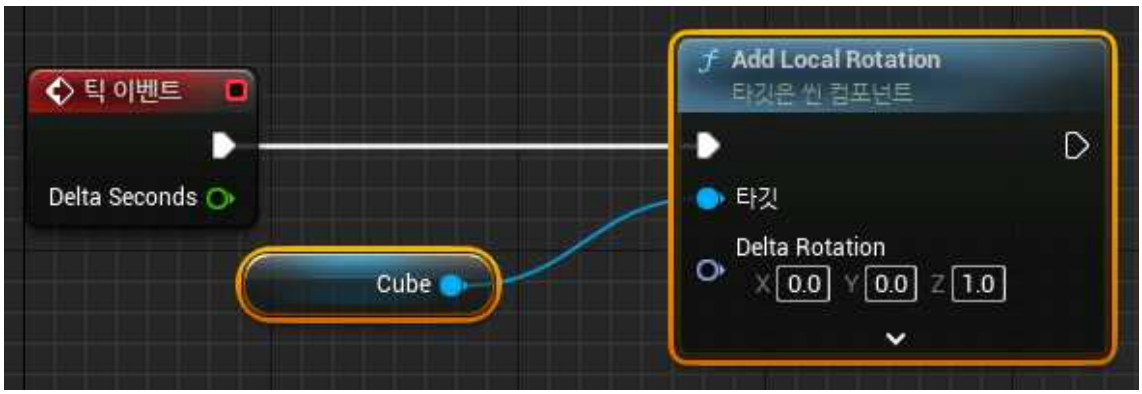
<참고> 클래스 블루프린트에 있는 변수의 속성 중에서 **인스턴스 편집가능**이 체크된 경우에는 해당 액터를 맵에 배치했을 때 레벨 에디터의 디테일 패널에서 배치된 액터 인스턴스의 해당 값을 변경할 수 있게 된다. 레벨에 여러 개의 인스턴스가 배치된 경우에 에디터에서 각 인스턴스마다 값을 다르게 지정해줄 수도 있다.

<참고> 생성된 변수의 디테일 탭에 나열되는 속성 중에 **프라이빗** 속성이 있다. 이 체크박스에 체크하여 프라이빗으로 설정한 변수는 블루프린트의 이벤트 그래프나 함수 그래프 내부에서만 추출 또는 할당할 수 있고, 파생된 블루프린트 클래스에서는 변경할 수 없다.

---

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

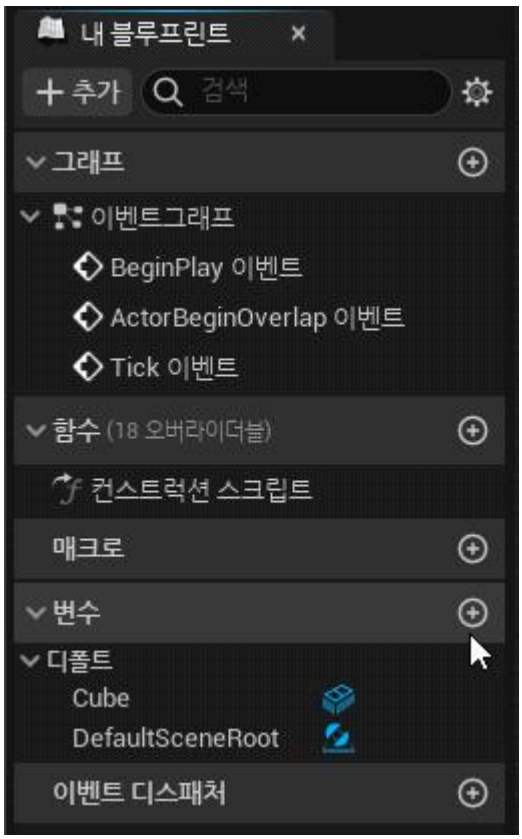
2. 블루프린트 클래스를 만들자.
이전 예제에서 만들었던 스핀되는 큐브인 **BP_MyCube**를 동일한 방법으로 만들어보자.
먼저, **콘텐츠 브라우저**에서 툴바에서 **+추가**를 클릭하고, 드롭다운 메뉴에서 **블루프린트 클래스**를 선택하고, **부모 클래스 선택** 창에서 **Actor**를 부모 클래스로 선택하자. 이름은 **BP_MyCube**으로 수정하자.
그다음, **BP_MyCube**을 더블클릭하여 **블루프린트 에디터**를 열자. **컴포넌트** 탭에서 **+추가**를 클릭하고 **큐브**를 선택하자. 큐브 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.
그다음, 콘텐츠 브라우저에서 **BP_MyCube**를 드래그하여 레벨에 배치하자. 위치는 (300,0,100)으로 하자.
플레이해보자. 스핀되는 큐브가 보일 것이다.
Z각도가 양수인 1의 값이고, 위에서 내려다 보았을 때에 시계방향으로 회전하는 것을 확인할 수 있다.

3. **BP_MyCube**를 더블클릭하여 블루프린트 에디터를 열자.

그다음, 왼쪽의 **내 블루프린트** 탭에서 **변수** 항목의 오른쪽에 있는 + 아이콘을 클릭하자.



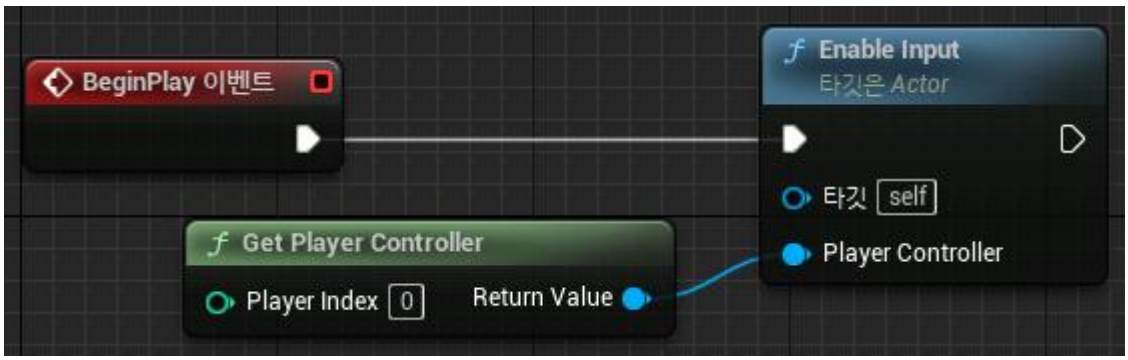
아이콘을 클릭한 후에는 디폴트로 이름이 **NewVar_0**이고 유형이 **부울**인 변수가 만들어진다.

4. 변수 이름을 클릭하여 **MyCounter**로 수정하자. 유형을 클릭하여 **인티저**로 수정하자.

유형을 쉽게 구분할 수 있도록 유형명 앞에 컬러를 함께 표시한다. **인티저**는 청녹색으로 표시된다. 그리고, 유형명 오른쪽에 눈 모양의 희미한 아이콘이 있다. 클릭하면 눈을 감은 모양과 눈을 뜬 모양으로 토글한다. 디폴트로는 눈을 감은 모양으로 되어 있다.
눈을 감은 모양인 경우에는 변수의 디폴트 값을 명시하는 것은 이 블루프린트 에디터의 디테일 탭에서만 가능하다. 만약 눈을 뜬 모양으로 바꾸어 두면 추가적으로 레벨에 배치된 각각의 인스턴스에 대해서도 에디터의 디테일 탭에서 디폴트 값을 수정할 수 있다.

5. 액터에서는 기본적으로 입력 이벤트가 전달되지 않는다. 입력 이벤트가 액터에게도 전달되도록 하려면 액터에서 수동으로 입력을 활성화해주어야 한다.
BeingPlay 이벤트 노드의 근처의 빈 곳에서 우클릭하고 **EnableInput** 노드를 배치하자.
그다음, 배치된 **EnableInput** 노드의 **PlayerController** 입력핀을 왼쪽으로 드래그하고 **GetPlayerController**

노드를 배치하자. 그래프가 아래와 같이 완성될 것이다.

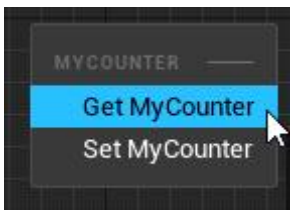


이제부터 플레이어 컨트롤러로 오는 입력 이벤트가 이 액터에게도 오게 된다.

<참고> 액터에 입력을 지정하는 것에 대한 자세한 내용은 다음의 문서를 참조하자.
https://docs.unrealengine.com/4.27/InteractiveExperiences/Input/ActorInput/

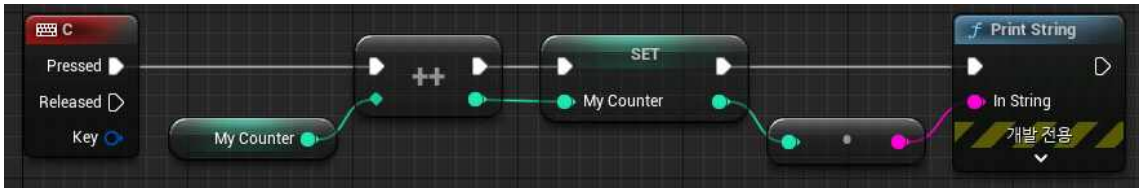
6. 이벤트 그래프의 격자판의 빈 곳에서 우클릭하고 **Input** » **키보드 이벤트** 아래의 **C** 키 입력 이벤트 노드를 배치하자.

7. **내 블루프린트** 탭에서 **MyCounter** 변수를 드래그해서 격자판에 드롭하자.



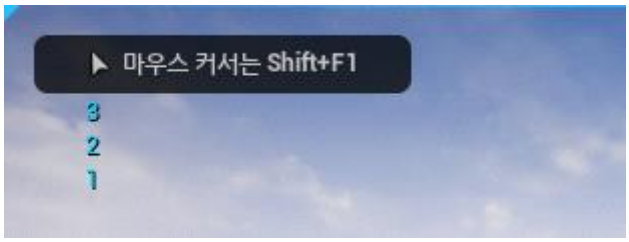
Get 노드를 배치할지 또는 **Set** 노드를 배치할지를 묻는 대화창이 뜬다. 우리는 각각 하나씩 배치하자. 한편, 선택을 위한 대화창없이 바로 노드를 배치할 수 있다. **Ctrl+드래그**로 배치하면 **Get** 노드가 배치되고, **Alt+드래그**로 배치하면 **Set** 노드가 배치된다.

8. **Get** 노드의 출력핀을 당기고 **++**를 검색하여 **IncrementInt** 노드를 배치하자.
그다음, **++**노드의 출력핀을 **Set** 노드의 **MyCounter** 입력핀에 연결하자.
그다음, **C** 입력 이벤트 노드의 **Pressed** 실행핀을 **++**노드의 실행핀에 연결하고, 그다음의 **Set** 노드의 실행핀에 연결하자.
그다음, **Set** 노드의 출력 실행핀을 드래그하고 **PrintString** 노드를 배치하자. 그리고 **Set** 노드의 출력핀을 **PrintString** 노드의 **InString** 입력핀에 연결하자. 자동 타입변환 노드가 삽입될 것이다.



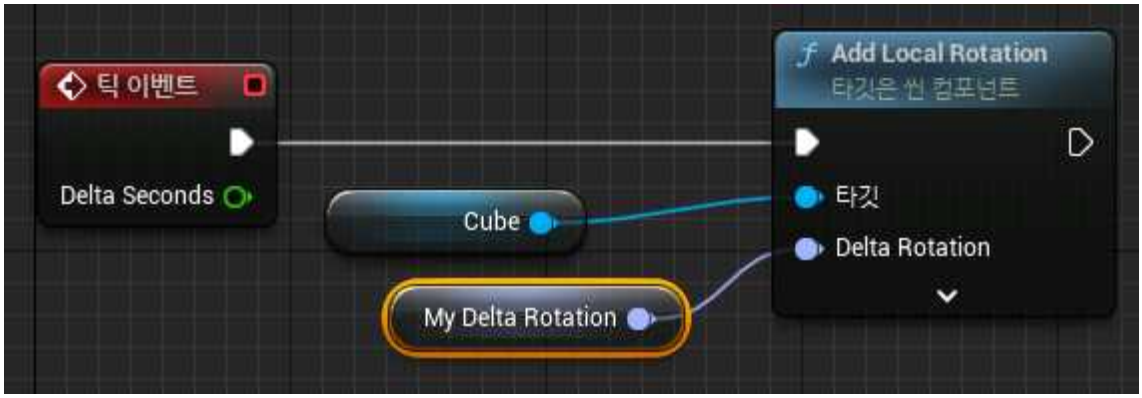
컴파일하고 저장하자.

9. 플레이해보자. **C** 키를 여러번 눌러보자. 1부터 시작하여 증가되는 숫자가 출력될 것이다.



10. 이제, 상수값을 변수로 승격하는 기능에 대해서 알아보자. **Tick 이벤트** 노드로 이동하자. 다음과 같은 그래프가 있을 것이다. 우리는 **AddLocalRotation** 노드의 **DeltaRotation** 입력핀에 (0,0,1)의 상수값을 지정하였다. 상수값 대신 변수값을 넣어주도록 해보자. 우선, **DeltaRotation** 입력핀을 왼쪽으로 드래그 해서 빈 곳에 드롭하면 액션선택 창이 뜬다. 액션선택 창의 위에 **변수로 승격**이 나타나는데 이것을 선택하자.

11. **변수로 승격**을 선택하면 **로테이터(Rotator)** 유형의 새로운 변수가 추가된다. 변수 이름은 디폴트 이름인 **Delta Rotation**을 **MyDeltaRotation**으로 바꾸어주자. 바꾸는 방법은 **내 블루프린트** 탭에서 변수 영역에 있는 변수명을 클릭하고 입력하면 된다. 또는 변수를 선택하고 **F2** 키를 누른 후에 입력하면 된다.



컴파일하고 **MyDeltaRotation**의 **디테일** 탭을 살펴보자. 기본값 영역에 **MyDeltaRotation**의 디폴트 값으로 이전의 상수값이었던 (0,0,1)이 그대로 명시되어 있을 것이다. 다만, 컴파일을 한 후에야 변수의 디폴트 값이 나타나므로 변수가 추가된 후에는 다시 컴파일하도록 하자.

12. 이제, 이벤트 그래프의 격자판의 빈 곳에서 우클릭하고 **Input** » **키보드 이벤트** 아래의 **R** 키 입력 이벤트 노드를 배치하자.
그다음, **내 블루프린트** 탭에서 **MyDeltaRotation** 변수의 **Get** 노드를 배치하자.
그다음, **Get** 노드의 출력핀을 드래그하고 ***를 검색하여 **곱하기** 함수를 배치하자. 그리고, **곱하기** 함수 노드의 입력핀에 -1을 입력하자.
그다음, **MyDeltaRotation** 변수의 **Set** 노드를 배치하자. 그리고, **곱하기** 노드의 출력핀을 **Set** 노드의 **MyDeltaRotation** 입력핀에 연결하자.
그다음, **R** 키 이벤트 노드의 **Pressed** 실행핀을 **Set** 노드의 입력 실행핀에 연결하자. 다음과 같은 모습의 그래프가 될 것이다.

13. 플레이해보자. **R** 키를 여러번 눌러보자. 누를 때마다 회전 방향이 바뀔 것이다.

지금까지, 변수에 대해서 학습하였다.

3. 구조체의 생성과 사용

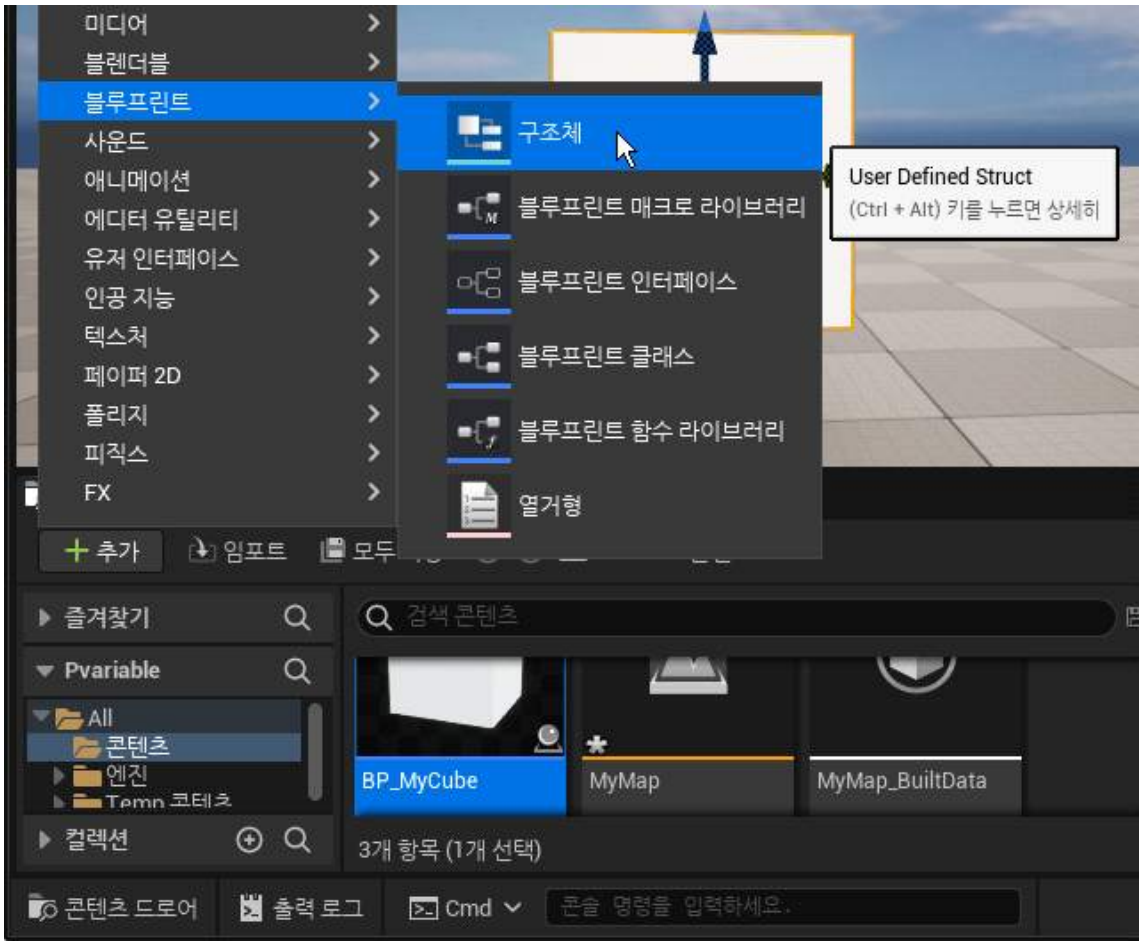
이 절에서는 구조체 변수의 생성과 사용에 대해서 학습한다.

<참고> 구조체에 대해서는 다음의 문서를 참조하자.
https://docs.unrealengine.com/4.27/ProgrammingAndScripting/Blueprints/UserGuide/Variables/Structs/

---

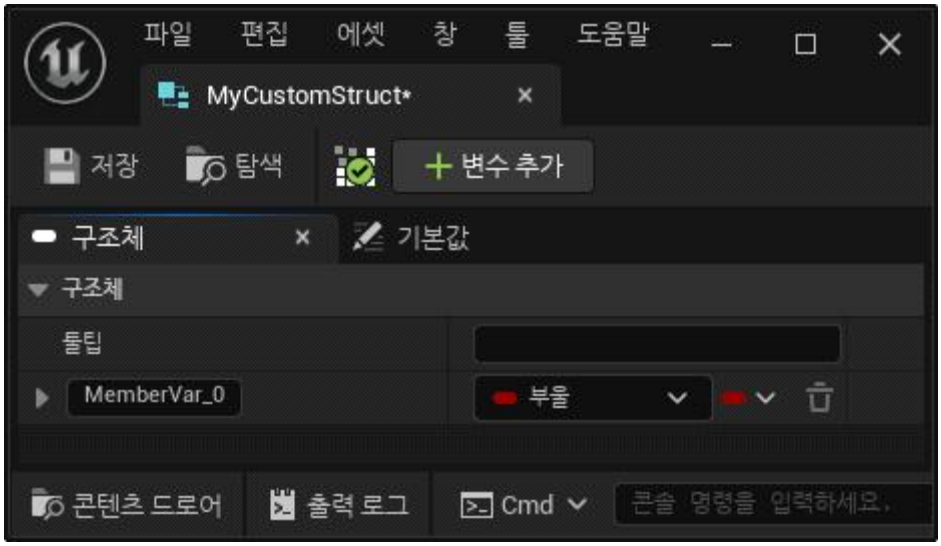
1. 이전 프로젝트 `Pvariable`에서 계속하자.

2. 커스텀 구조체를 생성해보자.
콘텐츠 브라우저에서 +추가를 클릭하고 **블루프린트** » **구조체**를 선택하자.



생성된 커스텀 구조체의 이름을 **MyCustomStruct**라고 수정하자.

3. **MyCustomStruct**를 더블클릭하여 **구조체 에디터**를 열자. **구조체 에디터**는 별도의 에디터가 아니라 블루프린트 에디터의 일부로 생각하면 된다.



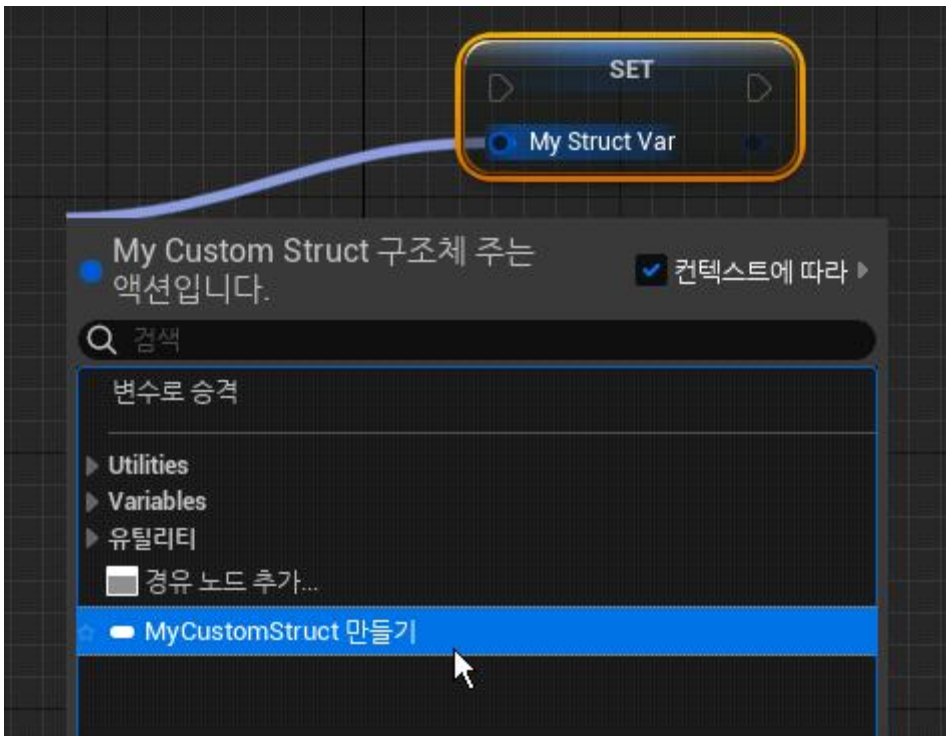
4. 툴바의 **+변수 추가** 버튼을 클릭할 때마다 하나씩의 요소가 구조체에 추가된다. 버튼을 세 번 클릭하여 세 개의 요소를 추가해보자.

그다음, 세 요소의 이름을 디폴트인 **MemberVar_0**, **MemberVar_1**, **MemberVar_2**에서 **HealthPoints**, **HasArmor**, **Nickname**로 수정하자. 그다음, 각 요소의 유형을 각각 **인티저**, **부울**, **스트링** 타입으로 지정하자. 각 **툴팁** 입력상자에도 의미를 쉽게 설명하는 문구를 입력해두기를 권장한다. 여기서는 생략하자.
구조체 내에서는 요소의 순서도 의미가 있다. 그 순서에 맞도록 메모리에 저장되기 때문이다. 따라서 순서가 중요한 경우에는 그 순서대로 요소를 추가해야 한다.
한편, 현재의 **구조체** 탭 옆에 **기본값** 탭이 있다. 여기에 디폴트로 사용될 구조체 요소의 초기값을 명시하면 된다. 초기에는 디폴트값이 지정된다. 디폴트값은 **인티저**는 **0**이고 **부울**은 **False**이고 **스트링**은 빈 문자열이다. 우리는 그대로 두자.
이제 저장하고 창을 닫자.

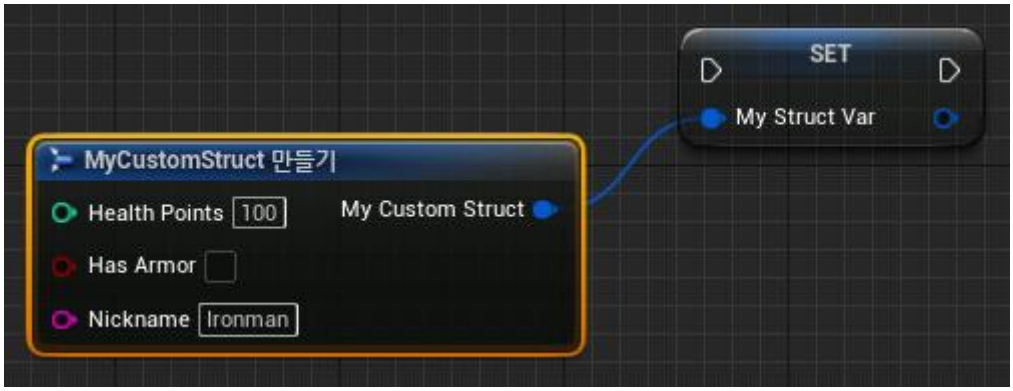
5. 이제, 레벨 블루프린트 에디터를 열자.
내 블루프린트 탭에서 변수를 추가하자. 변수 이름을 **MyStructVar**라고 하자.
디폴트 유형인 **부울**로 되어 있는 변수 유형 아이콘을 클릭하자. 유형 선택 창에서 바로 전에 생성하였던 우리의 커스텀 구조체 유형인 **MyCustomStruct**를 검색해보자.

우리의 커스텀 구조체 유형 **MyCustomStruct**을 변수의 유형으로 사용할 수 있음을 알 수 있다. 이 유형을 선택하여 지정하자.

6. **내 블루프린트** 탭에서 **MyStructVar**를 드래그해서 **Set** 노드를 배치하자.
Set 노드의 입력핀을 왼쪽으로 드래그해서 떼면 액션선택 창이 뜬다. 아래에 있는 **MyCustomStruct 만들기**를 선택하자.

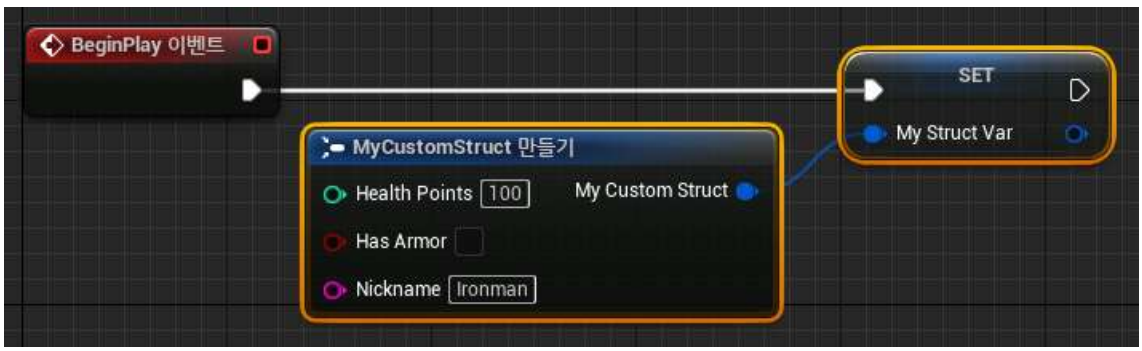


7. 배치된 구조체 만들기 노드의 입력핀에서 **HealthPoints**에는 100을 입력하고 **Nickname**에는 `Ironman`을 입력하자.

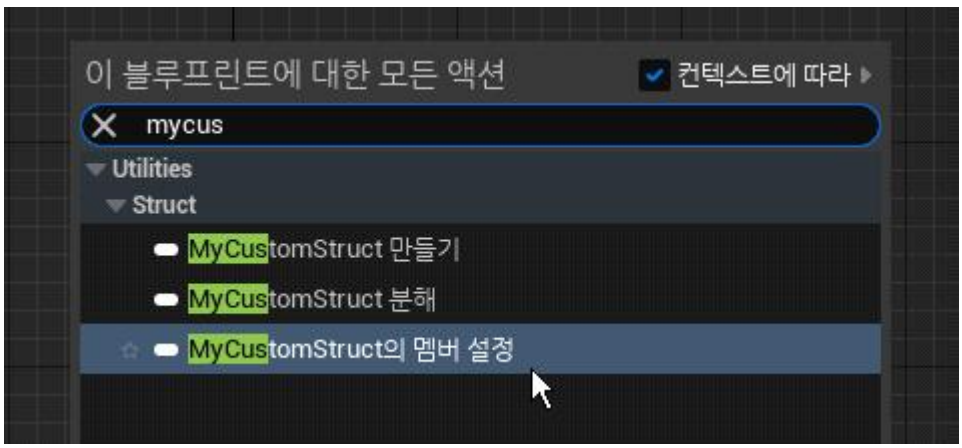


참고로, 구조체 만들기 노드를 클릭하면 디테일 탭에서 **핀 옵션**이 나타난다. 여기서 구조체의 각 요소를 체크박스로 선택해서 입력핀이 보이지 않도록 할 수도 있다. 노드를 간단하게 표시하고 싶은 경우에 사용하면 된다.

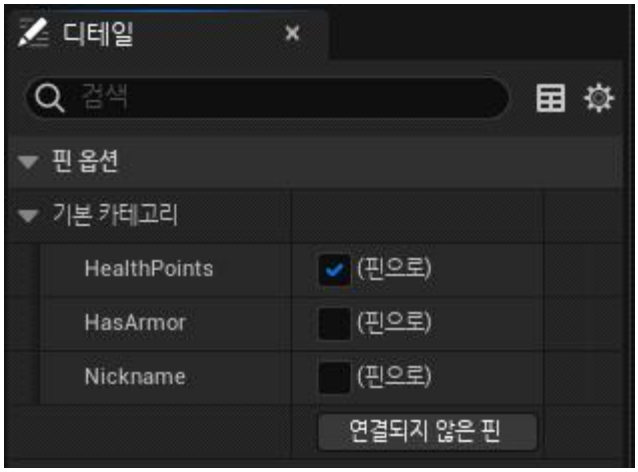
8. 그다음, **BeginPlay 이벤트** 노드의 실행핀을 **Set** 노드의 입력 실행핀에 연결하자.



9. 격자판의 빈 곳에서 우클릭하고 **MyCustomStruct**를 검색해보자. 세 가지 노드가 나열된다. 이번에는 **MyCustomStruct의 멤버 설정**을 선택하여 노드를 배치하자.

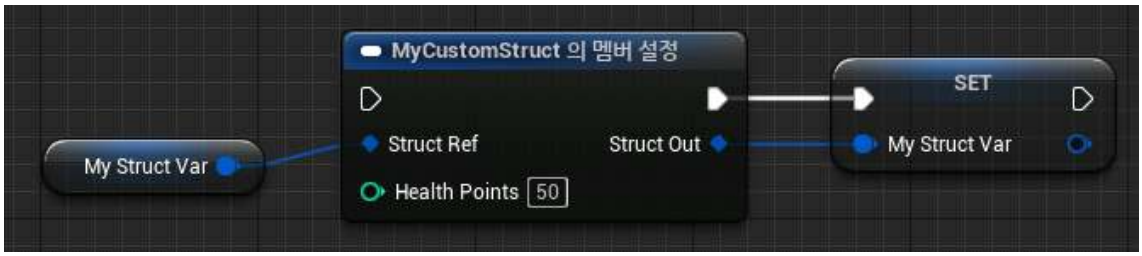


10. 배치된 **MyCustomStruct의 멤버 설정** 노드를 클릭하고 오른쪽 디테일 탭을 보자. 구조체의 모든 요소가 나열되어 있고 체크해제되어 있을 것이다. 여기에서 특정 입력핀에 체크하면 체크된 핀이 멤버 설정 노드의 입력핀으로 나타나게 된다. 우리는 **HealthPoints**에 체크하자. 이제 노드에서 **HealthPoints** 입력핀이 보일 것이다.

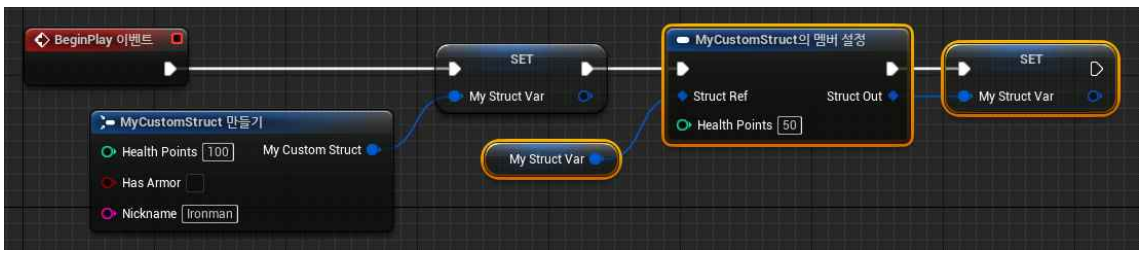


11. 구조체의 **멤버 설정** 노드는 구조체의 특정 요소의 값만 선택적으로 바꾸고자 하는 경우에 사용되는 노드이다. 우리는 구조체 변수의 **HealthPoints**의 값만 바꾸고 나머지 값은 그대로 유지시키고자 한다.

내 블루프린트 탭에서 드래그하여 **MyStructVar**의 **Get** 노드와 **Set** 노드를 하나씩 배치하자. 그리고, **Get** 노드를 **멤버 설정** 노드의 **Struct Ref** 입력핀에 연결하자. 그리고, **멤버 설정** 노드의 **Struct Out** 출력핀을 **Set** 노드의 입력핀에 연결하자. **멤버 설정** 노드의 실행핀도 **Set** 노드의 실행핀에 연결하자. 그다음, **HealthPoints** 입력핀에 50을 지정하자.

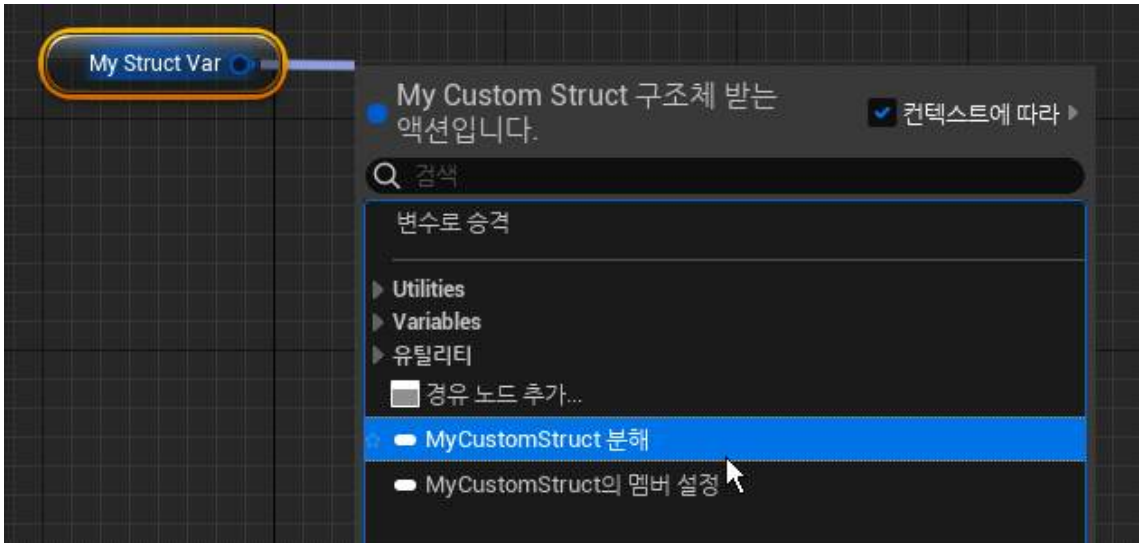


12. 그다음, **BeginPlay 이벤트** 노드 그래프의 마지막에 연결되도록 그래프의 마지막 **Set** 노드의 출력 실행핀을 **멤버 설정** 노드의 입력 실행핀에 연결하자.

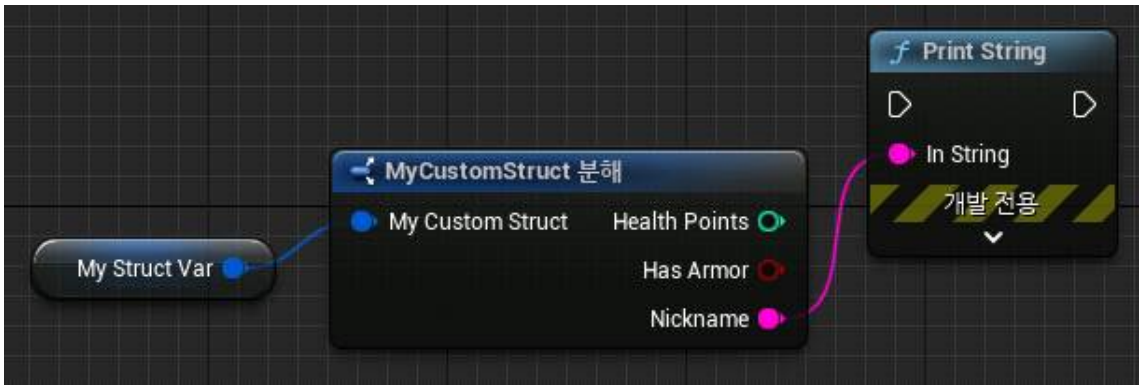


13. **내 블루프린트** 탭에서 **MyStructVar**를 드래그해서 **Get** 노드를 배치하자.

Get 노드의 출력핀을 드래그해서 떼면 액션선택 창이 뜬다. 아래에 있는 **MyCustomStruct 분해**를 선택하자. 이 노드는 구조체의 각 요소를 접근할 수 있도록 해준다.

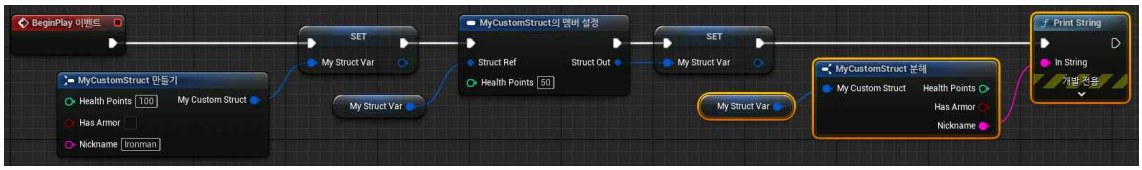


14. 그리고 **PrintString** 노드를 배치하고 구조체의 분해 노드의 **Nickname** 출력핀을 연결하자.

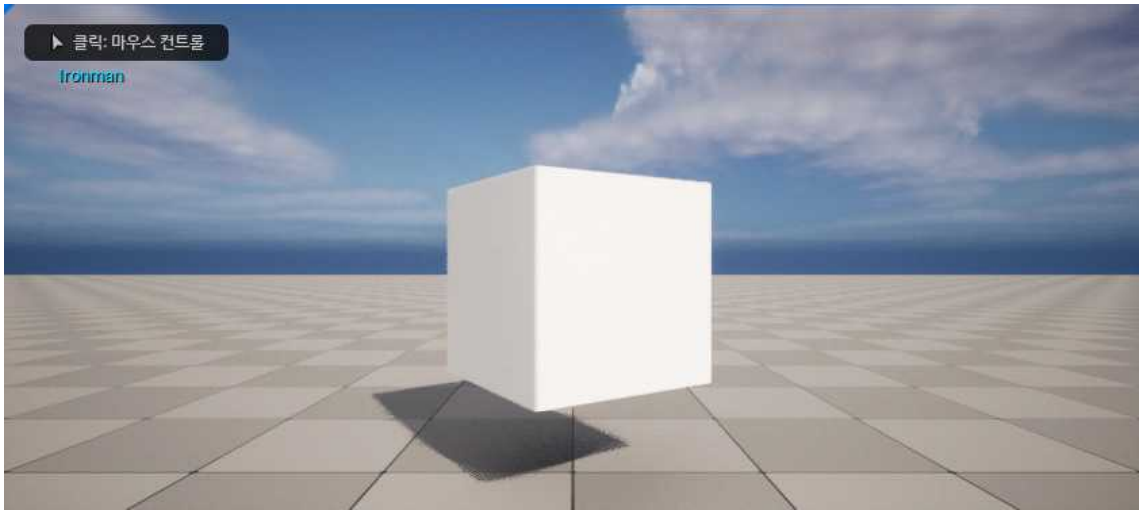


참고로, 구조체 분해 노드를 클릭하면 디테일 탭에서 핀 옵션이 나타난다. 여기서 구조체의 각 요소를 체크박스로 선택하여 출력핀이 보이지 않도록 할 수 있다. 또한 **연결되지 않은 핀 숨김** 버튼을 클릭하면 다른 노드와의 연결이 없는 모든 출력핀을 나타나지 않게 체크박스 해제한다.

15. 그다음, **BeginPlay 이벤트** 노드 그래프의 마지막에 연결되도록 그래프의 마지막 **Set** 노드의 실행핀을 **PrintString** 노드의 입력 실행핀에 연결하자.
이제 **BeginPlay 이벤트** 그래프는 다음과 같이 보일 것이다.



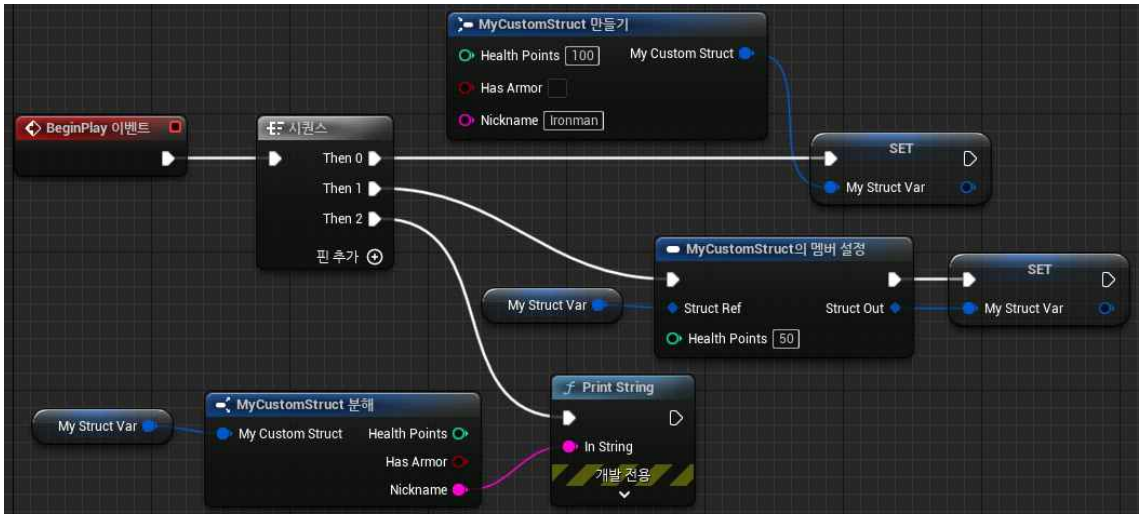
16. 플레이해보자. 화면에 `Ironman`이 출력될 것이다.



17. 한편, **BeginPlay 이벤트** 그래프를 보면 순차적으로 실행되는 노드 부분들이 너무 좌우로 길게 연결되어 있어 보기에 불편하다.
이런 경우에 적당한 흐름 제어 노드가 있다. 바로 **시퀀스** 노드이다. 빈 곳에서 우클릭하고 **Sequence**를 검색하여 시퀀스 노드를 배치하자. 많이 사용되는 노드여서 키보드 **S** 키를 누른 상태에서 좌클릭하면 바로 배치된다.

이 노드는 입력 실행핀으로 펄스가 들어오면 출력 실행핀들을 순차적으로 펄스를 보내준다. 즉 가장 먼저 **Then 0** 출력핀에 펄스를 보내고 연결된 모든 노드를 실행한다. 모든 노드의 실행이 종료된 후에 **Then 1** 출력핀에 펄스를 보내고 연결된 모든 노드가 실행한다. 핀 추가 버튼을 눌러 필요한 만큼의 **Then** 출력핀을 추가할 수 있다.

18. **시퀀스** 노드를 사용하여 **BeginPlay 이벤트** 그래프를 다음과 같이 정리하여보자.
기존의 연결을 끊으려면 핀 위에서 **Alt+좌클릭**하면 된다.



지금까지, 구조체의 생성과 사용에 대해서 학습하였다.

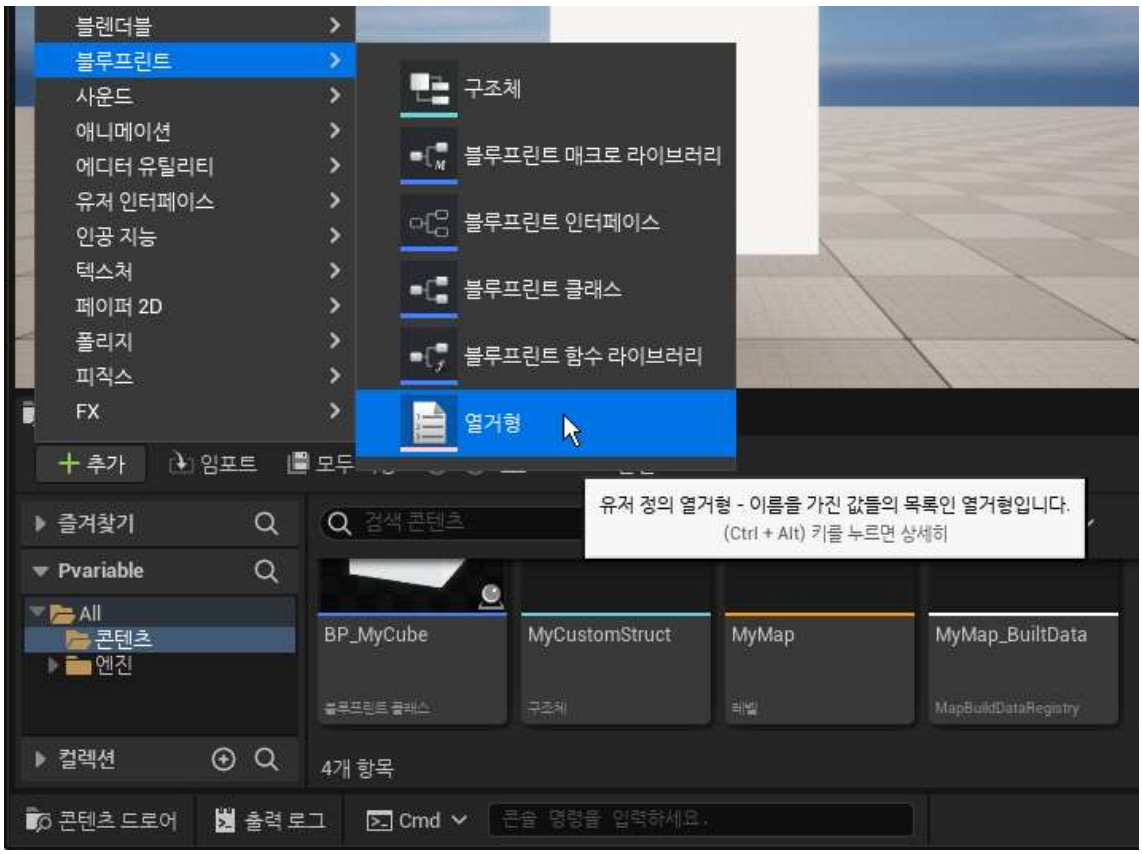
4. 열거형 변수의 생성과 사용

이 절에서는 열거형 변수의 생성과 사용에 대해서 학습한다.

---

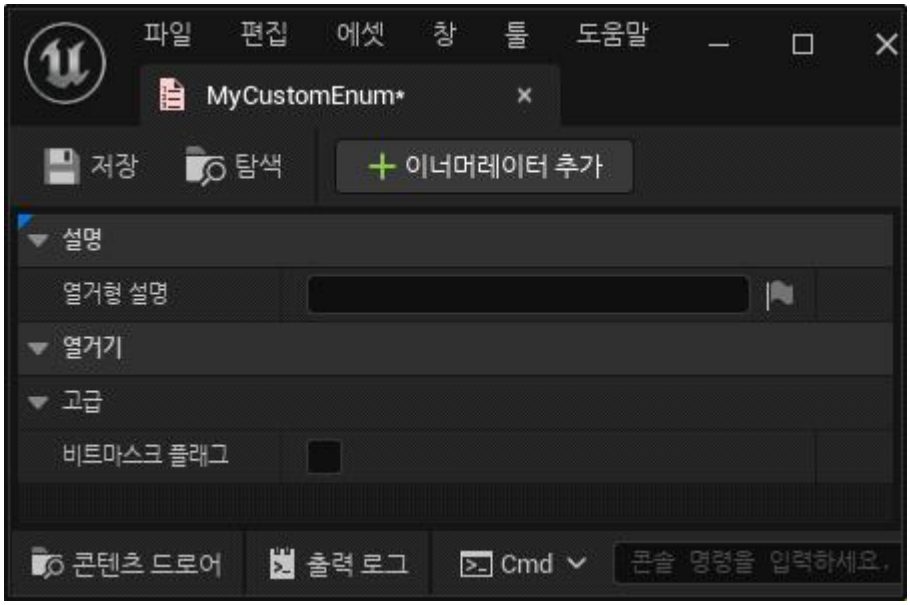
1. 이전 프로젝트 `Pvariable`에서 계속하자.

2. 커스텀 열거형을 생성해보자.
콘텐츠 브라우저에서 +추가를 클릭하고 **블루프린트** » **열거형**을 선택하자.

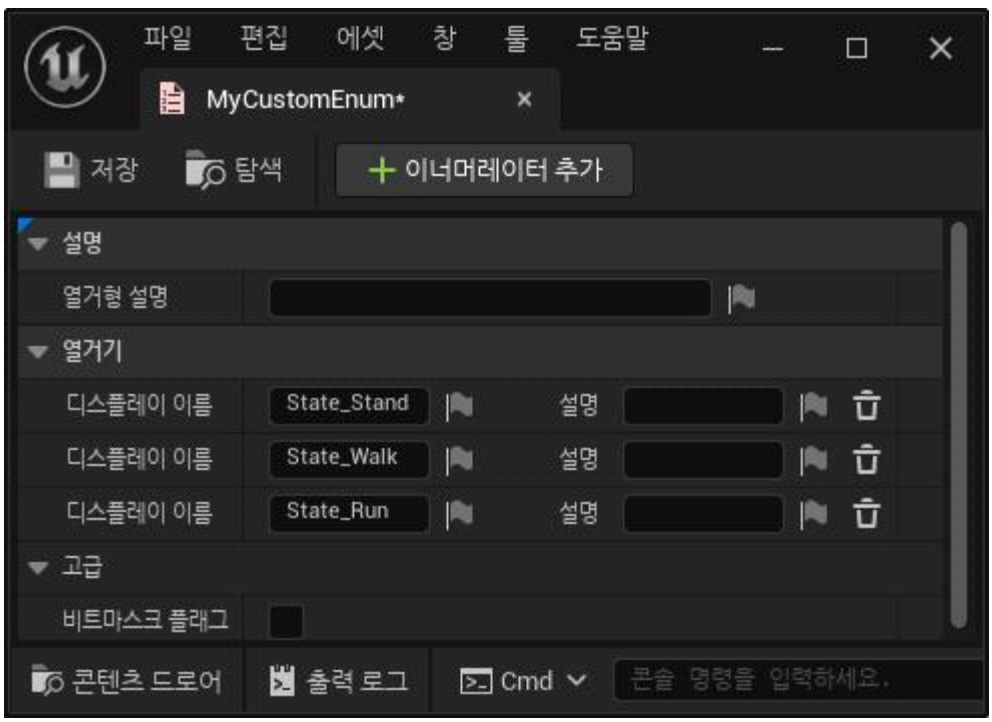


생성된 커스텀 열거형의 이름을 **MyCustomEnum**이라고 하자.

3. **MyCustomEnum**을 더블클릭하여 **열거형 에디터**를 열자. **열거형 에디터**는 별도의 에디터가 아니라 블루프린트 에디터의 일부로 생각하면 된다.



4. 툴바의 **+이너머레이터 추가** 버튼을 클릭할 때마다 하나씩의 열거형 요소가 추가된다. 다음과 같이 추가해보자.



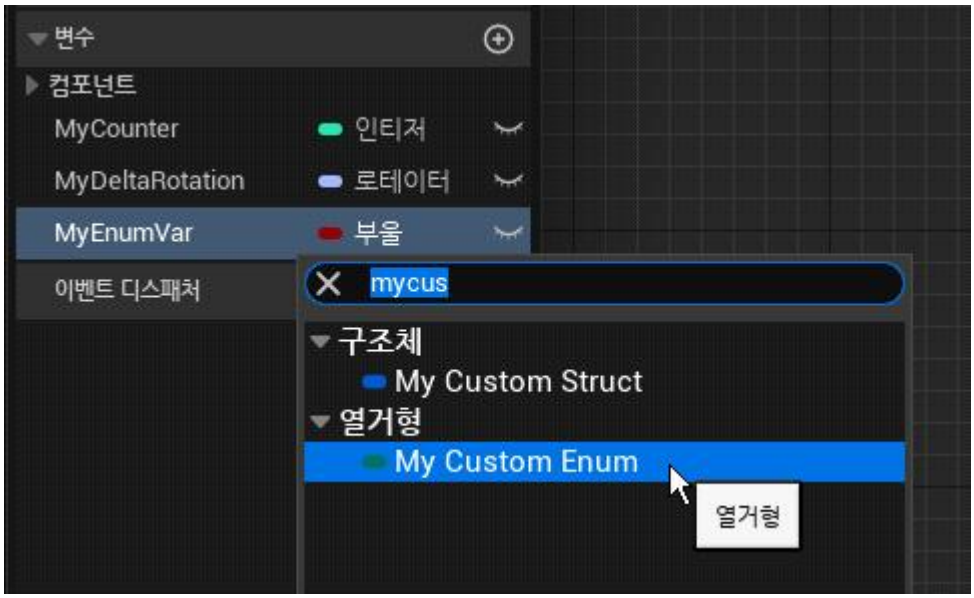
+이너머레이터 추가 버튼을 세 번 클릭하여 세 개의 열거형 요소를 추가하자.

각 요소의 **디스플레이 이름**이 디폴트로 **NewEnumerator0**, **NewEnumerator1**, **NewEnumerator2** 식으로 되어 있을 것이다. 이것을 **State_Stand**, **State_Walk**, **State_Run**로 수정하자. 각 **설명** 입력상자는 툴팁 기능을 제공하는데 사용되므로 해당 요소의 의미를 쉽게 설명하는 문구를 입력해두기를 권장한다. 여기서는 생략하자.

열거형 내에서는 요소의 순서도 의미가 있다. 그 순서에 맞도록 인덱스되기 때문이다. 따라서 요소의 순서도 고려해서 입력하자.

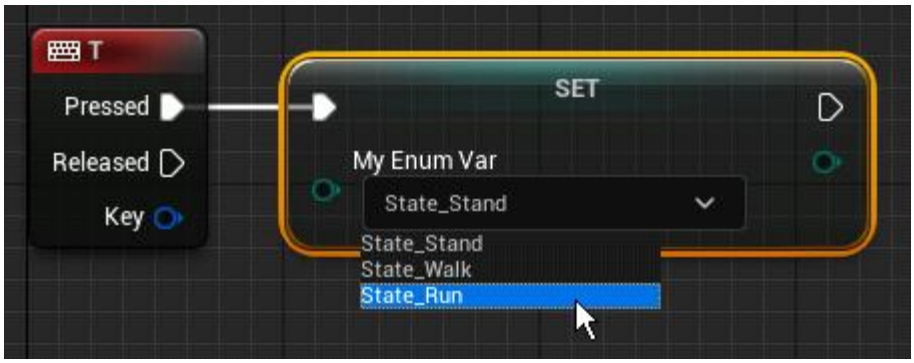
이제 저장하고 창을 닫자.

5. 이제, **BP_MyCube**의 블루프린트 에디터를 열자.
내 블루프린트 탭에서 변수를 추가하자. 변수 이름을 **MyEnumVar**라고 하자.
변수 유형 아이콘을 클릭하고 바로 전에 생성하였던 우리의 커스텀 구조체 유형인 **MyCustomEnum**를 검색해보자.

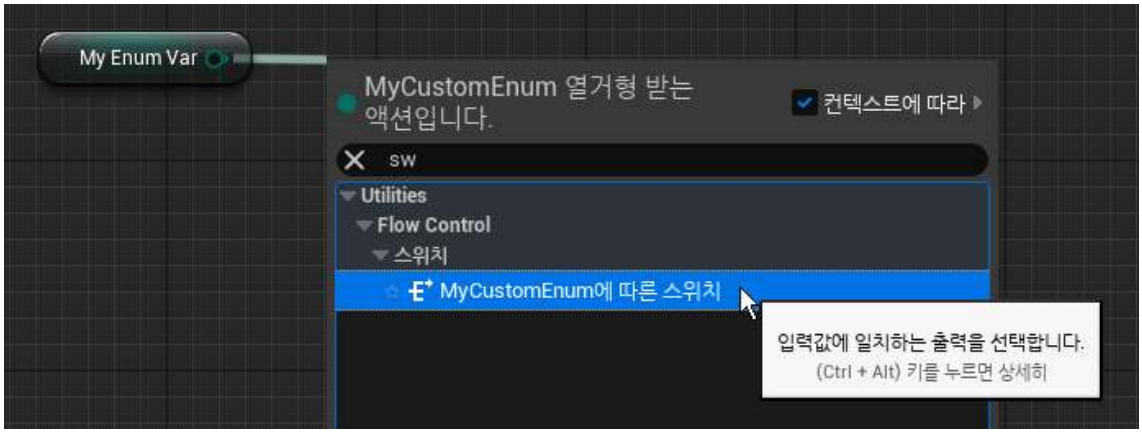


우리의 커스텀 구조체 유형 **MyCustomEnum**을 변수의 유형으로 사용할 수 있음을 알 수 있다. 이 유형을 선택하여 지정하자.

6. 키보드 **T** 키 입력 이벤트 노드를 배치하자.
그다음, **내 블루프린트** 탭에서 **MyEnumVar**를 드래그해서 **Set** 노드를 배치하자.
그다음, **T** 키 입력 이벤트 노드의 **Pressed** 실행핀을 **Set** 노드의 입력 실행핀에 연결하자.
그다음, **Set** 노드의 입력핀이 디폴트로 **State_Stand**로 되어 있을 것이다. 이것을 **State_Run**으로 변경하자.



7. **내 블루프린트** 탭에서 **MyEnumVar**를 드래그해서 **Get** 노드를 배치하자. 그다음, **Get** 노드의 출력핀을 당기고 액션선택 창에서 검색하여 **MyCustomEnum에 따른 스위치** 노드를 배치하자. 이 **스위치** 노드는 **MyEnumVar**의 값에 따라서 출력 실행핀의 해당 핀으로 펄스를 내보낸다.



8. **스위치** 노드의 **State Run** 실행핀을 드래그하고 **PrintString** 노드를 배치하자. 노드의 InString 입력핀에 '`Test passed!`'를 입력하자.

9. 이전의 **T** 키 입력 이벤트 노드 그래프에서의 **Set** 노드의 출력 실행핀을 **스위치** 노드의 입력 실행핀에 연결하자. 그래프고 다음과 같이 구성된다.

10. 플레이해보자. **T** 키를 눌러보자. 키를 누를 때마다 화면에 `Test passed!`가 출력될 것이다.

지금까지, 열거형에 대해서 학습하였다.

5. 함수의 생성과 사용

이 절에서는 함수의 생성과 사용에 대해서 학습한다.
우리는 지금까지 주로 **이벤트 그래프**에 대해서 다루었다. 앞으로 학습할 **함수**도 **이벤트 그래프**와 같은 그래프의 다른 종류이다. 사실상 커스텀 이벤트 그래프를 작성하여 사용하는 것과 함수 그래프를 작성하여 사용하는 것은 활용에 있어서 크게 차이가 없다.

커스텀 이벤트 그래프와 함수 그래프의 차이점에 대해서 알아보자. 이벤트 그래프와 함수는 대부분의 사용 방법이 동일하게 때문에 둘을 구분하는 것이 어렵게 느껴질 수 있다. 이들의 차이점을 나열하면 다음과 같다.
먼저, 이벤트 그래프는 노드 네트워크를 작성할 때에 격자판을 다른 이벤트 그래프와 함께 사용할 수 있으나, 함수는 자신의 전용 격자판을 사용한다.
또한, 이벤트 그래프에서는 리턴이 없으나, 함수는 값을 리턴할 수 있다.
또한, 이벤트 그래프는 **Delay** 노드와 같은 잠복성(latent) 액션을 사용할 수 있고 타임라인 기능도 사용할 수 있다. 그러나 함수에서는 이들을 사용할 수 없다. 함수를 호출하면 실행이 즉각적으로 끝나야 한다.
또한, 이벤트에는 로컬 변수의 개념이 없으나, 함수는 함수 내부에서만 사용되는 로컬 변수를 정의하여 사용할 수 있다.
또한, 이벤트 그래프는 실행이 시작되는 진입점(entry point)이 여러 개가 있을 수 있지만, 함수는 진입점이 하나만 있다.

목표하는 기능의 작업을 커스텀 이벤트로 구현할 것인지 함수로 구현할 것인지에 대해서 결정하는 방법을 알아보자. 이 두 가지를 처음부터 엄격하고 구분할 필요는 없다. 정하기가 어려우면 일단 커스텀 이벤트로 구현하자. 이후에 출력값의 리턴이 필요해진 경우에는 구현된 커스텀 이벤트를 함수 형태로 바꾸자. 또한, 잠복성 액션이나 타임라인을 사용할 가능성이 없고 독립된 형태로 구현해 두고 싶은 경우에도 함수로 바꾸자.

이제부터는 조건식에 대해서 알아보자. 조건식(condition)은 프로그래밍에서 흐름 제어를 위해 꼭 필요한 필수적인 요소이다. UE에서도 여러 조건식 노드를 제공한다. 조건식 노드는 입력핀의 값을 검사하여 출력핀으로 참 또는 거짓을 출력한다. 조건식 노드에는 <, >, <=, >=, ==, != 등의 노드가 있다.
또한, 조건식과 조건식의 조합으로 더 복잡한 조건식을 만들 수 있다. 조합하는 노드로 **AND**, **OR**, **NAND**, **NOR**, **XOR** 등의 노드가 있다.

이제부터 함수의 생성과 사용에 대해서 알아보자.

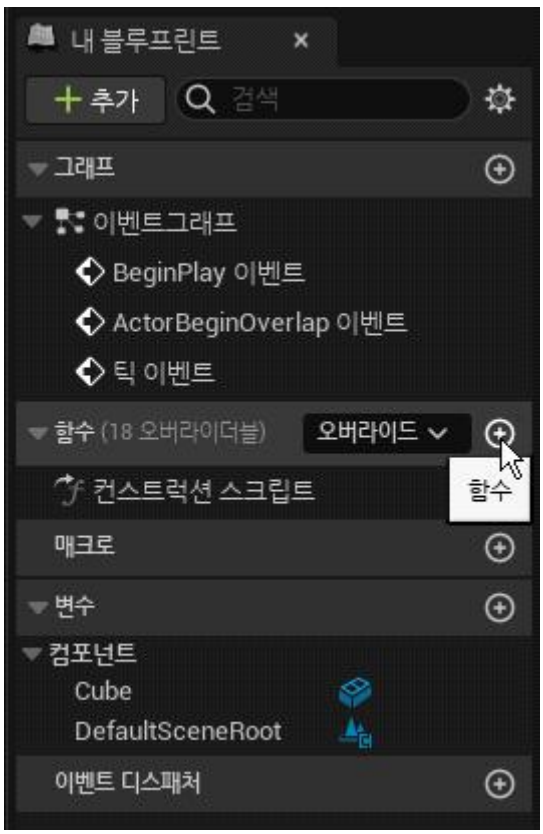
---

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

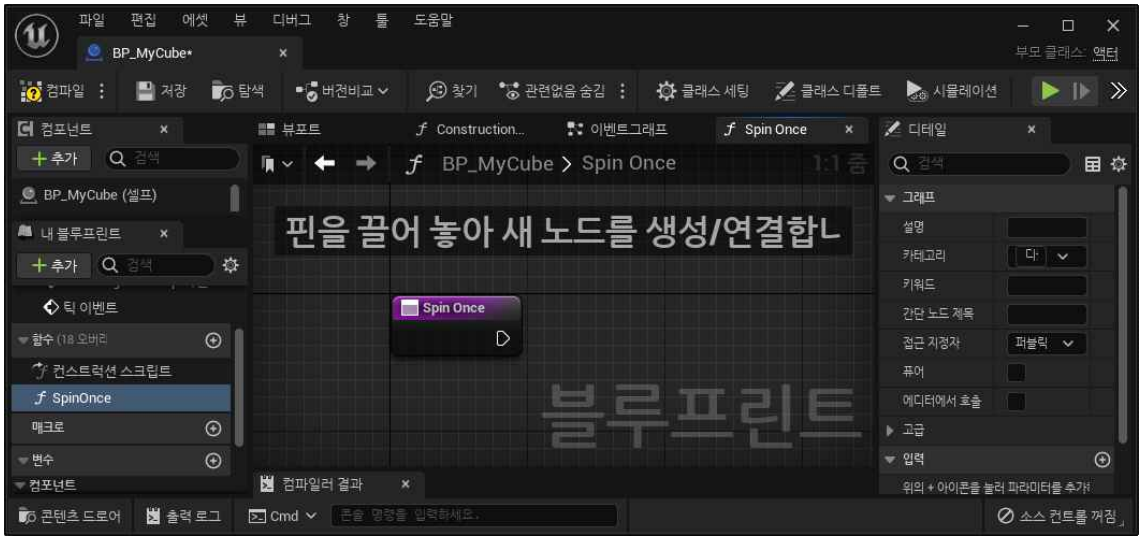
2. 블루프린트 클래스로 만들자.
이전 예제에서 만들었던 스핀되는 큐브인 **BP_MyCube**를 동일한 방법으로 만들어보자.
먼저, **콘텐츠 브라우저**에서 툴바에서 **+추가**를 클릭하고, 드롭다운 메뉴에서 **블루프린트 클래스**를 선택하고, **부모 클래스 선택** 창에서 **Actor**를 부모 클래스로 선택하자. 이름은 **BP_MyCube**으로 수정하자.
그다음, **BP_MyCube**을 더블클릭하여 **블루프린트 에디터**를 열자. **컴포넌트** 탭에서 **+추가**를 클릭하고 **큐브**를 선택하자. 큐브 스태틱 메시 컴포넌트가 추가될 것이다. 디폴트로 **Cube**라는 이름으로 그대로 두자.

3. 이제, 함수를 추가해보자. **내 블루프린트** 탭에서 **함수** 영역의 오른쪽에 있는 + 아이콘을 클릭하자.



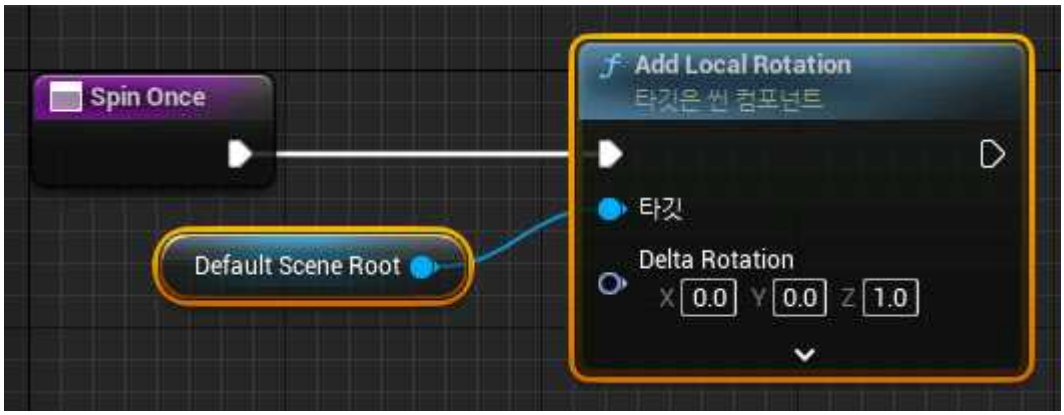
디폴트 함수이름을 **SpinOnce**로 바꾸자.

4. 함수가 생성되면 다음과 같이 새로운 함수 그래프가 생긴다. 이벤트 그래프와는 별개로 함수 전용의 함수 그래프가 각 함수마다 생기는 것이다.



SpinOnce 함수 그래프에는 SpinOnce라는 보라색 함수 노드가 생긴다. 함수의 노드 네트워크의 시작점에 해당하는 노드이다. 우리는 여기서부터 노드 네트워크를 작성해나가면 된다.

5. 함수 그래프 탭에서 격자판의 빈 곳에에서 우클릭하고 **AddLocalRotation (DefaultSceneRoot)**를 선택하여 배치하자. 그리고, 노드의 **DeltaRotation** 입력핀의 **Z**에 1을 입력하자. 그리고, **SpinOnce** 함수 노드의 실행핀에 연결하자. 다음과 같은 모습이 될 것이다.



이전에서는 **AddLocalRotation (Cube)**를 배치하였으나 이번에는 **AddLocalRotation (DefaultSceneRoot)**를 배치하였다. 두 노드에 어떤 차이가 있는지 알아보자.

두 노드의 차이는 어느 컴포넌트의 회전을 바꾸는지에 대한 차이가 있다. **Cube** 컴포넌트의 회전을 바꿀지 또는 **DefaultSceneRoot** 컴포넌트의 회전을 바꿀지에 대한 차이다.

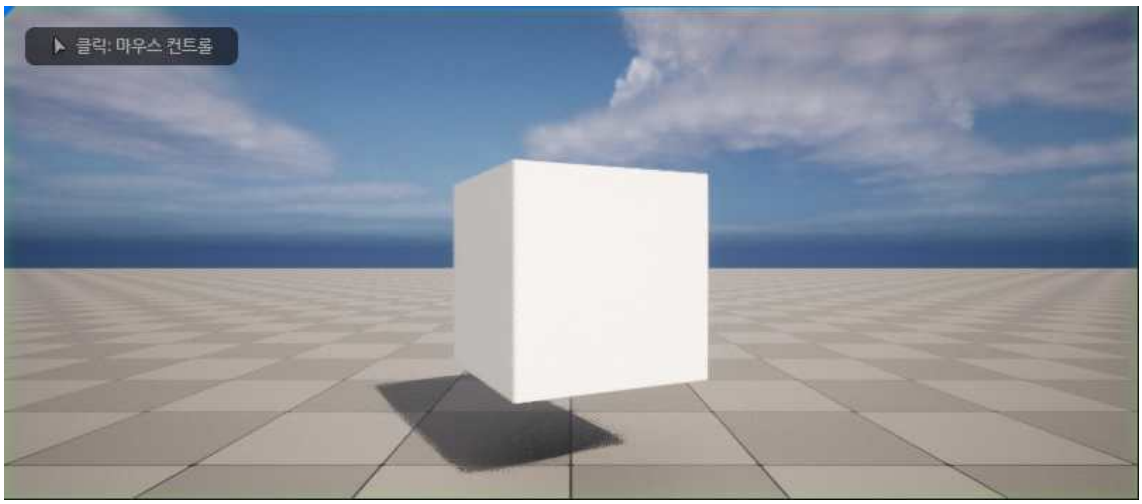
DefaultSceneRoot 컴포넌트는 계층 구조의 루트 노드에 해당하고 **Cube** 컴포넌트는 그 아래에 부착되어 있다. 계층 구조에서 각 컴포넌트는 상위 컴포넌트에 상대적인 트랜스폼으로 정의된다. 따라서 루트 컴포넌트의 트랜트폼을 바꾸면 그 아래의 모든 컴포넌트의 트랜스폼이 바뀌게 된다.

그리고, 액터의 트랜스폼을 따로 존재하는 것이 아니라 바로 루트 컴포넌트의 트랜스폼을 의미한다. 여기서 우리는 **DefaultSceneRoot** 컴포넌트의 회전을 바꾸었으므로 액터의 회전이 바뀌게 되는 것이다. 만약 이전과 같이 **Cube** 컴포넌트의 회전을 바꾸었다면 스태틱 메시는 회전되겠지만 루트 컴포넌트의 회전은 변함이 없이 그대로 있으면서 그 아래의 큐브만 국부적으로 회전되는 것이다.

6. 이벤트 그래프 탭으로 이동하자.

Tick 이벤트 노드의 실행핀을 드래그하고 SpinOnce 함수호출 노드를 검색하여 배치하자.
저장하고 컴파일하자.

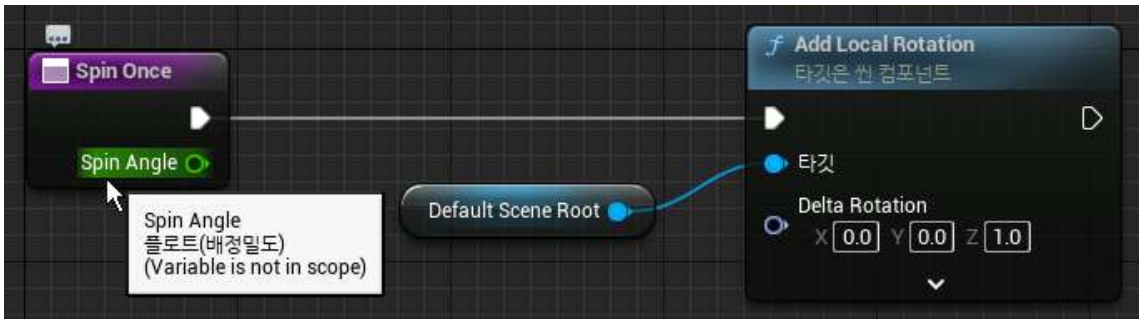
7. 레벨 에디터의 콘텐츠 브라우저에서 BP_MyCube를 드래그하여 레벨에 배치하자. 위치는 (300,0,100)으로 하자. 플레이해보자. 스핀되는 큐브가 보일 것이다.



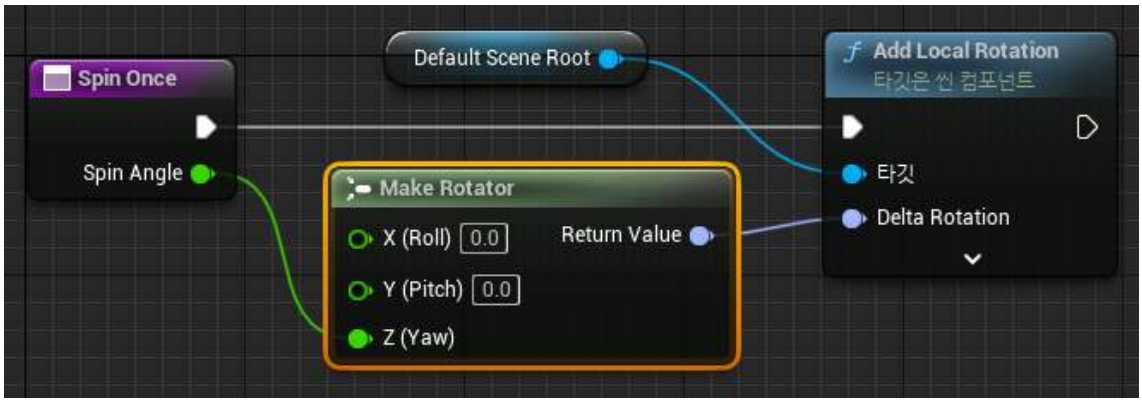
8. BP_MyCube 블루프린트 에디터의 **내 블루프린트** 탭에서 SpinOnce 함수를 더블클릭하여 함수 그래프로 이동하자.
SpinOnce에 입력 인자를 추가해보자. 먼저, 격자판에서 SpinOnce 함수 노드를 선택하자. 그다음, **디테일** 탭에서 **입력** 영역의 오른쪽 + 아이콘을 클릭하자.

9. 그다음, 입력인자명을 SpinAngle로 수정하고 인자유형을 **플로트**로 수정하자. 이제 입력인자를 추가하였다.

10. 함수 그래프에서 **SpinOnce** 함수 노드에 **SpinAngle** 출력핀이 추가되어 보일 것이다.



11. 함수 그래프에서 격자판의 빈 곳에 **MakeRotator** 노드를 검색하여 추가하자. 그다음, 함수 노드의 **SpinAngle** 출력핀을 **MakeRotator** 노드의 **Z(Yaw)** 입력핀에 연결하자. 그다음, **MakeRotator** 노드의 **Return Value** 출력핀을 **AddLocalRotation** 노드의 **DeltaRotation** 입력핀에 연결하자.

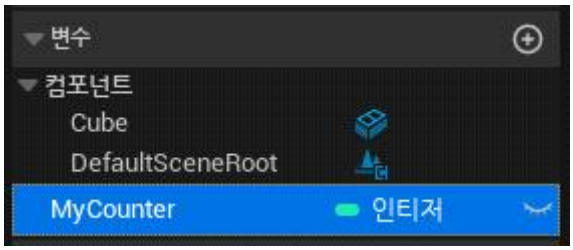


12. 이벤트 그래프 탭의 **Tick 이벤트** 그래프로 가서, **SpinOnce** 함수호출 노드의 **SpinAngle** 입력핀의 값을 **2**로 수정하자.

컴파일하자. 레벨을 플레이해보자. 이전보다 두 배 빠르게 큐브가 스핀될 것이다.

13. 이제, 변수를 추가하자.

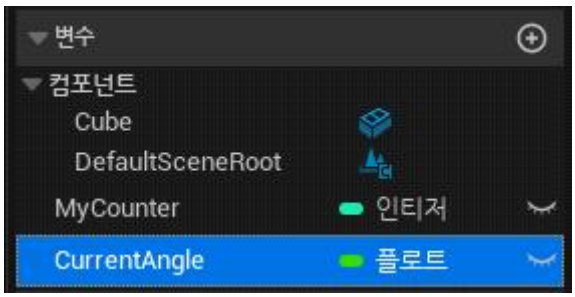
BP_MyCube 블루프린트 에디터의 **내 블루프린트** 탭에서 **변수** 항목의 오른쪽에 있는 + 아이콘을 클릭하자. 변수 이름을 클릭하여 **MyCounter**로 수정하자. 유형을 클릭하여 **인티저**로 수정하자.



MyCounter의 디폴트값은 **디테일** 탭의 하단에 기본값 영역에서 지정할 수 있다. 초기에 자동으로 0으로 설정되므로 우리는 그대로 두자.

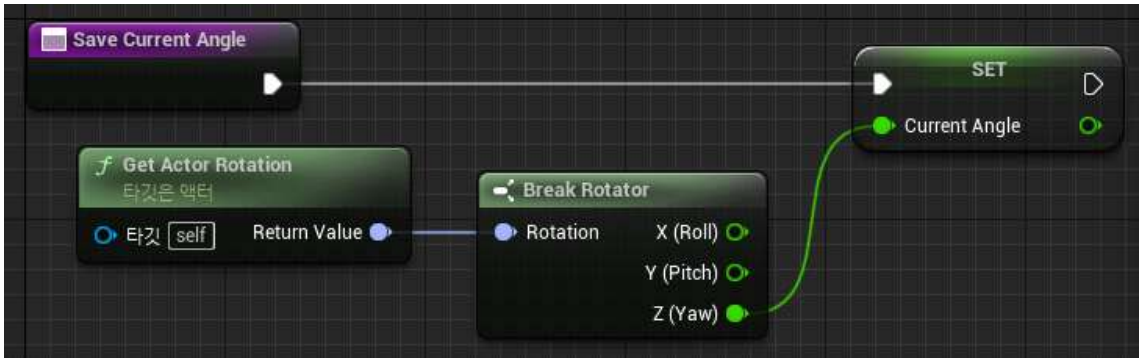
14. 그다음, 또다른 변수를 추가하자.

BP_MyCube 블루프린트 에디터의 **내 블루프린트** 탭에서 **변수** 항목의 오른쪽에 있는 + 아이콘을 클릭하자. 변수 이름을 클릭하여 **CurrentAngle**로 수정하자. 유형을 클릭하여 **플로트**로 수정하자.

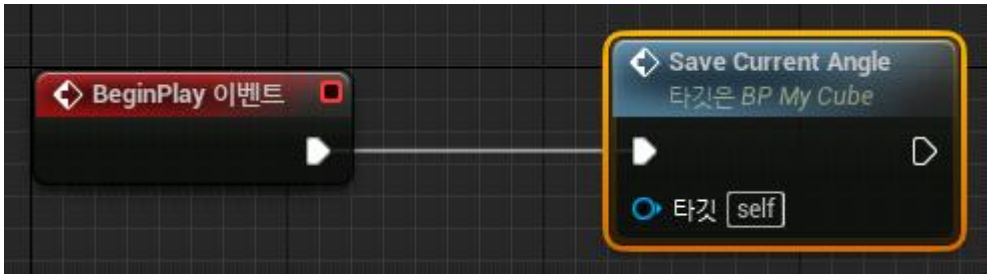


15. 이제부터, **BP_MyCube** 블루프린트에서 두 번째 함수를 추가해보자.

먼저, **내 블루프린트** 탭에서 **함수**의 오른쪽 +아이콘을 클릭하고 함수명은 **SaveCurrentAngle**로 지정하자. 그다음, 다음과 같이 함수 그래프를 만들자.

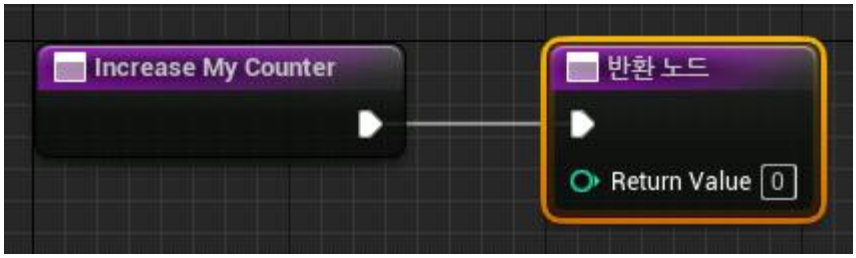


16. 이벤트 그래프 탭으로 가서 **BeginPlay 이벤트** 뒤에 **SetCurrentAngle** 함수 호출 노드를 추가하자.



17. 이제부터, **BP_MyCube** 블루프린트에서 세 번째 함수를 추가해보자.
이번 함수에서는 출력인자를 리턴하도록 해보자.
먼저, **내 블루프린트** 탭에서 **함수**의 오른쪽 +아이콘을 클릭하고 함수명은 **IncreaseMyCounter**로 지정하자.
그다음, **디테일** 탭에서 **출력** 영역의 오른쪽 +아이콘을 클릭하고 출력인자를 추가하자. 출력인자명은 **ReturnValue**로 수정하고 유형은 **인티저**로 수정하자. 이제 출력인자를 리턴할 수 있는 **반환 노드**가 생길 것이다.

18. 출력인자를 추가한 후에 함수 그래프를 보자. 함수 노드 외에 반환 노드가 추가되어 있을 것이다.

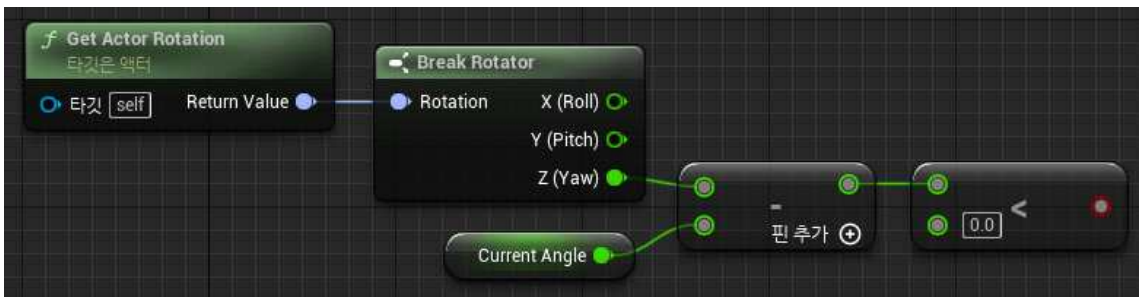


19. 함수 **IncreaseMyCounter**의 그래프를 완성해보자.
먼저, **MyCounter** 변수의 **Get** 노드를 배치하자.

그다음, **Get** 노드의 출력핀을 드래그하고 +를 검색하여 **추가** 노드를 배치하자. 그리고, + 노드의 두 번째 입력핀에 1을 지정하자.
그다음, **MyCounter** 변수의 **Set** 노드를 배치하자.
그다음, + 노드의 출력핀을 **Set** 노드의 입력핀으로 연결하자. 그다음, **Set** 노드의 출력핀을 반환 노드의 **ReturnValue** 입력핀에 연결하자.
그다음, 함수의 실행핀을 **Set** 노드와 **반환** 노드로 순차적으로 연결하자. 이제, 다음과 같이 그래프가 완성되었다.

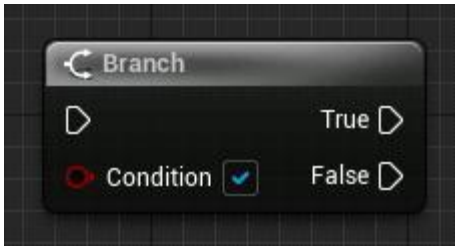
컴파일하자.

20. 이제부터, **BP_MyCube** 블루프린트에서 네 번째 함수를 추가해보자.
먼저, **내 블루프린트** 탭에서 **함수**의 오른쪽 + 아이콘을 클릭하고 함수명은 **UpdateMyCounter**로 지정하자. 이제 함수 그래프를 만들어보자.
이 함수에서는 액터가 한바퀴 회전하면 **MyCounter**를 1씩 증가하도록 구현해보자.
이를 위해서는 한바퀴 회전했는지를 판단할 수 있어야 한다.
최초의 액터의 자세의 각도는 **BeginPlay** 이벤트 노드에서 **SaveCurredntAngle** 함수를 호출하여 초기화해두었다. 이후에는 **Tick** 이벤트 노드가 **SpinOnce** 함수를 호출할 때마다 액터가 회전하면서 각도가 증가할 것이다.
액터는 스핀하므로 한바퀴 회전한 후에는 다시 시작할 것이다. 이 순간을 판단해보자. **GetActorRotation** 함수로 읽어오는 각도는 클램프된 각도로 [-180,180)의 범위값이다. 따라서 스핀하는 중에 179도 다음의 각도는 180도가 아닌 -180도이며 그다음의 각도는 -179도이다. 액터의 현재 자세에 대한 각도를 읽어서 이전에 저장된 각도와 비교해보고 만약 각도가 더 작아졌다면 한바퀴를 회전하고 리셋된 것에 해당할 것이다. 이 부분을 노드 네트워크로 구현하면 다음과 같다.



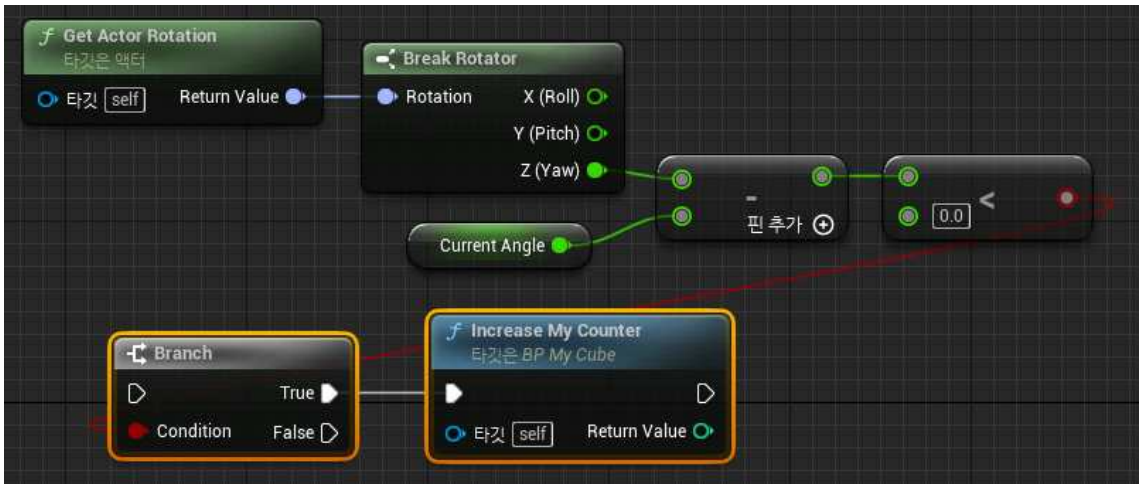
액터의 현재 자세의 각도와 **CurrentAngle**을 빼서 그 값이 음수이면 **true**가 리턴되고 그렇지 않으면 **false**가 리턴된다. 마지막의 < 노드는 조건식 노드로 주어진 값이 음수인지를 판단한다.

21. 함수 그래프를 계속 작성하자.
조건식이 참이면 **IncreaseMyCounter** 함수를 실행시켜서 **MyCounter**를 증가시키자.
조건식의 결과에 따라서 둘 중 하나를 실행하는 흐름 제어 노드가 **Branch** 노드이다. **Branch** 노드는 **Condition** 입력핀의 값이 참이면 **True** 출력 실행핀으로 펄스를 보내고 거짓이면 **False** 실행핀으로 펄스를 보낸다.



액션선택 창에서 **Branch** 또는 **If**로 검색하면 된다. **Branch** 노드는 자주 사용하는 노드여서 키보드 **B**를 누르고 **좌클릭**하면 바로 배치된다.

22. 조건식이 참인 경우에는 **IncreaseMyCounter** 함수를 실행시켜서 **MyCounter**를 증가시키자. **Branch** 노드의 **True** 실행핀을 **IncreaseMyCounter** 노드의 입력 실행핀에 연결하면 된다. 조건식이 거짓인 경우에는 증가시키지 않는다.



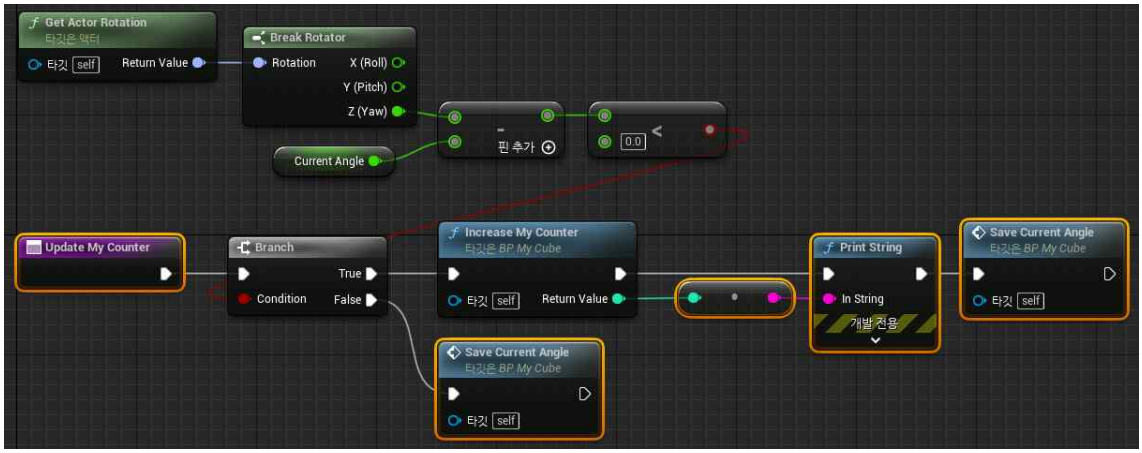
23. 조건식이 참이어서 **IncreaseMyCounter** 함수가 실행되는 경우에는 증가된 값을 **PrintString** 함수로 화면에 출력하자.

그다음, **CurrentAngle**을 회전 후의 액터 자세에 대한 회전 각도로 업데이트해주어야 한다.

이것은 조건식의 결과와 상관없이 항상 업데이트되도록 해주어야 한다. 따라서 **Branch** 노드의 **True** 실행핀뿐만 아니라 **False** 실행핀에도 연결하여 실행되도록 해주자.

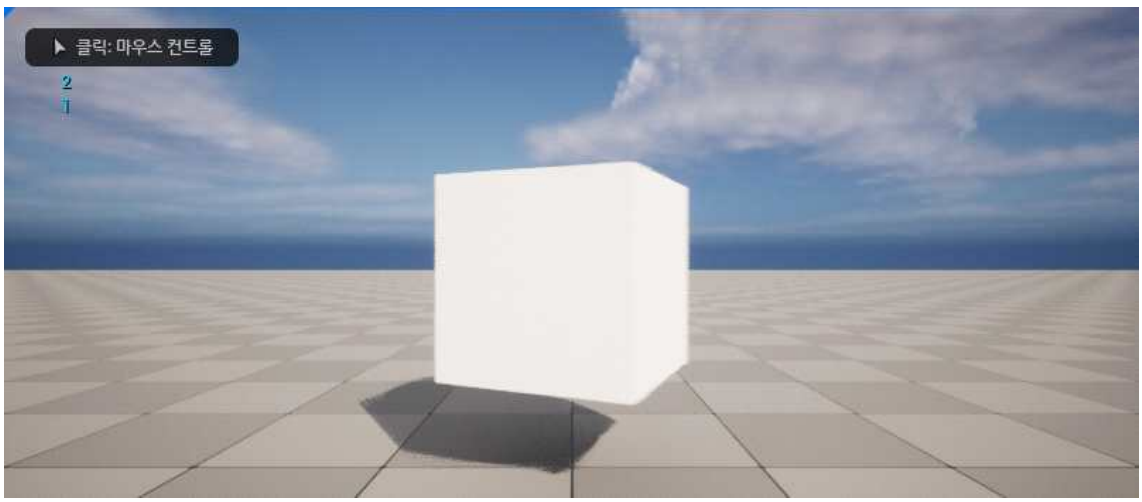
그다음, **Branch** 노드가 실행되도록 **UpdateMyCounter** 함수 노드의 실행핀을 **Branch** 노드의 실행핀에 연결하자.

이제, 다음과 같이 그래프가 완성되었다.



24. **Tick 이벤트** 노드 그래프에서 **SpinOnce** 호출 노드 뒤에 **UpdateMyCounter** 함수 호출 노드를 배치하자. 이제 큐브가 매 프레임 스핀할 때마다 한 바퀴 회전했는지를 체크하고 **MyCounter** 값을 업데이트 할 것이다.

25. 플레이해보자. 상자가 한바퀴 회전할 때마나 **MyCounter**가 화면에 출력될 것이다.



BP_MyCube 블루프린트 에디터에서 이벤트 그래프 탭을 다시 살펴보자. **BeginPlay 이벤트** 노드와 **Tick 이벤트** 노드의 그래프가 깔끔하고 간단하게 되어 있음을 알 수 있다. 내부 구현을 모두 함수에서 해두었기 때문이다. 이렇게 함수를 사용하면 더욱 큰 규모의 스크립트로 복잡하지 않게 확장해갈 수 있다.

지금까지, 함수에 대해서 학습하였다.

□