

# 09\_ 입력 처리

## <제 목 차 례>

09_ 입력 처리 .....	1
1. 개요 .....	2
2. 입력 매핑 설정 .....	4
3. 입력 이벤트 처리 .....	10

인천대학교 컴퓨터공학부 박종승  
무단전재배포금지

## 1. 개요

이 장에서는 키보드나 마우스와 같은 입력 장치로부터의 사용자 입력을 처리하는 방법에 대해서 학습한다.

사용자가 입력 하드웨어 장치를 조작하여 입력 행동을 하면 게임 엔진은 이를 이벤트 처리 방식으로 처리한다. 즉, 입력 하드웨어 장치를 상시 체크하는 루프를 돌면서 입력을 확인하는 폴링 처리 방식이 아니라 특정 사건이 발생하면 이를 통지하여 처리하도록 요청하는 이벤트 처리 방식이다. 사실상 대부분의 입력 시스템은 이벤트 처리 방식으로 동작한다.

이벤트 처리 방식에서의 처리 절차를 알아보자.

먼저, **입력 이벤트(input event)**를 정의해야 한다. 한 이벤트는 특정 입력 장치에서 특정 키가 눌러지는 사건으로 정의할 수 있다. 예를 들어, 키보드의 F 키가 눌러지는 사건을 ‘F 키’ 이벤트라는 한 이벤트로 정의할 수 있다.

그다음, 특정 입력 이벤트가 발생되었을 때에 처리되어야 할 작업을 정의해야 한다. 처리되어야 할 작업의 정의는 하나의 함수 형태로 구현하면 된다. 예를 들어, ‘F 키’ 이벤트가 발생되었을 때에는 총을 발사하는 작업을 진행할 수 있다. 이 경우에는 총을 발사하는 작업을 하나의 함수로 구현한다. 이 함수를 ‘fire’ 함수라고 하자. 이 함수는 바로 호출하는 용도의 함수가 아니라 함수를 특정 목록에 미리 등록해둔 후에 나중에 필요한 경우에 호출하는 함수이다. 이러한 함수를 **콜백 함수(callback function)**라고 한다.

그다음, 특정 입력 이벤트가 발생할 때에 특정 콜백 함수가 호출되도록 등록시켜주어야 한다. 예를 들어서 ‘F 키’ 이벤트에 대해서 ‘fire’ 함수를 등록시켜주어야 한다. 이 절차를 **바인드(bind)**라고 한다. 이제, 플레이 시에 해당 입력 이벤트가 발생하면 게임 엔진은 바인드된 콜백함수를 자동으로 호출해 줄 것이다. ‘F 키’ 이벤트가 발생하면 자동으로 ‘fire’ 함수가 호출된다. 우리가 직접 ‘fire’ 함수를 호출하는 일은 없다.

입력 매핑에 대해서 알아보자.

우리는 ‘F 키’ 이벤트를 정의하였고 이 이벤트가 발생하면 발사하는 ‘fire’ 함수를 호출하도록 하였다. 그런데 만약 마우스 왼쪽 버튼을 눌러도 발사되도록 기능을 추가하고자 한다면, 마우스의 왼쪽 버튼이 눌러지는 사건을 ‘마우스 좌클릭’ 이벤트라는 또다른 이벤트로 정의하고, 이 이벤트에 대해서 바인드해줘야 한다. 게임패드 X 버튼을 눌러도 발사되도록 기능을 추가하려면 또 유사한 작업을 반복해야 한다.

이렇게 하드웨어 수준에서의 구체적인 사건을 이벤트로 정의하는 것은 응용프로그램 개발자에게 불편한 일이다. 다양한 종류의 하드웨어가 사용될 수 있으며 또한 하드웨어 사양도 수시로 변할 수 있기 때문이다.

이에 대한 해결책으로 구체적인 사건을 이벤트로 정의하는 대신에, 의미 중심의 행위를 이벤트로 정의하면 된다. 예를 들어 발사를 위한 입력 행위를 ‘발사’ 이벤트로 정의하는 것이다. ‘발사’ 이벤트를 정의한 후에는 ‘발사’ 이벤트에 해당하는 구체적인 하드웨어 조작들을 나열해줘야 한다. 예를 들어, ‘발사’ 이벤트에는 키보드 F 키를 누르는 입력, 마우스 왼쪽 버튼을 누르는 입력, 게임패드 X 버튼을 누르는 입력을 나열한다. 이렇게 입력 이벤트와 특정 입력 행위를 연결하는 절차를 **입력 매핑(input mapping)**이라고 한다.

입력 매핑에서는 다양한 장치로부터의 이질적인 입력 데이터 포맷을 통일된 단일 형태의 입력 데이터 포맷으로 바꾸어서 제공해주어야 한다. 예를 들어, 키보드 F 키를 누를 때에 키보드 장치가 제공하는 입력 데이터 포맷과 마우스 왼쪽 버튼을 누를 때에 마우스 장치가 제공하는 입력 데이터 포맷은 서로 다를 것이다. 그러나 입력 매핑 후의 ‘발사’ 이벤트가 표현하는 입력 데이터 포맷은 단일 포맷이어야 한다. 이러한 개념은 입력 매핑으로 하드웨어 입력 장치와 게임 응용프로그램을 분리시켜서 게임 응용프로그램이 장치 독립적이 되도록 하는 역할을 한다.

이러한 입력 매핑 방식은 여러 장점이 있다. 먼저, 게임 응용프로그램은 다양한 입력 장치의 세부적인 특성과 무관하게 알고리즘에만 집중하여 하나의 버전으로만 구현할 수 있게 한다. 또한, 응용 프로그램은 하드웨어와 독립적인 특징이 있으므로 특정 하드웨어 사양에 포팅하기가 쉬워진다. 또한, 게임의 단축키 변경이나 입력 방법 변경을 쉽게 대처할 수 있다. 또한, 키보드 장치의 화살표 키를 조이스틱 기능으로 사용하는 등의 융통성있는 입력 방법 구현이 가능하다.

입력 이벤트의 분류에 대해서 알아보자.

언리얼에서는 입력의 종류를 두 가지로 구분하였다. 속성이 근본적으로 다른 두 가지 종류의 입력에 대해서 각각 다른 방식의 입력 데이터 표현 방식을 도입하였다.

두 입력의 종류는 액션 입력과 축 입력이다. 두 입력에 대해서 입력 매핑도 액션 매핑(action mapping)과 축 매핑(axis mapping)으로 구분하여 제공한다.

**액션 입력(action input)**은 입력값이 불리언 값으로 표현되는 입력이다. 키보드나 마우스나 게임 패드 등의 입력 장치에서 누른 상태(Pressed)와 누르지 않은 상태(Released)의 두 가지의 상태로만 표현되는 입력이다.

**축 입력(axis input)**은 입력값이 범위값으로 표현되는 입력이다. 조이스틱이나 마우스와 같은 입력 장치는 움직인 크기가 입력값에 영향을 미치는 장치이고, 이런 장치의 입력이 바로 축 입력에 적당하다. 예를 들어 조이스틱을 그대로 두면 0에 해당하고, 왼쪽으로 완전히 기울이면 -1에 해당하고, 오른쪽으로 완전히 기울이면 +1에 해당하고, 왼쪽으로 절반만 기울이면 -0.5에 해당하는 식이다.

한편, 액션 입력과 축 입력은 장치에 따라 결정되는 것은 아니다. 입력 데이터를 어떻게 해석할 것인지에 따라서 달라진다. 예를 들어서 키보드의 화살표 키를 조이스틱과 같이 동작하도록 할 수 있다. 왼쪽/오른쪽 화살표 키는 조이스틱을 왼쪽/오른쪽으로 완전히 기울인 것에 해당하도록 처리할 것이라면 키보드의 화살표 키는 축 입력으로 처리하면 된다.

키보드 왼쪽 화살표 키 입력으로부터의 입력값은 0과 1의 제한적인 값만 입력될 것인데, 이것을 누름과 누르지 않음으로 해석할 것이라면 액션 입력으로 처리해야 하고, 왼쪽으로 완전히 기울임과 전혀 기울이지 않음으로 해석할 것이라면 축 입력으로 처리하면 된다.

## 2. 입력 매핑 설정

이 절에서는 입력 매핑의 설정에 대해서 학습한다.

먼저, 축 매핑에 대해서 학습한다.

대부분의 게임에서 가장 필요한 입력 처리는 키보드, 마우스, 게임패드 등의 입력 장치를 통해서 캐릭터를 움직이도록 구현하는 것이다. 캐릭터의 움직임 제어는 키를 누르고 있는 동안에 계속 캐릭터를 움직이도록 구현하는 것이 일반적이다. 따라서 이런 경우는 축 매핑으로 구현해야 한다. 그다음, 액션 매핑에 대해서 학습한다.

축 매핑에서는 누르고 있는 동안에 계속 입력 이벤트가 발생하지만, 액션 매핑에서는 누르는 시간과 무관하게 누르는 동작에 대해서 단 한 번의 입력 이벤트만 발생한다. 점프, 아이템 사용, 발사 등의 경우에 액션 매핑이 사용된다.

이제부터 캐릭터를 움직이는 축 매핑과 점프 등을 수행하는 액션 매핑을 구현해보자.

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

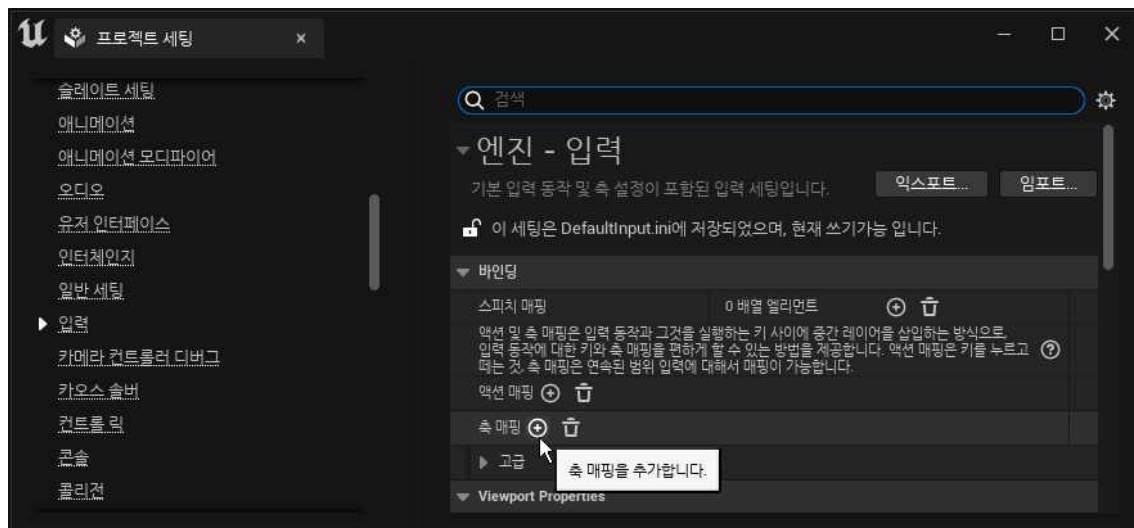
그리고, **프로젝트 세팅** 창에서 **맵&모드** 탭에서의 **에디터 시작 맵** 속성값을 **MyMap**으로 수정하자.

**2.** 이제부터, 입력을 위한 작업을 시작해보자.

먼저, 축 매핑을 설정해보자.

우선, **편집** » **프로젝트 세팅**... 메뉴를 클릭하여 **프로젝트 세팅** 창을 열자. 왼쪽 탭에서 **엔진** 카테고리 아래의 **입력**을 클릭하자. 이제 오른쪽 탭에서 우리의 프로젝트의 입력 세팅을 진행하면 된다.

**3.** 플레이어가 캐릭터를 전후좌우로 움직일 수 있도록 입력을 세팅해보자. 누르고 있는 동안 계속 움직이도록 해야 하므로 **축 매핑**을 설정해야 한다.

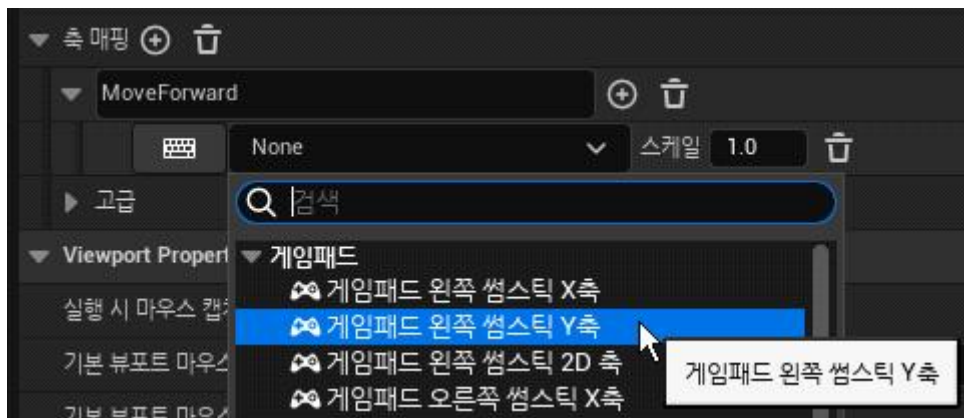


먼저 앞으로 전진 이동하는 축 매핑을 추가하자. 오른쪽 탭에서 **축 매핑**의 ‘+’ 버튼을 클릭하여 새로운 축 매핑을 추가하면 된다. 축 매핑이 추가되면, **축 매핑** 왼쪽에 목록을 펼칠 수 있도록 ‘▶’ 아이콘이 표시된다. 이것을 클릭하여 펼치자. 디폴트 이름이 **새 축 매핑\_0**로 되어 있을 것이다. 이것을 **MoveForward**로 수정하자.

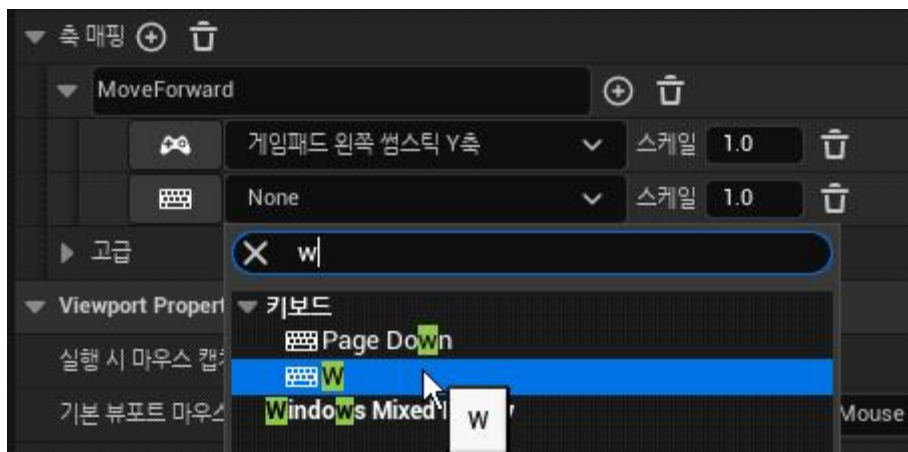
<참고> 만약 ‘+’ 버튼을 클릭하여 추가하는 도중에 잘못 추가된 내용이 있다면, ‘+’ 버튼의 바로 오른쪽에 있는 휴지통 아이콘을 클릭해서 제거하고 다시 추가하면 된다.

**4.** 하나의 축 매핑 아래에는 게임패드, 키보드, 마우스 등의 여러 입력을 추가할 수 있다. 새 축 매핑이 생성될 때에 디폴트로 키보드 입력이 하나 추가되어 있을 것이다.

이제, 게임 패드를 사용하여 전진하도록 지정해보자. 키보드 아이콘의 오른쪽의 **None** 입력 상자를 클릭하자. 입력을 선택할 수 있는 드롭다운 메뉴가 뜰 것이다. 드롭다운 메뉴에서는 게임패드, 키보드, 마우스 등의 다양한 입력 장치에 대해서 각 장치의 모든 입력 이벤트가 나열된다. 여기서 **게임패드**의 **게임패드 왼쪽 썸스틱 Y축**을 선택하자.

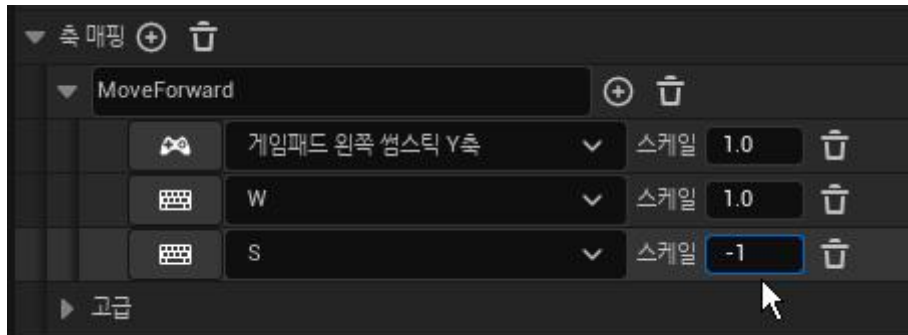


**5.** 게임패드뿐만 아니라 키보드로도 전진할 수 있도록 하자. 이를 위해서, **MoveForward**의 오른쪽 ‘+’ 아이콘을 클릭하여 입력을 하나 더 추가하자. 추가된 입력 항목의 **None** 입력 상자를 클릭하고, 드롭다운 메뉴에서 **키보드**를 펼치고 그 아래에서 **W**를 선택하자. 검색 상자에서 **w**를 입력하면 더 쉽게 찾을 수 있다.



**6.** 이제, 후진 이동을 생각해보자. 뒤로 후진 이동은 새 축 매핑을 만들 필요 없이 기존의 **MoveForward**를 사용해서 구현할 수 있다. **MoveForward**의 오른쪽 ‘+’ 아이콘을 클릭하여 입력을 하나 더 추가하

자. 추가된 입력 항목의 **None** 입력 상자를 클릭하고, 드롭다운 메뉴에서 **키보드**의 **S**를 선택하자. 그리고 오른쪽의 **Scale** 값을 1.0에서 -1.0로 수정하자.



<참고> 속성 **Scale**에 대해서 알아보자. **Scale**은 축 매핑에만 있는 특별한 속성이다. 마우스나 조이스틱과 같은 아날로그 스타일의 입력값은 정해진 범위 내의 값이 입력된다. 조작하지 않는 상태인 중립 상태가 0의 값에 해당하고 좌우 또는 상하로 움직이면 움직인 정도에 비례해서 양수 및 음수값이 발생한다. 입력 장치의 경우 X축 값은 좌우 방향을 표현하며, 오른쪽이 양수 방향에 해당하고 왼쪽이 음수 방향에 해당한다. Y축은 상하 방향을 표현하며, 위쪽이 양수 방향이 해당하고 아래쪽이 음수 방향에 해당한다.

최종 입력값은 입력 장치로부터의 입력값에 **Scale** 값이 곱해져서 결정된다. **Scale**의 디폴트값인 1이 사용되면 입력값이 그대로 최종값으로 결정된다. **Scale** 값을 0.5로 하면 입력 감도를 50%로 낮추게 된다. **Scale** 값을 -1로 바꾸면 방향을 역전시키는 효과를 발휘하게 된다.

위의 우리의 축 매핑 설정을 보자. **게임패드 왼쪽 썸스틱 Y축**의 경우에는, 썸스틱을 위로 기울이는 경우에는 양수값이 발생되고 아래로 기울이는 경우에는 음수값이 발생된다. **Scale**은 1로 그대로 두어서 감도는 그대로 유지된다.

<참고> 축 매핑으로 사용되는 입력 장치로부터의 전달되는 입력 값은 조이스틱이나 마우스와 같은 아날로그 스타일의 입력값을 가정하고 있지만 키보드나 버튼과 같은 디지털 스타일의 입력값일 수도 있다. 키보드나 버튼과 같은 디지털 스타일의 입력값은 눌려진 상태인지 또는 눌려지지 않은 상태인지에 따라서 1 또는 0의 값이 사용된다. 따라서 디지털 스타일의 경우에는 두 개의 키나 버튼을 사용해서 하나는 양수 방향에 배정하고 다른 하나는 음수 방향에 배정한다.

위의 우리의 축 매핑 설정을 보자. 키보드 **W**의 경우에는 **Scale**을 1로 두었고 **S**의 경우에는 **Scale**을 -1로 두었다. 즉, **W**가 눌러지면 최종 입력값이 1이 되어 전진을 표현하고 **S**가 눌러지면 -1이 되어 후진을 표현하게 된다.

<참고> 우리는 전진 이동과 후진 이동을 하나의 축 매핑으로 구현하였다. 그러나, 개발자의 선호에 따라서 각각의 축 매핑으로 구현할 수도 있다. 각각의 축 매핑으로 구현할 경우에는, 입력 값이 양수 범위로만 생각하고 **Scale**도 1로 그대로 둔다. 입력 선택에서는, 수평 이동의 경우에는 접미사 **X축** 대신 접미사 **우**와 **좌**가 붙은 입력을 사용하고 수직 이동의 경우에는 접미사 **Y축** 대신 접미사 **상**과 **하**가 붙은 입력을 사용하면 된다. 위의 **MoveForward**의 경우를 생각해보자. 축 매핑을 **MoveForward**와 **MoveBackward**의 두 개로 두자. **MoveForward**에는 게임패드 **게임패드 왼쪽 썸스틱 상**과 키보드 **W**를 입력으로 선택한다. **MoveBackward**에는 게임패드 **게임패드 왼쪽 썸스틱 하**와 키보드 **S**를 입력으로 선택한다. **Scale**은 모두 1로 그대로 둔다.

좌우 이동의 경우에도, **MoveLeft**와 **MoveRight**의 두 개로 두자. **MoveLeft**에는 게임패드 **게임패드 왼쪽 썸스틱 좌**와 키보드 **A**를 입력으로 선택한다. **MoveRight**에는 게임패드 **게임패드 왼쪽 썸스틱 우**와 키보드 **D**를 입력으로 선택한다. **Scale**은 모두 1로 그대로 둔다.

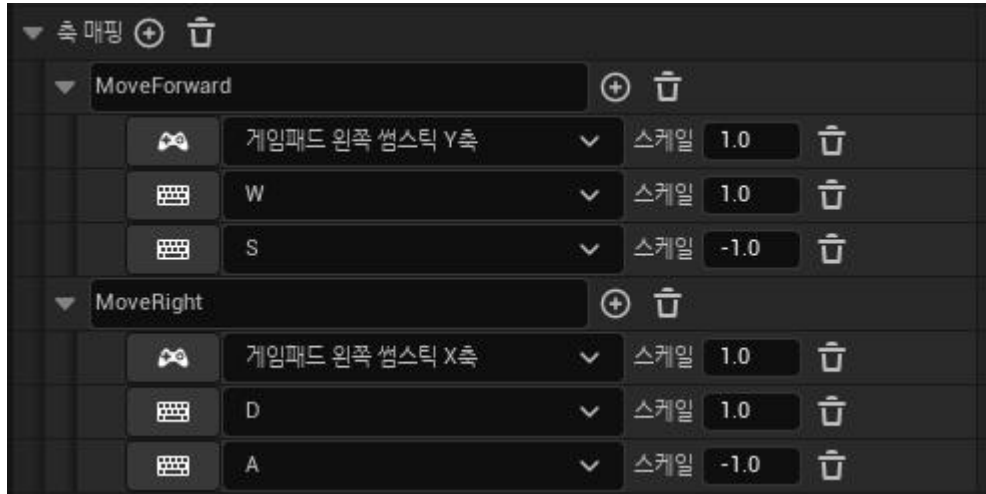
한편, 양수와 음수를 함께 고려하여 축 매핑의 수를 줄이는 것이 더 간결한 코드로 연결될 수 있다. 따라서 우리는 앞으로 음수 방향을 별도의 축 매핑으로 만들지 않을 것이다.

<참고> 입력에 대해 보다 구체적인 내용은 다음의 링크를 참조하자.

<https://docs.unrealengine.com/input-in-unreal-engine/>

**7.** 이제 좌우로 이동하는 축 매핑을 추가하자. 오른쪽 탭에서 **축 매핑**의 ‘+’ 버튼을 클릭하여 새로운 축 매핑을 추가하면 된다. 새 축 매핑의 이름은 **MoveRight**로 수정하자.

**MoveRight**에서도 **MoveForward**에서와 같은 방식으로 진행하면 된다. **게임패드**의 **게임패드 왼쪽 썸스틱 X축**을 추가하고, **키보드**의 **D**를 추가하고, **키보드**의 **A**를 추가하자. 또한, **키보드**의 **A**는 **Scale**을 -1로 수정하자.



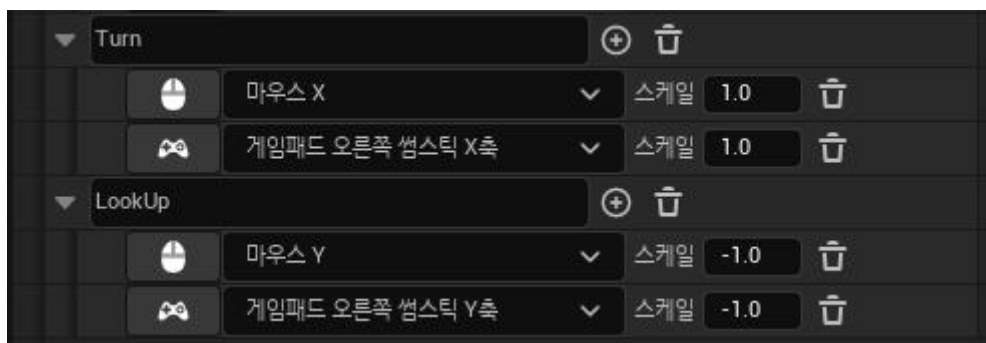
**8.** 계속해서 카메라 회전을 위한 축 매핑 **Turn**과 **LookUp**을 추가하자.

오른쪽 탭에서 **축 매핑**의 ‘+’ 버튼을 클릭하여 새로운 축 매핑을 추가하면 된다. 새 축 매핑의 이름은 **Turn**으로 수정하자.

**마우스**의 **마우스 X**를 추가하고, **게임패드**의 **게임패드 오른쪽 썸스틱 X축**을 추가하자.

다음으로, 다시 **축 매핑**의 ‘+’ 버튼을 클릭하여 새로운 축 매핑을 추가하자. 새 축 매핑의 이름은 **LookUp**으로 수정하자.

**마우스**의 **마우스 Y**를 추가하고, **게임패드**의 **게임패드 오른쪽 썸스틱 Y축**을 추가하자. 그리고 두 입력 모두에 대해서 **Scale**은 -1로 수정하자.



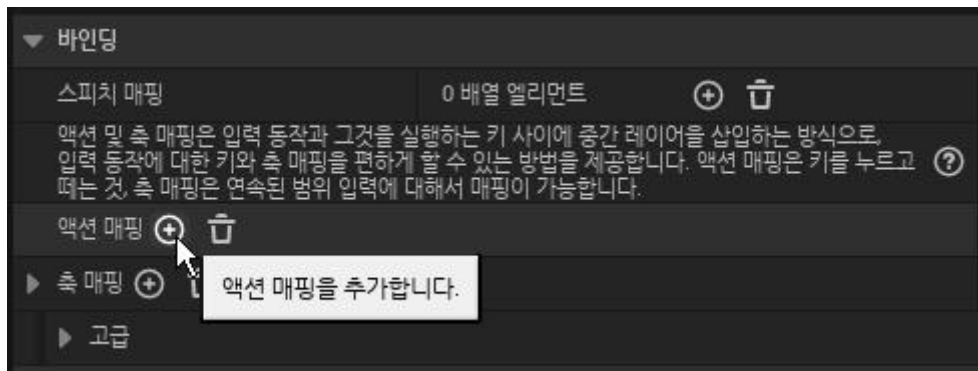
여기서, **LookUp**은 카메라 피치(pitch) 회전에 해당하고 **Turn**은 카메라 요(yaw) 회전에 해당한다. 이 두 축 매핑에서 마우스 움직임으로 캐릭터를 회전시킬 수 있도록 할 것이다. 이제 모든 축 매핑 지정을 완료하였다.

**9.** 다음으로 액션 매핑을 설정해보자.

**프로젝트 세팅** 창에서 **엔진** » **입력** 설정의 오른쪽 탭에서 계속하자.

**바인딩** 카테고리 아래에 있는 **액션 매핑**의 오른쪽에 있는 ‘+’ 아이콘을 클릭하자.

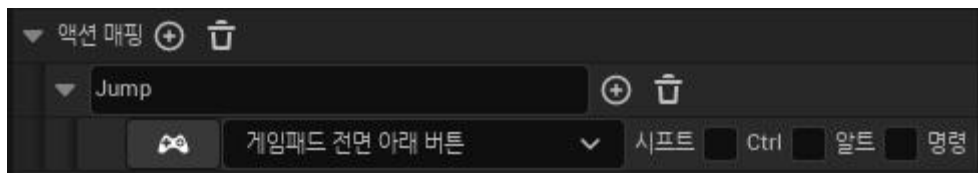




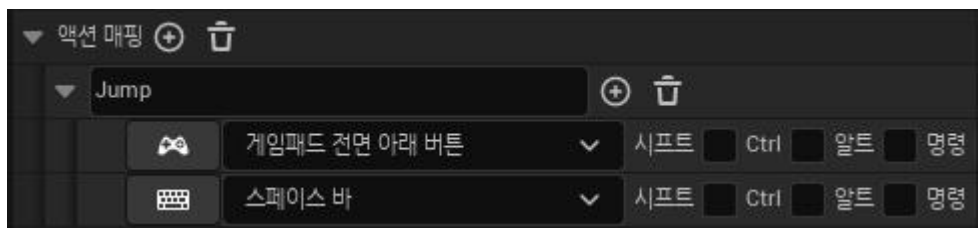
새 액션 매핑이 추가될 것이다.

새 액션 매핑이 보이지 않는다면, **액션 매핑** 왼쪽에 목록을 펼칠 수 있도록 '▶' 아이콘을 클릭하여 펼치자. 디폴트 이름이 **새 액션 매핑\_0**로 되어 있을 것이다. 이것을 **Jump**로 수정하자.

**10.** 이제 점프를 위한 게임패드 입력을 추가해보자. **Jump** 아래에 디폴트로 키보드 입력이 하나 추가되어 있을 것이다. 키보드 아이콘의 오른쪽 **None** 입력 상자를 클릭하고, 드롭다운 메뉴에서 **게임패드 전면 아래 버튼**을 선택하자.



**11.** **Jump**의 오른쪽의 '+' 아이콘을 클릭하고 입력을 추가하자. **키보드**의 **스페이스 바**를 선택하자.



<참고> 오른쪽의 **Shift, Ctrl, Alt, Cmd**에 체크하면, 그 키도 함께 눌러야 입력 이벤트가 발생된다. 그러나 이러한 복합 키 방식은 입력을 어렵게 하므로, 가급적이면 사용하지 않는 것이 좋다.

**12.** 이제 모든 매핑 절차가 완료되었다. 전체적으로 다음과 같은 모습이 될 것이다.




## 엔진 - 입력

기본 입력 동작 및 축 설정이 포함된 입력 세팅입니다.

엑스포트...

임포트...


 이 세팅은 DefaultInput.ini에 저장되었으며, 현재 쓰기가 가능합니다.

### 바인딩

스피치 매핑

0 배열 엘리먼트

액션 및 축 매핑은 입력 동작과 그것을 실행하는 키 사이에 중간 레이어를 삽입하는 방식으로, 입력 동작에 대한 키와 축 매핑을 편하게 할 수 있는 방법을 제공합니다. 액션 매핑은 키를 누르고 떼는 것, 축 매핑은 연속된 범위 입력에 대해서 매핑이 가능합니다. 

### 액션 매핑

Jump



게임패드 전면 아래 버튼



시프트

Ctrl

알트

명령



스페이스 바



시프트

Ctrl

알트

명령

### 축 매핑

MoveForward



게임패드 왼쪽 스틱 Y축



스케일 1.0





W



스케일 1.0





S



스케일 -1.0



MoveRight



게임패드 왼쪽 스틱 X축



스케일 1.0





D



스케일 1.0





A



스케일 -1.0



Turn



마우스 X



스케일 1.0





게임패드 오른쪽 스틱 X축



스케일 1.0



LookUp



마우스 Y



스케일 -1.0





게임패드 오른쪽 스틱 Y축



스케일 -1.0



 고급

지금까지, 축 매핑과 액션 매핑에 대해서 학습하였다.

### 3. 입력 이벤트 처리

이 절에서는 입력 이벤트 처리를 위한 구현에 대해서 학습한다. 입력 이벤트를 처리하는 스크립트는 이벤트 그래프 탭에서 노드 네트워크의 형태로 구현해야 한다.

이제부터 입력 이벤트의 처리를 위한 이벤트 그래프 구현에 대해서 학습해보자.

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

**2.** 이전 프로젝트에서 생성해둔 블루프린트 클래스를 가져오자.

윈도우 파일 탐색기에서 이전의 게임 모드 학습에서 다루었던 **Pcustomgamemode** 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.

<참고> 우리는 쉽고 간편한 실습 진행을 위해서 위와 같이 파일 복사를 진행하자. 그러나, 윈도우 파일 탐색기에서 애셋 파일을 복사하는 것은 권장하는 것이 아니다. 여기에서는, 동일한 버전이고 또한 아직 레벨이 작성되지 않은 상태여서 문제가 생기지는 않을 것이다. 그러나 일반적인 환경이라면 오류가 발생할 수도 있으므로 공식적으로 권장하는 방법을 사용하자.

공식적으로 권장하는 이주 방법으로 진행하면 다음과 같다.

먼저, 이전 프로젝트인 **Pcustomgamemode** 프로젝트를 열자. 그다음, **콘텐츠 브라우저**에서 3개의 애셋 파일(**BP\_MyCharacter**, **BP\_MyGameMode**, **BP\_MyPlayerController**)을 선택하고 애셋 위에서 우클릭하고, **애셋 액션 » 이주**를 선택하자. 그리고, 새 프로젝트의 **Content** 폴더에 저장되도록 폴더를 지정하자. 이제, 이전 프로젝트 창은 종료하자. 이제 이전 프로젝트에서 작성한 블루프린트 클래스의 이주가 완료되었다.

**3.** 이전 프로젝트에서 설정한 입력 매핑을 가져오자.

우선, 새 프로젝트를 종료하자.

그다음, 이전의 프로젝트 **Pinputmapping** 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 프로젝트를 다시 로드하자.

이제, **프로젝트 세팅** 창을 살펴보자. 이전 프로젝트에서 작업하였던 입력 매핑이 그대로 명시되어 있을 것이다.

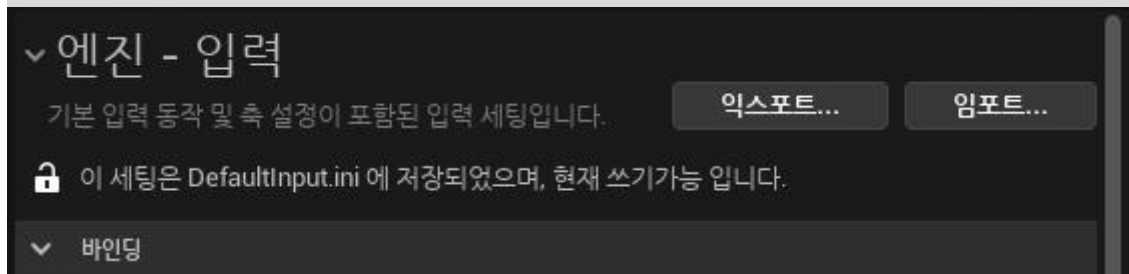
<참고> 프로젝트가 실행 중인 상태에서는 프로젝트 설정 파일을 복사해도 반영되지 않는다. 따라서 프로젝트를 종료한 후에 설정 파일을 복사하고 다시 프로젝트를 시작하자.

<참고> 우리는 쉽고 간편한 실습 진행을 위해서 위와 같이 파일 복사를 진행하자. 그러나, 윈도우 파일 탐색기에서 프로젝트 설정 파일을 복사하는 것은 권장하는 것이 아니다. 복잡한 환경이라면 오류가 발생할 수도 있으므로 공식적으로 권장하는 방법을 사용하자.

공식적으로 권장하는 이주 방법으로 진행하면 다음과 같다.

먼저, 이전의 프로젝트 **Pinputmapping** 프로젝트를 열자. 그다음, **프로젝트 세팅** 창을 열자. 그다음, 왼쪽 탭에

서 **엔진** » **입력**을 클릭하자.



그다음, 오른쪽 탭에서 상단의 **익스포트** 버튼을 클릭하자. 설정 파일은 각 설정 항목마다 별도로 있으므로 반드시 **엔진** » **입력**으로 이동한 후에 클릭해야 한다. 저장할 설정 파일의 위치와 파일명을 묻는 대화 상자가 뜨면 접근하기 쉬운 폴더 위치를 선택하고 파일명을 입력하고 **저장**을 클릭하자.

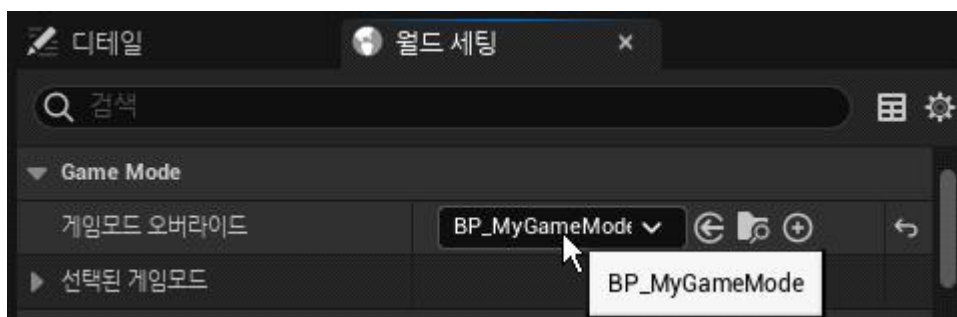
이제 이전 프로젝트 창은 닫아도 된다.

새 프로젝트 창을 열고 **프로젝트 세팅** 창을 열자. 그다음, 왼쪽 탭에서 **엔진** » **입력**을 클릭하자.

그다음, 오른쪽 탭에서 상단의 **임포트** 버튼을 클릭하자. 로드할 설정 파일의 위치를 묻는 대화 상자가 뜨면 조금 전에 저장한 설정 파일을 선택하고 **열기**를 클릭하자.

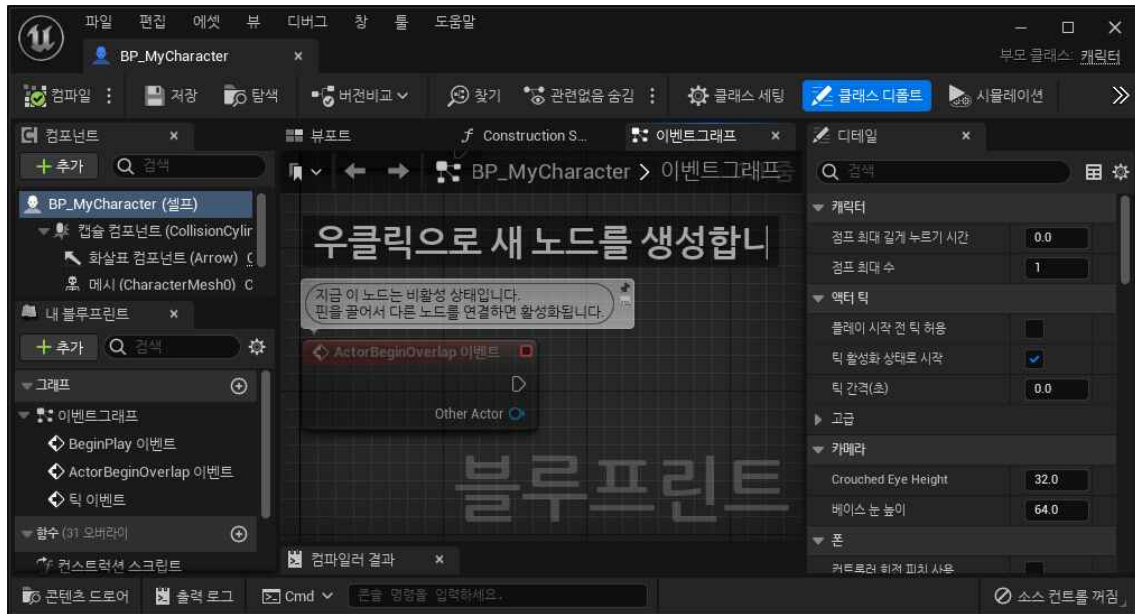
이전 프로젝트에서 **입력** 탭에서 명시했던 모든 설정이 그대로 동일하게 복사될 것이다.

**4.** 새 프로젝트의 **월드 세팅** 탭에서 **게임모드 오버라이드** 속성에 **BP\_MyGameMode**를 지정하자. 이제 이전 프로젝트에서의 모든 작업이 포함되도록 준비과정을 완료하였다.

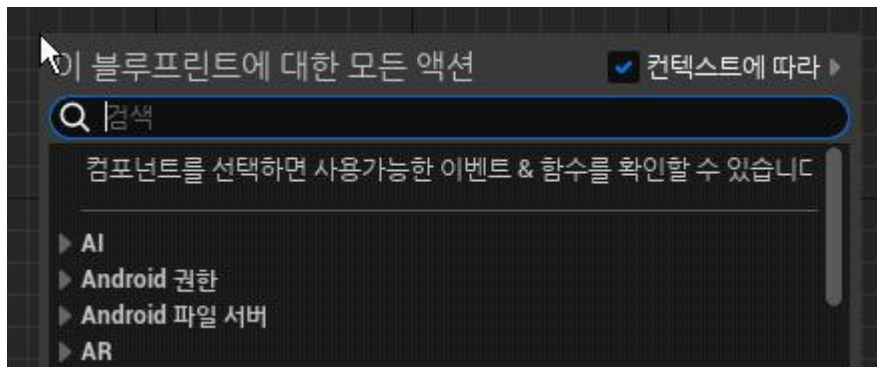


**5.** 이제부터, 입력 이벤트를 처리하도록 구현을 시작해보자.

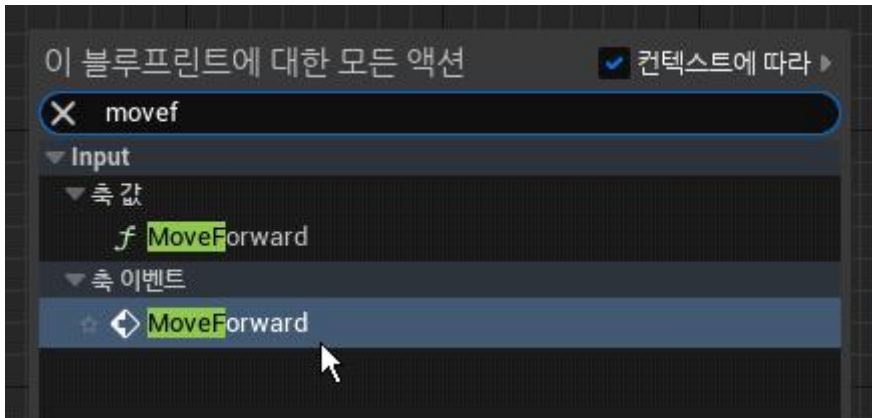
먼저, **BP\_MyCharacter**를 더블클릭하여 **블루프린트 에디터**를 열자.



6. 블루프린트 에디터 중간의 **이벤트 그래프** 탭을 클릭하여 이벤트 그래프 격자판으로 이동하자. **BeginPlay 이벤트**, **ActorBeginOverlap 이벤트**, **Tick 이벤트**의 세 이벤트 노드가 이미 배치되어 있을 것이다. 우리는 그 아래의 빈 공간으로 이동하자. 격자판 위에서 **우클릭+드래그**하면 격자판의 현재 위치를 이동할 수 있다.
- 빈 공간에서 우클릭하자. 배치 가능한 노드를 나열해주고 배치할 노드를 선택할 수 있도록 하는 **액션선택** 창이 뜬다.

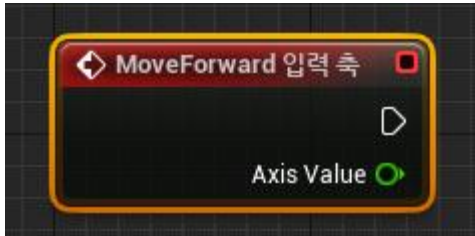


7. 검색상자에서 우리가 **축 매핑**에 작성했던 **MoveForward**를 검색해보자.



두 개가 검색될 것이다. 하나는 축 입력의 값을 리턴하는 함수 노드이고 다른 하나는 축 이벤트 노드로 서로 다른 노드이다. 명칭 앞에 아이콘 심볼이 표시되어 있어서 쉽게 구분할 수 있다. 우리는 축 이벤트 노드를 선택하자.

8. 다음과 같은 이벤트 노드가 배치될 것이다.

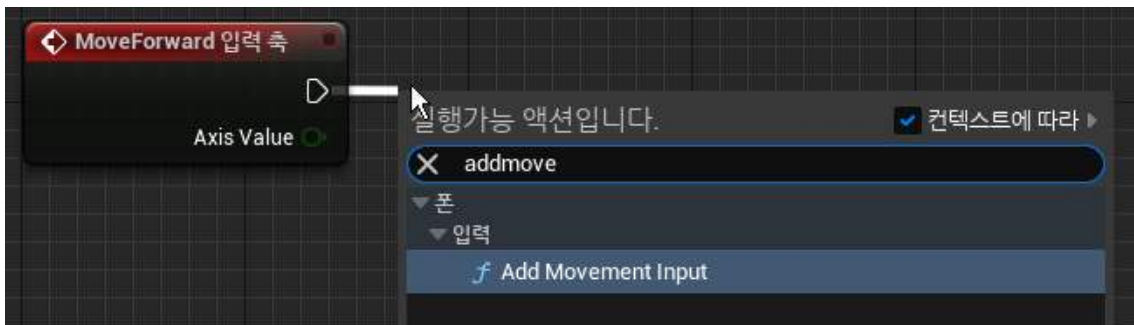


이 노드는 입력이 들어오게 되면 매 프레임당 한번씩 실행되도록 실행 신호가 발생된다. 이벤트 노드는 실행의 시작 지점이므로 입력 핀은 없다.

흰색 핀은 실행핀에 해당하고 흰색 실행핀 이외의 핀은 입력 인자값이나 출력 리턴값에 해당한다. 핀의 컬러는 타입에 따라서 결정된다. 위의 녹색은 실수형 타입에 해당하는 컬러이다.

9. 이제, 앞으로 전진할 수 있도록 움직이는 기능의 노드를 추가해보자.

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. 액션선택 창이 뜰 것이다. **AddMovementInput** 노드를 검색해서 배치하자.



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

**11. AddMovementInput** 노드는 액터를 이동시키는 노드이다.

이 노드의 입력 인자를 살펴보자.

컬러로 표시된 각 입력 핀은 모두 디폴트로 사용될 상수값을 가지고 있다. 입력 인자 이름 옆에 흰색 네모칸에 디폴트 상수값이 명시되어 있다. 흰색 네모칸을 클릭하여 디폴트 상수값을 바꿀 수 있다. 특별한 연결이 없는 경우에는 이 디폴트 상수값이 입력 인자값으로 사용된다.

첫 번째 인자인, **타겟**은 움직일 대상 폰을 의미하며 **self**로 되어 있다. 즉 현재의 블루프린트인 **BP\_MyCharacter**를 이동시킨다.

그다음, **WorldDirection**은 이동 방향이다. 우리는 (1,0,0)을 입력하여 X축으로 이동하도록 하자.

그다음, **ScaleValue**는 이동 량이다. **MoveForward 입력 축** 이벤트 노드의 출력핀 **AxisValue**는 -1에서 1까지의 실수값을 리턴한다. 우리는 이 값을 **ScaleValue**에 연결시키자.

그다음, 마지막 인자 **Force**는 불리인 타입의 인자이다. 폰이 없거나 하는 특별한 상황에서는 이동 입력이 무시되도록 처리되는데 그 경우에도 실행할 지에 대한 플래그이다. 네모칸이 비어있는 것은 **false**를 의미한다. 네모칸을 클릭하면 체크표시가 생기며 **true**를 의미하게 된다.





노드에서 왼쪽의 여러 입력 인자에 대한 입력핀이 있다. 입력핀이 다른 핀에 연결된 경우에는 해당 값이 전달되어 인자값으로 입력되며 입력핀 아이콘의 모양이 채워진 형태로 표시된다.

**12.** 이제, 컴파일하고 저장하자.

레벨 에디터에서 플레이해보자.

**W** 키와 **S** 키를 눌러보자. 전후진하는 것을 확인할 수 있을 것이다.

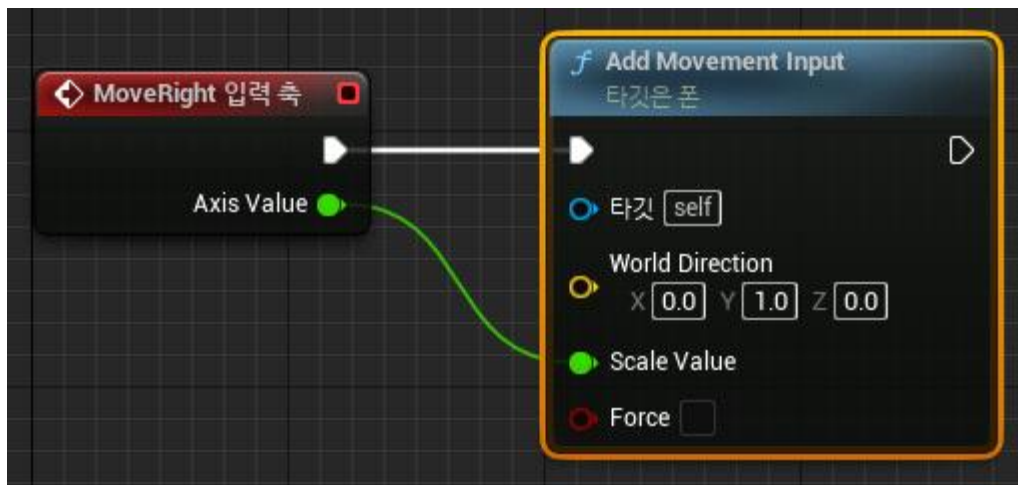
**13.** 이제, **MoveRight** 이벤트에 대해서도 진행해보자.

이전에서와 동일한 방법으로, 격자판의 빈 공간에서 우클릭하고, **액션선택** 창에서 **MoveRight** 축 이벤트 노드를 검색하여 배치하자.

**14.** 그다음, **AddMovementInput** 노드를 배치해야 한다. 이전과 동일하게 액션선택 창에서 검색하여 배치할 수도 있지만, 이번에는 이미 배치되어 있는 **AddMovementInput** 노드를 복사해서 사용해보자. **MoveForward**에 연결되어 있는 **AddMovementInput** 노드를 선택한 후에 우클릭하고 복제를 선택하자. 또는 **Ctrl+C**를 누르고 **Ctrl+V**를 눌러도 복제된다.

복제된 노드를 **MoveRight** 노드로 이동하고 출력 실행핀을 연결하자. 또한, **AxisValue** 핀도 **ScaleValue** 핀에 연결하자.

오른쪽 방향으로 이동해야 하므로, **WorldDirection** 입력핀의 값은 (0,1,0)으로 입력하자.



**15.** 이제, 컴파일하고 저장하자.

레벨 에디터에서 플레이해보자.

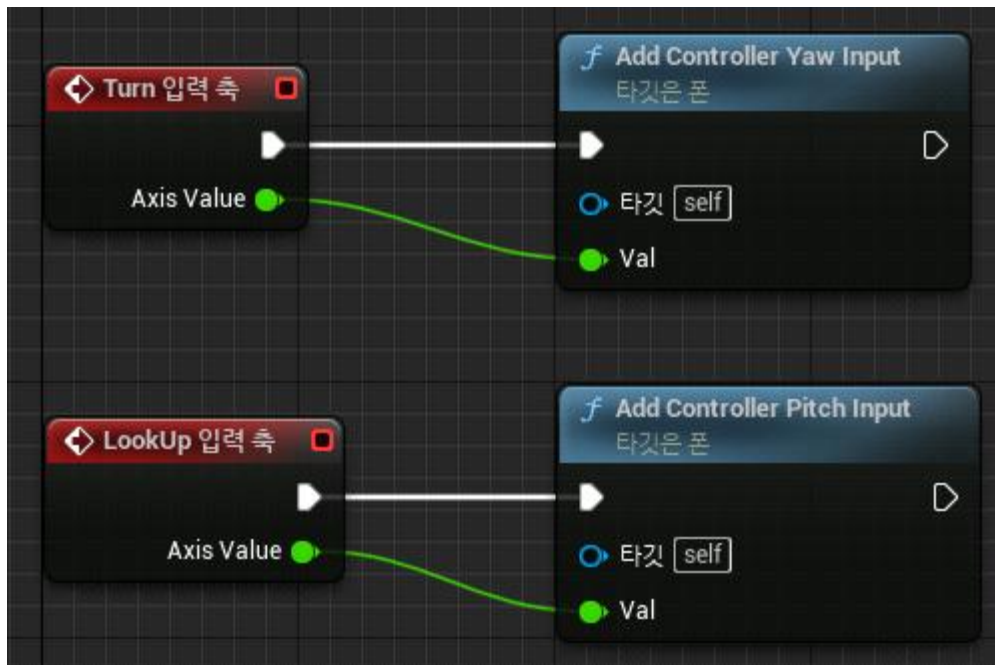
**D** 키와 **A** 키를 눌러보자. 좌우로 이동하는 것을 확인할 수 있을 것이다.

지금까지, 플레이어의 이동에 대해서 구현하였다.

**16.** 지금부터는, 카메라 회전을 구현해보자.

카메라 회전을 위한 **Turn** 축 매핑과 **LookUp** 축 매핑에 해당 처리를 구현하면 된다. 이전과 동일한 방식으로 다음의 이벤트 그래프를 구현하자.





Turn 축 매핑은 **마우스 X** 입력에 해당하고, 시야를 좌우로 돌리는 요(yaw) 회전에 해당한다. 요 회전은 **AddControllerYawInput** 함수를 호출하면 된다. 마우스 오른쪽이 마우스 좌표계로 +X에 해당한다. 양수 값은 월드에서 요 회전이 시계방향으로의 회전에 해당한다.

LookUp 축 매핑은 **마우스 Y** 입력에 해당하고, 시야를 위로 올려다보거나 아래로 내려다보는 피치(pitch) 회전에 해당한다. 피치 회전은 **AddControllerPitchInput** 함수를 호출하면 된다. 마우스 위쪽이 마우스 좌표계로 +Y에 해당한다. 양수 값은 월드에서 피치 회전이 아래로 향한다. 따라서 이전에 **축 매핑**에서 마우스 입력 값으로부터 **Scale**을 -1로 하여 반전된 값을 받도록 하였다. 이렇게 하면 마우스를 위로 올리는 경우가 위를 쳐다보는 것에 해당하게 된다.

**17.** 이제, 컴파일하고 저장하자.

레벨 에디터에서 플레이해보자.

마우스를 움직여서 카메라를 회전해보자. 카메라가 잘 회전될 것이다.

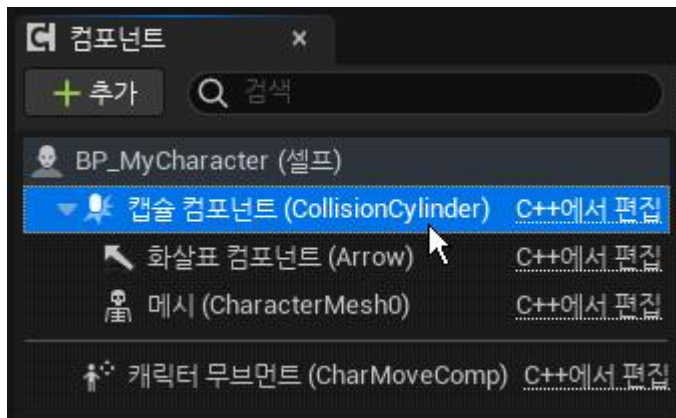
**18.** 다시 플레이해보자. 이번에는, 플레이어를 움직이면서 카메라도 함께 움직여보자.

이 경우에는 움직이는 방향이 잘못되어있음을 느낄 것이다.

**MoveForward**에 가 X축으로 되어 있고 **MoveRight** 이가 Y축으로 되어 있기 때문이다.

**씬 컴포넌트**는 자신이 향하는 방향에 대한 정보를 리턴하는 함수를 제공한다. 자신이 향하는 전방 방향 벡터를 **GetForwardVector** 함수로 제공하고 자신의 오른쪽 방향 벡터를 **GetRightVector** 함수로 제공한다. 우리를 이를 사용해서 X축 대신 캐릭터가 향하는 전방 벡터로 대체하고 Y축 대신 우향 벡터로 대체해보자.

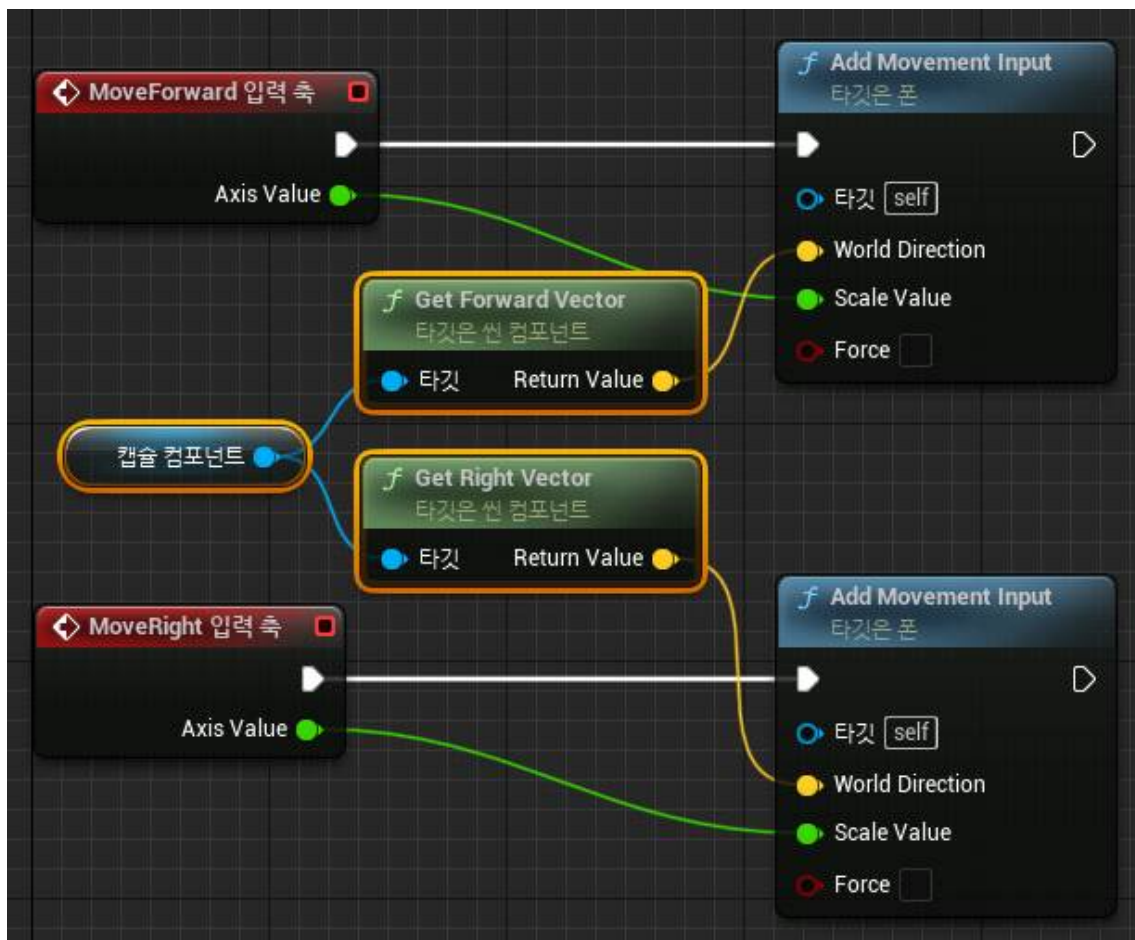
먼저 컴포넌트 탭에서 **CapsuleComponent**를 드래그해서 이벤트 그래프의 빈 곳에 드롭하자.



CapsuleComponent의 레퍼런스 노드가 생성될 것이다.

**19.** CapsuleComponent 레퍼런스 노드의 출력핀을 당겨서 GetForwardVector 노드를 배치하자. 그리고 리턴값을 MoveForward에 대한 WorldDirection 입력핀에 연결하자.

다시 한번, CapsuleComponent 레퍼런스 노드의 출력핀을 당겨서 GetRightVector 노드를 배치하자. 그리고 리턴값을 MoveRight에 대한 WorldDirection 입력핀에 연결하자. 다음과 같은 그래프가 될 것이다.



**20.** 이제, 컴파일하고 저장하자.

레벨 에디터에서 플레이해보자.

플레이어를 움직이면서 카메라도 함께 움직여보자. 전후방으로의 움직임과 좌우로의 움직임이 올바

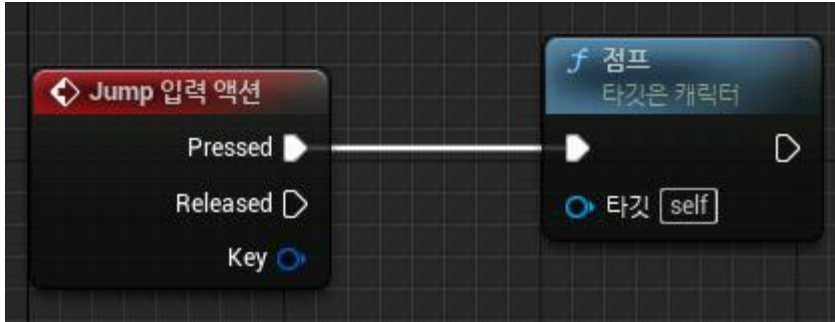
르게 동작할 것이다.

**21.** 이제, **Jump** 이벤트에 대해서도 진행해보자.

이전에서와 동일한 방법으로, 격자판의 빈 공간에서 우클릭하고, **액션선택** 창에서 **Input** 카테고리에 있는 **Jump** 액션 이벤트 노드를 검색하여 배치하자.

**22.** 그다음, 액션선택 창에서 **캐릭터** 카테고리에 있는 **Jump** 함수를 검색하여 배치하자.

**Jump** 액션 이벤트 노드의 출력 실행핀을 **Jump** 함수의 입력 실행핀에 연결하자.



위의 **Jump** 함수는 캐릭터 클래스에서 기본으로 제공하는 함수이다. 캐릭터 클래스는 사람의 걷거나 점프와 같은 움직임 기능을 제공한다.

**23.** 이제, 컴파일하고 저장하자.

레벨 에디터에서 플레이해보자.

**스페이스** 키를 눌러보자. 점프하는 것을 확인할 수 있을 것이다.

---

지금까지, 입력 이벤트 처리를 위한 구현에 대해서 학습하였다.

□