23_ 캐릭터 애니메이션 적용하기

<제목 차례>

인천대학교 컴퓨터공학부 박종승

1. 개요

이 장에서 애니메이션 기능을 가진 캐릭터를 만드는 방법에 대해서 학습한다.

먼저 스켈레탈 메시에 대해서 알아보자.
캐릭터의 애니메이션을 위해서는 스태틱 메시가 아닌 스켈레탈 메시를 사용한다. **스켈레탈 메시**(skeletal mesh)는 스태틱 메시 정보와 더불어 추가적으로 관절 정보를 가진다. 스켈레탈 메시에서 사용하는 관절 정보를 **스켈레톤**(skeleton)이라고 한다.
애니메이션은 프레임마다 관절을 조금씩 움직여서 구현한다. 애니메이션을 위한 관절 움직임 정보를 저장해둔 애셋을 **애니메이션 시퀀스**(animation sequence)라고 한다. 애니메이션 시퀀스에 저장된 관절 움직임을 재생하면 스켈레탈 메시가 움직이게 된다.
스켈레탈 메시와 스켈레톤은 별도의 애셋으로 분리되어 있다. FBX 파일 형태의 스켈레탈 메시를 임포트할 때에 기존의 스켈레톤을 사용할 수도 있고 자동으로 생성되도록 할 수도 있다. 또한 이후에 다른 스켈레톤으로 다시 할당할 수도 있다. 다시 할당하기 위해서는 **콘텐츠 브라우저**에서 스켈레탈 메시 애셋 위에서 우클릭하고 팝업메뉴에서 **스켈레톤 » 스켈레톤 할당**을 선택하면 된다.
또한, 스켈레탈 메시의 임포트 시에 메시의 제작에 사용된 DCC 툴에 따라서 좌표계 정의나 스케일에 차이가 있을 수 있다. 따라서 스켈레탈 메시의 임포트 시에 별도의 수정 작업이 필요할 수 있다.

스켈레탈 메시, 스켈레톤, 애니메이션 시퀀스의 관계에 대해서 알아보자.
스켈레탈 메시에는 단 하나의 스켈레톤만 할당할 수 있다. 그리고, 하나의 스켈레톤은 여러 스켈레탈 메시에서 공통으로 사용할 수 있다.
한편, 애니메이션 시퀀스는 특정 스켈레톤을 가정하고 제작한다. 따라서 특정 스켈레톤을 가정하여 제작한 애니메이션 시퀀스를 다른 스켈레톤에 적용할 수는 없다.
따라서 하나의 스켈레탈 메시에 대하여 최대한 많은 애니메이션 시퀀스를 만들어 사용하는 것이 좋다.

<참고> 그러나 현실에서는 캐릭터의 모양과 특성에 따라서 스켈레탈 메시나 또는 스켈레톤이 달라질 수 밖에 없다. 이런 경우에 애니메이션 시퀀스를 공유하는 방법으로 리타게팅(retargeting) 기법이 이다.
한 스켈레톤의 애니메이션 시퀀스를 다른 스켈레톤에 적용하는 방법에 대해서는 다음의 문서를 참조하자.
https://docs.unrealengine.com/using-retargeted-animations-in-unreal-engine/
동일한 스켈레톤을 사용하지만 메시의 모양이 다른 경우에 신체의 비율 조절 등을 수행하는 리타게팅에 대해서는 다음의 문서를 참조하자.
https://docs.unrealengine.com/ik-rig-animation-retargeting-in-unreal-engine/

이 장에서는 캐릭터 클래스를 전체적으로 제작해본다. 입력에 따른 움직임 처리와 현재의 움직임 상황을 판단하여 알맞은 애니메이션을 플레이하도록 한다.

<참고> 이 장에서의 예제에 대해서는 다음의 문서를 참조하자.
https://docs.unrealengine.com/setting-up-character-movement/

2. 캐릭터 제작 준비하기

이 절에서 입력에 의해 움직이고 움직임에 따라 애니메이션되는 캐릭터를 제작하는 방법에 대해서 학습한다.

캐릭터 블루프린트 클래스에 대해서 알아보자.
Character 파생 블루프린트 클래스에는 **CharacterMovement** 컴포넌트가 포함되어 있다. 이 컴포넌트는 캐릭터의 이족 보행 움직임을 제공한다. 여기에서 의미하는 움직임이란 애니메이션에서의 메시의 움직임을 의미하는 것이 아니라 월드 공간에서의 액터의 이동이나 회전을 의미한다.
CharacterMovement 컴포넌트는 걷기, 점프나 낙하하기, 수영하기의 움직임을 구현한다. 또한 네트워크 환경에서의 캐릭터 움직임에 대한 문제를 해결해준다.

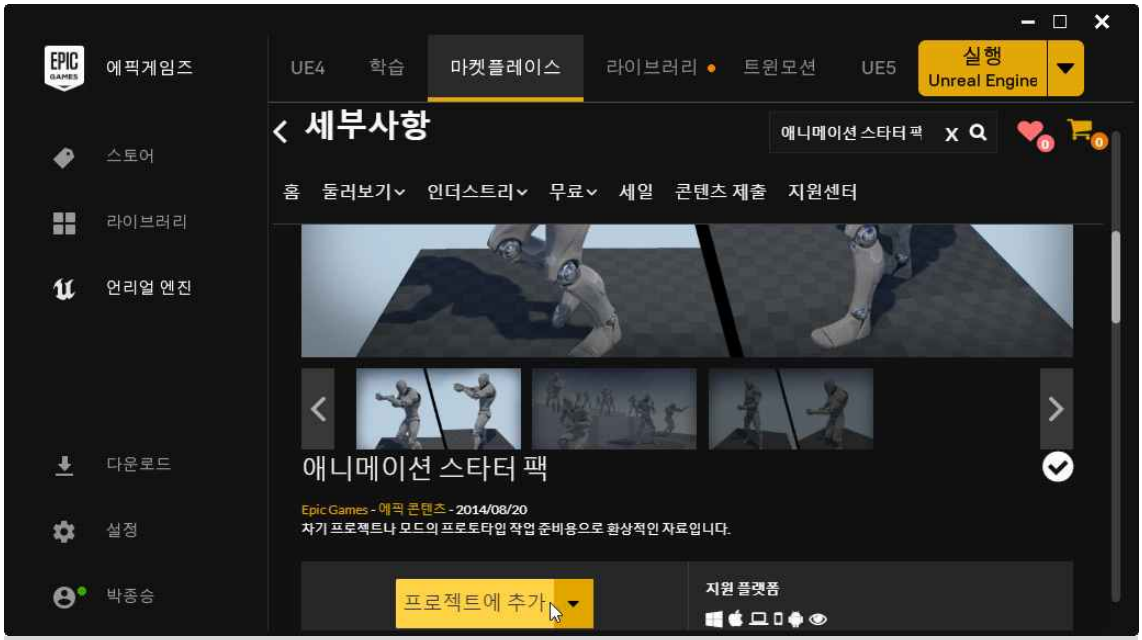
이제부터 예제를 통해서 학습해보자.

---

1. 새 프로젝트 Pcharprepare을 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 Pcharprepare로 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. 언리얼 엔진에서 제공하는 **애니메이션 스타터 팩**을 준비하자.
다운로드해둔 **애니메이션 스타터 팩** 폴더인 AnimStarterPack 폴더가 있다면 이 폴더를 새 프로젝트의 **Content** 폴더 아래로 복사하자.

<참고> 폴더를 복사하는 방법은 정식적인 방법이 아니다.
다운로드해둔 **애니메이션 스타터 팩** 폴더가 없거나 폴더 복사 시에 문제가 발생하는 경우에는 정식적인 방법으로 **애니메이션 스타터 팩**을 설치해야 한다. **애니메이션 스타터 팩**을 설치하는 정식 방법을 알아보자.
먼저, **Epic Games Launcher** 프로그램을 실행하자. **에픽게임즈 런처** 창이 뜨면, 왼쪽 탭에서 **얼리얼 엔진** 메뉴를 클릭하여 선택하자.
그다음, 오른쪽 탭에서 **마켓플레이스**를 선택하자.
그다음, **콘텐츠 검색** 입력상자에서 **애니메이션 스타터 팩**(Animation Starter Pack)을 검색하자.
Epic Games에서 출시한 **애니메이션 스타터 팩**이 보일 것이다. 그다음, 무료 구매 버튼을 클릭하면 자신의 라이브러리에 추가될 것이다.



그다음, **프로젝트에 추가** 버튼을 클릭하자. 그리고, 프로젝트 선택 창에서 추가할 프로젝트를 선택하고 **프로젝트에 추가** 버튼을 클릭하자. 이제, **Epic Games Launcher** 프로그램을 종료하자.
추가한 프로젝트의 **Content** 폴더 아래를 살펴보자. 폴더 아래에 AnimStarterPack 폴더가 생겼을 것이다.

<참고> **Content** 폴더 아래에 추가된 AnimStarterPack 폴더의 내부를 살펴보자. 이 폴더 아래에는 UE4_Mannequin 폴더가 있고 그 외에 67개의 파일이 있다. 이들 파일들 중에서 62개는 **애니메이션 시퀀스**이고 2개의 **블렌드 스페이스**(BS_CrouchWalk, BS_Jog)가 있다.
그 외에, **Character** 파생 블루프린트 클래스인 Ue4ASP_Character 블루프린트 클래스가 있다. 그리고 이 캐릭터 클래스가 사용하는 **애니메이션 블루프린트**인 UE4ASP_HeroTPP_AnimBlueprint가 있다.
그리고, 쇼케이스 레벨인 Showcase가 있다.
한편, UE4_Mannequin 폴더에는 스켈레탈 메시를 구성하기 위한 메시, 머티리얼, 텍스처가 각 Mesh, Materials, Textures 하위 폴더에 있다. 이 중에서 Mesh 폴더에는 **스켈레탈 메시**인 SK_Mannequin가 있다. 또한 같은 위치에 이 메시의 **스켈레톤**인 UE4_Mannequin_Skeleton와 **피직스 에셋**인 SK_Mannequin_PhysicsAsset가 있다.

<참고> 마켓플레이스에서 **애니메이션 스타터 팩**을 검색으로 찾기가 힘들면 아래의 링크를 클릭하고 찾을 수 있다.

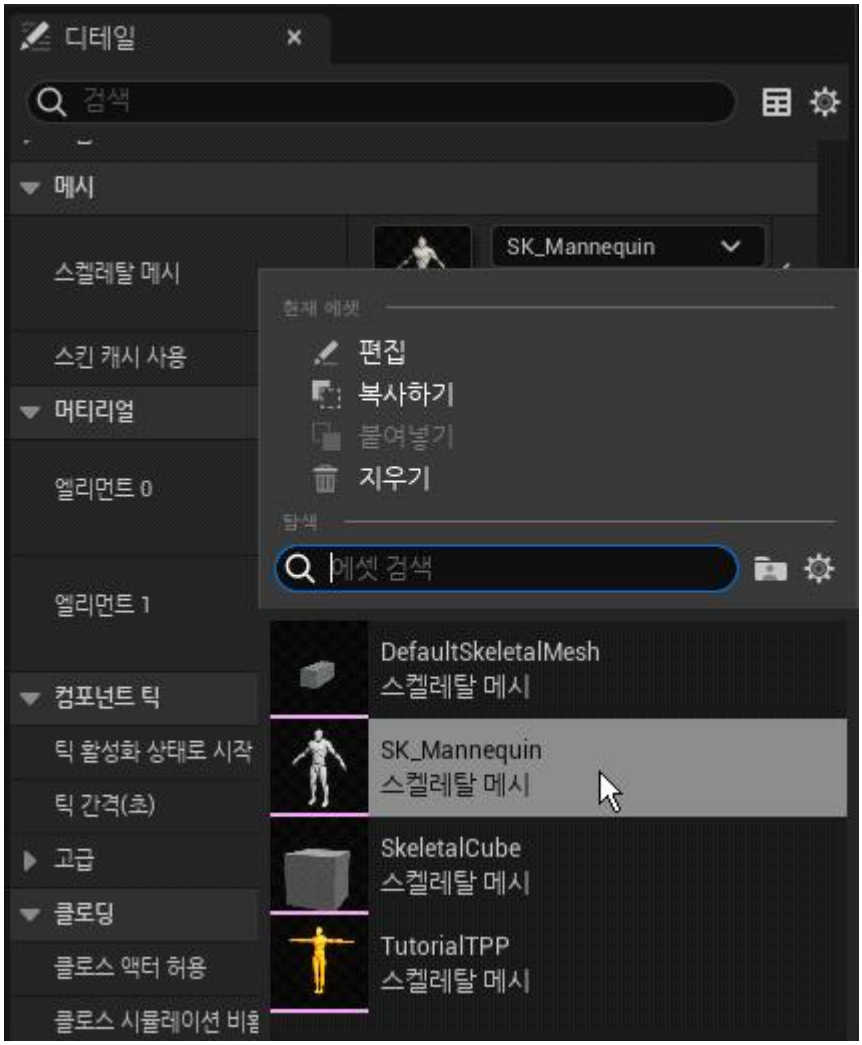
https://www.unrealengine.com/marketplace/product/animation-starter-pack

프로젝트에 **애니메이션 스타터 팩**을 추가하는 것이 어려우면 아래의 링크를 참조하자.

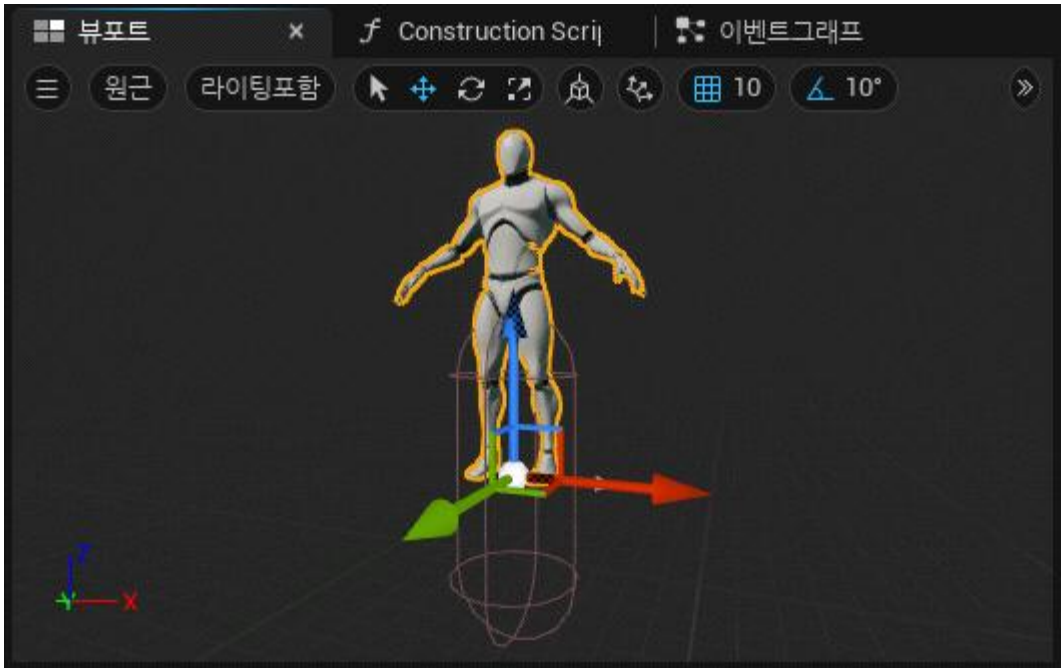
https://docs.unrealengine.com/assets-and-content-packs-in-unreal-engine/

3. 플레이어 캐릭터 클래스를 만들자.
콘텐츠 브라우저에서 **+추가**를 클릭하고 **블루프린트 클래스**를 선택하자. **부모 클래스 선택** 창에서 **캐릭터**(**Character**)를 선택하자. 생성된 클래스 이름을 **MyPlayerCharacter**로 하자.

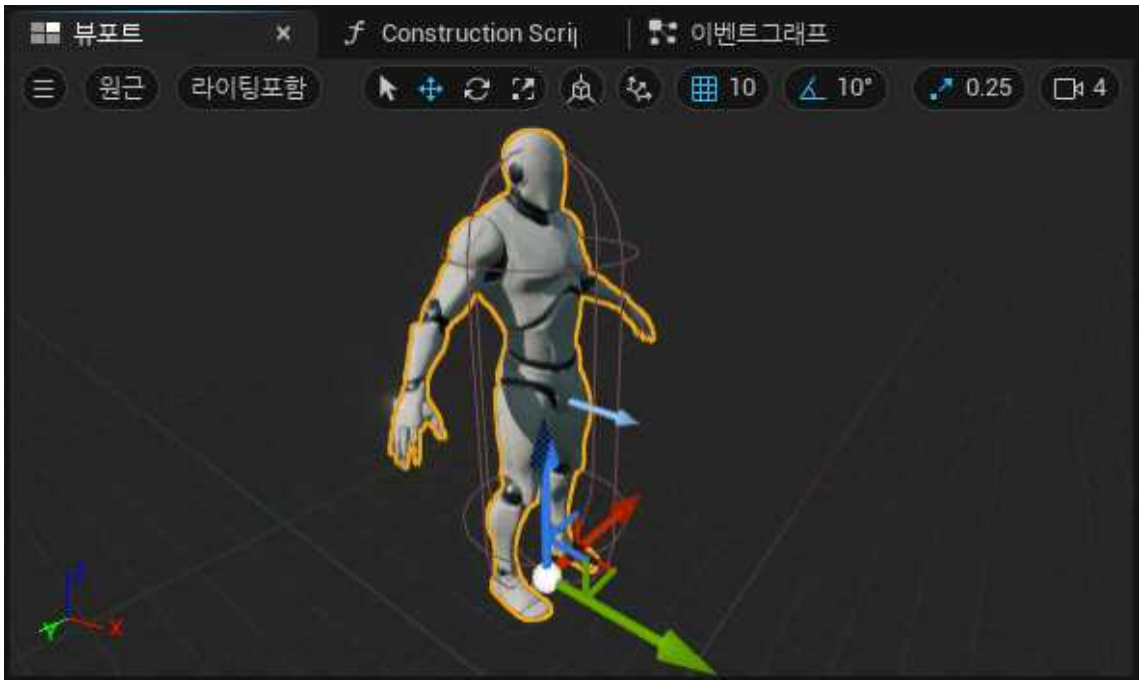
4. 그다음, **MyPlayerCharacter**를 더블클릭하여 블루프린트 에디터를 열자. **컴포넌트** 탭에서 스켈레탈 메시 컴포넌트인 **메시**(**Mesh**) 컴포넌트를 선택하자. 오른쪽 **디테일** 탭에서 **메시** 영역에 있는 **스켈레탈 메시**(**SkeletalMesh**) 속성을 찾자. 디폴트 값인 **없음**을 클릭하고 선택 목록에서 **SK_Mannequin**을 선택하자. **SK_Mannequin**는 우리가 설치한 AnimStarterPack 폴더 아래의 UE4_Mannequin/Mesh/SK_Mannequin 애셋이다. 나머지 세 스켈레탈 메시는 엔진에서 기본으로 제공하는 애셋이다.



5. 이제 다음과 같이 보일 것이다.



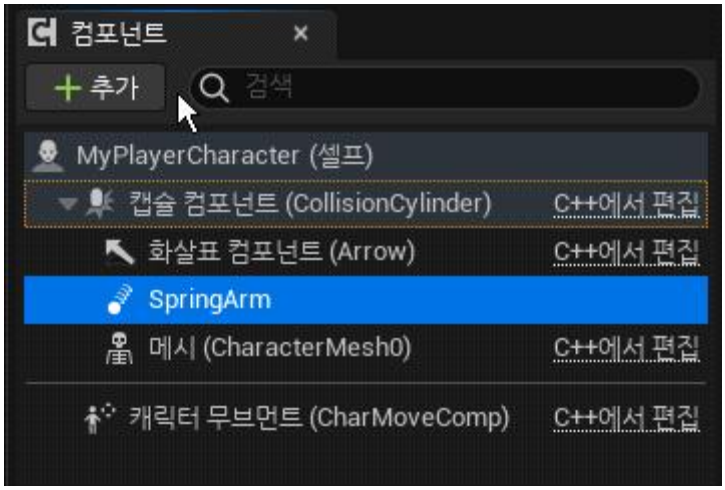
6. 우리가 추가한 스켈레탈 메시 컴포넌트의 위치가 **캡슐 컴포넌트**(CapsuleComponent)보다 너무 위쪽에 위치해서 위치를 낮추어야 한다. 또한, **화살표 컴포넌트**(ArrowComponent)의 화살표 방향이 캐릭터가 향하는 정면 방향을 표시하는데 스켈레탈 메시 컴포넌트의 방향이 맞지 않는다. 따라서 스켈레탈 메시의 위치와 방향을 수정하자. **컴포넌트** 탭에서 **메시**(Mesh) 컴포넌트를 선택하고 **디테일** 탭에서 **트랜스폼** 영역의 **위치**를 (0,0,0)에서 (0,0,-90)으로 수정하고, **회전**을 (0,0,0)에서 (0,0,-90)으로 수정하자.



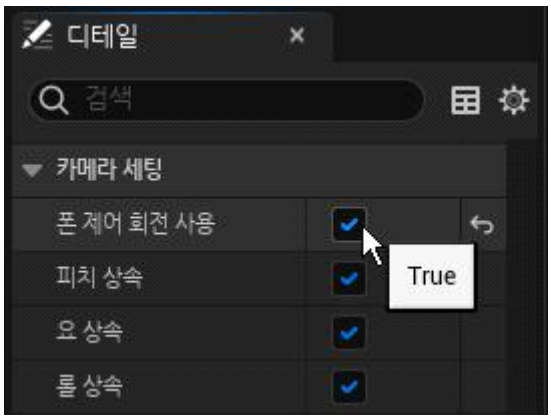
7. 이제, 카메라를 추가하자.

먼저, **컴포넌트** 탭에서 **캡슐 컴포넌트**(CapsuleComponent)를 선택하자. 그다음, **+추가**를 선택하고 드롭다

운 메뉴에서 **스프링 암(Spring Arm) 컴포넌트**를 검색하여 추가하자. **Spring Arm** 컴포넌트는 플레이어 캐릭터와 카메라를 막대로 연결하여 플레이어의 뒤쪽 위에서 아래로 내려다보는 삼인칭 시점 카메라를 구현하는 컴포넌트이다. 이름은 **SpringArm**으로 그대로 두자.

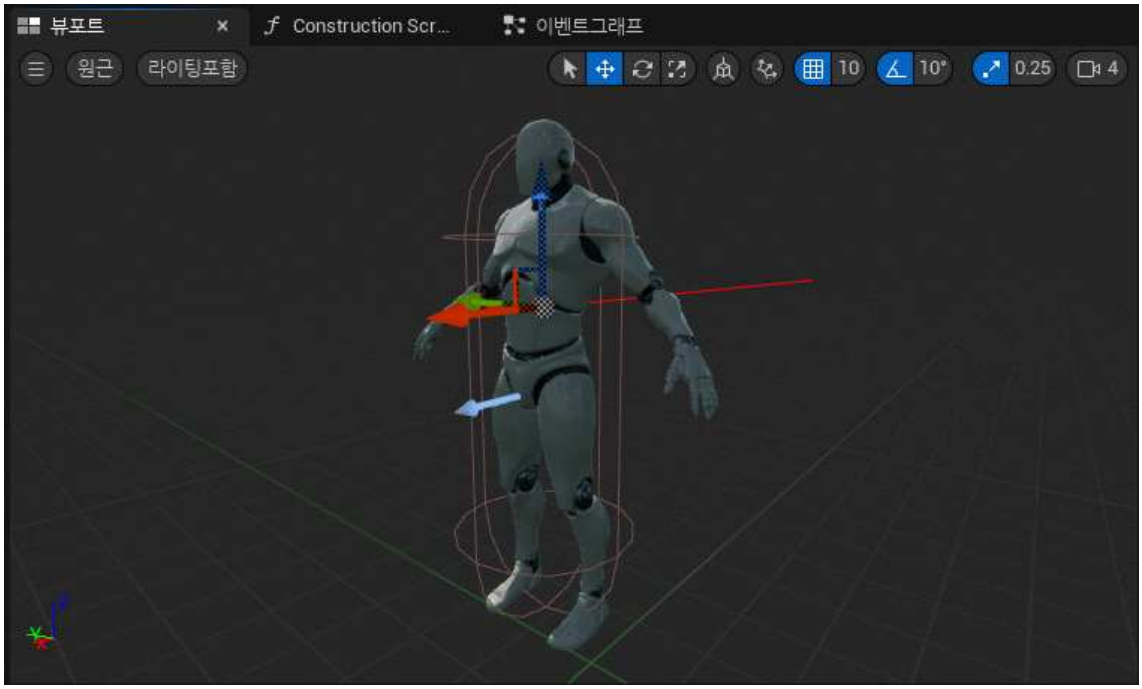


8. 그다음, **SpringArm**이 선택된 상태에서 **디테일** 탭에서 **카메라 세팅** 영역에서 **폰 제어 회전 사용(Use Pawn Control Rotation)**에 체크하자. 이 옵션은 디폴트로 꺼져 있으니 체크해주어야 한다.

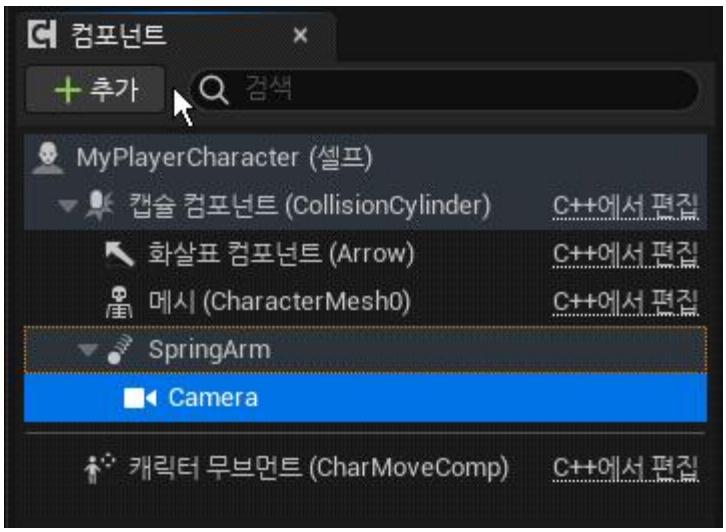


<참고> 스프링 암에 있는 **Use Pawn Control Rotation** 플래그는 카메라를 제어하는 중요한 플래그이다. 이 플래그에 체크되어 있지 않으면 폰의 컨트롤 회전값과 연동되지 않아서 마우스를 움직여도 카메라가 회전되지 않는다. 이 옵션이 체크되어있는 경우에는, **Pawn**의 컨트롤 회전값을 **SpringArm**에 적용하여 그 아래에 달려있는 카메라가 함께 연동되어 회전되도록 한다.
또한, **Use Pawn Control Rotation** 플래그를 체크할 때에 **Inherit Pitch/Yaw/Roll**에도 함께 체크되어 있어야 폰의 컨트롤 회전값 중에서 해당하는 회전값이 전달되어 온다.
일인칭의 경우에는 스프링 암을 사용하지 않고 캡슐 컴포넌트 아래에 카메라 컴포넌트를 바로 붙여서 사용하면 되고, 카메라 컴포넌트의 **Use Pawn Control Rotation** 플래그를 **true**로 하면 된다.
삼인칭의 경우에는 스프링암 컴포넌트 아래에 카메라 컴포넌트를 붙여서 사용하는데,
스프링암의 **Use Pawn Control Rotation** 플래그는 **true**로 하지만 카메라 컴포넌트의 **Use Pawn Control Rotation** 플래그는 **false**로 한다.

9. 추가된 **SpringArm**의 위치를 약간 올리자. **디테일** 탭에서 **트랜스폼** 영역에서의 **위치**을 (0,0,0)에서 (0,0,30)으로 수정하자.



10. **컴포넌트** 탭에서 **SpringArm**를 선택하고 **+추가**를 클릭하고 **Camera** 컴포넌트를 추가하자. 이름을 디폴트 이름인 **Camera**로 그대로 두자.



11. 그다음, **컴포넌트** 탭에서 **캐릭터 무브먼트** 컴포넌트를 선택하자. 그다음, 디테일 탭에서 **내비 무브먼트** 영역의 **무브먼트 기능(Movement Capabilities)** 아래에 있는 **크라우치 가능(Can Crouch)**에 체크하자.

이제 커스텀 플레이어 폰의 모든 설정이 완료되었다. 컴파일하고 저장하자.

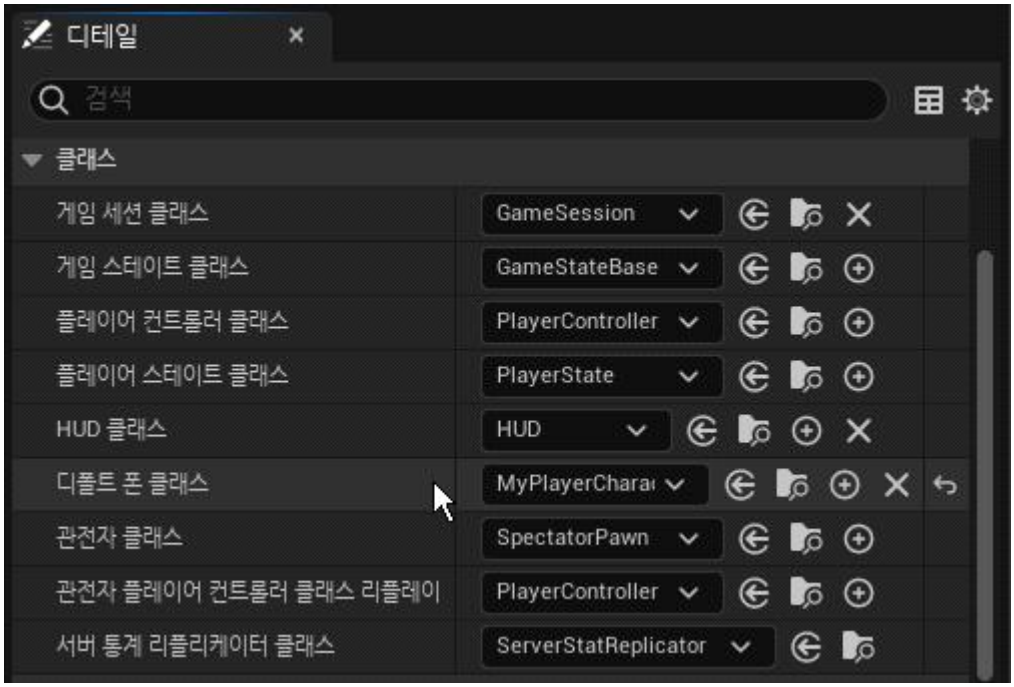
12. 우리의 커스텀 플레이어 폰이 게임 플레이 시에 사용되도록 하려면 게임 모드에 명시해주어야 한다. 이제 게임 모드를 만들자.

콘텐츠 브라우저에서 **+추가**를 클릭하고 **블루프린트 클래스**를 추가하자. **부모 클래스 선택** 창에서 **게임 모드 베이스(Game Mode Base)**를 선택하자. 이름을 **MyGameMode**로 수정하자.

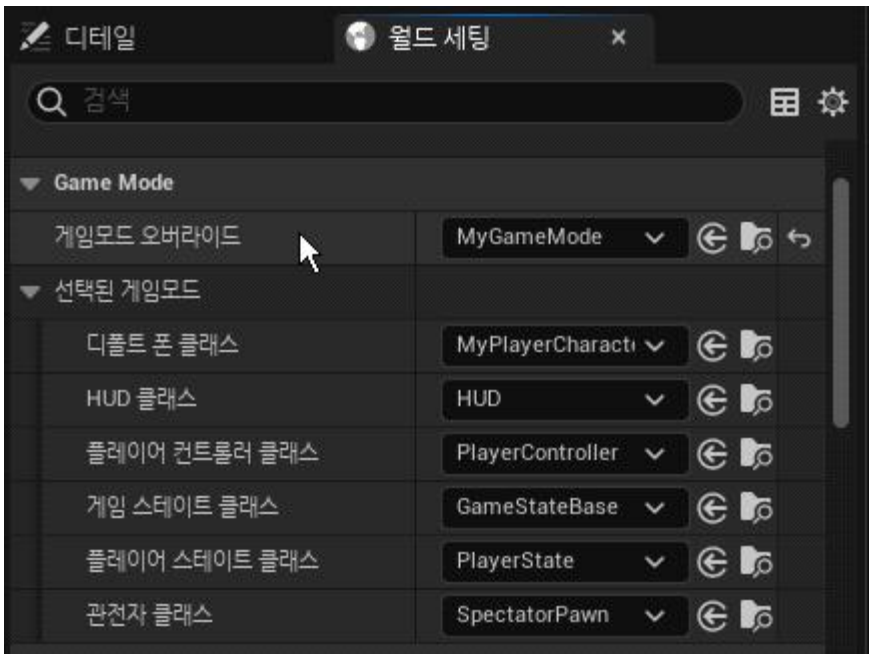
그다음, **MyGameMode**를 더블클릭하여 블루프린트 에디터를 열자.

그다음, **디테일** 탭에서 클래스 영역의 **디폴트 폰 클래스(Default Pawn Class)**를 찾자. 속성값이 **DefaultPawn**으로 되어 있는데 이를 **MyPlayerCharacter**로 수정하자.

컴파일하고 저장하자.



13. 레벨 에디터의 **월드 세팅** 탭으로 가자. **Game Mode** 영역에서 **게임모드 오버라이드** 속성이 **None**으로 되어 있는데 이것을 **MyGameMode**로 수정하자. 이제 이 레벨에서 사용되는 게임 모드를 우리의 커스텀 게임 모드로 지정하였다.



이제 모든 설정이 완료되었다.
컴파일하고 저장하자.

<참고> **월드 세팅** 탭에서 **Game Mode** 영역 아래에 있는 **선택된 게임모드** 영역을 펼치면 게임 모드의 주요 속성을 바로 수정할 수 있도록 해준다. 만약 여기에서 수정한다면 **MyGameMode**로 블루프린트 에디터에서 수정하는 것과 동일한 효과를 발휘한다. 즉 여기에서 수정하면 **MyGameMode**가 바로 수정된다.

14. 한편, **월드 세팅** 탭에서 게임 모드를 지정하는 대신 **프로젝트 세팅** 창에서도 게임 모드를 지정할 수 있다.
프로젝트 세팅 창에서 **프로젝트** » **맵&모드**를 클릭하자. 오른쪽 탭에서 **Default Modes** 영역에서 **기본 게임모드** 속성값이 **GameModeBase**으로 되어 있는데 이것을 **MyGameMode**로 수정하면 된다.

월드 세팅 탭에서 게임 모드를 지정하는 것과 **프로젝트 세팅** 창에서 게임 모드를 지정하는 것의 차이를 알아보자. **월드 세팅** 탭에서 게임 모드를 지정하면 해당 레벨에 대해서만 지정된 게임 모드가 사용된다. **월드 세팅** 탭에서 **게임모드 오버라이드** 속성이 디폴트인 **None**으로 되어 있는 경우에는 **프로젝트 세팅** 창에서 지정된 게임 모드가 사용된다.

이 절에서는 애니메이션 기능을 가진 캐릭터를 제작하기 위한 준비 과정을 학습하였다.

3. 커스텀 플레이어 폰의 입력 설정하기

이 절에서 커스텀 플레이어 폰의 입력을 설정하는 방법에 대해서 학습한다. 입력 설정은 애니메이션 학습을 위한 준비 과정에 해당한다. 특별히 입력과 관련된 새로운 내용을 다루는 것은 아니며 이전에서 다루었던 입력 매핑을 처음부터 다시 구현해본다.

먼저 프로젝트 세팅 창에서 입력 바인딩을 설정한다.
액션 매핑으로, 점프 동작, 쭈그리고 앉는 동작, 전력 질주하는 동작에 대한 매핑을 추가하자. 각 매핑의 이름을 **Jump**, **Crouch**, **Sprint**라고 하자.
축 매핑으로, 전후방으로의 움직임, 좌우로의 움직임, 사야를 좌우로 돌리기, 시야를 위아래로 회전하기에 대한 매핑을 추가하자. 각 매핑의 이름을 **MoveForward**, **MoveRight**, **Turn**, **LookUp**라고 하자.

매핑을 모두 추가한 후에는 각 매핑에 대한 이벤트 그래프를 작성하면 된다.
MoveForward 매핑과 **MoveRight** 매핑에 대한 이벤트 그래프는 폰의 **AddMovementInput** 노드를 사용하여 구현하면 된다.
LookUp 매핑과 **Turn** 매핑에 대한 이벤트 그래프는 폰의 **AddControllerPitchInput** 노드와 **AddControllerYawInput** 노드를 사용하여 구현하면 된다.
Jump 매핑에 대한 이벤트 그래프는 **Character**의 **Jump** 함수와 **StopJumping** 함수를 사용하여 구현하면 된다.
Sprint 매핑에 대한 이벤트 그래프는 **Character**의 **CharacterMovement** 컴포넌트의 **SetMaxWalkSpeed** 함수를 사용하여 구현하면 된다.
Crouch 매핑에 대한 이벤트 그래프는 **Character**의 **Crouch** 함수와 **UnCrouch** 함수를 사용하여 구현하면 된다.

이제부터 예제를 통해서 학습해보자

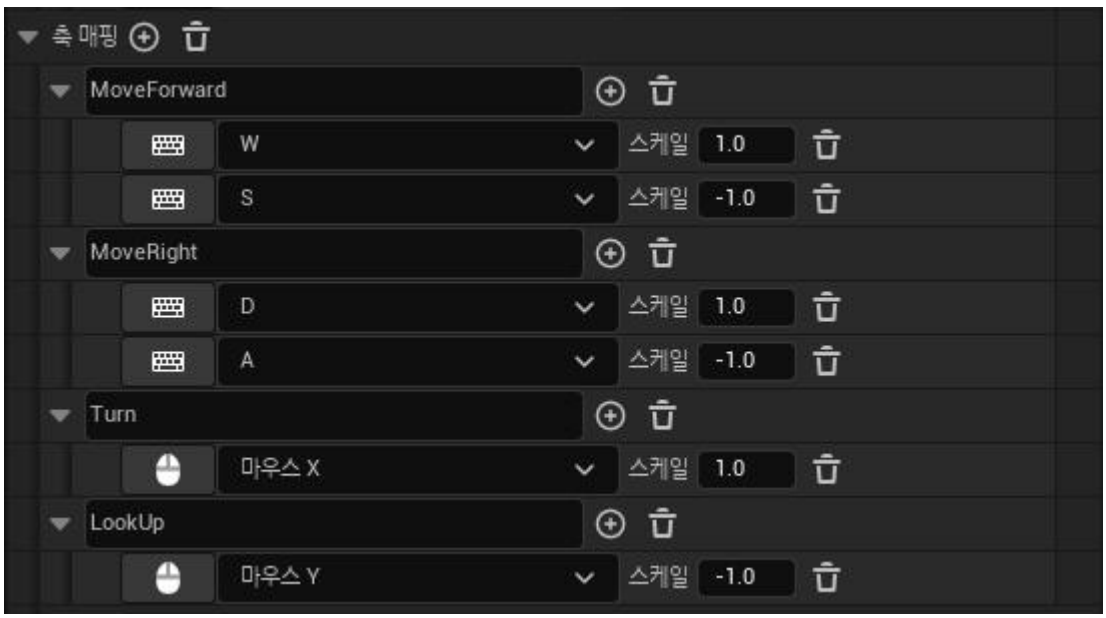
---

1. 이전 예제의 프로젝트 `Pcharprepare`에서 이어서 계속하자.

2. 이제부터 입력 키 매핑과 입력 이벤트 그래프를 작성하자.
먼저, **프로젝트 세팅** 창을 열자. 그다음, 왼쪽의 **엔진** » **입력** 탭을 클릭하자. 그다음, **바인딩** 영역에서 **액션 매핑**을 추가하자. **Jump**, **Crouch**, **Sprint**의 세 매핑을 추가하자. 각각 키보드의 **스페이스 바**, **왼쪽 Ctrl**, **왼쪽 Shift** 키에 매핑하자.

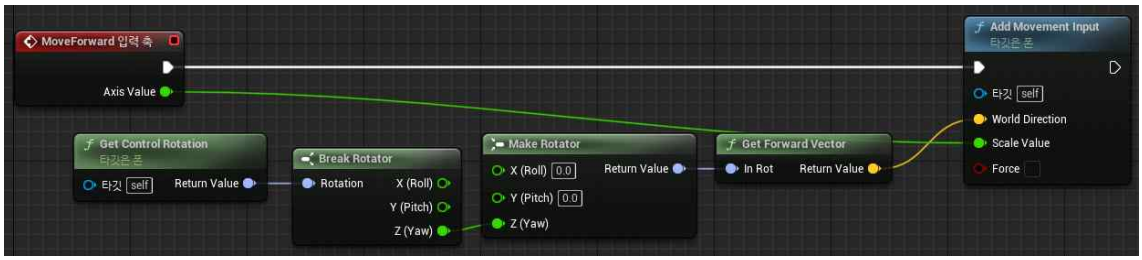
3. **바인딩**에서 **축 매핑**을 추가하자.

MoveForward, **MoveRight**, **Turn**, **LookUp**의 네 개의 매핑을 추가하자. 각각 키보드 W/S, 키보드 D/A, 마우스 X, 마우스 Y에 매핑하자. 키보드 S와 키보드 A와 마우스 Y는 **스케일**을 -1로 수정하자.



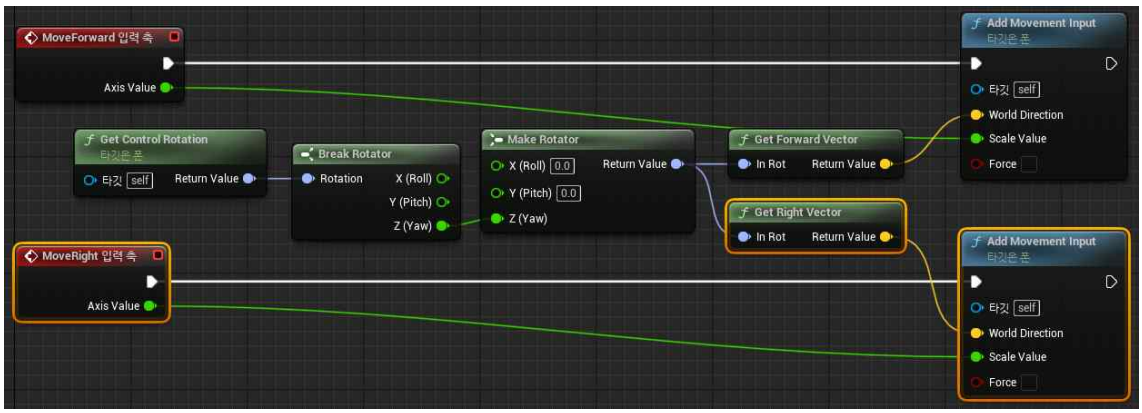
4. **MyPlayerCharacter** 블루프린트 에디터로 가자. **이벤트 그래프** 탭에서 격자판의 빈 곳에서 우클릭하고 **MoveForward 축 이벤트** 노드를 배치하자.

실행핀을 당기고 **AddMovementInput** 함수 노드를 배치하자. 이벤트 노드의 **AxisValue** 출력핀을 배치된 노드의 **ScaleValue** 입력핀에 연결하자.
그다음, **GetControlRotation** 함수 노드를 배치하자. 그다음, 출력핀을 드래그해서 **BreakRotator** 노드를 배치하자.
그다음, **MakeRotator** 노드를 배치하자. 그다음, **BreakRotator** 노드의 **Z(Yaw)** 출력핀을 당겨서 **MakeRotator**의 **Z(Yaw)** 입력핀에 연결하자.
그다음, **MakeRotator** 노드의 출력핀을 당기고 **GetForwardVector**를 배치하자.
그다음, **GetForwardVector** 노드의 출력핀을 당겨서 **AddMovementInput** 노드의 **WorldDirection** 입력핀에 연결하자.

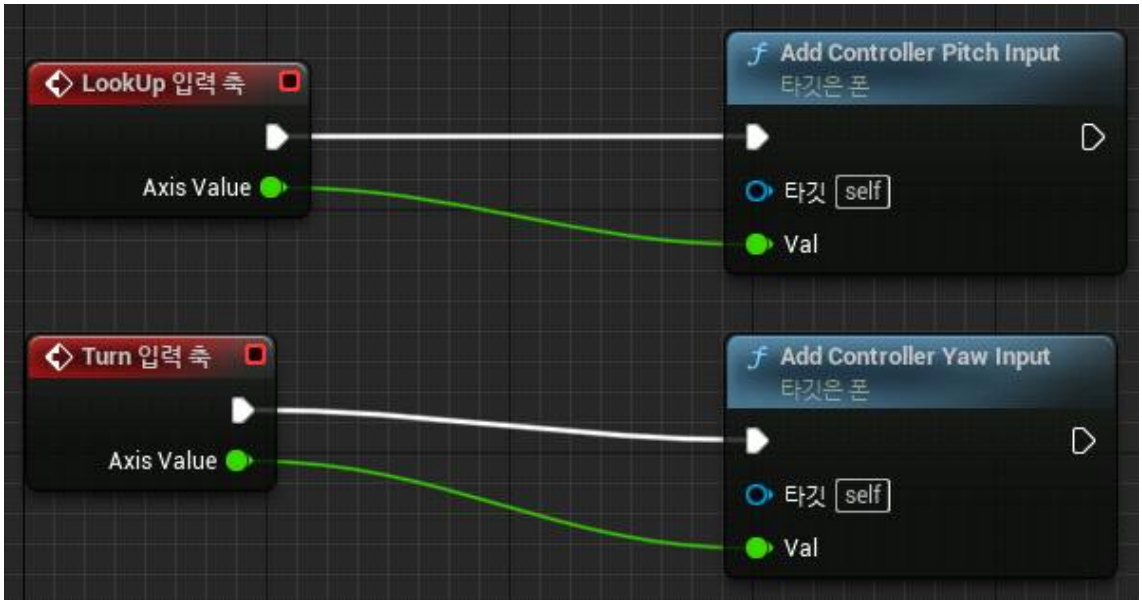


구성된 그래프는 컨트롤 회전의 전방 방향으로 이동하도록 한다. 전방 방향 벡터를 구할 때에는 **Z(Yaw)** 각도만 고려한다.

5. 빈 곳에서 우클릭하고 **MoveRight** 축 이벤트 노드를 배치하자.
그다음, 이벤트 노드의 **AxisValue** 입력핀을 당기고 **AddMovementInput** 노드를 배치하자.
그다음, 이전에 배치되었던 **MakeRotator** 노드의 출력핀을 당기고 **GetRightVector**를 배치하자.
그다음, **GetRightVector**의 출력핀을 **AddMovementInput** 노드의 **WorldDirection** 입력핀에 연결하자.



6. 빈 곳에서 우클릭하고 **LookUp** 축 이벤트 노드를 배치하자.
그다음, **Axis Value** 출력핀을 당겨서 **Add Controller Pitch Input** 노드를 배치하자.
빈 곳에서 우클릭하고 **Turn** 축 이벤트 노드를 배치하자.
그다음, **Axis Value** 출력핀을 당겨서 **Add Controller Yaw Input** 노드를 배치하자.



컴파일하고 저장하자.

이제 **MoveForward**, **MoveRight**, **Turn**, **LookUp**의 네 축 매핑에 대해서 모두 완료하였다.

7. 이제부터 **Jump**, **Crouch**, **Sprint**의 세 액션 매핑에 대해서 작성해보자.

이벤트 그래프에서 우클릭하고 **Jump** 액션 이벤트 노드를 배치하자.

그다음, 이벤트 노드의 **Pressed** 실행핀을 당기고 **점프**(**Jump**) 함수 호출 노드를 배치하자.

그다음, 이벤트 노드의 **Released** 실행핀을 당기고 **Stop Jumping** 함수 호출 노드를 배치하자.

그다음, 우클릭하여 **Sprint** 액션 이벤트 노드를 배치하자.

그다음, **컴포넌트** 탭에서 **캐릭터 무브먼트**(**CharacterMovement**) 컴포넌트를 드래그하여 레퍼런스 노드를 배치하자.

그다음, 레퍼런스 노드를 당겨서 **SetMaxWalkSpeed**를 배치하자. 그리고, **최대 걷기 속도**(**MaxWalkSpeed**) 입력핀에 1000을 지정하자. 그리고, **Sprint** 액션 이벤트 노드의 **Pressed** 출력 실행핀을 **Set** 노드의 입력 실행핀에 연결하자.

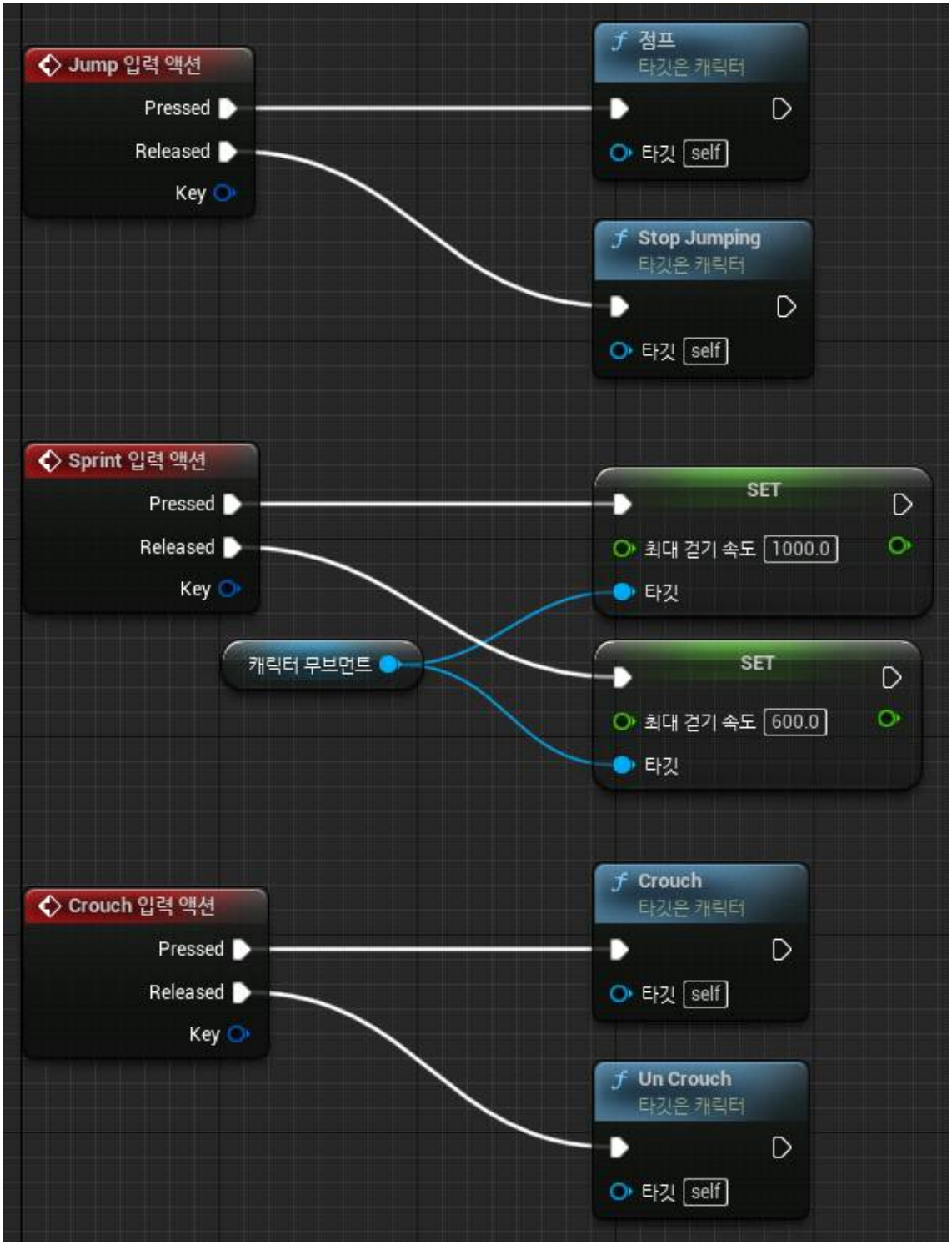
그다음, 레퍼런스 노드를 당겨서 또다시 **SetMaxWalkSpeed**를 배치하자. 그리고, **최대 걷기 속도**(**MaxWalkSpeed**) 입력핀에 600을 지정하자. 600은 걷기의 디폴트값이다. 그리고, **Sprint** 액션 이벤트 노드의 **Released** 출력 실행핀을 두 번째 **Set** 노드의 입력 실행핀에 연결하자.

그다음, **이벤트 그래프**에서 우클릭하고 **Crouch** 액션 이벤트 노드를 배치하자.

그다음, 이벤트 노드의 **Pressed** 실행핀을 당기고 **Crouch** 함수 호출 노드를 배치하자.

그다음, 이벤트 노드의 **Released** 실행핀을 당기고 **Un Crouch** 함수 호출 노드를 배치하자.

컴파일하고 저장하자.



이 절에서는 커스텀 플레이어 폰의 입력을 설정하는 방법에 대해서 학습하였다.

4. 블렌드 스페이스 만들기

이 절에서 블렌드 스페이스를 만드는 방법에 대해서 학습한다.

먼저 블렌드 스페이스에 대해서 알아보자.
애니메이션 스테이트 머신에서 대기, 걷기, 뛰기의 스테이트를 두고 현재의 폰의 속력에 따라서 전이하도록 만들었다고 하자. 속력이 1 미만이면 대기이고 5 초과이면 뛰기이고 그 중간은 걷기라고 해보자. 속력이 1에서 5 사이의 구간이면 동일한 걷기 애니메이션을 재생하게 된다. 속력이 5에 가까울수록 걷기 애니메이션보다 월드에서의 이동 속도가 빨리지므로 폰이 미끌어지며 끌려가는것처럼 보이게 된다. 또한 5에서 조금만 넘어서면 갑자기 뛰기로 바뀌게 된다. 이러한 문제를 해결하는 방법으로 블렌드 스페이스가 있다.
블렌드 스페이스(Blend Space)는 애님그래프에서 사용할 수 있는 특별한 애셋으로 두 입력값에 따라 애니메이션을 블렌딩시켜주는 애셋이다.

<참고> 블렌드 스페이스에 대한 자세한 내용은 다음의 문서를 참조하자.
https://docs.unrealengine.com/blend-spaces-in-unreal-engine/

<참고> 하나의 입력에 따라 두 애니메이션을 섞는 단순 블렌딩은 **Blend** 노드를 사용하면 된다. **Blend** 노드에 대해서는 다음의 문서를 참조하자.
https://docs.unrealengine.com/animation-blueprint-blend-nodes-in-unreal-engine/

<참고> 블렌드 스페이스는 기본적으로 두 개의 입력값을 고려하는 2차원적인 블렌드 공간이다. 한편, 하나의 입력값만을 고려하는 1차원적인 블렌드 공간도 있다.

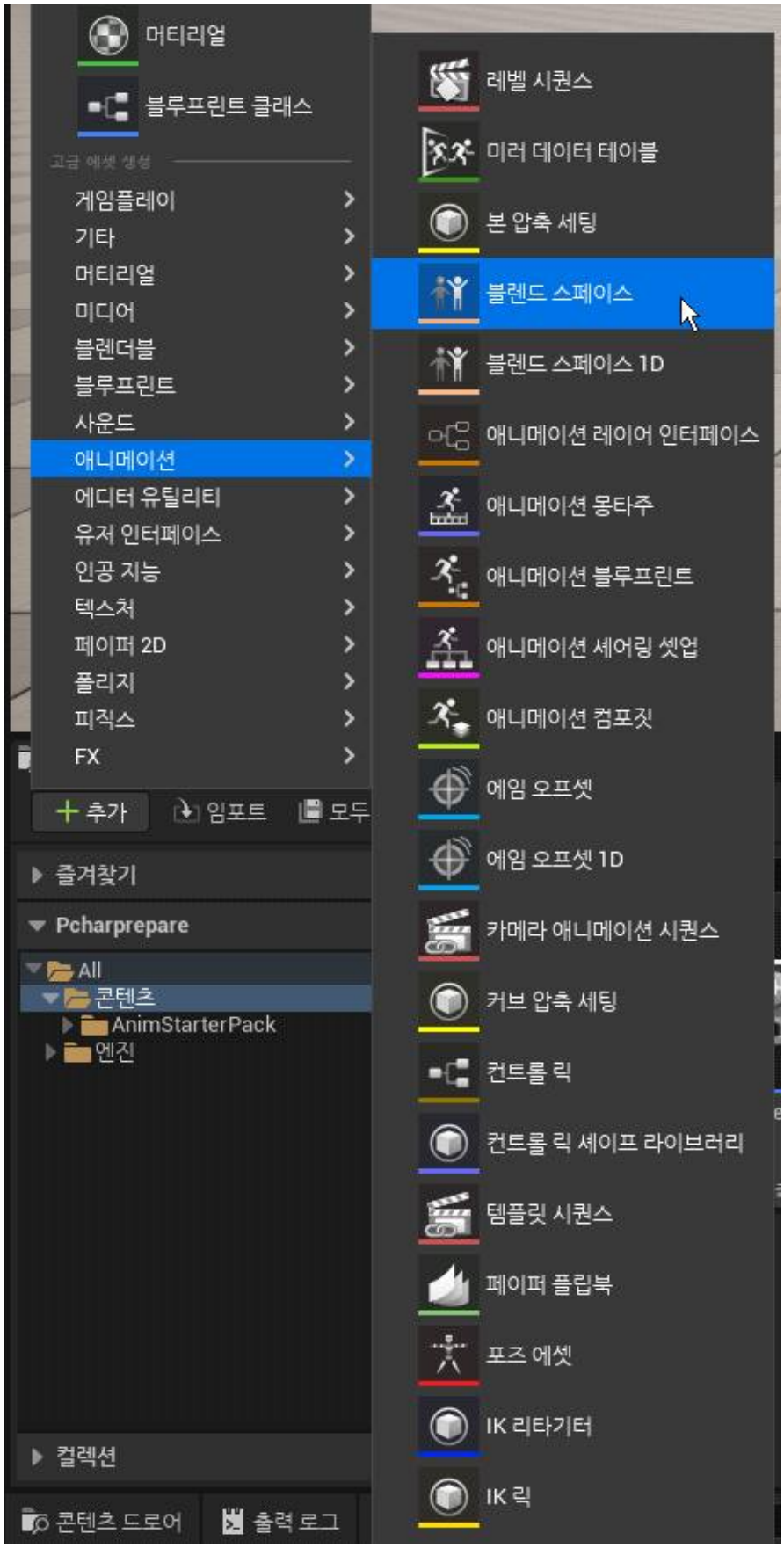
이제부터 예제를 통해서 학습해보자

---

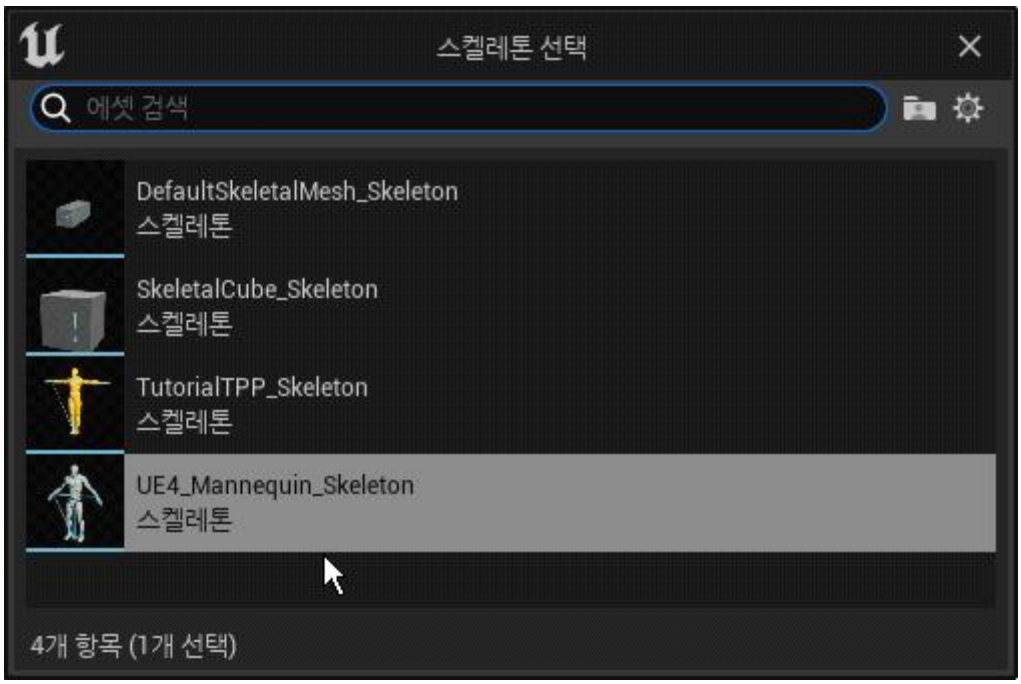
1. 이전 예제의 프로젝트 `Pcharprepare`에서 이어서 계속하자.

2. 이제부터 블렌드 스페이스를 만들자.
먼저, 서있기나 걷기나 뛰기의 직립 보행 애니메이션을 위한 로코모션 블렌드 스페이스를 만들자. 블렌드 스페이스는 두 개의 입력값에 기반하여 애니메이션을 블렌딩하는 특별한 애셋이다. 우리는 캐릭터의 이동 속력과 방향에 따라서 전후좌우 이동을 블렌딩하는 블렌드 스페이스를 만들어보자.

3. 먼저 **콘텐츠 브라우저**에서 **+추가**를 클릭하고 **애니메이션** » **블렌드 스페이스**를 선택하여 추가하자.

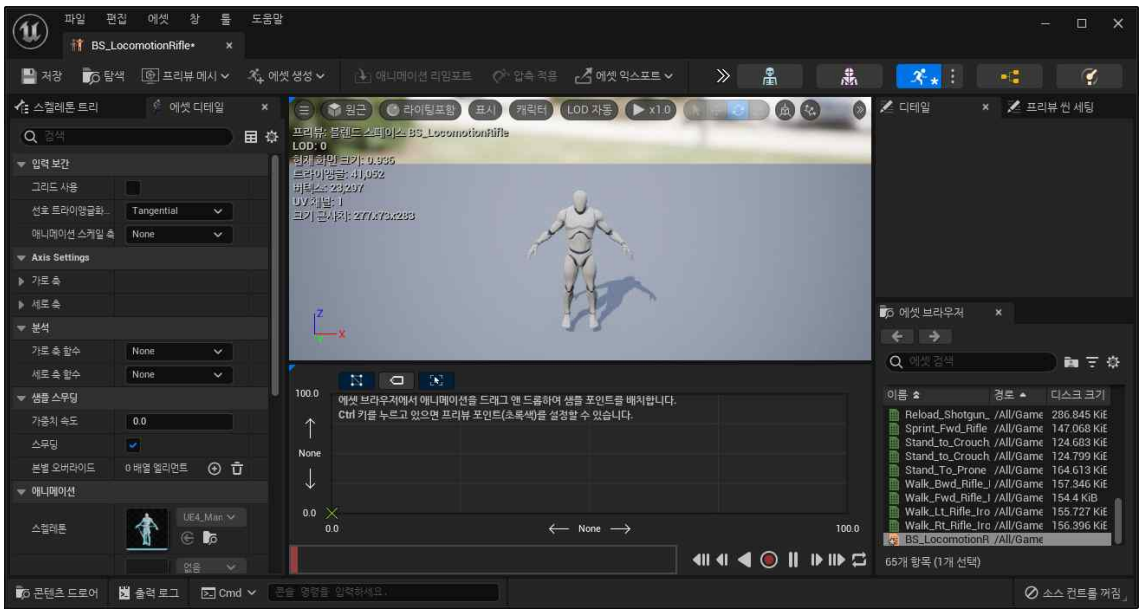

머티리얼
블루프린트 클래스
고급 에셋 생성
게임플레이
기타
머티리얼
미디어
블렌더블
블루프린트
사운드
애니메이션
에디터 유틸리티
유저 인터페이스
인공 지능
텍스처
페이퍼 2D
폴리지
피직스
FX
추가
임포트
모두
즐겨찾기
Pcharprepare
All
콘텐츠
AnimStarterPack
엔진
컬렉션
콘텐츠 드로어
출력 로그
레벨 시퀀스
미러 데이터 테이블
본 압축 세팅
블렌드 스페이스
블렌드 스페이스 1D
애니메이션 레이어 인터페이스
애니메이션 몽타주
애니메이션 블루프린트
애니메이션 셰어링 셋업
애니메이션 컴포짓
에임 오프셋
에임 오프셋 1D
카메라 애니메이션 시퀀스
커브 압축 세팅
컨트롤 릭
컨트롤 릭 세이프 라이브러리
템플릿 시퀀스
페이퍼 플립북
포즈 에셋
IK 리타기터
IK 릭

4. 그다음, **스켈레톤 선택** 창에서 **UE4_Mannequin_Skeleton**을 선택하자.



생성된 블렌드 스페이스 애셋의 이름을 **BS_LocomotionRifle**로 수정하자.

5. 블렌드 스페이스 **BS_LocomotionRifle**을 더블클릭하자. 애니메이션 에디터가 열릴 것이다.



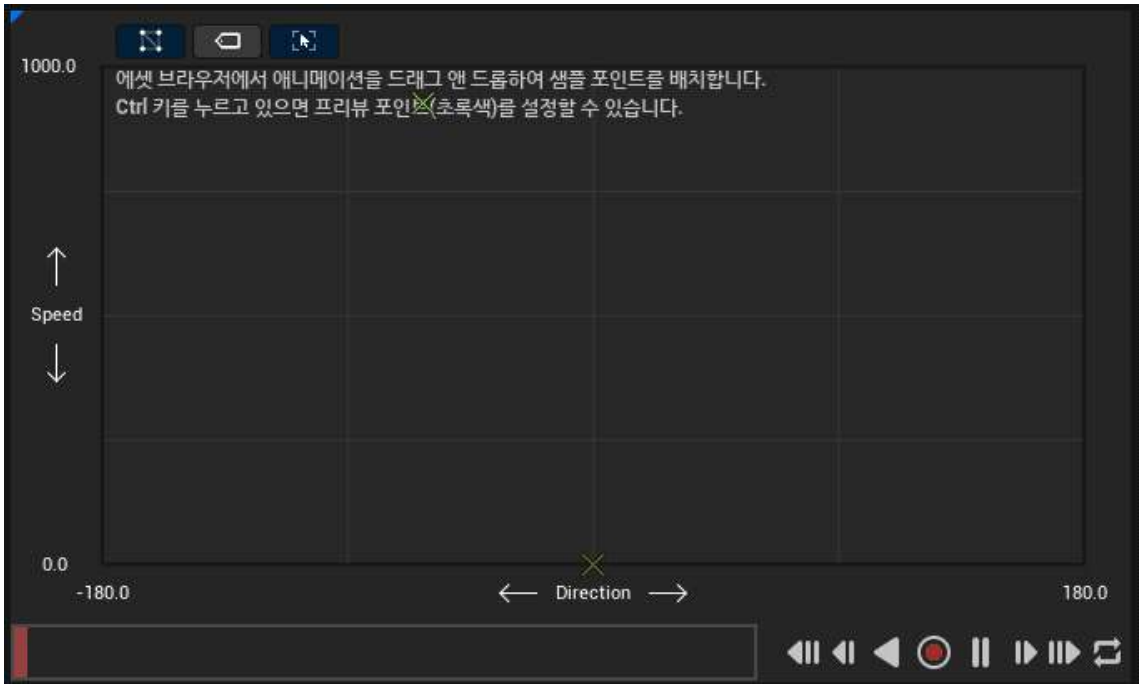
6. 왼쪽의 **애셋 디테일** 탭에서 **Axis Settings** 영역의 **가로 축**과 **세로 축**을 펼쳐보자.

먼저, **가로 축**에 대해서, **이름**(**Name**) 속성값을 **None**에서 **Direction**으로 수정하자. 그다음, **최소 축 값**(**Minimum Axis Value**)을 0에서 -180으로 수정하고, **최대 축 값**(**Maximum Axis Value**)을 100에서 180으로 수정하자.

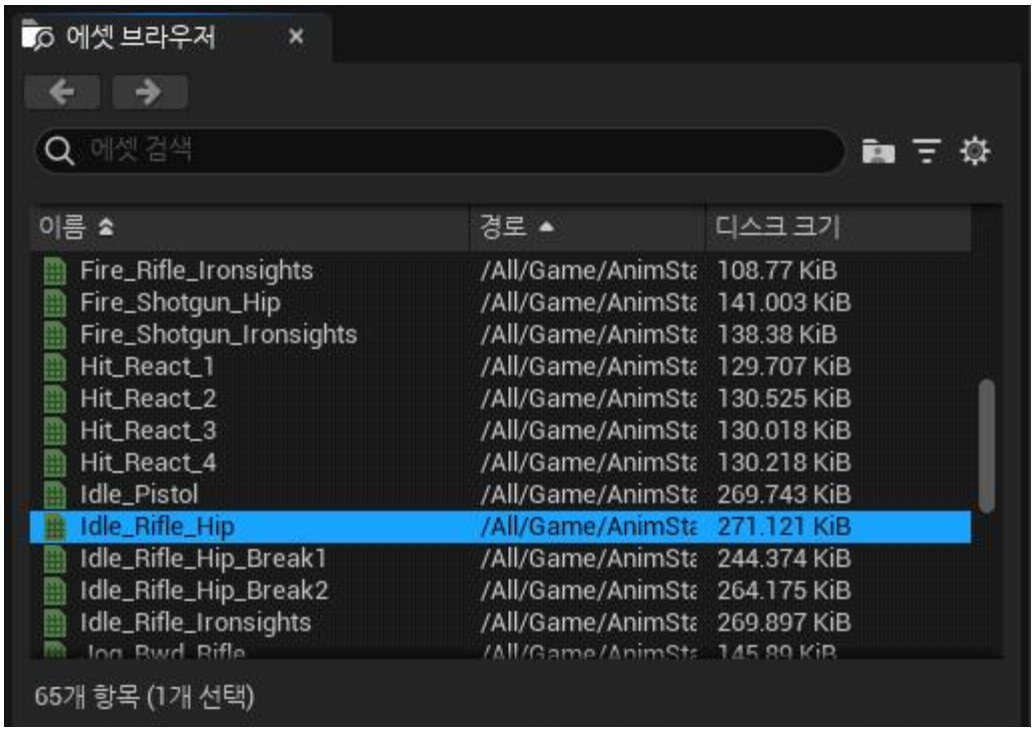
그다음, **세로 축**에 대해서, **이름**(**Name**)을 **None**에서 **Speed**로 수정하자. 그다음, **최소 축 값**(**Minimum Axis Value**)은 0으로 그대로 두고, **최대 축 값**(**Maximum Axis Value**)을 100에서 1000으로 수정하자.

컴파일하고 저장하자.

7. 뷰포트의 아래에 격자판이 있을 것이다. 이것이 블렌드 스페이스이다. 격자판은 가로는 **Direction** 축이고 범위가 [-180,180]이다. 세로는 **Speed** 축이고 범위가 [0,1000]이다. 각 축에 대해서 4등분 격자선이 표시되어 있다. 각 격자선과 축의 만나는 지점이 총 5x5이므로 25개의 교점이 있다. 이들 교점에 애니메이션 시퀀스를 드래그해서 배치할 수 있다.

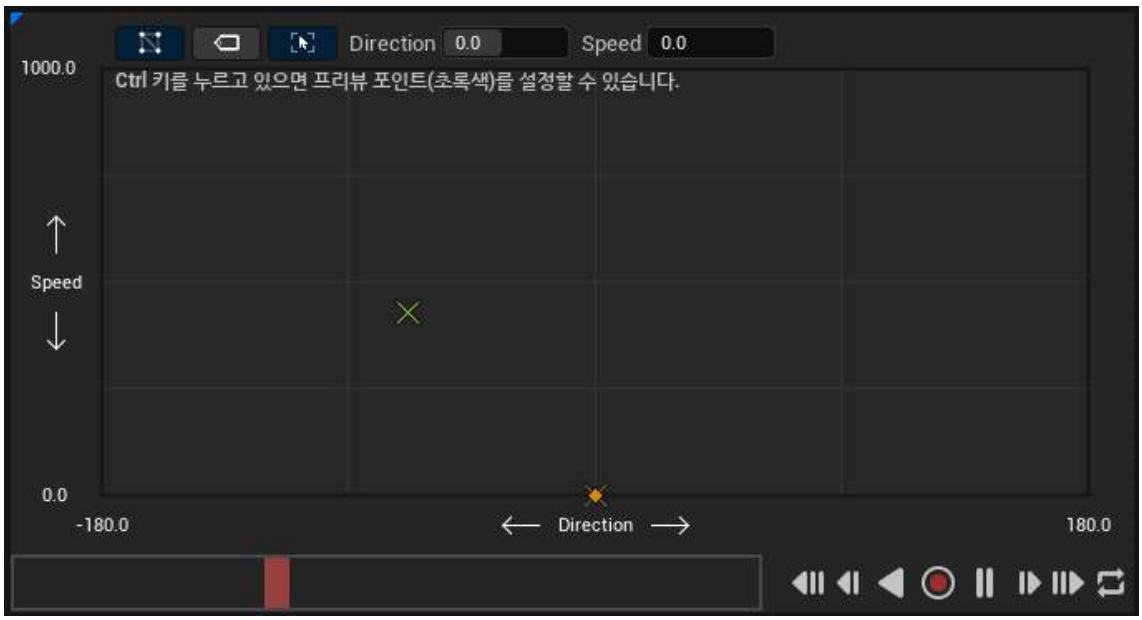


8. 애니메이션 에디터의 오른쪽 아래에는 **애셋 브라우저**가 있다. **애셋 브라우저**에서는 현재 프로젝트에서 사용할 수 있는 애니메이션 시퀀스 애셋와 블렌드 스페이스 애셋의 목록을 보여준다.



9. **애셋 브라우저**에서 **Idle_Rifle_Hip** 애니메이션 시퀀스를 찾자. 검색상자를 사용하면 쉽게 찾을 수 있다. 찾은 **Idle_Rifle_Hip**을 드래그해서 블렌드 스페이스의 **Direction**이 0이고 **Speed**가 0인 (0,0) 위치에 배치하자. 배치된 후에는 교점에 마름모 모양이 표시된다. 이 모양을 클릭하면 위쪽에 좌표값이 표시된다. 드래그하여 위치를 이동할 수도 있다. 정확한 배치를 위해서는 상단의 **Direction**과 **Speed**

값에 0,0을 직접 입력하면 된다.



10. 그다음, (0,0) 위치 이외의 지점에 대해서도 애셋 브라우저에서의 애니메이션 시퀀스를 드래그해서 블렌드 스페이스의 교점에 배치하자. 아래의 배치도에 나타내기 위해 각 애니메이션 시퀀스에 대해서 ①와 같은 원문자로 표시하였다.

Idle_Rifle_Hip(①)을 (-180,0), (-90,0), (90,0), (180,0)에도 추가적으로 배치하자.

Walk_Fwd_Rifle_Ironsights(②)를 (0,250)에 배치하자.

Walk_Lt_Rifle_Ironsights(③)을 (-90,250)에 배치하자.

Walk_Rt_Rifle_Ironsights(④)을 (90,250)에 배치하자.

Walk_Bwd_Rifle_Ironsights(⑤)을 (-180,250)과 (180,250)에 배치하자.

Jog_Fwd_Rifle(⑥)를 (0,500)에 배치하자.

Jog_Lt_Rifle(⑦)를 (-90,500)에 배치하자.

Jog_Rt_Rifle(⑧)를 (90,500)에 배치하자.

Jog_Bwd_Rifle(⑨)를 (-180,500)과 (180,500)에 배치하자.

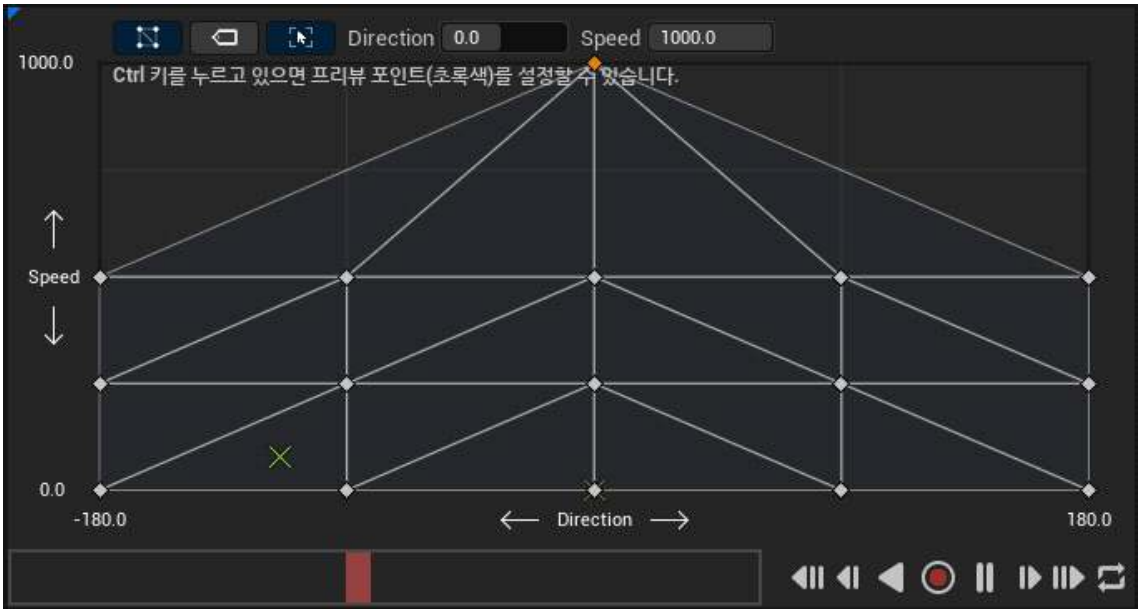
Sprint_Fwd_Rifle(⑩)를 (0,1000)에 배치하자.

전체적으로 아래의 표와 같이 배치될 것이다.

Direction / Speed	-180	-90	0	90	180
1000			⑩		
750					
500	⑨	⑦	⑥	⑧	⑨
250	⑤	③	②	④	⑤
0	①	①	①	①	①

컴파일하고 저장하자.

11. 모든 애니메이션 시퀀스가 배치된 후에는 다음과 같이 보일 것이다.



오렌지색 점은 현재 선택된 점이다. 다른 점을 클릭하면 선택이 바뀐다. 상단의 좌표값을 입력하면 선택된 점의 좌표가 바뀐다.

녹색 십자 마커는 현재 뷰포트에서 프리뷰되는 지점을 나타낸다. **Ctrl** 키를 누르고 클릭할 필요 없이 마우스를 움직이기만 하면 위치가 바뀐다. 이 기능을 사용하면 현재의 블렌드 스페이스의 애니메이션을 쉽게 확인할 수 있다.

12. 블렌드 스페이스를 하나 더 만들자.

쭈그리고 앉아서 있거나 이동하는 애니메이션에 대한 블렌드 스페이스를 만들자. **콘텐츠 브라우저**에서 **+추가**를 클릭하고 **애니메이션 » 블렌드 스페이스**를 선택하여 추가하자.

그다음, **스켈레톤 선택** 창에서 **UE4_Mannequin_Skeleton**을 선택하자.

그다음, 생성된 블렌드 스페이스 애셋의 이름을 **BS_LocomotionCrouch**로 수정하자.

그다음, **BS_LocomotionCrouch**을 더블클릭하자. 애니메이션 에디터가 열릴 것이다. 왼쪽의 **애셋 디테일** 탭에서 **Axis Settings** 영역의 **가로 축**과 **세로 축**을 펼쳐보자.

가로 축에 대해서, **이름(Name)** 속성값을 **None**에서 **Direction**으로 수정하자. **최소 축 값(Minimum Axis Value)**을 -180으로 수정하고 **최대 축 값(Maximum Axis Value)**을 180으로 수정하자.

세로 축에 대해서, **이름(Name)** 속성값을 **None**에서 **Speed**로 수정하자. **최소 축 값(Minimum Axis Value)**은 0으로 그대로 두고 **최대 축 값(Maximum Axis Value)**을 300으로 수정하자.

13. 오른쪽의 애셋 브라우저에서 애니메이션 시퀀스를 드래그해서 블렌드 스페이스 격자판의 교점에 배치하자.

Crouch_Idle_Rifle_Hip(①)을 (-180,0), (-90,0), (0,0), (90,0), (180,0)에 배치하자.

Crouch_Walk_Fwd_Rifle_Hip(②)를 (0,300)에 배치하자.

Crouch_Walk_Lt_Rifle_Hip(③)을 (-90,300)에 배치하자.

Crouch_Walk_Rt_Rifle_Hip(④)을 (90,300)에 배치하자.

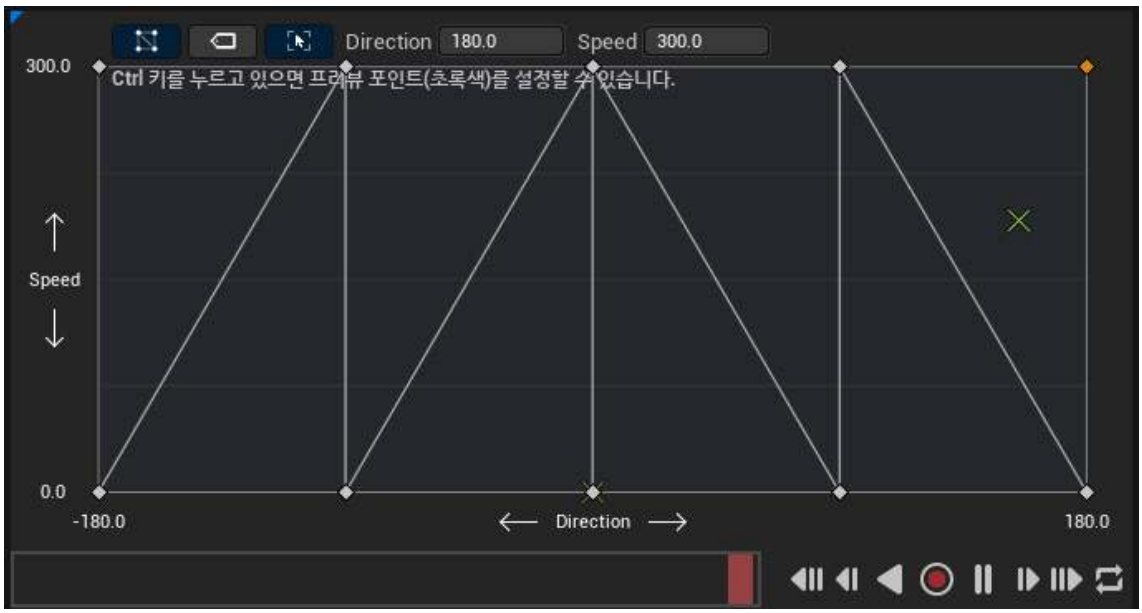
Crouch_Walk_Bwd_Rifle_Hip(⑤)을 (-180,300), (180,300)에 배치하자.

전체적으로 아래의 표와 같이 배치될 것이다.

Direction / Speed	-180	-90	0	90	180
300	⑤	③	②	④	⑤
225					
150					
75					
0	①	①	①	①	①

컴파일하고 저장하자.

14. 모든 애니메이션 시퀀스가 배치된 후에는 다음과 같이 보일 것이다.



이제 필요한 두 개의 블렌드 스페이스를 모두 완성하였다.

이 절에서는 블렌드 스페이스를 만드는 방법을 학습하였다.

5. 애니메이션 블루프린트 만들기

이 절에서 애니메이션 블루프린트를 만드는 방법에 대해서 학습한다.
애니메이션 블루프린트(animation blueprint)는 스켈레톤에 적용하여 애니메이션을 구현하는 블루프린트이다. 기존의 블루프린트는 기존의 액터 클래스의 기능을 추가하는 용도인 반면에 **애니메이션 블루프린트**는 미리 지정해둔 스켈레톤에 대한 애니메이션 시퀀스들의 재생 기능을 구현하는 용도의 특별한 블루프린트이다.
애니메이션 블루프린트는 캐릭터 폰 내에 있는 스켈레탈 메시 컴포넌트에 연결되어 사용된다.

애니메이션 블루프린트는 스켈레탈 메시의 애니메이션 재생 방법을 명시한다. 애니메이션 블루프린트는 별도의 애셋으로 존재하므로 캐릭터 폰과 연결하기 위해서는 캐릭터 폰에 지정해주어야 한다. 한 캐릭터 폰의 애니메이션 재생 방법으로 특정 애니메이션 블루프린트 에셋이 사용되도록 지정하는 방법을 알아보자. 캐릭터 내에 포함된 스켈레탈 메시 컴포넌트인 **Mesh** 컴포넌트의 **Animation Mode** 속성을 **Use Animation Blueprint**로 두고 **Anim Class** 속성에 애니메이션 블루프린트 애셋을 지정하면 된다.
한편 **Animation Mode** 속성을 다른 방식으로 지정할 수도 있다. 속성을 **Use Animation Assset**으로 지정하면 단일 애니메이션 애셋을 재생할 수 있다. 애니메이션 애셋이라 함은 애니메이션 시퀀스 애셋이나 또는 블렌드 스페이스 애셋을 의미한다. 속성을 **Use Animation Assset**으로 바꾼 후에는 그 아래에 **Anim to Play** 속성이 나타날 것이다. 이 속성값에 원하는 애니메이션 시퀀스나 블렌드 스페이스 애셋을 지정하면 된다. **애니메이션 블루프린트**는 여러 애니메이션 애셋을 사용하는 복잡한 방법의 애니메이션 구현이 가능하지만, 단일 애니메이션 애셋 재생 모드의 경우에는 하나의 애니메이션 애셋만을 재생하게 된다. 단순한 애니메이션이 반복되는 경우에는 이 모드를 사용하면 된다.

애니메이션 블루프린트 에디터에는 디폴트로 에디터 중간에 두 개의 격자판 탭을 보여준다. 하나는 이벤트 그래프 탭이고 다른 하나는 **애님그래프**(**AnimGraph**) 탭이다. 애니메이션 블루프린트를 작성하기 위해서는 일반적인 형태의 그래프인 이벤트 그래프도 작성하고 특별한 형태의 그래프인 애님그래프도 작성해야 한다.
첫 번째 탭인, 이벤트 그래프 탭에서는 주로 **Blueprint Update Animation 이벤트** 노드 그래프를 작성한다. 이 이벤트 그래프는 애니메이션 업데이터 시마다 실행되는 그래프이며, 애니메이션과 관련된 상태값을 수정하는 기능을 수행한다.
또다른 탭인, **AnimGraph** 탭에서는 **AnimGraph**를 작성한다. **AnimGraph**는 스켈레탈 메시의 최종 포즈를 계산하는 그래프이다. 이 그래프는 여러 애니메이션 시퀀스 애셋이나 블렌드 스페이스 애셋으로부터 현재 프레임에서의 최종 포즈를 결정한다.

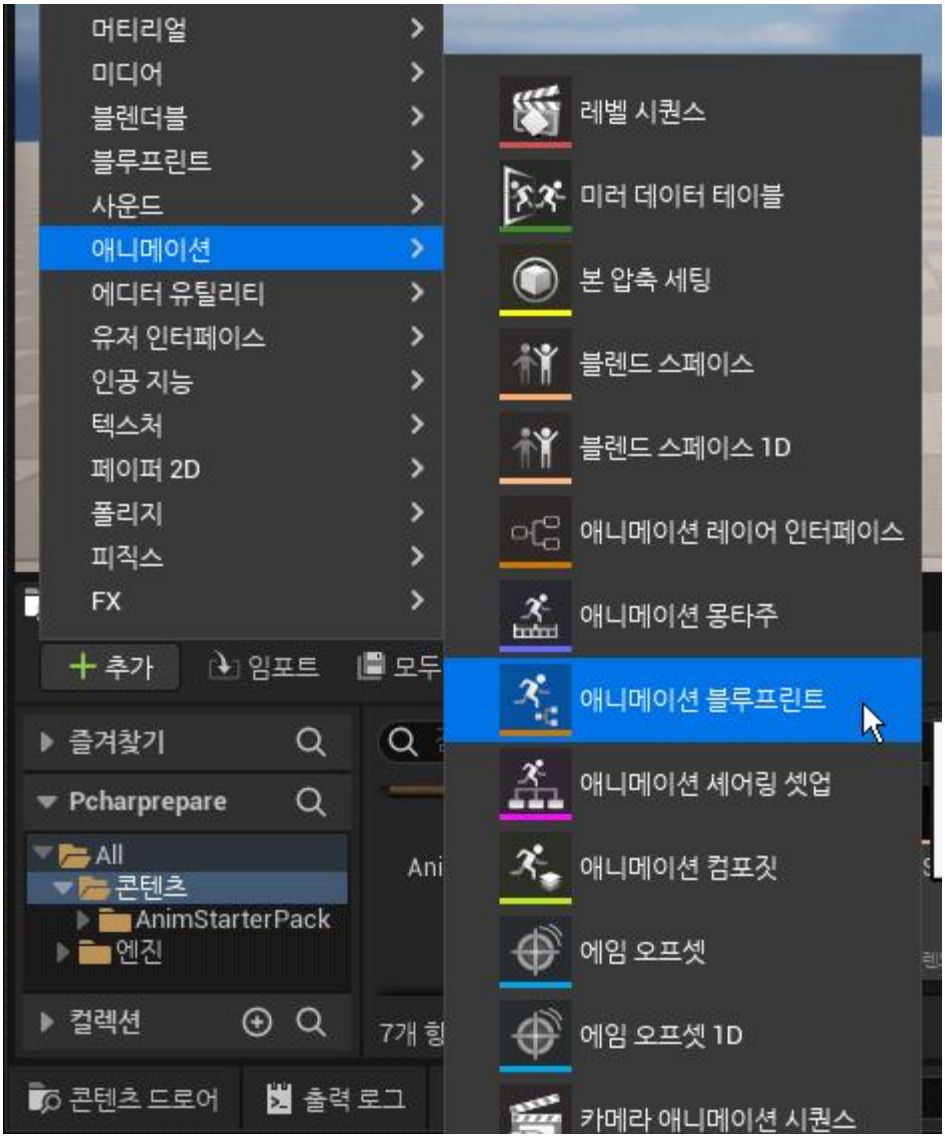
이번 예제에서는 이벤트 그래프 탭에서의 작성 방법을 학습한다. **AnimGraph** 작성 방법은 다음 예제에서 다룬다.
이제부터 예제를 통해서 학습해보자

---

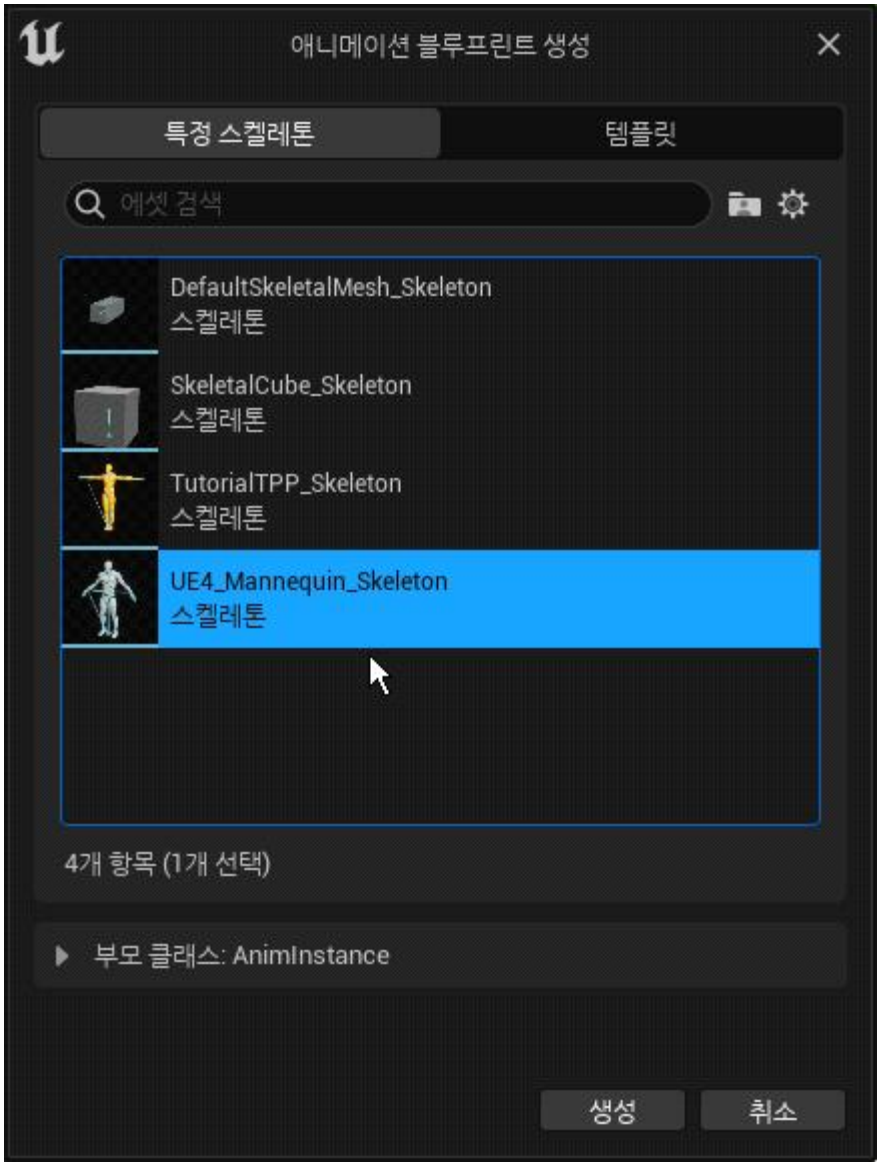
1. 이전 예제의 프로젝트 `Pcharprepare`에서 이어서 계속하자.

2. 이제부터 애니메이션 블루프린트를 만들자.

애니메이션 블루프린트에서는 플레이어의 현재의 활동 상태에 따라서 어떤 애니메이션을 재생할지를 결정한다. 또한 애니메이션 블루프린트에서는 스테이스 머신을 만들어서 그 안에서 걷기와 쭈그리기 상태를 생성하고 이들 간의 전이를 설정한다.
먼저, 콘텐츠 브라우저에서 **+추가**를 클릭하고 **애니메이션** » **애니메이션 블루프린트**를 선택하자.



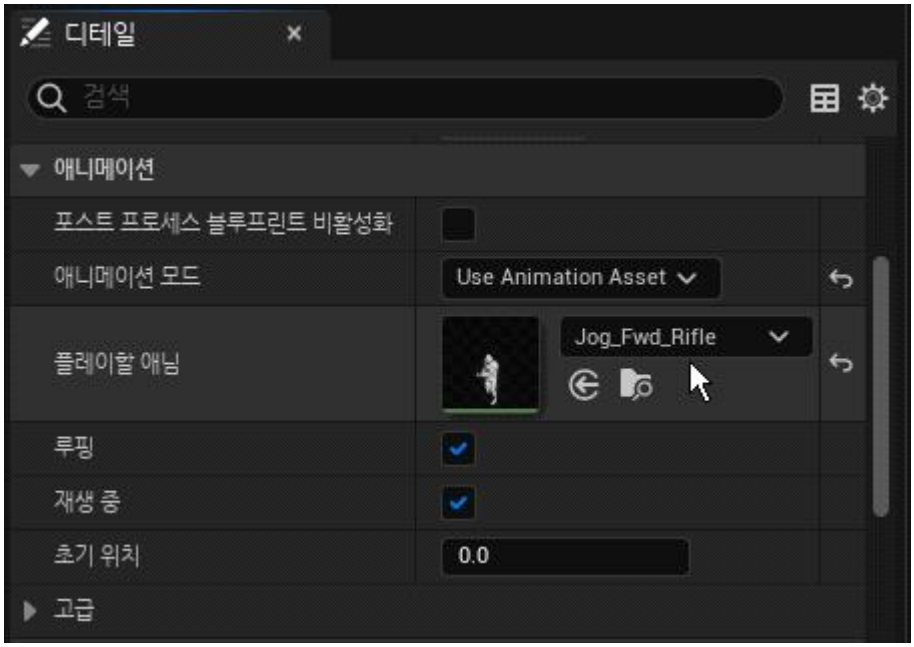
3. **애니메이션 블루프린트** 생성을 위한 대화창이 뜬다. 이 대화창에서는 타깃 스켈레톤을 선택하면 된다. 우리는 **UE4_Mannequin_Skeleton**을 선택하자.
그다음, 아래의 부모 클래스는 **AnimInstance**로 그대로 두고 **생성** 버튼을 클릭하자.
애니메이션 블루프린트가 생성될 것이다. 우리는 디폴트 이름인 **NewAnimBlueprint**을 **PlayerCharacter_AnimBP**로 수정하자.



<참고> 이 대화창에서는 생성할 애니메이션 블루프린트 클래스의 부모 클래스도 선택할 수 있다. 엔진이 제공하는 기본적인 부모 클래스는 **AnimInstance**이다. 모든 애니메이션 블루프린트 클래스는 이 클래스를 상속해서 구현해야 한다. 따라서 특별한 경우가 아니면 애니메이션 블루프린트 클래스의 부모 클래스로 **AnimInstance**를 선택하자.
만약 우리가 이미 만들어둔 애니메이션 블루프린트 클래스가 있고 그 클래스를 다시 상속해서 만들고 싶다면 자신이 만들어둔 클래스를 부모클래스로 지정할 수도 있다.

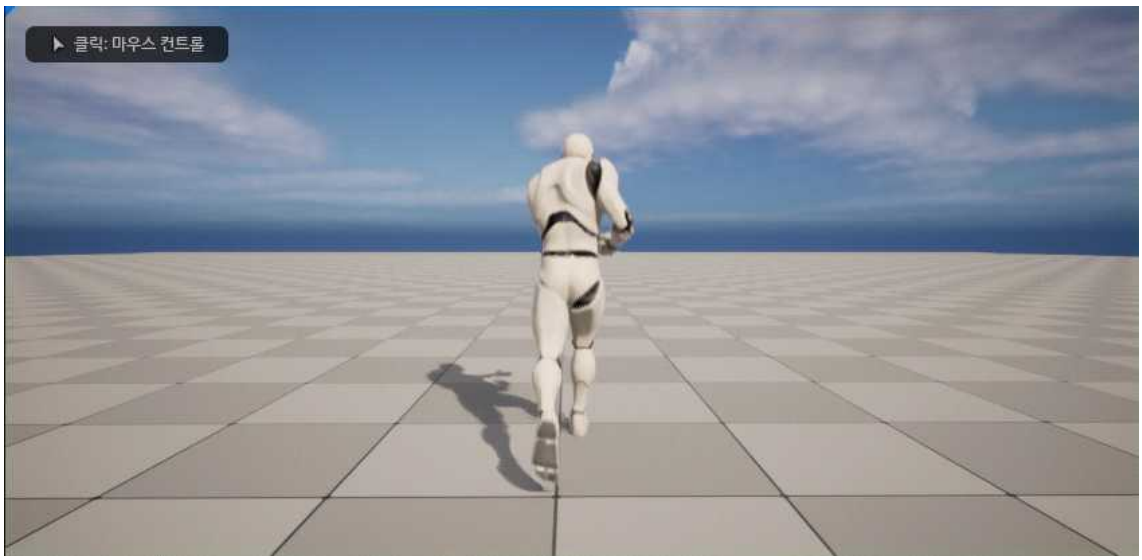
4. 콘텐츠 브라우저에서 **MyPlayerCharacter**를 더블클릭하여 블루프린트 에디터를 열자.
그다음, 왼쪽의 **컴포넌트** 탭에서 스켈레탈 메시 컴포넌트인 **메시(Mesh)** 컴포넌트를 선택하자.
그다음, 오른쪽의 디테일 탭에서 **애니메이션** 영역을 살펴보자.
애니메이션 모드(Animation Mode) 속성이 디폴트로 **Use Animation Blueprint**로 되어 있다. 스켈레탈 메시의 애니메이션 구현은 대부분 **애니메이션 블루프린트**로 구현하기 때문이다.
우리는 테스트를 위해 **Use Animation Blueprint** 이외의 다른 방식을 먼저 사용해보자. 단일 애니메이션 애셋을 재생하도록 **애니메이션 모드** 속성값을 **Use Animation Assset**으로 바꾸어보자.

Use Animation Assset으로 바꾼 후에는 그 아래에 **플레이할 애님(Anim to Play)** 속성이 나타날 것이다. 디폴트 속성값인 **없음**을 클릭하고 Jog_Fwd_Rifle 애니메이션 시퀀스를 선택하자.



컴파일하고 저장하자.

5. 레벨을 플레이해보자. 앞으로 뛰는 애니메이션이 재생될 것이다.



단일 애니메이션 애셋 재생에 대한 테스트를 해보았다.

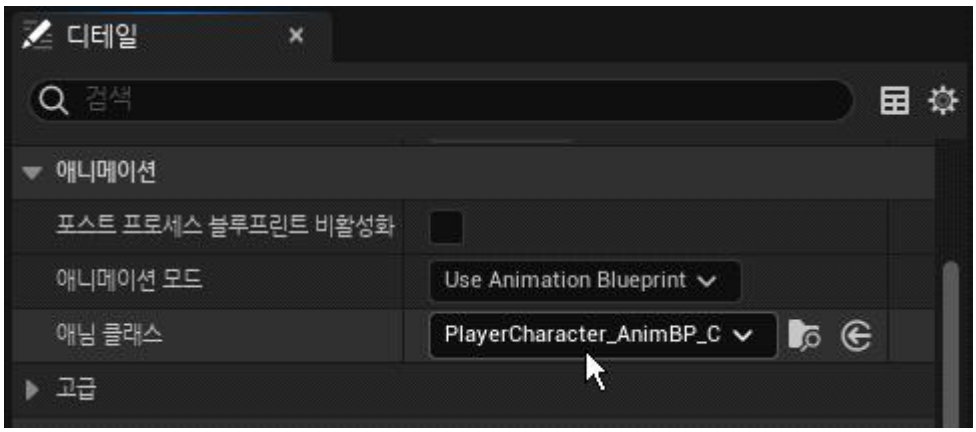
6. 이제, 단일 애니메이션 애셋 재생이 아닌 애니메이션 블루프린트를 사용한 애니메이션이 되도록 지정하자.

다시, **MyPlayerCharacter**를 블루프린트 에디터로 돌아가자. 그리고 다시, **컴포넌트** 탭의 **메시** 컴포넌트를 선택하고 디테일 탭의 **애니메이션** 영역을 보자.

그리고, **애니메이션 모드(Animation Mode)** 속성값을 디폴트인 **Use Animation Blueprint**로 바꾸자.

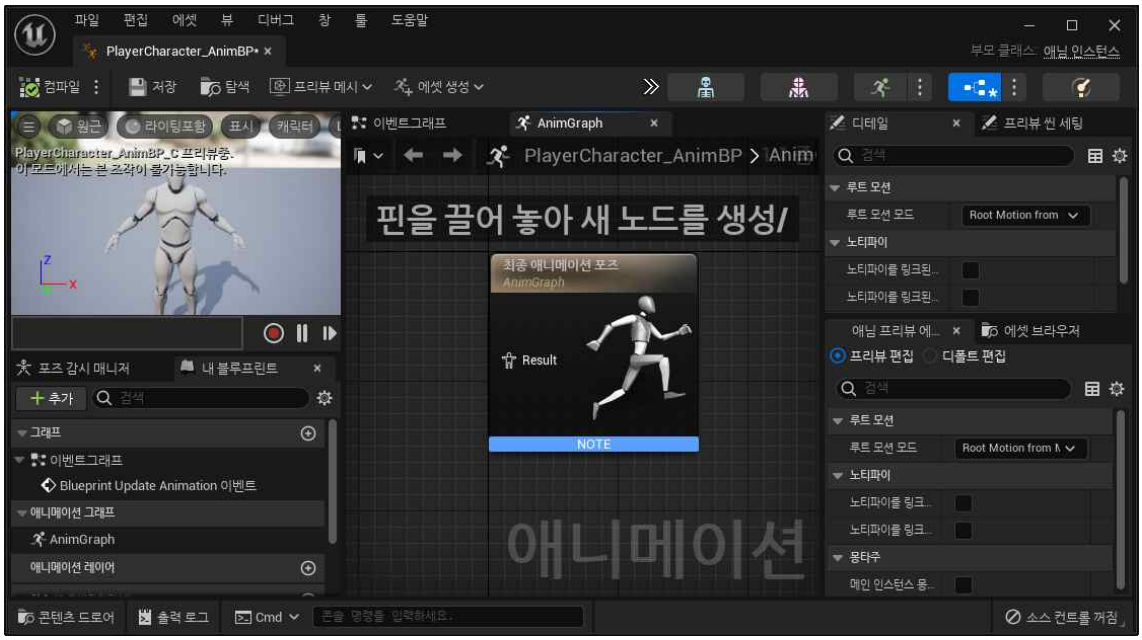
그리고, 그 아래의 **애님 클래스(Anim Class)** 속성을 찾자. 속성값이 **없음**으로 되어 있을 것이다. 이곳에

PlayerCharacter_AnimBP를 선택하여 지정하자.



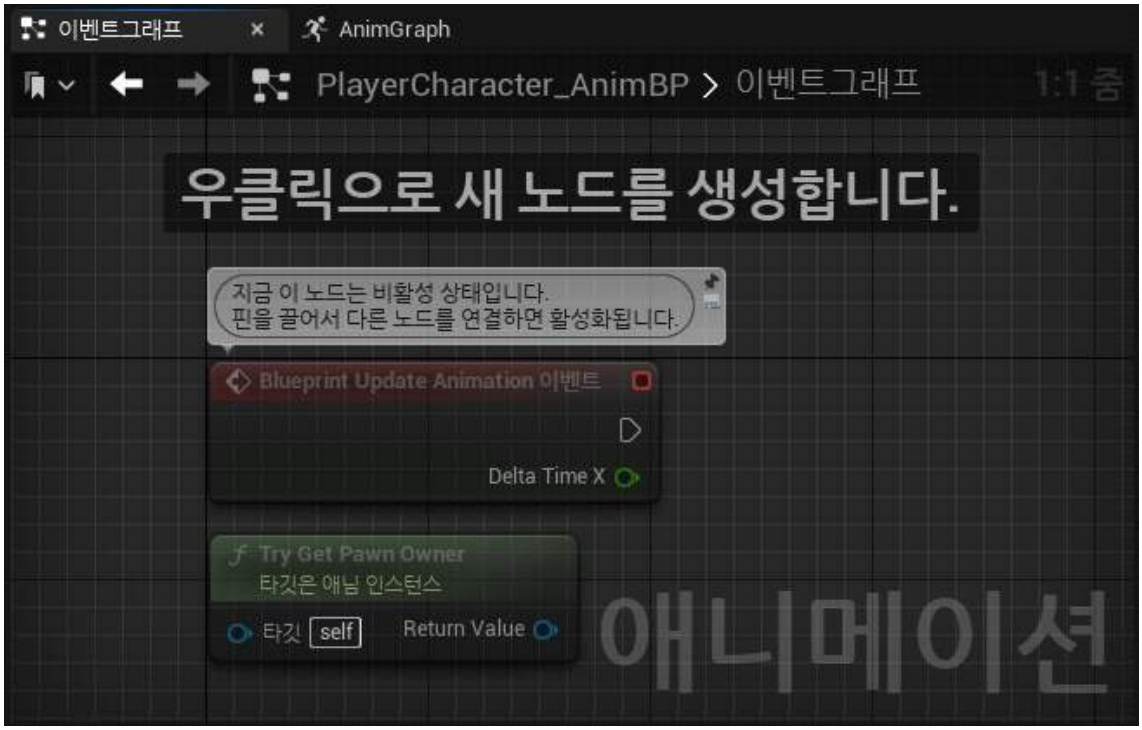
컴파일하고 저장하자.

7. 이제, **PlayerCharacter_AnimBP**를 더블클릭하여 애니메이션 블루프린트 에디터를 열자.



8. 왼쪽 아래의 **내 블루프린트** 탭에서 **변수** 영역에서 오른쪽 +를 클릭하고 변수를 두 개를 추가하자. 변수 이름은 **IsCrouched**와 **IsJumping**으로 하자. 변수 유형은 둘 다 **부울(Boolean)**로 하자.

9. 애니메이션 블루프린트 에디터의 중간에는 두 개의 탭이 있다. 하나는 **이벤트 그래프** 탭이고 다른 하나는 **AnimGraph** 탭이다. 애니메이션 블루프린트를 작성하는 것은 이 두 탭에서의 노드 네트워크를 제작하는 작업에 해당한다.
우리는 먼저 **이벤트 그래프** 탭을 선택하자.
격자맵에는 **Blueprint Update Animation 이벤트** 노드와 **Try Get Pawn Owner** 함수 노드가 배치되어 있을 것이다.



10. **Blueprint Update Animation 이벤트** 그래프를 작성해보자.

먼저, **Try Get Pawn Owner** 함수 노드의 **Return Value** 출력핀을 드래그하고 **MyPlayerCharacter에 형변환** 노드를 배치하자. 배치된 형변환 노드의 **As MyPlayerCharacter** 출력핀을 드래그하고 **Get IsCrouched(앉음 여부)** 노드를 배치하자.

그다음, 내 블루프린트 탭에서 **IsCrouched** 변수를 드래그해서 **Set** 노드를 배치하자. 그리고, 이전의 **Get IsCrouched** 노드의 출력을 **Set** 노드의 입력핀에 연결하자.

그다음, **Blueprint Update Animation 이벤트** 노드의 실행핀을 형변환 노드의 입력 실행핀에 연결하고, 형변환 노드의 출력 실행핀을 **Set** 노드의 실행핀에 연결하자.

위의 그래프를 살펴보자.

키보드 **Ctrl** 키를 누르면 캐릭터 클래스의 **Crouch**가 호출되고 캐릭터 무브먼트 컴포넌트의 변수인 **IsCrouched**가 **true**가 된다. 키를 떼면 **UnCrouch**가 호출되고 **IsCrouched**가 **false**가 된다. 따라서 애니메이션을 위해서는 캐릭터 클래스의 **IsCrouched**의 현재의 값을 알아야 한다. 위의 그래프는 캐릭터 클래스의 **IsCrouched**를 읽어와서 커스텀 플레이어 폰 클래스의 **IsCouched** 변수에 복사한다.

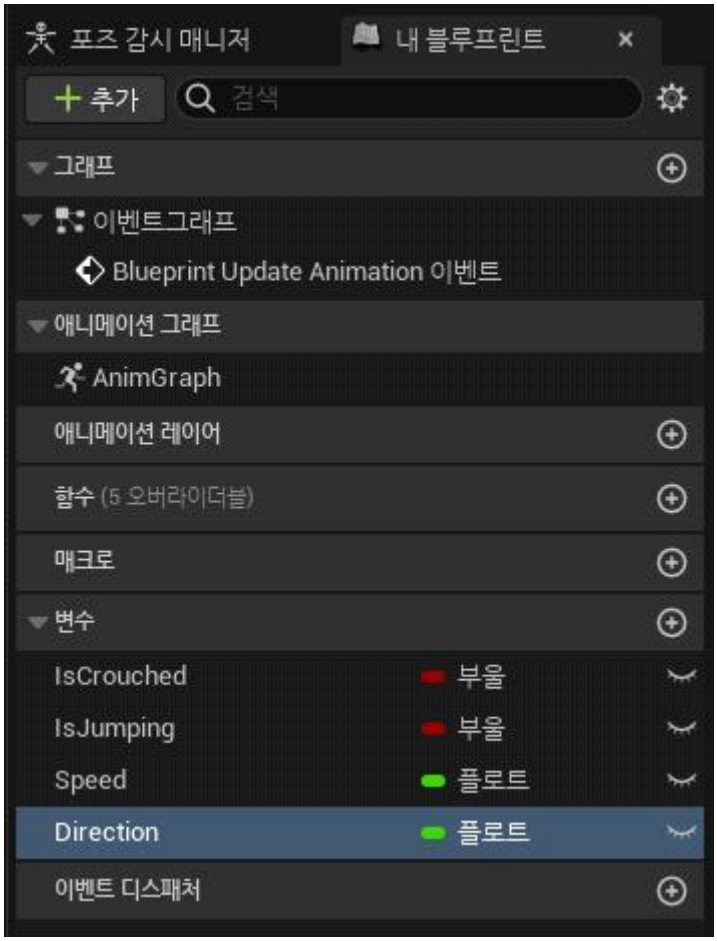
11. 형변환 노드의 **As MyPlayerCharacter** 출력핀을 드래그하고 **Get PressedJump(점프 눌림)** 노드를 배치하자.

그다음, 내 블루프린트 탭에서 **IsJumping** 변수를 드래그해서 배치되어 있는 **Get PressedJump** 노드의 출력핀에 드롭하자. **Set IsJumping** 노드가 배치되고 연결될 것이다.

여기서, 변수를 출력핀에 드롭하는 조작법에 대해서 알아보자.
이 방법은 변수의 **Set** 노드를 빈 곳에 배치한 후에 이전의 출력 노드와 연결시키는 작업을 한 번에 수행하는 편리한 방법이다. **내 블루프린트** 탭에서 변수를 드래그할 때에 빈 곳에 드롭하여 **Set** 노드를 배치하는 대신에, 나중에 배치 후에 값을 읽어올 노드의 출력핀에 가져가보면 **Make IsJumping=IsJumping**라는 메시지와 체크 표시가 뜰 것이다. 이 때에 드롭하면 변수의 **Set** 노드가 배치되고 동시에 드롭된 출력핀에서 **Set** 노드로 링크도 서로 연결된다.
만약 변수를 드래그해서 배치된 노드의 출력핀이 아닌 입력핀에 드롭하게 되면 변수의 **Get** 노드가 배치되면서 동시에 **Get** 노드의 출력이 드롭된 입력핀으로 서로 연결된다.
계속 작업을 진행하자.
그다음, 이전의 **Set IsCrouched** 노드와 **Set IsJumping** 노드의 실행핀을 연결하자.
컴파일하고 저장하자.

위의 그래프를 보자. 키보드 **스페이스** 키를 누르면 캐릭터 클래스의 **Jump**가 호출되고 캐릭터 무브먼트 컴포넌트의 변수인 **PressedJump**가 **true**가 된다. 키를 떼면 **StopJumping**가 호출되고 **PressedJump**가 **false**가 된다. 위의 그래프는 캐릭터 클래스의 **PressedJump**를 읽어와서 커스텀 플레이어 폰 클래스의 **IsJumping** 변수에 복사한다.

12. **PlayerCharacter_AnimBP** 애니메이션 블루프린트 에디터에서 계속하자.
내 블루프린트 탭에서 변수를 두 개를 추가하자. 이름을 **Speed**와 **Direction**으로 하자. 유형은 모두 **플로트**로 하자.



13. 속력을 구해서 **Speed** 변수에 지정하자.

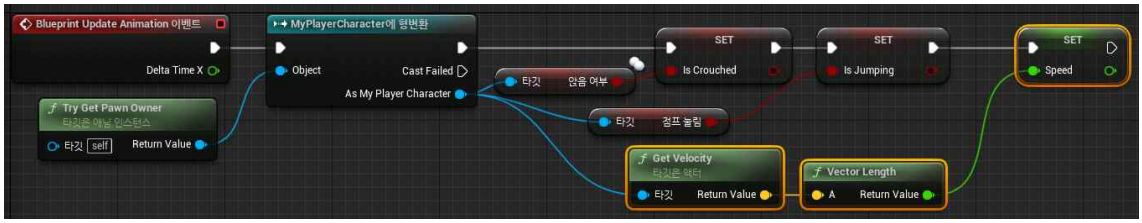
Blueprint Update Animation 이벤트 그래프에서, 형변환 노드의 **As MyPlayerCharacter** 출력핀을 드래그하고 **Get Velocity** 노드를 배치하자.

그다음, **Get Velocity** 노드의 출력핀을 드래그하고 **VectorLength** 함수를 배치하자.

그다음, **내 블루프린트** 탭에서 **Speed** 변수를 드래그해서 배치되어 있는 **VectorLength** 노드의 출력핀에 드롭하자.

그다음, **Set IsJumping** 노드의 실행핀을 **Set Speed** 노드의 실행핀에 연결하자.

우리는 **GetVelocity**로 속도를 계산하고 **VectorLength**로 속도의 길이를 계산하여 속력을 구하였다.

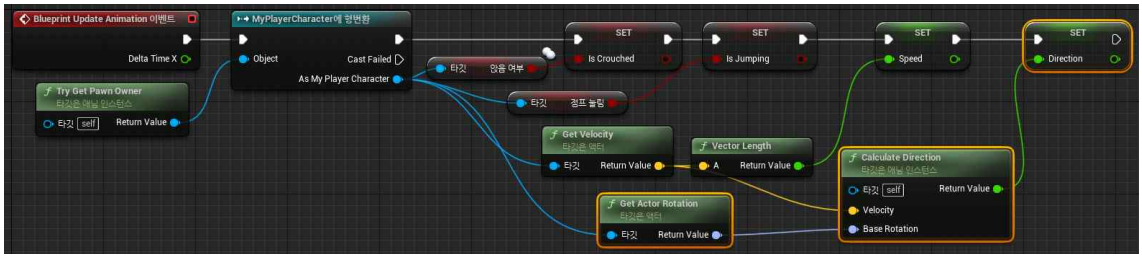


14. 움직이는 방향을 구해서 **Direction**에 지정하자.

Blueprint Update Animation 이벤트 그래프에서, 형변환 노드의 **As MyPlayerCharacter** 출력핀을 드래그하고 **GetActorRotation** 노드를 배치하자.

그다음, **GetActorRotation** 노드의 출력핀을 드래그하고 **Calculate Direction** 함수를 배치하자.

그다음, 이전에 배치된 **Get Velocity** 노드의 출력핀을 드래그해서 **Calculate Direction** 노드의 **Velocity** 입력핀에 연결하자.
그다음, **내 블루프린트** 탭에서 **Direction** 변수를 드래그해서 배치되어 있는 **Calculate Direction** 노드의 출력핀에 드롭하자.
그다음, **Set Speed** 노드의 실행핀을 **Set Direction** 노드의 실행핀에 연결하자.



위의 그래프를 보자. **CalculateDirection**은 바라보는 방향과 움직이는 방향 사이의 각도를 반환한다. 반환되는 값의 범위는 [-180,180]이다. 이 노드의 출력값은 흔히 블렌드 스페이스의 방향값으로 사용된다. **BaseRotation** 입력핀에는 **GetActorRotation**으로 현재의 액터의 회전 방향을 구해서 넣어주면 된다. **Velocity** 입력핀에는 **GetVelocity**를 호출하여 현재의 움직임 벡터를 구해서 넣어주면 된다. 컴파일하고 저장하자.

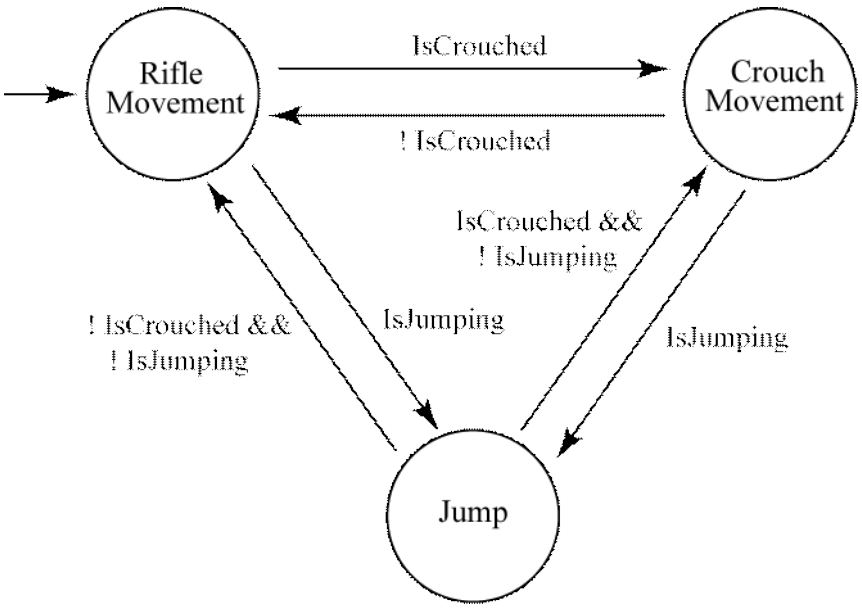
이 절에서는 애니메이션 블루프린트를 만드는 방법을 학습하였다.

6. 애님그래프 만들기

이 절에서 애님그래프에 대해서 학습한다.

애니메이션 블루프린트 에디터에 있는 두 번째 탭인 **AnimGraph** 탭에서는 **AnimGraph**를 작성한다. **AnimGraph**는 스켈레탈 메시의 최종 포즈를 계산하는 그래프이다. 이 그래프는 여러 애니메이션 시퀀스 애셋이나 블렌드 스페이스 애셋으로부터 현재 프레임에서의 최종 포즈를 결정한다.
AnimGraph에 대해서 알아보자.
AnimGraph에서는 블루프린트 노드 그래프와는 다른 방법으로 그래프를 작성한다.
블루프린트 그래프에서는 붉은색의 이벤트 노드를 시작으로 오른쪽으로 실행 흐름이 진행되는 그래프의 형태이다. 그러나 애님그래프에서는 가장 오른쪽에 **최종 애니메이션 포즈** 노드로 연결될 포즈를 결정하는 그래프의 형태이다.
하나의 애니메이션 애셋으로부터 최종 포즈를 결정할 수 있는 경우라면 애니메이션 애셋의 재생 노드를 배치하고 그 출력을 **최종 애니메이션 포즈** 노드로 연결해주면 된다. 그러나 여러 애니메이션 애셋으로부터 알고리즘에 따라서 최종 포즈를 결정해야 하는 경우라면 애니메이션 스테이트 머신을 사용해야 한다.
AnimGraph 작성에서 가장 중요한 내용은 **애니메이션 스테이트 머신**이다. 애니메이션 스테이트 머신은 한 스켈레탈 메시의 애니메이션 흐름을 여러 스테이트의 연결로 시각적으로 구성하는 기법이다. 스테이트 간의 연결에서의 흐름은 전이 규칙(transition rule) 에 의해서 제어된다.
간단한 애니메이션 스테이트 머신의 예를 들어보자.
예를 들어 대기, 걷기, 뛰기의 세 스테이트를 정의하자. 연결된 폰은 항상 모든 스테이트 중에서 한 스테이트에 있게 된다. 그 스테이트에 있는 동안에 스테이트에 해당하는 애니메이션을 재생한다. 걷기 스테이트에 있다면 걷기 애니메이션을 재생하게 된다. 현재 스테이트에서 전이 규칙이 만족되면 다른 스테이트로 전이한다. 현재의 폰이 걷기 스테이트에 있다가 이동 속도가 더 빨라진다면 뛰기 스테이트로 전이할 것이다.
각 스테이트는 다시 **AnimGraph**의 형식으로 작성된다. 가장 오른쪽에는 **출력 애니메이션 포즈** 노드가 있다. 포즈를 결정하는 그래프를 작성하여 노드의 포즈 입력핀에 연결해주면 된다. 각 스테이트에서는 단일 애니메이션 애셋을 재생해도 되고 다시 애니메이션 스테이트 머신을 만들어도 된다.

우리가 이제부터 구현할 스테이트 머신을 알아보자.
최종 애니메이션 포즈 노드에 입력될 최종 포즈는 애니메이션 스테이트 머신에 의해서 결정되도록 하자. 애니메이션 스테이트 머신은 다음과 같은 모습으로 작성하자.



스테이트 머신에는 3개의 스테이트가 있다. **RifleMovement**는 총을 들고 서서 이족 보행으로 움직이는 스테이트이다. **CrouchMovemnt**는 쭈그리고 앉아서 이족 보행으로 움직이는 스테이트이다. **Jump**는 점프하는 스테이트이다. **Jump** 스테이트에 대해서는 단일 애니메이션 애셋을 재생하면 된다. 그 외의 두 스테이트에서는 여러 애니메이션 애셋을 사용하여 블렌드 스페이스를 제작하여 이를 재생할 것이다.

각 스테이트 간의 전이를 위해서 두 변수를 도입하자. **Ctrl** 키를 누르는 동안은 쭈그리고 앉게 되며 키를 떼기 전까지 **IsCrouched**가 **true**가 된다. 또한 점프 키가 눌러진 경우에는 **IsJumping**이 **true**가 된다. **IsCouched** 가 **true**이면 **CrouchMovemnt**로 전이하고 **IsJumping**이 **true**이면 **Jump**로 전이하자. 둘다 **false**이면 **RifleMovement**로 전이하자.

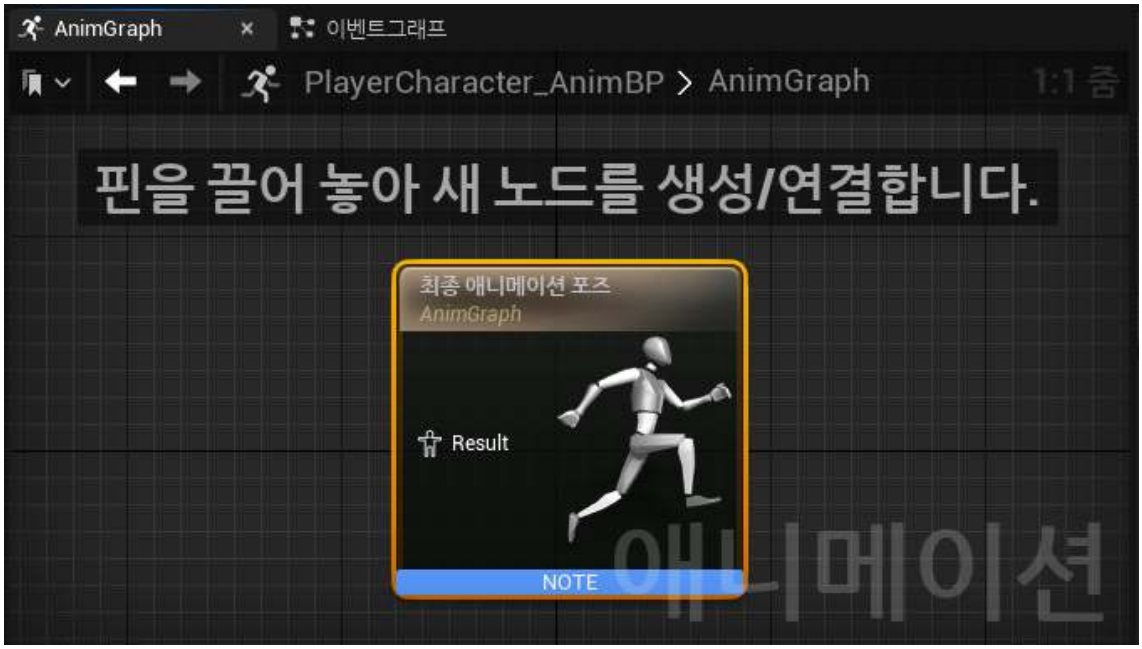
이제부터 예제를 통해서 학습해보자

<참고> **AnimGraph**에 대한 자세한 내용은 다음의 링크를 참조하자.
https://docs.unrealengine.com/graphing-in-animation-blueprints-in-unreal-engine/

---

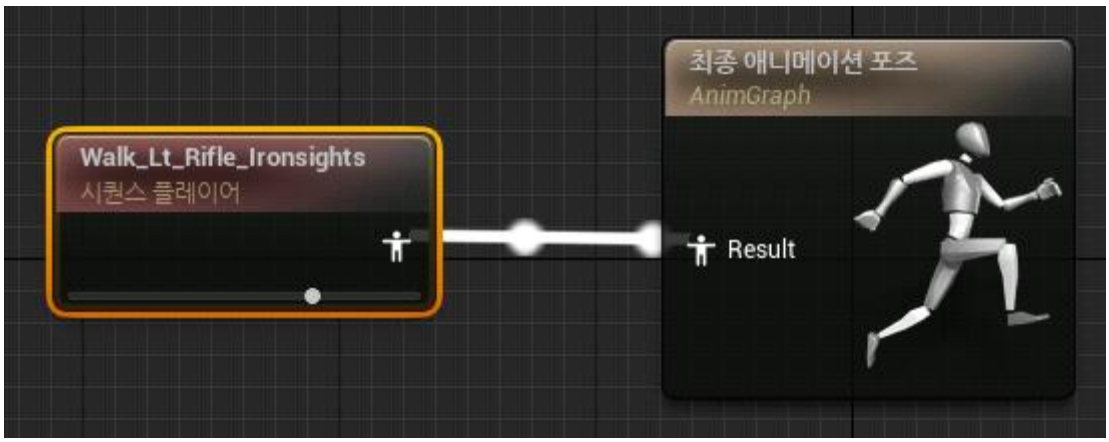
1. 이전 예제의 프로젝트 `Pcharprepare`에서 이어서 계속하자.

2. 애니메이션 블루프린트 애셋인 **PlayerCharacter_AnimBP**를 더블클릭하여 애니메이션 블루프린트 에디터를 열자.
그다음, **AnimGraph** 탭을 클릭하자. 그래프에는 **최종 애니메이션 포즈** 노드만 하나 놓여있을 것이다.



노드의 입출력 핀의 모양이 사람 모양의 아이콘으로 되어 있다. 이 아이콘은 애니메이션의 현재 포즈값을 의미한다.

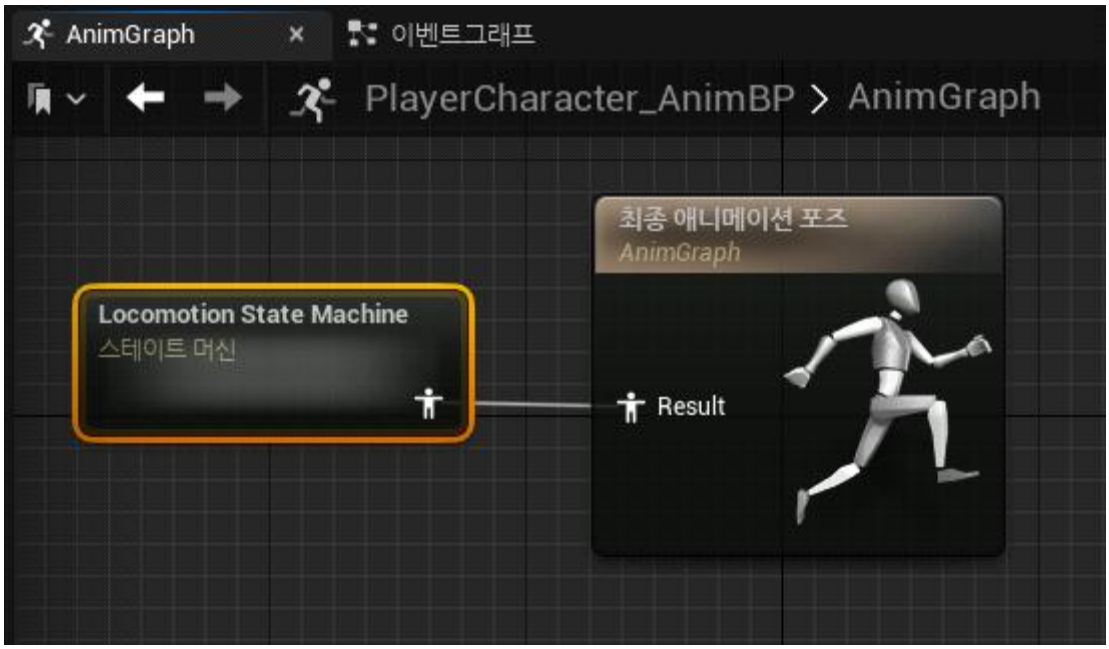
3. 우선, 테스트를 해보자.
오른쪽 아래의 애셋 브라우저에서 **Walk_Lt_Rifle_Ironsights** 애니메이션 시퀀스 애셋을 드래그해서 배치하자. 재생 노드가 배치될 것이다. 재생 노드의 포즈 출력핀을 **최종 애니메이션 포즈** 노드의 입력 핀에 연결하자. 컴파일하고 저장하자.



4. 레벨 에디터에서 플레이해보자. 움직여보자. 항상 걷는 애니메이션이 재생될 것이다. **최종 애니메이션 포즈** 노드의 입력 포즈가 항상 걷기 애니메이션 시퀀스의 재생 결과이기 때문이다.

이제 테스트를 해 보았다.
이제 다시 **Walk_Lt_Rifle_Ironsights** 재생 노드를 제거하고 원래 상태로 돌려놓자.

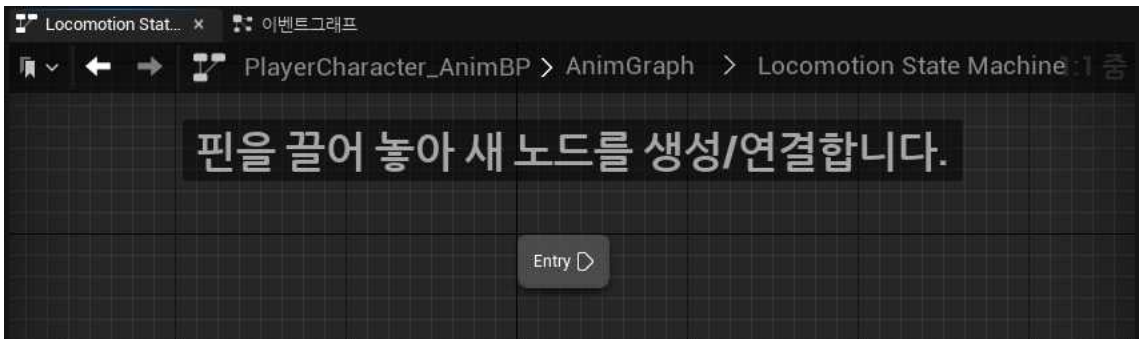
5. 격자판의 빈 곳에서 우클릭하고 **새 스테이트 머신 추가...(Add New State Machine...)** 노드를 검색하여 배치하자.
배치된 노드를 클릭하고 이름을 **Locomotion State Machine**으로 수정하자.
그다음, **Locomotion State Machine**의 오른쪽 출력핀을 드래그해서 **최종 애니메이션 포즈** 노드의 왼쪽 입력핀에 연결하자.



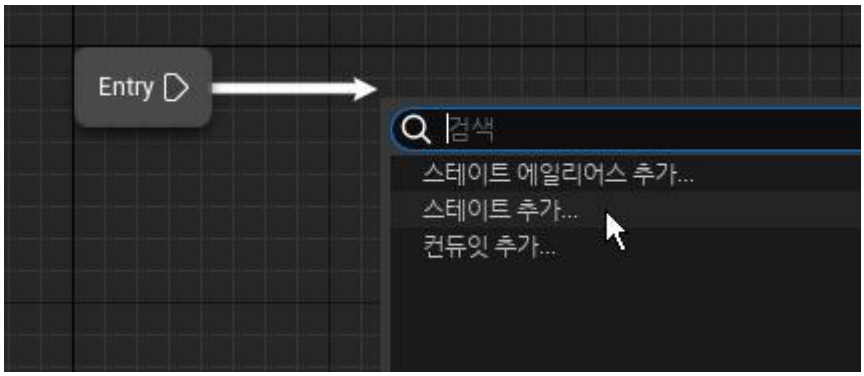
스테이트 머신의 출력 포즈를 최종 애니메이션 포즈로 결정하도록 작성하였다.

6. 이제 스테이트 머신을 작성하면 된다.
Locomotion State Machine 스테이트 머신 노드를 더블클릭하자.
격자판이 다음과 같이 바뀔 것이다. **AnimGraph** 내의 **Locomotion State Machine** 스테이트 머신 그래프를

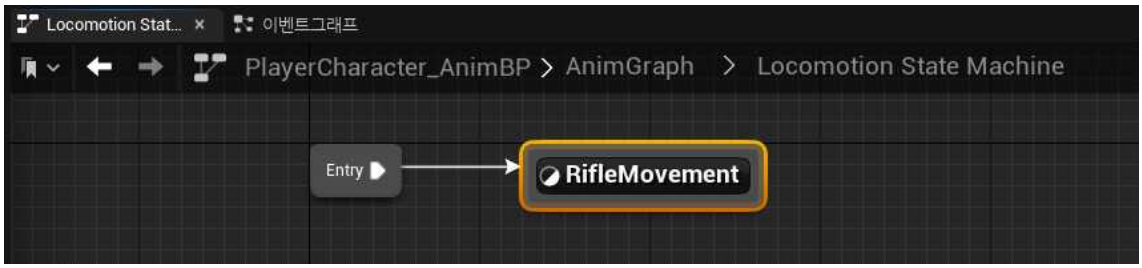
작성하고 있음을 알 수 있다. 디폴트로 **Entry** 노드만 배치되어 있다.



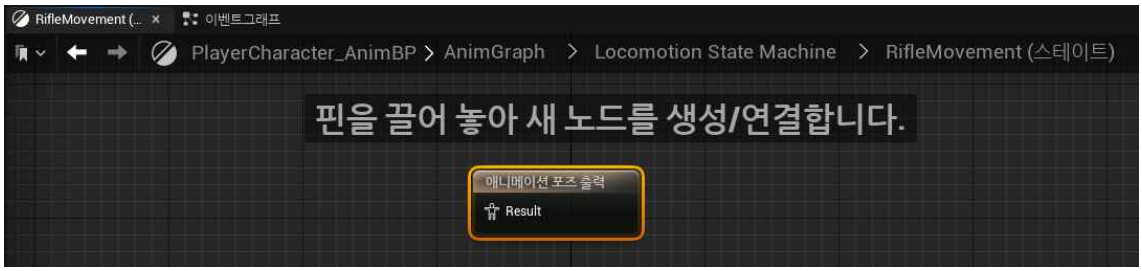
7. **Entry** 핀을 드래그하고 액션선택 드롭다운 메뉴에서 **스테이트 추가**를 선택하자.



8. 추가된 스테이트의 이름을 **RifleMovement**라고 수정하자.



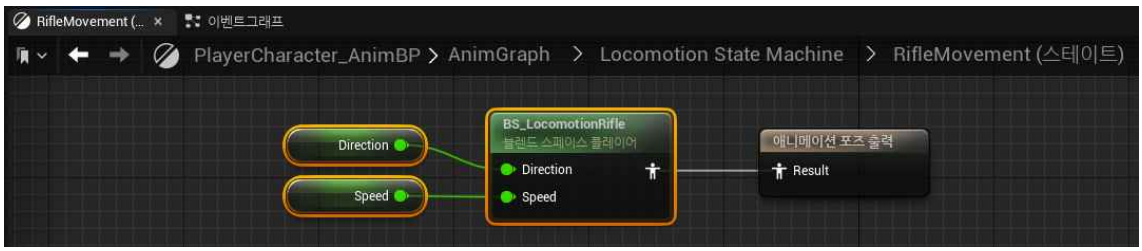
9. **RifleMovement** 스테이트를 더블클릭하자. 다음과 같이 스테이트 그래프가 나타난다. 처음에는 **출력 애니메이션 포즈** 노드만 있다.



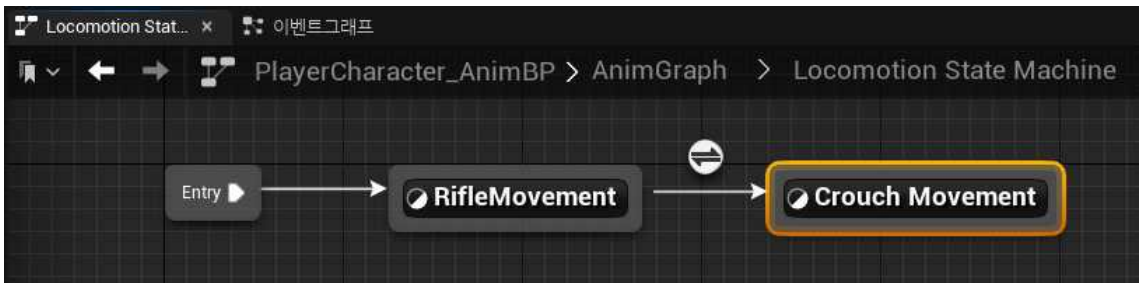
이 스테이트에 있는 경우에 플레이할 애니메이션을 결정해서 **출력 애니메이션 포즈** 노드에 넣어주면 된다.

10. 오른쪽 아래의 **애셋 브라우저**로 가자. 우리가 이전에 만들어둔 **BS_LocomotionRifle** 블렌드 스페이스 애셋이 있다. 이것을 드래그해서 그래프에 배치하자. 그래프에 **BS_LocomotionRifle** 블렌드 스페이

스 재생 노드가 배치될 것이다. 노드의 출력 포즈 핀을 **출력 애니메이션 포즈** 노드의 입력 **Result** 핀에 연결하자.
그다음, 왼쪽 **내 블루프린트** 탭에서 변수 **Direction**을 드래그해서 **BS_LocomotionRifle** 노드의 **Direction** 입력핀에 드롭하자. **Get Direction** 노드가 배치되고 연결될 것이다.
그다음, 변수 **Speed**를 드래그해서 **BS_LocomotionRifle** 노드의 **Speed** 입력핀에 드롭하자. **Get Speed** 노드가 배치되고 연결될 것이다.
컴파일하고 저장하자.

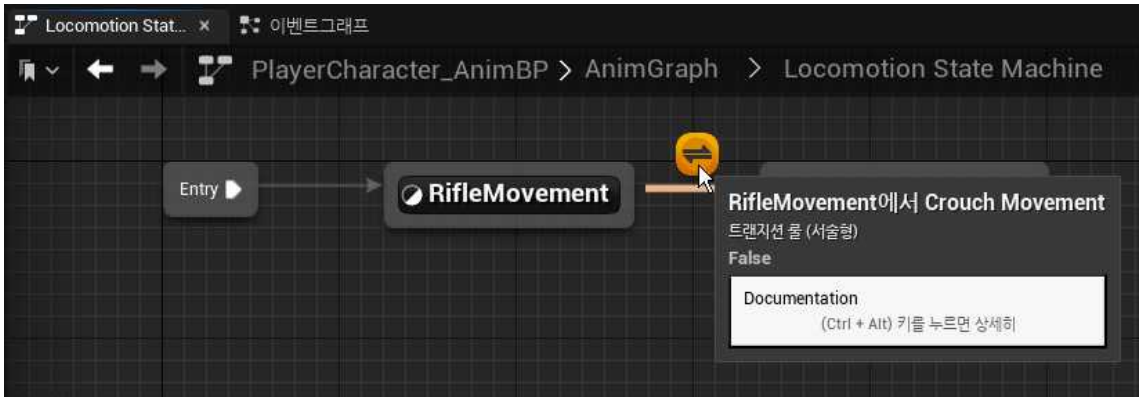


11. 다시 **Locomotion State Machine** 스테이트 머신 그래프로 돌아가자.
그다음, **RifleMovement** 상태 노드를 드래그하고 액션선택 창에서 **스테이트 추가**를 선택하여 새 스테이트 노드를 추가하자. 이름은 **Crouch Movement**라고 하자.

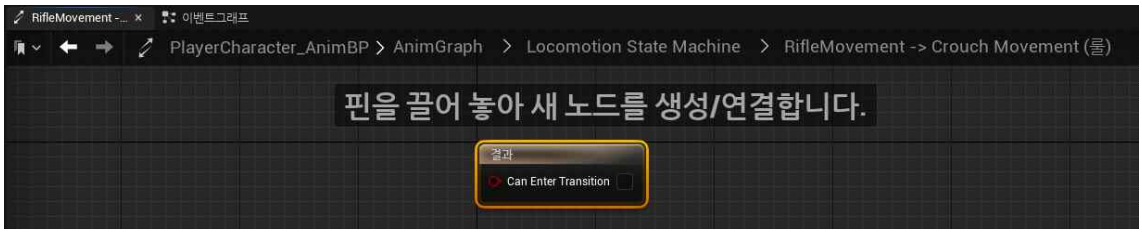


스테이트 노드를 살펴보자.
스테이트 노드의 노드명 부분을 드래그하면 노드를 이동할 수 있다. 노드의 두꺼운 테두리에서 드래그를 시작하면 밖으로 나가는 상태전이와 더불어 나간 후의 노드를 새로 만들어 배치할 수 있다.
여러 그래프 사이를 쉽게 이동하는 방법을 알아보자.
왼쪽의 **내 블루프린트** 탭에서 **애니메이션 그래프** 영역을 보면 **AnimGraph**의 구조를 계층 구조로 나타내주고 있다. 여기서 이동할 대상 그래프를 더블클릭하면 된다.
또한, 계층 구조에서 현재 자신의 상위 수준의 그래프로 이동할 때에는 격자탭의 제목으로 보이는 `PlayerCharacter_AnimBP > AnimGraph > Locomotion State Machine` 식의 현재 위치 표시에서 이동할 대상 위치를 한번 클릭하면 상위 수준 그래프로 이동한다.
또한, 계층 구조에서 현재 자신의 하위 수준의 그래프로 이동할 때에는 그래프 내에서의 해당 노드를 더블클릭하면 된다.

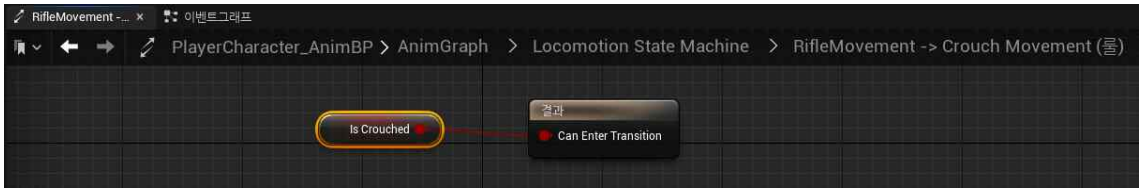
12. 이제 **RifleMovement** 상태에서 **CrouchMovement** 상태로의 전이가 만들어졌다. 전이 화살표 위에 아이콘을 전이 규칙(transition rule) 노드라고 한다. 이 전이 규칙 노드를 더블클릭하자.



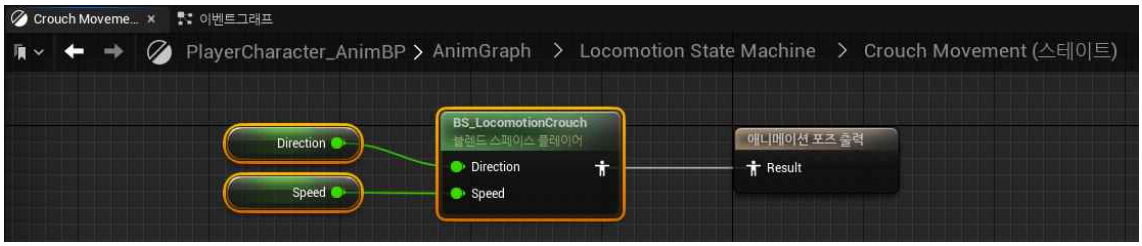
13. 전이 규칙 그래프가 열린다. 디폴트로 **결과** 노드가 하나 배치되어 있다. 전이 조건을 그래프로 작성해서 그 불리언 값을 결과 노드의 **Can Enter Transition** 입력핀에 연결하면 된다.
RifleMovement 스테이트에 있으면서 해당 스테이트의 애니메이션을 재생하는 동시에 해당 스테이트에서 다른 스테이트로 향하는 모든 전이 규칙 노드를 실행한다. 전이 규칙 노드를 실행할 때에 true가 결과 노드에 입력되는 경우에 다른 스테이트로 전이된다.



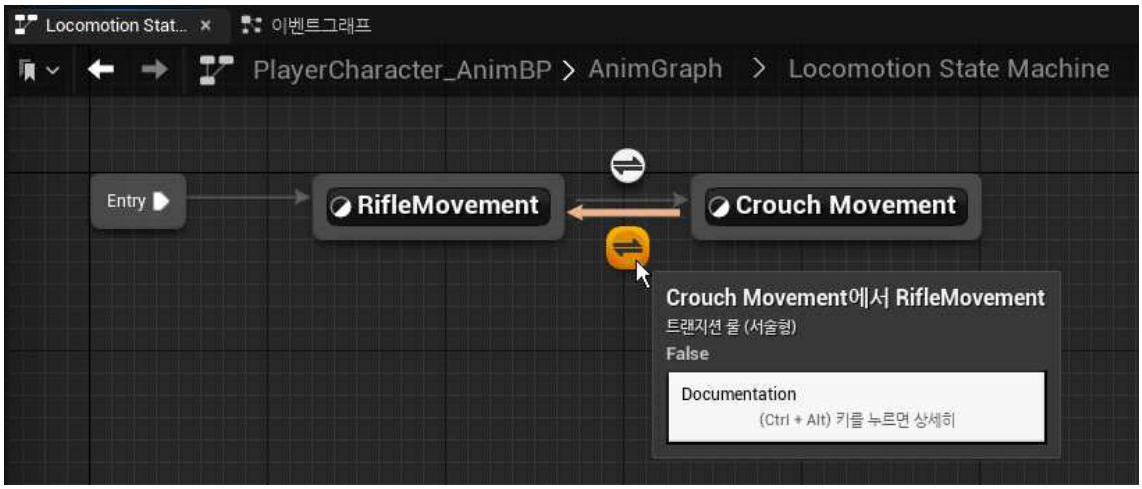
14. 내 블루프린트 탭에서 **IsCrouched** 변수를 드래그해서 **Can Enter Transition** 입력핀에 드롭하자. 컴파일하고 저장하자.



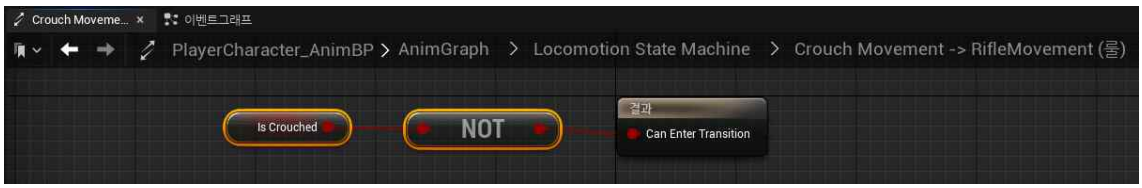
15. 다시 **Locomotion State Machine** 스테이트 머신 그래프로 돌아가자.
그다음, **CrouchMovement** 상태 노드를 더블클릭하여 상태 그래프로 이동하자.
애셋 브라우저에서 **BS_LocomotionCrouch**를 드래그하여 그래프에 배치하자. **BS_LocomotionCrouch** 블렌드스페이스 플레이어 노드가 배치될 것이다. 노드의 출력 포즈핀을 출력 애니메이션 포즈의 **Result** 핀에 연결하자.
그다음, 내 블루프린트 탭에서 **Speed** 변수를 드래그하여 노드의 **Speed** 입력핀에 드롭하고, 또한 **Direction** 변수를 드래그하여 노드의 **Direction** 입력핀에 드롭하자. 컴파일하고 저장하자.



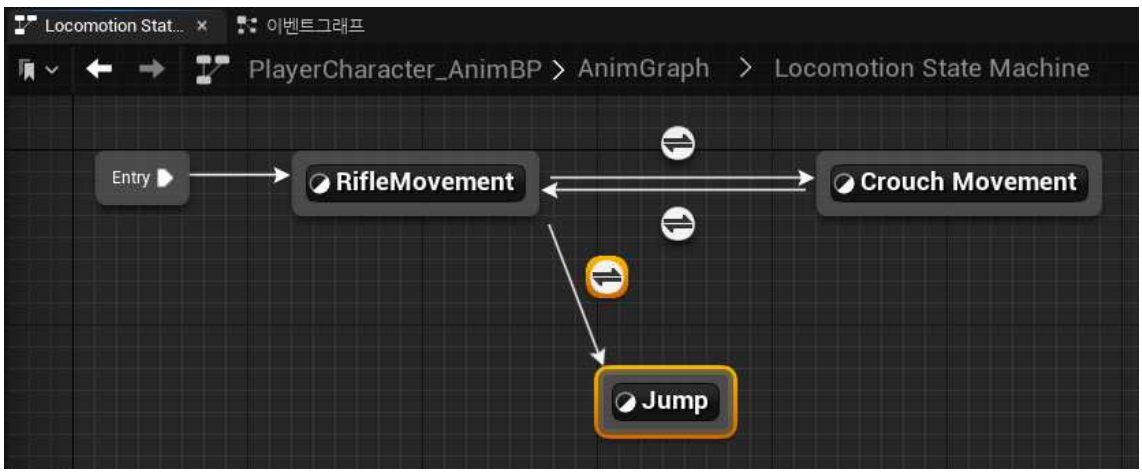
16. 다시 **Locomotion State Machine** 스테이트 머신 그래프로 돌아가자.
그다음, **CrouchMovement** 상태 노드에서 **RifleMovement** 상태 노드로 드래그하여 전이를 추가하자.



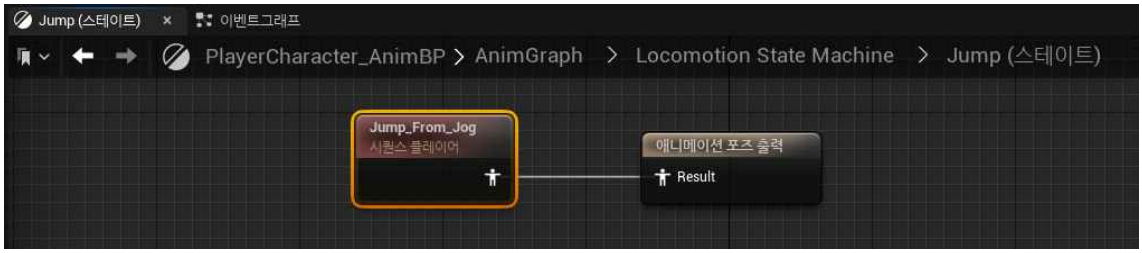
17. 추가된 전이 노드를 더블클릭하여 전이 그래프를 열자.
내 블루프린트 탭에서 **IsCrouched** 변수를 드래그해서 **Get** 노드를 배치하자.
그다음, **Get** 노드의 출력핀을 드래그하고 **NOT Boolean** 노드를 검색하여 배치하자.
그다음, 노드의 출력값을 결과 노드의 입력핀에 연결하자. 컴파일하고 저장하자.



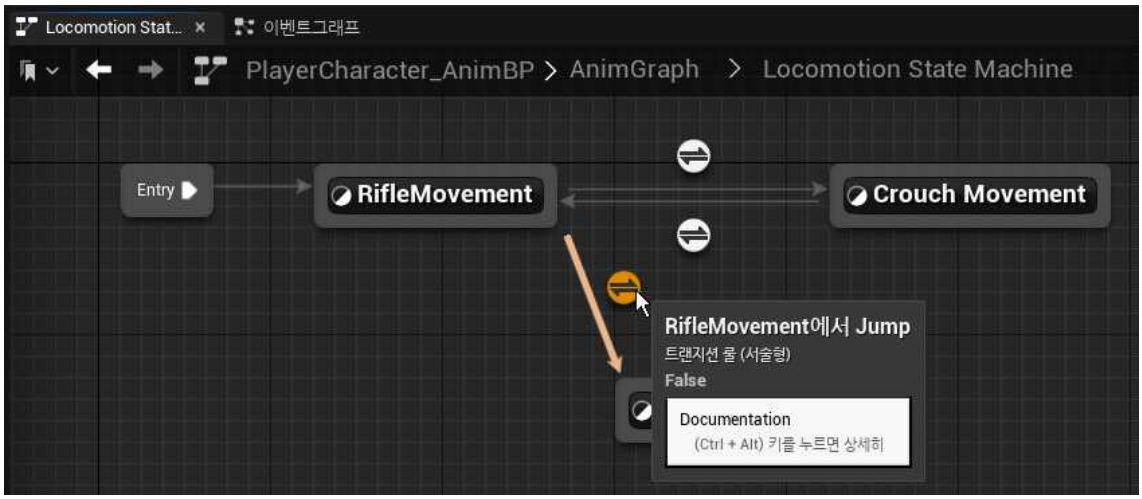
18. 다시 **Locomotion State Machine** 스테이트 머신 그래프로 돌아가자.
그다음, **RifleMovement** 상태 노드에서 드래그하고 액션선택 창에서 **스테이트 추가**를 선택하여 새 상태 노드를 추가하자. 상태 노드의 이름을 **Jump**라고 수정하자.



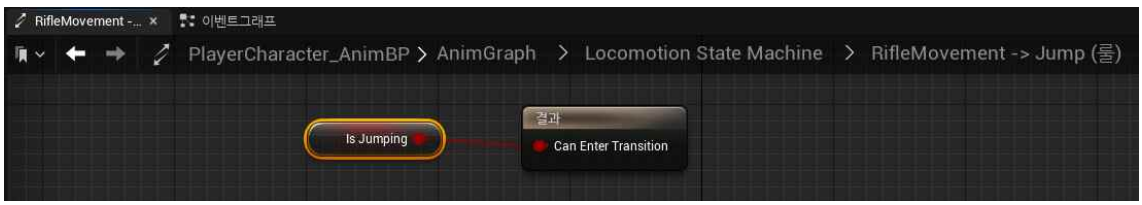
19. 추가된 상태 노드를 더블클릭하여 상태 그래프를 열자.
애셋 브라우저에서 **Jump_From_Jog** 애니메이션 시퀀스를 드래그하여 그래프에 배치하자. **Jump_From_Jog** 재생 노드가 배치될 것이다. 노드의 출력 포즈핀을 **출력 애니메이션 포즈**의 **Result** 핀에 연결하자.



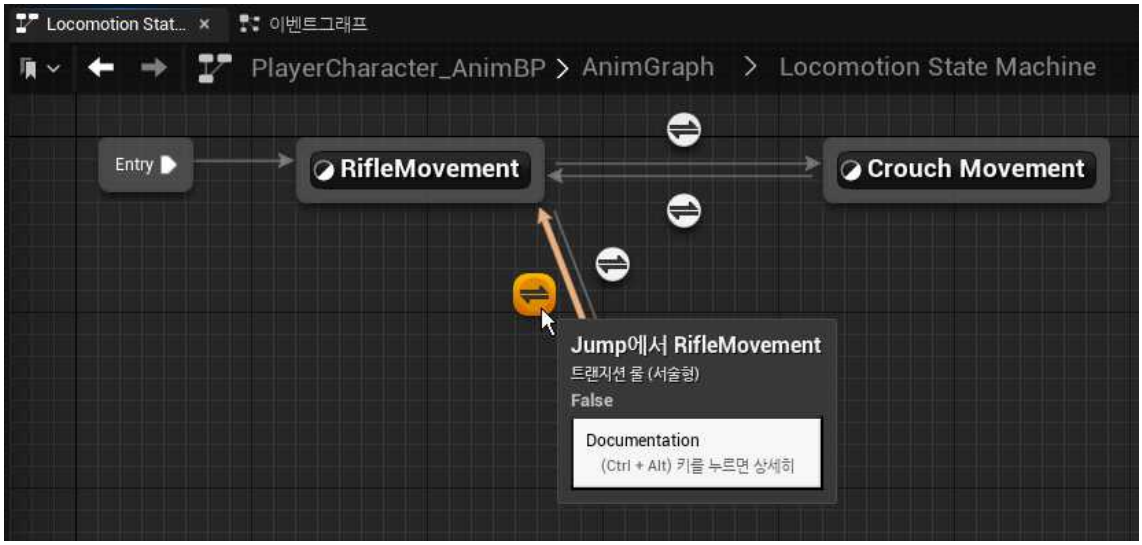
20. 다시 **Locomotion State Machine** 스테이트 머신 그래프로 돌아가자.
RifleMovement 상태 노드에서 **Jump** 상태 노드로의 전이 노드를 더블클릭하여 전이 그래프를 열자.



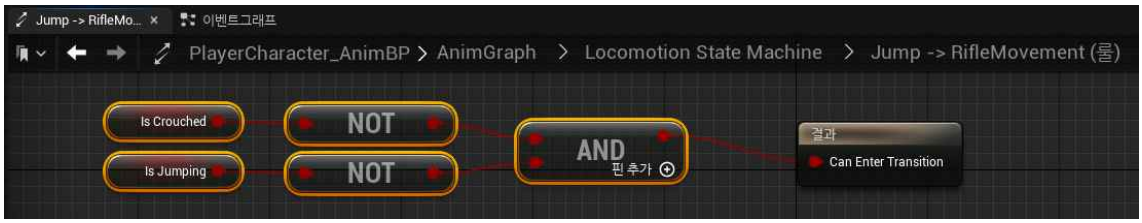
21. 내 블루프린트 탭에서 **IsJumping** 변수를 드래그해서 결과 노드의 **Can Enter Transition** 입력핀에 드롭하자.



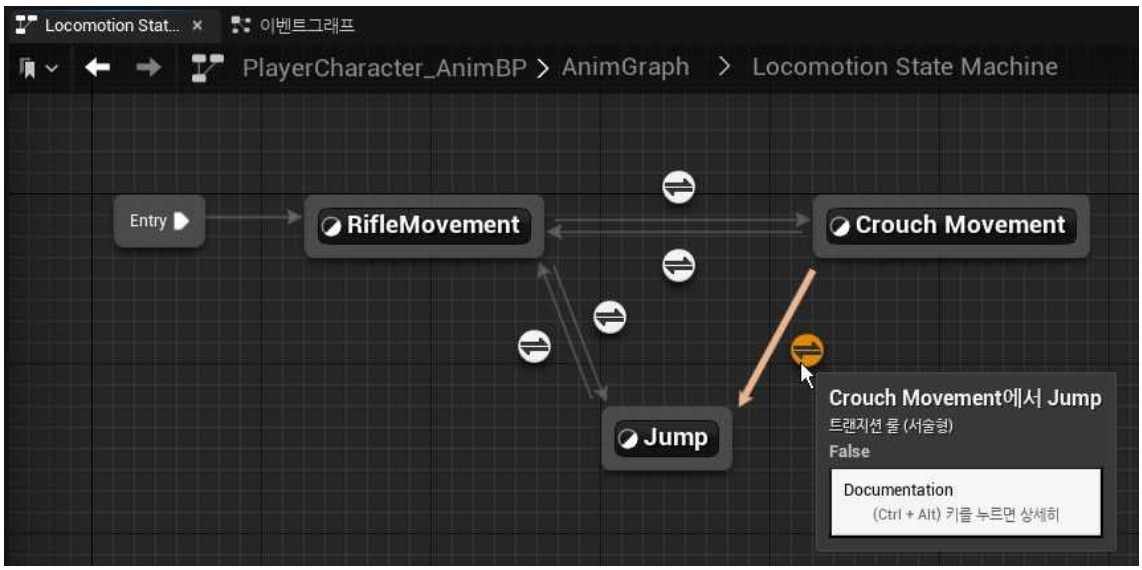
22. 다시 **Locomotion State Machine** 스테이트 머신 그래프로 돌아가자.
Jump 상태 노드에서 **RifleMovement** 상태 노드로의 전이를 추가하자.



23. 전이 노드를 더블클릭하여 전이 그래프를 열자.
내 블루프린트 탭에서 **IsCrounched** 변수와 **IsJumping** 변수를 드래그해서 **Get** 노드를 배치하자.
그다음, 각 노드에 대해서 드래그하여 **NOT Boolean** 노드를 배치하자.
그다음, **AND Boolean** 노드를 배치하고 두 **NOT Boolean** 노드를 연결하자.
그다음, **AND Boolean** 노드의 출력을 결과 노드의 **Can Enter Transition** 입력핀에 연결하자.

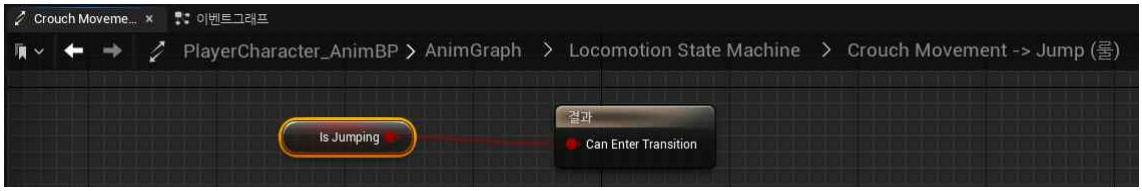


24. 다시 **Locomotion State Machine** 스테이트 머신 그래프로 돌아가자.
CrouchMovement 상태 노드에서 **Jump** 상태 노드로 연결하여 전이를 만들자.

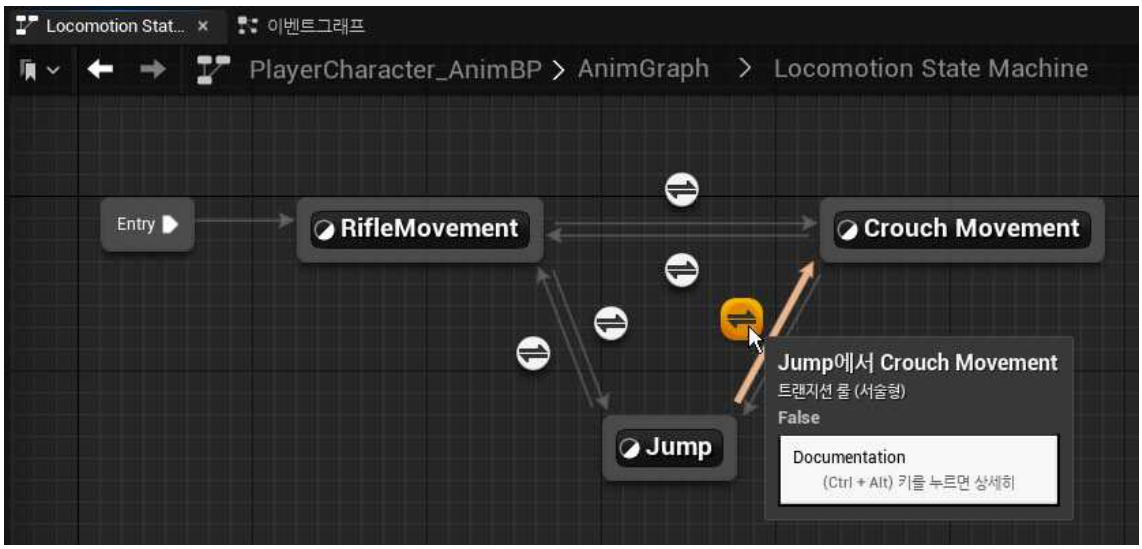


25. 전이 노드를 더블클릭하여 전이 그래프를 열자.

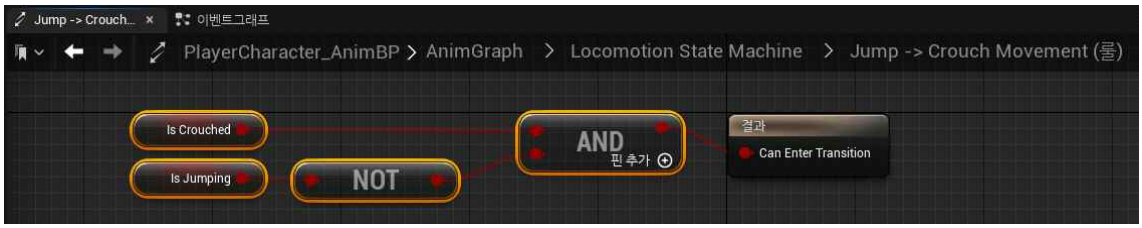
내 블루프린트 탭에서 **IsJumping** 변수를 드래그해서 결과 노드의 **Can Enter Transition** 입력핀에 드롭하자.



26. 다시 **Locomotion State Machine** 스테이트 머신 그래프로 돌아가자.
Jump 상태 노드에서 **CrouchMovement** 상태 노드로 연결하여 전이를 만들자.



27. 전이 노드를 더블클릭하여 전이 그래프를 열자.
내 블루프린트 탭에서 **IsCrouched** 변수와 **IsJumping** 변수를 드래그해서 **Get** 노드를 배치하자.
그다음, **Get IsJumping** 노드에서 드래그하여 **NOT Boolean** 노드를 배치하자.
그다음, **AND Boolean** 노드를 배치하고 **Get IsCrouched** 노드의 출력과 **NOT Boolean** 노드의 출력을 **AND Boolean** 노드의 입력으로 연결하자.
그다음, **AND Boolean** 노드의 출력을 결과 노드의 **Can Enter Transition** 입력핀에 연결하자.



이제 모두 완성되었다. 컴파일하고 저장하자.

28. 이제, 레벨 에디터에서 플레이해보자.
먼저, 스페이스 바를 눌러 점프를 테스트해보자.
그다음, 왼쪽 **Ctrl** 키를 눌러 쭈그려 앉기를 테스트해보자.
그다음, 점프 중인 상태에서나 쭈그려 앉은 상태에서 이동해보자. 두 동작이 모두 합쳐져서 진행될 것이다.

그다음, 왼쪽 **Shift** 키를 눌러 스프린트로 이동해보자. 빠른 속도로 이동할 것이다.

이 절에서는 애님그래프에 대해서 학습하였다.

□