16_ 타임라인

<제목 차례>

인천대학교 컴퓨터공학부 박종승

1. 개요

이 장에서는 타임라인에 대해서 학습한다.

레벨에는 다양한 액터들이 배치된다. 이들 중에는 정적인 액터와 더불어 움직임이 있는 동적인 액터들이 있다. 움직이는 동적인 액터 중에는 정해진 시간 간격에 맞추어 움직임을 반복하는 액터들이 있다. 이러한 액터의 움직임을 쉽게 구현할 수 있도록 도와주는 도구가 있다. 바로 **타임라인**(timeline)이다. 타입라인을 사용하면 시간과 값의 변화를 커브 에디터로 쉽게 명시할 수 있다. 타임라인 노드는 실행이 계속해서 흐르면서 편집된 커브에 따라서 타임라인이 재생되도록 해준다.

타임라인의 개념에 대해서 자세히 알아보자.
타임라인의 수평축은 시간(time) 축이다. 왼쪽에서 오른쪽으로 진행하면 시간이 흐르는 방향에 해당한다.
자동문을 생각해보자. 센서가 발동되면 문이 열리기 시작하고 3초 후에는 문이 완전히 열린 후에 정지된다고 해보자. 이 경우 타임라인의 수평축은 0에서 3까지의 범위로 생각하면 된다.
타임라인의 수직축은 해당 시간 시점에서의 값을 나타낸다. 자동문의 폭이 120cm라고하자. 타임라인의 각 초마다 자동문의 상대 이동거리값을 정해주면 된다. 0초에서는 0이고, 1초에서는 40이고, 2초에서는 80이고, 마지막 3초에서는 120이 되도록 하면 된다. 시간의 흐름에 따른 자동문의 상대 위치값이 수학에서의 수식 그래프 모양으로 만들어진다. 이 수식 그래프를 **트랙**(track)이라고 한다. 트랙의 종류에 따라서 수직축이 정해진다. 수직축이 상대 이동거리값인 경우에는 실수값 하나로 표현하면된다. 이를 **플로트** 트랙이라고 한다.
한편, 파리가 날아다니는것과 같이 시간의 흐름에 따라서 위치벡터를 임의로 명시하고자 하는 경우도 있을 것이다. 이 경우에는 실수값 하나로 표현하는 대신에 위치벡터로 표현해야 한다. 측 수직축이 위치벡터가 되어야 한다. 이를 **벡터** 트랙이라고 한다. 사실상 **벡터** 트랙은 **플로트** 트랙의 3개가 합쳐져서 각각 X,Y,Z 좌표값을 표현하는 모양에 해당한다.
이 외에도 **컬러** 트랙, **이벤트** 트랙이 있다.

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

2. 처음으로 타임라인 만들고 사용하기

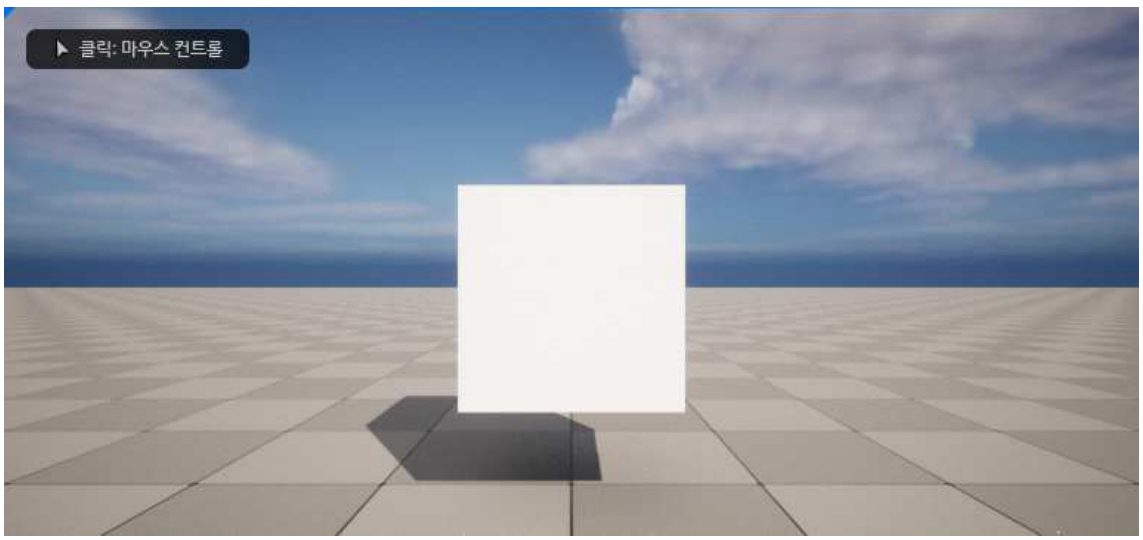
이 절에서는 타임라인에 대해서 학습한다.
예제를 진행하면서 타임라인에 대해서 학습해보자.

---

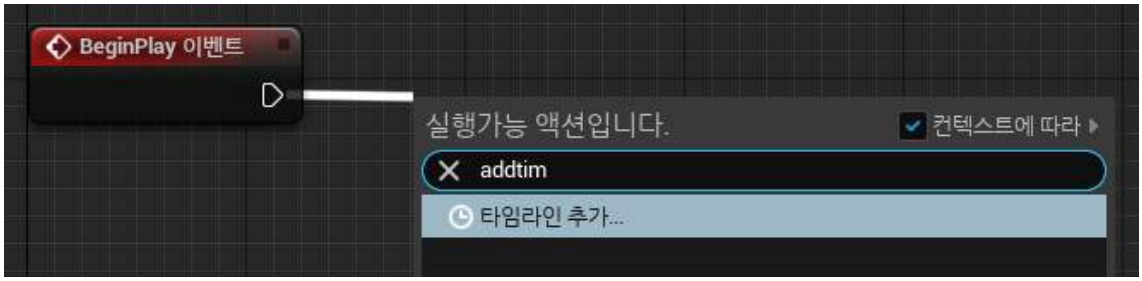
1. 새 프로젝트 **Ptimeline**을 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 Ptimeline으로 입력하고 **생성** 버튼을 클릭하자. 프로젝트가 생성되고 언리얼 에디터 창이 뜰 것이다. 창이 뜨면, 메뉴바에서 **파일** » **새 레벨**을 선택하고 **Basic**을 선택하여 기본 템플릿 레벨을 생성하자. 그리고, 레벨 에디터 툴바의 저장 버튼을 클릭하여 현재의 레벨을 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 큐브 액터를 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.

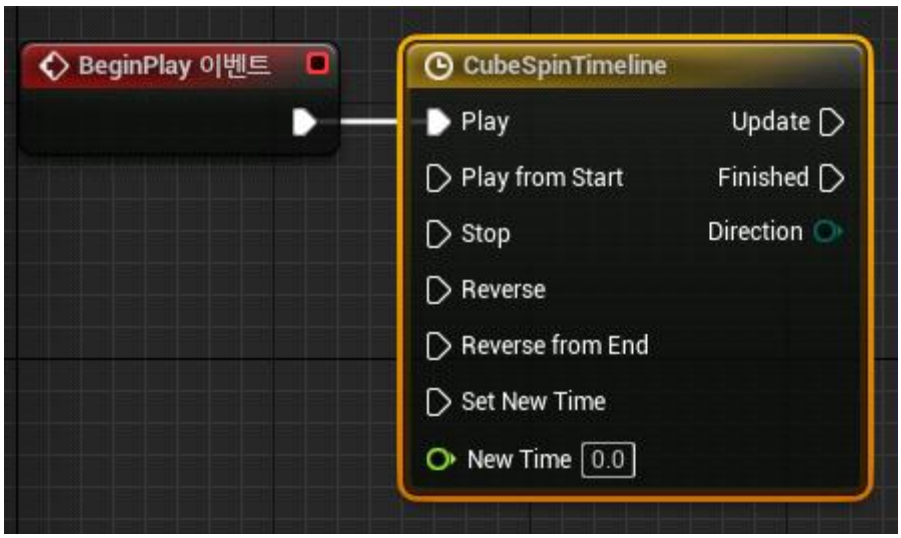
3. 레벨 에디터에서 큐브를 드래그해서 레벨에 배치하자. 위치는 (300,0,100)에 배치하자.
플레이해보자. 움직이지는 않을 것이다.



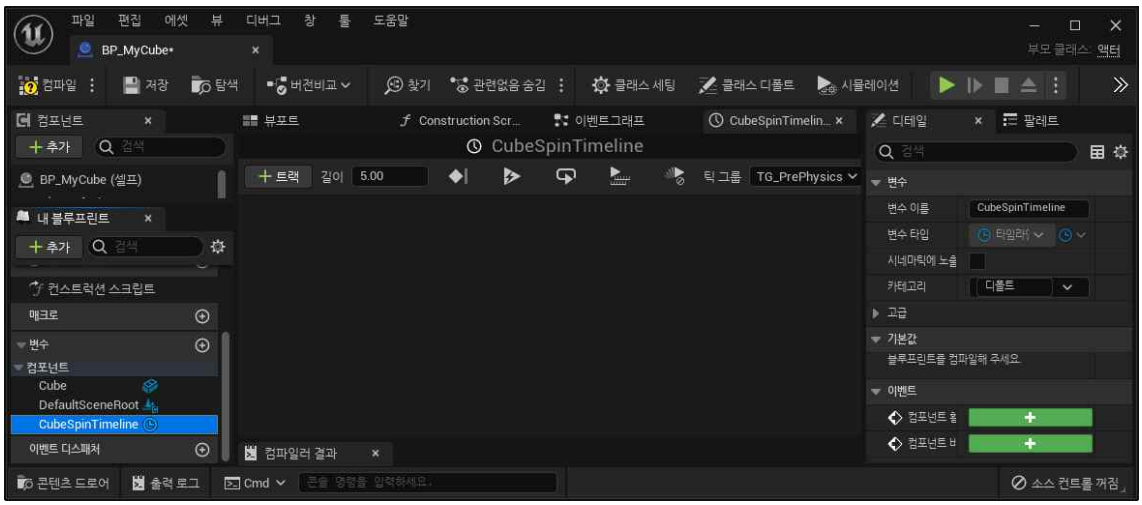
4. **BP_MyCube 블루프린트 에디터**의 이벤트 그래프 탭으로 가자. **BeginPlay** 이벤트 노드의 출력 실행핀을 드래그하고 액션선택 창에서 **AddTimeline**을 검색하여 **타임라인 추가**를 찾아 선택하자.



5. 타임라인의 이름은 **CubeSpinTimeline**으로 지정하자.

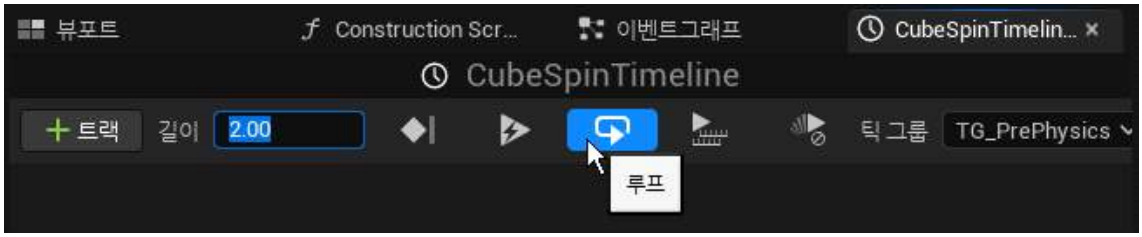


6. 타임라인 노드를 더블클릭하자. 블루프린트 에디터 내에서 **CubeSpinTimeline**이라는 탭이 생긴다. 이 탭이 바로 타임라인 에디터에 해당한다. 아직 트랙이 없는 타임라인이어서 내부가 비어있다.

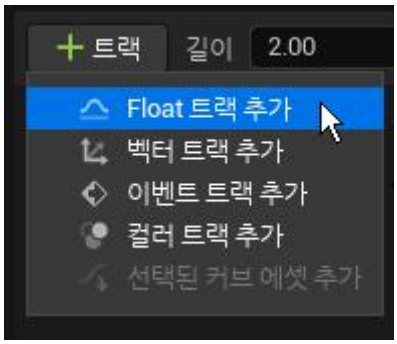


7. 타임라인 탭에 있는 툴바를 보자. **길이** 속성이 있다. **길이** 속성은 타임라인의 수평축의 길이를 초 단위로 나타낸다. 디폴트로 5로 되어 있는데 이것을 2로 수정하자. 이제 우리는 큐브를 2초 동안 움직이도록 할 것이다.

그리고, 그 오른쪽에 루프 체크박스가 있다. 디폴트로 체크해제되어 있는데 이것을 체크하자. 이제 큐브는 2초 동안의 움직임을 계속해서 반복할 것이다.

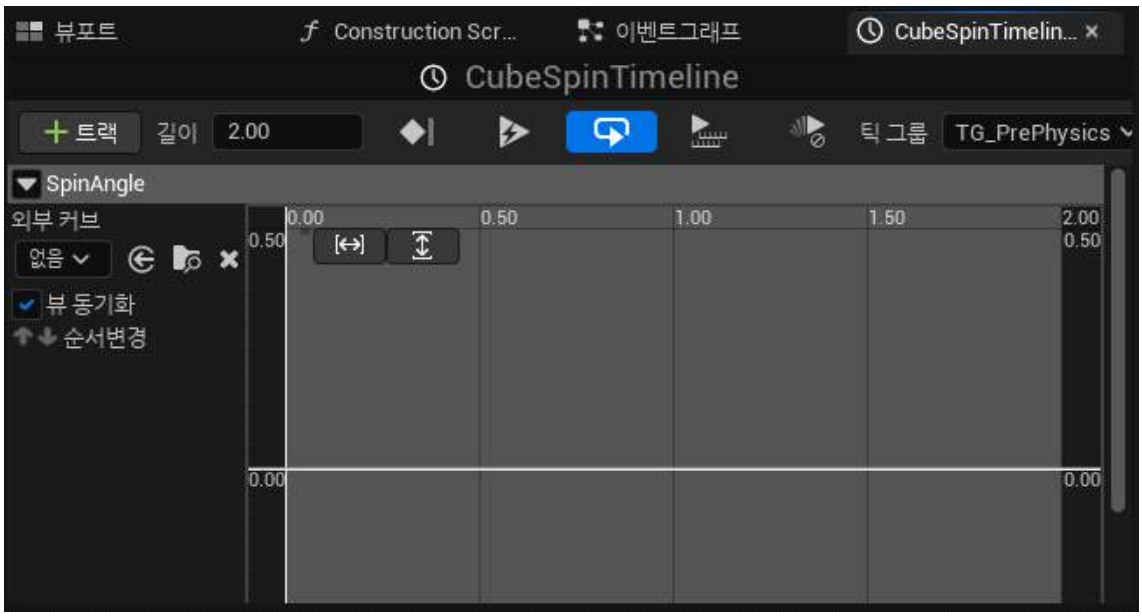


8. 타임라인 툴바의 왼쪽의 **+트랙**을 클릭하자.



목록에 추가할 수 있는 트랙 유형이 나열된다. 트랙의 유형에는 **Float 트랙**, **벡터 트랙**, **이벤트 트랙**, **컬러 트랙**의 네 종류의 트랙이 있다. 이 중에서 **Float 트랙 추가**를 클릭하자.
우리는 가장 간단한 유형인 **Float 트랙**부터 배워보기로 하자.

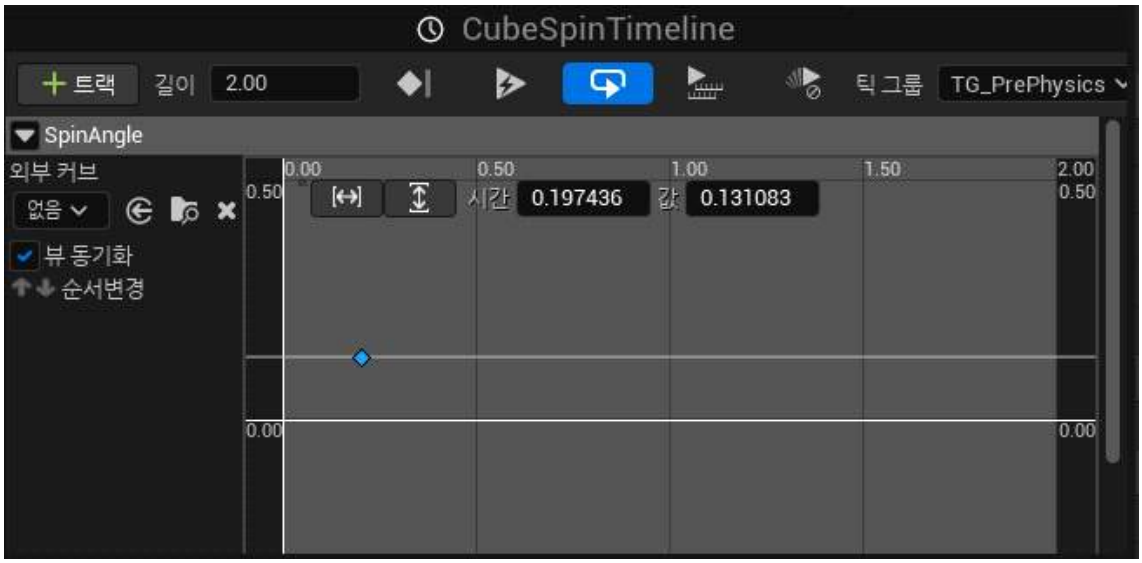
9. 추가된 트랙의 이름을 **SpinAngle**으로 수정하자. 이제 다음의 모습으로 보일 것이다.



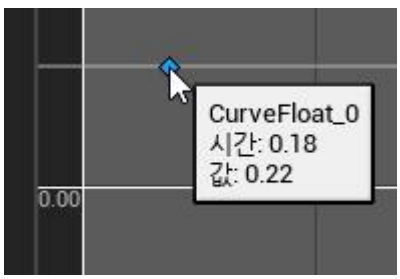
타임라인이 나타난다. 타임라인의 가로축이 **시간** 축이고 세로축이 **값** 축이다. **Float 트랙**에서는 **값** 축은 하나의 **플로트** 값으로 나타난다.
타임라인의 상단에서 좌우로 0.00, 0.50, 1.00, 1.50, 2.00으로 표시된 5개의 격자칸이 가로축인 시간 축의 표시이다. 현재 0에서 2까지의 구간을 표시하고 있다.
타임라인의 왼쪽과 오른쪽에서 상하로 -0.50, 0.00, 0.50으로 표시된 3개의 격자칸이 세로축인 값 축의 표시이다. 현재 -0.5에서 0.5까지의 구간을 표시하고 있다.

10. 이제 타임라인을 작성해야 한다.
우리는 2초 동안에 한 바퀴를 회전하도록 하자. 이렇게 하려면 0초에서 각도가 0이 되도록 하고 2초에서 각도가 360이 되도록 하면 된다. 그 중간의 값은 자동 생성될 것이다. 따라서 타임라인에 (0,0)와 (2,360)의 2개의 지점을 표시해주면 된다.
타임라인에서는 표시할 지점을 **키**(key)라고 한다. 우리는 2개의 키를 추가하고 각각이 (0,0), (2,360)에 배치되도록 해보자.
키를 추가하는 방법은 타임라인 위에서 추가할 위치에서 우클릭하면 팝업메뉴가 나타난다. **CurveFloat_0에 키 추가**를 선택하자. 여기서 **CurveFloat_0**의 마지막 숫자는 내부적으로 자동으로 정해지는 트랙 ID를 표시한 것으로 자동으로 정해지므로 신경 쓰지 않아도 된다. **CurveFloat_0** 대신 우리가 정해준 **SpinAngle**이라고 생각하자.

11. 이제 키가 추가되었다. 키는 우클릭하여 팝업창을 열 때의 클릭된 위치에 추가된다. 추가된 키는 작은 파란색 점으로 표시된다.



12. 키의 위치가 정확하지 않다. 위치를 수정해보자. **키**를 좌클릭+드래그해서 위치를 이동해보자.



13. 만약 원하는 키의 위치를 정확히 아는 경우에는 키값을 직접 입력하면 된다.
현재 선택된 키의 위치인 **시간**과 **값**이 상단에 표시된다. 텍스트 상자를 클릭하여 **시간**에 0을 입력하고 **값**에 0을 입력하자.

이제 키가 정확히 원점에 배치되었을 것이다.

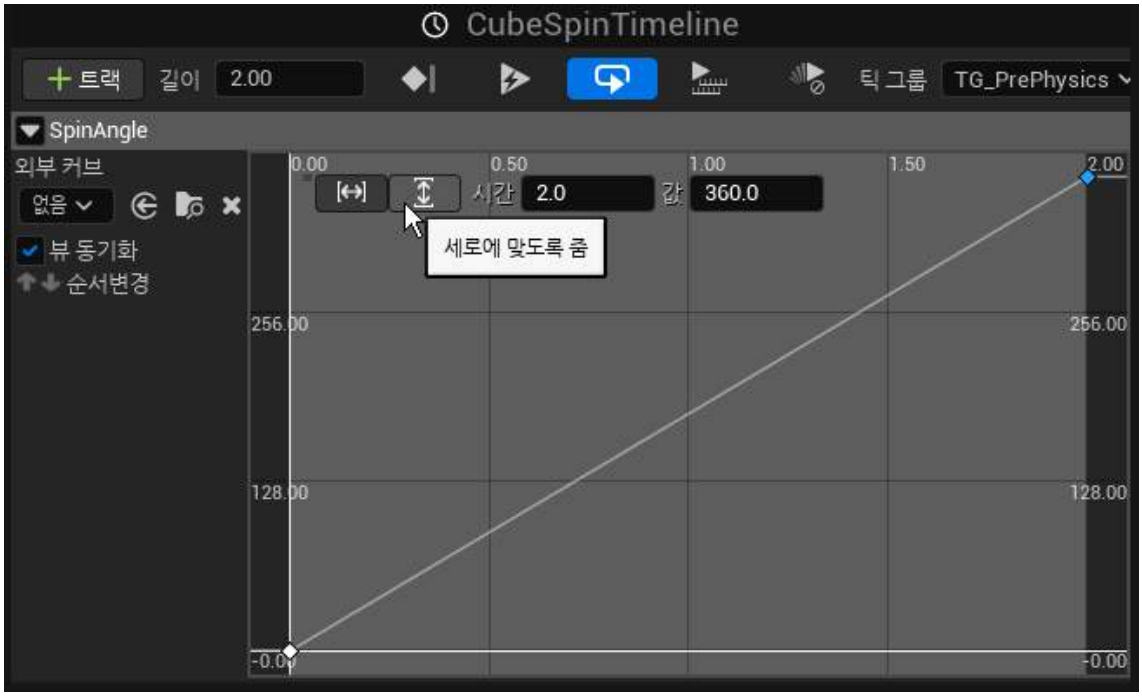
14. 두 번째 키를 입력하자.
이번에는 다른 방식으로 키를 추가해보자. 키를 추가하는 더 간편한 방식이 있다. 추가할 위치에서 **Shift+좌클릭**하면 **키**가 창이 뜨지 않고 키가 바로 추가된다.

추가된 키를 드래그하거나 또는 상단의 입력창에 직접 입력하여 위치를 수정해주어야 한다.
우리는 **시간**에 2를 입력하고 **값**에 360을 입력하자.

15. 두 번째 키에 대해서 (2, 360)을 입력한 후에는 두 번째 키가 보이지 않을 것이다. 그 이유는 두 번째 키가 현재 표시 구간을 벗어나 있기 때문이다. 현재에는 **시간**은 0~2 구간을 표시하고 있고 **값**은 -0.5~0.5 구간을 표시하고 있다. 따라서 값의 표시 구간을 수정해주어야 한다. 타임라인의 왼쪽 상단을 보면 맞춤 줌 기능의 아이콘이 있다. 이 아이콘은 현재 포함된 모든 키가 화면에 나타나도록 가로축과 세로축의 구간을 자동으로 맞추어준다. 왼쪽이 가로 방향으로 맞도록 줌하는 기능이고 오른쪽이 세로 방향으로 맞도록 줌하는 기능이다. 우리는 오른쪽 아이콘을 클릭하자.

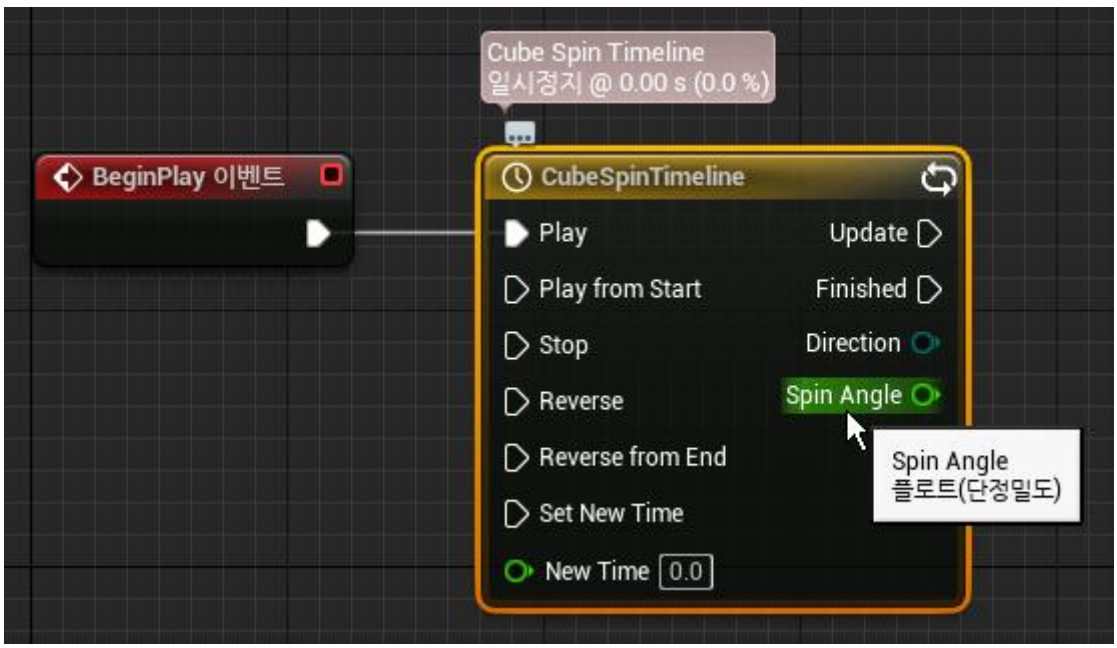
맞춤 기능에 대해서 조금 더 알아보자.
어떤 경우에는 가로 세로 맞춤 줌 기능이 마음에 들지 않을 수도 있다. 이럴 때에는 수동으로 표시 범위나 줌을 조절하자. 마우스 **휠버튼+스크롤**하면 줌 수준이 조절된다. 또한 **우클릭+드래그**하면 표시 구간이 이동된다. 조작이 쉽고 직관적이므로 직접 연습해보자.

16. 이제 다음과 같이 보일 것이다.



현재 선택된 키는 파란색으로 표시되며 그 외의 키는 흰색으로 표시된다. 나중에 키의 위치를 다시 조절하고자 하는 경우에는 해당 키를 드래그하여 움직이거나 값을 직접 입력하면 된다.
이제 **CubeSpinTimeline** 타임라인을 모두 완성하였다. 저장하고 **CubeSpinTimeline** 탭을 닫자.

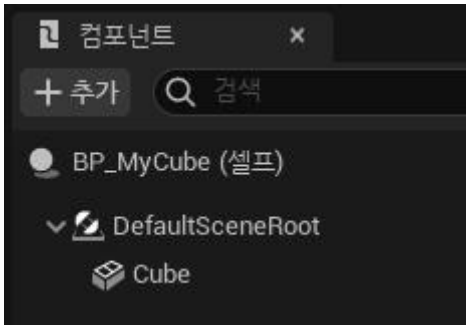
17. 이벤트 그래프에 있는 **CubeSpinTimeline** 타임라인 노드를 살펴보자.



이전과 다르게 **SpinAngle**이라는 출력핀이 추가되어 있다. 타임라인에 트랙을 추가하게 되면 그 트랙의 값이 출력핀으로 출력되는 것이다.

18. 격자판의 빈 곳에서 우클릭하고 **SetRelativeRotation** 노드를 검색하여 배치하자.
검색하면 **SetRelativeRotation (Cube)**와 **SetRelativeRotation (DefaultSceneRoot)**의 두 노드가 있을 것이다.

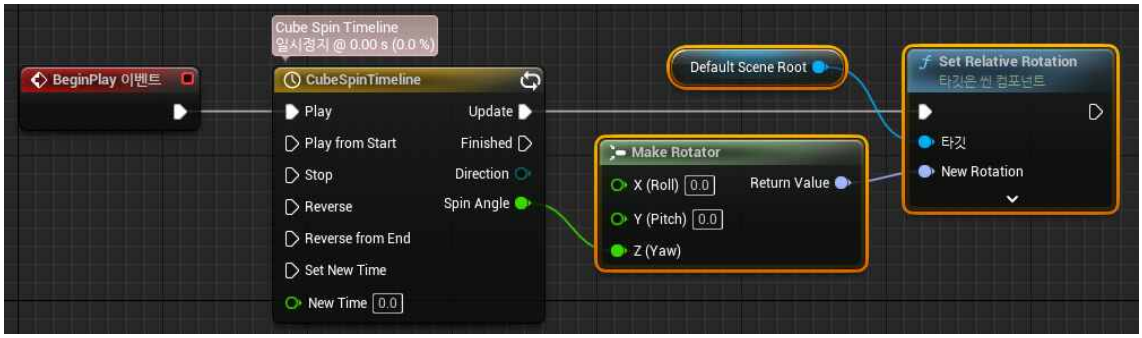
우리는 이번에는 **SetRelativeRotation (DefaultSceneRoot)**를 선택하여 배치하자.
참고로, 지금부터 **SetRelativeRotation (Cube)**와 **SetRelativeRotation (DefaultSceneRoot)**의 배치에 대한 차이점을 살펴보자. 실제로 어느 노드를 선택하든지 큐브가 스핀되는 동일한 결과를 얻을 것이다. 그러나 그 의미에는 차이가 있다. 그 차이를 자세히 알아보자.
액터 내에 포함된 모든 씬 컴포넌트는 하나의 계층 구조를 이루고 있다. **DefaultSceneRoot**이 계층 구조의 최상위에 있는 루트 컴포넌트이고 그 아래에 스태틱 메시 컴포넌트인 **Cube** 컴포넌트가 있다. 액터의 위치는 바로 루트 컴포넌트의 위치이다. 루트 컴포넌트 아래에 부착되어 있는 그 외의 모든 씬 컴포넌트의 위치는 자신의 바로 위의 부모 컴포넌트의 위치에 상대적으로 정의된다.



먼저, **SetRelativeRotation (Cube)**를 실행하는 경우를 생각해보자. **Cube** 컴포넌트는 실제 메시를 가지고 있는 컴포넌트이므로 이 컴포넌트를 회전시키면 메시가 회전될 것이다. 그러나 루트 컴포넌트는 그대로 있을 것이므로 외부에서 **GetActorRotation**으로 액터의 회전을 확인하면 액터의 회전에는 변화가 없음을 알 수 있다.
이번에는 **SetRelativeRotation (DefaultSceneRoot)**를 실행하는 경우를 생각해보자. **DefaultSceneRoot**를 회전시키면 그 아래의 모든 컴포넌트에게 영향을 미치게 된다. 그 아래의 컴포넌트들의 회전값이 실제로 바뀌는 것은 아니다. 바뀌지 않더라도 루트를 제외한 모든 컴포넌트는 자신의 직접 부모 컴포넌트에 상대적으로 동작하기 때문이다. 외부에서 **GetActorRotation**으로 액터의 회전을 확인하면 루트 컴포넌트의 회전이 바뀌므로 액터의 회전도 바뀐 것으로 알 수 있다.

<참고> 한 클래스 블루프린트에서 포함된 메시 컴포넌트의 애니메이션 움직임 구현을 위해서 상대적인 움직임을 제어하는 함수인 **SetRelativeLocation**, **SetRelativeRotation** 등을 사용하면 편리하다. 이들 함수는 루트 컴포넌트에 상대적인 씬 컴포넌트의 움직임을 제어한다.

19. 배치된 **SetRelativeRotation** 노드의 **NewRotation** 입력핀에 **Rotator** 유형의 입력값을 넣어주어야 한다. **MakeRotator** 노드를 배치하고 그 출력을 연결시켜주자.
그리고, 타임라인 노드의 **SpinAngle** 출력을 **MakeRotator** 노드의 **Z(Yaw)** 입력에 연결하자.
그리고, 타임라인 노드의 Update 출력 실행핀을 **SetRelativeRotation** 노드로 연결하자.
다음과 같은 그래프가 완성될 것이다.



타임라인 노드를 살펴보자.

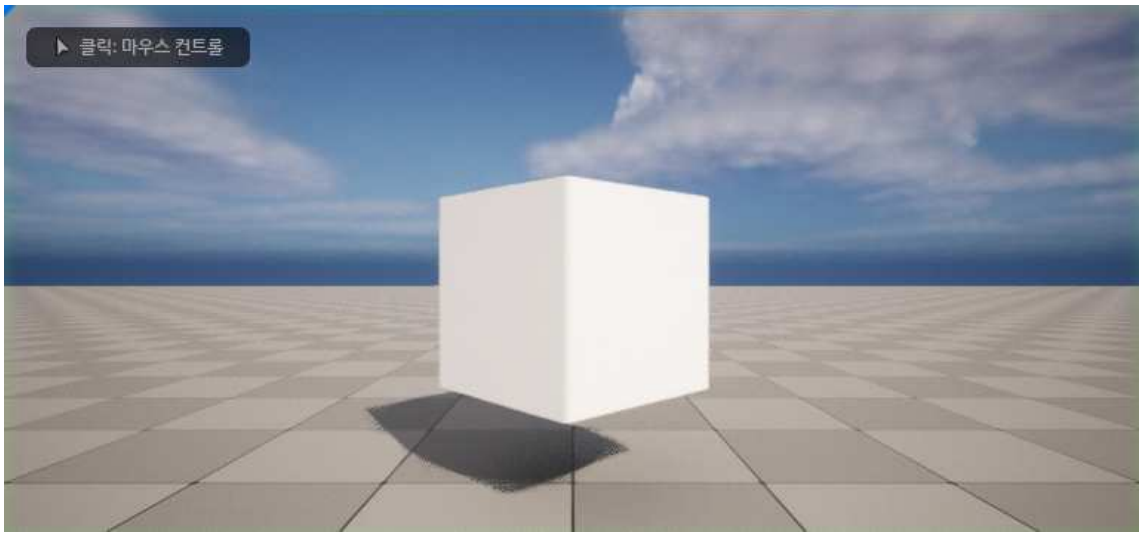
우선 출력핀을 보면 **Update** 실행핀이 있고 **Finished** 실행핀이 있다. **Update** 실행핀으로는 타임라인이 재생되는 동안 계속해서 실행 펄스가 흐르게 된다. 타임라인의 재생이 모두 종료되면 **Update** 실행핀의 펄스는 끊어지고 **Finished** 실행핀으로 실행 펄스가 한번 흐른다.

우리는 타임라인 설정에서 루프에 체크하여 계속 반복되도록 하였다. 이런 경우에는 계속해서 **Update**에만 실행핀이 흐르고 **Finished**에는 실행핀이 흐르지 않는다.

컴파일하고 저장하자.

20. 플레이해보자. 큐브가 2초에 한바퀴의 속도로 회전할 것이다.

큐브의 회전 결과는 이전에서의 예제와 유사하나 회전하는 방식은 전혀 다르게 구현하였다. 이전에서는 **Tick 이벤트** 노드를 사용하여 회전하도록 하였으나 이번에는 타임라인을 사용하여 무한 반복하는 방식으로 회전하도록 하였다.



지금까지, 타임라인을 만들고 사용하는 방법에 대해서 학습하였다.

3. 타임라인과 선형 보간을 함께 사용하기

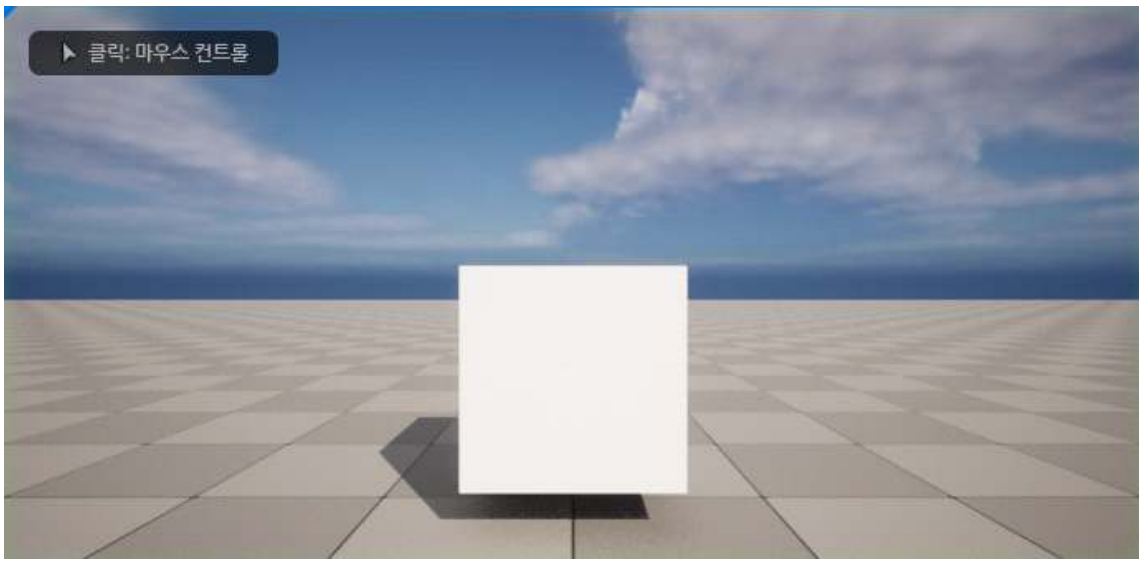
이 절에서는 타임라인과 선형 보간을 함께 사용하는 방법에 대해서 학습한다.

타임라인의 트랙을 작성할 때에 여러 개의 키를 입력하여 곡선을 표현한다. 입력된 키는 곡선 상의 점으로 가정한다. 연속적인 곡선을 만들기 위해서는 키를 보간해야 한다. 이때 키의 보간 방식에 따라 곡선의 모양이 결정된다. 이제부터 키 보간 기능에 대해서 알아보자. **키 보간**이란 키와 키 사이에서의 **시간**에 대해서 **값**을 어떻게 계산할 지의 보간 방법이다.
디폴트로 되어 있는 **선형** 보간은 두 인접 키를 직선으로 연결하고 그 직선 상의 값으로 결정하는 방법이다. 선형 보간 함수를 Lerp이라고 한다. 두 키 값 A와 B 사이를 선형 보간하는 경우를 생각해 보자. 0과 1 사이의 값인 α값으로 선형 보간하는 Lerp(A,B,α) 함수는 A*(1-α)+B*α 의 값을 리턴한다. 즉 α가 0이면 A를 리턴하고, 1이면 B를 리턴하고, 그 사이의 값에 대해서는 선형 비례만큼 가중치 평균한 값을 리턴한다.
선형 외의 다른 보간 방법 중에서, **상수**는 키와 키 사이에서는 이전 키의 값이 다음 키까지 유지되는 계단식 함수를 사용하는 보간 방법이다. 이것은 잘 사용되지 않는다.
선형 보간의 단점은 키 사이에서 급격하게 꺾이게 된다는 점이다. 이의 해결을 위하여 연결이 표시되지 않고 부드럽게 바뀌도록 스무딩하고 또한 바뀌는 속도가 가속이나 감속되도록 이징(easing)해주어야 한다.
키 보간 메뉴에서 부드러운 바뀜에 되도록 하는 보간 방법인 **자동**, **사용자**, **꺾임** 방법을 제공한다. 이들은 모두 3차 큐빅 곡선 보간 방법이다. 이 방법에서는 순간 기울기를 조절하는 탄젠트를 사용하여 부드러운 곡선 모양을 만들 수 있도록 한다. 탄젠트는 중심이 키 위치에 있는 고정 길이의 선분으로 접선 방향을 표시해준다. 탄젠트 선분의 양 끝점이 표시되며 이를 드래그하여 탄젠트의 방향을 수정할 수 있다.
자동, **사용자**, **꺾임** 방법은 모두 거의 유사하나 약간의 차이가 있다. 차이점에 대해서 알아보자. **자동**의 경우에는 자신의 키에 이웃한 위치에 있는 키가 이동되거나 새로 추가되는 경우에도 곡선을 부드럽게 만들기 위해서 자신의 키의 탄젠트가 자동으로 조절된다. 한편, **사용자**의 경우에는 주변 키가 이동되더라도 자신의 키의 탄젠트를 고정시키고 변경되지 않도록 한다.
자동의 경우에는 탄젠트가 자동으로 계산된다는 의미로 이해하면 된다. **자동**으로 지정된 키의 경우 위치를 수정하는 것은 상관없지만 탄젠트를 수정하게 되면 **자동**에서 **사용자**로 저절로 바뀌게 된다. 한편, **꺾임**은 탄젠트의 양 끝점을 독립적으로 수정할 수 있다. 따라서 서로 다른 입사각과 반사각을 표시할 수 있다.
한 키의 보간 방법을 특정한 것으로 지정하면 그 키에서 다음 키까지의 구간에 대해서 지정된 보간 방법이 적용된다. 키의 이전까지의 구간에 대해서는 이전 키에 지정된 보간 방법이 지정된다.
키 보간 메뉴에서 아래에 있는 **평탄화**와 **직선화**는 보간 방법이 아니라 3차 큐빅 곡선에서 사용되는 탄젠트를 편리하게 수정해주는 보조 기능을 제공한다.

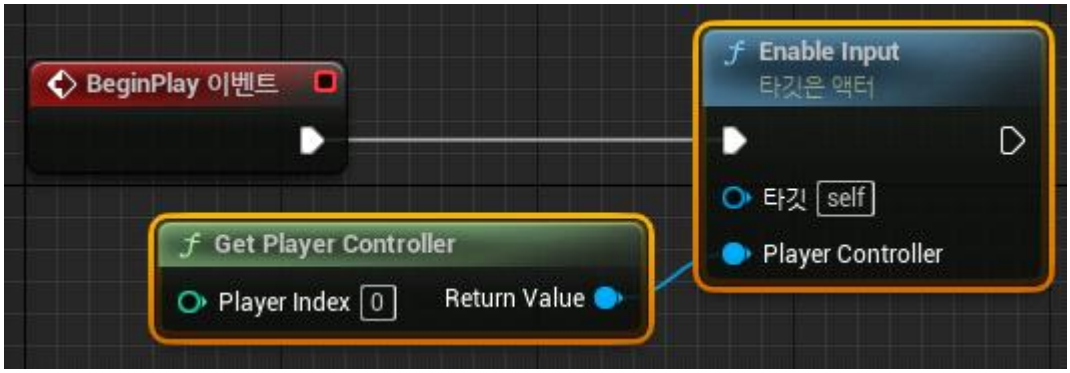
타임라인에서 실수값을 출력하는 **Float** 트랙 하나만을 사용하더라도 **벡터** 트랙 타임라인처럼 동작하도록 할 수 있다. **Float** 트랙의 값을 선형 보간을 위한 α값으로 사용하면 된다. **Float** 트랙 타임라인으로 출력되는 α값으로 두 벡터를 선형 보간하고 보간된 벡터값을 사용하면 된다. 선형 보간을 위한 함수로 Lerp을 사용하면 된다.

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

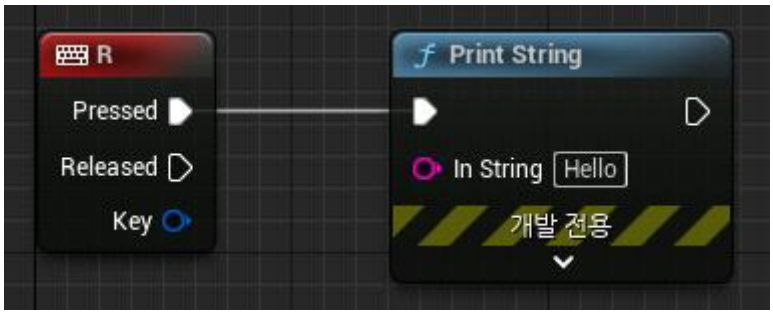
2. 블루프린트 클래스로 큐브 액터를 만들어보자.
이전 예제에서 만들었던 스핀되는 큐브인 **BP_MyCube**를 동일한 방법으로 만들어보자.
먼저, **콘텐츠 브라우저**에서 툴바에서 **+추가**를 클릭하고, 드롭다운 메뉴에서 **블루프린트 클래스**를 선택하고, **부모 클래스 선택** 창에서 **Actor**를 부모 클래스로 선택하자. 이름은 **BP_MyCube**으로 수정하자.
그다음, **BP_MyCube**을 더블클릭하여 **블루프린트 에디터**를 열자. **컴포넌트** 탭에서 **+추가**를 클릭하고 **큐브**를 선택하자. 큐브 스태틱 메시 컴포넌트가 추가될 것이다. 디폴트로 **Cube**라는 이름으로 그대로 두자. 저장하고 컴파일하자.
그다음, 콘텐츠 브라우저에서 **BP_MyCube**를 드래그하여 레벨에 배치하자. 위치는 (300,0,70)으로 하자.
플레이해보자. 바닥 위에 놓인 큐브가 보일 것이다.



3. **BP_MyCube**를 더블클릭하여 블루프린트 에디터를 열자. 입력 이벤트가 액터에 전달되도록 액터에서 입력을 활성화하자. 이벤트 그래프 탭에서 **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.

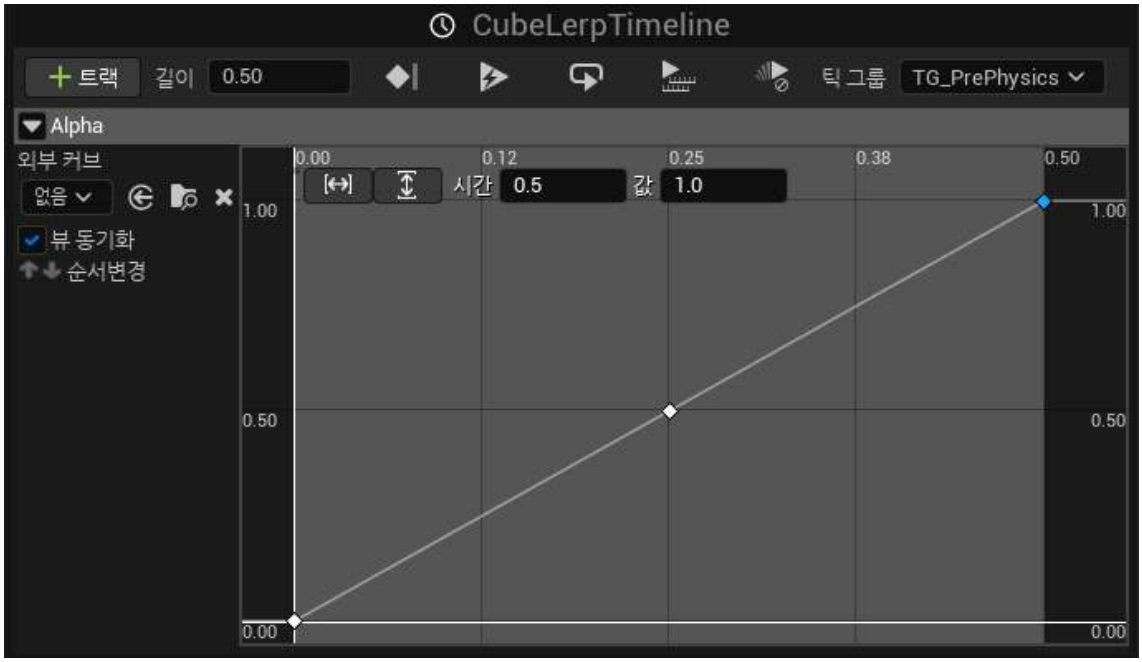


4. 이벤트 그래프의 격자판의 빈 곳에서 우클릭하고 **Input** » **키보드 이벤트** 아래의 **R** 키 입력 이벤트 노드를 배치하자.



플레이해보자. **R** 키를 누를 때에 문자열이 출력되는지를 확인하자.
이제 예제를 진행할 준비가 되었다.

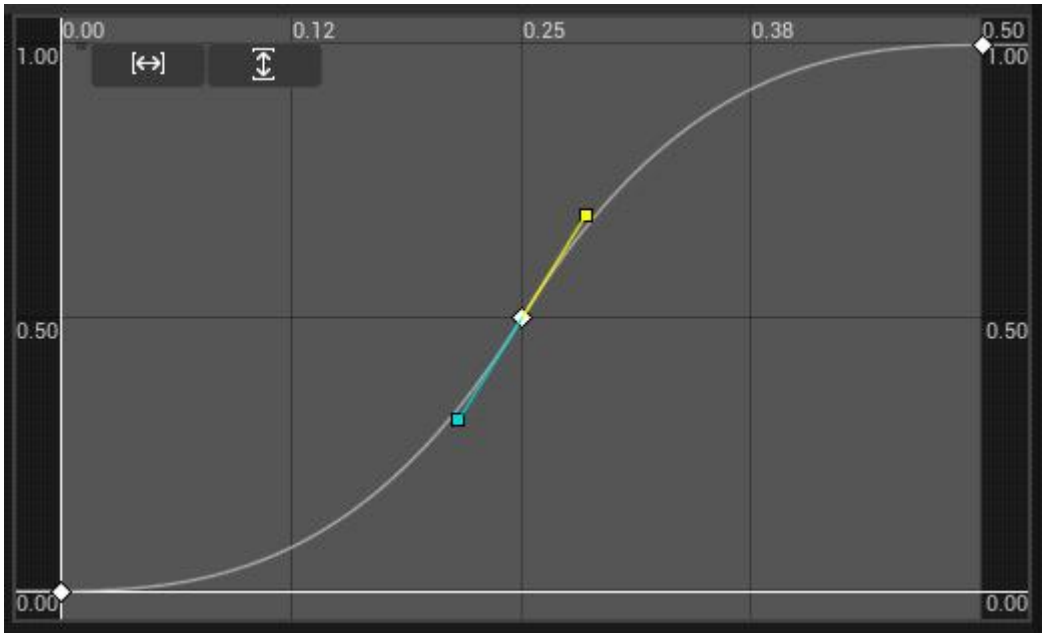
5. 이제 타임라인을 추가하고 타임라인을 편집하자.
BP_MyCube 블루프린트 에디터의 이벤트 그래프에서 **R** 입력 이벤트 노드 네트워크에 마지막 노드인 **PrintString** 노드를 제거하자. 그다음, **R** 입력 이벤트 노드의 **Pressed** 출력 실행핀을 드래그하고 액션선택 창에서 **AddTimeline**을 검색하여 **타임라인 추가**를 찾아 선택하자. 타임라인의 이름은 **CubeLerpTimeline**으로 지정하자.
타임라인 노드를 더블클릭하자.
그다음, 타임라인 탭의 툴바에서 **길이** 속성에 0.5를 입력하자. 이번에는 **루프**에는 체크하지 말고 그대로 두자.
그다음, 타임라인 툴바의 가장 왼쪽의 **+트랙**을 클릭하고 **Float 트랙 추가**를 선택하자. 추가된 트랙의 이름을 **Alpha**로 수정하자. 우클릭하고 **이름변경**을 선택하거나 **F2** 키를 누르면 변경할 수 있다.
그다음, **Shift+좌클릭**하여 3개의 키를 추가하자. 각각 (0, 0), (0.25, 0.5), (0.5,1)이 되도록 두 키의 (**시간**, **값**)을 입력하자. 그리고 맞춤 줌 아이콘을 클릭하고 마우스 스크롤하여 전체 그래프가 보이도록 하자.



6. 세 키 중의 한 키의 위에 마우스 커서를 두고 우클릭해보자.

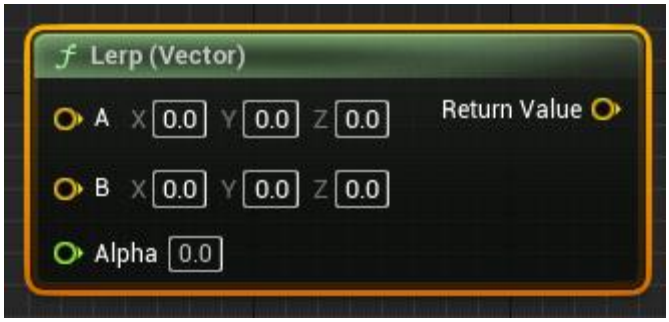
키 보간 메뉴창이 뜬다. 디폴트로 **선형**으로 되어 있다.

7. 우리는 모든 키에 대해서 먼저 **사용자**로 바꾸자. 각 키에 대해서 탄젠트 선분이 추가된다. 그다음, 첫 번째 키와 마지막 키에 대해서는 탄젠트 선분을 수평으로 두고, 중간 키에 대해서는 경사가 45도보다 더 크도록 기울이자. 다음과 같이 될 것이다.



이러한 모양의 곡선을 이즈인 이즈아웃(Ease In Ease Out) 곡선이라고 한다. 이는 가장 많이 사용되는 이징 방법이다. 출발 시에 매우 느리게 진행하기 시작하면서 조금씩 빨라지고, 중간에서는 최대 속력으로 진행하고, 도착 시에는 점점 느려지다가 정지하는 모습이다. 자연스러운 느낌의 동작을 보여주는 형태이다. 자연스러운 움직임을 보여주며 애니메이션 시에 많이 사용된다.
이제 **CubeLerpTimeline** 타임라인을 모두 완성하였다. 컴파일하고 저장하자. 이제 **CubeLerpTimeline** 탭을 닫자.

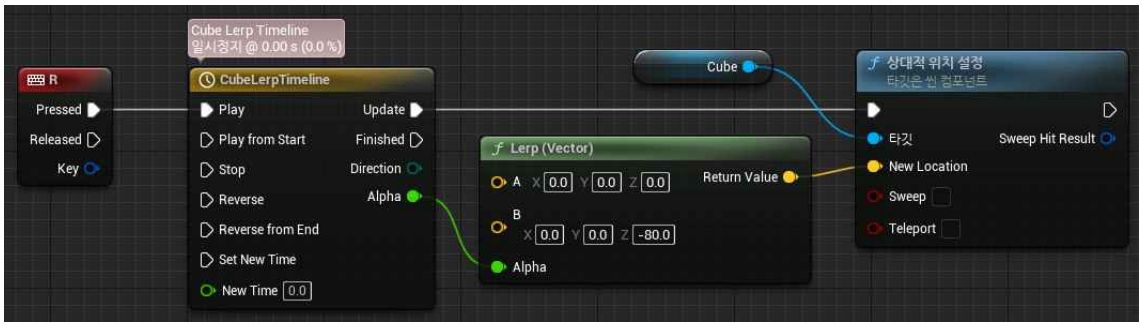
8. 이제 애니메이션 동작을 계획해보자.
우리가 배치해둔 큐브의 크기는 100x100x100이다. 큐브의 피봇은 큐브의 중심에 있다. 바닥면의 높이는 20이어서, 큐브가 바닥면에 붙어있도록 큐브를 높이 70에 배치하였다.
우리는 타임라인을 통해서 큐브를 80만큼 내려서 큐브의 상단이 20만큼만 보이도록 해보자.
이벤트 그래프 탭에서 **R** 입력 이벤트 그래프에 연결되어있는 **CubeLerpTimeline** 타임라인 노드로 가자. 타임라인 노드의 **Alpha** 출력값은 0에서 1까지의 값을 출력한다. **Alpha** 값이 0이면 0만큼 내리도록 하고 1이면에는 80만큼 내리도록 해보자. 즉 **Alpha** 값이 (0,1) 사이의 값일 때 (0,80)의 사이를 선형 보간함을 의미한다. 이러한 기능을 제공하는 함수가 바로 **Lerp** 함수이다.
이벤트 그래프에서 우클릭하여 **Lerp**을 검색해보자. 선형 보간할 대상 유형에 따라서 많은 함수들이 나열될 것이다. 그 중에서 **Lerp (Vector)**를 선택하자. 이 함수는 두 벡터 사이를 선형 보간하는 함수이다.



배치된 **Lerp** 노드를 살펴보자. 입력핀에는 벡터 유형의 **A**와 **B**가 있고 플로트 유형의 **Alpha**가 있다.

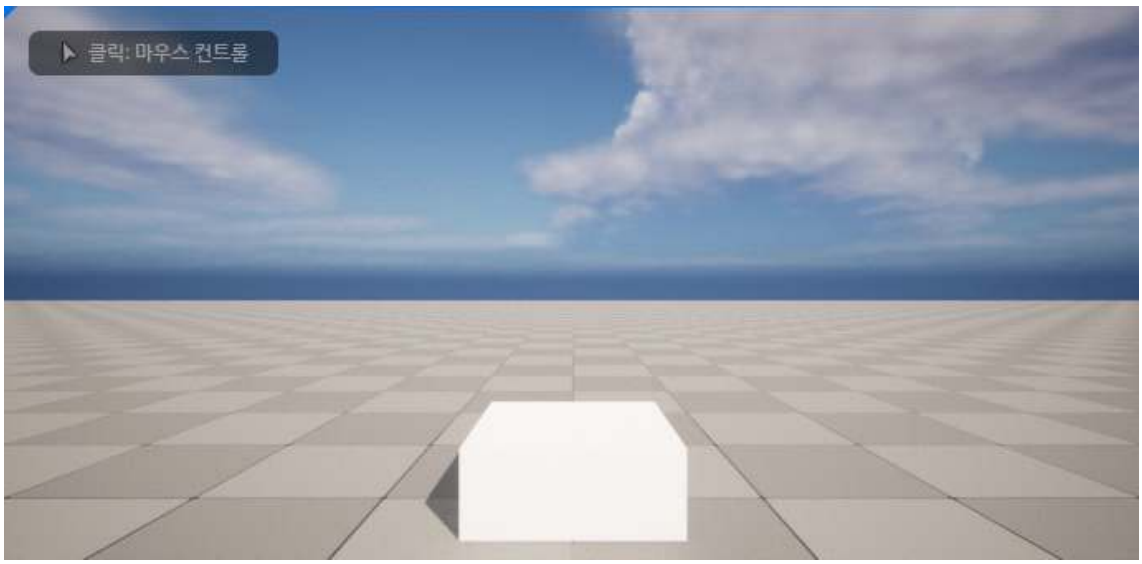
Alpha가 0이면 A를 리턴하고 1이면 B를 리턴하고 그 사이에 값에 대해서는 A*(1-Alpha)+B*Alpha 의 값을 리턴한다. 우리는 A를 (0,0,0)으로 하고 B를 (0,0,-80)으로 하자.

9. 타임라인 노드의 Alpha 출력핀을 Lerp 노드의 Alpha 입력핀에 연결하자.
그다음, 액션선택 창에서 검색하여 SetRelativeLocation (Cube) 노드를 배치하자.
여기서, 만약 SetRelativeLocation (DefaultSceneRoot) 노드를 배치하여 실행하게 되면 루트 노드의 상대 위치가 바뀌게 된다. 루트 노드의 상대 위치는 액터의 절대 위치에 해당한다. 따라서 루트 노드에 대해서는 상대값의 의미가 아닌 절대값의 의미이므로 우리는 루트가 아닌 Cube 컴포넌트의 노드를 배치하였다.
그다음, 타임라인 노드의 Update 실행핀을 SetRelativeLocation (Cube) 노드에 연결하자.
그다음, Lerp 노드의 출력을 SetRelativeLocation 노드의 NewLocation 입력핀에 연결하자.
이제 다음과 같은 모양의 그래프가 완성되었다.



컴파일하고 저장하자.

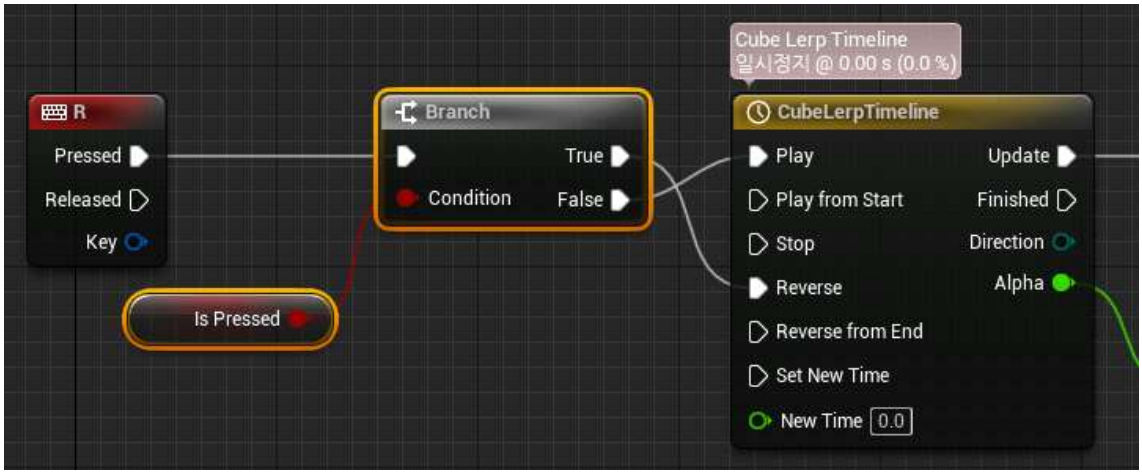
10. 플레이하고 R 키를 눌러보자. 상자가 부드럽게 내려가는 것을 확인할 수 있다.



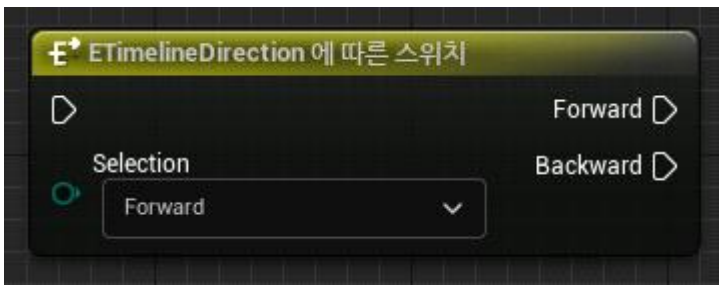
11. 이제 다시 R 키를 누르면 올라오도록 해보자.
먼저 블루프린트 에디터의 내 블루프린트 탭에서 변수를 추가하자. 변수명은 IsPressed으로하고 유형은 부울(Boolean)로 하자. 큐브가 눌러진 상태이면 이 변수가 true를 가지도록 하자. Boolean 변수는 초기값은 false으로 되므로 따로 초기화는 필요없다.

12. IsPressed가 true인 경우에는 R 키를 눌렀을 때에 큐브가 다시 올라오도록 해야 한다. 이를 위해서

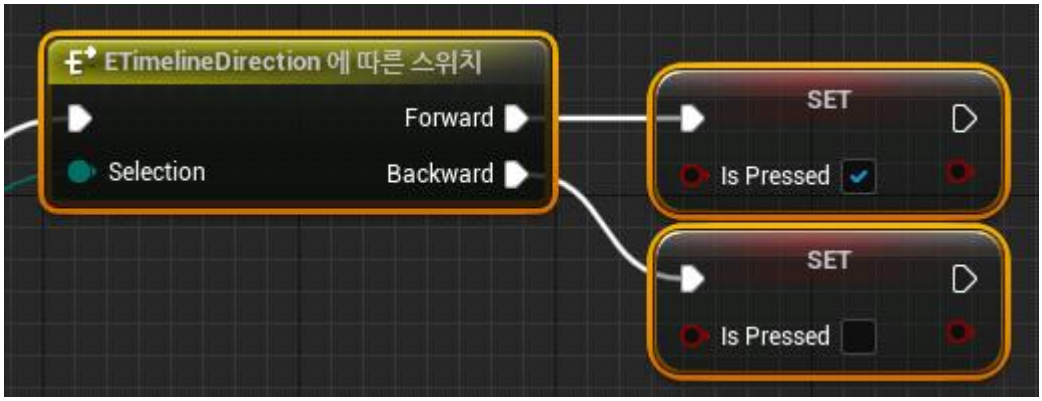
는 새로운 타임라인을 만들어야 한다. 그러나 기존 타임라인을 그대로 역재생하면 되는 경우에는 기존의 타임라인을 그대로 활용하면 된다. 타임라인에는 **Reverse**라는 입력 실행핀이 있다. 이 실행핀으로 펄스가 주어지면 타임라인을 시간 축의 뒤에서부터 앞으로 역으로 플레이한다.



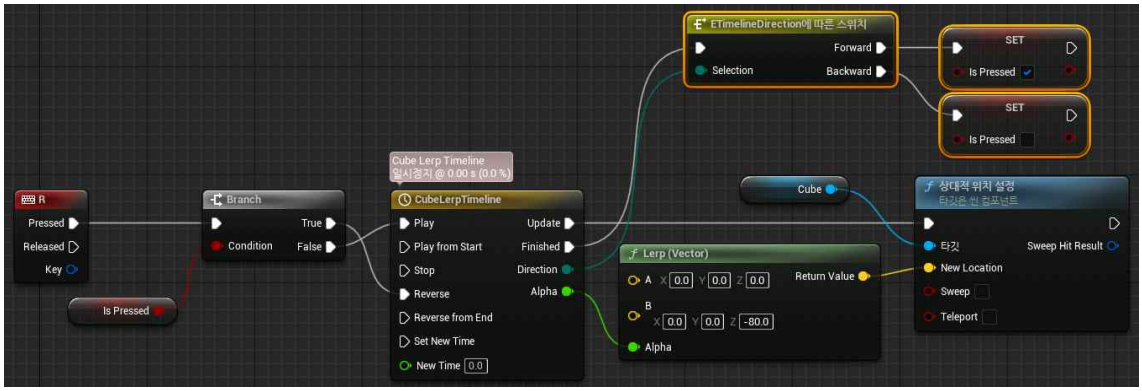
13. 플레이해도 변화가 없을 것이다. 아직 **IsPressed**에 **true**를 지정하는 부분이 없기 때문이다.
한편 타임라인 노드에는 **Direction** 출력핀이 있다. 이 출력핀은 현재 타임라인이 정방향으로 플레이되는지 또는 역방향으로 플레이되는지에 대한 정보를 제공한다.
플레이 방향 정보를 **ETimelineDirection** 열거형 타입의 값을 제공한다. 이 열거형 타입은 **Forward** 또는 **Backward**의 두 값을 가진다.
엔진은 각 열거형 타입에 대해서 흐름 제어 연산을 위한 스위치 노드를 제공한다. 즉, 열거형 타입명 **EnumType**에 대해서 **EnumType에 따른 스위치**라는 노드를 제공한다. 아래는 **ETimelineDirection에 따른 스위치** 노드이다. 열거형 타입의 값을 가지는 열거형 변수를 **Selection** 입력핀에 연결하면 된다. 변수의 값에 따라서 여러 출력핀 중의 하나에 대해서 펄스가 흐르게 된다.



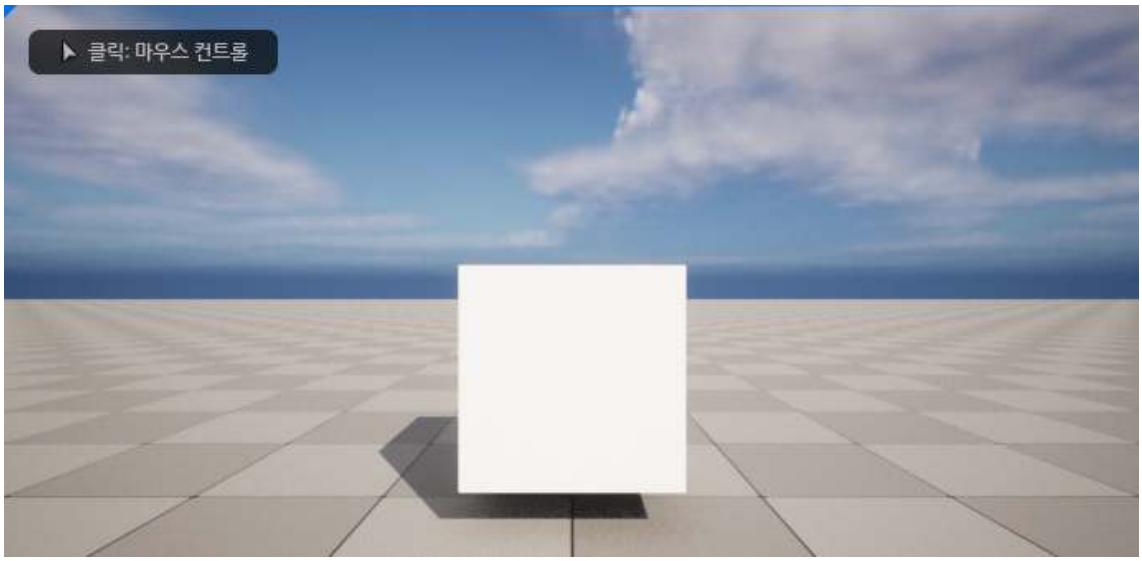
14. 타임라인 노드의 **Direction** 출력핀을 스위치 노드의 **Selection** 입력핀에 연결하자.
그다음, 스위치 노드의 **Forward** 출력핀에 대해서는 **IsPressed**를 **true**로 지정하는 **Set** 노드를 배치하고, **Backward** 출력핀에 대해서는 **false**로 지정하는 **Set** 노드를 배치하자. 그다음, 이전의 타임라인 노드의 **Finished** 출력 실행핀을 스위치 노드의 입력 실행핀에 연결하자.



15. 전체적으로 다음과 같은 모습이 될 것이다.



16. 플레이해보자. R 키를 반복해서 눌러보자. R 키를 누르면 큐브가 내려갈 것이다. R 키를 다시 누르면 큐브가 원래대로 올라갈 것이다. 이렇게 R 키를 누를 때마다 큐브가 내려가고 올라가는 것을 번갈아 진행할 것이다.

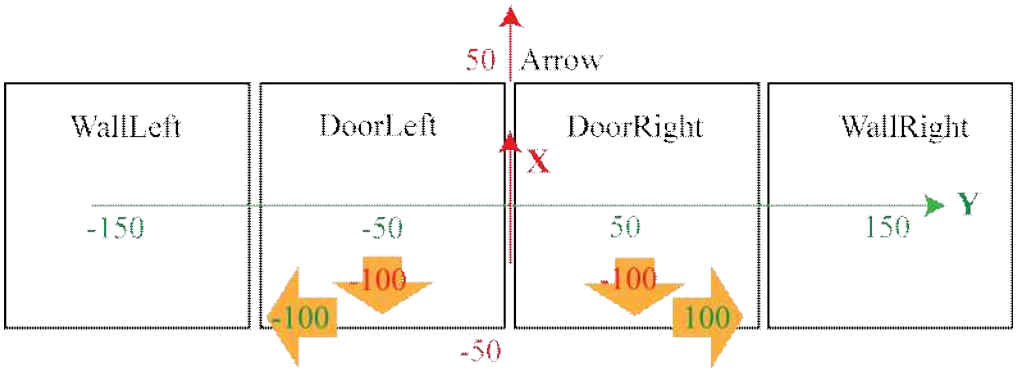


지금까지, 타임라인과 선형 보간을 함께 사용하는 방법에 대해서 학습하였다.

4. 타임라인에서 벡터 트랙 사용하기

이 절에서는 타임라인에서 벡터 트랙을 사용하는 방법에 대해서 학습한다.

이번 실습에서는 양방향 미닫이문을 만들어보자. 아래의 그림과 같이 네 개의 문짝에 해당하는 메시가 있다. 안쪽의 두 문짝만 움직이고 바깥쪽의 두 문짝은 고정되어 있다.



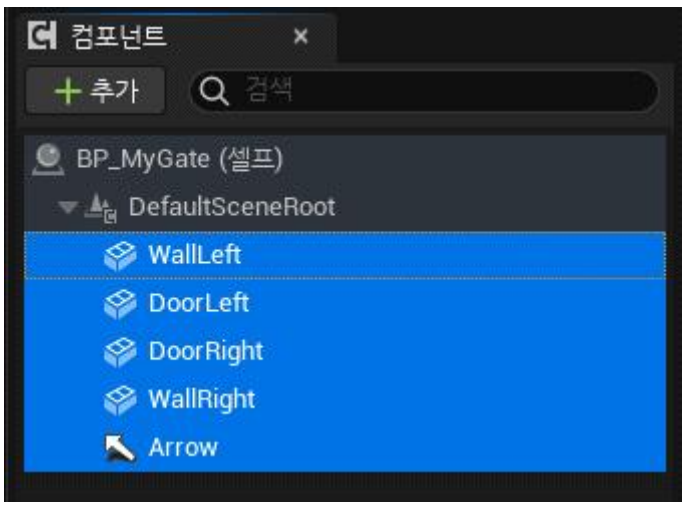
X축의 양의 방향이 문이 외부로 향하는 방향이다. 문짝 명칭에 있어서 좌우의 구분은 문의 내부에서 외부를 바라볼 때의 기준이다.
WallLeft와 **WallRight**는 고정된 벽이고 **DoorLeft**와 **DoorRight**가 이동하여 문이 열린다. 문이 열리는 시간은 2초이다. 처음 1초 동안에는 **DoorLeft**와 **DoorRight**가 모두 X축으로 -100 이동하자. 그다음의 1초 동안에는 **DoorLeft**는 Y축으로 -100을 이동하고 **DoorRight**는 Y축으로 100을 이동한다. 그 외의 것들은 변하지 않는다.
이제부터 특정 이벤트가 발생하면 문이 2초에 걸쳐서 열리도록 만들어보자. 또한 역순으로 닫히는 것도 만들어보자.
이번 경우에는 타임라인에서의 움직임을 **Float** 트랙의 선형 보간으로는 표현할 수 없다. 처음 1초 동안은 X축만 움직이고 그다음의 1초 동안은 Y축만 움직이기 때문에다. 이런 경우에는 **벡터** 트랙을 사용하면 된다. 타임라인에 **벡터** 트랙을 추가하여 구현해보자.

---

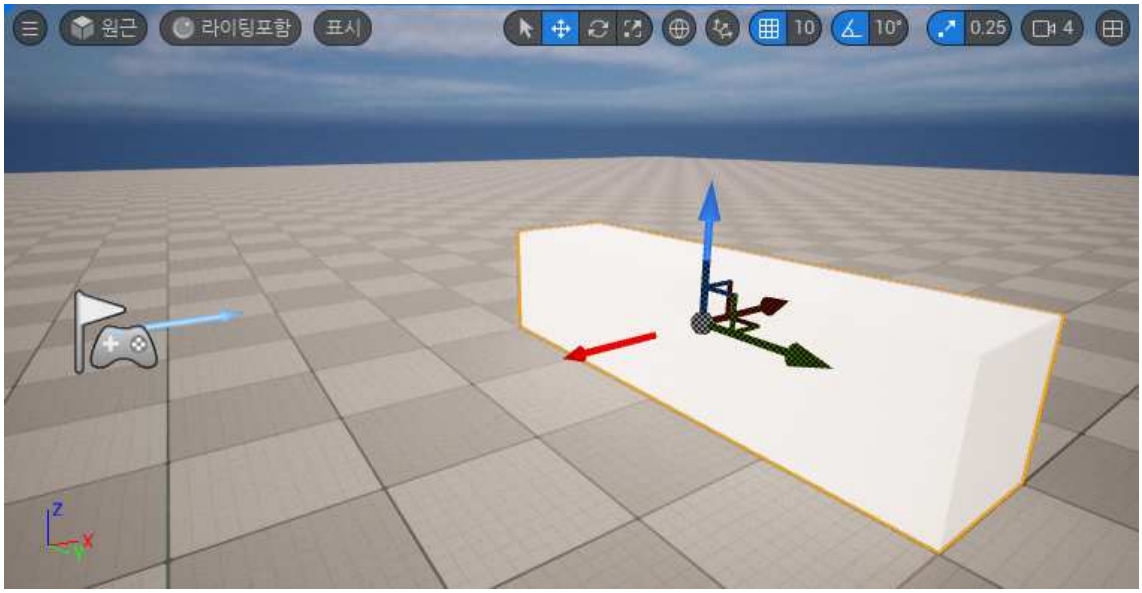
1. 새 프로젝트 **Ptimelinevector**를 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 Ptimelinevector로 입력하고 **생성** 버튼을 클릭하자. 프로젝트가 생성되고 언리얼 에디터 창이 뜰 것이다.
창이 뜨면, 메뉴바에서 **파일** » **새 레벨**을 선택하고 **Basic**을 선택하여 기본 템플릿 레벨을 생성하자. 그리고, 레벨 에디터 툴바의 저장 버튼을 클릭하여 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. 이전과는 다른 새로운 블루프린트 클래스로 만들자. 양방향 미닫이문에 해당하는 액터 클래스를 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_MyGate**으로 수정하자.

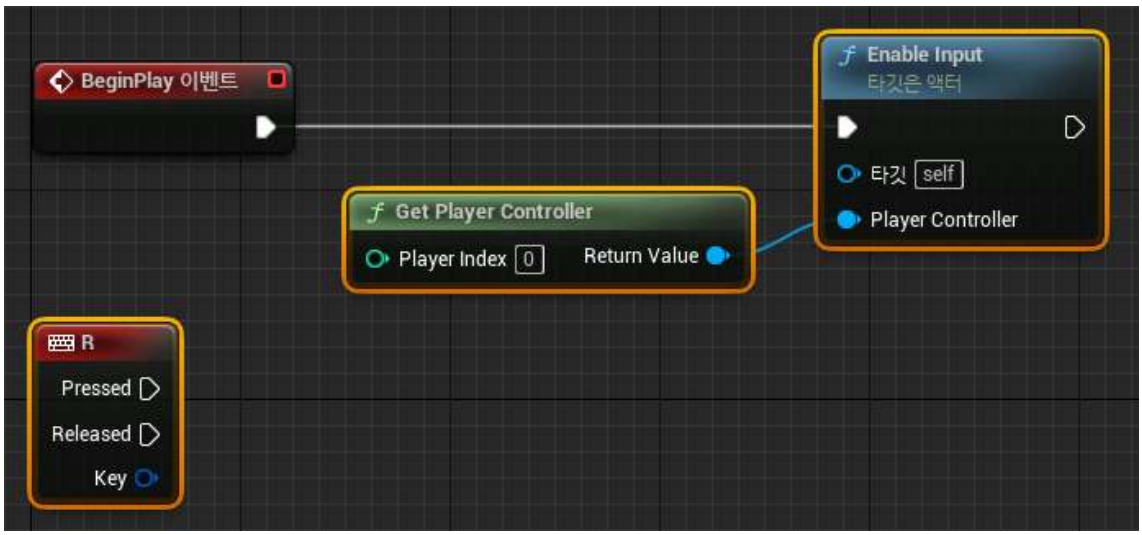
그다음, **BP_MyGate**를 더블클릭하여 **블루프린트 에디터**를 열자.
문을 구성하는 네 개의 문짝은 각각 큐브를 사용하여 구성하자. **컴포넌트** 탭에서 **+추가**를 클릭하고 **큐브**를 선택하자. 큐브 스태틱 메시 컴포넌트가 추가될 것이다. 반복해서 모두 4개를 추가하자. 각각의 이름을 **WallLeft**, **DoorLeft**, **DoorRight**, **WallRight**로 수정하자. 모두 루트 컴포넌트 아래에 배치하자. 새 컴포넌트를 추가할 때에는 현재 선택된 컴포넌트의 자식으로 추가된다. 따라서 추가할 때마다 루트 컴포넌트를 선택한 후에 추가해야 한다.
각 컴포넌트의 위치를 (0,-150,0), (0,-50,0), (0,50,0), (0,150,0)으로 하자. 이 위치는 루트 컴포넌트에 대한 각 컴포넌트의 상대 위치가 된다.
또한, 컴포넌트 탭에서 루트 컴포넌트가 선택된 상태에서 **+추가**를 클릭하고 **화살표**(**Arrow**) **컴포넌트**를 검색하여 추가하자. 이 컴포넌트도 루트 컴포넌트 아래에 배치하자. 이 컴포넌트의 위치는 (50,0,0)으로 하자. 이 컴포넌트는 문이 향하는 방향을 알려주기 위한 목적이다. 내부에서 외부로 향하는 방향을 알려준다.
저장하고 컴파일하자.



3. 콘텐츠 브라우저에서 **BP_MyGate**를 드래그하여 레벨에 배치하자. 위치는 (350,0,50)으로 하자. 그리고 회전은 (0,0,180)으로 하자. 이렇게 회전해두면 문이 플레이어를 바라보게 된다.
플레이해보자. 바닥 위에 놓여진 좌우로 긴 직육면체가 보일 것이다.

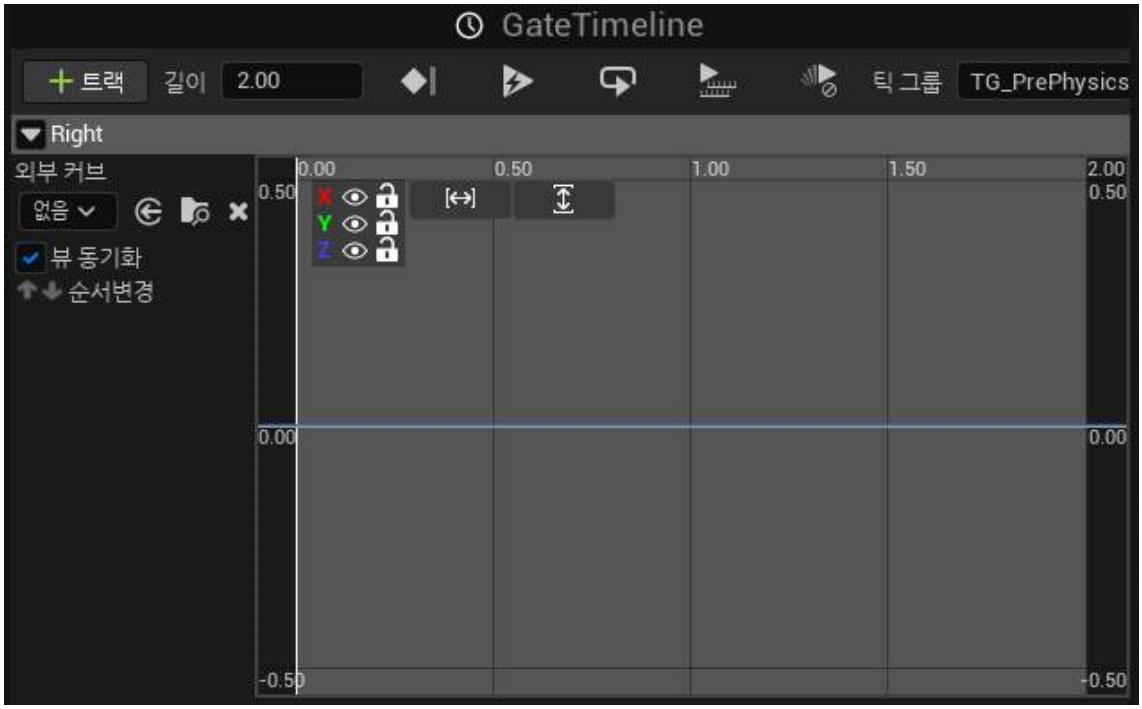


4. 이전 예제에서와 동일한 방법으로 입력 이벤트를 사용할 수 있도록 하자.
BP_MyGate를 더블클릭하여 블루프린트 에디터를 열자. 입력 이벤트가 액터에 전달되도록 액터에서 입력을 활성화하자. **BeingPlay** 이벤트 노드의 근처의 빈 곳에서 우클릭하고 **EnableInput** 노드를 배치하자. 그다음, 배치된 **EnableInput** 노드의 **PlayerController** 입력핀을 왼쪽으로 드래그하고 **GetPlayerController** 노드를 배치하자. 그래프가 아래와 같이 완성될 것이다.
그다음, 이벤트 그래프의 격자판의 빈 곳에서 우클릭하고 **Input** » **키보드 이벤트** 아래의 **R** 키 입력 이벤트 노드를 배치하자.
이제 예제를 진행할 준비가 되었다.



5. 이제 타임라인을 추가하고 타임라인을 편집하자.
BP_MyGate 블루프린트 에디터의 이벤트 그래프에서 **R** 입력 이벤트 노드의 **Pressed** 출력 실행핀을 드래그하고 액션선택 창에서 **AddTimeline**을 검색하여 **타임라인 추가**를 찾아 선택하자. 타임라인의 이름은 **GateTimeline**으로 지정하자.
타임라인 노드를 더블클릭하자.

그다음, 타임라인 탭의 툴바에서 **길이** 속성에 2를 입력하자.
그다음, 타임라인 툴바의 가장 왼쪽의 **+트랙**을 클릭하고 **벡터 트랙 추가**를 선택하자. 추가된 트랙의 이름을 **Right**로 수정하자. 다음과 같은 모습으로 보일 것이다.



위의 그림을 보자. **벡터** 트랙은 (X,Y,Z)의 세 좌표값을 세 개의 커브로 표시하는 트랙이다. X,Y,Z를 각각 빨간색,초록색,파란색으로 표시한다. 커브가 하나만 있는 것처럼 보이지만 실제로 세 개의 커브가 겹쳐있는 것이다. 왼쪽 위의 X,Y,Z 옆의 눈 모양의 아이콘을 클릭하면 각 커브의 보이기와 숨기기를 토글할 수 있다. 또한 그 오른쪽의 자물쇠 모양의 아이콘을 클릭하면 해당 커브가 실수로 수정되지 않도록 잠금하는 기능을 토글한다.
우리는 이제 각각의 세 커브를 작성해야 한다.

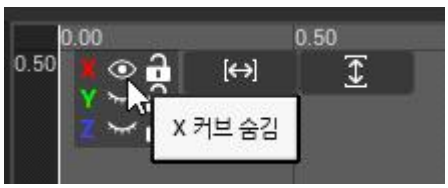
6. 이제 세 커브를 작성해보자.
키를 추가하기 위해서는 타임라인에서 **Shift+좌클릭**하면 된다.
커브 위가 아닌 빈 공간에서 **Shift+좌클릭**하면 세 커브 모두에 키가 추가된다. 그러나 만약 커브 위에서 **Shift+좌클릭**하면 그 커브에 대해서만 키가 추가된다. 여러 커브가 겹쳐져 있다면 가장 위의 커브에 대해서만 키가 추가된다.
한편, 마우스를 **우클릭**하고 팝업메뉴를 통해서도 키를 추가할 수 있다. 커브 위가 아닌 빈 공간에서 우클릭하면 모든 커브에 키를 추가하는 메뉴가 나타난다. **모든 커브에 키를 추가**하는 메뉴와 **모든 커브에 키와 값을 추가**하는 메뉴가 나타나는데 차이점은 마우스 우클릭 시의 수평축 위치만 사용해서 커브의 해당 시간에 추가하는 것인지 또는 수직축 위치까지 사용해서 커브의 해당 시간의 해당 값에 추가하는 것인지의 차이이다. 또한, 빈 공간이 아닌 특정 커브 위에서 우클릭하는 경우에는 **모든 커브에 키 추가** 메뉴가 아닌 **해당 커브에 키 추가** 메뉴가 나타난다.

7. 이제 **DoorRight**만 생각해보자. 처음 1초 동안에는 X축으로 -100 이동하고 그다음의 1초 동안에는 Y축으로 100 이동하면 된다.
따라서, X축 커브는 (0,0), (1,-100), (2,-100)으로 하자. Y축 커브는 (0,50), (1,50), (2,150)으로 하자.

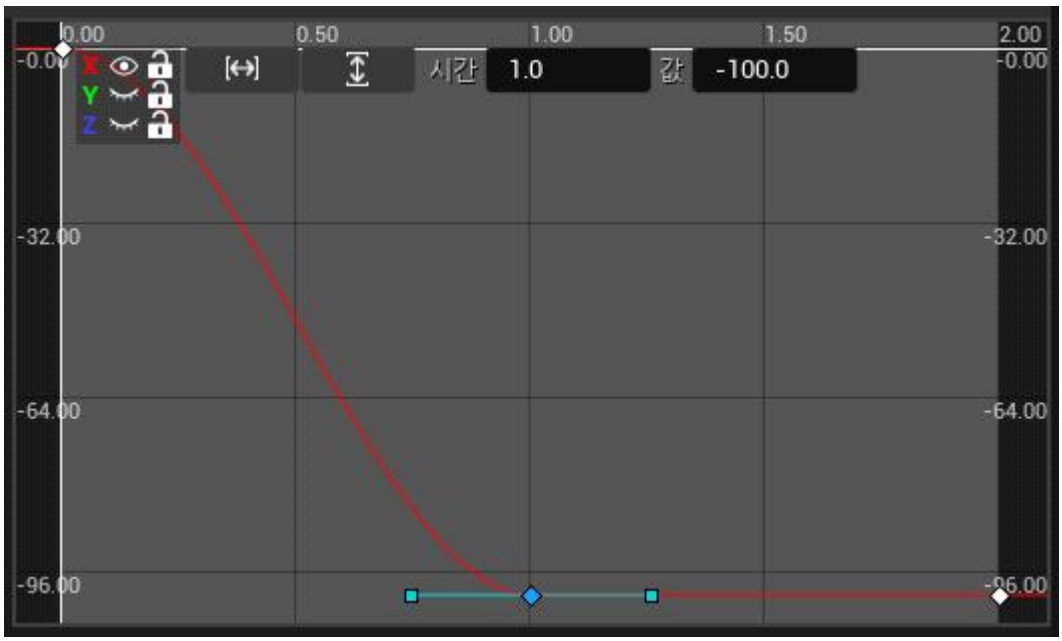
Z축 커브는 (0,0), (1,0), (2,0)으로 하자.
우리는 먼저 세 커브에 대해서 각각 세 개씩의 키를 추가하자. 타임라인의 커브 위가 아닌 빈 지점에서 **Shift+좌클릭**하면 된다. 아래의 그림처럼 될 것이다.

정확한 위치는 신경 쓰지 말고 대략적으로 커브와 떨어진 빈 공간에 클릭하자. 클릭된 후에는 클릭된 위치에 키가 입력되는 것이 아니라 커브 상의 위치로 키가 입력된다. 위의 그림에서 세 커브가 겹쳐서 하나처럼 보이지만 실제로 세 커브에 키가 모두 추가된 것이다.

8. 이제 X축부터 커브를 작성해보자. 먼저 왼쪽 위의 눈 모양의 아이콘을 X만 남겨두고 나머지는 모두 꺼지도록 클릭하여 커브가 보이지 않도록 하자. 사실 더 안전하게 하려면 눈모양이 꺼진 커브에 대해서는 실수로 수정되지 않도록 자물쇠도 잠금으로 바꾸어주면 좋을 것이다. 우리는 자물쇠 사용은 생략하자.

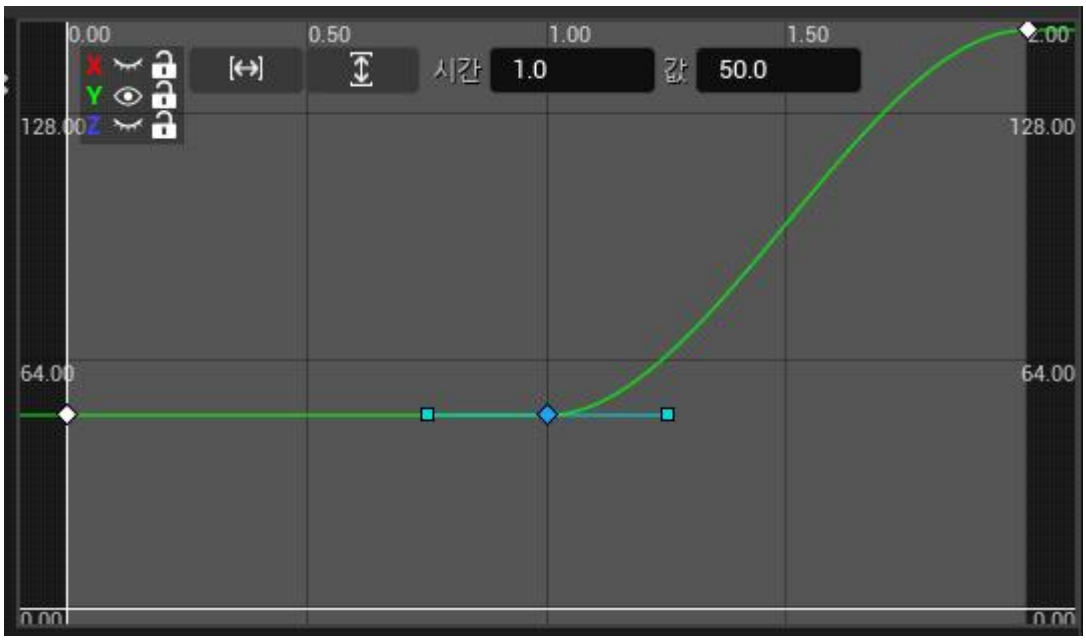


9. X축 커브만 보이도록 한 후에는 각 키를 클릭하고 (**시간**,**값**)을 (0,0), (1,-100), (2,-100)로 입력하자. 입력하는 도중에 키의 위치가 커브의 표시 범위를 벗어날 것이다. 이때에는 가로 세로 맞춤 줌 기능을 사용하거나 마우스 **휠버튼+스크롤**이나 **우클릭+드래그**를 통해서 적당하게 표시 범위를 맞추면 된다.
그다음, 모든 키에 대해서 우클릭하고 **자동**을 선택하고 그다음, 다시 우클릭하고 **평탄화**를 선택하자.

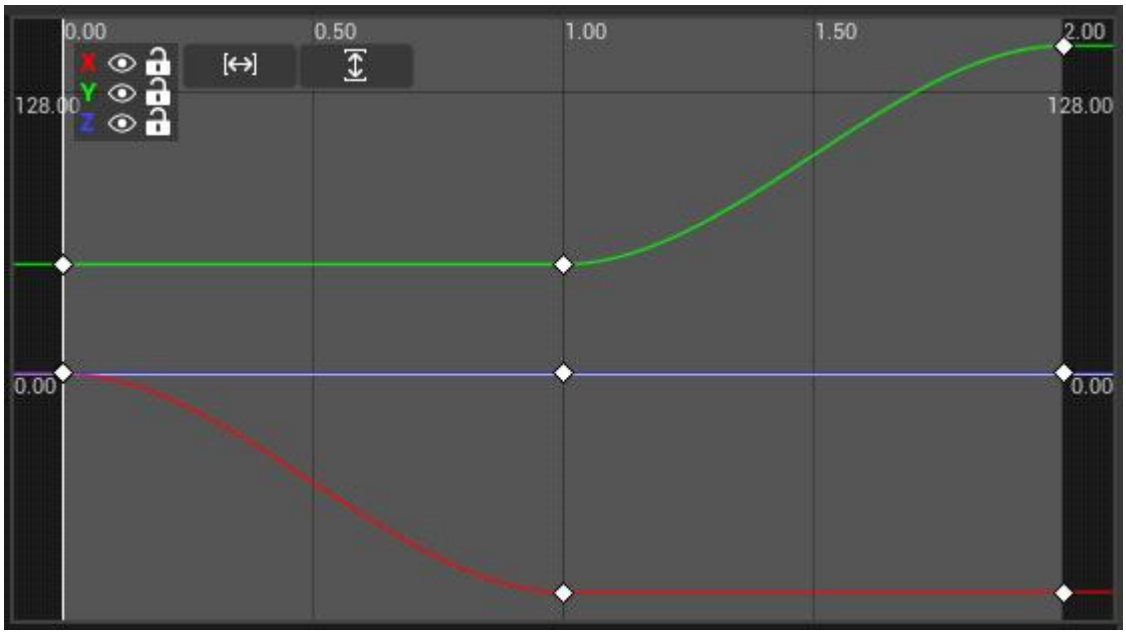


10. 이제 Y축 커브를 작성해보자. 먼저 Y만 보이도록 하자. 그다음, 각 키를 클릭하고 (**시간**,**값**)을 (0,50), (1,50), (2,150)로 입력하자.

그다음, 모든 키에 대해서 우클릭하고 **자동**을 선택하고 그다음, 다시 우클릭하고 **평탄화**를 선택하자.

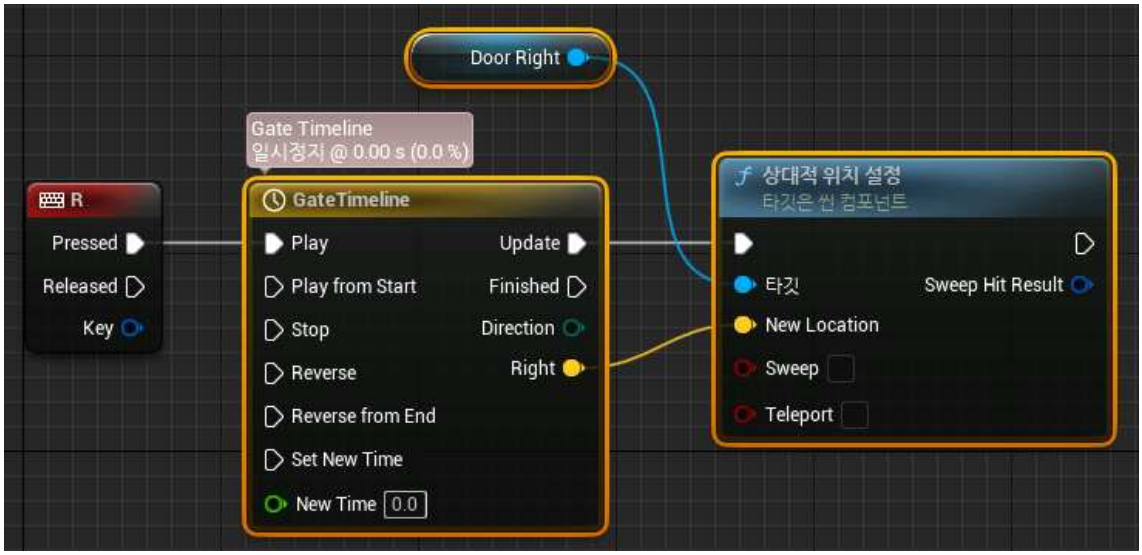


11. 이제 Z축 커브를 작성해보자. 먼저 Z만 보이도록 하자. 그다음, 각 키를 클릭하고 (**시간,값**)을 (0,0), (1,0), (2,0)로 입력하자.
Z축 커브는 사실상 변화가 없이 항상 0인 곡선이다.
이제 세 곡선을 모두 표시하면 아래와 같이 표시된다.



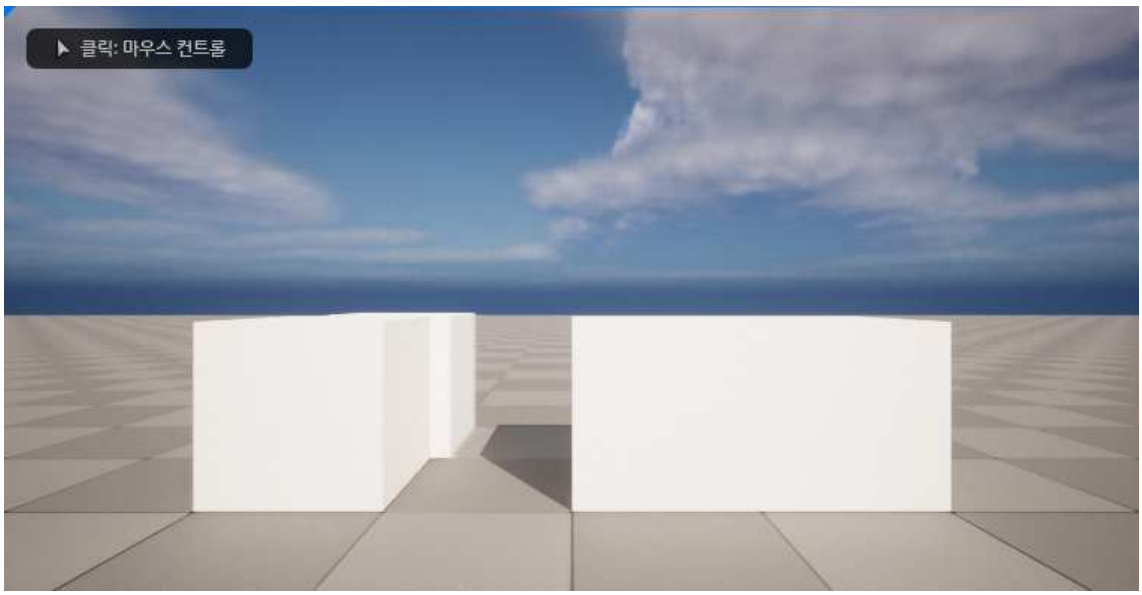
이제 타임라인을 모두 완성하였다.
컴파일하고 저장하자. **GateTimeline** 탭을 닫자.

12. 이벤트 그래프 탭의 **GateTimeline** 노드로 가자. 타임라인 노드의 **Update** 실행핀을 드래그하고 **SetRelativeLocation (DoorRight)** 노드를 배치하자.
그다음, 타임라인 노드의 **Right** 출력핀을 **NewLocation** 입력핀에 연결하자.



컴파일하고 저장하자.

13. 플레이해보자. R 키를 누르면 오른쪽 문이 제대로 움직일 것이다.



14. 이제 왼쪽 문을 완성해보자.

GateTimeline 타임라인에 왼쪽 문을 위한 새로운 벡터 트랙을 추가하고 오른쪽 문의 경우와 유사하게 진행하면 된다.

우리는 새로운 트랙을 추가하는 대신에 기존 트랙을 활용하는 방법으로 진행해보자.

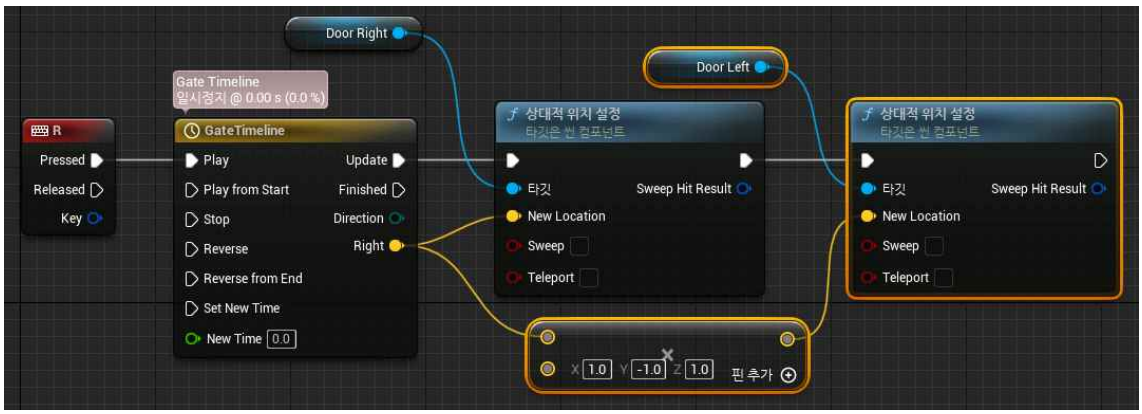
왼쪽 문과 오른쪽 문은 좌우대칭이므로 Y축의 부호만 바뀌고 나머지는 모두 동일하다. 따라서 **GateTimeline** 타임노드의 **Right** 출력값에서 Y값만 부호를 바꾸면 된다. 즉, **Right** 값에 (1,-1,1)을 곱하면 **Left** 값을 얻을 수 있다.

이제 왼쪽 문을 위한 노드 네트워크를 추가해보자.

먼저, **SetRelativeLocation (DoorRight)** 노드의 오른쪽에 **SetRelativeLocation (DoorLeft)** 노드를 배치하고 실행핀을 연결하자.

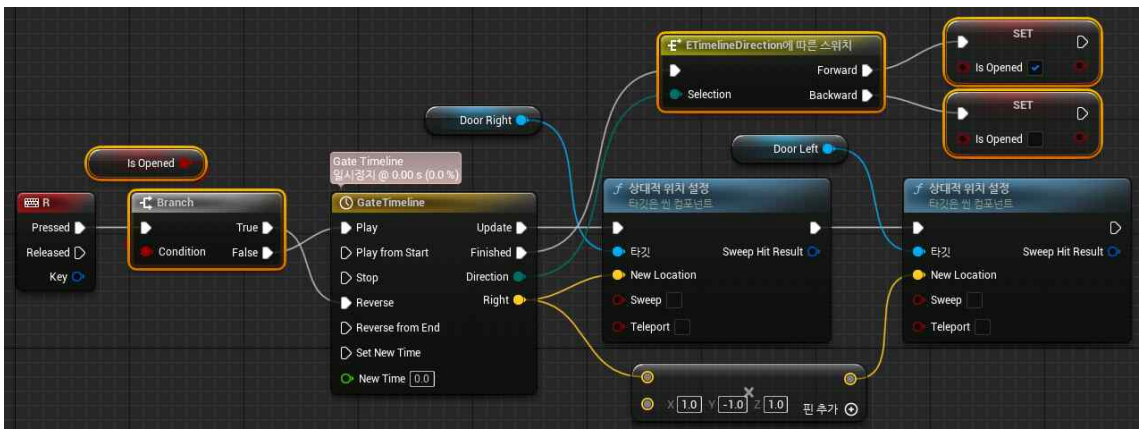
그다음, **GateTimeline** 노드의 **Right** 값에 (1,-1,1)을 곱한 값을 **SetRelativeLocation (DoorRight)** 노드의의

NewLocation 입력핀에 연결하자.



컴파일하고 저장하자.
플레이해보자. R 키를 누르면 오른쪽 문과 왼쪽 문이 모두 제대로 움직일 것이다.

15. 이번에도 R 키를 누르면 문이 열리고 닫히는 것을 토글하도록 해보자. 먼저 **Boolean** 유형의 **IsOpened** 변수를 추가하자. 초기값은 그대로 두면 **false**이다. 처음 R 키를 누르면 **IsOpened**가 **false**이고 문을 여는 순방향으로 타임라인을 진행한다. 그리고 타임라인이 종료된 후에는 **IsOpened**를 **true**로 한다. 다음에 다시 R 키를 누르면 타임라인을 역방향으로 재생한다. 재생후에는 다시 **IsOpened**를 **false**로 바꾼다. 완성된 그래프는 다음과 같은 모습이다.



16. 플레이해보자. R 키를 반복해서 눌러보자. 문이 닫히고 열리는 것을 번갈아 진행할 것이다.

지금까지, 타임라인에서 벡터 트랙을 사용하는 방법에 대해서 학습하였다.

□