20_ 내비게이션

<제목 차례>

인천대학교 컴퓨터공학부 박종승

1. 개요

이 장에서부터는 인공지능을 시작한다. 인공지능의 시작을 위해서 이 장에서는 먼저 내비게이션에 대해서 학습한다.

인공지능의 구현을 위해서는 적 캐릭터의 폰과 이를 제어하는 AI 컨트롤러를 만들어야 한다. **폰**과 **컨트롤러**의 관계에 대해서 알아보자. 폰은 게임의 등장인물의 몸체에 해당하고 컨트롤러는 그 몸체의 뇌에 해당한다. 물리적인 형상이나 간단한 반응은 몸체에서 처리하고 중요한 지시나 움직임은 뇌에서 처리하는 관계로 이해하자.

폰(pawn)은 플레이어나 AI가 제어할 수 있는 모든 액터의 기반 클래스로 월드 내에서 플레이어나 AI 객체에 대한 물리적인 표현에 해당한다. 물리적인 표현이라 함은 플레이어나 AI 객체의 시각적인 모습과 더불어 콜리전이나 물리 시뮬레이션 반응에 관계된 월드와의 상호작용도 포함한다. 캐릭터는 이족보행 이동 기능의 컴포넌트를 추가한 폰의 자식 클래스이다.

컨트롤러(controller)는 폰의 파생 클래스를 빙의하여 그 동작을 제어할 수 있는 액터이다. 컨트롤러의 자식 클래스에는 사람이 조정하는 폰에 사용되는 **플레이어 컨트롤러**(**PlayerController**)가 있고, 사람이 아닌 인공지능이 조정하는 폰에 사용되는 **AI 컨트롤러**(**AIController**)가 있다. 따라서 적 캐릭터의 인공지능은 바로 이 AI 컨트롤러 내에서 구현하게 된다.

폰과 컨트톨러의 관계는 일대일 대응 관계에 있다. 컨트롤러는 항상 하나의 폰만 제어한다. 컨트롤러가 폰의 제어권을 획득하는 절차를 **빙의**(possess)라고 한다. 폰이 생성될 때에는 설정에 따라 명시된 컨트롤러에 의해 빙의될 수 있다. 그러나 플레이 도중에 스폰된 폰은 자동적으로 컨트롤러에 빙의되지 않는다. 컨트롤러는 **Possess** 함수로 폰의 제어권을 획득하며 **Unpossess** 함수로 제어권을 해제한다. 폰에서 발생하는 입력 이벤트나 콜리전 이벤트 등의 대부분의 이벤트에 대해서 그 폰을 제어하는 컨트롤러에게도 통지된다. 컨트롤러는 그 이벤트에 대해서 폰에 우선하여 처리할 수 있다. 즉 동일한 이벤트 그래프가 폰과 컨트롤러에 있다면 컨트롤러의 이벤트 그래프가 실행되고 폰의 이벤트 그래프가 무시된다. 폰과 컨트롤러 중에서 이벤트 처리를 어디에서 구현할 것인가는 개발자가 선택할 몫이다. 그러나 단순하고 국부적인 이벤트 처리의 구현은 폰에서 담당하도록 해도 되겠지만 게임 진행과 관련된 이벤트 처리라면 컨트롤러에서 담당하는 것이 바람직하자.

<참고> 폰과 컨트롤러의 관계에 대해서 다음의 게임 프레임워크 문서를 참조하자.
https://docs.unrealengine.com/gameplay-framework-in-unreal-engine/

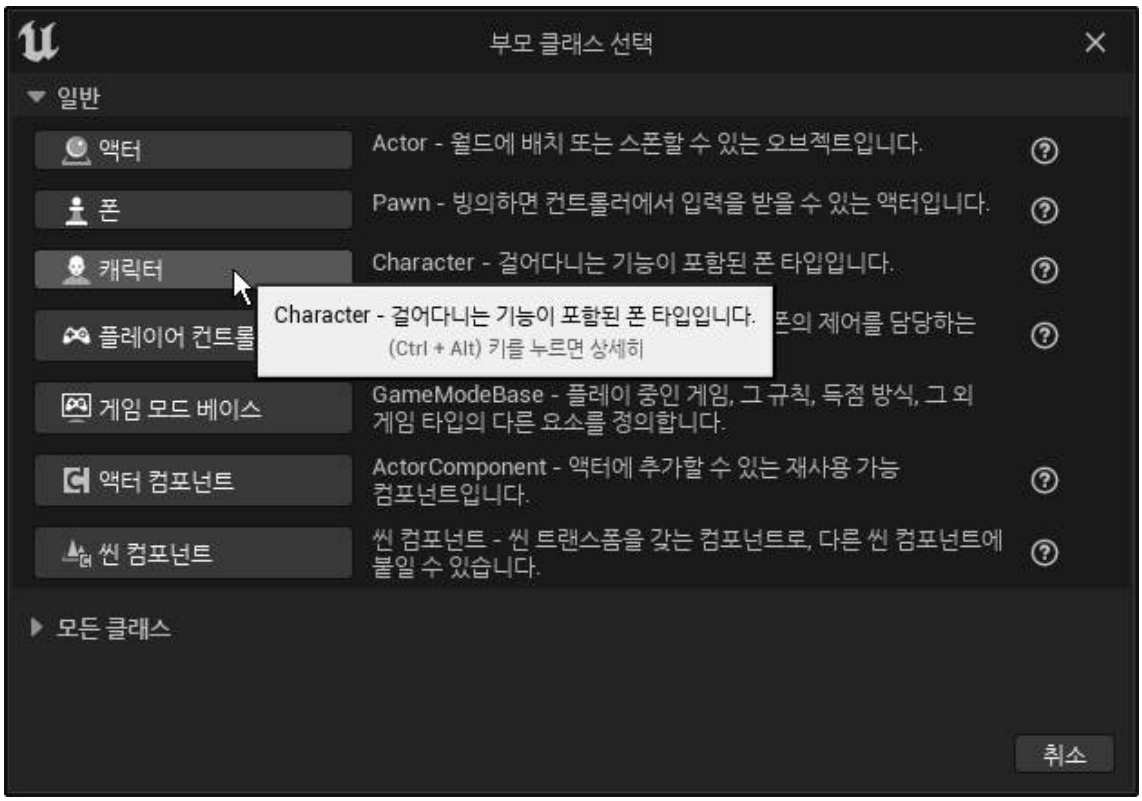
2. 적 캐릭터와 AI 컨트롤러 만들기

이 절에서 적 캐릭터와 AI 컨트롤러를 만드는 방법을 학습한다.
이제부터 예제를 통해서 학습해보자

---

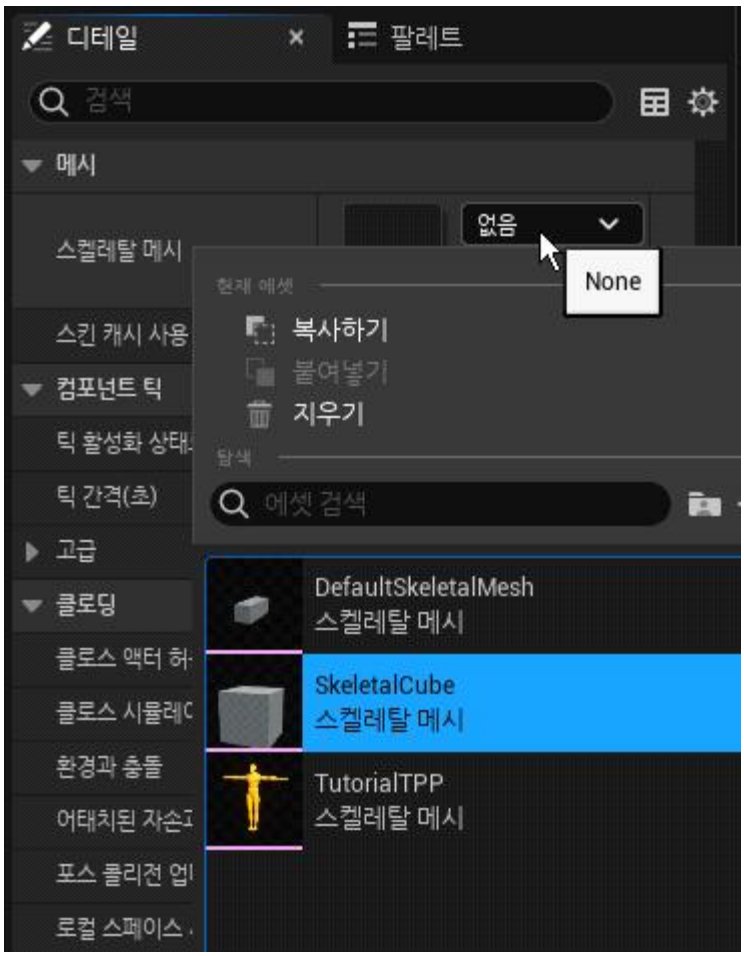
1. 새 프로젝트 **Paimoveto**를 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 Paimoveto로 입력하고 **생성** 버튼을 클릭하자. 프로젝트가 생성되고 언리얼 에디터 창이 뜰 것이다. 창이 뜨면, 메뉴바에서 **파일** » **새 레벨**을 선택하고 **Basic**을 선택하여 기본 템플릿 레벨을 생성하자. 그리고, 레벨 에디터 툴바의 저장 버튼을 클릭하여 현재의 레벨을 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 (0,0,92)로 지정하자.

2. 적 캐릭터의 폰을 만들자.
먼저, **콘텐츠 브라우저**에서 **+추가**를 클릭하고 **블루프린트 클래스**를 선택하자. **부모 클래스 선택** 창에서 부모 클래스로 **Character**를 선택하자. 생성된 블루프린트 클래스 이름을 **EnemyPawn**으로 하자.



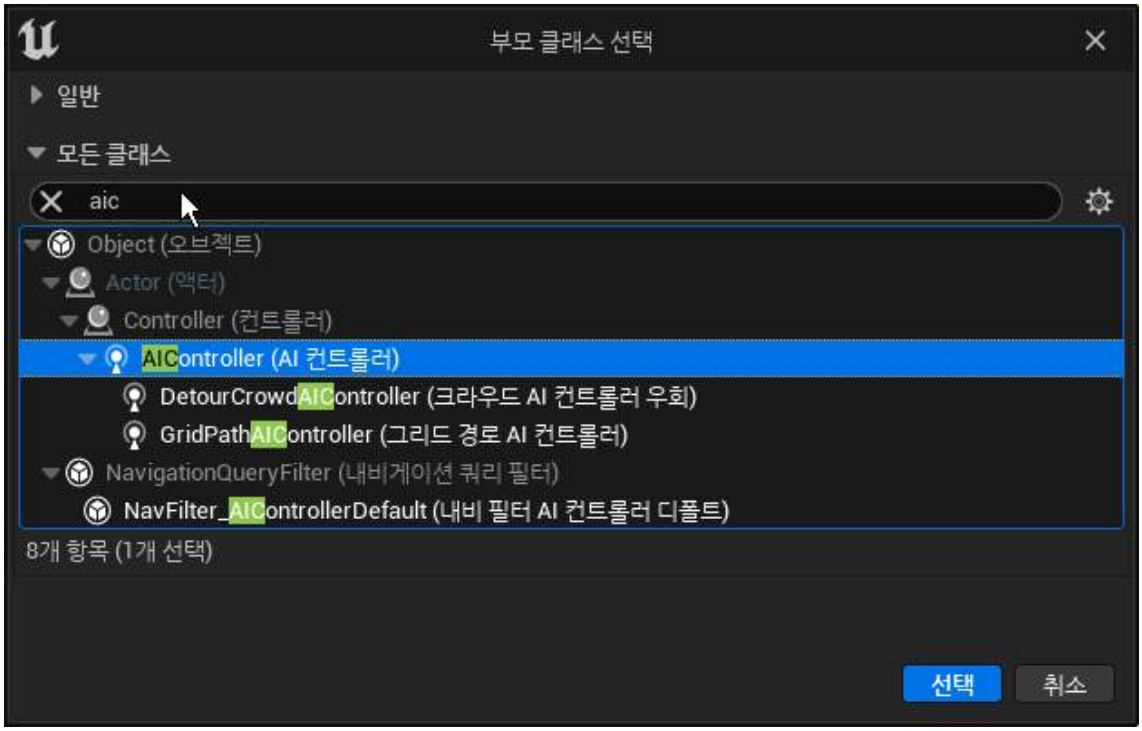
3. **EnemyPawn**을 더블클릭하여 블루프린트 에디터를 열자.
그다음, 왼쪽의 **컴포넌트** 탭에서 **메시(Mesh)** 컴포넌트를 선택하고 오른쪽의 **디테일** 탭에서 **메시** 영역의 **Skeletal Mesh** 속성을 찾아보자. 속성값이 **없음**으로 되어 있을 것이다. 여기에 폰의 스켈레탈 메시 애셋을 지정하면 된다. 우리는 따로 스켈레탈 메시를 제작하지 않았으므로, 엔진에서 제공하는 큐브

모양의 스켈레탈 메시인 **SkeletalCube**를 선택하자.



이제 적 캐릭터의 폰을 생성하였다.
컴파일하고 저장하자.

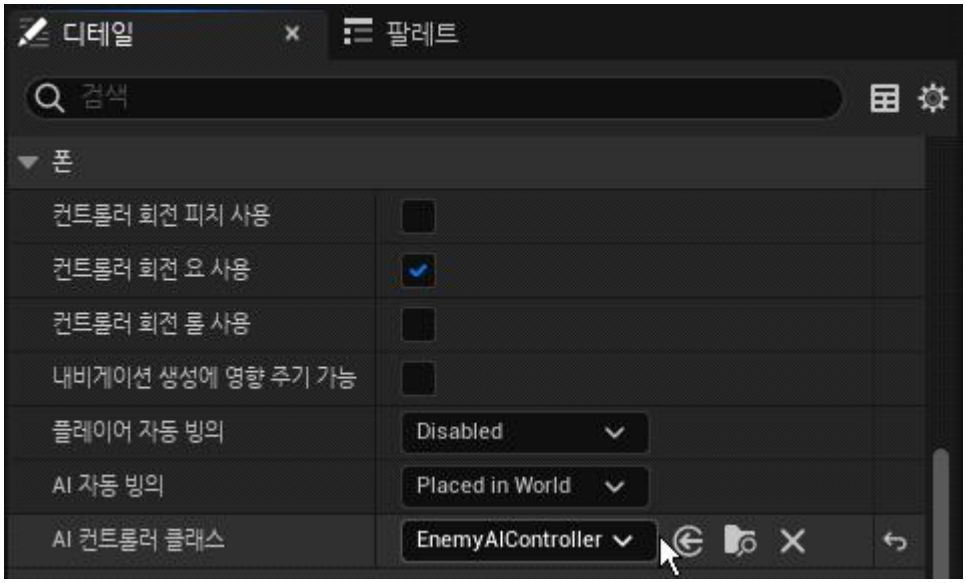
4. 적 캐릭터를 제어하는 AI 컨트롤러를 생성하자.
먼저, **콘텐츠 브라우저**에서 **+추가**를 클릭하고 **블루프린트 클래스**를 선택하자. **부모 클래스 선택** 창에서 **일반** 영역 아래에 있는 **모든 클래스** 영역을 펼치자. 부모 클래스로 **AIController**를 검색하여 선택하자. 생성된 블루프린트 클래스 이름을 **EnemyAIController**으로 하자.



이제 적 캐릭터의 폰을 제어하는 AI 컨트롤러를 생성하였다.
컴파일하고 저장하자.

5. 레벨의 플레이에서 **EnemyPawn**이 처음으로 생성될 때에 **EnemyAIController**에 의해서 빙의되도록 하자.

EnemyPawn을 더블클릭하여 블루프린티 에디터를 열자.
툴바의 **클래스 디폴트** 버튼을 클릭하거나 또는 컴포넌트 탭에서 루트 노드인 **EnemyPawn (셀프)**를 선택하자.
그다음, **디테일** 탭에서 **폰** 영역에 있는 **AI 컨트롤러 클래스(AI Controller Class)** 속성을 찾아보자. 속성값이 **AIController**로 되어있을 것이다. 이 속성값은 레벨에 배치된 **EnemyPawn**이 생성될 때에 자동으로 빙의되도록 할 **AI 컨트롤러** 클래스를 명시한다. 디폴트 값은 **AI 컨트롤러**의 최상위 기반 클래스인 **AIController**로 되어 있다. 따라서, **EnemyPawn**이 최초로 생성될 때에 디폴트로 **AIController** 클래스가 생성되고 **EnemyPawn**을 빙의할 것이다. 만약 **None**으로 설정하면 **AI 컨트롤러** 클래스가 생성되지 않고 빙의되지도 않는다.
우리는 속성값을 **AIController**에서 **EnemyAIController**로 수정하자. 이제 레벨에 배치된 **EnemyPawn**이 생성될 때에 자동으로 **EnemyAIController**가 생성되어 **EnemyPawn**을 빙의할 것이다.
컴파일하고 저장하자.



<참고> 만약 이 **AI Controller Class** 속성이 보이지 않는다면 컴포넌트 탭에서 루트 노드가 아닌 다른 자식 컴포넌트가 선택되어 있는지 확인해보자. 이 속성은 세부 컴포넌트 속성이 아닌 액터 수준의 속성이므로 컴포넌트 탭에서 가장 상위의 **EnemyPawn (셀프)**가 선택되어 있어야 보인다. 또는 툴바에서 **클래스 디폴트** 버튼을 클릭해도 된다.

<참고> 폰의 **AI Controller Class** 속성값은 레벨이 플레이될 때에 스폰되는 폰 인스턴스에만 영향을 미친다. 만약 플레이된 후에 폰 액터를 스폰하는 경우에는 **AI 컨트롤러** 클래스가 자동으로 생성되지 않는다. 이 경우에는 기존의 **AI 컨트롤러**나 또는 새로운 **AI 컨트롤러**를 스폰한 후에 **AI 컨트롤러**의 빙의 함수인 **Possess** 함수를 호출하여 연결하면 된다.

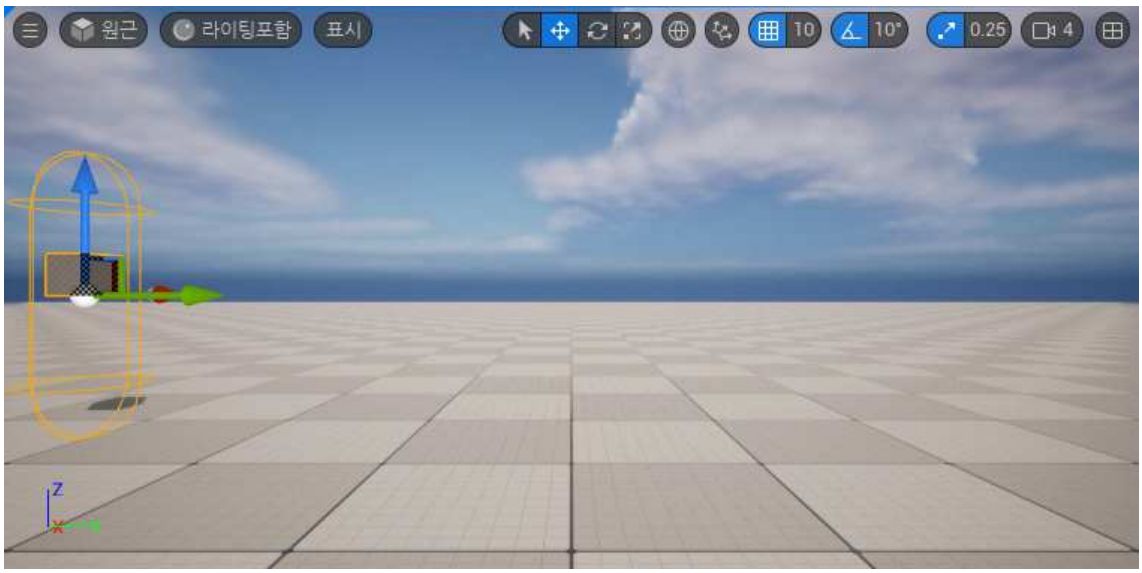
6. **EnemyPawn** 블루프린트 에디터의 **디테일** 탭에서 **폰** 영역에는 **컨트롤러 회전 피치/요/롤 사용**(**Use Controller Rotation Pitch/Yaw/Roll**) 속성이 있다. 디폴트로 **컨트롤러 회전 요 사용**(**Use Controller Rotation Yaw**)만 체크되어 있을 것이다. 우리는 바꾸지 않고 그대로 두자.

<참고> 폰의 속성인 **UseControllerRotationPitch,Roll,Yaw** 속성에 대해서 알아보자. 폰에도 회전 정보가 있고 컨트롤러에도 회전 정보가 있다. 이 속성으로 이들을 하나로 일치시킬 것인지를 결정할 수 있다.
컨트롤러는 내부에 자체적으로 회전값을 가지고 있다. 이 회전값은 제어하는 폰의 바라보는 방향을 결정하는 용도로 사용되며, 입력으로부터 이 회전값을 수정하여 폰의 방향을 바꾸는 것이 일반적이다. 한편, 폰도 액터이므로 폰 자체의 회전 방향을 위해서 별도로 회전값을 가진다. 그러나 폰의 자체 회전값을 별도로 제어하기보다는 일반적으로 폰의 회전은 컨트롤러 회전값과 연동하도록 한다.
플레이어 컨트롤러가 폰을 제어할 때에, 플레이어 컨트롤러의 회전값에 따라서 폰도 함께 회전되도록 할 수 있다. 이를 위해서는, 폰의 디테일 속성에 **UseControllerRotationPitch,Roll,Yaw**가 있다. 이 속성을 **true**로 하면 컨트롤러와 폰의 회전을 일치시키다.
일반적으로 두 회전은 일치되도록 하여 사용한다. 특히 일인칭 시점의 경우는 일치시키는 것이 일반적이다. 특히 요 회전은 폰이 향하는 방향을 바꾸는 회전에 해당하고 자주 사용되며 대부분 플레이어 컨트롤러의 회전과 일치시킨다.
한편, 삼인칭 시점에서는 일인칭에서의 경우와는 다르게 컨트롤러가 회전되더라도 캐릭터가 함께 회전되지 않도록 **UseControllerRotationPitch,Yaw,Roll**을 모두 **false**로 지정한다. 이렇게 하면 컨트롤러의 회전에 따라 카메라만 회전하고 캐릭터는 회전하지 않게 된다. 이것이 삼인칭에서의 주요한 차이점이라고 볼 수 있다.

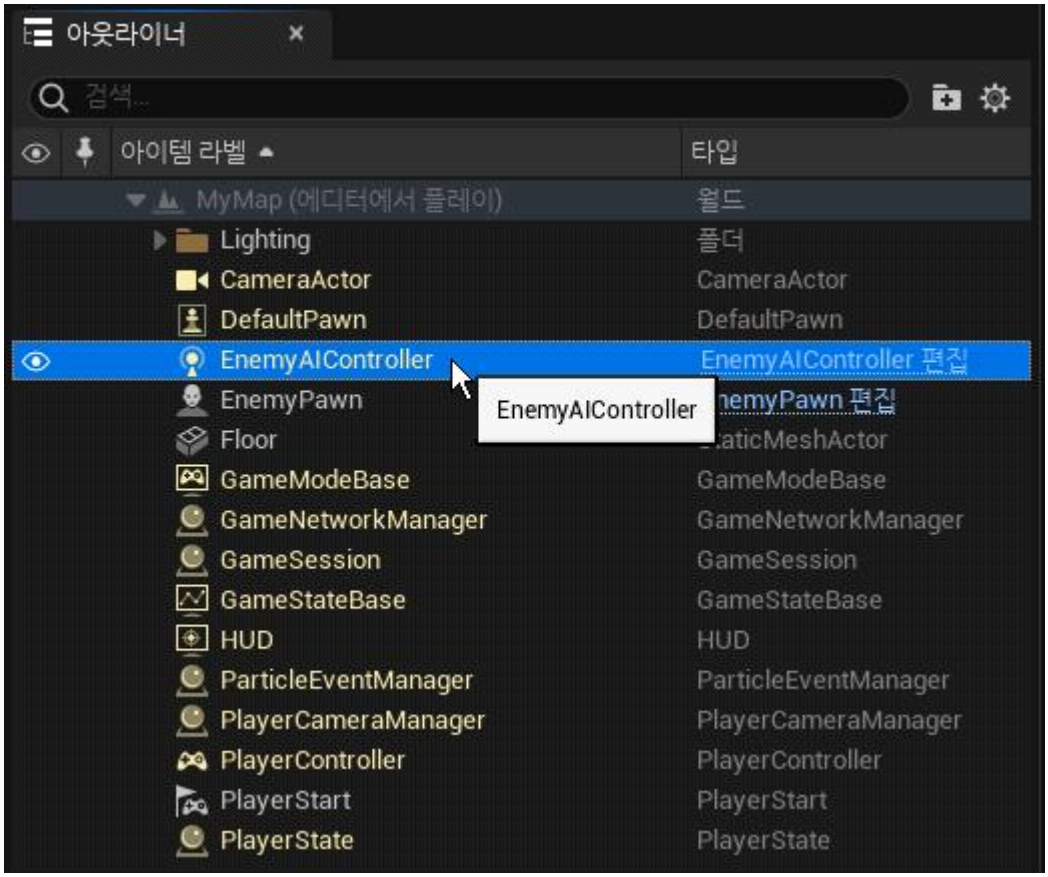
7. 컴포넌트 탭에서 **캐릭터 무브먼트**(**CharacterMovement**) 컴포넌트를 선택하고 **디테일** 탭에서 **캐릭터**

무브먼트 (회전 세팅) 영역에 있는 **무브먼트 방향으로 회전 조정(Orient Rotation to Movement)** 속성이 있다. 디폴트로는 체크되어 있지 않을 것이다. 우리는 여기에 체크하자. 이 속성에 체크하면 폰이 움직이는 방향으로 향하도록 폰을 회전시켜준다.

8. 이제, **콘텐츠 브라우저**에서 **EnemyPawn**을 드래그하여 레벨에 배치하자. 위치는 (350,-300,88)에 배치하자.



9. 플레이해보자. 플레이 중에 **아웃라이너**를 살펴보자. **EnemyAIController** 인스턴스가 생성된 것을 확인할 수 있다.



이 절에서는 적 캐릭터와 AI 컨트롤러를 만드는 방법을 학습하였다.

3. 내비 메시를 배치하고 폰을 이동시키기

이 절에서 내비게이션과 내비 메시에 대해서 학습한다.
먼저 내비 메시를 만드는 방법을 알아본다. 그다음, 레벨 블루프린트에서 **AIMoveTo** 함수를 사용하여 폰을 이동시키는 방법을 알아본다.

배경 요소를 고려하면서 캐릭터를 명시한 목적지 위치로 이동시키는 기능을 내비게이션 기능이라고 한다. 내비게이션은 인공지능에서 가장 기본적인 기능이다.
캐릭터의 이동 가능한 영역을 나타내는 메시를 **내비게이션 메시**라고 한다. 줄여서 **내비 메시**라고도 한다. 엔진에서는 현재 레벨로부터 **내비 메시**를 만들고 그 정보를 사용하여 캐릭터가 목적지까지 자동으로 이동되도록 하는 기능을 제공한다.
내비 메시를 만드는 방법을 알아보자. 내비 메시는 엔진에서 현재의 레벨로부터 자동으로 생성한다. 따라서 우리는 내비 메시의 생성에 대해서 신경쓸 필요가 없다.
다만, 내비 메시를 만들고 싶은 공간의 범위를 지정해주어야 한다. 이 공간 범위는 **내비 메시 바운드 볼륨**(Nav Mesh Bounds Volume)이라는 볼륨으로 지정한다. 우리는 캐릭터가 이동할 수 있는 대상 영역이 포함되도록 적절한 크기의 **내비 메시 바운드 볼륨**을 배치해주면 된다.

이제부터 예제를 통해서 학습해보자

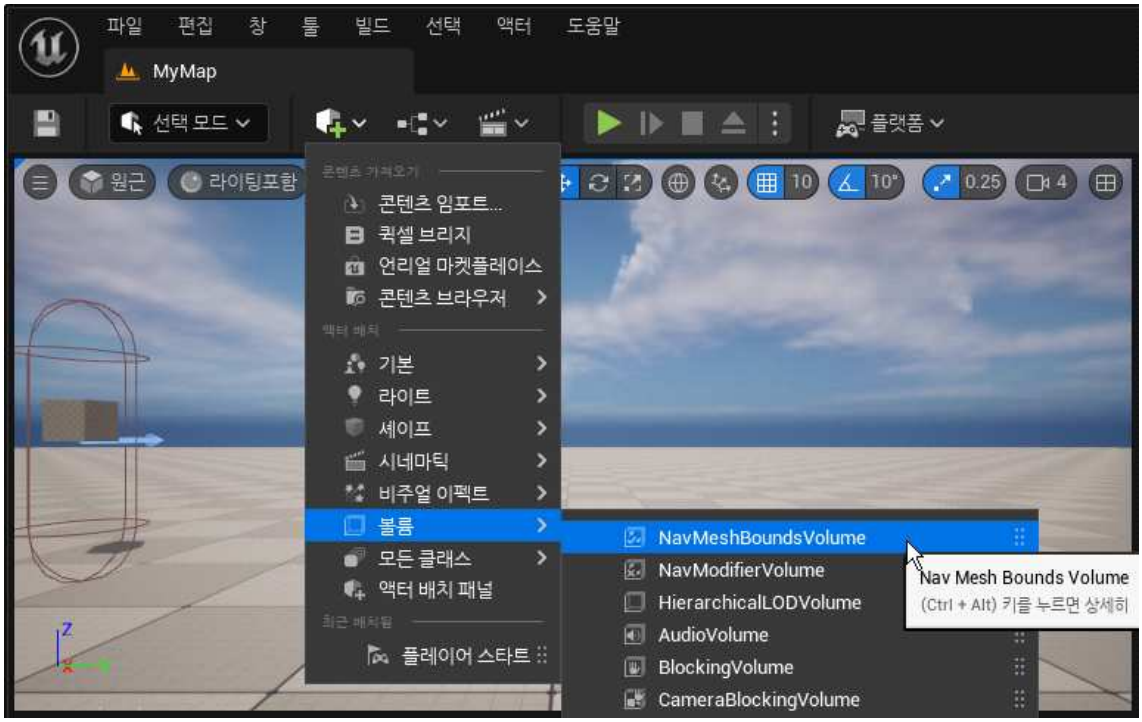
---

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

2. 레벨의 모습을 보다 쉽게 알아볼 수 있도록 전체 공간을 더 작게 만들자.
먼저, **아웃라이너**에서 **Floor**를 선택하자. 그다음, **디테일** 탭에서 **스케일**을 (8,8,8)에서 (1,1,1)로 수정하자.

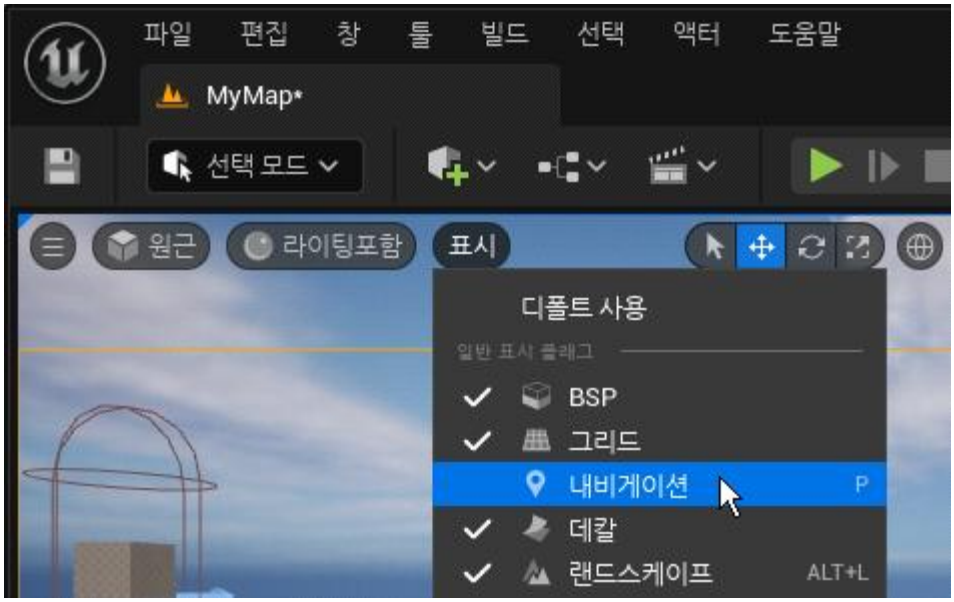
3. 이제, **내비 메시**가 생성되도록 해보자.
레벨에 **내비 메시 바운드 볼륨**(NavMeshBoundsVolume)이 배치되어 있으면 엔진은 자동으로 **내비 메시**를 생성해준다.

툴바의 **액터 배치** 아이콘을 클릭하고 **볼륨** 아래의 **NavMeshBoundsVolume**을 드래그하여 레벨에 배치하자.
또는, **액터 배치** 탭에서 **볼륨** 탭을 클릭하고 **NavMeshBoundsVolume**을 드래그해서 레벨에 배치해도 된다.
배치된 인스턴스의 이름은 디폴트인 **NavMeshBoundsVolume**으로 그대로 두자.

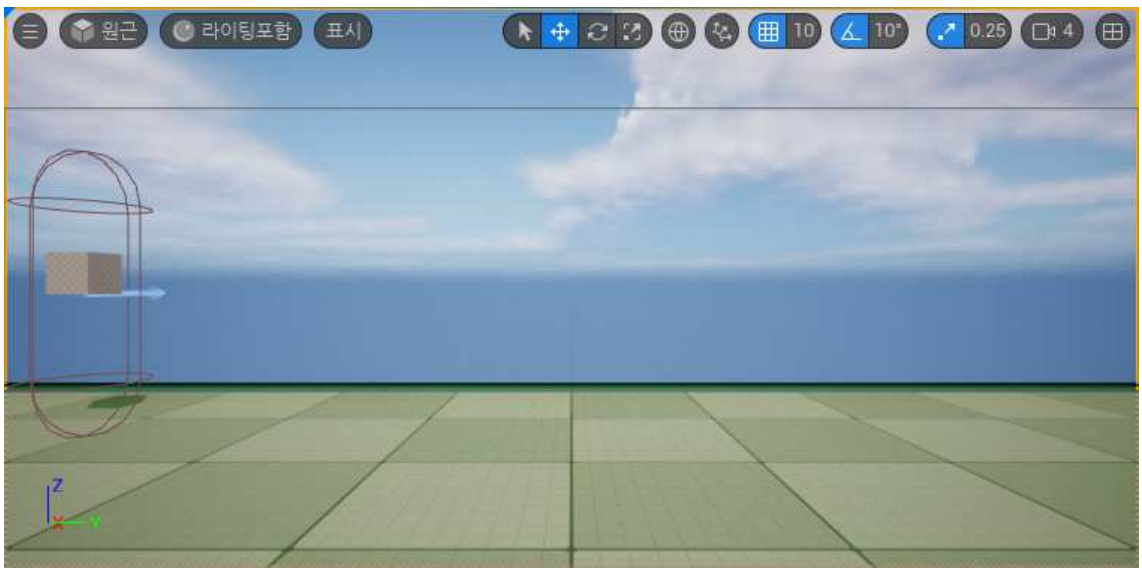


배치된 **내비 메시 바운드 볼륨**이 바닥 영역과 우리가 앞으로 배치할 배경 영역을 충분히 포함하도록 크기와 위치를 조절하자. 디테일 탭에서 **위치**는 (0,0,100)으로 하고, **스케일**은 (5,5,1.5)로 입력하자.

4. 이제, **뷰포트** 상단의 **표시**를 클릭하고 드롭다운 메뉴에서 **내비게이션**을 선택하자. 이 메뉴가 **내비 메시**를 표시하는 토글 메뉴이다. 뷰포트 내에서 키보드 **P** 키를 눌러도 된다. 자주 사용되는 메뉴이므로 잘 알아두자.

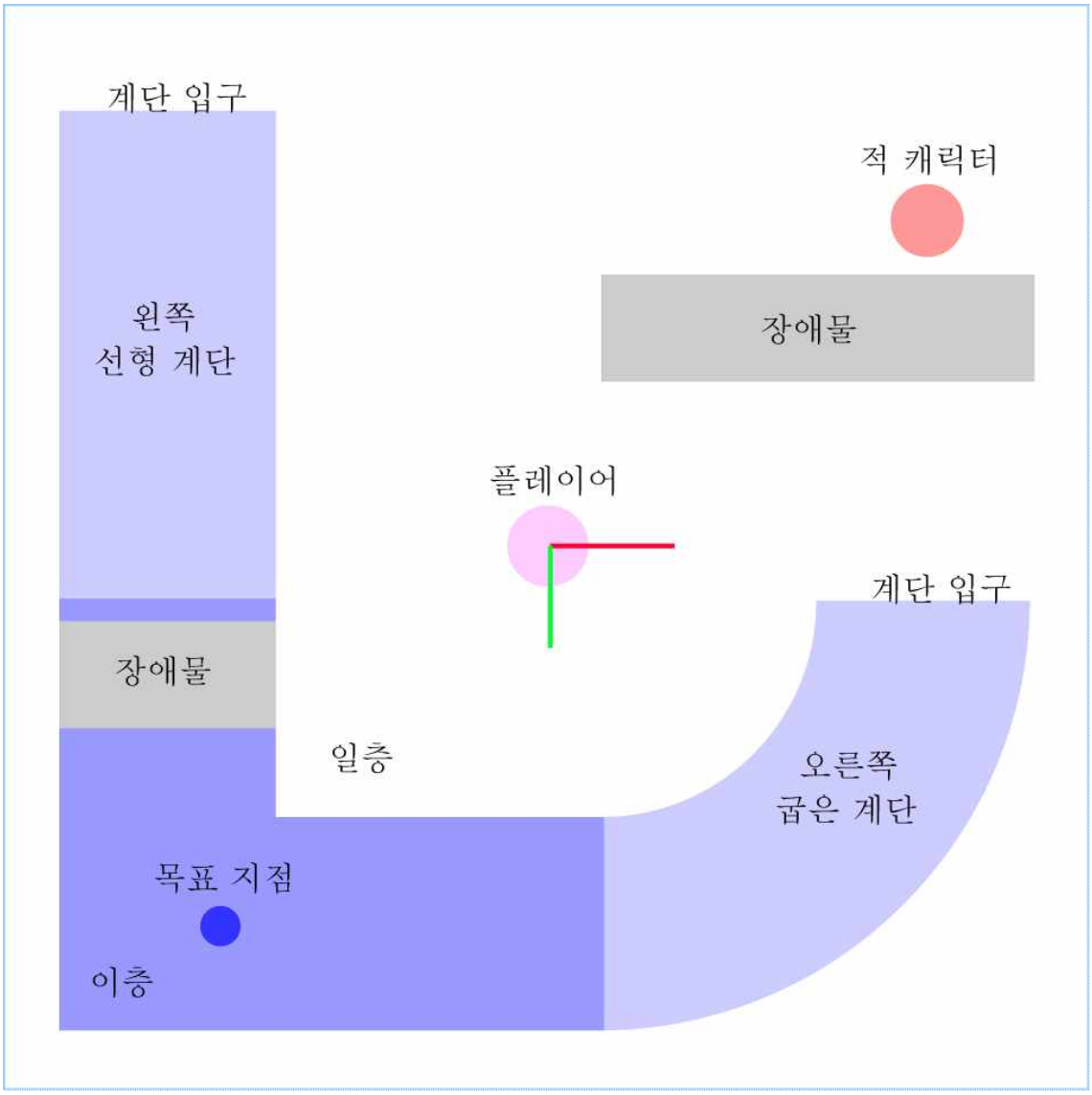


5. 이제 바닥 평면 위가 녹색으로 표시될 것이다. 녹색 영역이 바로 이동 가능 영역에 해당한다.

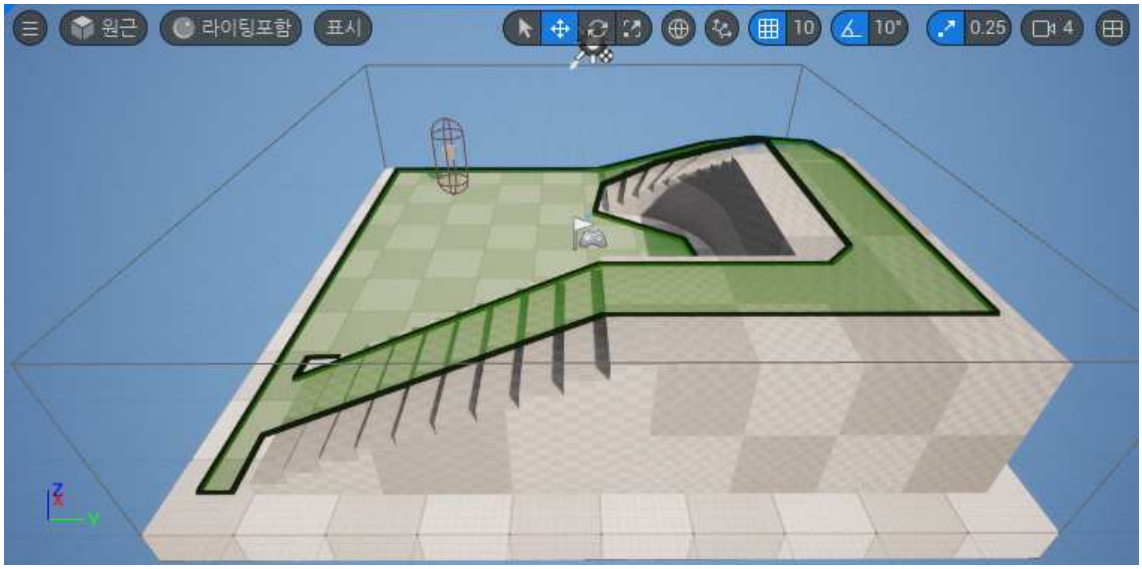


6. 이제 레벨을 디자인해보자.

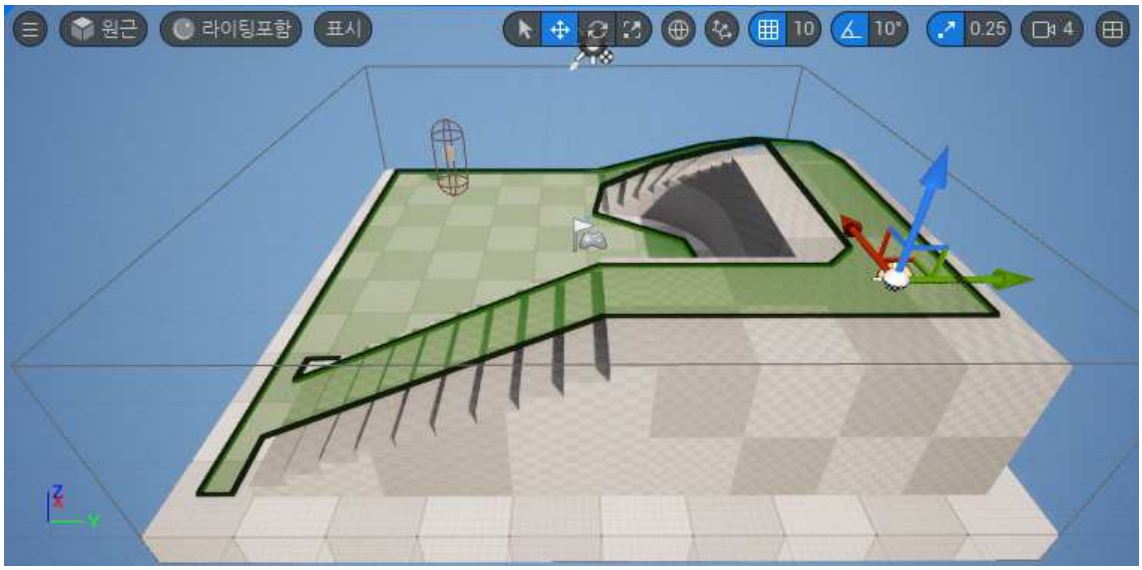
전면 오른쪽에 이층으로 올라갈 수 있도록 굽은 계단을 배치하자. 계단 위에서는 이층 복도로 이어지도록 하자. 이층 복도는 모서리에서 꺾이면서 이어지도록 하자. 그 후에는 다시 일층으로 내려올 수 있는 선형 계단을 배치하자.



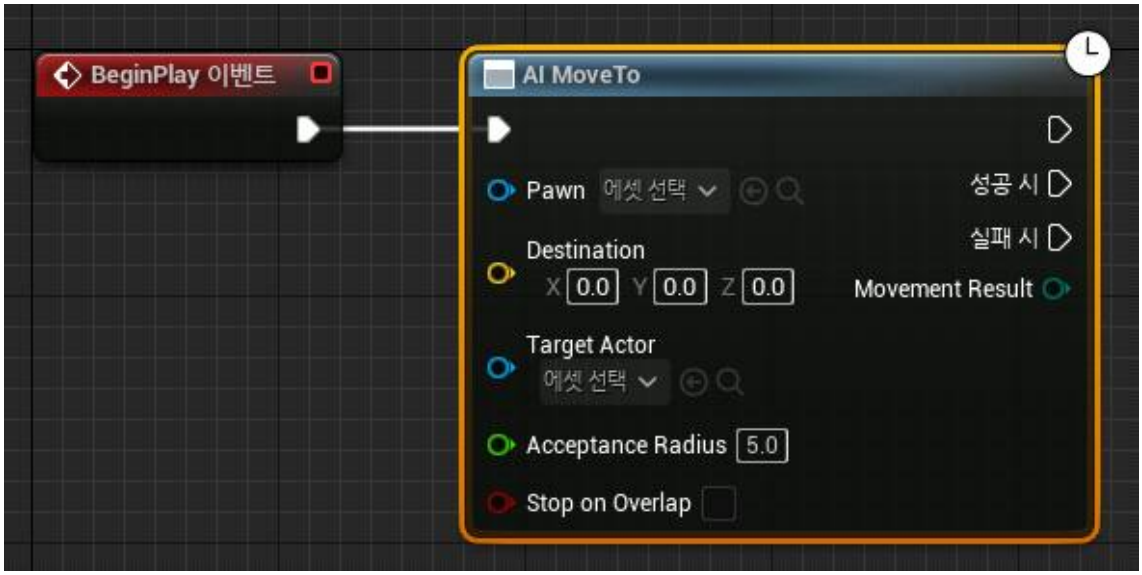
7. 이제 레벨에 배치해보자.
메뉴바에서 **창** » **액터 배치**를 선택하여 **액터 배치** 탭이 보이도록 하자.
그다음, **액터 배치** 탭에서 **지오메트리** 탭을 클릭하자.
먼저, **굽은 계단**을 드래그하여 배치하자. 인스턴스의 이름은 **CurvedStairRight**로 수정하자. **위치**는 (50,50,0)으로 지정하자.
그다음, **박스**를 드래그하여 배치하자. 이름은 **UpstairCorridorRight**로 수정하자. **위치**는 (-200,350,100)으로 지정하자. **스케일**은 (2.5,1,1)으로 지정하자.
그다음, 또다른 **박스**를 드래그하여 배치하자. 이름은 **UpstairCorridorRear**로 수정하자. **위치**는 (-350,150,100)으로 지정하자.
그다음, **선형 계단**을 드래그하여 배치하자. 이름은 **LinearStairRear**로 수정하자. **위치**는 (-250,-400,20)으로 지정하자. **회전**은 (0,0,90)으로 지정하자. **스케일**은 (1.5,1.1)로 지정하자. 계단의 경사도가 너무 높으면 올라가지 못할 수도 있으므로 경사도를 낮추었다.
배치할 때마다 **내비 메시**도 함께 바뀌는 것을 볼 수 있다.



8. 이제 이층의 모서리에 지점 표시를 해두자.
액터 배치 탭에서 **타깃 포인트(TargetPoint)**를 검색하여 **타깃 포인트** 액터를 레벨로 드래그하여 배치하자. 배치된 인스턴스의 이름을 **TPUpstairCorridorCorner**로 수정하자. **위치**는 (-350,350,200)으로 하자. 이층 복도의 꺾이는 모서리 위치에 배치될 것이다.

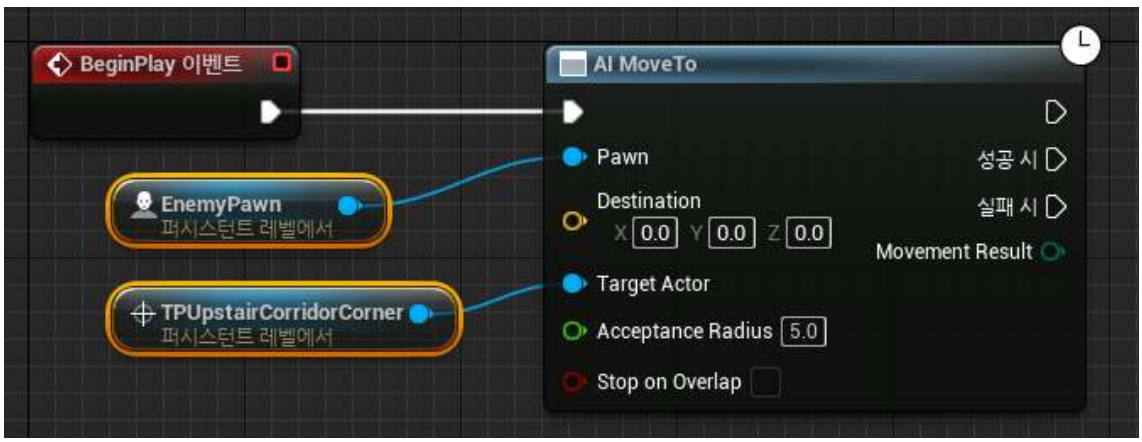


9. 이제, **툴바**의 **블루프린트** » **레벨 블루프린트 열기**를 선택하여 레벨 블루프린트 에디터를 열자. 그리고, 레벨 블루프린트 에디터에서 이벤트 그래프 탭의 **BeginPlay 이벤트** 노드로 이동하자. 실행핀을 드래그하고 액션선택 창에서 **AIMoveTo** 노드를 배치하자.



노드를 살펴보자. 먼저, **Pawn** 입력핀에는 이동할 대상 폰을 입력한다. 그다음, **Destination** 입력핀에는 목적지의 위치 벡터를 입력한다. 그다음, **TargetActor** 입력핀은 목표물 액터의 레퍼런스를 입력한다. **Destination**을 입력하든지 또는 **TargetActor**를 입력하든지 하나만 입력하면 된다.

10. 그다음, 레벨 에디터의 **아웃라이너**에서 **EnemyPawn**을 드래그하여 이벤트 그래프 격자판의 빈 곳에 배치하자. 그리고, 배치된 **EnemyPawn** 레퍼런스 노드의 출력핀을 **AIMoveTo** 노드의 **Pawn** 입력핀에 연결하자.
그다음, **아웃라이너**에서 **타깃 포인트**인 **TPUpstairCorridorCorner**를 드래그하여 격자판의 빈 곳에 배치하자. 그리고, 격자판에 배치된 **TPUpstairCorridorCorner** 레퍼런스 노드의 출력핀을 **AIMoveTo** 노드의 **TargetActor** 입력핀에 연결하자.

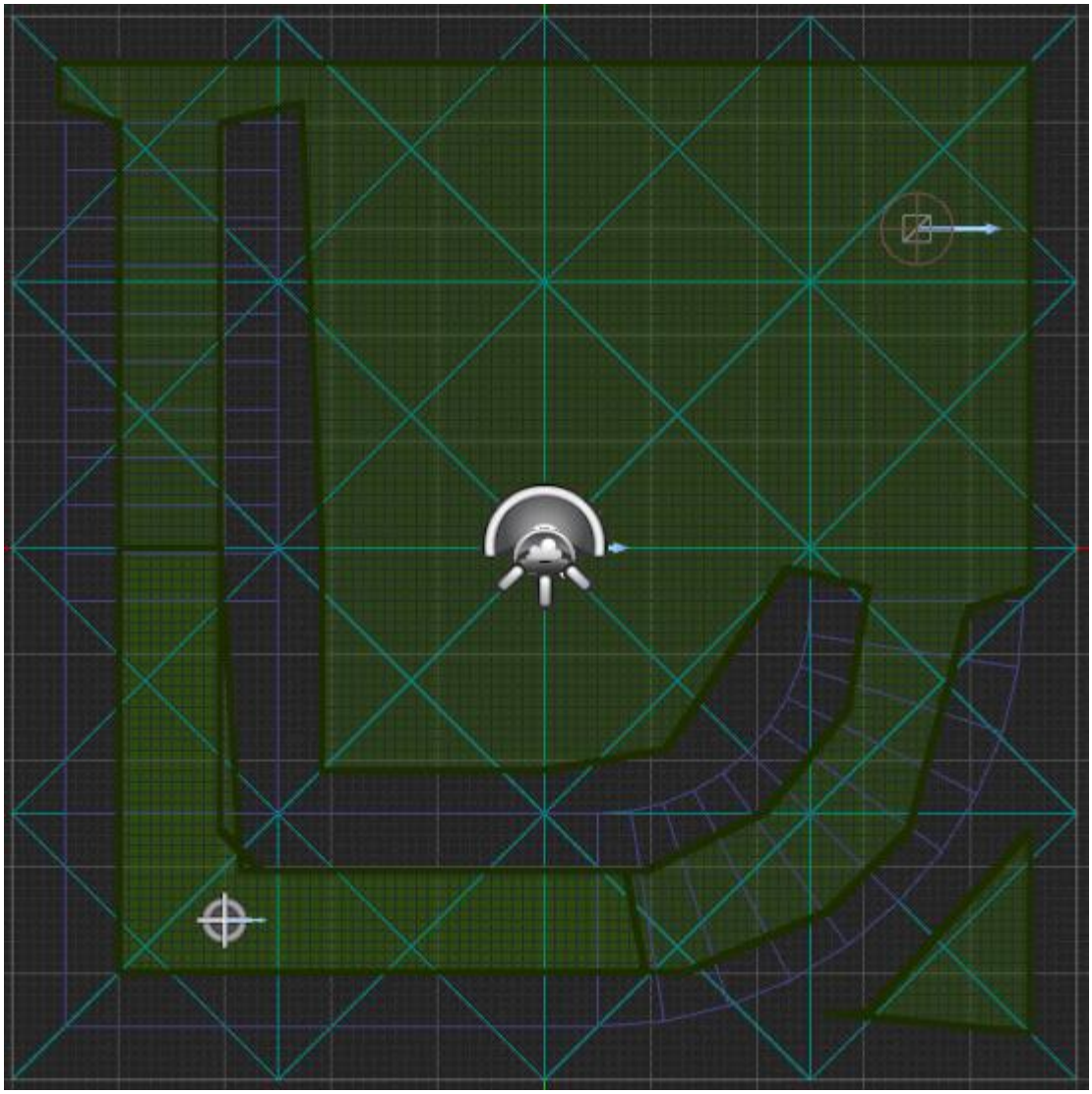


컴파일하고 저장하자.

11. 플레이해보자.
EnemyPawn가 **TPUpstairCorridorCorner** 위치로 길을 찾아서 이동하는 것을 확인할 수 있다.
한편, 플레이어의 시작 위치가 정해져 있어서 전체적인 이동 모습을 관찰할 수는 없을 것이다. 이런 경우에는 **시뮬레이트** 모드로 플레이하면 된다.
먼저, 이동하는 전체 모습을 볼 수 있도록 뷰포트에서 카메라를 높은 곳으로 이동하여 아래로 바라보도록 하자. 그다음, 툴바의 **플레이** 버튼의 오른쪽의 세로점 아이콘을 클릭하고 **시뮬레이트**를 선택하

자. 또는, 뷰포트에서 단축키 **Alt+S**를 입력하면 바로 **시뮬레이트**한다. **시뮬레이트**하면 **EnemyPawn**가 **TPUpstairCorridorCorner** 위치로 이동하는 경로를 관찰할 수 있다. 오른쪽의 굽은 계단을 통해서 이층 복도로 올라갈 것이다.

12. 원근 뷰가 아닌 직교 뷰에서 이동하는 모습을 관찰하는 것이 더 편리할 때가 있다.
플레이는 항상 원근 뷰로만 가능하지만 **시뮬레이트**는 직교 뷰에서도 가능하다.
이번에는 원근 뷰가 아닌 직교 뷰에서 시뮬레이트 모드로 플레이해보자.
먼저, 뷰포트의 뷰모드를 **원근**에서 **상단**으로 바꾸자.
그다음, 전체 맵이 보이도록 **휠버튼+스크롤**하여 줌 수준을 조절하자.
그다음, **시뮬레이트**해보자.
EnemyPawn가 목표 지점까지 가장 가까운 경로인 오른쪽의 굽은 계단을 통해서 이층 복도로 올라갈 것이다.



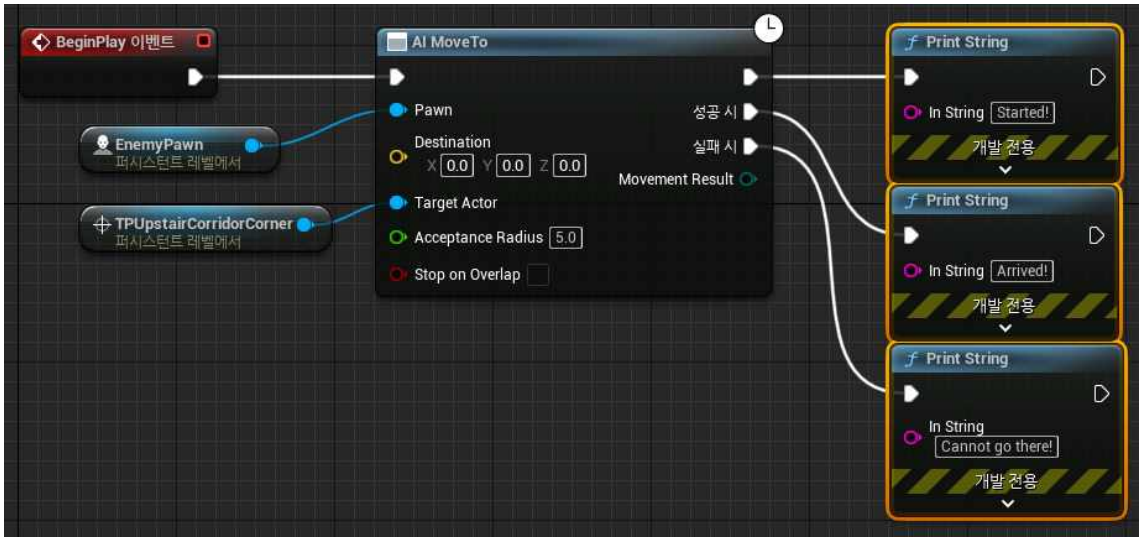
한편, 뷰모드가 **상단**이 아닌 다른 모든 직교 뷰에 대해서도 **시뮬레이트**가 가능하다. **정면**으로 바꾸고 **시뮬레이트**해서 확인해보자. 구조물의 모양에 따라 가장 적절한 직교 뷰 모드로 바꾸고 **시뮬레이트** 기능을 사용하면 된다. 이번 예제에서는 **상단**으로 두고 **시뮬레이트**하는 것이 가장 적절할 것이다.

13. 이제, **AIMoveTo** 노드의 출력핀을 살펴보자.

AIMoveTo 노드는 실행의 성공 여부나 실행 시간을 미리 확정할 수 없는 노드이다. **AIMoveTo** 노드가 실행되면 별도의 실행 쓰레드를 만들어 작업을 시작시킨 후에 바로 출력 실행핀으로 펄스가 흐르도록 한다. 실행을 시작시킨 후에 바로 다음 노드의 실행으로 넘어간다는 의미이다.

한편, **AIMoveTo** 노드의 작업은 목적지에 도달하는 것이 성공하거나 또는 실패하면 종료된다. 성공하면 **성공 시(OnSuccess)** 출력 실행핀으로 펄스가 흐르고 실패하면 **실패 시(OnFail)** 출력 실행핀으로 흐른다.

우리는 일반 실행핀, **OnSuccess** 실행핀, **OnFail** 실행핀 각각에 **PrintString** 노드를 연결하고 '`Started!`', '`Arrived!`', '`Cannot go there!`'를 출력하도록 해보자.



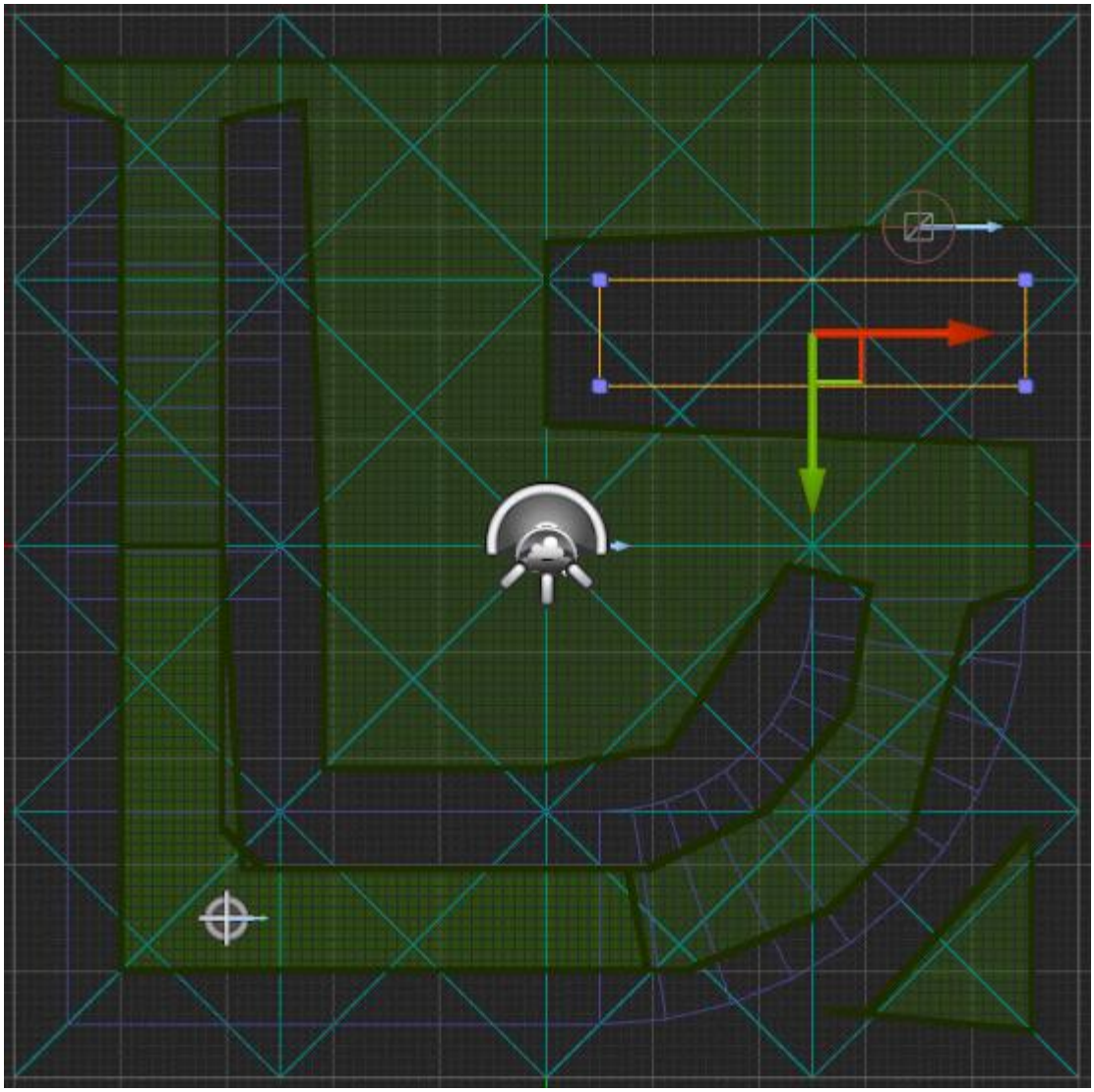
14. 이제, 장애물을 설치해서 더 복잡한 구조를 만들어보자.

액터 배치 탭의 **지오메트리** 탭에서 **박스**를 드래그해서 배치하자. 배치된 인스턴스 이름은 **PartitionWall Left**로 수정하자. **위치**는 (250,-200,50)으로 하고, **스케일**은 (2,0.5,0.5)로 하자.

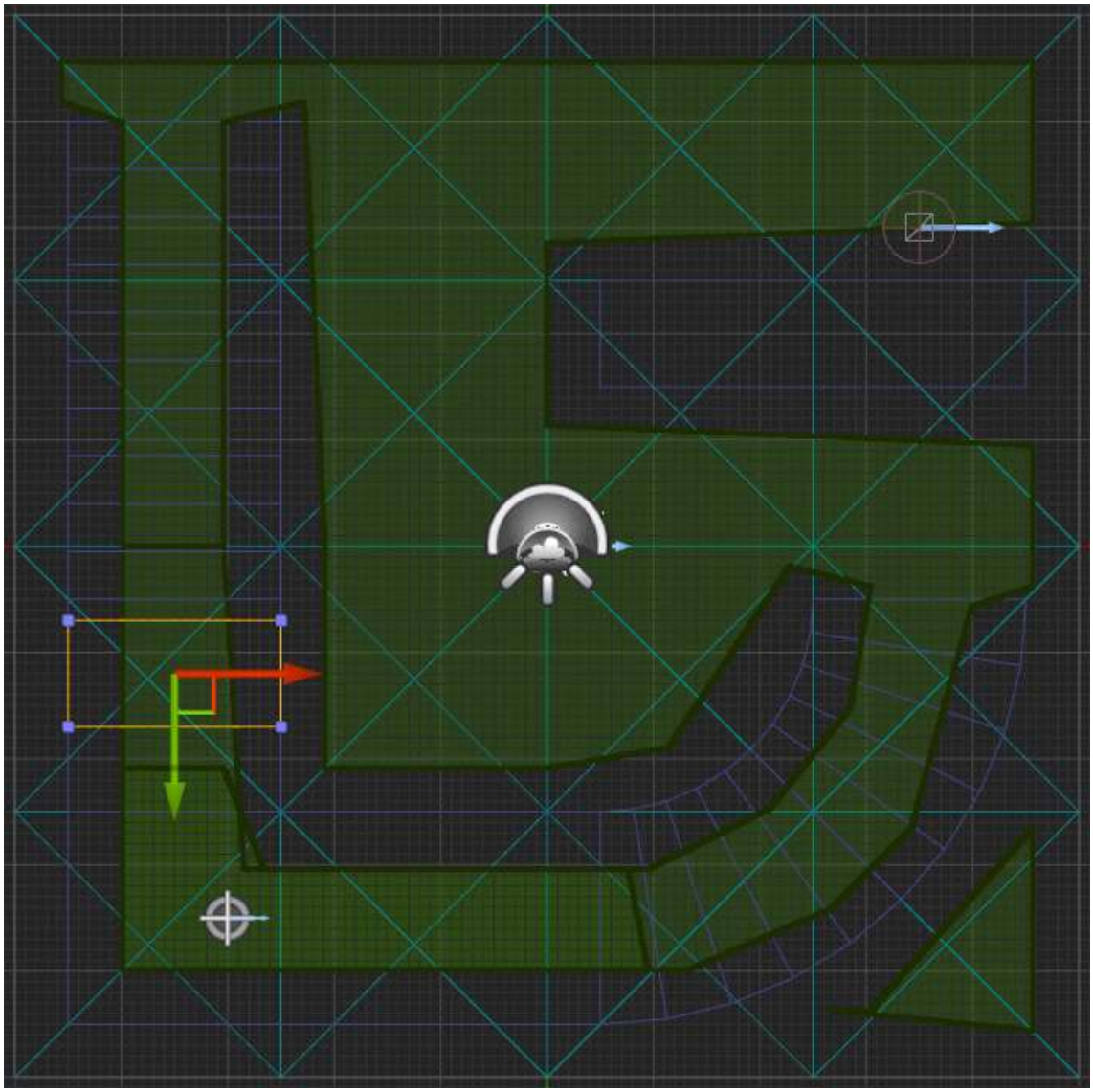
배치된 장애물 때문에 보행 가능한 녹색 영역이 바뀌었을 것이다.

뷰모드를 **상단**으로 두고 **시뮬레이트** 모드로 플레이해보자.

EnemyPawn가 장애물 때문에 더 빠른 다른 길을 찾아서 왼쪽의 직선 계단을 통해서 이층으로 올라갈 것이다.



15. 이제, 또다른 장애물을 설치해보자.
액터 배치 탭의 **지오메트리** 탭에서 **박스**를 드래그해서 배치하자. 이름은 **DroppedBoxUpstairRear**로 수정하자. **위치**는 (-350,120,270)으로 하고, **스케일**은 (1,0.5,0.5)로 하자.



뷰모드를 **상단**으로 두고 **시뮬레이트** 모드로 플레이해보자.
EnemyPawn가 왼쪽이 막혀서 갈 수 없으므로 이번에는 장애물을 돌아서 오른쪽의 굽은 계단을 통해서 이층으로 올라갈 것이다.

16. 내비게이션 기능은 정적으로 배치된 액터들뿐만 아니라 동적인 액터들도 고려하면서 동작한다.
이번에는 뷰모드를 **원근**으로 바꾸자.
그다음, 플레이어가 **EnemyPawn**의 움직임을 확인할 수 있도록 시작 지점에서의 바라보는 방향을 수정하자. **PlayerStart**를 선택하고 디테일 탭에서 회전을 (0,0,0)에서 (0,0,-90)으로 수정하자.
그다음, **시뮬레이트** 모드가 아닌 **선택된 뷰포트** 모드로 플레이해보자.
왼쪽이 막혀서 갈 수 없으므로 오른쪽으로 가기 위해 플레이어 쪽으로 다가올 것이다. 그러나 플레이어가 막고 있어서 플레이어 뒤로 지나갈 수가 없을 것이다. 몇 초 이후에 **EnemyPawn**는 이동을 포기하고 실패 문자열을 출력하고 정지할 것이다.

17. 다시 플레이해보자.
이번에는 플레이한 후에 플레이어를 옆으로 움직여서 **EnemyPawn**가 지나갈 수 있도록 비켜주자. **EnemyPawn**가 플레이어를 지나가서 목표 지점에 도달할 것이다.

이 절에서는 내비게이션과 내비 메시에 대해서 학습하였다.

4. 폰을 통한 컨트롤러로의 속성값 전달

이 절에서 폰을 통한 컨트롤러로의 속성값 전달 방법에 대해서 학습한다.

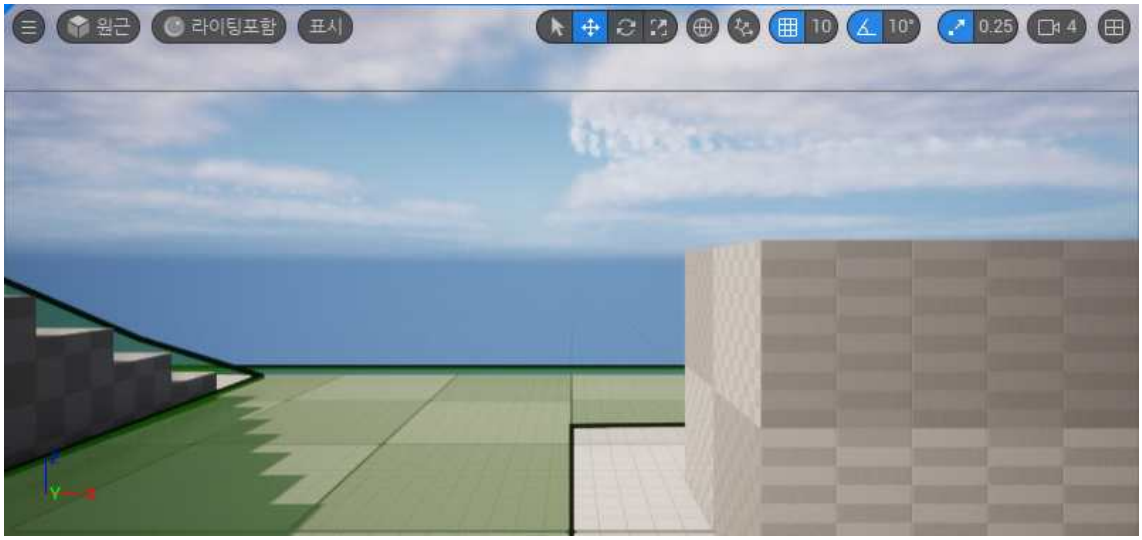
우리는 이전 예제에서 레벨 블루프린트에서 **AIMoveTo** 함수를 호출하였다. 그러나 적 캐릭터를 움직이는 일은 적 캐릭터를 제어하는 AI 컨트롤러에서 수행하는 것이 적절하다. 따라서 AI 컨트롤러에서 **AIMoveTo** 함수를 호출하도록 해보자. AI 컨트롤러에서 **AIMoveTo** 함수를 호출하기 위해서는 AI 컨트롤러가 적 캐릭터를 이동시킬 목적지를 알아야 한다.
목적지 위치 정보를 AI 컨트롤러에게 지정해주는 방법을 생각해보자. 레벨에 배치된 **타깃 포인트**가 목적지라고 해보자. 만약 AI 컨트롤러가 레벨에 배치되어 있다면 우리는 레벨 에디터에서 AI 컨트롤러의 디폴트 속성값 설정을 통해서 목적지 정보를 쉽게 설정할 수 있다. 그러나 AI 컨트롤러는 플레이된 후에 스폰된다. 따라서 레벨 에디터에서 속성값을 미리 설정해둘 수가 없다. 이런 경우에는 AI 컨트롤러가 스폰될 때에 실행되는 스크립트에서 속성값을 설정하도록 해야 한다. 그러나 이때에도 현재 레벨의 배치 인스턴스 정보를 스크립트에서 미리 알 수가 없으므로 속성값으로 설정하기가 어렵다.
우리의 해결책은 다음과 같다. 우리는 레벨에 배치된 다른 액터를 통해서 AI 컨트롤러에게 목적지 정보를 전달해주도록 하자. AI 컨트롤러가 제어하는 폰이 레벨에 배치되어 있다고 하자. 이 폰을 통해서 AI 컨트롤러에게 전달해주자.

이제부터 예제를 통해서 학습해보자

---

1. 새 프로젝트 `Ppawncontroller`를 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 `Ppawncontroller`로 입력하고 **생성** 버튼을 클릭하자. 프로젝트가 생성되고 언리얼 에디터 창이 뜰 것이다.

2. 이전 프로젝트 `Paimoveto`에서 **Content** 폴더 아래에 4개의 애셋 파일(`MyMap.umap`, `MyMap_BuiltData.uasset`, `EnemyPawn.uasset`, `EnemyAIController.uasset`)이 있을 것이다. 이들을 모두 복사하여 새 프로젝트의 **Content** 폴더 아래에 붙여넣자.
그리고, **프로젝트 세팅** 창에서 **맵&모드** 탭에서의 **에디터 시작 맵** 속성값을 `MyMap`으로 수정하자.
그다음, 콘텐츠 브라우저에서 **MyMap** 애셋을 더블클릭하여 레벨을 열어보자.
그다음, 뷰포트 내부를 한번 클릭한 후에 **P** 키를 입력하자. **내비 메시**가 보일 것이다.
이제, 이전 프로젝트와 동일한 상태가 되었다.

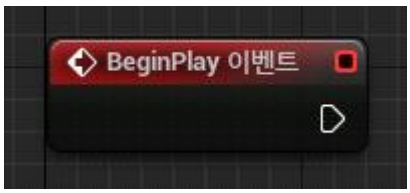


3. 예제를 진행하기 전에 이전 예제를 약간 수정하자.

먼저, 이전 예제에서는 이층에 장애물을 배치해두었는데 이것을 삭제하자. **아웃라이너**에서 **DroppedBoxUpstairRear**를 찾아서 삭제하자. 이제 이층 장애물이 제거되었다.

다음으로, 작성하였던 레벨 블루프린트 스크립트를 삭제하자. 이전 예제에서는 레벨 블루프린트에서 **BeginPlay** 이벤트 그래프에서 **AIMoveTo**를 호출하였다. 그러나 이번 예제에서는 AI 컨트롤러가 **AIMoveTo**를 호출하도록 할 것이다. 따라서, 이전 예제에서 작성해둔 레벨 블루프린트 스크립트를 삭제하자.

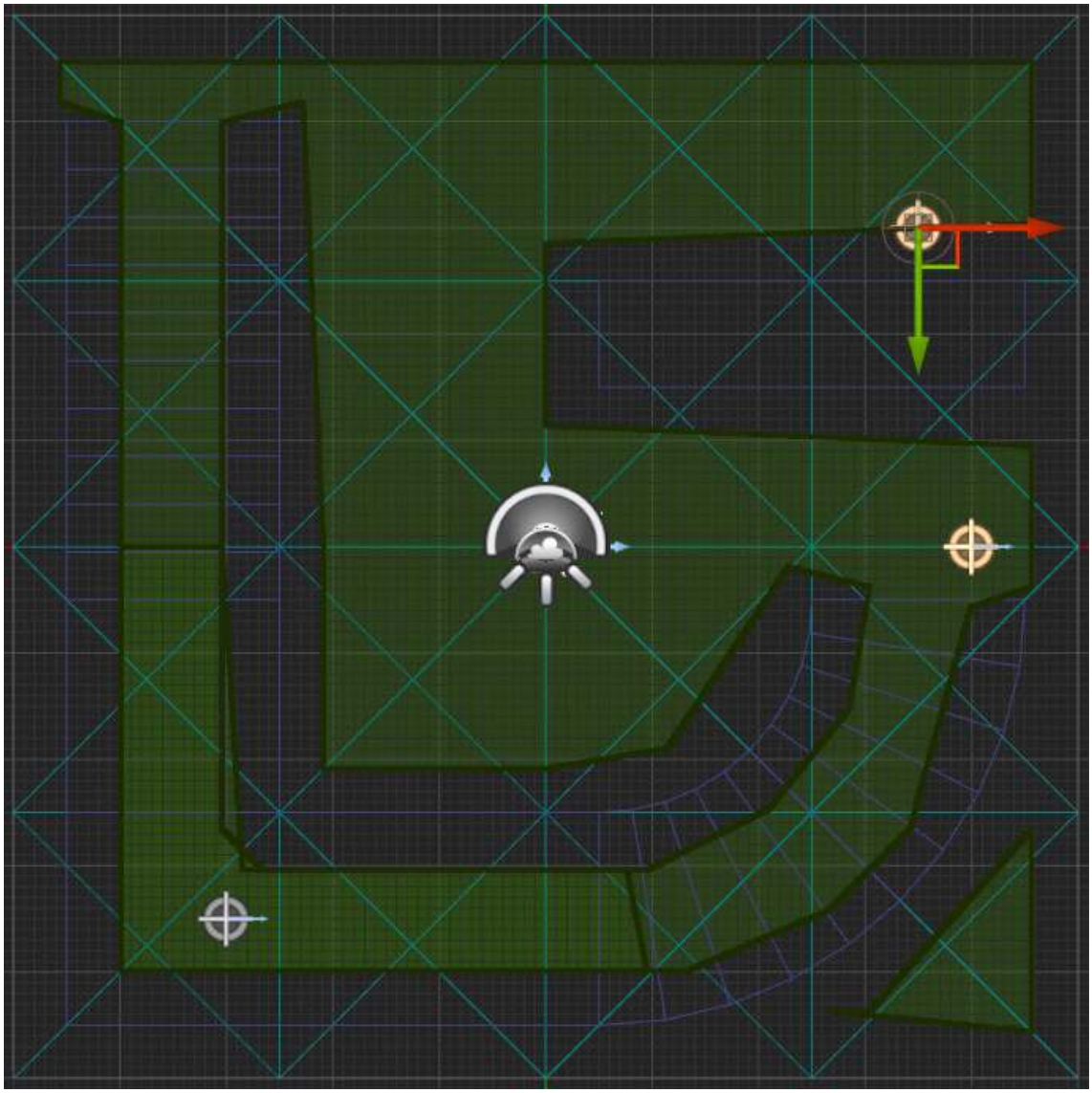
이를 위해서 먼저, 툴바의 **블루프린트** 버튼을 클릭하고 **레벨 블루프린트 열기**를 선택하여 **레벨 블루프린트 에디터**를 열자. 그다음, **BeginPlay 이벤트** 그래프에서 **BeginPlay** 이벤트 노드를 제외한 모든 노드를 삭제하자. 이제 레벨 블루프린트에서는 아무 일도 하지 않는다.



4. 이전 프로젝트에서 이층 복도 모서리에 **TPUpstairCorridorCorner**를 두어서 위치를 표시하였다. 우리는 추가적으로 두 개의 타깃 포인트를 더 배치하자.

첫 번째로, **EnemyPawn**의 시작 위치를 표시하기 위한 타깃 포인트를 배치하자. 액터 배치 탭에서 **TargetPoint**를 검색하여 **EnemyPawn**의 위치에 배치하자. 또는 기존의 타깃 포인트 액터를 복사하여 배치해도 된다. 배치된 인스턴스의 이름은 **TPEnemyPawnStartPoint**로 수정하자. 배치할 위치는 (350,-300,0)로 수정하자.

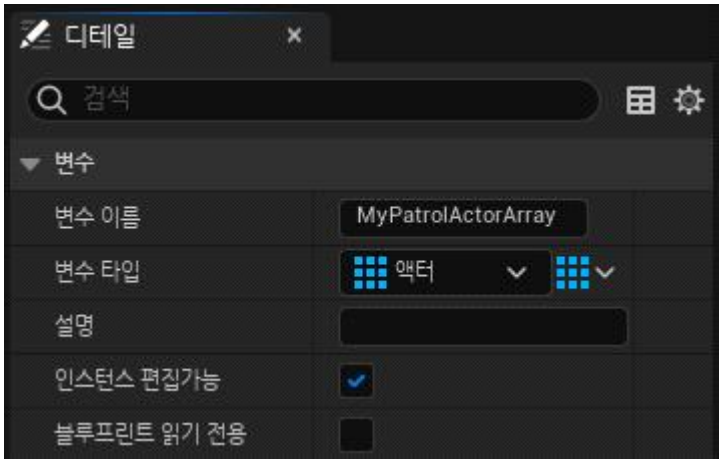
두 번째로, 오른쪽 굽은 계단 아래 지점을 표시하기 위한 타깃 포인트를 배치하자. 이름은 **TPCurvedStairRightBase**로 수정하자. 위치는 (400,0,0)으로 수정하자.



5. 콘텐츠 브라우저에서 **EnemyPawn**을 더블클릭하여 블루프린트 에디터를 열자. **내 블루프린트** 탭에서 변수를 추가하자

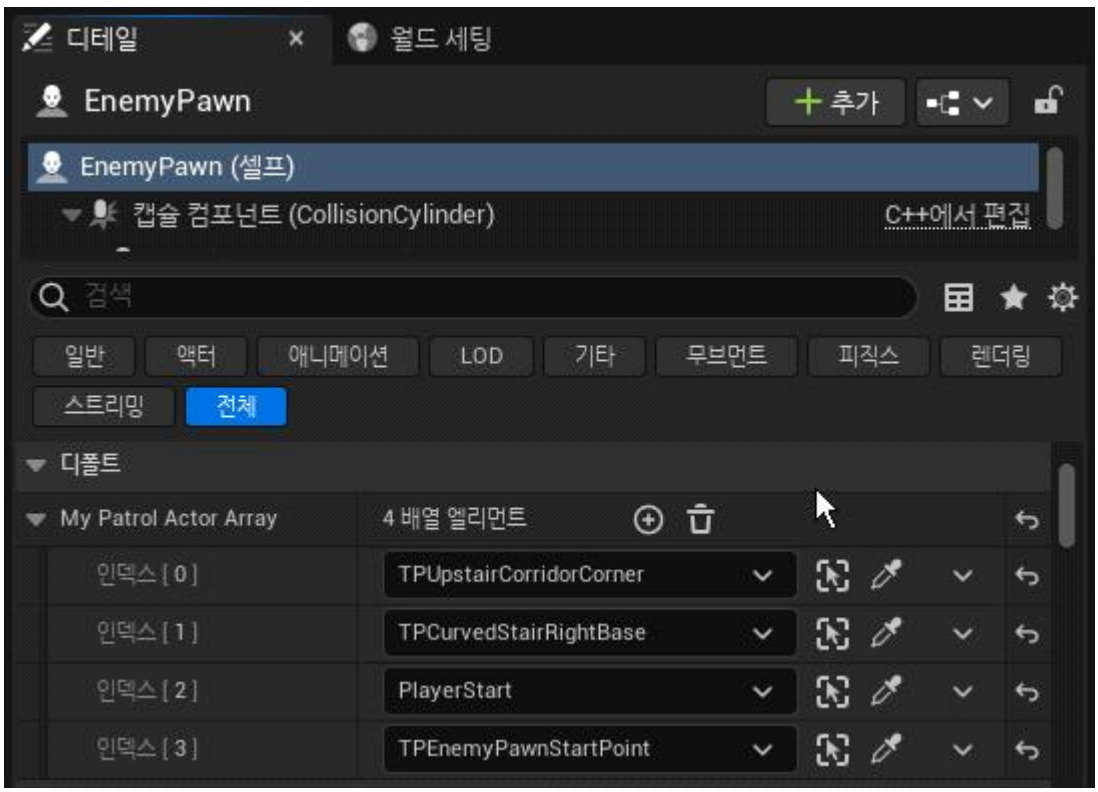
변수 이름을 **MyPatrolActorArray**라고 하자. 변수 유형은 **오브젝트 타입**의 **액터(Actor)** 레퍼런스로 하자. 그리고 오른쪽의 배열 아이콘을 클릭하여 배열 변수로 지정하자.

그다음, 그 아래의 **인스턴스 편집가능**에 체크되어 있도록 하자. 이제 레벨 에디터에서 속성값을 설정할 수 있다.



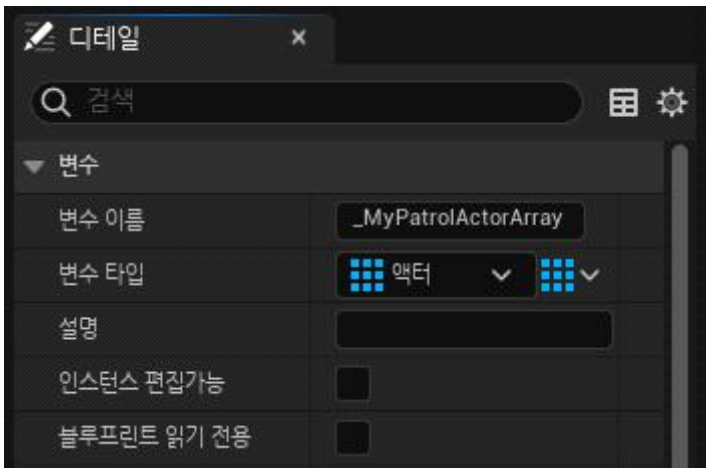
컴파일하고 저장하자.

6. 레벨 에디터의 **아웃라이너** 탭에서 **EnemyPawn**을 선택하고 **디테일** 탭으로 이동하자. **전체** 탭을 선택하고 디폴트 영역을 찾아보면 **MyPatrolActorArray** 속성이 있을 것이다.
MyPatrolActorArray 속성의 오른쪽의 + 아이콘을 네 번 클릭하여 4개의 배열 엘리먼트가 되도록 하자. 각 인덱스 0,1,2,3의 엘리먼트에 대하여 **TPUpstairCorridorCorner**, **TPCurvedStairRightBase**, **PlayerStart**, **TPEnemyPawnStartPoint**를 선택하여 지정하자.

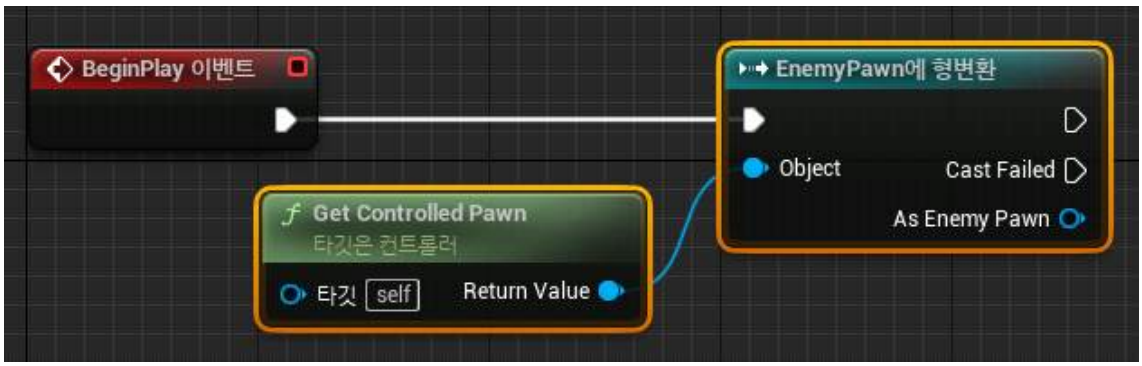


7. 이제, 콘텐츠 브라우저에서 **EnemyAIController**를 더블클릭하여 블루프린트 에디터를 열자.
내 블루프린트 탭에서 변수를 추가하자. 변수 이름을 **_MyPatrolActorArray**라고 하자. 변수 유형은 **오브젝트 타입**의 **액터(Actor)** 레퍼런스로 하자. 그리고 오른쪽의 배열 아이콘을 클릭하여 배열 변수로 지정하자.

여기서, 변수 이름을 **EnemyPawn**에서의 변수 이름과 다르게 지정해도 되지만 우리의 경우에는 항상 동일한 값을 복사해올 목적이므로 동일한 이름으로 지정하였고, 단지 구분을 위해서 접두사로 밑줄 문자를 붙였다.
컴파일하자.



8. **EnemyAIController**의 이벤트 그래프 탭으로 가자.
격자판의 빈 곳에 **GetControlledPawn** 노드를 배치하자. 이 노드는 플레이어 컨트롤러나 AI 컨트롤러가 빙의해서 제어하고 있는 폰을 리턴해준다. 이 노드는 출력핀으로 **Pawn** 타입의 레퍼런스를 리턴한다. 우리의 **EnemyPawn**에 정의되어 있는 변수인 **MyPatrolActorArray**를 접근하려면 **EnemyPawn** 타입으로 형변환을 해야 한다. 우리는 **GetControlledPawn** 노드의 출력핀을 드래그하고 **EnemyPawn에 형변환** 노드를 배치하자.
그다음, **BeginPlay 이벤트** 노드와 형변환 노드의 실행핀을 연결하자.

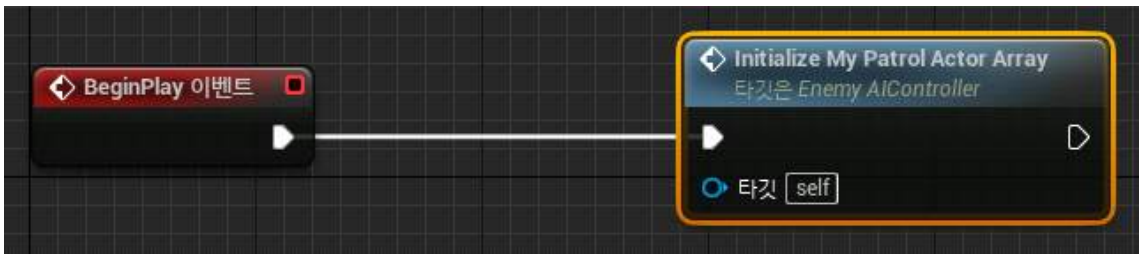


9. 이제, 형변환 노드의 **As EnemyPawn** 출력핀으로 **EnemyPawn**의 레퍼런스를 얻을 수 있다.
As EnemyPawn 출력핀을 드래그해서 **Get MyPatrolActorArray** 노드를 배치하자.
그다음, 내 블루프린트 탭에서 변수 영역의 **_MyPatrolActorArray** 변수를 드래그해서 **Set** 노드를 배치하자. 그리고, 형변환 노드의 실행핀과 **Set** 노드의 실행핀을 연결하자.
그다음, **Get MyPatrolActorArray** 노드의 출력핀을 **Set _MyPatrolActorArray** 노드의 입력핀에 연결하자.

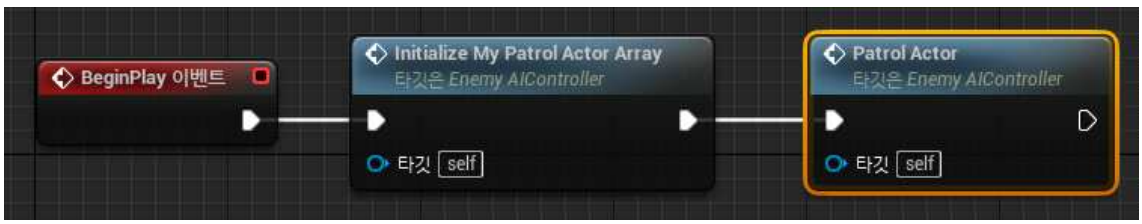
참고로, **Set** 노드에서 밑줄문자가 생략되어 보이지 않음에 유의하자.
이제 **EnemyPawn**의 **MyPatrolActorArray** 배열이 **EnemyAIController**의 **_MyPatrolActorArray** 배열로 그대로 복사된다.
컴파일하고 저장하자.

10. 노드 네트워크를 정리하자. **BeginPlay** 이벤트 노드 이외의 노드들을 모두 선택하고 우클릭하고 **함수로 접기**를 선택하자. 함수 이름은 **InitializeMyPatrolActorArray**로 지정하자.

11. 이제 **BeginPlay** 이벤트 그래프는 다음의 모습이 될 것이다.

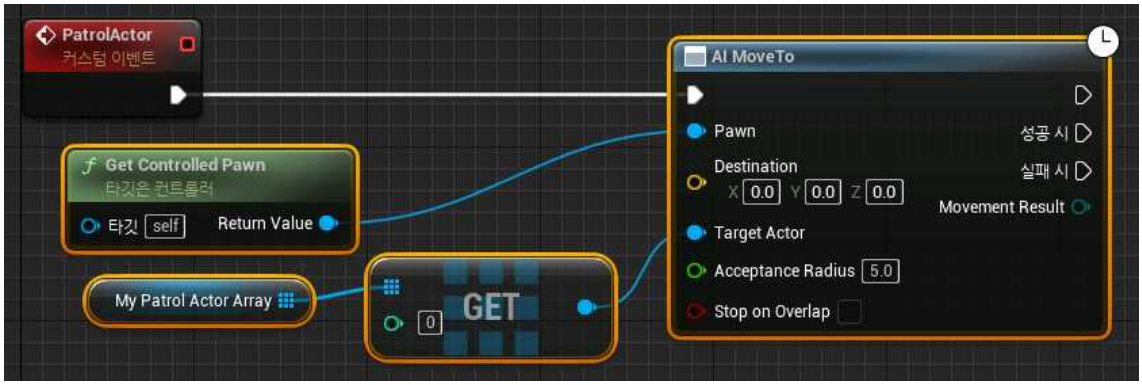


12. 이제, **EnemyAIController**의 이벤트 그래프 탭에서 **PatrolActor** 이벤트 노드를 추가하자. 격자판의 빈 곳에서 우클릭하고 **커스텀 이벤트 추가(AddCustomEvent)**를 검색하여 **커스텀 이벤트** 노드를 배치하자. 노드의 이름을 **PatrolActor**라고 수정하자.
그리고 **BeginPlay 이벤트** 노드 그래프의 뒤에 **PatrolActor** 함수 호출 노드를 액션선택 창에서 찾아 배치하자.

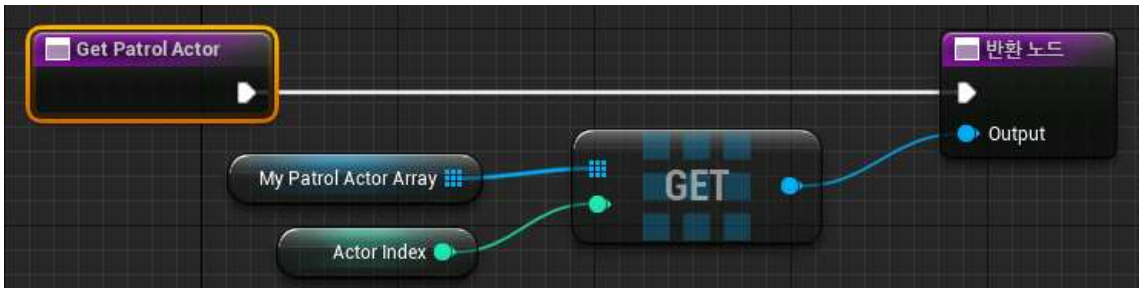


13. 이제, **PatrolActor** 이벤트 그래프를 만들어보자.
먼저, **PatrolActor** 커스텀 이벤트 노드의 실행핀을 당겨 **AIMoveTo** 노드를 배치하자.
그다음, **GetControlledPawn** 노드를 배치하고 출력핀을 **AIMoveTo** 노드의 **Pawn** 입력핀에 연결하자.
그다음, **_MyPatrolActorArray** 배열 변수의 **Get** 노드를 배치하자. 그리고 배치된 배열 **Get** 노드의 출력핀을 당겨 배열요소의 **Get** (사본) 노드를 배치하자. 이 배열요소의 **Get** 노드는 요소의 사본 값을 읽기

전용으로 리턴한다. 그리고 배치된 배열요소의 **Get** 노드의 출력핀을 **AIMoveTo**의 **TargetActor** 입력핀에 연결하자.

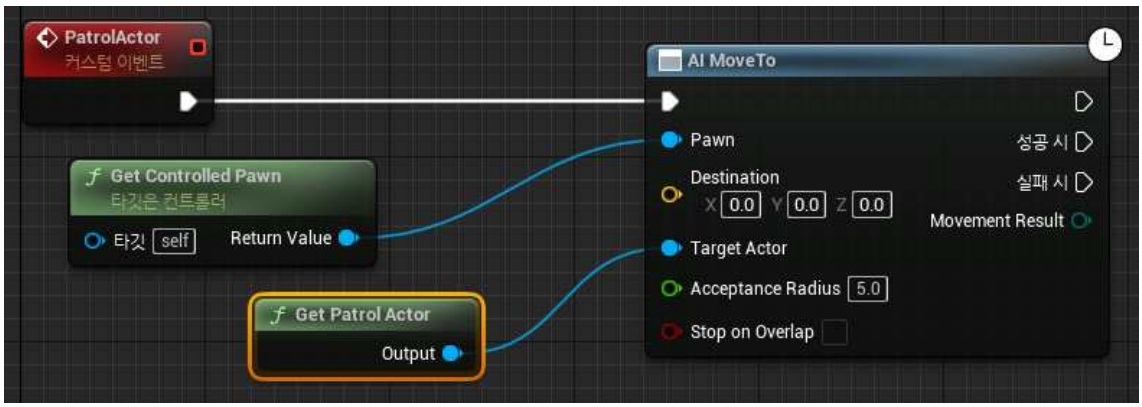


14. 이제, 배열의 엘리먼트를 가져오는 배열요소 **Get** 노드의 인덱스 입력핀을 왼쪽으로 드래그하고 **변수로 승격**을 선택하자. 새로 생성된 변수의 이름을 **ActorIndex**라고 지정하자.
그다음, 배열요소 **Get** 노드와 그 입력핀에 연결된 두 노드를 선택하고 우클릭하여 **함수로 접기**를 선택하자. 함수명을 **GetPatrolActor**로 지정하자.



그다음, **GetPatrolActor** 함수를 더블클릭하여 함수 그래프로 이동하자. 이미 함수 그래프에 있다면 함수 노드를 클릭하자. 그다음, 디테일 탭에서 **퓨어**에 체크하여 순수 함수로 만들자.

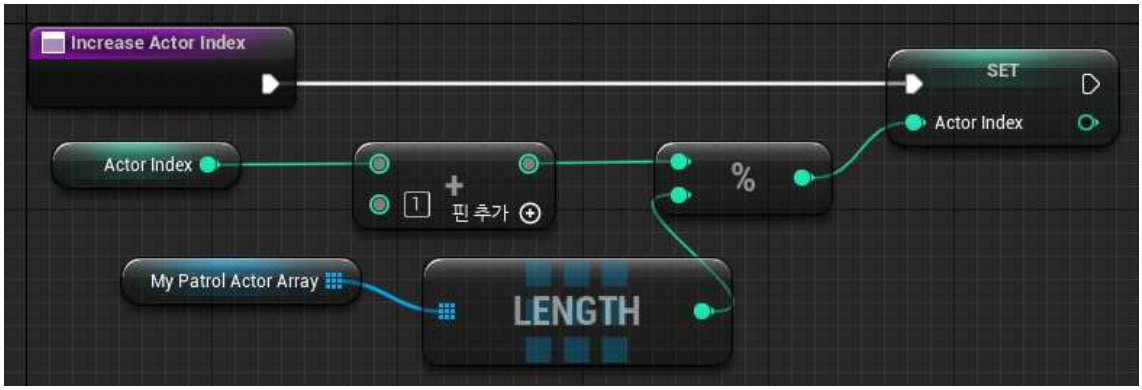
15. 이제, **PatrolActor** 이벤트 그래프는 다음의 모습이 될 것이다.



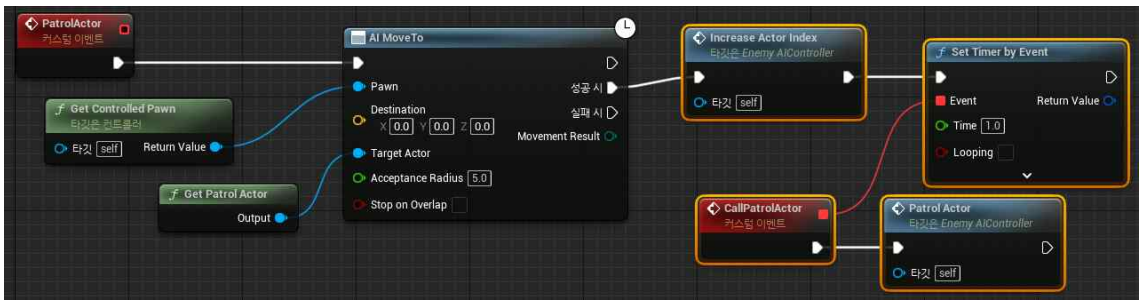
플레이해보자.
이전 예제에서와 같이 이층 모서리로 이동할 것이다. **_MyPatrolActorArray** 배열의 0번 엘리먼트가 해당 위치의 타깃 포인트이기 때문이다. 플레이어가 경로를 가로막게 되면 다른 경로를 선택하여 이동할 것이다.

16. 이제, **AIMoveTo** 함수가 성공적으로 수행되어 목적지에 도착한 경우에는 다음 패트롤 위치로

이동하도록 해보자.
이것을 구현하기 위해서 먼저 **ActorIndex** 변수를 증가시키는 **IncreaseActorIndex** 함수를 구현하자. **EnemyAIController** 블루프린트 에디터에서 계속 작업하자.
내 블루프린트 탭에서 함수를 추가하고 이름을 **IncreaseActorIndex**로 지정하자. 함수 그래프로 이동하자.
먼저, **ActorIndex**의 **Get** 노드와 **Set** 노드를 하나씩 배치하자. 그다음, **Get** 노드의 값을 **+** 노드로 1 증가시키자. 그다음, **%** 노드를 통해서 나머지 값을 구해서 그 값을 **Set** 노드에 연결하자.
배열의 개수만큼 나눈 나머지 값을 구하면 인덱스 값이 순환되도록 한다. 배열의 엘리먼트 개수는 **_MyPatrolActorArray**의 **Get** 노드를 배치하고 **Length** 함수를 호출하여 얻을 수 있다.
이제 **IncreaseActorIndex** 함수 그래프는 다음과 같은 모습으로 완성되었다.



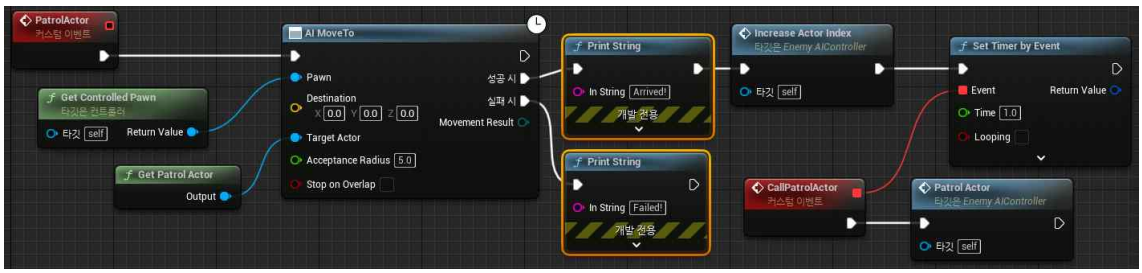
17. 이제, **PatrolActor** 이벤트 그래프로 이동하자.
먼저, 그래프의 가장 마지막에 있는 **AIMoveTo** 노드의 **성공 시(OnSuccess)** 출력 실행핀을 드래그하고 **IncreaseActorIndex** 함수 호출 노드를 배치하자.
그다음, **IncreaseActorIndex** 노드의 출력 실행핀을 드래그해서 **SetTimer byEvent** 노드를 배치하자.
그다음, **Time** 입력핀에는 1을 입력하여 1초 타이머가 작동한 후에 이벤트 그래프를 호출하도록 하자.
그다음, **Event** 델리게이트 입력핀을 왼쪽으로 드래그하고 **커스텀 이벤트 추가**를 선택하여 새 커스텀 이벤트 노드를 추가하자. 추가된 이벤트 노드의 이름을 **CallPatrolActor**로 하자.
그다음, **CallPatrolActor** 이벤트 노드의 출력 실행핀을 당기고 **PatrolActor** 함수 호출 노드를 배치하자.
참고로, **CallPatrolActor** 이벤트 노드를 만드는 대신에, **SetTimer byEvent** 노드의 **Event** 델리게이트 입력핀을 **PatrolActor** 커스텀 이벤트 노드의 델리게이트 출력핀에 바로 연결해도 된다. 그러나 노드 네트워크가 복잡하게 되는 것을 피하기 위해서 우리는 새 이벤트 노드를 추가하였다.



18. 플레이해보자. **EnemyPawn**가 각 위치를 순차적으로 패트롤할 것이다.

EnemyPawn가 PlayerStart 위치로 다가오면 플레이어를 다른 곳으로 비켜주면 계속 경로를 진행할 것이다.

19. 이제, 확인을 위해서 PatrolActor 이벤트 그래프를 약간 수정해보자. AIMoveTo 노드의 **성공 시(OnSucess)** 실행핀과 **실패 시(OnFail)** 실행핀에 각각 PrintString 노드를 추가하자.



20. 다시 플레이해보자.

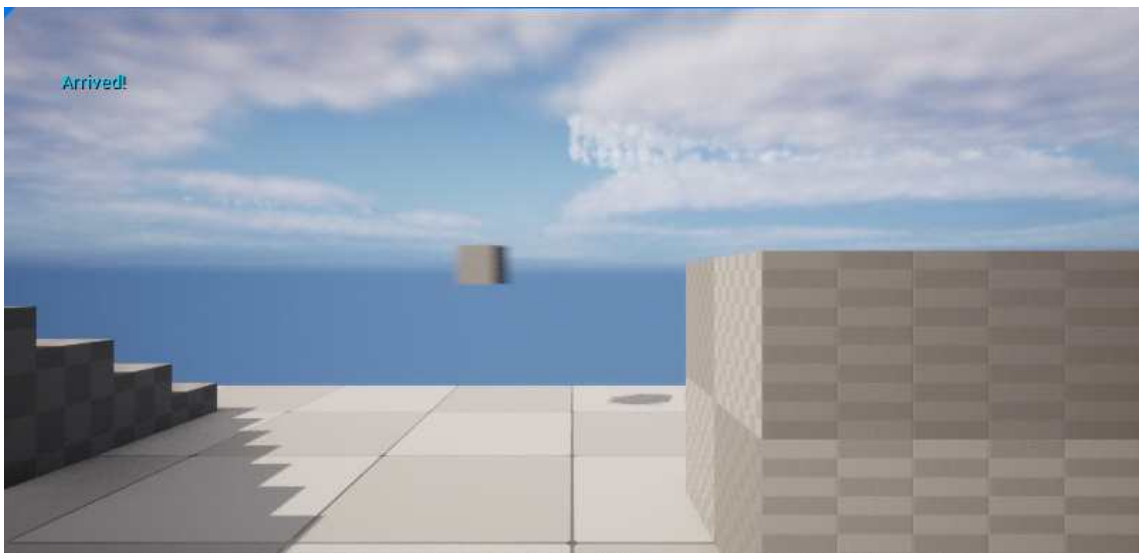
플레이어가 계속 움직이지 않고 있으면 AIMoveTo 노드가 실패하여 'Failed!'가 출력된다. PlayerStart 위치에 플레이어가 있기 때문에 플레이어에 막혀서 더이상 PlayerStart 위치로 도달하지 못하기 때문이다.

이런 경우에는 AIMoveTo 노드의 AcceptanceRadius 입력핀을 사용하면 된다. 이 입력핀으로 주어지는 거리값 이내일 경우에 목적지에 도달했다고 인정해주는 반경 거리값이다. 우리는 이 값을 5에서 100으로 수정하자.

이제 다시 플레이해보자.

이제는 'Arrived!'가 출력될 것이다.

플레이 시에 디폴트인 **선택된 뷰포트** 모드로도 실행해보고 또한 Alt+S 키를 클릭하여 **시뮬레이트** 모드로도 실행해보자.



<참고> 만약 'Arrived!'가 출력된 후에 다시 'Failed!'가 출력되는 경우가 생길 수도 있다. 이 경우는 PlayerStart 위치에 도달하는 것은 성공했으나 플레이어에 가로막혀서 그다음 위치로 가지 못하고 있기 때문에 실패하는 것이다. 이를 해결하기 위해서 PlayerStart 위치를 패트롤 동선 뒤로 약간 수정해보자. **아웃라이너**에서 PlayerStart를 클릭하고 **위치**를 디폴트인 (0,0,92)에서 (-100,100,92)로 수정하자. 이제 플레이해보자. 실패하지 않고 계속 패트롤할 것이다.

이 절에서는 폰을 통한 컨트롤러로의 속성값 전달 방법에 대해서 학습하였다.

□