21_ 비헤이비어 트리의 기초

<제목 차례>

인천대학교 컴퓨터공학부 박종승

1. 개요

이 장에서는 인공지능에서 가장 중요한 부분인 **비헤이비어 트리**에 대해서 학습한다.

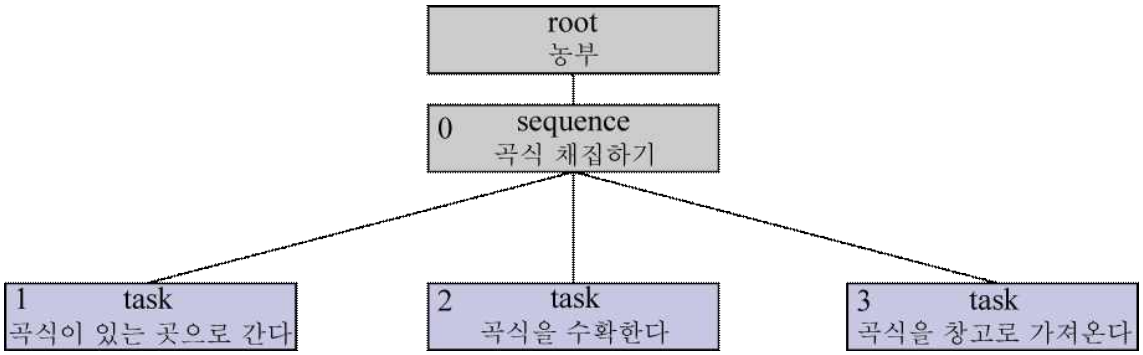
비헤이비어 트리란 게임 캐릭터의 행동 패턴을 구현하는 기법이다. 즉, **비헤이비어 트리**에서 NPC가 현재 상황에서 어떤 행동을 해야 하는지를 결정하고 행동을 수행한다. **비헤이비어 트리**는 NPC의 행동 표현을 위한 매우 효과적인 방법이며 널리 사용되고 있는 방법이다.
이제부터 **비헤이비어 트리**에 대해서 하나씩 학습해보자.
비헤이비어 트리는 행동 패턴을 트리 구조로 표현하는 방법이다. **비헤이비어 트리**의 각 노드를 순회하면서 해당 노드의 행위를 실행한다. 트리의 노드 순회를 무한히 반복해서 실행하는 방식으로 행동 패턴을 구현한다.
트리의 순회는 **깊이 우선 탐색**(Depth-First Search)의 방법으로 순회한다. 즉 루트 노드에서 출발하여 각 자식 노드를 왼쪽 자식 노드부터 오른쪽 자식 노드의 순서로 순차적으로 탐색한다. 자식 노드를 탐색할 때에 그 자식 노드가 또다시 자신의 자식 노드를 가지는 경우가 있을 것이다. 이런 경우에는 기존의 탐색을 일시 정지하고 자신의 자식 노드를 탐색하기 시작한다. 자신의 자식 노드의 탐색이 완료된 후에 기존의 탐색을 재개한다. 이런 순회 방법을 **깊이 우선 탐색**이라고 한다.
한편, 각 노드는 실행 시에 실행 결과에 따라서 성공과 실패의 리턴값이 있다. 노드의 실행 후에 리턴값을 자신의 부모 노드에 리턴해주어서 부모 노드가 실행 결과를 알 수 있도록 한다. 부모 노드는 자식 노드의 리턴값에 따라서 트리의 순회를 계속할 지의 여부를 수정할 수 있다.

비헤이비어 트리의 노드 유형에 대해서 알아보자.
먼저, **루트** 노드 유형에 대해서 알아보자. **루트** 노드는 트리의 가장 상단에 배치된 노드이다. 루트 노드는 항상 배치되어 있고 또한 트리에서 단 하나만 존재한다. 루트 노드는 실행의 출발점을 나타내는 노드이고 그 아래에 붙어있는 단 하나의 자식 노드를 계속해서 실행시켜주는 역할을 한다. 루트 노드 아래에는 단 하나의 **Composite** 노드만을 붙일 수 있다.
그다음, **루트** 노드를 제외한 **비헤이비어 트리**의 노드 유형에는 두 가지 유형이 있다. 하나는 **Composite** 노드이고 다른 하나는 **Task** 노드이다.
먼저, **Composite** 노드에 대해서 알아보자. **Composite** 노드는 실제로 작업을 수행하는 노드가 아니고 트리의 중간에 배치되어서 그 자식 노드를 어떻게 실행할 지를 결정해주는 특수 목적의 노드이다. **Composite** 노드 아래에는 다수개의 **Composite** 노드나 **Task** 노드를 붙일 수 있다.
그다음, **Task** 노드에 대해서 알아보자.
Task 노드는 실제 작업을 수행하는 노드이다. **Task** 노드는 반드시 트리의 종단에 배치되어야 한다. 즉 **Task** 노드는 자식 노드를 가질 수 없다.

Composite 노드의 유형에는 **Selector**, **Sequence**, **Simple Parallel**의 세 유형의 노드가 있다.
이제부터, 가장 중요한 두 유형인 **Selector** 노드와 **Sequence** 노드에 대해서 알아보자.
먼저, **Sequence** 노드에 대해서 알아보자. **Sequence** 노드는 자식 노드를 왼쪽에서 오른쪽의 순서대로 실행한다. **Sequence** 노드는 만약 자신의 자식 노드중의 하나가 중간에 실패하면 모든 것을 중단하고 자신의 부모 노드에게 실패를 리턴한다. 만약 모든 자식 노드가 모두다 성공하면 부모 노드에게 성공을 리턴한다.
다음으로, **Composite** 노드의 또다른 유형인 **Selector** 노드에 대해서 알아보자.

Selector 노드도 **Sequence** 노드와 동일하게 자신의 자식들을 왼쪽에서 오른쪽의 순서로 하나씩 실행한다. 그러나 **Selector** 노드는 자신의 자식 노드중 하나라도 성공를 리턴하면 다른 모든 자식 노드의 실행을 중지하고 부모에게 성공을 리턴한다.

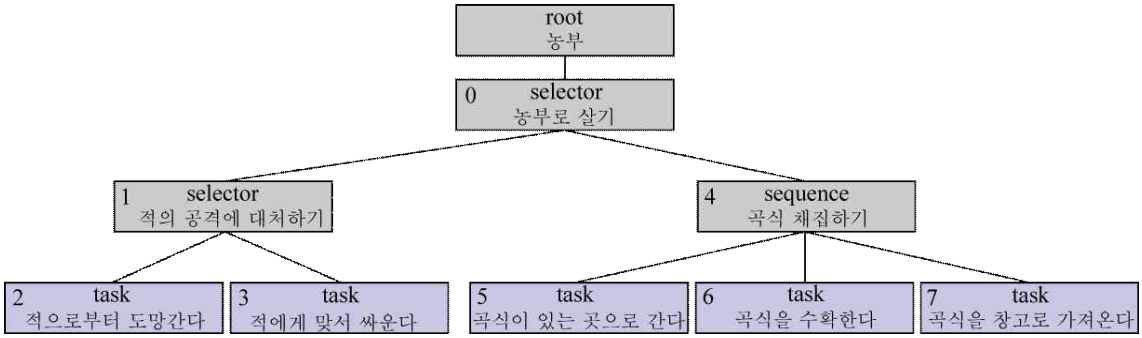
예를 들어보자.
먼저 **Sequence** 노드 사용 예를 들어보자.
게임에서 농부가 있다. 농부는 곡식이 있는 곳을 찾아가서, 그곳의 곡식을 수확하고, 수확한 곡식을 창고로 가져와서 저장해둔다. 이러한 과정을 무한히 반복하면서 진행한다. 이를 비헤이비어 트리로 표현하면 아래와 같다.



그림에서 루트 노드에는 0번 노드인 **Sequence** 노드가 붙어있다. 이 **Sequence** 노드는 1,2,3번의 세 **Task** 노드를 자식 노드로 가지고 있다.
먼저, 루트 노드는 0번 노드를 실행한다. 그다음, 0번 노드는 1,2,3번 노드를 순차적으로 실행하려고 시도한다.
먼저, 1번 노드가 실행된다. 1번 노드가 성공하면 0번 노드는 2번 노드를 이어서 실행한다. 한편, 1번 노드가 실패하면 이는 곡식이 없거나 곡식이 있는 곳으로 가는 경로가 차단되었다는 의미일 것이다. 이 경우 2,3번 노드의 실행은 무의미하므로 0번 노드는 실행을 중지하고 루트 노드에게 실패를 리턴한다.
그다음, 2번 노드가 실행될 때에 성공하면 0번 노드는 3번 노드를 이어서 실행한다. 한편, 2번 노드가 실패하면 0번 노드는 실행을 중지하고 루트 노드에게 실패를 리턴한다.
그다음, 3번 노드가 실행될 때에 성공이나 실패를 리턴하면 0번 노드는 그 결과를 루트 노드에 리턴한다.
지금까지, **Sequence** 노드의 예제를 알아보았다.
Sequence 노드는 자신의 모든 자식 노드가 성공일 때에만 성공을 리턴하고 그 외에는 실패를 리턴하게 된다. **Sequence** 노드는 일련의 과정을 순서대로 하나씩 모두 수행해서 목적을 완성하는 경우에 사용된다.
루트 노드는 자신의 하나뿐인 자식 노드의 성공이나 실패 여부와 무관하게 자식 노드의 실행이 종료되면 또다시 자식 노드를 반복해서 실행해준다.

다음으로, **Selector** 노드의 사용 예를 들어보자..
게임에서 적이 쳐들어와서 농부를 공격할 수도 있다고 해보자. 적이 쳐들어왔다면 농부는 곡식을 채집하기보다는 적의 공격에 대처하는 것이 우선이다. 공격에 대처하는 동안에는 곡식을 채집하지 않을 것이다. 또한 적의 공격에 대처하는 방법으로 농부는 공격력이 약하므로 우선적으로 도망가는

것을 선택할 것이다. 만약 막다른 길에 들어서서 도망가는 것이 불가능하게 되었다면 적에게 맞서 싸울 것이다.



위의 그림을 보자. 0번 노드에 **Selector** 노드가 배치되어 있다. 1번 노드와 4번 노드 중에서 하나만 선택하여 실행한다는 것이다. 1번 노드가 성공하면 4번 노드는 실행하지 않는다. 1번 노드가 실패할 때에는 적의 공격이 없다는 의미이므로 4번 노드를 실행한다.

1번 노드도 **Selector** 노드이다. 따라서 2번 노드와 3번 노드 중에서 하나만 선택하여 실행한다. 2번 노드가 성공하면 적으로부터 도망가는 것이 가능한 경우이므로 3번 노드는 실행하지 않는다. 2번 노드가 실패할 때에는 3번 노드를 실행한다.

지금까지, **Selector** 노드의 예제를 알아보았다.

Selector 노드는 여러 선택 옵션이 있을 때에 이 중에서 우선 순위가 높은 옵션을 하나만 선택하는 경우에 사용된다.

이제 노드 내에 붙일 수 있는 부가 기능에 대해서 알아보자.

Composite 노드나 **Task** 노드에는 노드 내에 **Decorator**나 **Service**를 붙일 수 있다. 이는 마치 액터 내에 여러 기능의 컴포넌트를 넣을 수 있는 것과 유사하다.

먼저, **Decorator**에 대해서 알아보자. **Decorator**는 노드에 부가적인 기능을 설정해준다. 예를 들어서 어떤 **Decorator**를 부착하여 노드의 실행 조건을 정해두고 그 조건이 만족될 경우에만 노드가 실행되도록 할 수 있다. 예를 들어서, 위의 예제에서 1번 노드에 '침입한 적이 존재하면'이라는 **Decorator**를 붙일 수 있다.

그다음, **Service**에 대해서 알아보자. **Service**는 노드에 센서의 탐지 기능을 설정해준다. 주변의 상황 변화에 대한 정보를 탐지해야 변화된 상황에 대처할 수 있다. 탐지를 위한 센서 역할을 수행하는 **Service**를 노드에 붙여두면 노드의 실행 시에 **Service**가 상황 정보를 수집하고 수집된 정보를 기록해두는 기능을 수행한다. 예를 들어서, 위의 예제에서 4번 노드에 침입한 적이 있는지를 체크하는 센서 기능의 **Service**를 붙여둘 수 있다. 침입한 적인 발견되면 메모판에 이를 기록해둔다. 그다음에 1번 노드의 실행 순서가 돌아오면 1번 노드는 메모판을 보고 침입한 적이 존재하다는 것을 알게 될 것이다. 따라서 1번 노드가 실패하지 않고 실행될 것이다.

이제부터, **블랙보드**에 대해서 알아보자.

위에서 **Service** 등이 메모판에 정보를 기록하고 다른 노드에서 그 정보를 읽어 사용한다고 설명하였다. 여기서 메모판의 역할을 하는 객체를 **블랙보드**라고하자. 블랙보드를 사용하여 노드에서 다른 노드로의 정보 전달을 편리하게 할 수 있다.

블랙보드는 **비헤이비어 트리**에서 사용하는 변수들의 컨테이너이다. **비헤이비어 트리**를 구현하기 위해서는 대부분의 경우 정보의 전달 목적으로 사용되는 **블랙보드**가 필요하다. 따라서 **비헤이비어 트리**를

다루기 전에 먼저 **블랙보드**를 생성해두자.

<참고> 비헤이비어 트리에 대한 자세한 내용은 다음의 링크를 참조하자.
https://docs.unrealengine.com/behavior-tree-in-unreal-engine---overview/

<참고> 비헤이비어 트리에 대해서 다음의 'Behavior trees for AI: How they work' 문서를 참조하자.
https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work

2. 프로젝트 준비하기

이전 장에서 진행한 예제를 다시 한번 복습해보자.
이번 절에서의 실습은 이전 장의 내용을 그대로 반복해서 진행하는 것이어서 새로운 내용이 없다.
이번 절의 실습을 생략해도 무방하지만 숙련을 위해서 따라해볼 것을 권장한다.
이제부터 예제를 통해서 학습해보자

---

1. 새 프로젝트 **Pbtprepare**을 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 Pbtprepare로 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.
그리고, **아웃라이너**에서 **Floor**를 선택하고, **디테일** 탭에서 **스케일**을 (8,8,8)에서 (1,1,1)로 수정하자.

2. 적 캐릭터 폰과 AI 컨트롤러를 만들고 서로 연결되도록 하자.
먼저, 적 캐릭터 폰을 만들자.
먼저, **콘텐츠 브라우저**에서 **+추가**를 클릭하고 **블루프린트 클래스**를 선택하자. **부모 클래스 선택** 창에서 부모 클래스로 **캐릭터(Character)**를 선택하자. 생성된 블루프린트 클래스 이름을 **EnemyPawn**으로 하자.
그다음, **EnemyPawn**의 블루프린트 에디터를 열고, 왼쪽의 **컴포넌트** 탭에서 **메시(Mesh)** 컴포넌트를 선택하고 오른쪽의 **디테일** 탭에서 **메시** 영역의 **스켈레탈 메시(Skeletal Mesh)** 속성값에 엔진에서 제공하는 스켈레탈 메시인 **SkeletalCube**를 선택하자. 컴파일하고 저장하자.
그다음, 적 캐릭터를 제어하는 AI 컨트롤러를 생성하자.
먼저, **콘텐츠 브라우저**에서 **+추가**를 클릭하고 **블루프린트 클래스**를 선택하자. **부모 클래스 선택** 창에서 **모든 클래스** 영역에서 부모 클래스로 **AIController**를 검색하여 선택하자. 생성된 블루프린트 클래스 이름을 **EnemyAIController**으로 하자. 컴파일하고 저장하자.
그다음, 레벨에서 **EnemyPawn**이 처음으로 생성될 때에 **EnemyAIController**에 의해서 빙의되도록 하자. **EnemyPawn**의 블루프린티 에디터에서 툴바의 **클래스 디폴트** 버튼을 클릭하고, **디테일** 탭에서 **폰** 영역에 있는 **AI 컨트롤러 클래스(AI Controller Class)** 속성값을 **AIController**에서 **EnemyAIController**로 수정하자. 컴파일하고 저장하자.
그다음, 콘텐츠 브라우저에서 **EnemyPawn**을 드래그하여 레벨에 배치하자. 이름은 **EnemyPawn**으로 그대로 두자. 위치는 (350,-300,88)에 배치하자.

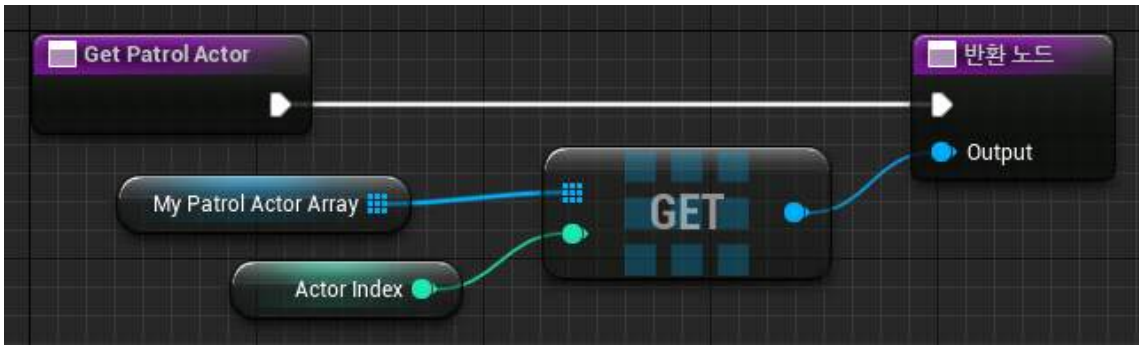
3. **내비 메시**가 생성되도록 하고 레벨을 꾸며보자.
먼저, **내비 메시**가 생성되도록 해보자.
액터 배치 탭에서 **볼륨** 탭을 클릭하고 **NavMeshBoundsVolume(내비 메시 바운드 볼륨)**을 드래그해서 레벨에 배치하자. 디테일 탭에서 **위치**는 (0,0,100)으로 하고, 스케일은 (5,5,1.5)로 입력하자. 뷰포트 내에서 키보드 **P** 키를 눌러 **내비 메시**가 표시되도록 하자.
이제, 레벨을 꾸며보자. **액터 배치** 탭에서 **지오메트리** 탭을 클릭하자.

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.
이제, 장애물을 설치해보자. **박스**를 드래그해서 배치하자. 배치된 인스턴스 이름은 **PartitionWallLeft**로 수정하자. **위치**는 (250,-200,50)으로 하고, **스케일**은 (2,0.5,0.5)로 하자.
그리고, **아웃라이너**에서 **PlayerStart**를 클릭하고 **위치**를 (-100,100,92)로 수정하자.

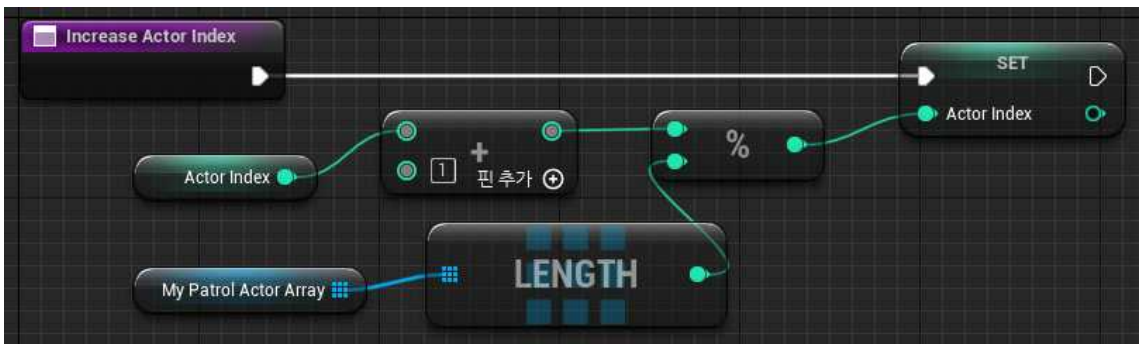
4. 레벨에 지점 표시를 하고 이를 **EnemyPawn**의 배열 변수의 디폴트 값으로 지정하자.
먼저, **타깃 포인트** 네 개를 배치하자.
첫 번째로, 이층의 모서리에 지점 표시를 해두자. **액터 배치** 탭에서 **TargetPoint**를 검색하여 **타깃 포인트** 액터를 레벨로 드래그하여 배치하자. 배치된 인스턴스의 이름을 **TPUpstairCorridorCorner**로 수정하자. **위치**는 (-350,350,200)으로 하자. 이층 복도의 꺾이는 모서리 위치에 배치될 것이다.
두 번째로, 오른쪽 굽은 계단 아래에 배치하자. 이름은 **TPCurvedStairRightBase**로 수정하자. 위치는 (400,0,0)으로 수정하자.
세 번째로, 바닥의 중심 위치에 배치하자. 이름은 **TPFloorCenter**로 수정하자. 위치는 (0,0,20)으로 수정하자.
네 번째로, **EnemyPawn**의 위치에 배치하자. 배치된 인스턴스의 이름은 **TPEnemyPawnStartPoint**로 수정하자. 위치는 (350,-300,0)로 수정하자.
그다음, **EnemyPawn**의 배열 변수의 디폴트 값으로 지정하자.
먼저, **EnemyPawn**의 블루프린트 에디터의 **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. 그리고, 그 아래의 **인스턴스 편집가능**에 체크하자. 컴파일하고 저장하자.
그다음, 레벨 에디터에서 **EnemyPawn**을 선택하고 **디테일** 탭으로 이동하자. **전체** 탭을 클릭하고 **디폴트** 영역에 있는 **MyPatrolActorArray** 속성을 찾자. 속성에서 4개의 배열 엘리먼트를 추가하자. 각 인덱스 0,1,2,3의 엘리먼트에 대하여 **TPUpstairCorridorCorner**, **TPCurvedStairRightBase**, **TPFloorCenter**, **TPEnemyPawnStartPoint**를 선택하여 지정하자.

5. **EnemyPawn**의 배열 변수를 **EnemyAIController**의 배열 변수에 복사하자.
먼저, **EnemyAIController**의 블루프린트 에디터를 열고 **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. 컴파일하자.
그다음, 배열 변수를 복사하는 함수를 작성하자. **EnemyAIController**의 **내 블루프린트** 탭에서 함수를 추가하자. 함수 이름은 **InitializeMyPatrolActorArray**로 지정하자. 함수 그래프를 다음과 같이 작성하자.
여기서, **Get** 노드는 **EnemyPawn**의 배열 변수 **MyPatrolActorArray**의 **Get** 노드이고, **Set** 노드는 **EnemyAIController**의 배열 변수 **_MyPatrolActorArray**의 **Set** 노드이다.

6. **EnemyAIController**의 **내 블루프린트** 탭에서 **인티저** 유형의 변수 **ActorIndex**를 추가하자.
그다음, **EnemyAIController**에 함수 **GetPatrolActor**를 추가하자. 함수 그래프에서 함수 노드를 클릭하고 **디테일** 탭에서 **출력**을 추가하자. 이름을 **Output**으로 하고 유형을 **오브젝트** 타입의 **Actor** 레퍼런스로 하자. 그리고, 디테일 탭에서 **퓨어**에 체크하여 순수 함수로 만들자.
함수 그래프를 작성하자. 먼저, **_MyPatrolActorArray**의 **Get** 노드를 배치하고, 그 노드의 출력을 당기고 배열요소 **Get** 노드도 배치하자. **ActorIndex**의 **Get** 노드를 배치하고 배열요소 **Get** 노드의 입력에 연결하자. 그리고, 배열요소 **Get** 노드의 출력을 **반환 노드**에 연결하자.
컴파일하자.



7. **EnemyAIController**에 새 함수 **IncreaseActorIndex**를 추가하자.
함수 그래프를 작성하자. **ActorIndex**의 **Get** 노드를 배치하고, 출력핀을 당겨 **+** 노드를 배치하자. 그리고, **+** 노드의 두 번째 입력핀에 1을 입력하자. 그다음, **+** 노드의 출력핀을 당겨 **%** 노드를 배치하자. 그리고, **%** 노드의 두 번째 입력핀에는 **_MyPatrolActorArray**의 **Get** 노드를 배치하고 그 노드의 **Length** 함수값을 입력하자. 그다음, **ActorIndex**의 **Set** 노드를 배치하고 **%** 노드의 출력핀을 **Set** 노드의 입력에 연결하자. 그다음, 함수 노드의 실행핀을 **Set** 노드의 입력 실행핀에 연결하자.
컴파일하자.



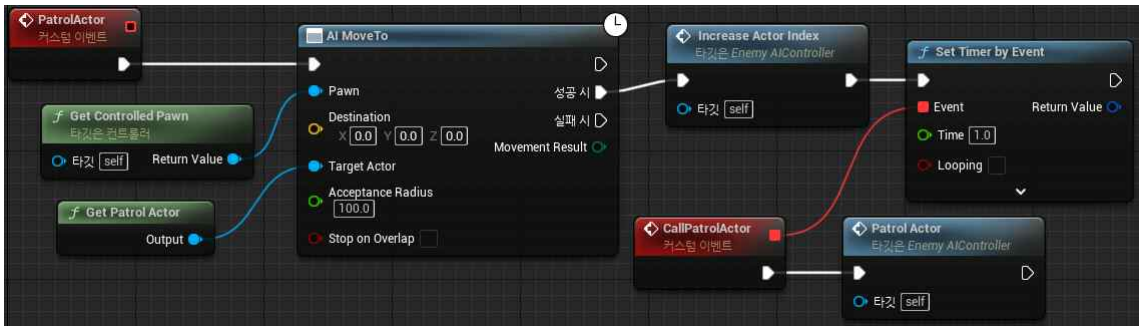
8. **EnemyAIController**의 이벤트 그래프 탭에서 **커스텀 이벤트** 노드를 추가하고 이름을 **PatrolActor**로 하자.

PatrolActor 이벤트 그래프를 작성하자.

먼저, **PatrolActor** 이벤트 노드를 당기고 **AIMoveTo** 노드를 배치하자. 그다음, **GetControlledPawn** 노드를 배치하고 **AIMoveTo** 노드의 **Pawn** 입력에 연결하자. 그다음, **GetPatrolActor** 노드를 배치하고 **AIMoveTo** 노드의 **TargetActor** 입력에 연결하자. 그리고, **AIMoveTo** 노드의 **AcceptanceRadius** 입력핀의 값을 5에서 100으로 수정하자.

그다음, **AIMoveTo** 노드의 **성공 시(OnSuccess)** 실행핀을 당기고 **IncreaseActorIndex** 노드를 배치하자. 그다음, **IncreaseActorIndex** 노드의 출력핀을 당기고 **SetTimer byEvent** 노드를 배치하자. 그리고, **Time** 입력핀에 1을 입력하자.

그다음, **SetTimer byEvent** 노드의 **Event** 델리게이트 입력핀을 왼쪽으로 당기고 **커스텀 이벤트 추가**를 선택하여 커스텀 이벤트 노드를 배치하고 이름을 **CallPatrolActor**라고 하자. 그다음, **CallPatrolActor**라고 이벤트 노드를 당기고 **PatrolActor** 함수 호출 노드를 배치하자.



9. **EnemyAIController**의 **BeginPlay** 이벤트 그래프를 작성하자.

BeginPlay 이벤트 노드를 당기고 **InitializeMyPatrolActorArray** 함수 호출 노드를 배치하자. 그다음, **InitializeMyPatrolActorArray** 노드를 당기고 **PatrolActor** 함수 호출 노드를 배치하자.

플레이해보자. 이전 예제에서와 동일하게 패트롤을 반복할 것이다.

상단 뷰로 바꾸고 **Alt+S**를 눌러 **시뮬레이션**으로 관찰하면 더 잘 확인할 수 있다.

이 절에서는 이전 예제에서의 과정을 다시 진행해보았다.

3. 블랙보드와 비헤이비어 트리 만들기

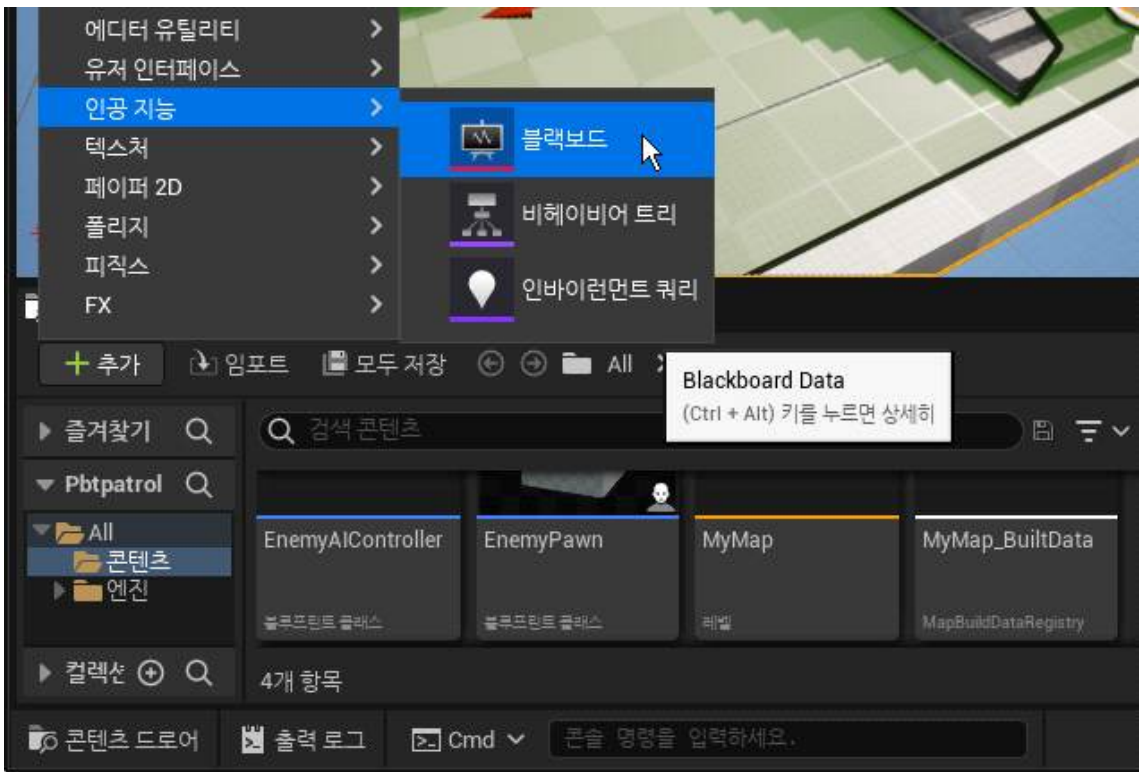
먼저, **블랙보드**를 생성하고 그다음, 비헤이비어 트리를 만들자.
비헤이비어 트리는 우리가 앞으로 인공지능 알고리즘을 구현할 도구이다. 비헤이비어 트리를 생성한 후에는 여기에서 데이터 공유 목적으로 사용할 블랙보드를 지정하자.
이제부터 예제를 통해서 학습해보자

---

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

2. 이전 프로젝트 Pbtprepare에서 **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 MyMap으로 수정하자.
그다음, 콘텐츠 브라우저에서 **MyMap** 애셋을 더블클릭하여 레벨을 열어보자.
그다음, 뷰포트의 빈 곳이나 아무 액터나 선택한 후에 **P** 키를 입력하자. **내비 메시**가 보일 것이다.
이제, 이전 프로젝트와 동일한 상태가 되었다.

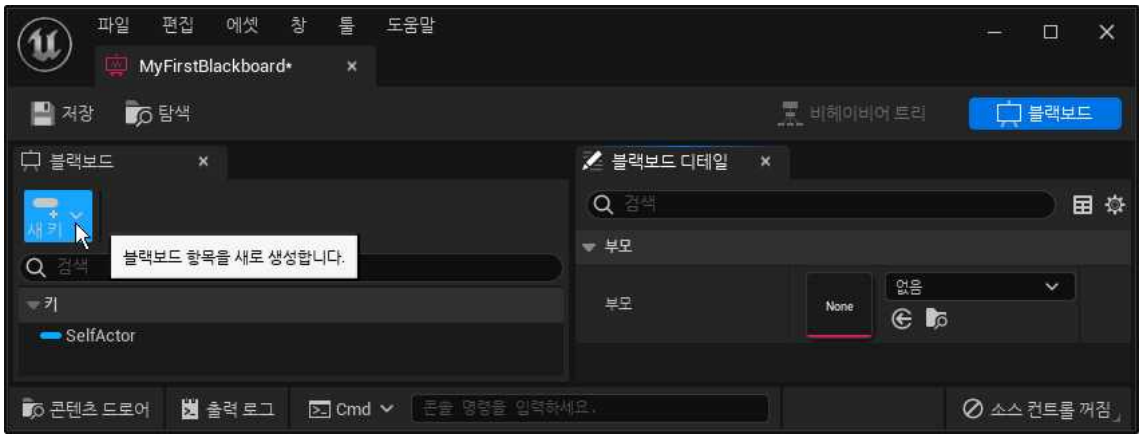
3. 블랙보드를 만들자.
콘텐츠 브라우저에서 **콘텐츠** 폴더를 선택하고, 툴바의 **+추가**를 클릭하자. 드롭다운 메뉴에서 **인공지능** » **블랙보드**을 선택하자. 이름을 **MyFirstBlackboard**라고 지정하자.



이제, 블랙보드를 생성하였다. 이 블랙보드는 나중에 비헤이비어 트리에서 정보를 기록하고 읽어가

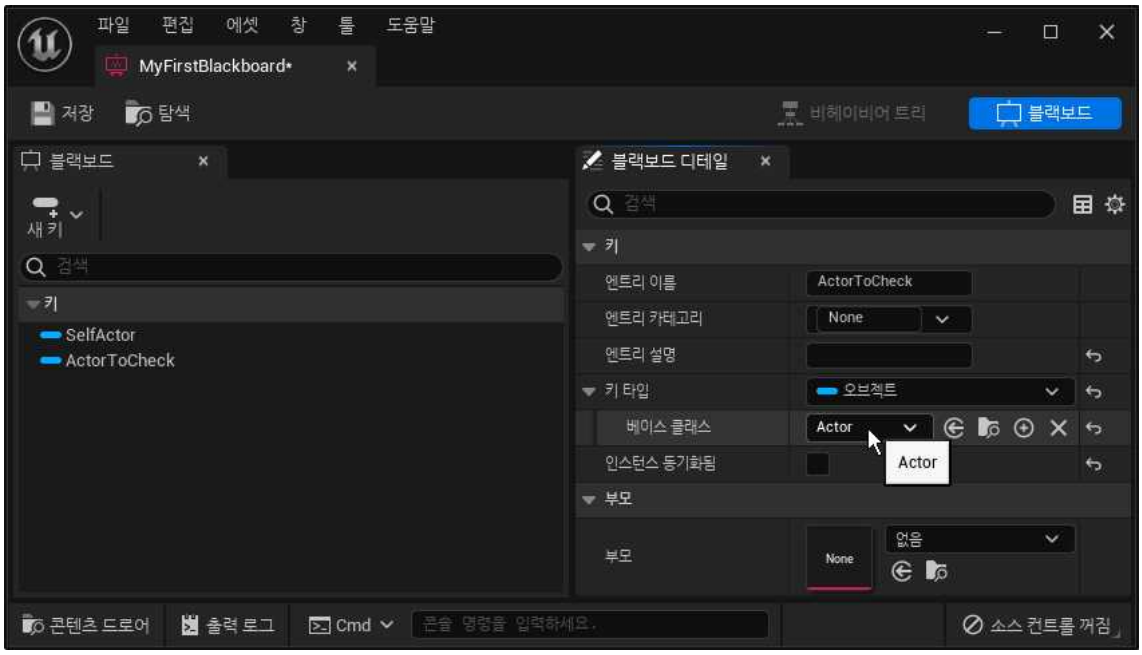
기 위한 용도로 사용될 것이다.

4. MyFirstBlackboard를 더블클릭하자. 에디터가 뜰 것이다. 이 에디터에서는 비헤이비어 트리 모드와 블랙보드 모드의 두 편집 모드가 있다. 툴바의 오른쪽 아이콘을 클릭해서 두 모드 중의 하나를 선택할 수 있다. 우리의 경우 블랙보드 모드가 선택되어 있을 것이다.
이제, 첫 번째 키를 추가하자.
패트롤할 위치의 액터 객체를 저장할 변수에 해당하는 새 키를 만들자. 먼저, 왼쪽의 **블랙보드** 탭에서 **새 키**를 클릭하자. 그리고, 드롭다운 메뉴에서 키의 타입을 선택해야 하는데 **오브젝트(Object)**를 선택하자. 이름은 디폴트인 **오브젝트Key**를 **ActorToCheck**로 수정하자.



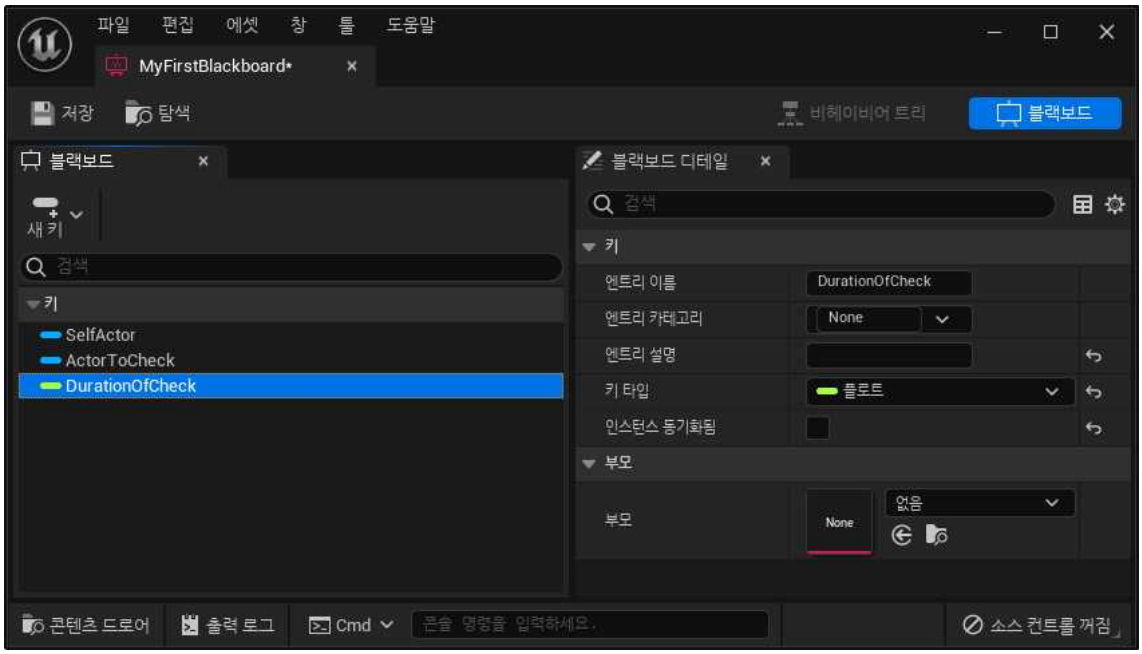
<참고> 블랙보드를 생성하면 초기에 디폴트로 **SelfActor** 이름의 키가 하나 존재한다. 아직 용도가 불분명한 것이므로 무시하고 그대로 두자.

5. 오른쪽의 **블랙보드 디테일** 탭에서 **키 타입(Key Type)** 속성을 찾고 그 왼쪽의 세모 아이콘을 클릭하여 펼치면 **베이스 클래스(Base Class)** 속성이 보인다. 속성값이 디폴트로 **Object**로 되어 있을 것이다. 이것을 **Actor**로 수정하자.



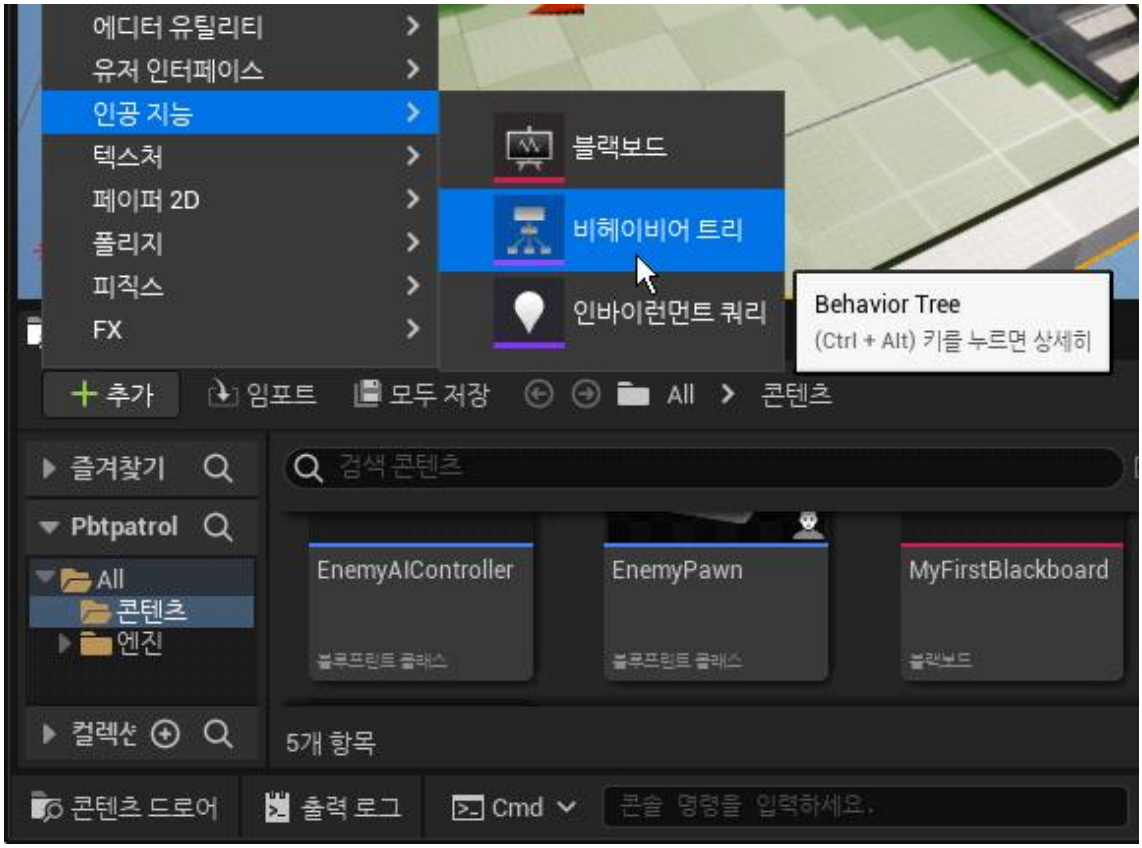
툴바의 **저장** 버튼을 클릭하여 저장하자.

6. 이제, 두 번째 키를 추가하자.
먼저, 왼쪽의 **블랙보드** 탭에서 **새 키**를 클릭하자. 그리고, 드롭다운 메뉴에서 키의 타입은 **플로트(Float)**를 선택하자. 이름은 디폴트인 **플로트Key**에서 **DurationOfCheck**로 수정하자.

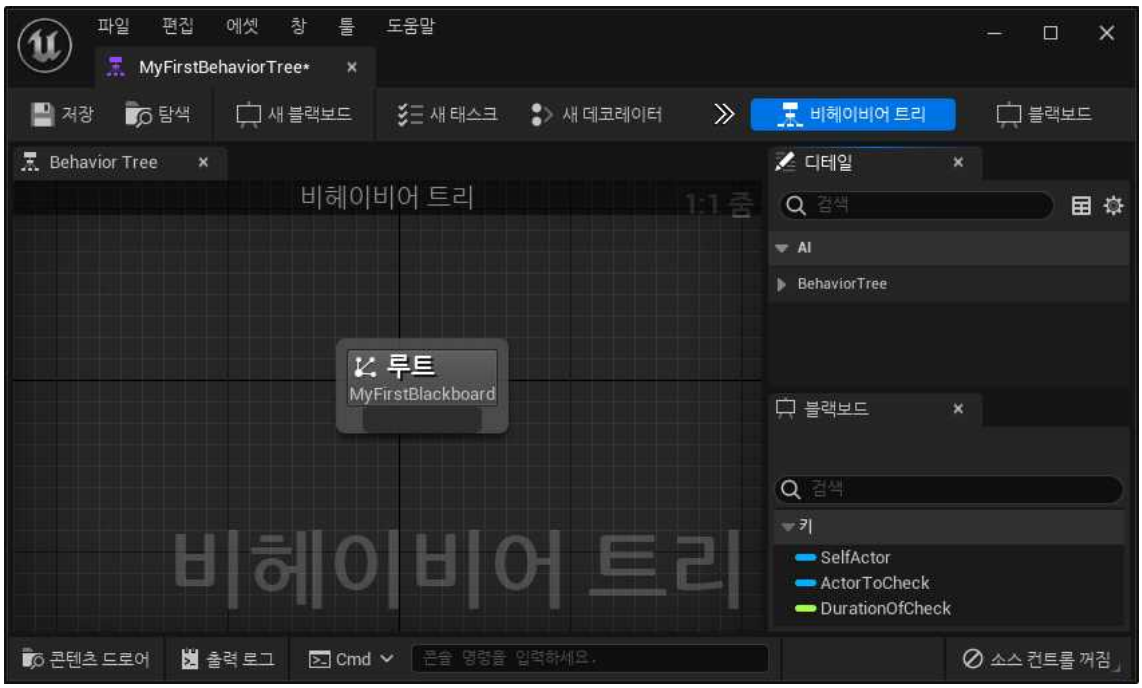


툴바의 **저장** 버튼을 클릭하여 저장하자.
이제, 블랙보드를 생성하고 블랙보드에 키를 추가하는 과정을 모두 완료하였다.

7. 한편, 비헤이비어 트리는 인공지능의 알고리즘을 구현한다. 즉, 상황에 대해 판단을 하고 그 판단에 따라 인공지능이 어떤 일을 하도록 한다. 지금부터, 비헤이비어 트리를 만들고 거기에서 사용할 블랙보드를 지정하자.
먼저 비헤이비어를 생성하자. **콘텐츠 브라우저**에서 **콘텐트** 폴더를 선택하고, 툴바의 **+추가**를 클릭하자. 드롭다운 메뉴에서 **인공 지능** » **비헤이비어 트리**를 선택하자. 이름은 **MyFirstBehaviorTree**로 지정하자.



8. **MyFirstBehaviorTree**를 더블클릭하여 **비헤이비어 트리 에디터**를 열자.

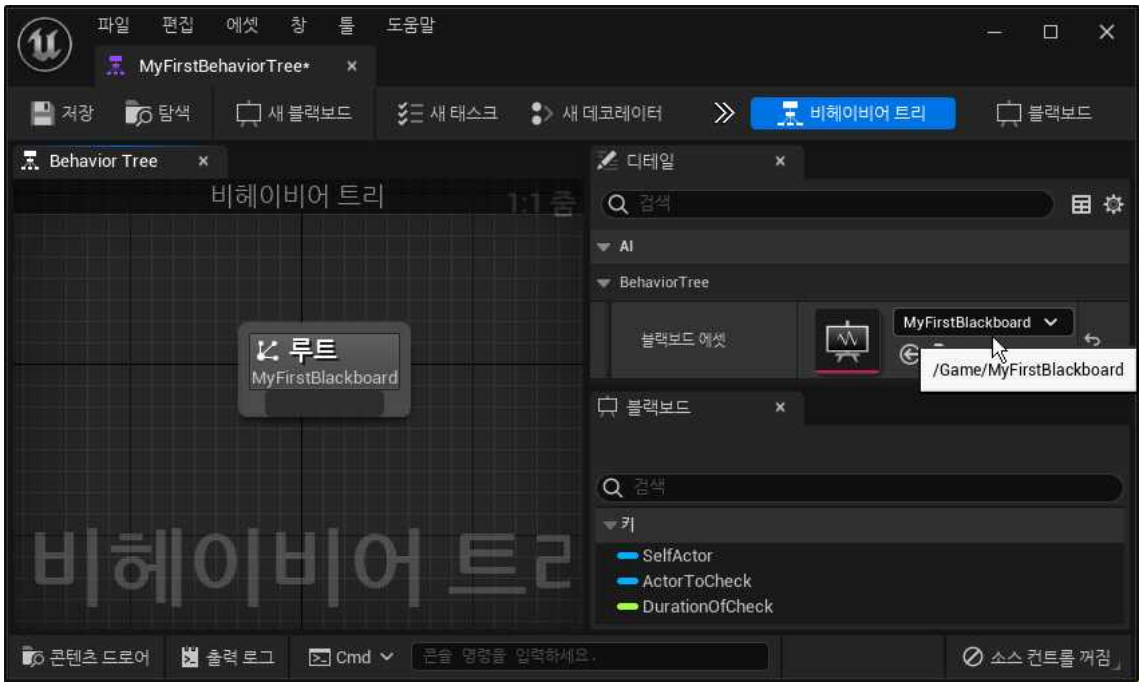


비헤이비어 트리 에디터를 살펴보자.

비헤이비어 트리 에디터와 블랙보드 에디터는 동일한 창에 배치되어 있고 오른쪽 상단의 탭을 클릭하여 서로 이동할 수 있다.

왼쪽의 격자판에는 디폴트로 루트 노드가 디폴트로 배치되어 있을 것이다. 여기서 비헤이비어 트리 그래프를 작성하면 된다.

9. 우리가 만든 블랙보드를 이 비헤이비어 트리에서 사용할 수 있도록 비헤이비어 트리의 속성에 지정하자.
먼저, **MyFirstBehaviorTree 비헤이비어 트리 에디터**에서 오른쪽 **디테일** 탭에서 **AI** 섹션 아래에 **BehaviorTree** 속성을 찾아 펼치자. 그 아래에 **블랙보드 애셋(Blackboard Asset)** 속성이 있을 것이다. 오른쪽에 속성값을 살펴보자. 엔진이 개발자의 편의를 위해서 이미 **MyFirstBlackboard**를 지정해 주었을 수 있다. 만약 속성값으로 **없음**으로 되어 있다면, 이것을 클릭하고 드롭다운 메뉴에서 **MyFirstBlackboard**를 선택하자. 블랙보드가 연결되었다면 격자판의 루트 노드에서도 연결된 블랙보드 이름이 노드 아래에 작게 표시된다.
디테일 탭 아래에 있는 **블랙보드** 탭에서는 연결된 블랙보드가 가지고 있는 키들이 나열된다. 우리가 블랙보드에 추가해둔 키인 **ActorToCheck**와 **DurationofCheck**가 보일 것이다.
툴바의 **저장** 버튼을 클릭하여 저장하자.
이제 비헤이비어 트리와 블랙보드가 연결되었다.

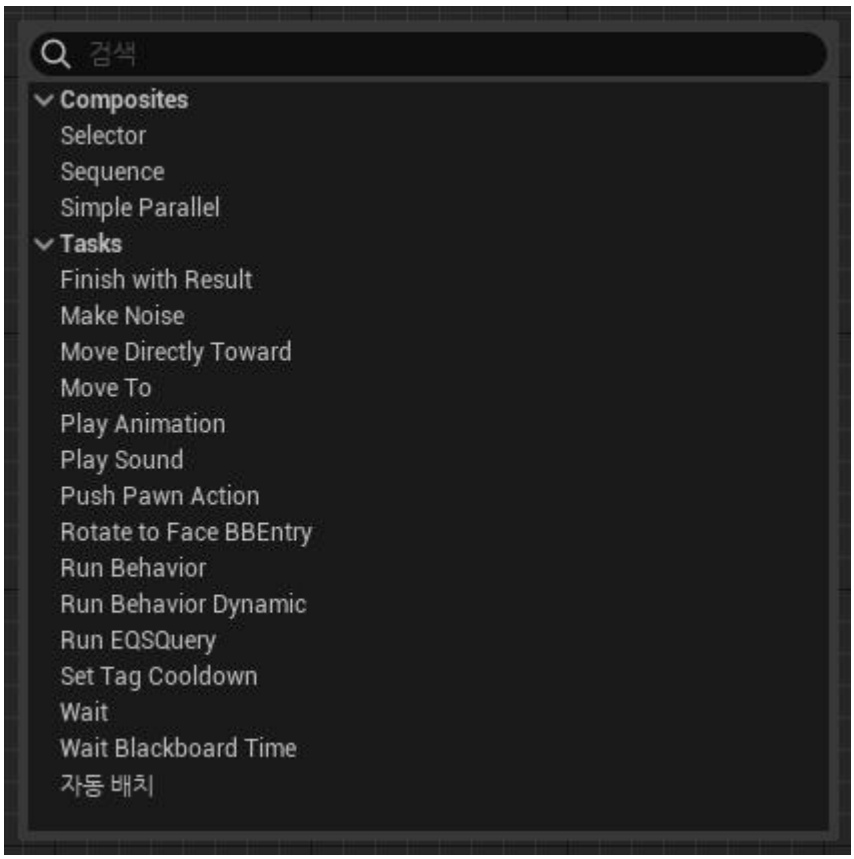


지금까지, 우리가 앞으로 인공지능 알고리즘을 구현할 비헤이비어 트리인 **MyFirstBehaviorTree**를 생성하였다. 그리고 블랙보드 **MyFirstBlackboard**를 생성하였고 이 블랙보드를 비헤이비어 트리에서 사용할 수 있도록 비헤이비어 트리의 **Blackboard Asset** 속성에 지정하였다.
우리는 앞으로 비헤이비어 트리를 통해서 블랙보드를 접근할 것이다.

4. 비헤이비어 트리 구동을 위한 설정과 커스텀 태스크 만들기

우리는 이전에서 **내비 메시**를 준비하였고, **블랙보드**와 **비헤이비어 트리**를 준비하였다. 이제 캐릭터와 연결하는 작업을 진행해보자. **비헤이비어 트**리를 캐릭터에 적용하기 위해서는 여러 세팅 절차를 거쳐야 한다.

또한, 이 절에서는 커스텀 **태스크**를 생성하는 방법을 학습한다.
엔진은 3개의 **Composite** 노드와 11개의 **Task** 노드를 제공한다. **Composite** 노드는 커스텀 노드를 작성할 수 없으며 **Task** 노드는 커스텀 노드를 작성할 수 있다.



엔진이 제공하는 11개의 **태스크** 노드만으로는 우리가 원하는 일을 마음대로 처리할 수 없다. 따라서 우리가 직접 커스텀 **태스크** 노드를 생성하고 이를 사용할 수 있어야 한다.

커스텀 **태스크** 노드는 쉽게 생성할 수 있다. 마치 함수 그래프를 작성하는 것과 유사하다.
태스크의 그래프에서 **ReceiveExecuteAI** 이벤트 노드를 배치하고 이 이벤트 노드의 그래프를 작성하면 된다. 이 이벤트 노드는 **태스크** 노드가 실행될 때에 호출되는 이벤트 노드이다. 따라서 **태스크**에서 수행되는 모든 작업은 이 이벤트 그래프에 구현하면 된다.
비헤이비어 트리에서의 모든 노드는 반드시 **true** 또는 **false**의 값을 리턴하도록 구현해야 한다. 이를 위해서, 이벤트 그래프의 마지막 종료 시점에서는 반드시 **FinishExecute** 노드가 실행되도록 해야 한다. 이 노드를 통해서 **true** 또는 **false**를 리턴하도록 해야 한다.

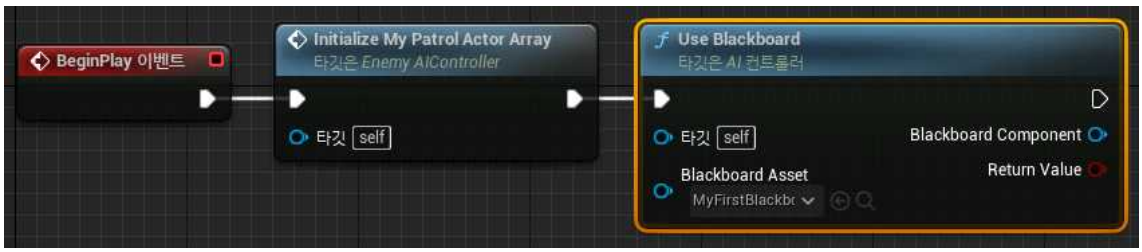
예제를 통해서 학습해보자.

---

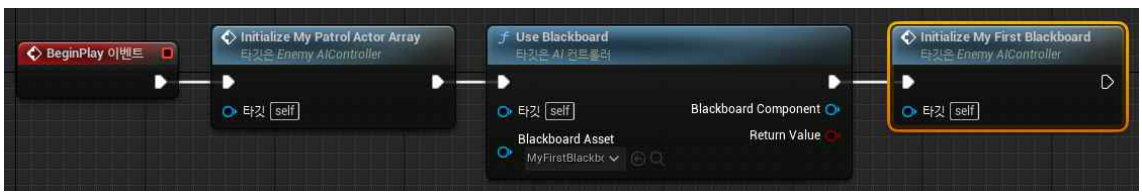
1. 이전 예제의 프로젝트 `Pbtpatrol`에서 이어서 계속하자.

2. 이전 예제에서는 **EnemyAIController**의 **BeginPlay** 이벤트 그래프에서 **PatrolActor** 함수 호출 노드를 실행한다. **PatrolActor** 이벤트 그래프에서는 패트롤을 반복하도록 구현하였다.
우리는 이번 예제에서는 **PatrolActor** 이벤트 그래프를 사용하지 않을 것이다. 그 대신 **비헤이비어 트리**로 동일한 과정을 수행하도록 구현할 것이다.
따라서 우선, **EnemyAIController** 의 **BeginPlay** 이벤트 그래프의 마지막에 붙어있는 **PatrolActor** 함수 호출 노드를 삭제하자. 삭제후에는 **BeginPlay** 이벤트 그래프는 다음과 같은 모습이 될 것이다.

3. 우리가 만든 **블랙보드**를 사용할 수 있도록 지정하자.
EnemyAIController 블루프린트 에디터의 이벤트 그래프 탭으로 가자.
먼저, **BeginPlay** 이벤트 그래프에서 **InitializeMyPatrolActorArray** 노드의 출력핀을 당기고 **UseBlackboard** 함수를 호출하자. **UseBlackboard** 함수 노드의 **BlackboardAsset** 입력핀에 **MyFirstBlackboard**를 지정하자.
이 **UseBlackboard** 노드는 AI가 블랙보드를 사용할 수 있도록 준비해준다. 따라서 이후로부터 AI는 우리가 만든 **블랙보드**를 사용할 수 있다.

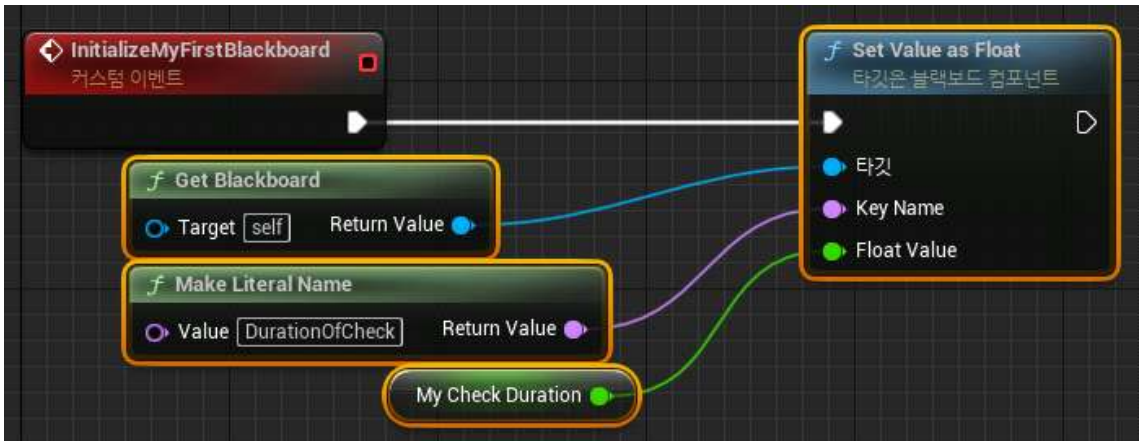


4. 우리가 만든 **블랙보드**를 초기화하는 이벤트 노드를 추가하자.
격자판의 빈 공간에서 우클릭하고 액션선택 창에서 **커스텀 이벤트 추가**를 선택하여 커스텀 이벤트 노드를 추가하고 이름을 **InitializeMyFirstBlackboard**로 수정하자. 아직 **InitializeMyFirstBlackboard** 이벤트 그래프를 완성하지 않았지만 **InitializeMyFirstBlackboard** 이벤트 그래프가 호출되도록 **BeginPlay** 이벤트 그래프의 맨 뒤에 **InitializeMyFirstBlackboard** 함수 호출 노드를 추가하자.



5. 이제 **InitializeMyFirstBlackboard** 이벤트 그래프를 작성해보자.
먼저, **Get Blackboard** 함수 노드를 배치하자.

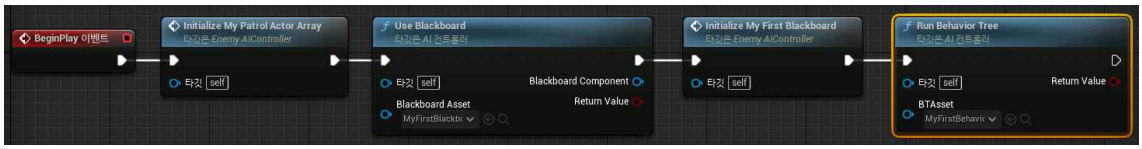
그다음, **Get Blackboard** 노드의 출력핀을 드래그하고 **Set Value as Float** 노드를 배치하자. 그리고 이벤트 노드의 실행핀을 연결하자.
그다음, **Set Value as Float** 노드의 **Key Name** 입력핀을 왼쪽으로 드래그하고 **Make Literal Name** 노드를 배치하자. 노드의 **Value** 입력핀에 블랙보드에서의 키 이름인 **DurationOfCheck**를 입력하자.
그다음, **Set Value as Float** 노드의 **Float Value** 입력핀을 왼쪽으로 드래그하고 **변수로 승격**을 선택하자. 변수 이름은 **MyCheckDuration**으로 수정하자. 컴파일하자.
그다음, 변수 **MyCheckDuration**를 선택하고 오른쪽 디테일 탭을 보면 가장 아래에 **기본값** 영역이 나타난다. **MyCheckDuration** 속성값을 디폴트인 0에서 1로 수정하자.
이제 다음과 같은 그래프가 완성되었다.



<참고> 우리는 컨트럴로에서 **MyCheckDuration** 변수를 선언하고 디폴트 값을 1로 지정하였다. 한편, 다른 방법으로 **EnemyPawn**으로부터 받아오도록 구현할 수도 있다. 이를 위해서는 **EnemyPawn**에서 플로트 변수를 선언하고 이를 편집가능으로 하자. 이렇게 해두면 레벨 에디터에서 **EnemyPawn**를 배치한 후에 인스턴스의 속성값을 편집하는 방식으로 편리하게 값을 지정할 수 있다. 그리고, **EnemyAIController**에서 **GetControlledPawn**을 실행하여 폰을 얻고 이를 **EnemyPawn**으로 형변환한 후에 해당 변수에 접근하여 값을 가져오면 된다.

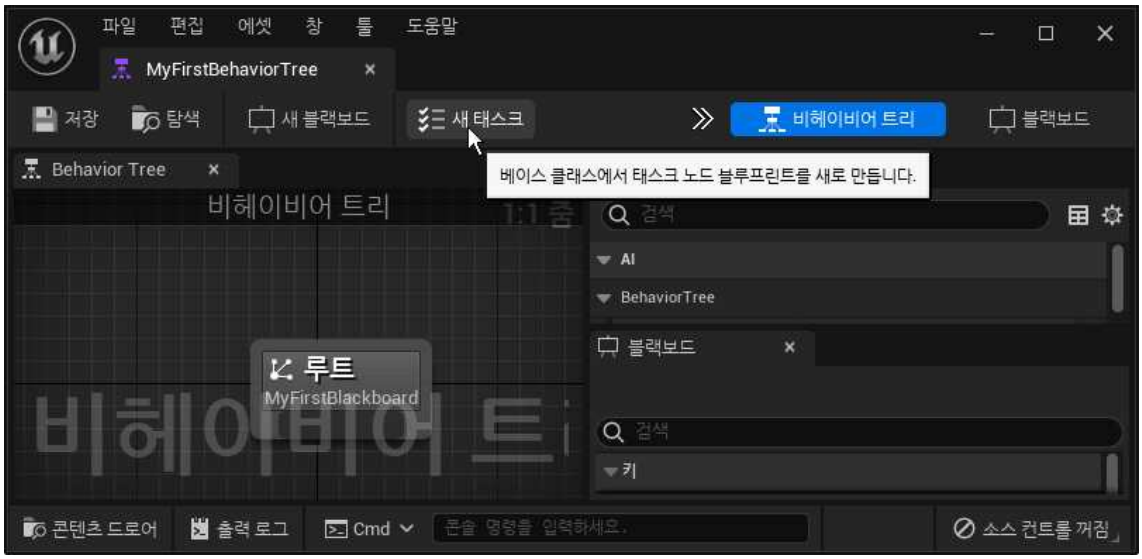
<참고> **MakeLiteralName** 함수에 대해서 알아보자.
엔진은 **MakeLiteral** 뒤에 타입명 붙은 여러 노드를 제공한다. 함수명 뒤에 붙을 수 있는 타입명은 **Bool**, **Byte**, **Float**, **Double**, **Int**, **Int64**, **Name**, **Text**, **String**이다.
MakeLiteral* 함수는 상수값을 만들어주는 함수이다. **MakeLiteral*** 함수 노드의 입력은 상수값이고 출력은 그 상수값을 제공하는 출력핀이다.
대부분의 경우에는 **MakeLiteral*** 함수가 필요없으나 종종 **MakeLiteral*** 함수가 필요한 경우가 있다. 어떤 노드에는 입력핀에 상수를 직접 입력하는 입력상자가 달려 있지 않은 경우가 있다. 이 경우에는 변수와의 연결로만 입력값을 제공해야 한다. 이럴 때에 변수를 만들지 않고 **MakeLiteral*** 함수를 추가하여 그 출력값을 연결해주면 된다.

6. 우리의 비헤이비어 트리가 실행되도록 하자.
EnemyAIController 블루프린트 에디터의 이벤트 그래프 탭에서 **BeginPlay** 이벤트 그래프의 마지막 위치로 가자. 그리고, 마지막 노드의 실행핀을 당기고 **Run Behavior Tree** 함수를 배치하자. 그다음, **Run Behavior Tree** 함수 노드의 **BTAsset** 입력핀에 **MyFirstBehaviorTree**를 지정하자.



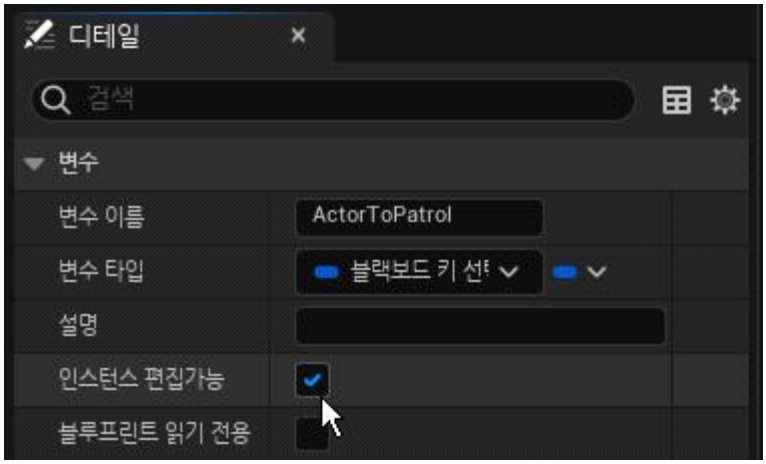
이제, 우리의 비헤이비어 트리가 실행되도록 하였다.

7. 이제부터, 비헤이비어 트리에서 사용될 태스크들을 작성해보자.
콘텐츠 브라우저에서 **MyFirstBehaviorTree**를 더블클릭하자. **비헤이비어 트리 에디터**가 열릴 것이다. 새 커스텀 태스크를 만들어보자. 툴바의 **새 태스크** 버튼을 클릭하자. 태스크가 생성되고 태스크의 블루프린트 에디터가 열릴 것이다. 각 태스크는 독립된 블루프린트 클래스이므로 각 태스크마다 해당 태스크의 블루프린트 에디터가 열린다.

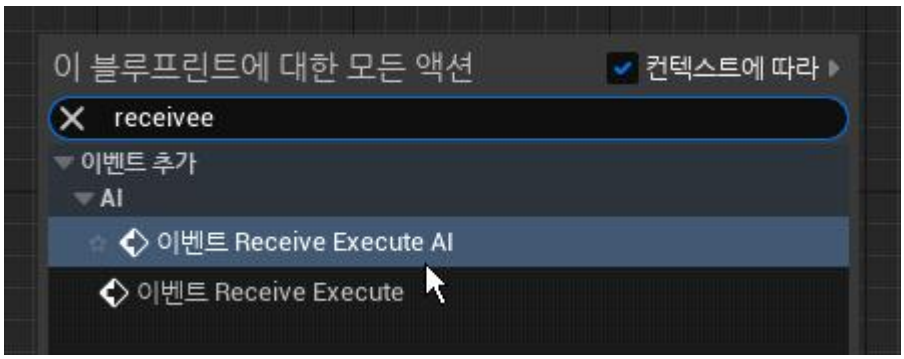


한편, 태스크 애셋이 생성될 때에 저장 위치를 묻는 창이 뜰 것이다. 디폴트로 **콘텐츠** 폴더에 **BTTask_BlueprintBase_New**와 같은 이름으로 저장되도록 보여준다. 우리는 애셋 이름을 **BTTask_GetActorToPatrol**로 수정하자.

8. **BTTask_GetActorToPatrol** 태스크가 생성된 후에 태스크 작성을 위한 블루프린트 에디터가 뜰 것이다. 만약 창이 뜨지 않는다면 콘텐츠 브라우저에서 **BTTask_GetActorToPatrol**를 더블클릭하면 태스크의 블루프린트 에디터가 뜬다.
BTTask_GetActorToPatrol 태스트의 블루프린트 에디터에서 왼쪽의 **내 블루프린트** 탭에서 변수를 추가하자. 변수 이름을 **ActorToPatrol**로 수정하자.
변수 유형을 **구조체** 아래의 **블랙보드 키 선택 툴(Blackboard Key Selector)**로 수정하자. 이 변수 유형은 비헤이비어 트리와 연관되어 구동되는 블루프린트에서만 사용되는 특수한 변수 유형이다. 이 유형의 블루프린트에서의 변수를 통해서 **블랙보드**에 있는 **키**로 접근할 수 있다.
그리고, **인스턴스 편집가능**에 체크하여 값을 수정할 수 있도록 하자. 태스크에서만 사용되는 이러한 특수한 유형의 변수를 편집가능으로 체크하면 **비헤이비어 트리 에디터**에서 **디테일** 탭에서 디폴트값을 수정할 수 있게 된다. 컴파일하자.



9. 이제 **BTTask_GetActorToPatrol** 태스크의 그래프를 작성하자.
격자판이 완전히 비어있을 것이다. 가장 먼저 이벤트 노드를 배치해야 한다.
빈 곳에서 우클릭하고 **ReceiveExecuteAI** 이벤트 노드를 검색하여 배치하자.

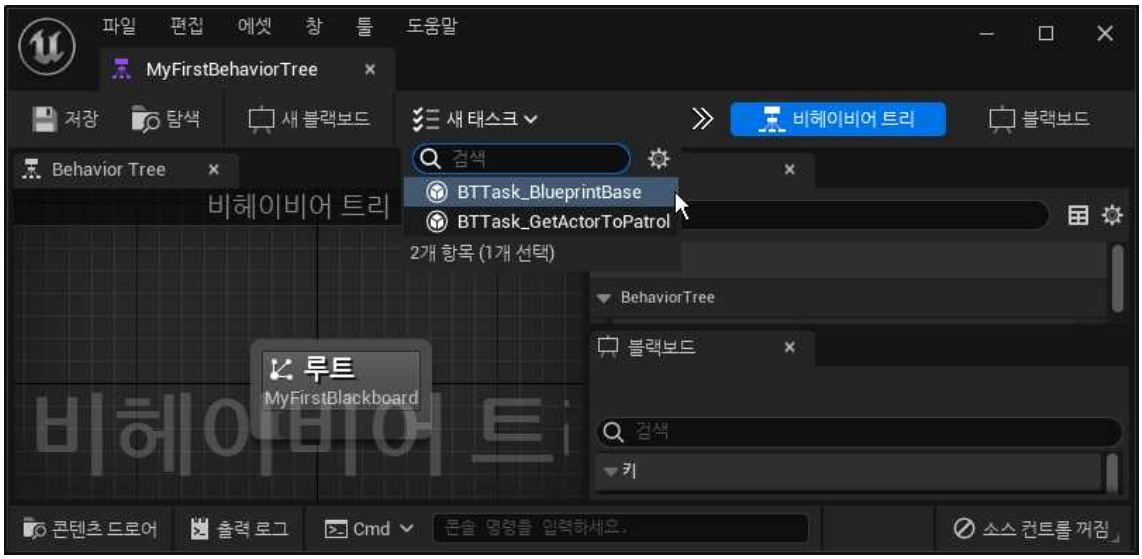


이 노드는 태스크 노드가 실행될 때에 호출되는 이벤트 노드이다. 태스크에서 수행되는 모든 작업은 이 이벤트 그래프에 구현하면 된다.

10. 그다음, **ReceiveExecuteAI** 이벤트 노드의 **OwnerController**의 출력핀을 당기고 **EnemyAIController**에 형변환 노드를 배치하자.
그다음, 형변환 노드의 **An EnemyAIController** 출력핀을 당기고 **GetPatrolActor** 함수 노드를 배치하자.
그다음, **GetPatrolActor** 함수의 출력핀을 당기고 **Set Blackboard Value as Object** 함수 노드를 배치하자.
이 함수는 블랙보드에 값을 지정하는 함수이다.
지정할 키를 노드의 **Key** 입력핀에 넣어주어야 한다.
이를 위해서 왼쪽 내 블루프린트 탭에서 변수 **ActorToPatrol**을 드래그해서 **Get** 노드를 배치하고 **Get** 노드의 출력핀을 **Set Blackboard Value as Object** 함수 노드의 **Key** 입력핀에 연결하자. 그리고 실행핀도 이전의 형변환 노드에 연결해주자.
그다음, **Set Blackboard Value as Object** 노드의 실행핀을 드래그하고 **FinishExecute** 노드를 배치하자.
그리고 노드의 **Success** 입력핀에 체크하여 **true**가 되도록 하자.

컴파일하자. 이제 **BTTask_GetActorToPatrol** 태스크를 완성하였다.

11. 이제 두 번째 태스크인 **BTTask_IncreaseActorIndex**를 작성해보자.
비헤이비어 트리 에디터에서 툴바의 **새 태스크** 버튼을 클릭하자. 이제는 바로 생성되지 않고 선택창이 뜰 것이다. 어느 클래스를 부모 클래스로 하여 태스크 노드를 만들 것인지를 선택하는 창이다. 우리는 **BTTask_BlueprintBase**를 선택하자.



BTTask_BlueprintBase는 모든 태스크 노드의 베이스 클래스이다. 우리는 대부분의 경우 이것을 선택하면 된다. 만약 미리 만들어진 태스크 노드의 파생 클래스로 만들고하 하는 경우에만 다른 클래스를 선택하면 된다. 처음에는 태스크 클래스가 **BTTask_BlueprintBase**만 있었으므로 자동으로 선택해주면서 선택창을 보여주지 않았다.
애셋 저장 이름을 묻는 창이 뜰 것이다. 디폴트로 **BTTask_BlueprintBase_New**와 같은 이름으로 보이는데 우리는 **BTTask_IncreaseActorIndex**로 수정하여 저장하자.

12. **BTTask_IncreaseActorIndex** 태스크의 블루프린트 에디터 창이 뜰 것이다. 창이 없다면 콘텐츠 브라우저에서 애셋을 더블클릭하면 된다.
격자판의 빈 곳에서 우클릭하고 **ReceiveExecuteAI** 이벤트 노드를 검색하여 배치하자.
그다음, 이벤트 노드의 **OwnerController**의 출력핀을 당기고 **EnemyAIController**에 형변환 노드를 배치하자.
그다음, 형변환 노드의 **An EnemyAIController** 출력핀을 당기고 **IncreaseActorIndex** 함수 노드를 배치하자.
그다음, **IncreaseActorIndex** 노드의 실행핀을 드래그하고 **FinishExecute** 노드를 배치하자. 그리고 노드의 **Success** 입력핀에 체크하여 **true**가 되도록 하자.

컴파일하자. 이제 **BTTask_IncreaseActorIndex** 태스크를 완성하였다.

이제 비헤이비어 트리를 작성하기 전의 모든 준비 과정을 완료하였다.

5. 비헤이비어 트리 작성하기

이제 비헤이비어 트리를 작성해보자.

먼저, 비헤이비어 트리의 노드에 대해서 다시 살펴보자.
루트 노드를 제외한 비헤이비어 트리의 노드의 종류에는 **Task** 노드와 **Composite** 노드가 있다. **Task** 노드는 실행할 블루프린트 코드를 가진다. 여기에 NPC에서 실행할 알고리즘을 구현하면 된다. **Task**는 실행 성공 여부를 리턴해야 한다. 즉, **Task** 내에서는 마지막에 **FinishExecute** 함수를 실행하면서 **true** 또는 **false**를 리턴한다.
Composite 노드에는 **Selector**, **Sequence**, **Simple Parallel**의 노드가 있다. **Selector** 노드는 트리의 각 노드를 왼쪽에서 오른쪽의 순서로 방문하며 실행한다. 만약 성공 노드를 만나면 실행을 중지하고 트리의 위로 돌아간다. **Sequence** 노드는 **Selector** 노드와 동일한 순서로 방문하며 실행하지만, 실패 노드를 만나면 실행을 중지하고 트리의 위로 돌아간다. **Simple Parallel** 노드는 두 작업을 동시에 병렬로 실행한다. 예를 들어 공격하면서 이동하는 행위를 동시에 하는 경우에 해당한다. 다만, 두 작업 중에서 하나는 단일 **Task** 노드여야 하고 다른 하나는 트리여도 된다.

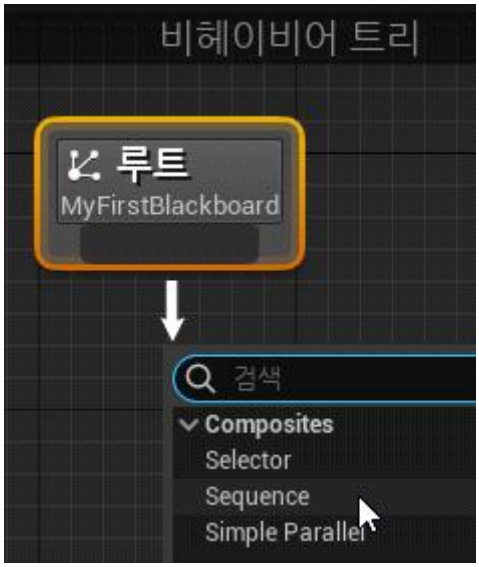
비헤이비어 트리를 구성하는 각 노드는 트리 형태를 이루도록 노드가 서로 연결되어 있다. 연결된 두 노드는 부모에서 자식으로의 방향성 링크로 연결된다. 연결의 시각적 표현을 위해서 노드의 위쪽에 입력핀이 있고 노드의 아래쪽에 출력핀이 있다. 입력핀과 출력핀 위에 마우스를 올리면 노란색으로 하이라이트된다.
자신의 부모 노드로부터 들어오는 링크는 자신의 입력핀으로 들어오고 자신의 자식 노드로 나가는 링크는 자신의 출력핀에서 나간다.
부모 노드는 하나만 있을 수 있으므로 입력핀에는 하나의 링크만 들어올 수 있다. 자식 노드는 많을 수 있으므로 출력핀에서 나가는 링크는 많을 수 있다.
루트 노드는 최상위 노드이므로 입력핀이 없다. 태스크 노드는 최하위 노드이므로 출력핀이 없다. 컴포짓 노드는 중간 수준 노드이므로 입력핀과 출력핀이 모두 있다.

예제를 통해서 학습해보자.

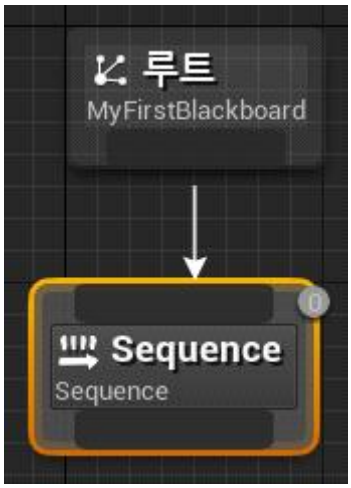
---

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

2. 이제부터 비헤이비어 트리를 작성해보자.
콘텐츠 브라우저에서 **MyFirstBehaviorTree**를 더블클릭하여 **비헤이비어 트리 에디터**를 열자. **비헤이비어 트리 에디터**에는 루트 노드가 디폴트로 배치되어 있을 것이다.
루트 노드의 출력핀에서 드래그를 시작해서 빈 곳에서 버튼을 떼면 선택창이 뜬다. 여기서 **Sequence** 노드를 선택하여 배치하자.

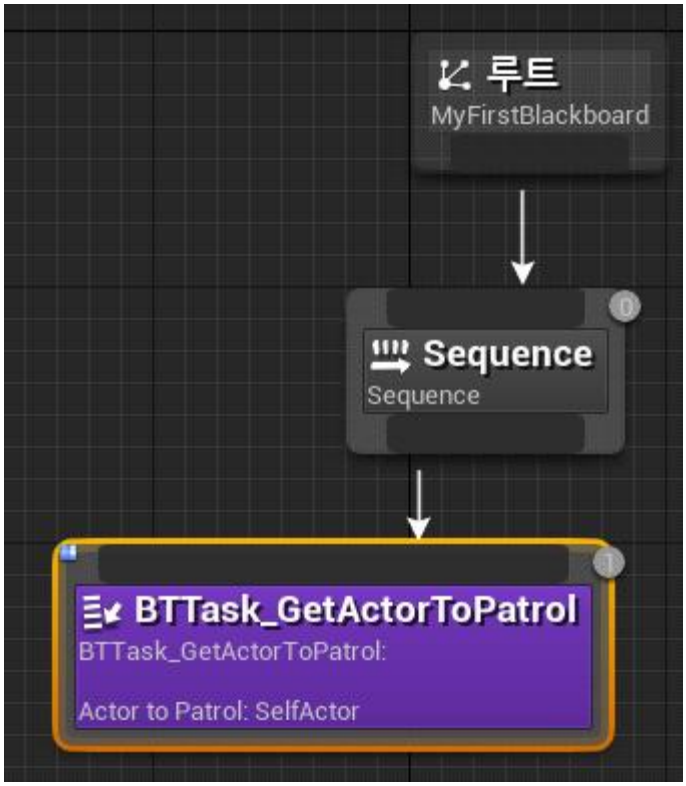


3. Sequence 노드가 배치되고 루트 노드의 출력핀에서 Sequence 노드의 입력핀으로 화살표가 연결될 것이다.

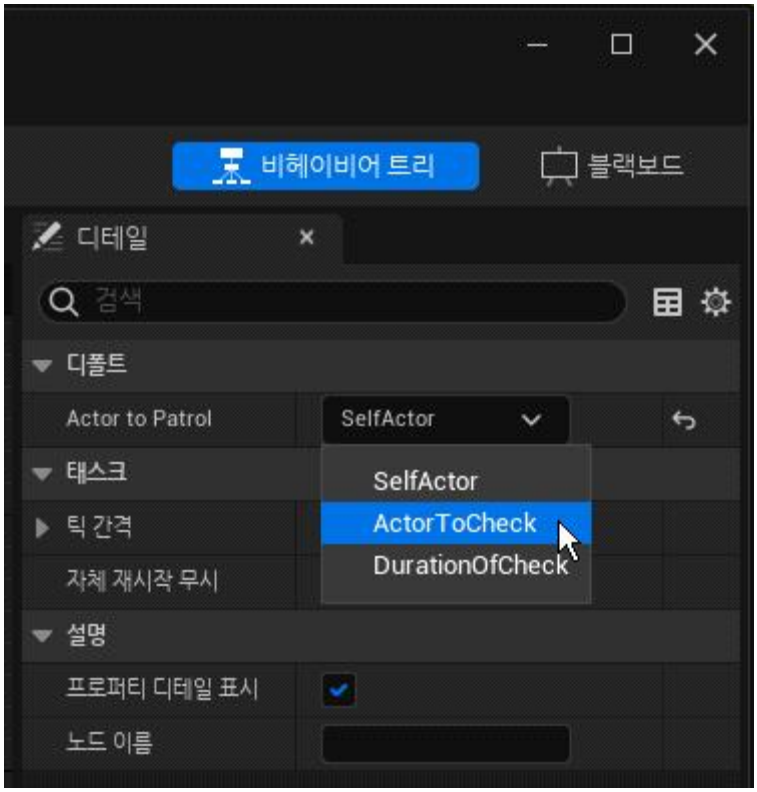


노드의 번호 매김에 대해서 알아보자.
루트 노드를 제외한 모든 노드는 오른쪽 위에 번호가 매겨진다. 루트 노드 아래의 노드부터 번호가 0부터 매겨진다. 번호는 실행 순서에 해당하는 깊이 우선 탐색의 순서로 매겨진다.
한 노드에 여러 자식 노드가 있으면 자식 노드는 배치 순서에 따라 왼쪽부터 오른쪽으로 번호가 매겨진다. 마우스로 자식 노드의 위치를 드래그하여 옮기면 배치 순서도 자동으로 바뀌게 된다.

4. 배치된 Sequence 노드에서 노드의 아래에 있는 출력핀으로 마우스를 움직이면 출력핀이 노란색으로 바뀔 것이다. 출력핀에서 아래로 드래그하고 선택창에서 Task 노드 중에서 BTTask_GetActorToPatrol 를 선택하자. 우리가 작성한 태스크가 배치될 것이다.

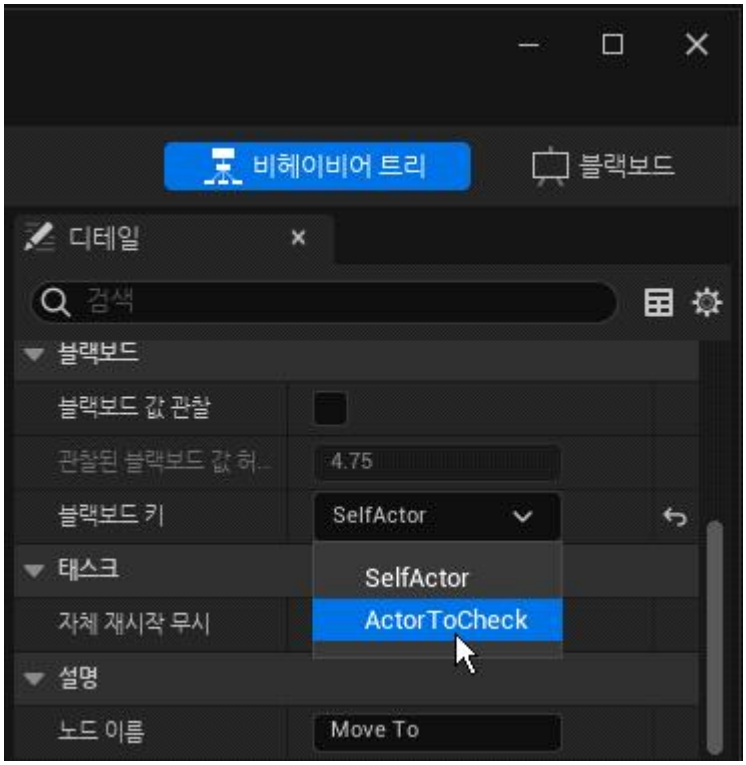


5. **MyFirstBehaviorTree 비헤이비어 트리 에디터**에서 **BTTask_GetActorToPatrol** 태스크 노드를 선택하고 **디테일** 탭을 살펴보자. **Actor to Patrol** 속성이 보일 것이다. 이는 변수 유형이 **Blackboard Key Selector**인 변수 **ActorToPatrol**가 속성으로 표시된 것이다. 그 오른쪽을 클릭하면 선택 가능한 값들이 나열된다. 바로 블랙보드에 있는 키들이다. 우리는 **ActorToCheck**을 선택하자.



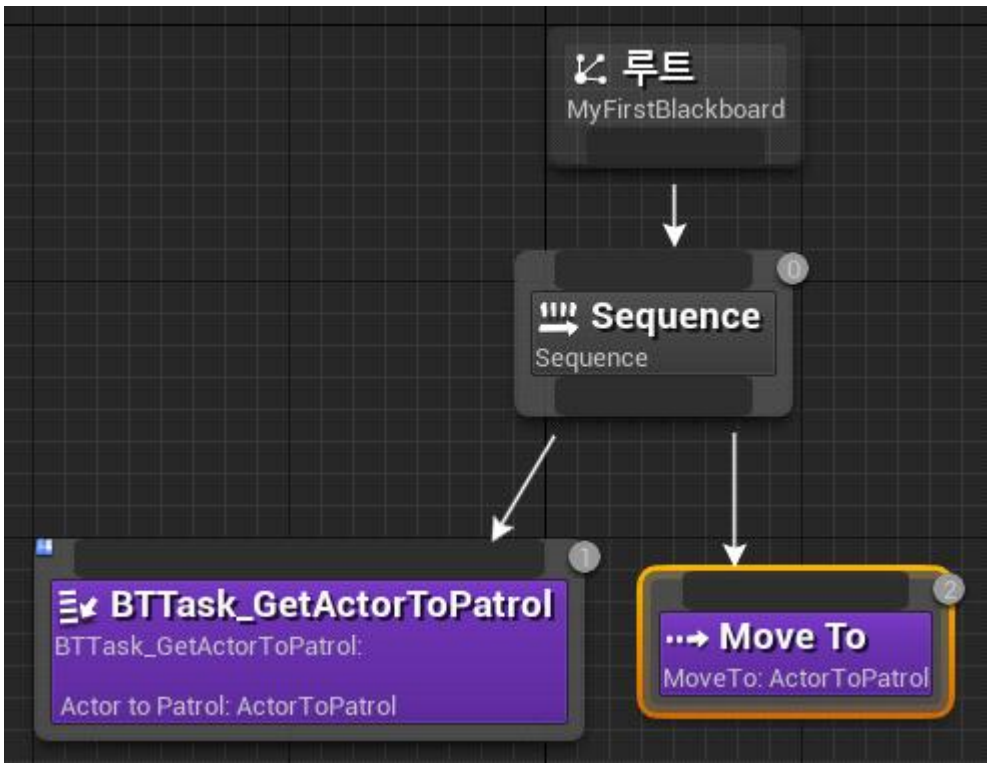
이제 **BTTask_GetActorToPatrol**에서의 변수 **ActorToPatrol**와 블랙보드에서의 키 **ActorToCheck**이 연결되었다. 변수 **ActorToPatrol**를 통해서 블랙보드에 읽고 쓸 수 있게 된 것이다.

6. 배치된 **Sequence** 노드의 출력핀에서 오른쪽 부분에서 아래로 드래그하고 선택창에서 **MoveTo** 태스크 노드를 선택하여 배치하자.
MoveTo 태스크는 엔진이 제공하는 태스크로, 블랙보드에 기록되어 있는 키에 해당하는 위치로 이동한다. 위치에 대한 키 값은 액터 또는 위치 벡터로 제공하면 된다.
그다음, **MoveTo** 태스크 노드의 **디테일** 창에서 **블랙보드** 영역에 있는 **블랙보드 키(Blackboard Key)** 속성 값에 **ActorToCheck**을 지정하자.

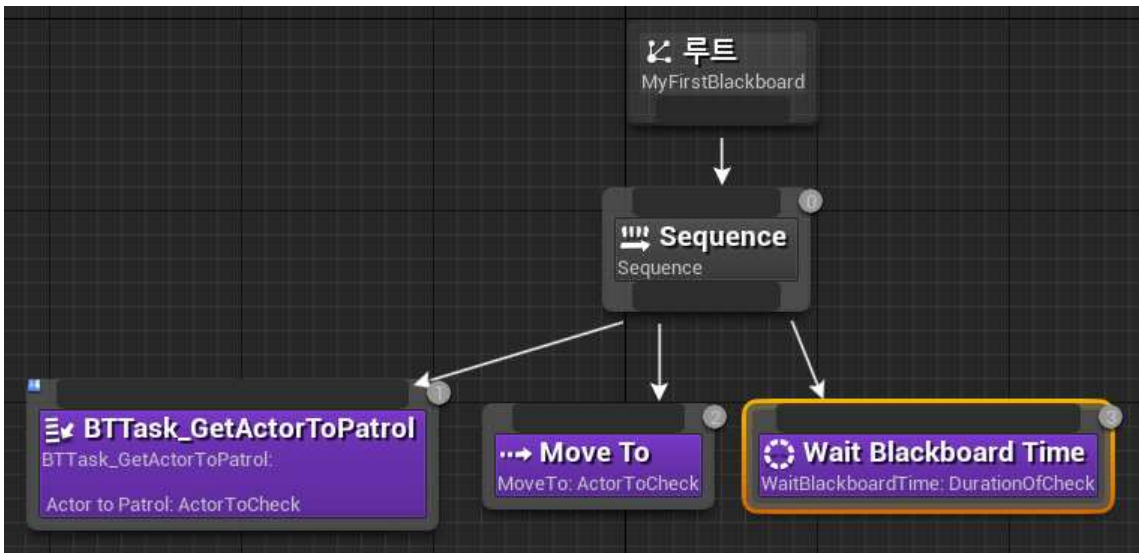


이제, **MoveTo** 태스크 노드가 실행되면 블랙보드의 **ActorToCheck** 키에 지정되어 있는 액터 위치로 이동할 것이다. 저장하자.

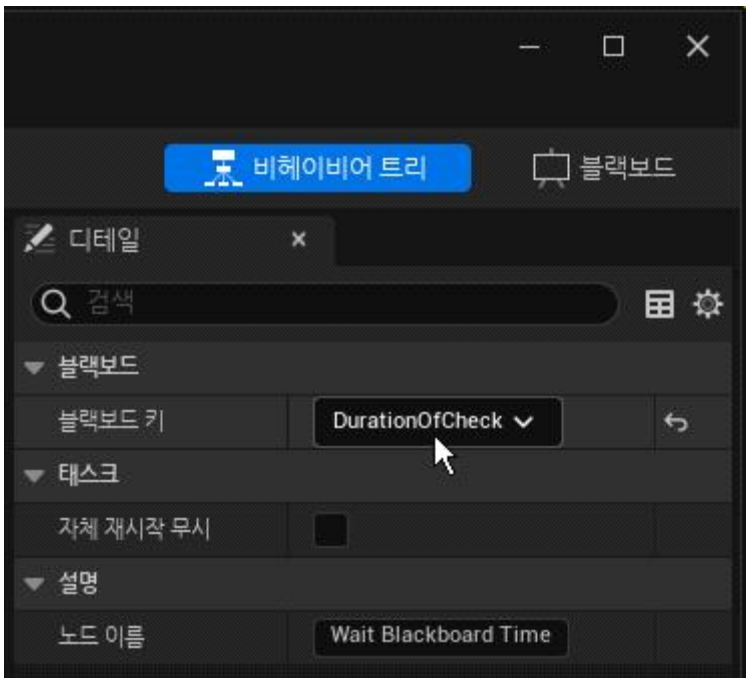
7. 이제, **Task** 노드를 비헤이비어 트리에 추가하였다. 비헤이비어 트리는 아래와 같은 모습이 되었을 것이다.



8. 배치된 **Sequence** 노드의 출력핀에서 오른쪽 부분에서 아래로 드래그하고 선택창에서 **Wait Blackboard Time** 태스크 노드를 선택하여 배치하자. 이 노드는 엔진에서 제공하는 태스크 노드로 블랙보드의 키에서 지정된 시간만큼 대기하는 작업을 진행한다. 저장하자. 비헤이이어 트리는 아래와 같은 모습이 되었을 것이다.

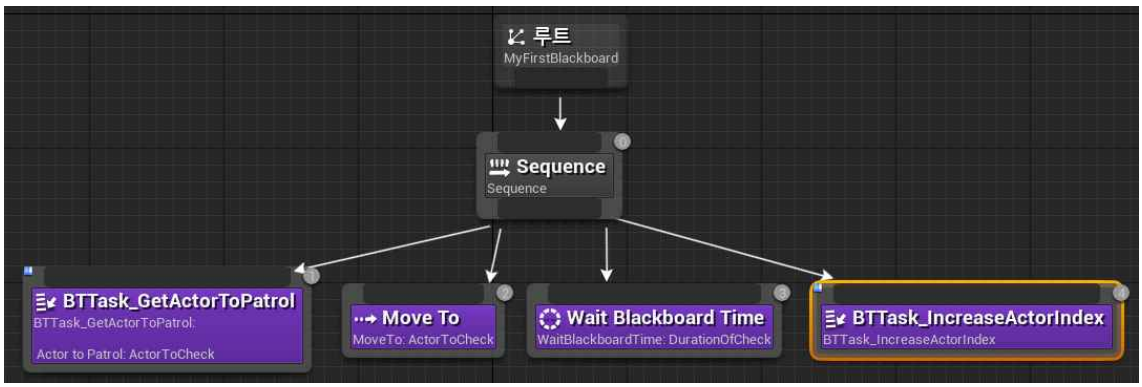


9. **Wait Blackboard Time** 태스크 노드를 선택하고 디테일 탭의 **Blackboard Key** 속성값을 살펴보자. 엔진이 이미 **DurationOfCheck**로 설정해두었을 것이다. 태스크는 이 키에 지정한 값만큼 대기할 것이다.



<참고> 엔진이 제공하는 또다른 태스크 노드인 **Wait** 태스크 노드가 있다. 이 노드는 **비헤이비어 트리 에디터**에서 지정해둔 시간만큼 대기하는 작업을 진행한다. 태스크 노드의 **디테일** 탭에 **Wait Time** 속성이 있다. 이 속성값에 대기할 시간을 초 단위로 설정하면 된다. 우리는 블랙보드의 활용법을 익히기 위해서 Wait 태스크 노드 대신 **Wait Blackboard Time** 태스크 노드를 사용하였다.

10. 배치된 **Sequence** 노드의 출력핀에서 오른쪽 부분에서 아래로 드래그하고 선택창에서 **BTTask_IncreaseActorIndex** 태스크 노드를 선택하여 배치하자. 저장하자.
이제, 우리의 **BTTask_IncreaseActorIndex** 태스크 노드를 비헤이비어 트리에 추가하였다. 비헤이이어 트리는 아래와 같은 모습이 되었을 것이다.



11. 이제 레벨 에디터에서 플레이해보자. 이전과 동일하게 각 지점을 패트롤할 것이다.
플레이할 때에 **비헤이비어 트리 에디터**를 관찰해보자. 실행 흐름이 동적으로 보이도록 표시될 것이다.

이 절에서는 비헤이비어 트리의 내부를 작성하는 방법에 대해서 학습하였다.

□