1. 언리얼 behavior tree이어야 하는 이유? (표준적인 behavior tree와의 차이점)

* 언리얼 엔진4에서의 behavior tree는 이벤트 주도형 behavior tree이다.

언리얼에서의 behavior tree는 관련된 변화를 지속적으로 확인하는 대신, 트리 내부에 변화를 발동시키는 이벤트를 수동적으로 기다린다. 따라서 퍼포먼스와 디버깅 양쪽 측면에 있어서 장점이 있다. 최적화가 잘 되어있다.

* 조건문이 리프 노드가 아니다.

트리에서 탐색하는 하나하나의 노드를 리프 노드라고 하는데, 언리얼 엔진의 behavior tree에서 액션을 갖고있는 노드는 “Task” 리프 노드이다. True와 false 반환 이외에는 아무것도 하지 않는다. 언리얼에서는 조건문을 “데코레이터”로 제작한다.

* 데코레이터의 장점

1. Behavior tree UI를 직관적이고 읽기 쉽게 만들어준다 -> 조건문이 자신이 제어하는 서브 트리의 루트에 있기 때문에 조건이 충족되지 않는 경우, 트리의 어느 부분에서 false가 발생하는지 바로 알 수 있다.
2. 모든 리프 노드가 액션 task이기 때문에 트리를 통해 실제 어떤 액션이 내려졌는지 알기 쉽다.
3. 그 데코레이터를 트리 내 중요 노드에서 이벤트를 기다리고 있는 관찰자 역할을 하도록 만드는 것이 쉽다. -> 언리얼 행동 트리의 이벤트 주도형 속성의 장점을 최대한 이끌어내는 것에 있어 매우 중요하다.

요약: 언리얼 만의 특별한 장점.

1. 명확하다: 직관적이고 이해하기 쉽다.
2. 디버깅이 편하다
3. 최적화가 쉽다.
4. 구체적인 계획?

* 세부 기획서에 적힌 알고리즘을 구현할 계획이다.

출처: [UE4 에서의 Behavior Tree – BBAGWANG](https://bbagwang.com/uncategorized/ue4-%EC%97%90%EC%84%9C%EC%9D%98-behavior-tree/)

* 언리얼 엔진4 5가지 종류의 노드

|  |  |
| --- | --- |
| 노드 유형 | 설명 |
| **Composite** | **분기의 루트를 정의하고, 그 분기가 어떻게 실행되는지에 대한 기본 규칙을 정의하는 노드다.** |
| **Task** | **비헤이비어 트리의 잎에 해당하는 것으로, 어떤 작업을 하며, 출력 연결이 없는 노드다.** |
| **Decorator** | **Conditionals, 조건문이라고도 한다.** **다른 노드에 붙어서 트리 내 분기 내지 노드 하나라도 그 실행 여부를 결정짓는다.** |
| **Service** | **컴포짓 노드에 붙어, 그 분기 실행 도중 정해진 빈도에 따라 실행된다.** **보통 블랙보드 업데이트나 검사를 하는 데 사용된다. 다른 비헤이비어 트리 시스템에서 전통적인 병렬 노드를 대체한다.** |
| **Root** | **컴포짓 노드에 붙어, 그 분기 실행 도중 정해진 빈도에 따라 실행된다.** **보통 블랙보드 업데이트나 검사를 하는 데 사용된다. 다른 비헤이비어 트리 시스템에서 전통적인 병렬 노드를 대체한다.** |