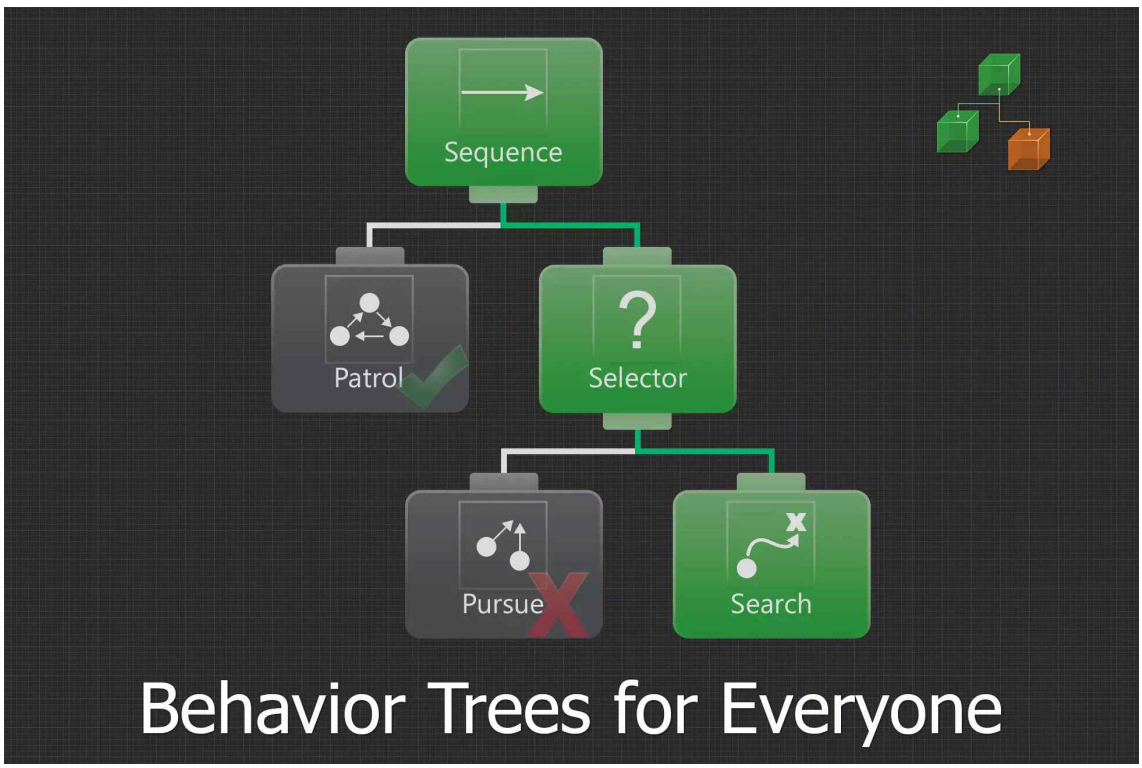
〈Behavior Designer〉

1. 한 줄 요약 / 핵심 장점

– Behavior Designer는 AI 행동 트리(Behavior Tree)를 Unity 안에서 비주얼 노드 기반으로 구현할 수 있는 유료 애셋입니다.

– 복잡한 AI 로직을 직관적으로 설계하고 디버깅할 수 있으며, FSM(상태 기계)보다 유연하고 확장 가능한 AI 설계에 적합합니다.

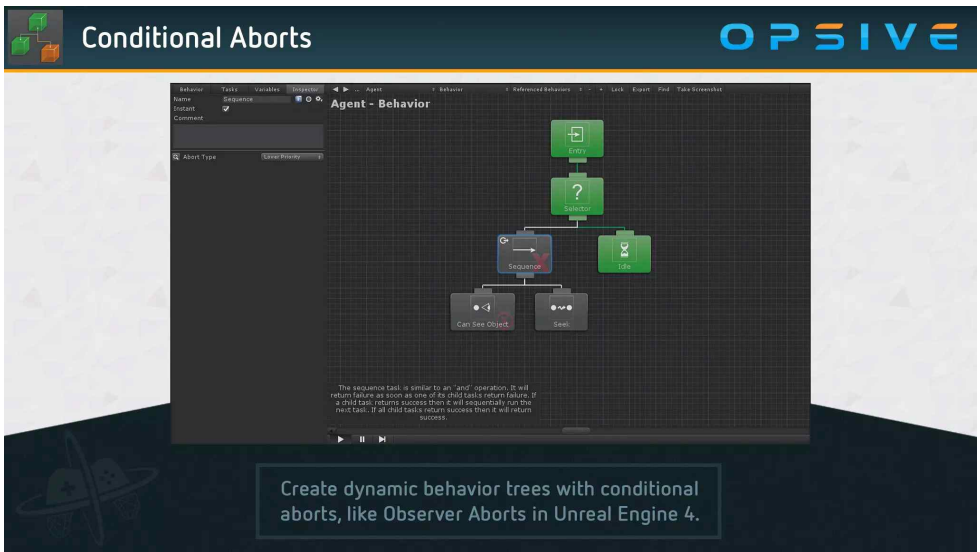


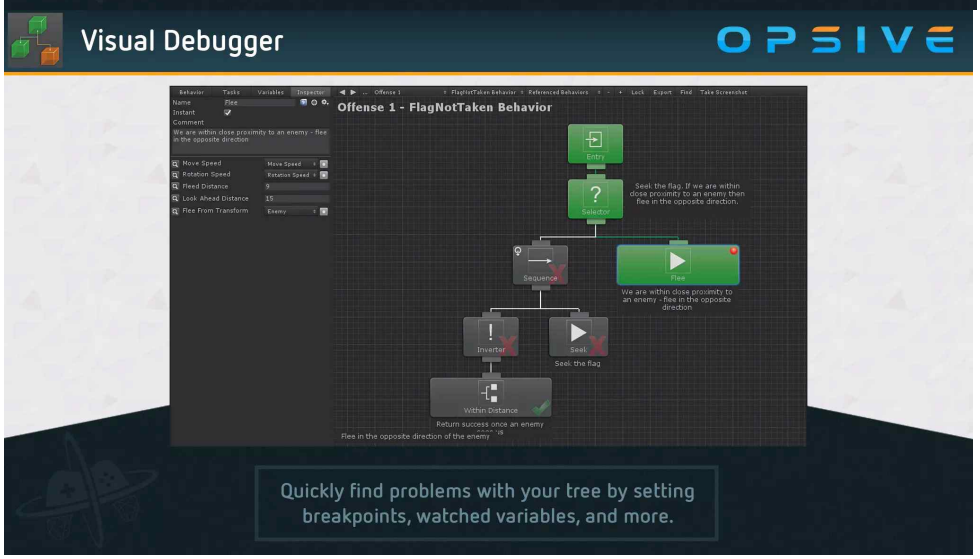
2. 적용 대상

- 적/아군 NPC AI가 복잡한 조건/행동을 가져야 할 때
- 상태 전환이 많아지고 FSM이 너무 복잡해졌을 때
- 플레이어 행동에 반응하는 동적 AI 설계가 필요할 때
- 시뮬레이션, 전략게임, RPG, 슈팅 게임에서 다양한 NPC 행동 패턴 필요할 때

3. 주요 기능

기능	설명
비주얼 행동 트리 에디터	노드 기반 편집기에서 행동(Tasks)과 조건(Condition)을 드래그 앤 드롭으로 연결
실시간 디버깅	실행 중 현재 어떤 노드가 실행되고 있는지 시각적으로 확인 가능
조건/행동(Task) 라이브러리 제공	Move, Wait, Patrol, Sequence, Selector 등 기본 노드 내장
커스텀 Task 제작 가능	C# 스크립트로 새로운 행동 노드 정의 가능
플러그인 확장	Movement Pack, Tactical Pack, Formations Pack 등 확장팩 존재
Parallel,Interrupt,Decorator 지원	복잡한 행동 조합 및 조건 제어 가능
Animator 통합	캐릭터 애니메이션과 직접 연동 가능





4. 기본 설치 및 세팅

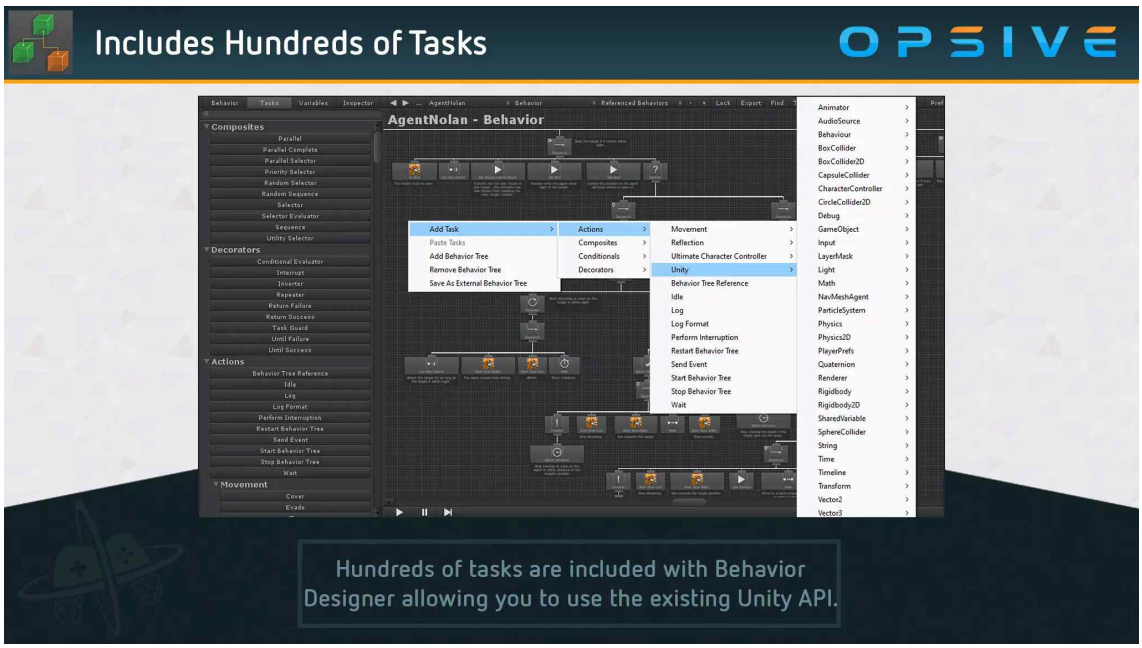
4.1 Unity Asset Store〉 Behavior Designer 구매 후 Import

4.2 NPC GameObject에 Behavior Tree 컴포넌트 추가

4.3 Behavior Designer 전용 Behavior Tree Editor 창 열기

- 메뉴: Tools〉 Behavior Designer〉 Editor

4.4 Task를 추가하고 연결하여 AI 로직 설계



5. 코드 예제

5.1 기본 Custom Task 만들기

```
using BehaviorDesigner.Runtime;
using BehaviorDesigner.Runtime.Tasks;
using UnityEngine;

public class MoveForward : Action
{
    public float speed = 2f;

    public override TaskStatus OnUpdate()
    {
        transform.position += transform.forward * speed * Time.deltaTime;
        return TaskStatus.Running; // 계속 실행
    }
}
```

5.2 조건 Task 만들기

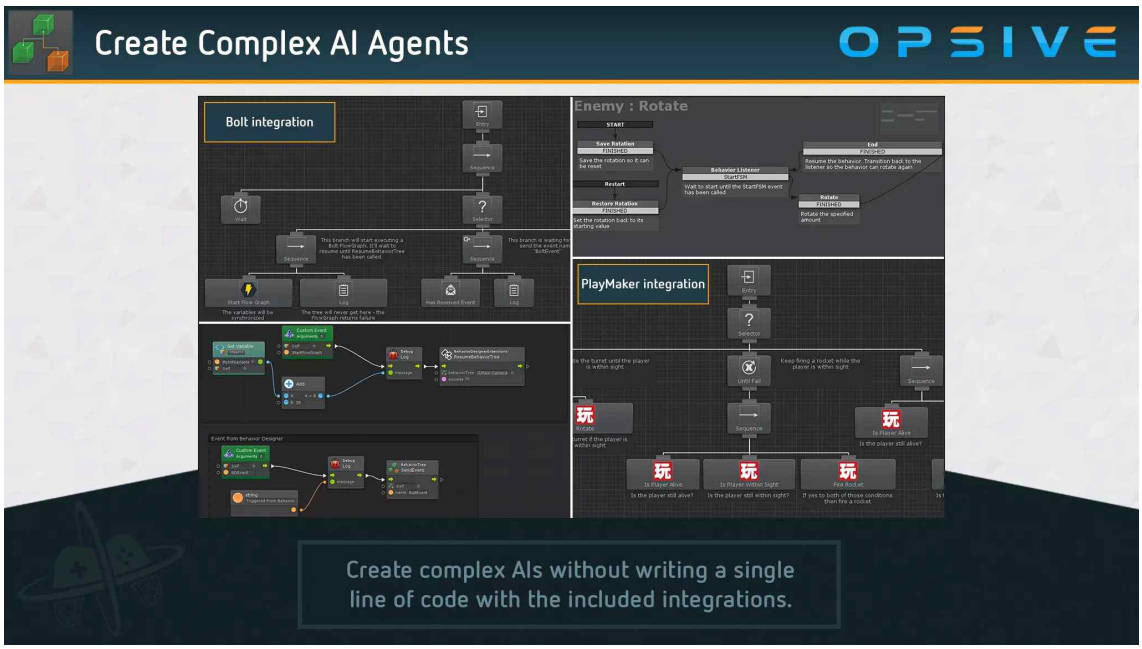
```
using BehaviorDesigner.Runtime;
using BehaviorDesigner.Runtime.Tasks;
using UnityEngine;

public class IsPlayerNear : Conditional
{
    public float detectRange = 5f;
    public GameObject player;

    public override TaskStatus OnUpdate()
    {
        float distance = Vector3.Distance(transform.position,
player.transform.position);
        return distance < detectRange ? TaskStatus.Success : TaskStatus.Failure;
    }
}
```

6. Behavior Tree 기본 구조

- Composite 노드
  - Sequence: 모든 자식이 성공해야 성공
  - Selector: 하나라도 성공하면 성공
- Decorator 노드
  - 조건부 실행, 반복, 제한자
- Task (Action / Conditional)
  - Action: 실제 행동 수행 (예: Move, Attack)
  - Conditional: 조건 검사

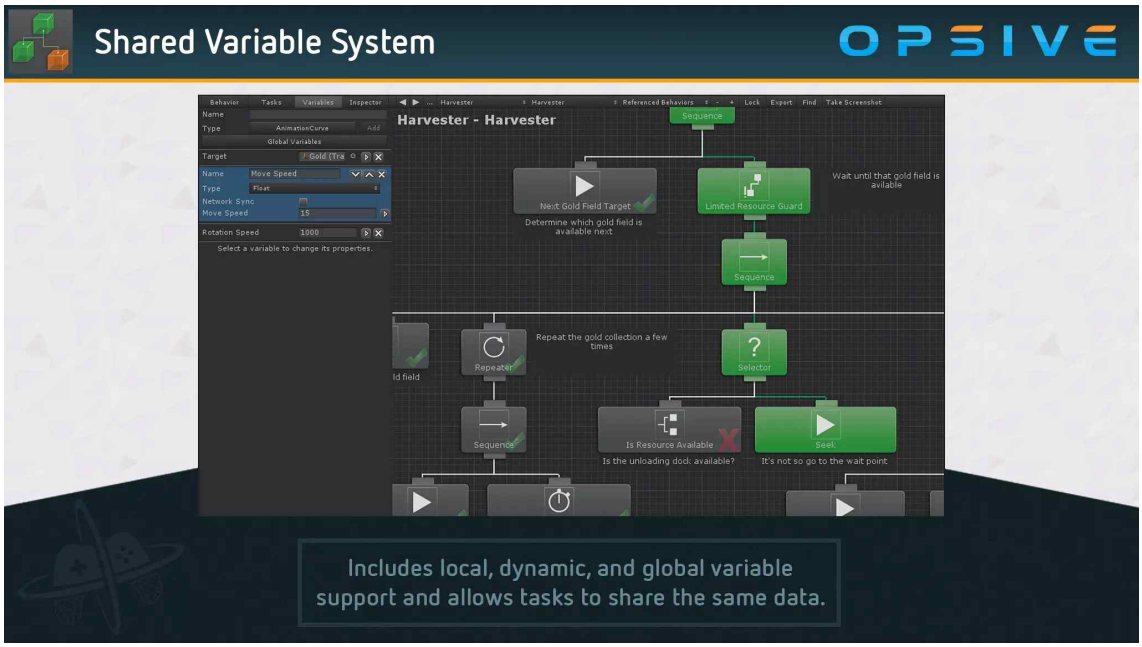
예:

```
Selector
 ├─ Sequence (조건: Player 발견 → 공격)
 │     ├─ IsPlayerNear?
 │     └─ Attack
 └─ Patrol
```



7. Unity 내 다른 시스템과의 연계

- NavMesh: Navigation Task와 함께 사용해 자동 경로 탐색/순찰 가능
- Animator: Attack, Run 등 애니메이션 상태와 동기화
- Inventory/Stat System: 조건 검사에 플레이어 체력, 아이템 여부 등 활용



8. Behavior Designer vs FSM (Finite State Machine)

항목	FSM	Behavior Tree(BD)
구조	상태(노드) + 전환(조건)	트리 구조(Composite + Task)
확장성	상태 수 증가 시 급격히 복잡	트리 구조라 복잡한 동작도 계층적 관리 가능
디버깅	전환 추적 어려움	실행 중 노드 상태 시각화 가능
표현력	단순 행동에는 적합	복합/조건적/계층적 행동에 적합

9. 실무 팁 & Best Practices

- 큰 트리 〉 여러 SubTree 분리: 복잡한 AI는 서브 트리로 모듈화
- 조건(Task) vs 데코레이터 구분: 조건은 재사용 Task로 만들어두면 유지보수 쉬움
- 디버깅 모드 적극 활용: 실행 시 색상으로 노드 상태 표시〉 AI 버그 찾기 쉬움
- 확장팩 활용: Movement Pack (NavMesh, AI 이동), Tactical Pack(전술 행동), Formations Pack(군집 행동) 적극 활용
- 성능 최적화: 매 프레임 모든 조건 검사 X 〉 Update Interval 조절

10. 요약 & 추천 사용 시나리오

- 액션 게임: 적 AI 패턴 (순찰〉 발견〉 추격〉 공격〉 도망)
- 전략 시뮬레이션: 유닛 행동 결정 (자원 채집, 전투, 후퇴)
- RPG NPC: 플레이어 상호작용, 퀘스트 반응
- 슈팅 게임: 탄막 패턴, 회피 행동

-Behavior Designer는 AI 로직의 시각화 + 확장성을 동시에 제공하기 때문에,
복잡한 NPC/적 행동을 설계하는 프로젝트에서는 사실상 산업 표준 수준의 선택지입니다.