

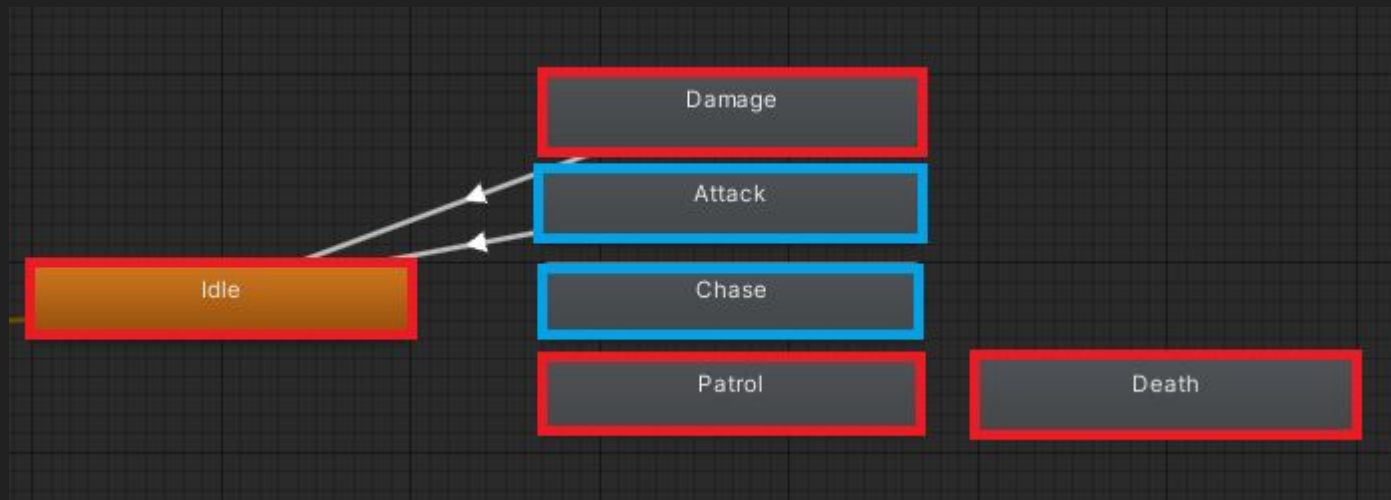
# 몬스터 상태머신 용례 및 사용

2023.11.08 손욱현

# 1. 에셋 준비

에셋 붉은색 사각형 이름맞추어 준비

만약 몬스터 범용 상태도 추가로 사용할 경우 하늘색 두개도 같이 준비

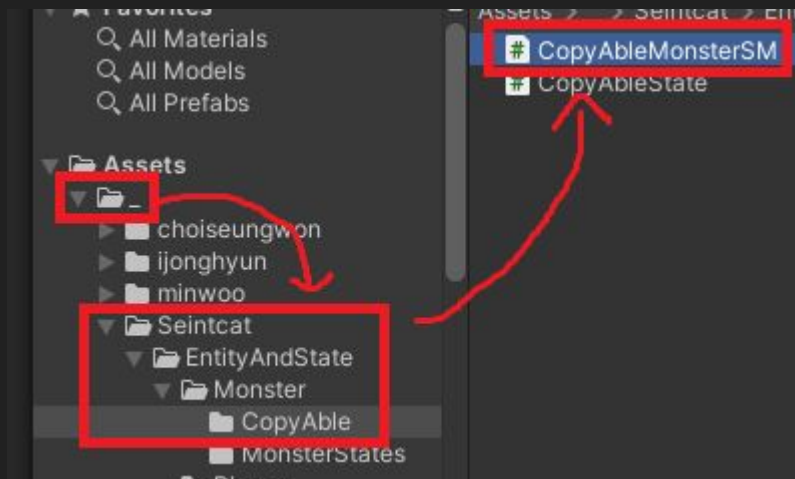


# 1. 에셋 준비

몬스터 상태머신 상속받아 사용

상속받은 컴포넌트에 해당 내용 복사

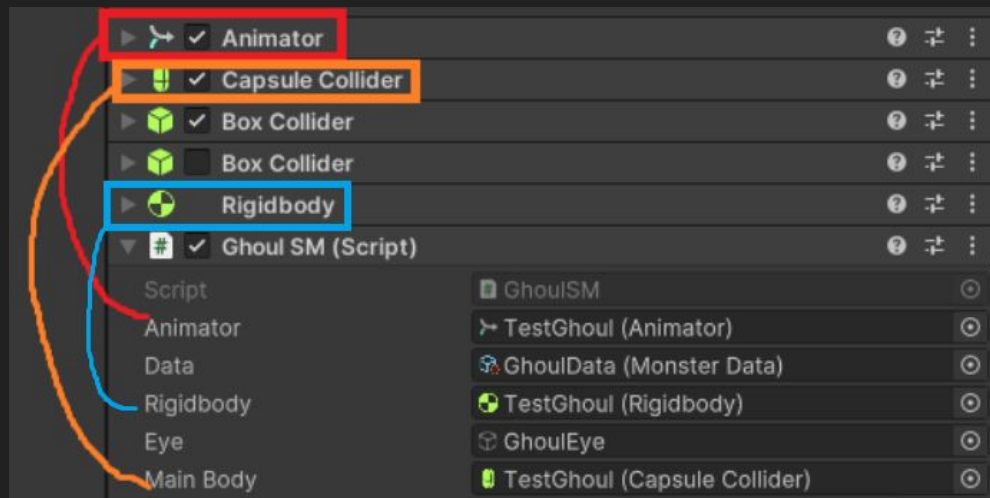
해당 파일(편집 절대 엄금)복사하여 사용



# 1. 에셋 준비

몬스터 최상단 오브젝트

1. 상태머신 컴포넌트 부착
2. 애니메이터, 콜라이더 배치
3. 리지드바디 배치

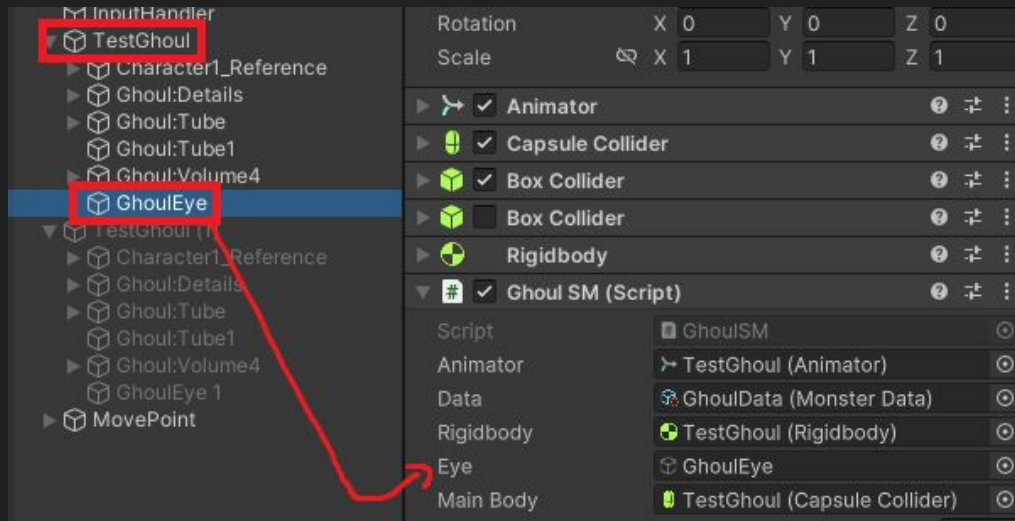


# 1. 에셋 준비

최상단 오브젝트 아래에 시야 오브젝트가 존재할 경우

몬스터 상태머신에 eye에 연결

시야 오브젝트 물리 레이어 Eye



# 1. 에셋 준비(선택 - 콜라이더 시야)

최상단 오브젝트 자식 오브젝트 등으로 시야 오브젝트 생성

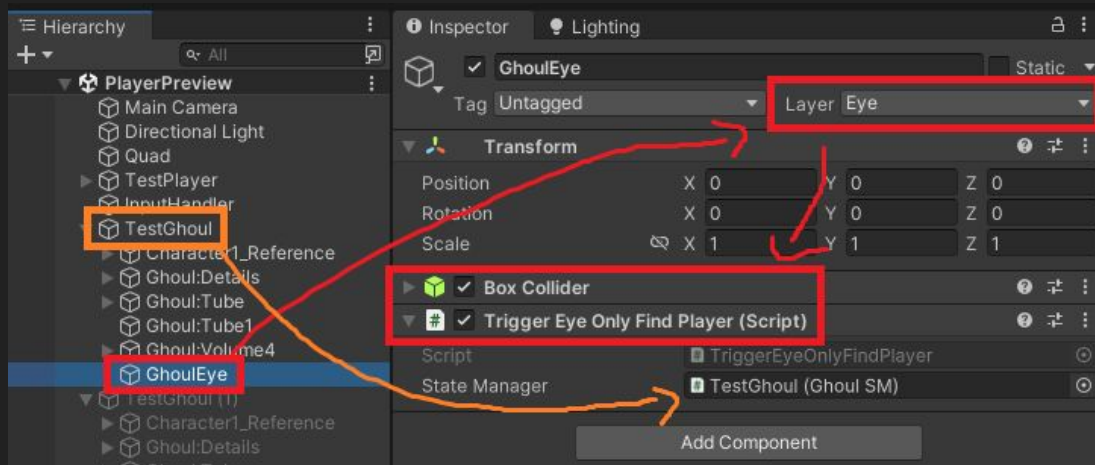
물리 레이어를 **Eye**로 설정

콜라이더 생성 > 트리거로

좌측 이름 컴포넌트 부착

컴포넌트에 상태머신 부착

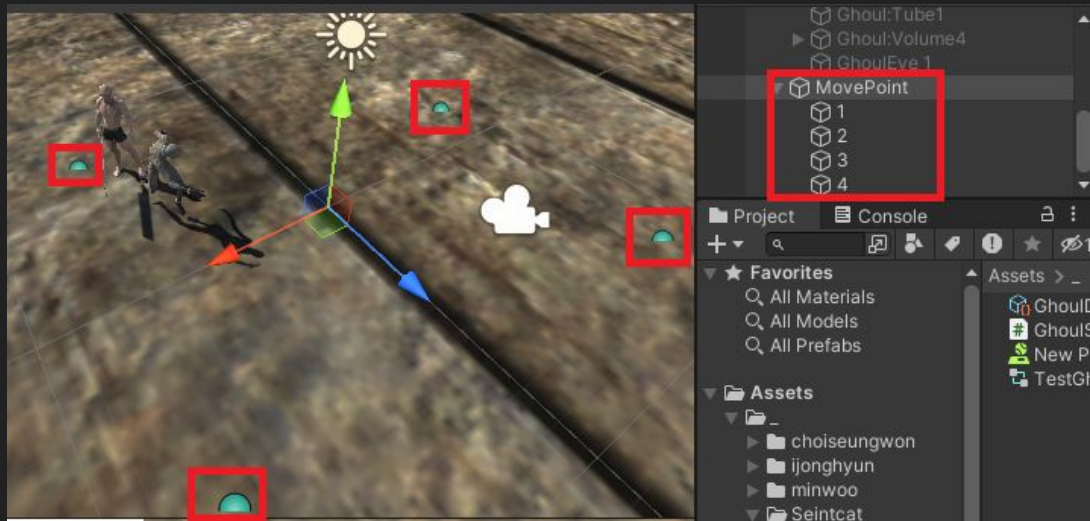
eye오브젝트는 콜라이더 하나만



# 1. 에셋 준비(선택 - 무브포인트)

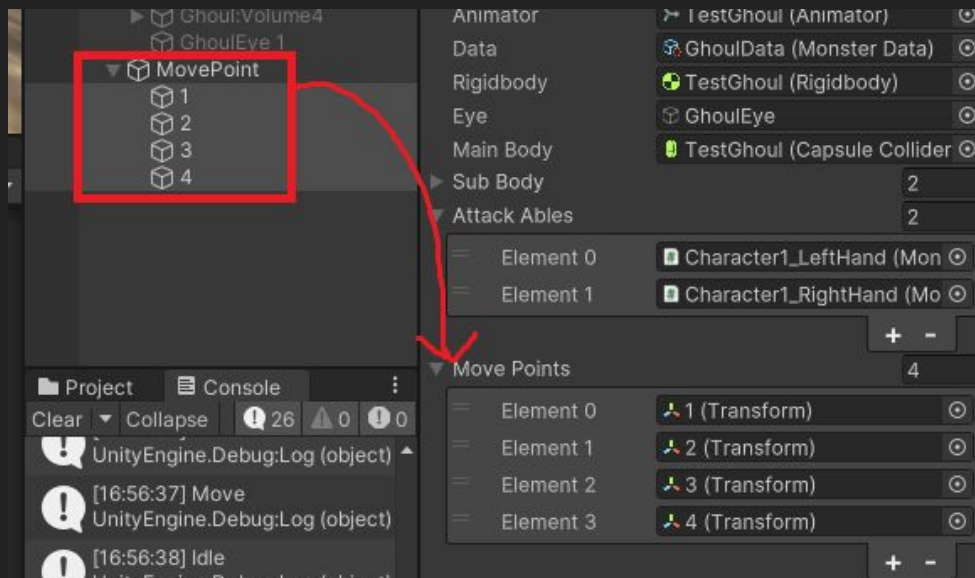
몬스터가 이동할 무브포인트 생성

실제로는 방에서 무브포인트 있고 연결하는 방안 고려



# 1. 에셋 준비(선택 - 무브포인트)

무브포인트 연결

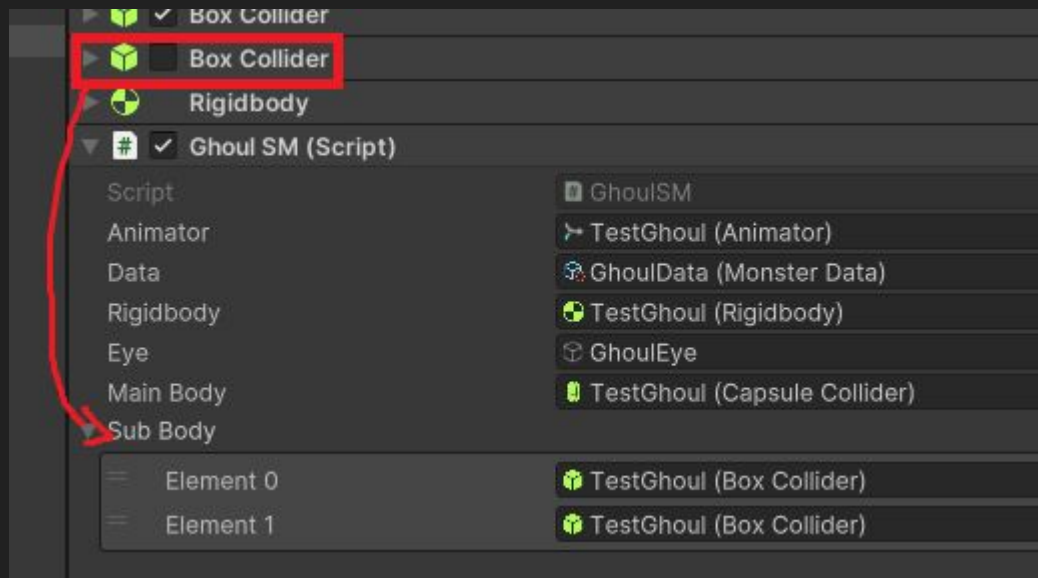




# 1. 에셋 준비(선택 - 서브 콜라이더)

모션따라 켜지고 꺼질 콜라이더 존재한다면 여기에 연결

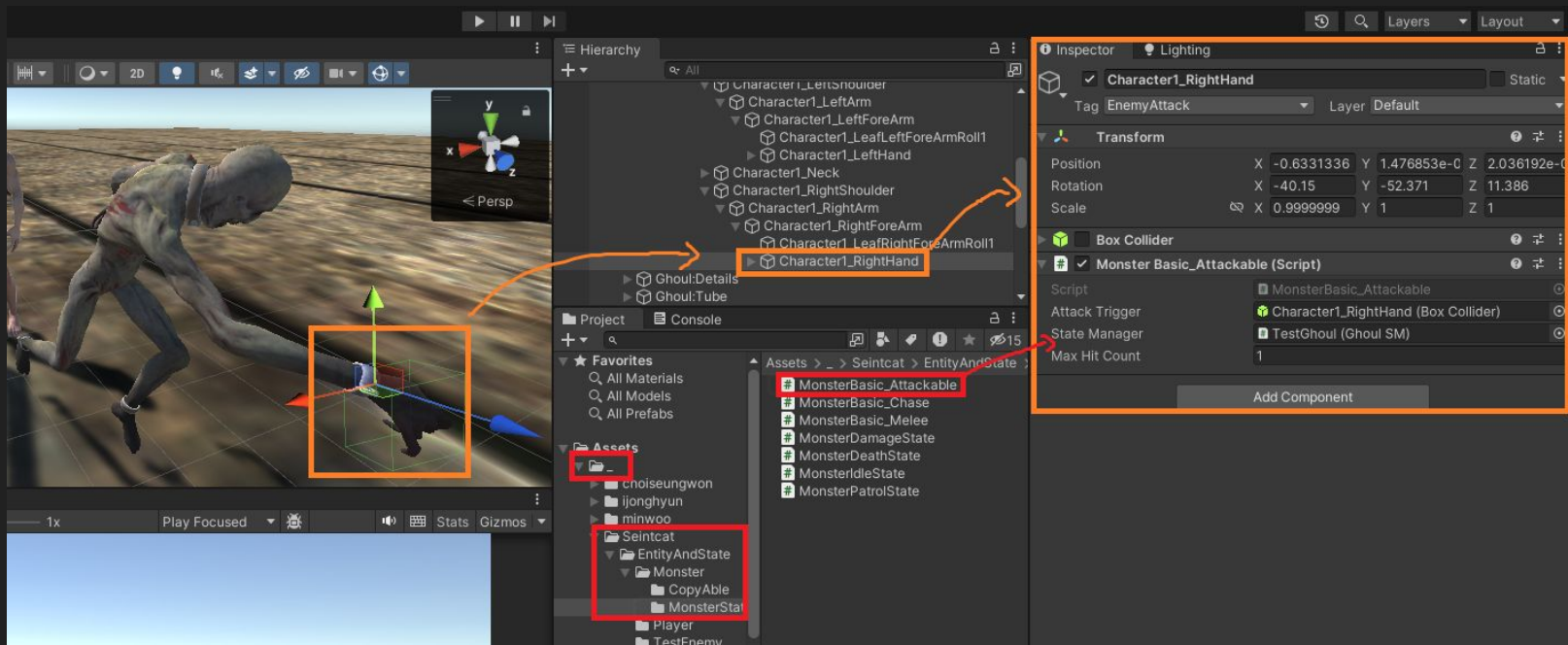
상태코드에서 제어 따로 작성



# 1. 에셋 준비(선택 - 근접공격시 판정 컴포넌트)

몬스터 근접공격 부위 선택 << 최상단 오브젝트의 자손이어야 함

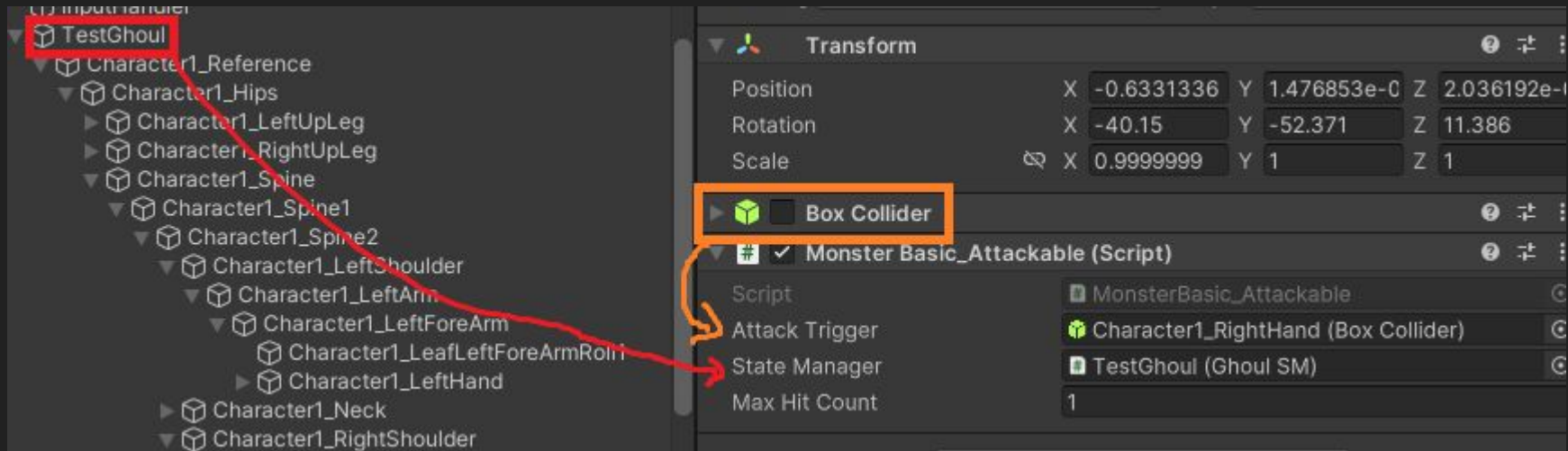
콜라이더 붙이고 애니메이션으로 확인



# 1. 에셋 준비(선택 - 근접공격시 판정 컴포넌트)

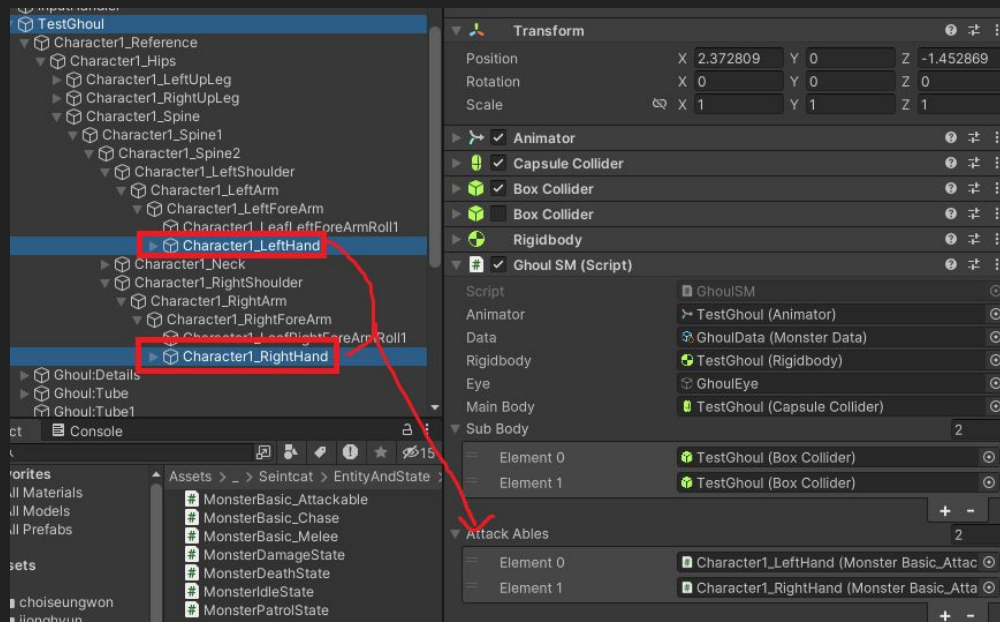
콜라이더와 상태머신을 공격 컴포넌트에 연결

hit카운트로 한 동작에 몇번 공격할지 결정



# 1. 에셋 준비(선택 - 근접공격시 판정 컴포넌트)

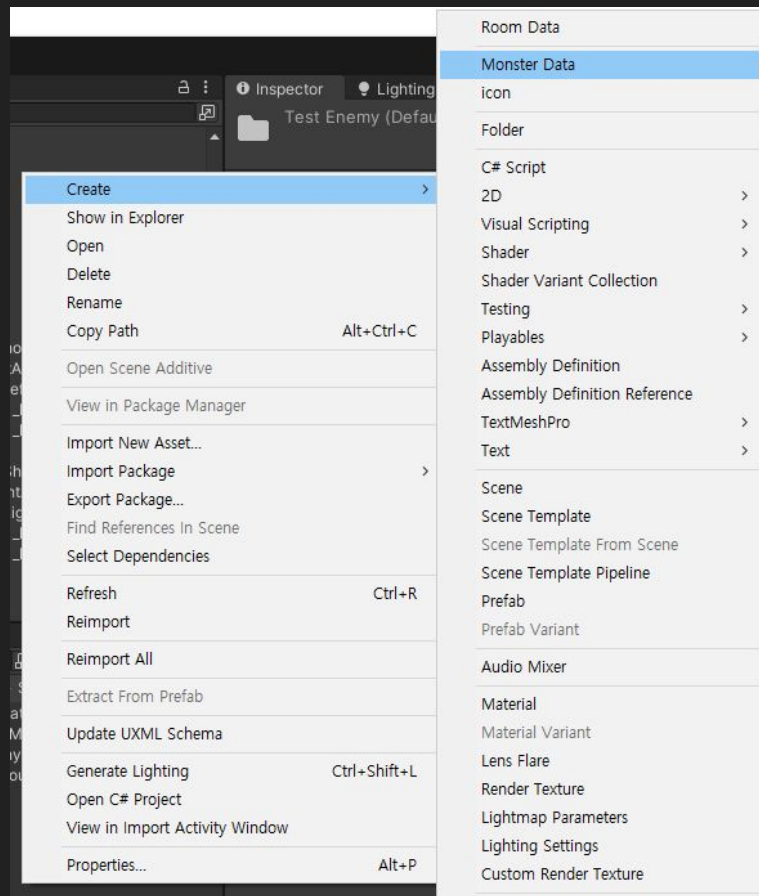
공격 컴포넌트 상태머신에 연결



# 1. 에셋 준비

우클릭 > 크레이트 > 몬스터 데이터

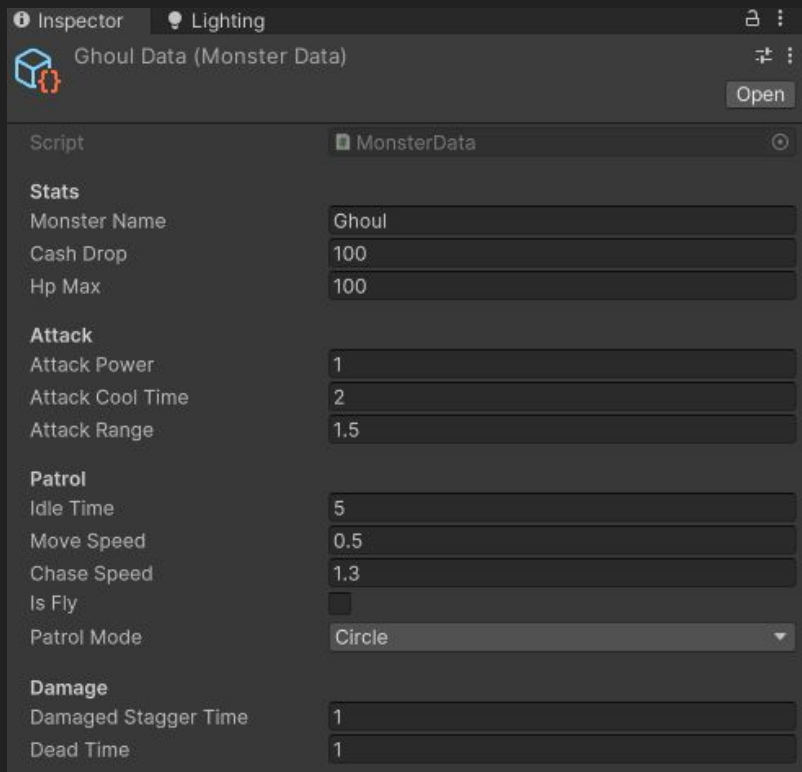
스크립터블 오브젝트 생성



# 1. 에셋 준비

## 몬스터 스탯 입력

- 이름
- 떨어지는 돈
- 공격력
- 공격 자체의 쿨타임
- 공격을 시작하는 사거리
- 이동간 대기시간
- 이동속도
- 추격시 속도
- 이동시 날아가는지
- 이동 무브포인트 순회 지점
- 데미지 받았을때 비틀거리는 시간
- 시체 사라지는 시간

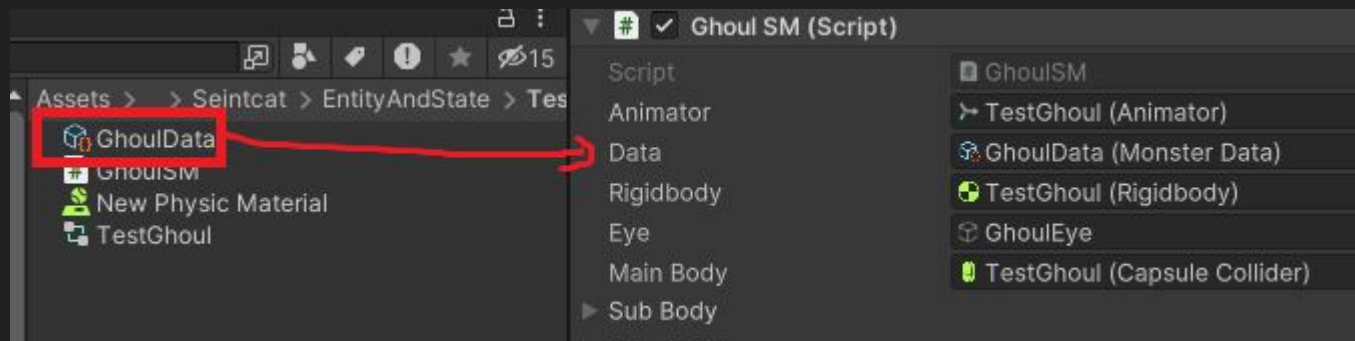


The screenshot shows the Unity Inspector window with the 'Ghoul Data (Monster Data)' script selected. The script is a C# script named 'MonsterData'. The configuration is as follows:

Category	Property	Value
Stats	Monster Name	Ghoul
	Cash Drop	100
	Hp Max	100
Attack	Attack Power	1
	Attack Cool Time	2
	Attack Range	1.5
Patrol	Idle Time	5
	Move Speed	0.5
	Chase Speed	1.3
	Is Fly	<input type="checkbox"/>
	Patrol Mode	Circle
Damage	Damaged Stagger Time	1
	Dead Time	1

# 1. 에셋 준비

상태머신에 몬스터 데이터 연결



## 2. 코드 작성(몬스터 상태머신)

에셋 준비 절차에 따라 상태머신 작성

상태머신이 기본적으로 가지고 있는 상태들

- 기본 대기 Idle
- 데미지 입음 Damage
- 죽음 Death
- 감시 순회 Patrol > 무브포인트 없음 동작X

```
3  /*
   * other code
   * ChangeState("name");
   * monsterIdleState.stateName = "Idle"
   * monsterDamageState.stateName = "Damage"
   * monsterDeathState.stateName = "Death"
   * monsterPatrolState.stateName = "Patrol"
   */
```



## 2. 코드 작성(몬스터 상태머신)

protected override List<State> monsterBattleStates

- 커스텀해서 만들 상태 코드 리스트

MakeState()

- monsterBattleStates 리스트에 커스텀 상태를 Add해야 함

base.MakeState()

- monsterBattleStates 리스트 안에 존재하는 상태를 allStates에 등록

참조 5개

```
public override void MakeState()
{
    // make custom states

    // make basic states
    base.MakeState();
}
```

## 2. 코드 작성(몬스터 상태머신)

```
public override void TargetChanged(List<GameObject> target)
```

- 게임오브젝트 `attackTarget`에 부여시키는 코드 작성

```
protected override void ReactDamage()
```

- 데미지 받고 난뒤 동작시

## 2. 코드 작성(몬스터 상태머신)

protected override void ResetStateMachine()

- 몬스터 초기화 이후 동작

## 2. 코드 작성(몬스터 상태머신)

기본 제공 상태들

MonsterBasic\_Chase

- 기본 추적 기능

MonsterBasic\_Melee

- 기본 근접공격 기능

다음과 같이 사용 가능

선택사항 애니메이터 준비 필요

```
// Don't edit this, Copy this!  
⚙ Unity 스크립트(자산 참조 2개) | 참조 0개  
public class GhoulSM : MonsterSM  
{  
    private List<State> _monsterBattleStates = new List<State>();  
    참조 3개  
    protected override List<State> monsterBattleStates => _monsterBattleStates;  
  
    ⚙ Unity 메시지 | 참조 0개  
    private void Awake()  
    {  
        ManagerStart();  
    }  
  
    참조 5개  
    public override void MakeState()  
    {  
        // make custom states  
        _monsterBattleStates.Add(new MonsterBasic_Chase());  
        _monsterBattleStates.Add(new MonsterBasic_Melee());  
    }  
}
```

## 2. 코드 작성(공격 컴포넌트)

모든 상태머신에 공격역할을 담당

이 컴포넌트가 공격하는게 아니고 다른 상태머신이 이 컴포넌트를 사용

## 2. 코드 작성(공격 컴포넌트)

공격 컴포넌트 => **AttackAble** 상속받아야 함, 구현할 땐 언더바 **O** 사용할 땐 언더바 **X**

**\_AttackStart()**

- 공격컴포넌트가 공격시작할때

**\_AttackStop()**

- 공격컴포넌트가 공격끝났을때

**\_GetDamage(GameObject obj)**

- 외부에서 공격컴포넌트를 참조할때 데미지값 **int**로 반환

## 2. 코드 작성(공격 컴포넌트)

Dictionary<GameObject, int> attackedObject

- 공격한 오브젝트, 공격한 횟수

Collider attackTrigger

- 공격 컴포넌트에 붙어있는 트리거

## 2. 코드 작성

시야 코드 `((ITargetCatch)stateManager).TargetChanged(new List<GameObject> { PlayerSM.playerObj });`

- `((ITargetCatch)stateManager).TargetChanged(new List<GameObject> { PlayerSM.playerObj });`

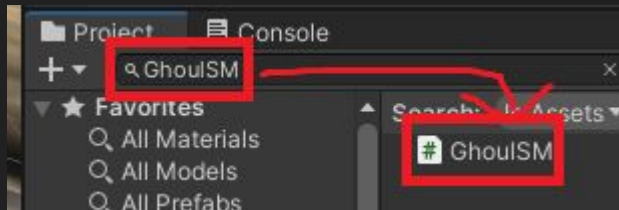
인터페이스 `ITargetCatch.TargetChanged(List<GameObject>)`

적이 공격하거나 하는 타겟 등이 여럿일수도 있음을 고려하여 `List<GameObject>`



### 3. 비교 대상

여기 기본 코드 있음



겁내 빠르게 얼렁뚱땅 설명함

안되는거 있음 질문하기