Unity를 이용한 2D 게임프로그래밍

Lecture 7
간단한 아케이드 게임의 전투와 게임 관리

강영민

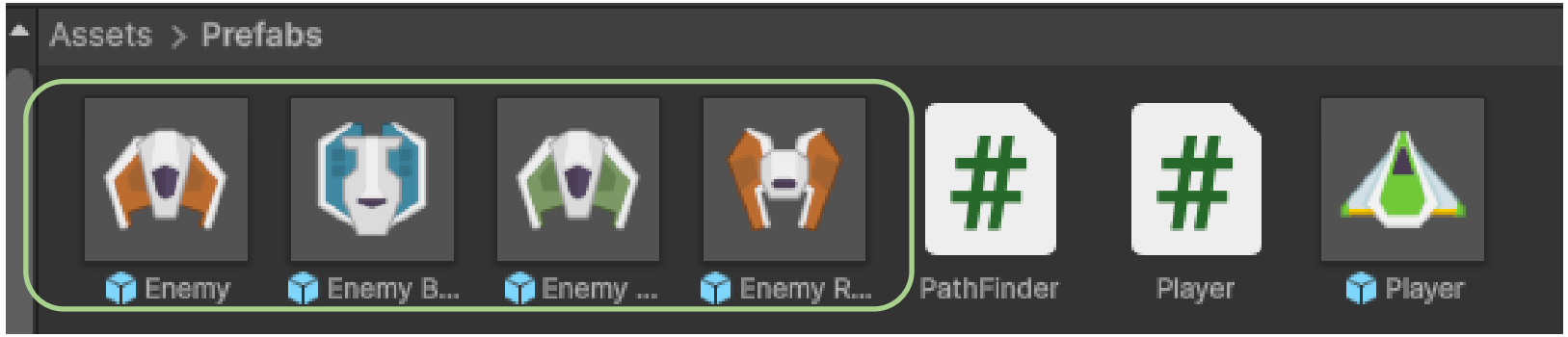
동명대학교 게임공학과

학습목표

- 입력에 의한 플레이어 제어 (new input system)
- Wave 설정 객체를 통한 적의 출현 (coroutines & loops)

캐릭터에 콜라이더 설정

- Prefab 폴더

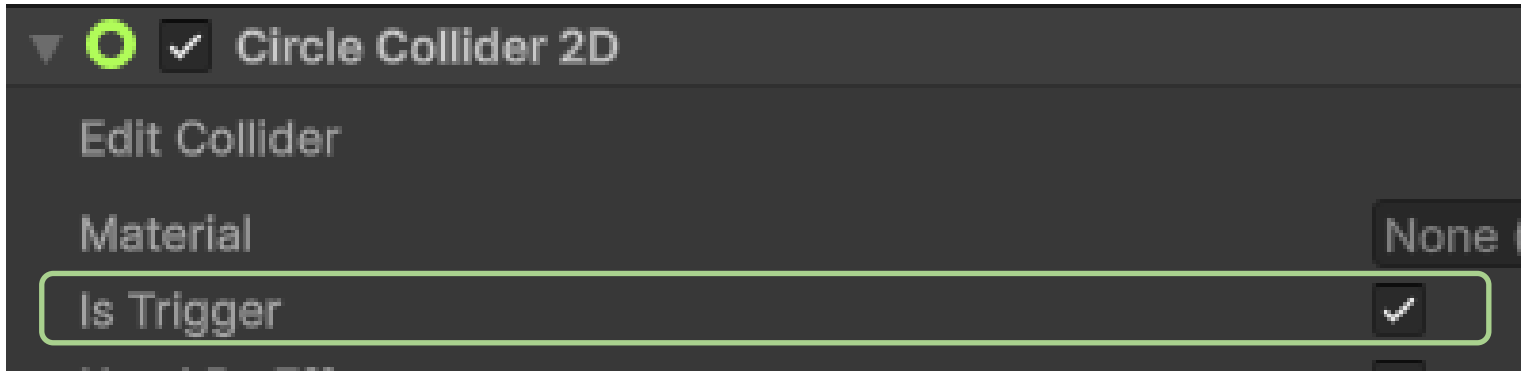


다수의 적 비행체 프리팹 생성한 상태

동시에 동일한 컴포넌트를 넣고 싶으면 다중 선택하여 컴포넌트 추가

캐릭터에 콜라이더 설정

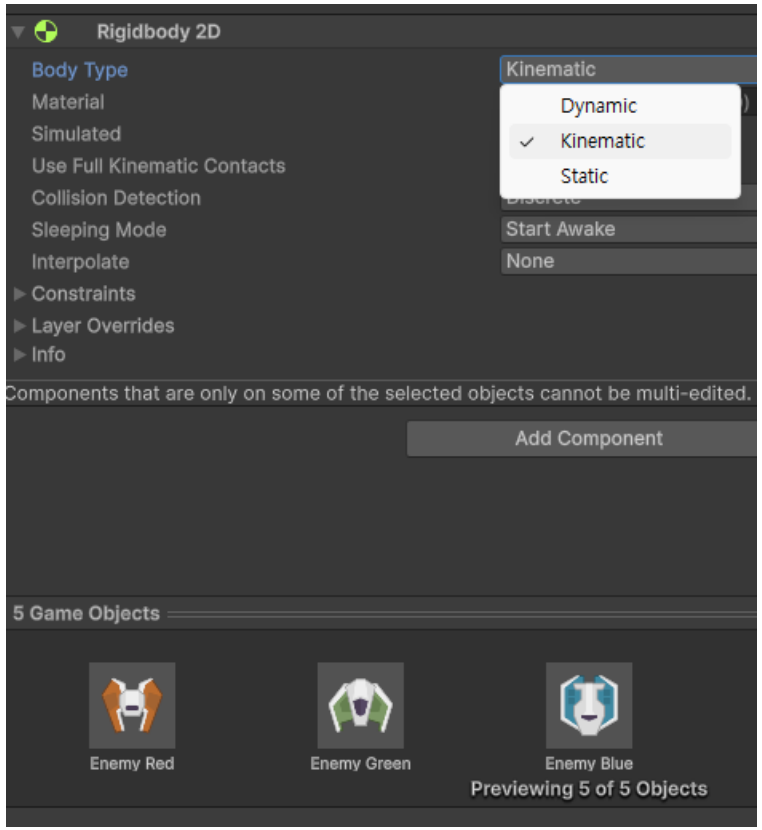
- Trigger로 설정

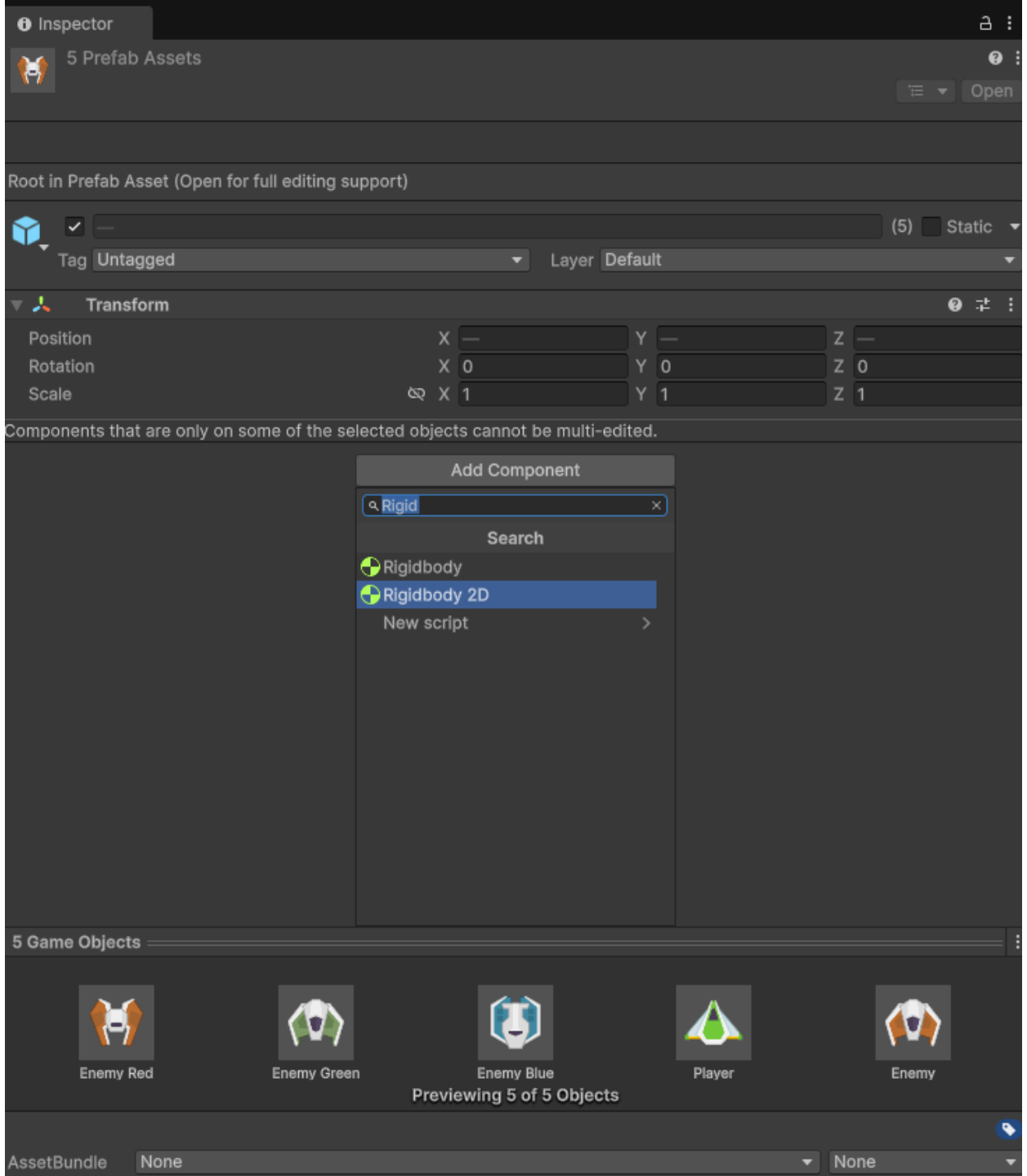


트리거 설정

RigidBody 2D 추가

- Body Type 변경
  - Dynamic ➔ Kinematic





Dynamic vs. Kinematic

- Dynamics (동역학)
  - 물리 시뮬레이션에 의해 힘이 계산되고 그 힘에 의해 물체가 가속/이동
    - 물체의 위치를 명시적으로 설정하지 않고 위치를 결정하는 조건을 지정
    - 물리 법칙에 따라 움직임 (자연스러운 물리 시뮬레이션)
    - 제어의 어려움 존재

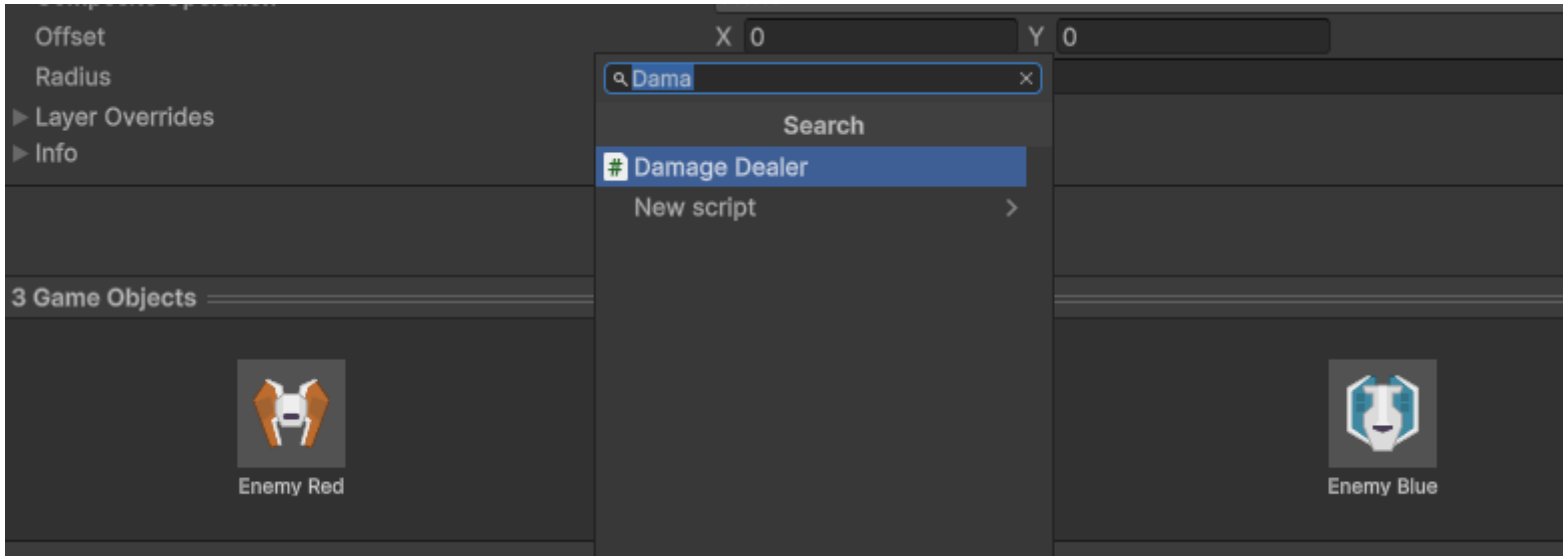
- Kinematics (운동학)
  - 물체의 움직임을 표현하는 곡선을 다루는 분야
    - 물체의 위치를 명시적으로 설정
    - 제어가 용이함

데미지를 처리하자 – Health.cs / DamageDealer.cs

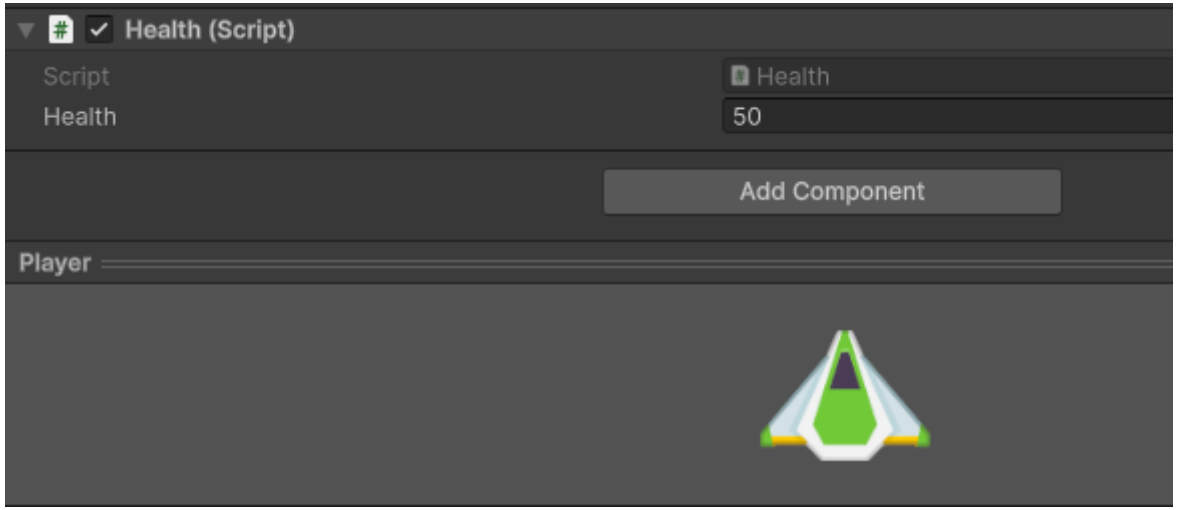
```cs
using UnityEngine;

Unity 스크립트 | 참조 0개
public class Health : MonoBehaviour
{
    [SerializeField] int health = 50;

    Unity 메시지 | 참조 0개
    void OnTriggerEnter2D(Collider2D other)
    {
        DamageDealer damageDealer = other.GetComponent<DamageDealer>();
        if (damageDealer != null )
        {
            TakeDamage(damageDealer.GetDamage( ));
            damageDealer.Hit( );
        }
    }

    참조 1개
    void TakeDamage(int damage)
    {
        health -= damage;
        if (health < 0)
        {
            Destroy(gameObject);
        }
    }
}
```

```cs
using UnityEngine;

Unity 스크립트 | 참조 2개
public class DamageDealer : MonoBehaviour
{
    [SerializeField] int damage = 10;

    참조 1개
    public int GetDamage( )
    {
        return damage;
    }

    참조 1개
    public void Hit( )
    {
        Destroy(gameObject);
    }
}
```

적기에 Damage Dealer 컴포넌트 추가



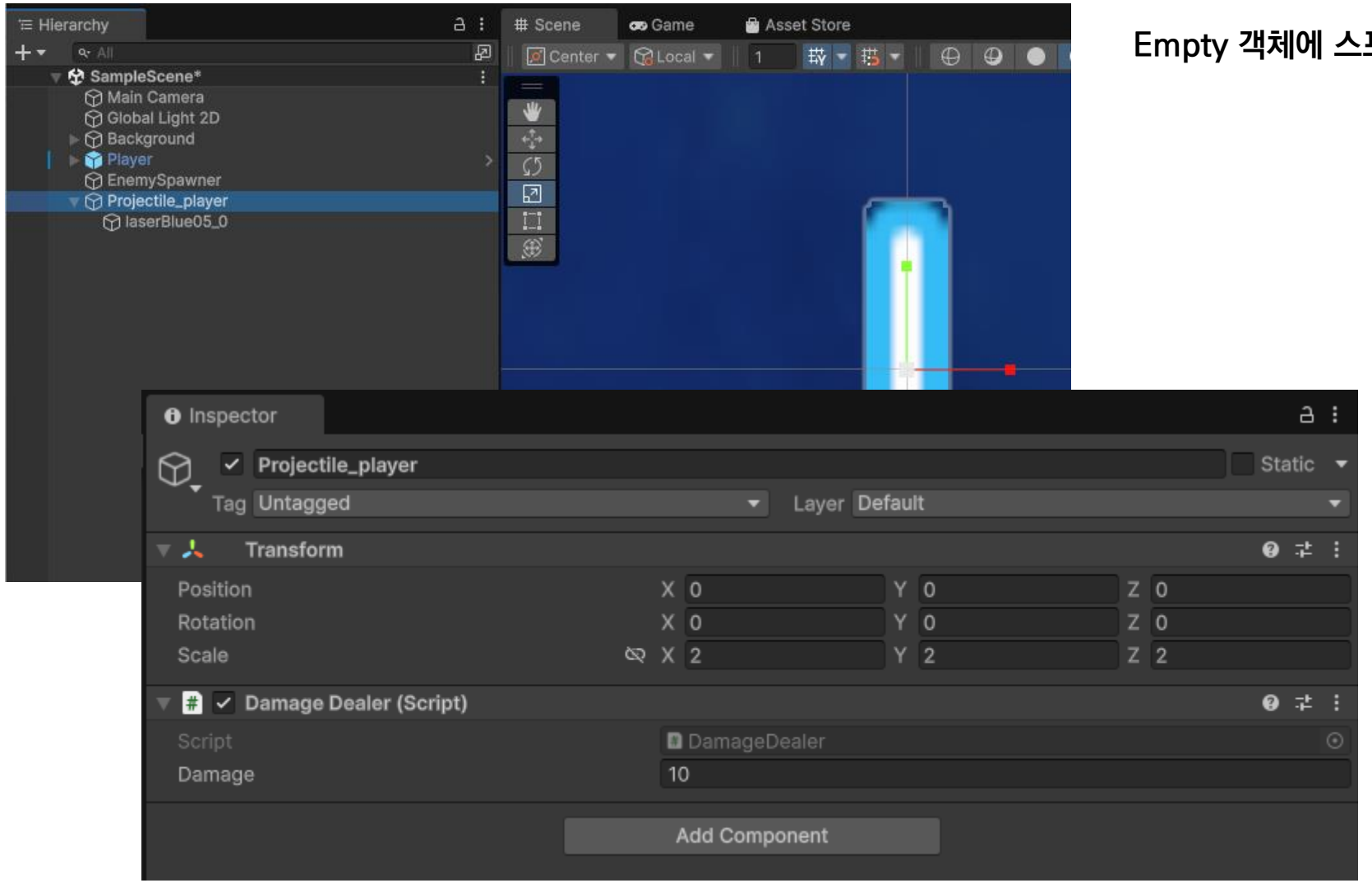
아군기에 Health 컴포넌트 추가



적기와 아군기의 충돌

- 적기가 가진 damage만큼 아군기 health 감소
  - 적기는 즉시 파괴
    - Health.cs 코드에 의해 호출되는 DamageDealer.cs의 Hit()에 의해
  - 아군기는 health가 0보다 작아지면 파괴

발사체 제작 – Player 발사체

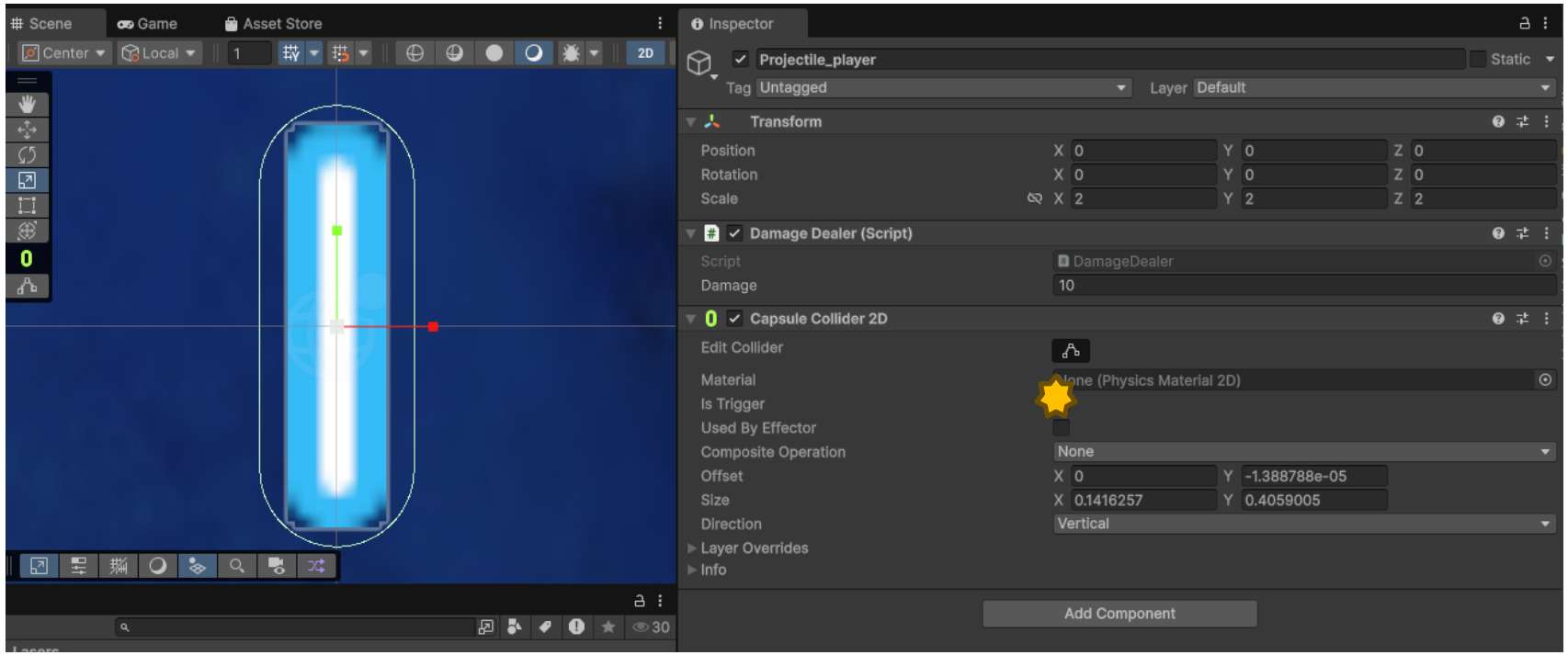


Empty 객체에 스프라이트 설치

Damage Dealer 컴포넌트 추가

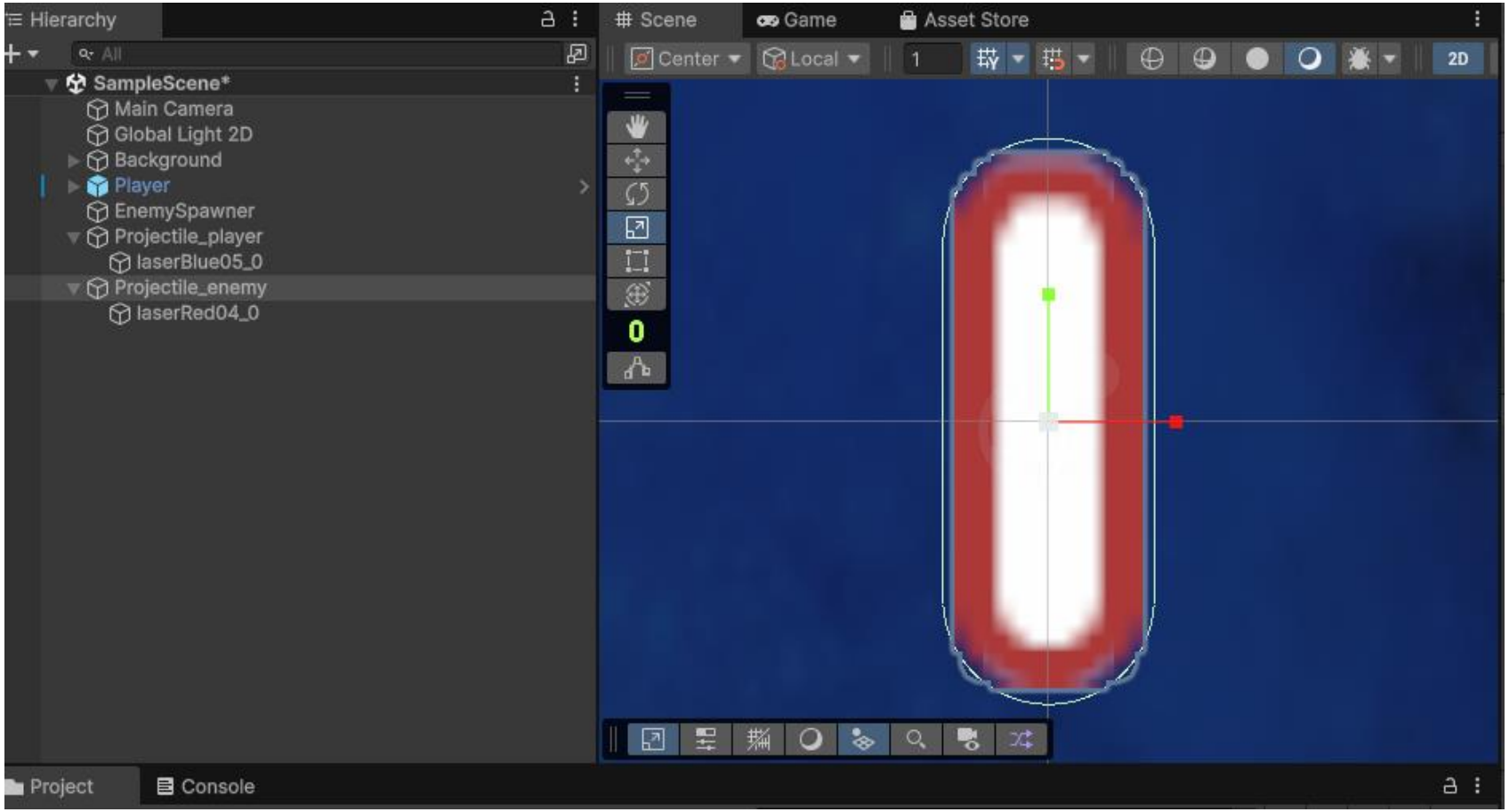
발사체 제작 – Player 발사체

발사체에 콜라이더 설정 – 트리거로 설정

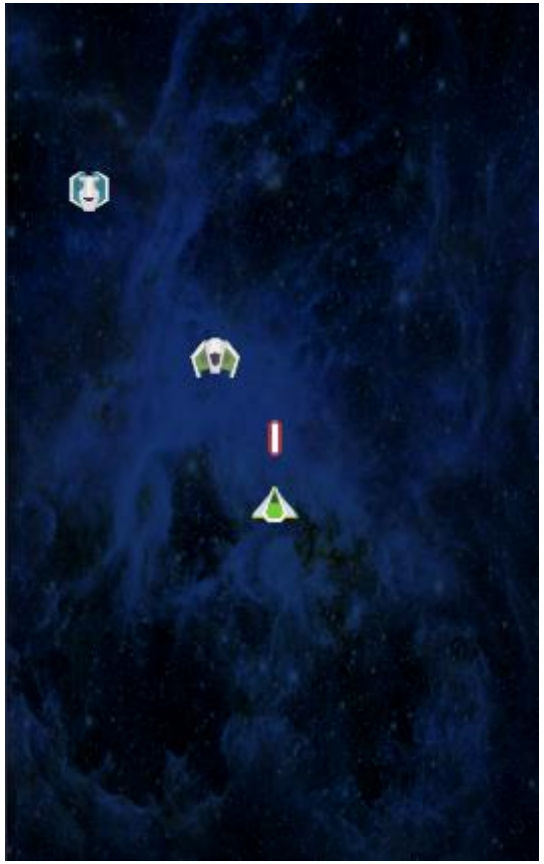


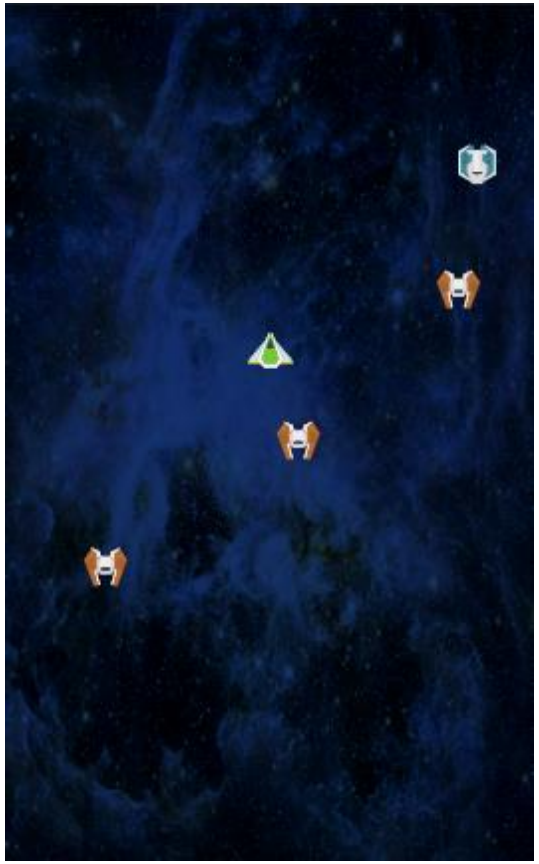
발사체 제작 – Enemy 발사체

플레이어 발사체를 복사하여 제작 – 스프라이트 변경



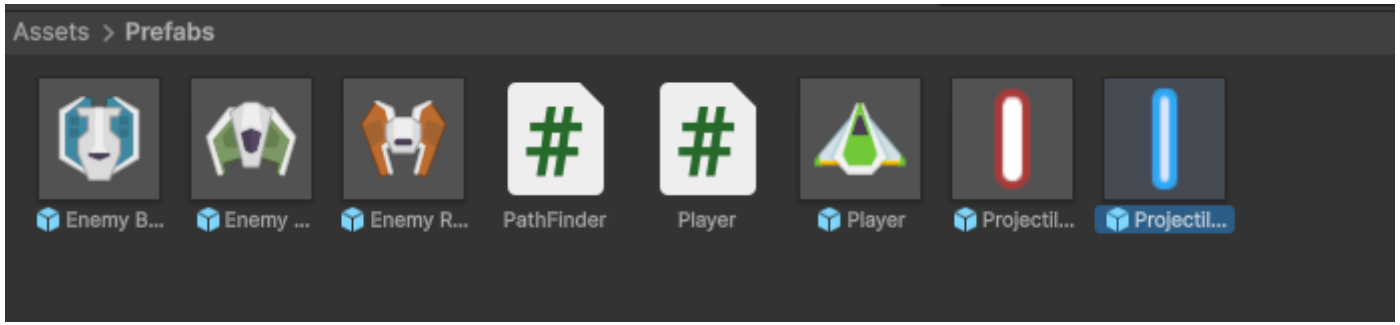
문제점: 플레이어 발사체가 플레이어와 반응





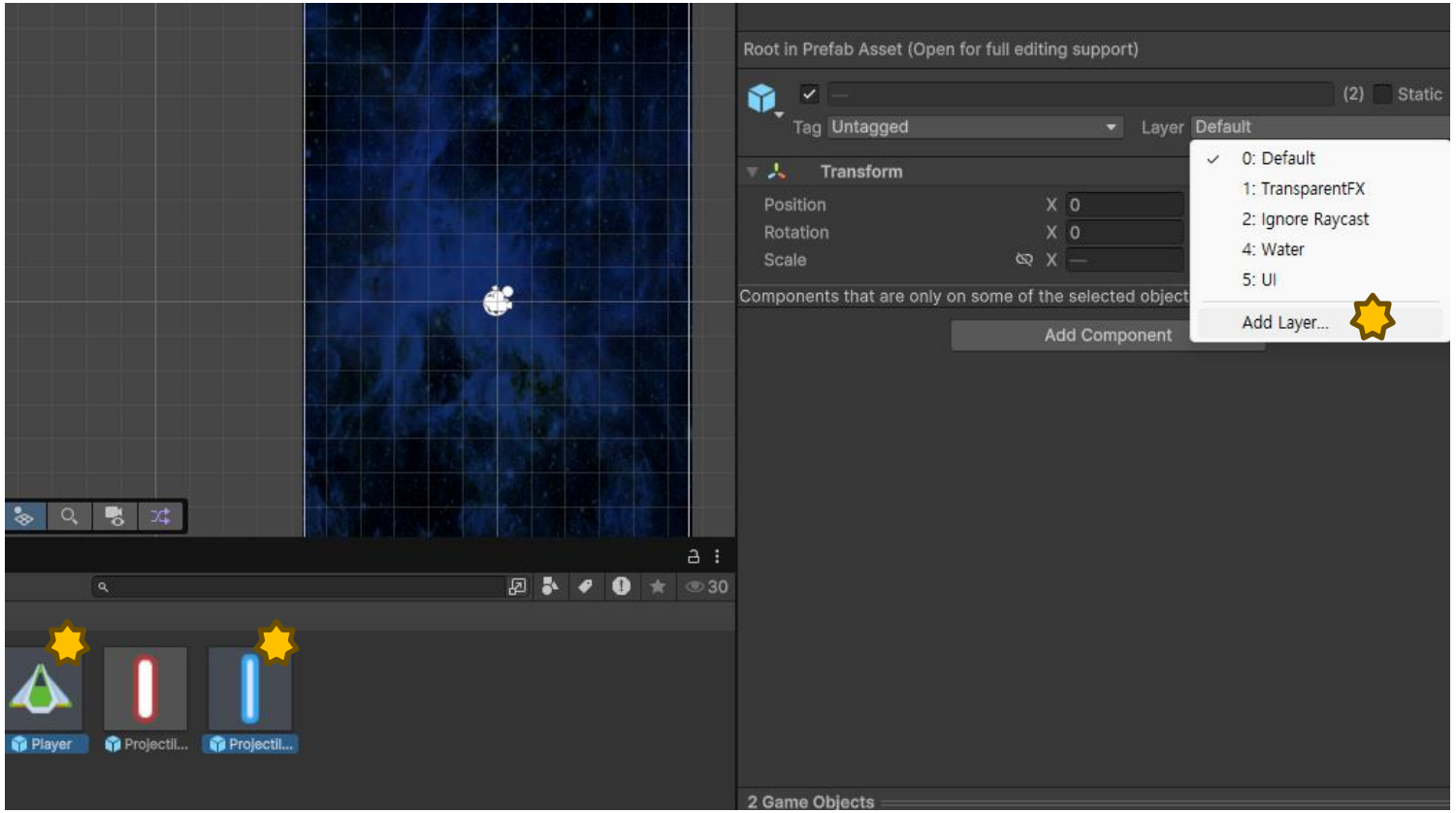
발사체를 프리팹으로

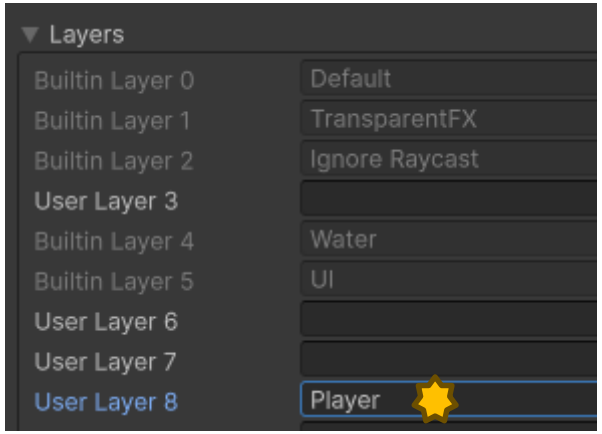
- 프리팹 폴더에 끌어다 놓고 Hierarchy에서 삭제



Layer

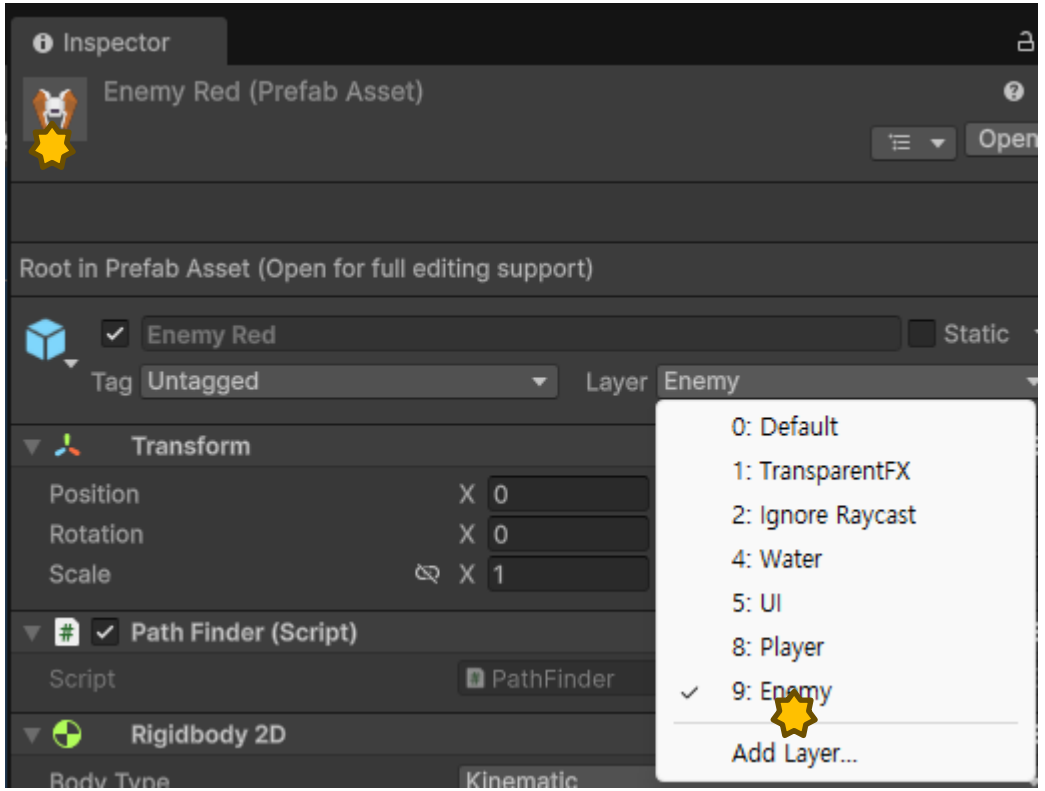
- 플레이어와 플레이어 발사체의 레이어 설정





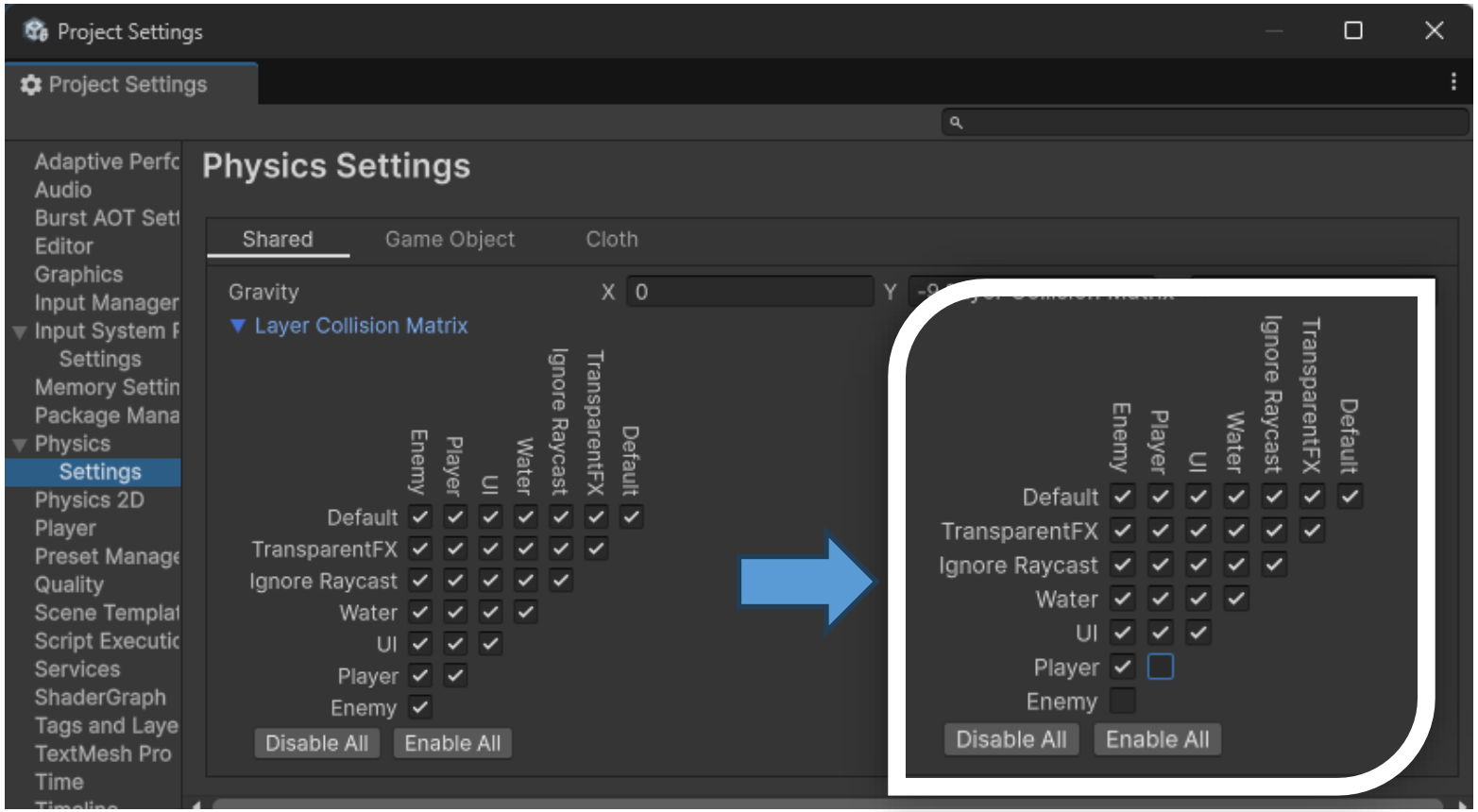
Layer

- 적과 적 발사체도 레이어 설정
  - Layer 추가 이후에는 지정을 해야 함



Layer Collision Matrix

- Edit → Project Settings



플레이어 슈팅

- Player.cs 코드 수정 (실제 슈팅 동작은 Shooter.cs)

```
Shooter shooter;

Unity 메시지 | 참조 0개
void Awake()
{
    shooter = GetComponent<Shooter>();
}

참조 0개
public void OnAttack(InputValue value)
{
    if (shooter != null)
    {
        shooter.isFiring = value.isPressed;
    }
}
```

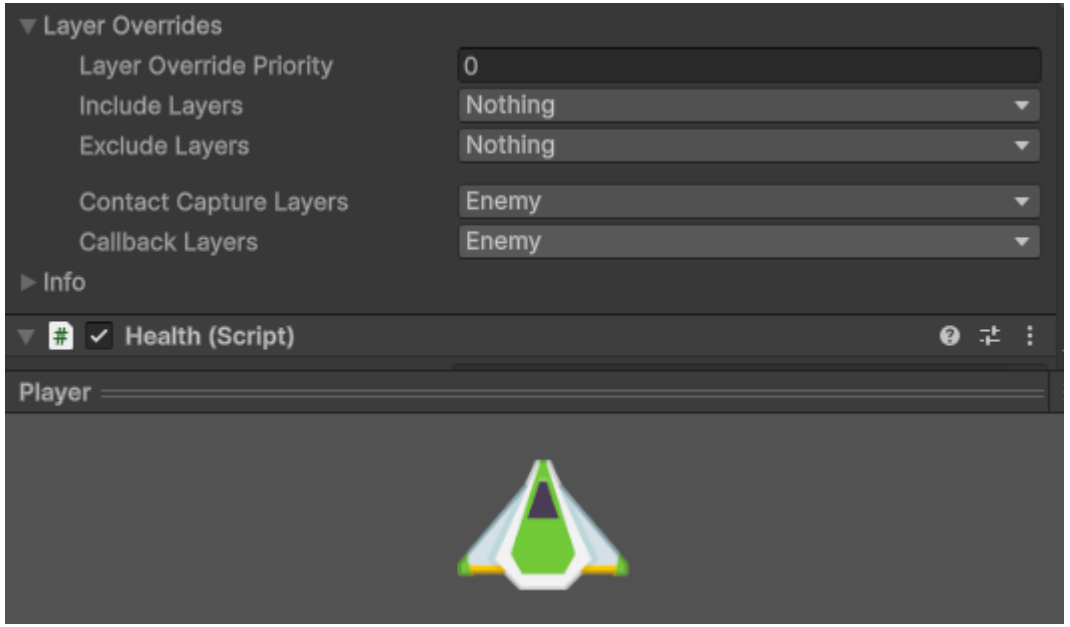
Shooter.cs

```
using System.Collections;
using UnityEngine;

Unity 스크립트(자산 참조 1개)|참조 2개
public class Shooter : MonoBehaviour
{

    [SerializeField] GameObject projectilePrefab;
    [SerializeField] float projectileSpeed = 10f;
    [SerializeField] float projectileLifeTime = 5f;
    [SerializeField] float firingRate = 0.2f;

    public bool isFiring;
    Coroutine firingRoutine;

    Unity 메시지|참조 0개
    void Update()
    {
        Fire();
    }

    참조 1개
    void Fire()
    {
        if (isFiring && firingRoutine == null)
        {
            firingRoutine = StartCoroutine(FireContinuously());
        }
        else if (!isFiring && firingRoutine != null)
        {
            StopCoroutine(firingRoutine);
            firingRoutine = null;
        }

    }
```

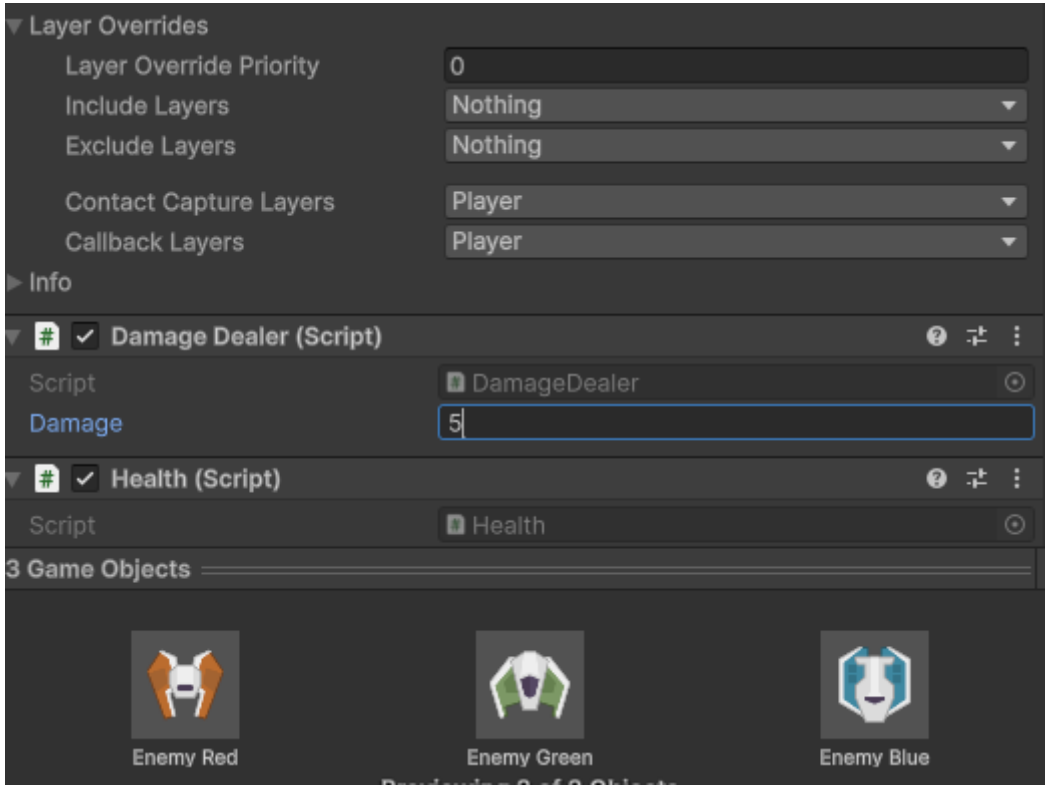
Shooter.cs

```
    IEnumerator FireContinuously()
    {
        while (true)
        {

            GameObject missile = Instantiate(projectilePrefab,
                transform.position, Quaternion.identity);

            Rigidbody2D rb = missile.GetComponent<Rigidbody2D>();
            if (rb != null)
            {
                rb.linearVelocity = transform.up * projectileSpeed;
            }
            Destroy(missile, projectileLifeTime);
            yield return new WaitForSeconds(firingRate);
        }

    }
}
```

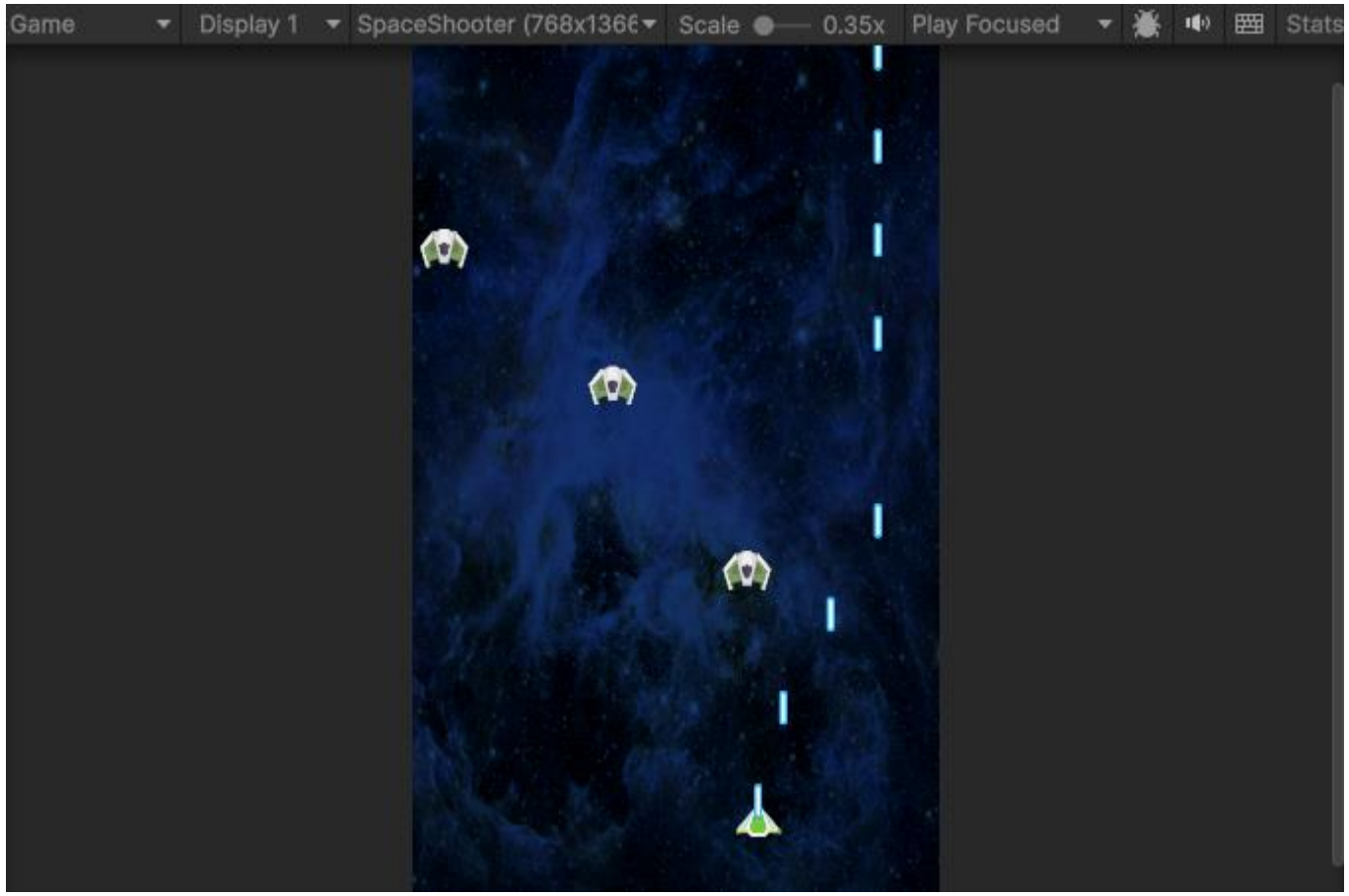
Layer Overrides



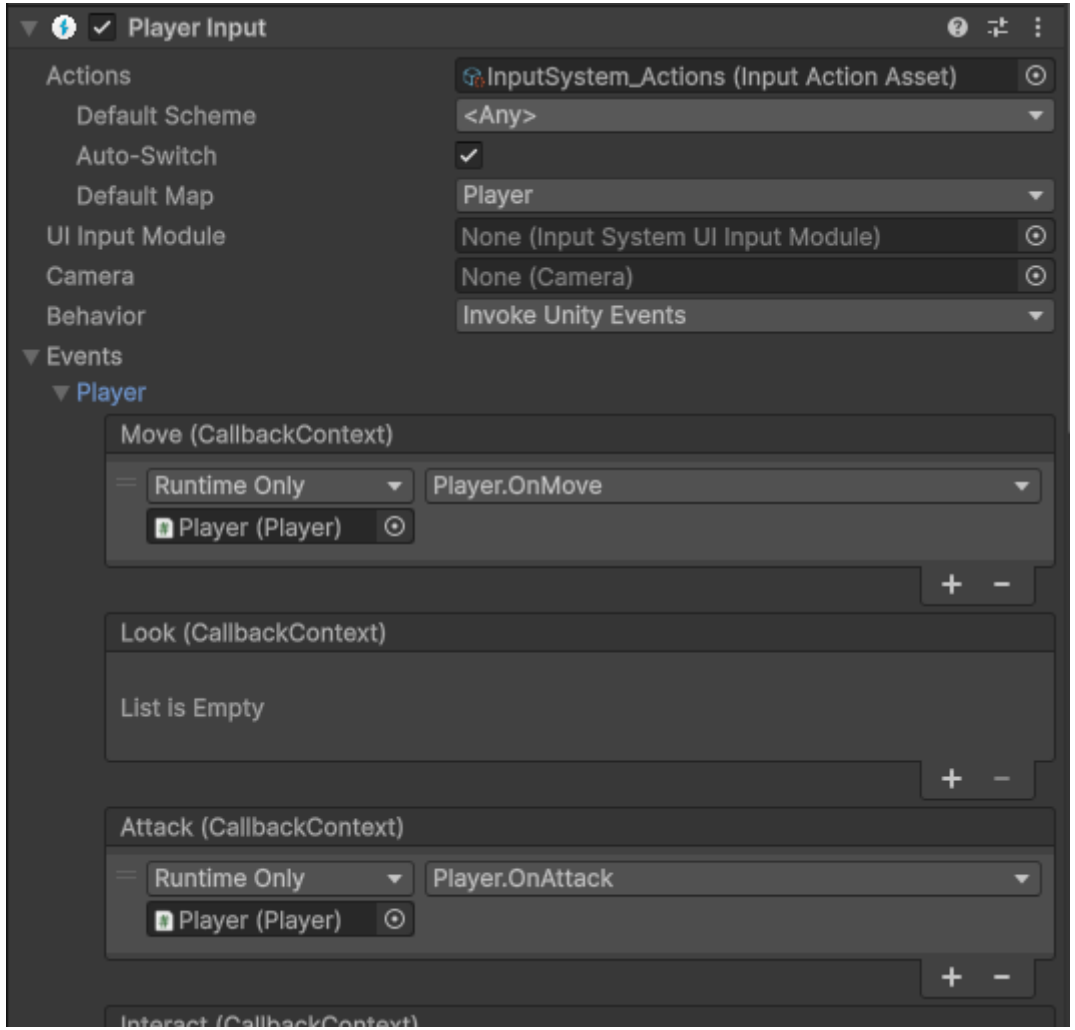
Enemies – Layer Overrides & Health Component



미사일을 발사할 수 있다 – 문제점은? 발사 중지



Invoke Unity Events를 사용하여 제어해 보자



Player.cs

```
using UnityEngine;
using UnityEngine.InputSystem;
using static UnityEngine.Rendering.DebugUI;
Unity 스크립트(자산 참조 2개)|참조 0개
public class Player : MonoBehaviour
{
    Vector2 moveInput = new Vector2(0f, 0f);
    private float speed = 5.0f;

    Vector2 minBounds;
    Vector2 maxBounds;



    // Start is called once before the first execution of Update after the MonoBehaviour is created
    Unity 메시지|참조 0개
    void Start()
    {
        Camera cam = Camera.main;
        minBounds = cam.ViewportToWorldPoint(new Vector2(0, 0));
        maxBounds = cam.ViewportToWorldPoint(new Vector2(1, 1));
    }

    // Update is called once per frame
    Unity 메시지|참조 0개
    void Update()
    {
        Vector2 moveDelta = moveInput * speed * Time.deltaTime;
        Debug.Log("rawInput in Update" + moveInput.ToString());
        Vector2 newPosition = (Vector2)transform.position + moveDelta;

        newPosition.x = Mathf.Clamp(newPosition.x, minBounds.x, maxBounds.x);
        newPosition.y = Mathf.Clamp(newPosition.y, minBounds.y, maxBounds.y);
        transform.position = newPosition;
    }
```

OnMove에 의해 실행될 내용 담기

```
public void OnMove(InputAction.CallbackContext context)
{

    if (context.performed)
    {
        Debug.Log("Performed");
        moveInput = context.ReadValue<Vector2>();
        Debug.Log(moveInput.ToString());
    }
    else if (context.canceled)
    {
        Debug.Log("Cancelled");
        //moveInput = new Vector2(0.0f, 0.0f);
        Debug.Log(moveInput.ToString());
    }
}
```

발사 제어

```
Unity 메시지 | 참조 0개
void Awake()
{
    shooter = GetComponent<Shooter>();
}

참조 0개
public void OnAttack(InputAction.CallbackContext context)
{
    if (context.performed)
    {
        Debug.Log("Attack started");
        shooter.isFiring = true;
    }
    else if (context.canceled)
    {
        Debug.Log("Attack stopped");
        shooter.isFiring = false;
    }
}
```

적도 발사하게 해 보라