

Unity 를 이용한 2D 게임프로그래밍

Lecture 3 Snowboard 게임 만들기

강영민

동명대학교 게임공학과

학습목표

- 유니티로 간단한 게임을 만들어 본다.
- 입력 시스템은 왜 필요한지 이해한다.
- 입력 시스템을 이용하여 캐릭터를 제어해 본다.
- 콜라이더와 트리거를 이해한다.
- 물리적 동작이 일어나게 한다.
- 게임 배경을 만들어 본다.
- 게임 캐릭터를 따라 카메라가 이동하게 한다.

원작 강의 – Rick Davidson

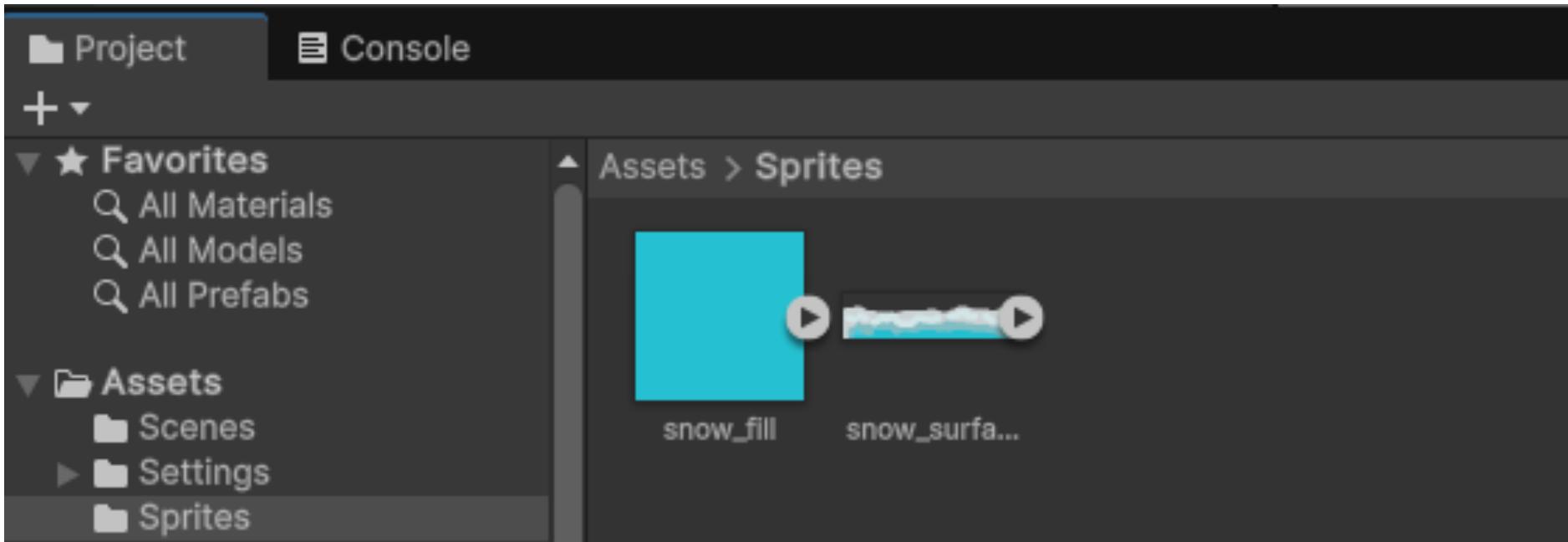


<https://www.udemy.com/course/best-c-unity-2d/learn/lecture/29503722#overview>

원작자: Rick Davidson

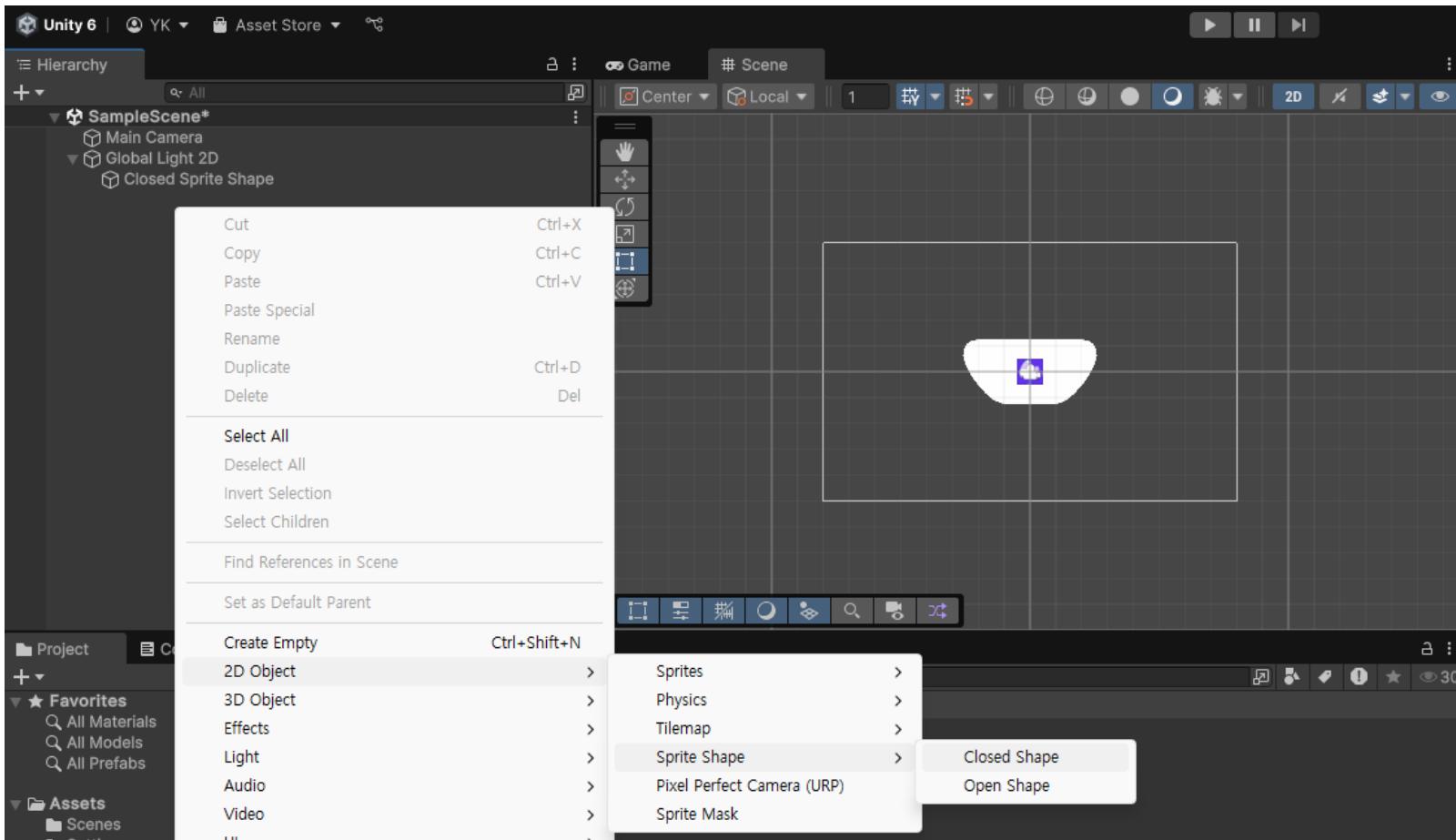
프로젝트를 생성하고 눈 애셋을 넣자

- 공유된 리소스 중에 snow_fill과 snow_surface를 프로젝트에 추가

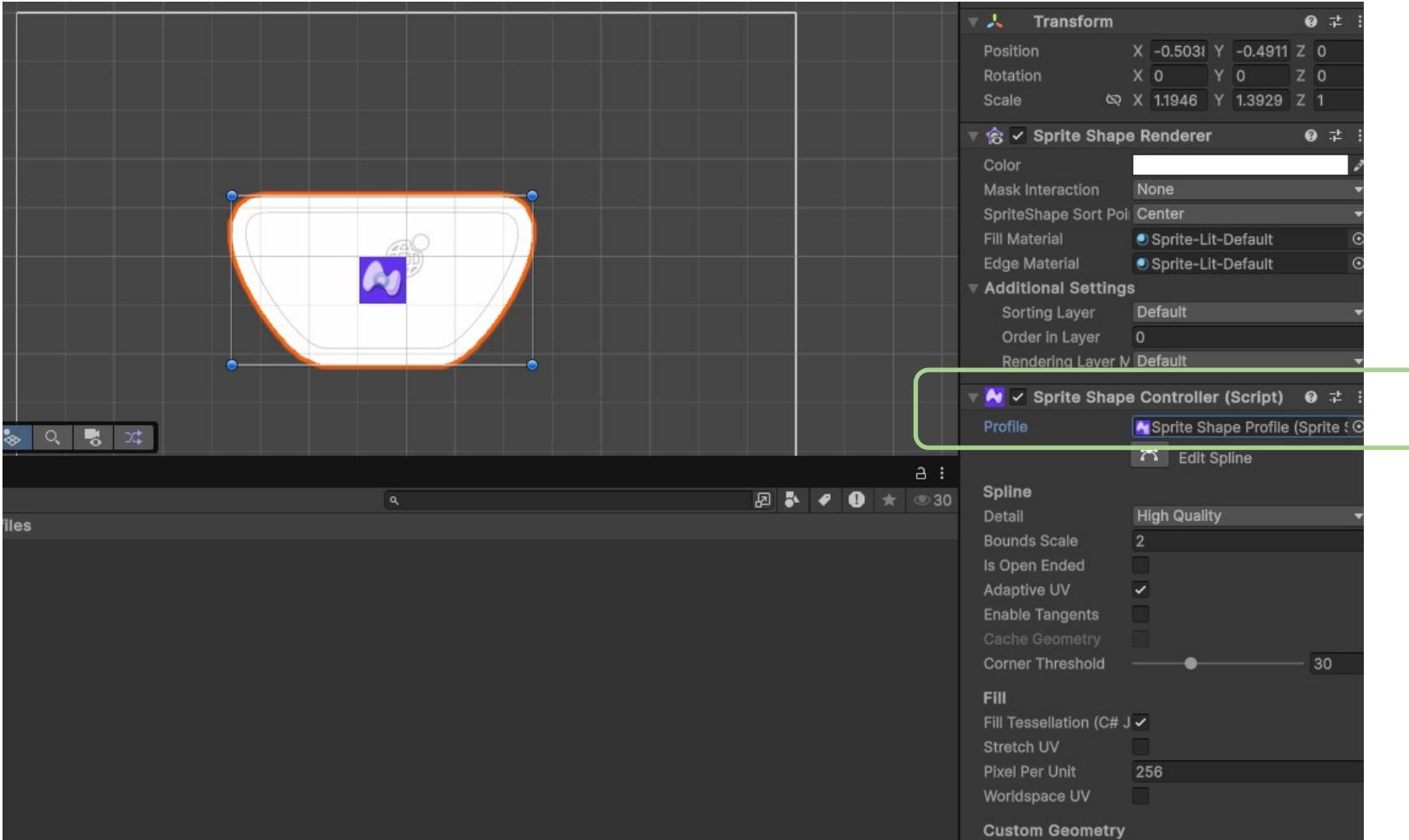


스프라이트 쉐이프(Sprite Shape) 추가

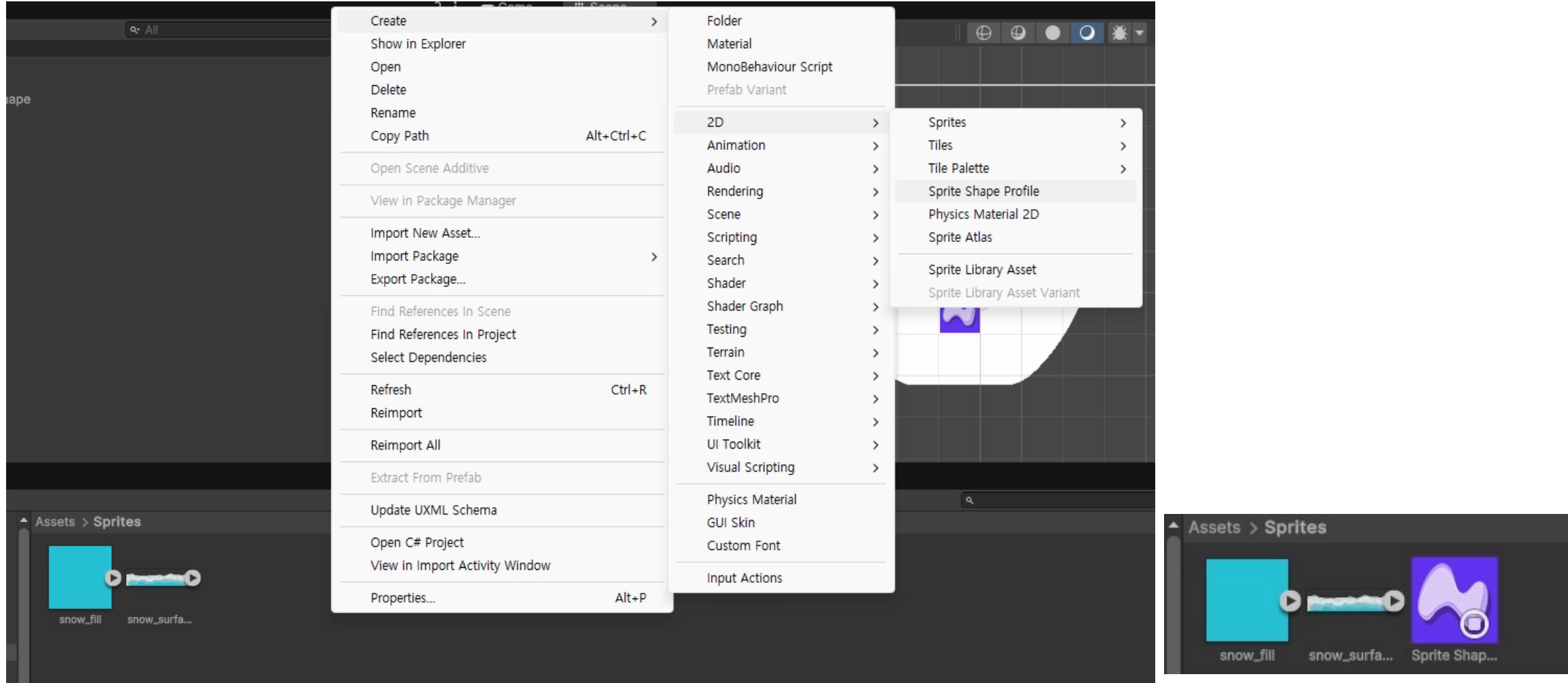
- Hierarchy에 Create → 2D Object → Sprite Shape → Closed Shape



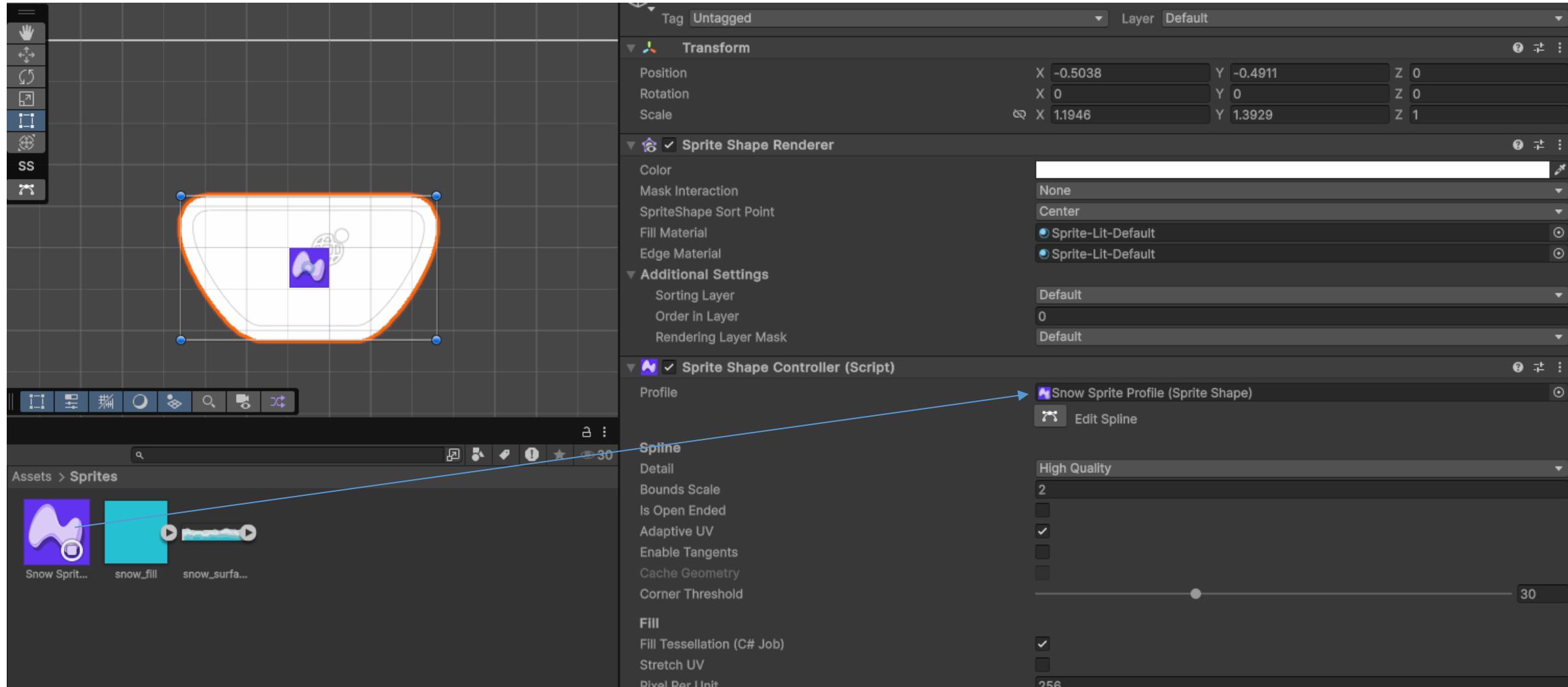
스프라이트 쉐이프 프로파일 - 디폴트



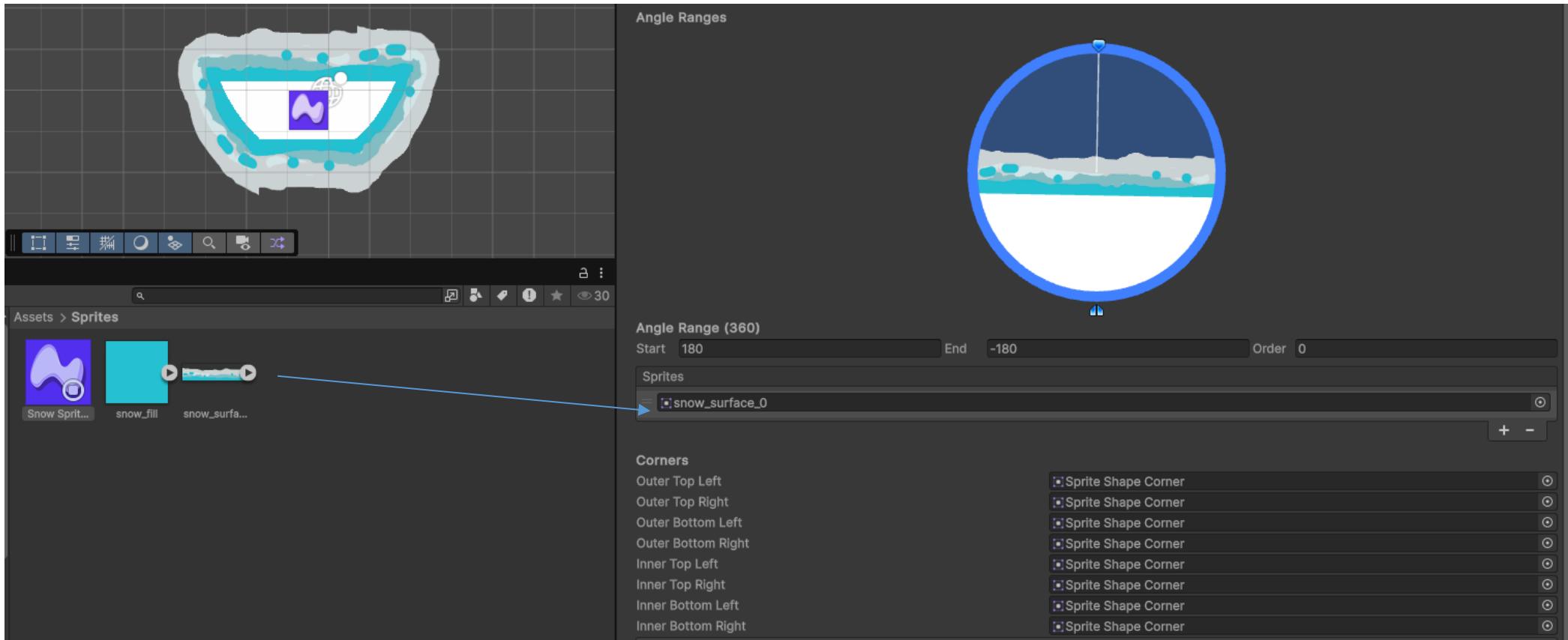
스프라이트 쉐이프 프로파일 – 나의 프로파일로 변경



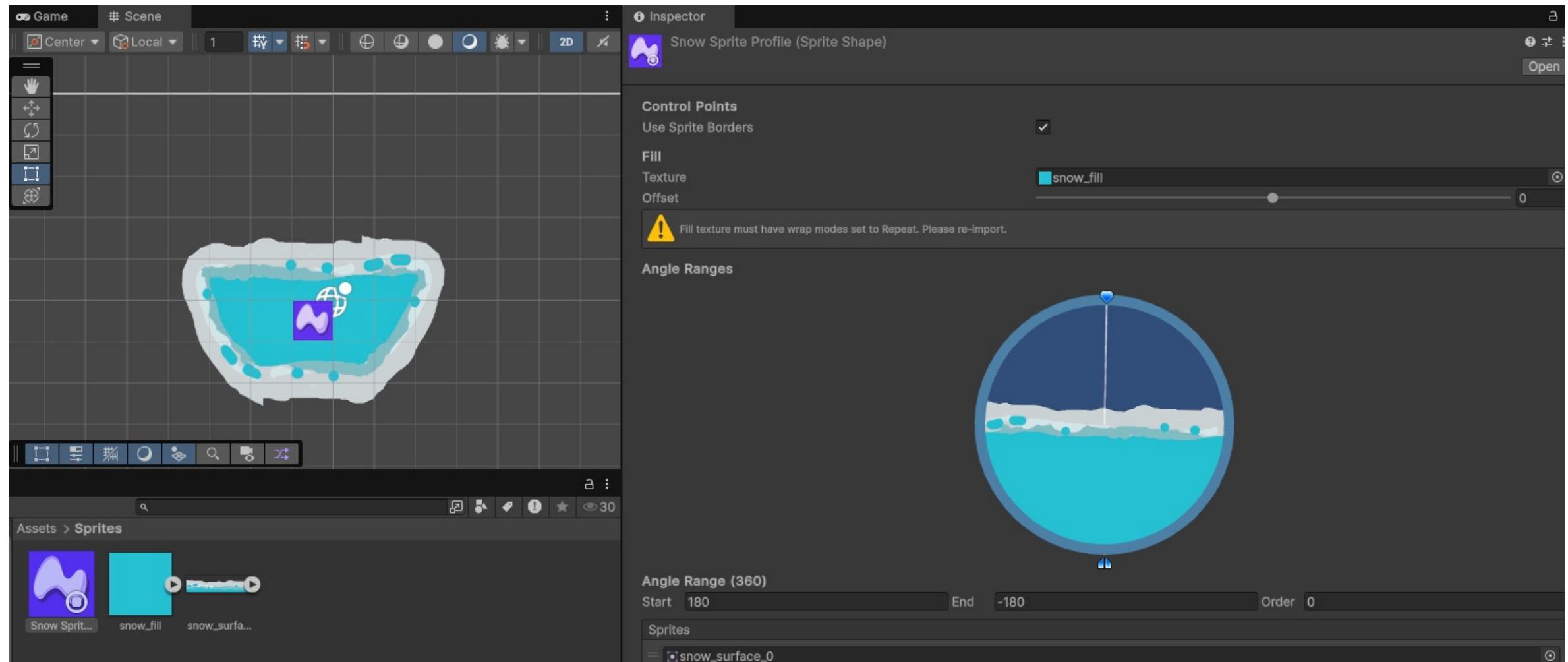
스프라이트 쉐이프에 나의 프로파일을 적용



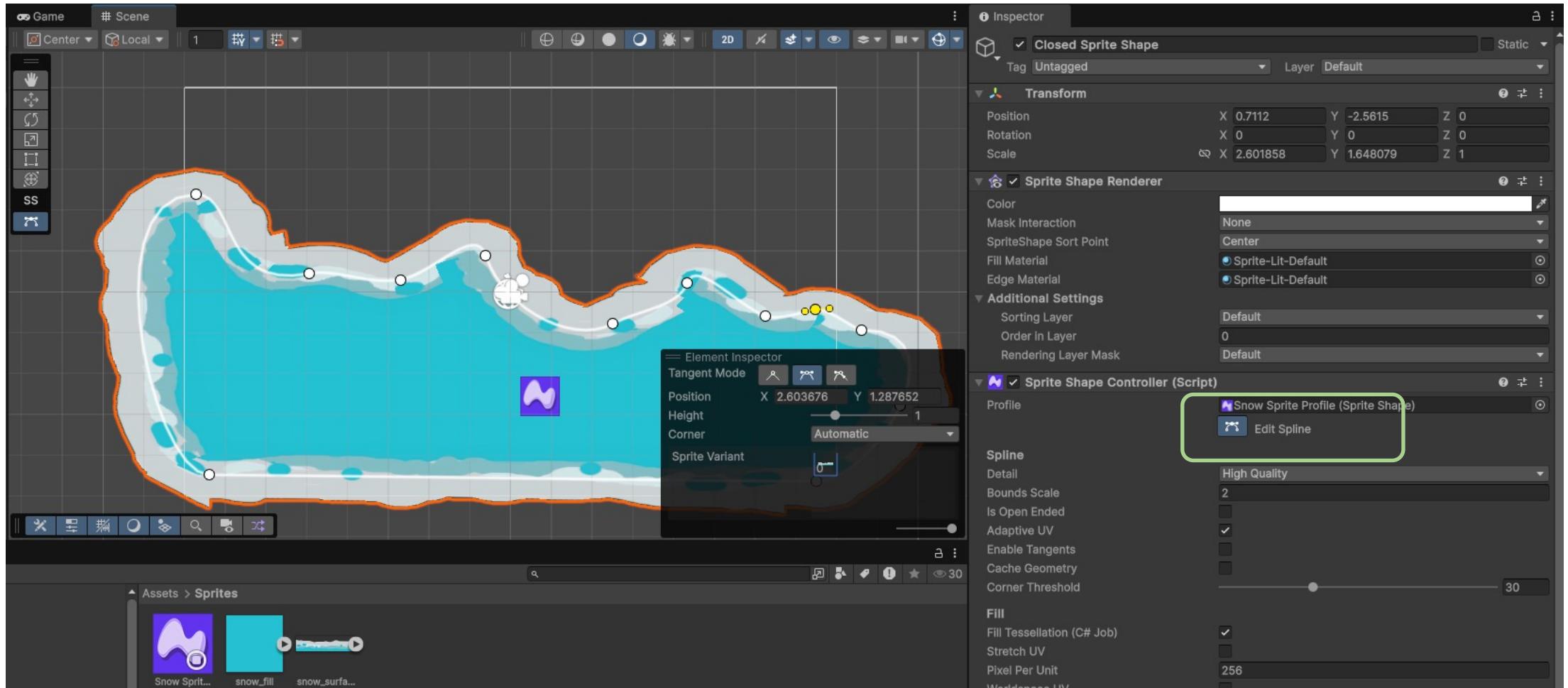
프로파일에 에지 스프라이트 적용



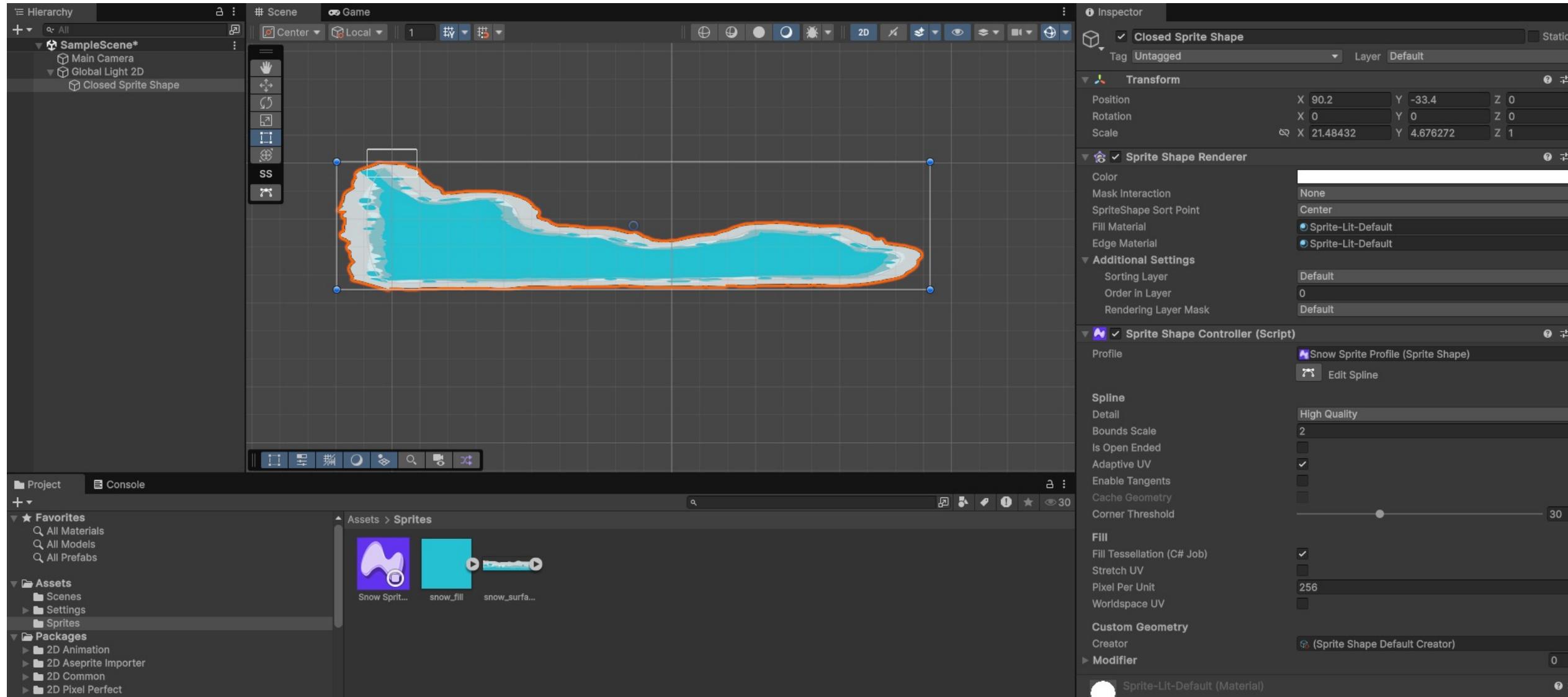
내부 채우기



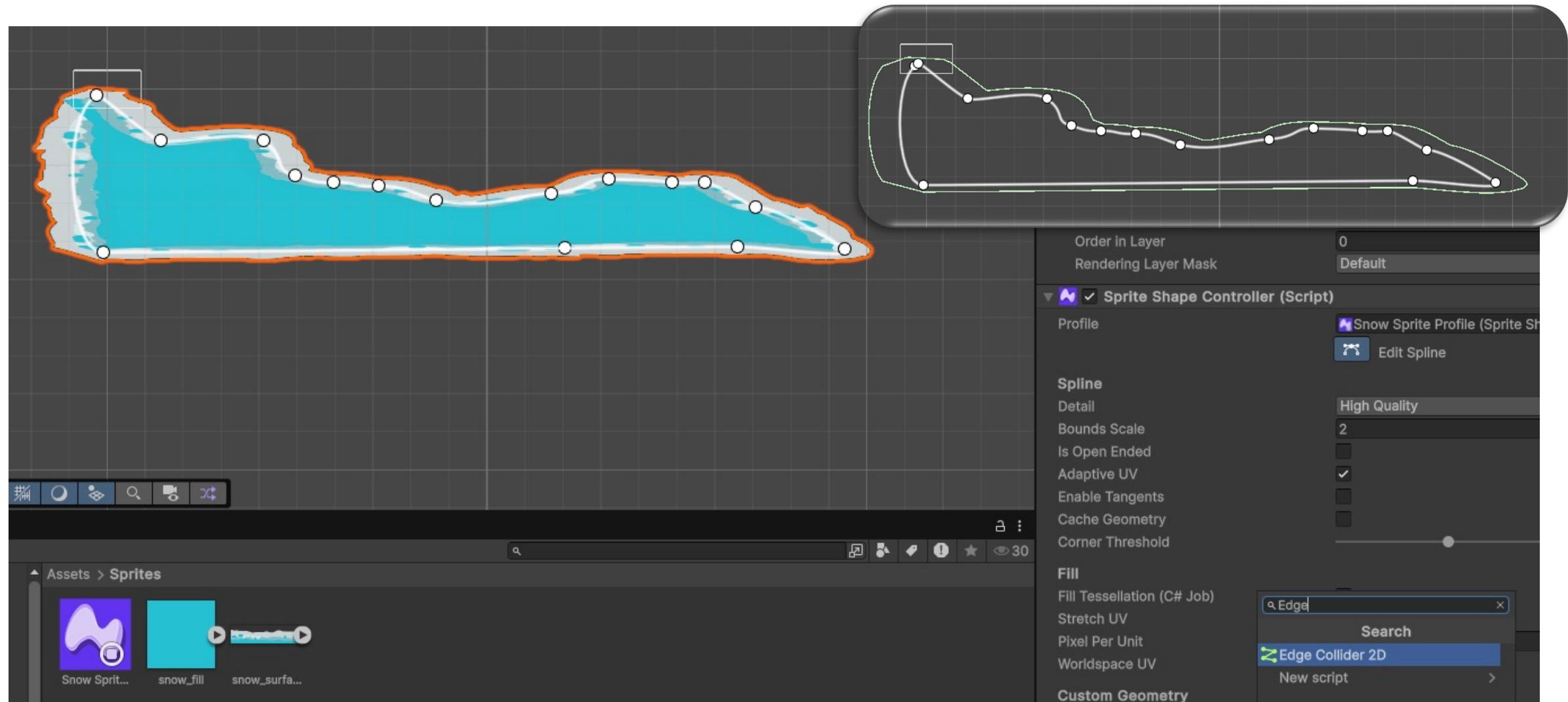
스프라이트 쉐이프의 모양 제어



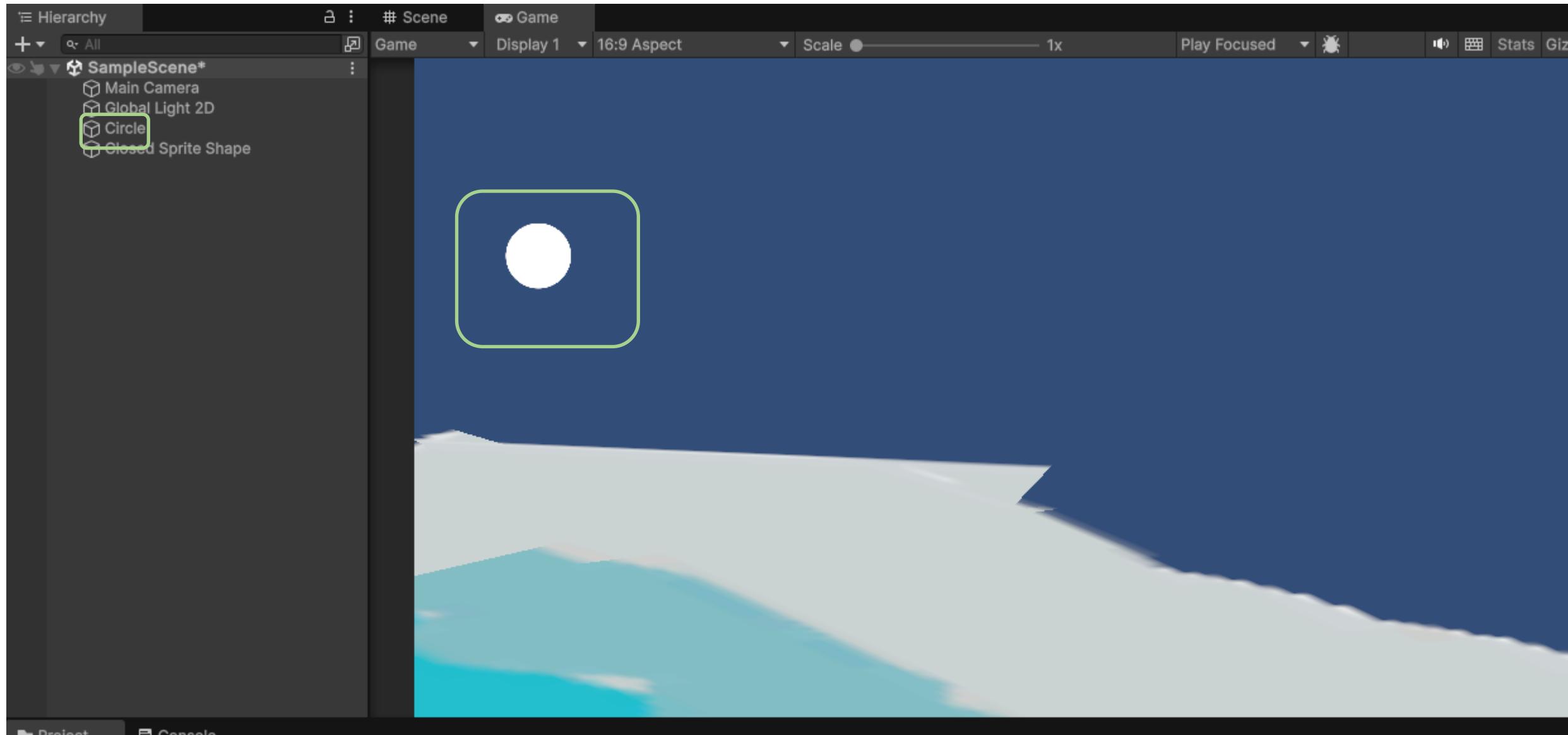
레벨 디자인 – 카메라보다 훨씬 큰 영역에 눈 산 배치



스프라이트 쉐이프에 에지 콜라이더 추가



경사를 따라 내려갈 공을 추가해 보라



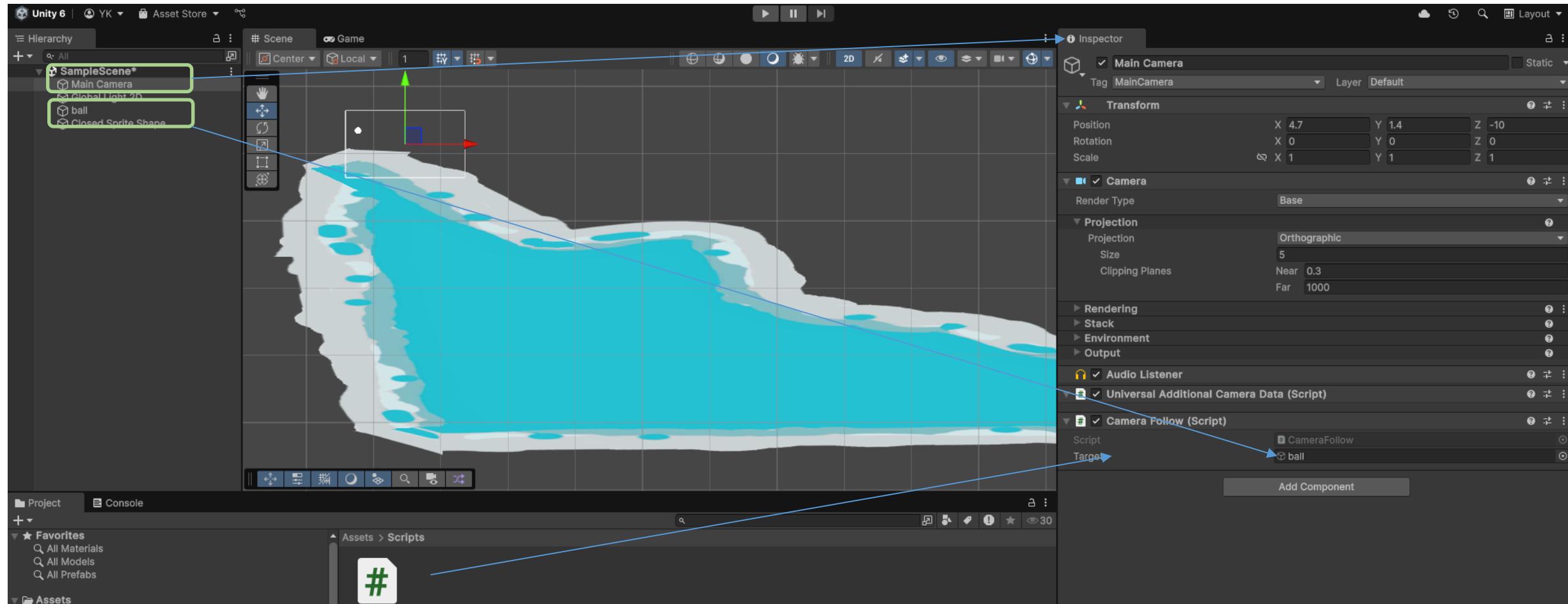
경사를 따라 공이 구르고, 카메라는 이를 쫓아 가보자

```
using UnityEngine;

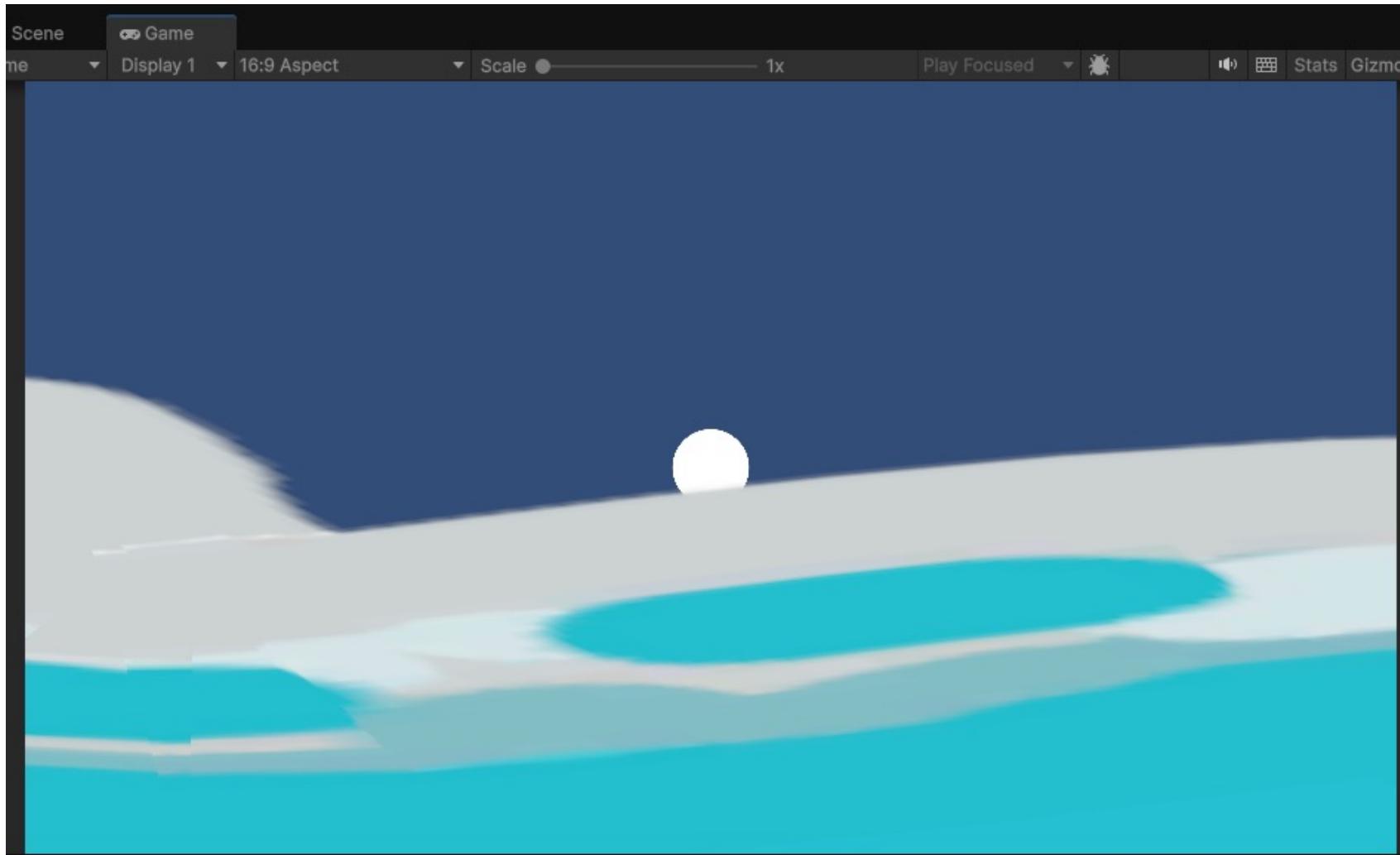
0 references
✓ public class CameraFollow : MonoBehaviour
{
    1 reference
    [SerializeField] GameObject target;
    // Start is called once before the first execution of Update after the MonoBehaviour is created
    0 references
    ✓ void Start()
    {
        ...
    }

    // Update is called once per frame
    0 references
    ✓ void Update()
    {
        transform.position = target.transform.position + new Vector3(0, 0, -10.0f);
    }
}
```

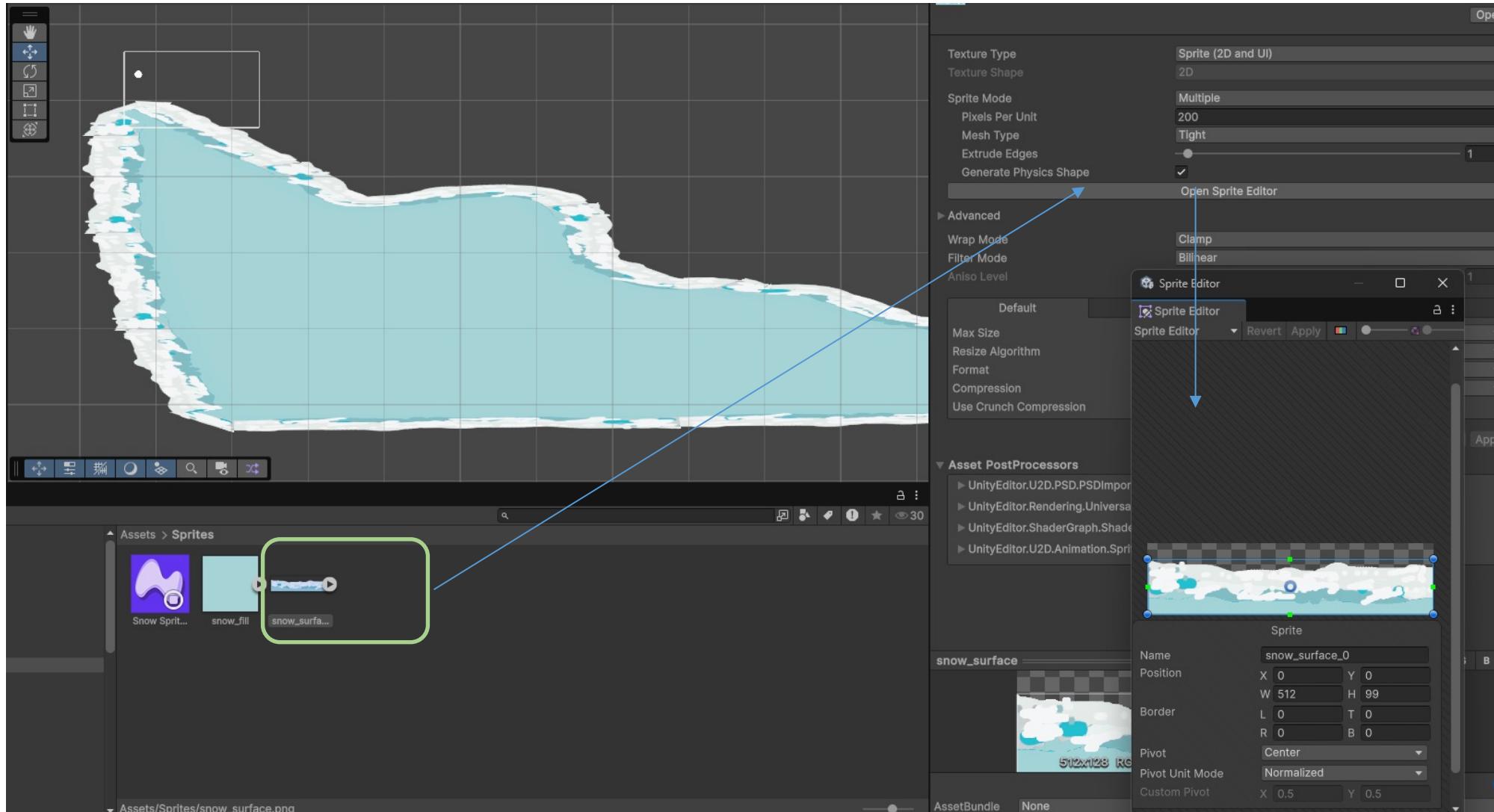
카메라에 코드 연결 + 대상인 공을 인자로 넘기기



테스트



스프라이트 에디터로 적용 영역 조정

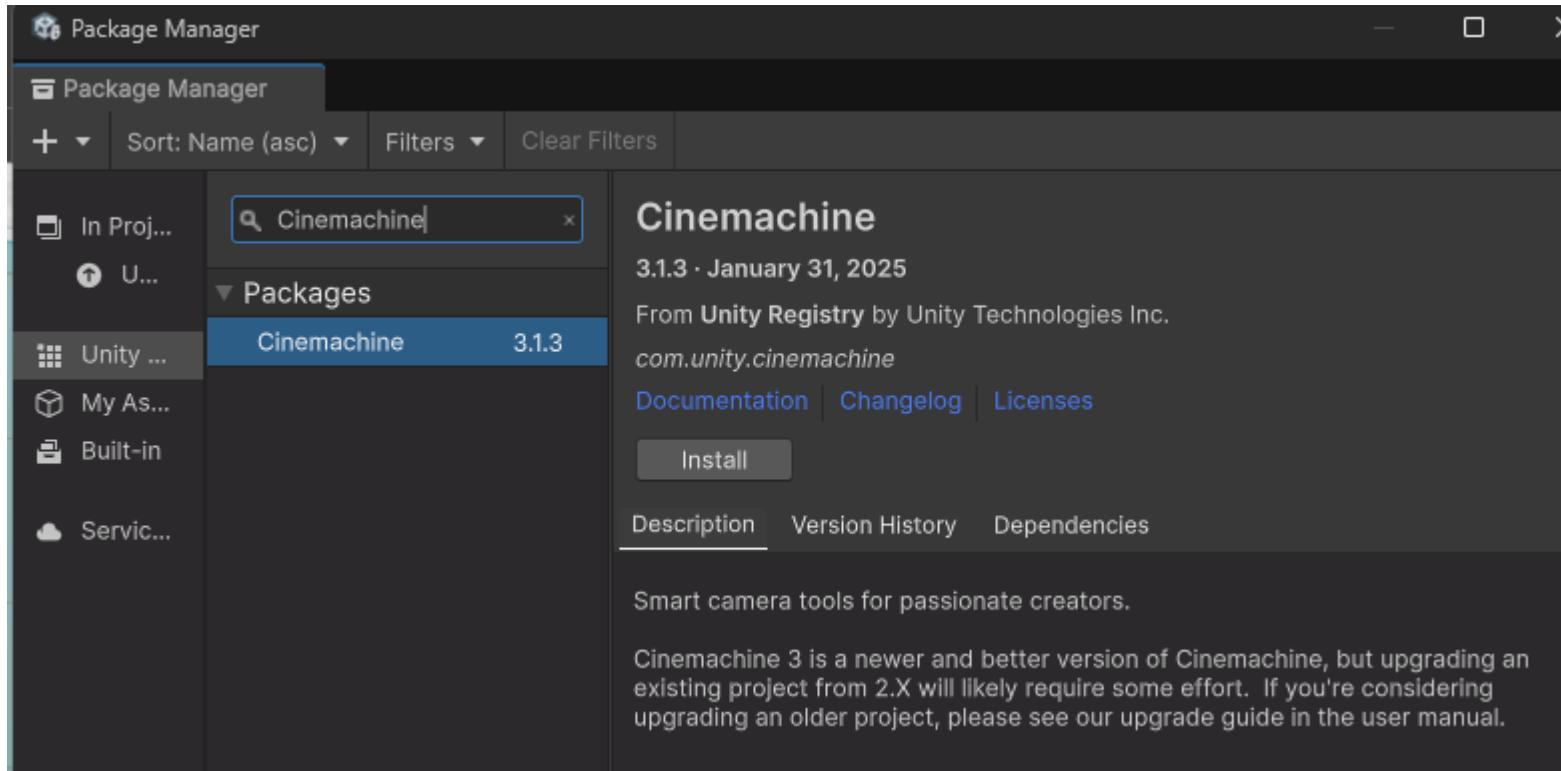


씨네머신

- 다수의 카메라를 장면에 도입할 수 있게 함
- 카메라에 적용되는 규칙을 간단히 정의할 수 있음
- 시네머신이 다수의 가상 카메라를 제어
- 규칙에 따라 카메라 선택과 동작 제어
- 시네머신: 패키지의 일종
- 사용하고 싶으면 프로젝트에 추가해야 함

씨네마신 패키지 사용

Windows → Package Manager → Unity Registry → Search cinemachine → Install!

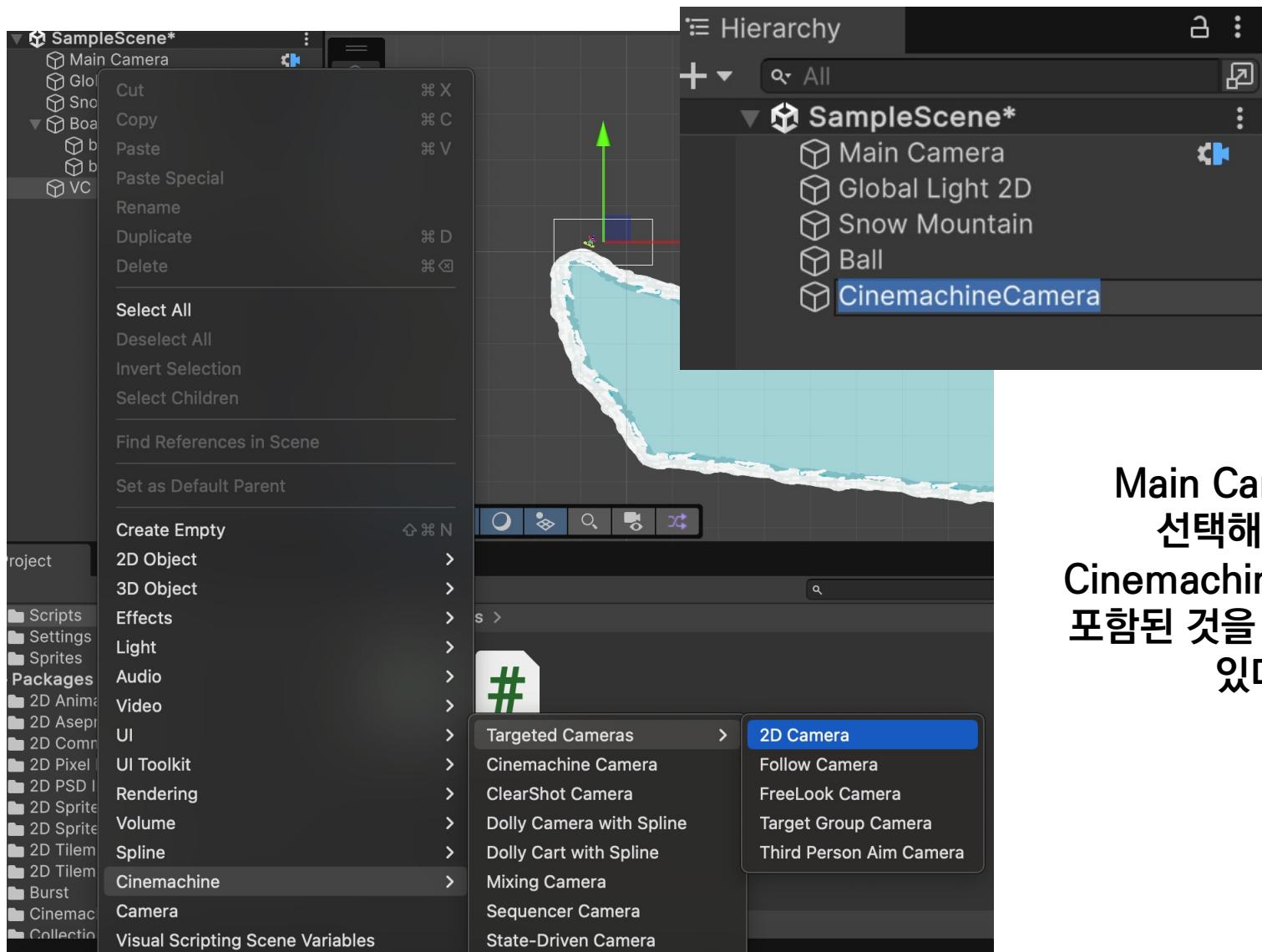


설치된 Cinemachine 패키지 확인

Components
메뉴에서 확인 가능



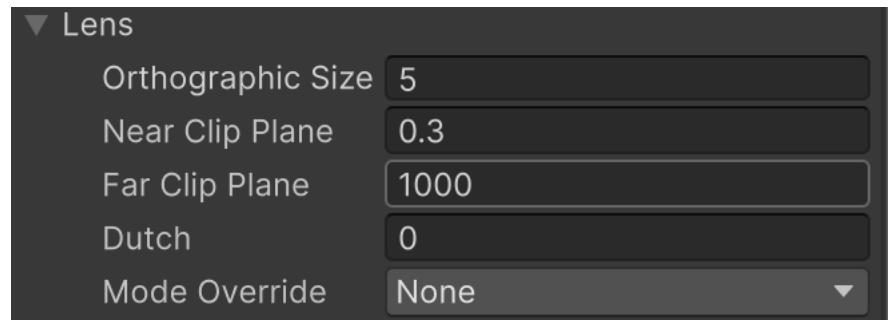
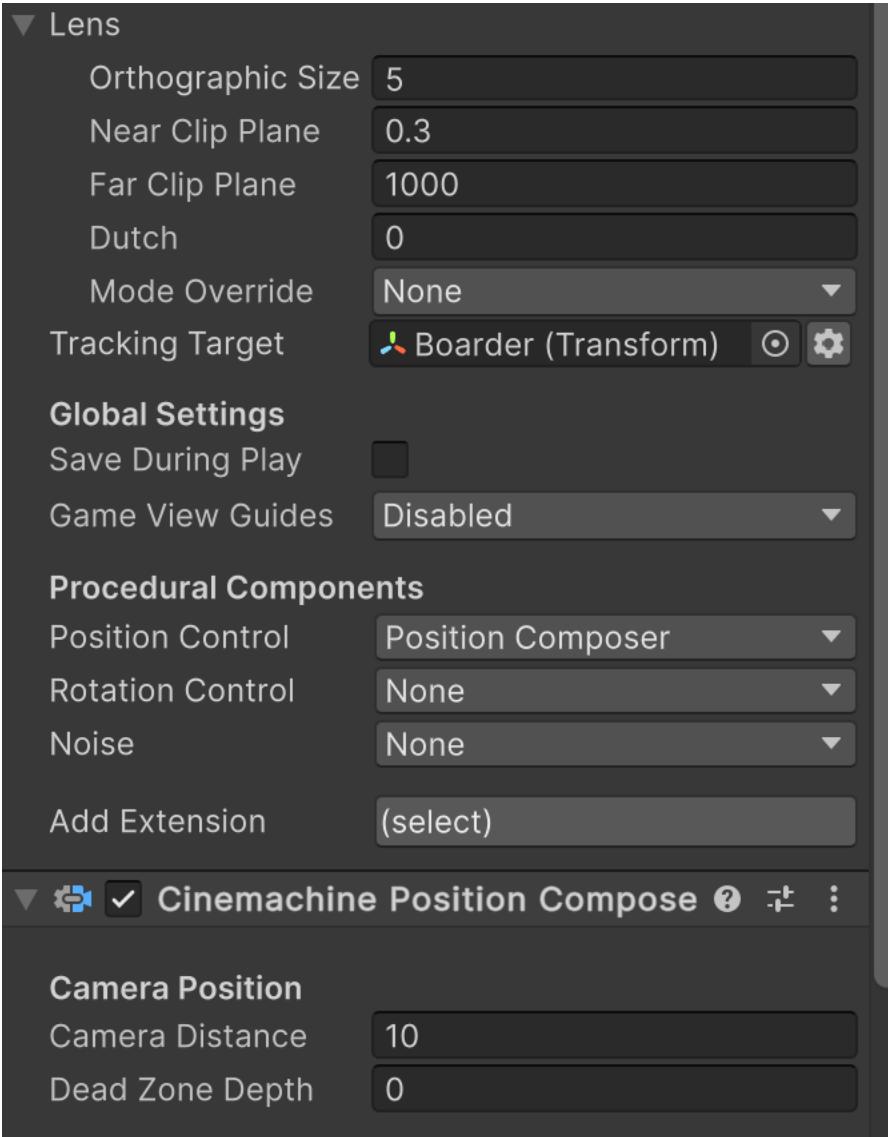
Hierarchy에서 create를 통해 설치 확인 가능



CinemachineCamera를
Targeted Camera – 2D Camera
로 생성해 보자

Main Camera를
선택해 보면
Cinemachine Brain이
포함된 것을 확인 할 수
있다

Targeted Camera 를 설정해 보자



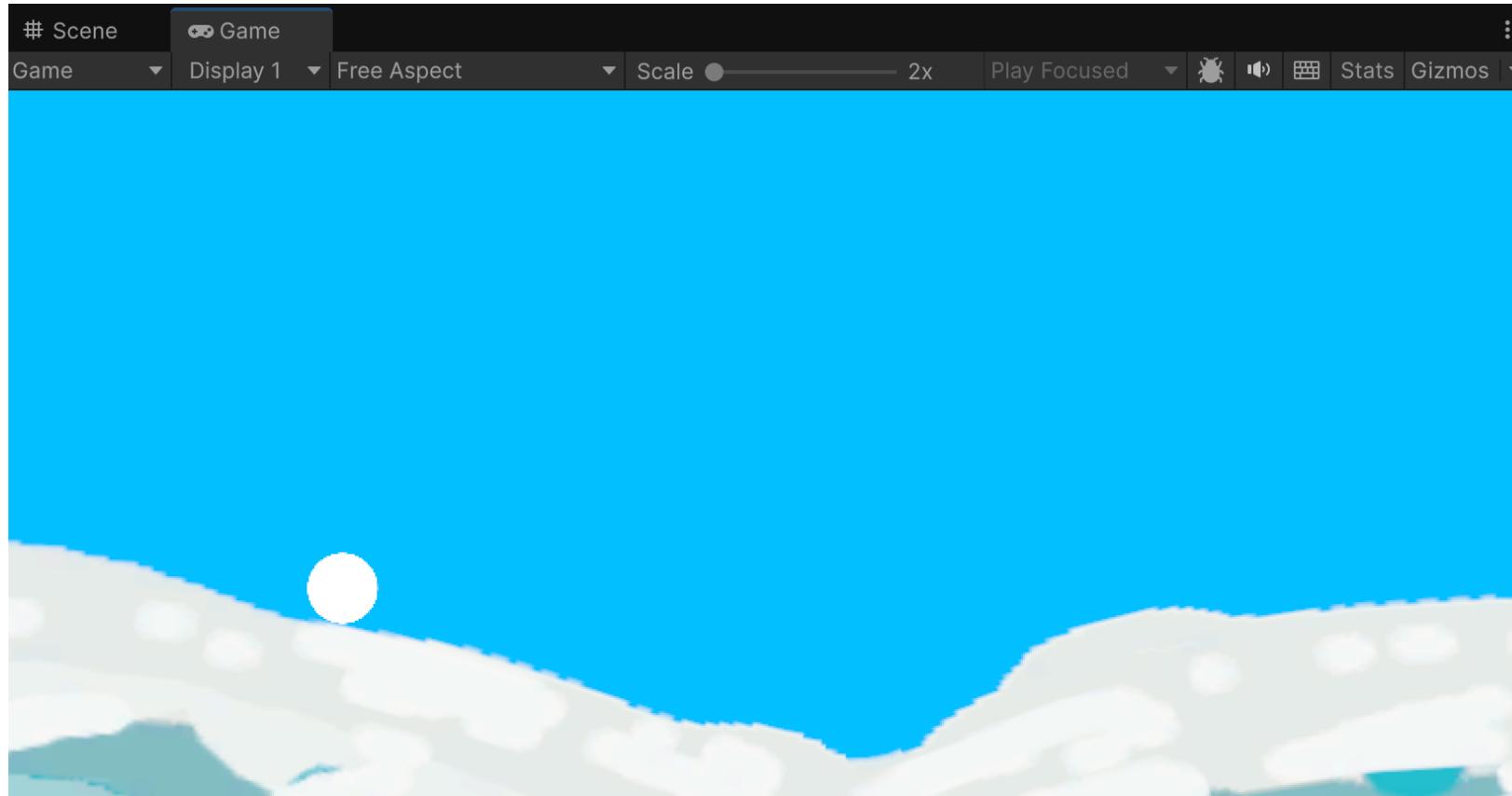
카메라 영역의 크기 조정

따라야 할 객체로
Ball 선택

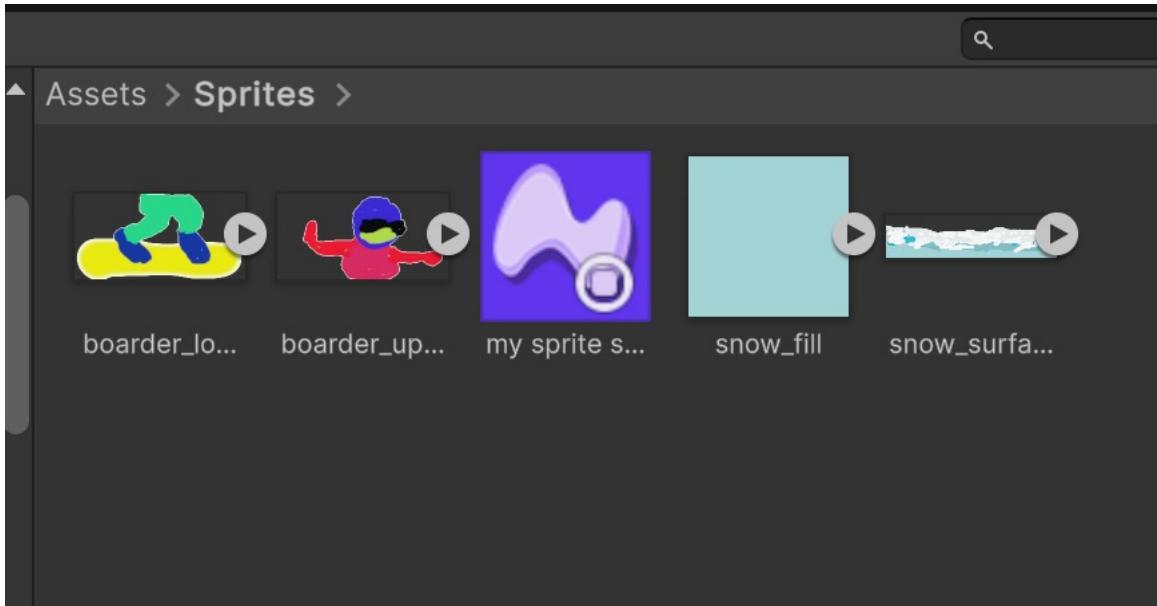
위치와 회전 동작 지정

포지션 오프셋으로
카메라가 보는 영역 조정

자연스러운 카메라 이동

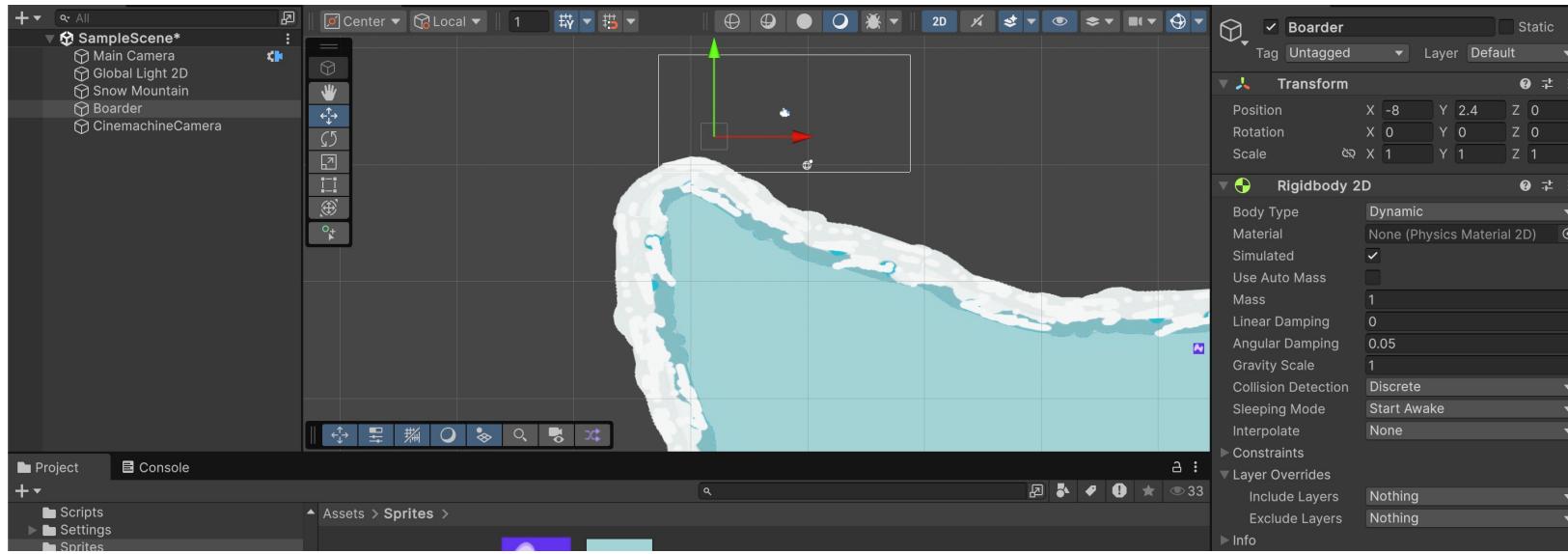


캐릭터 설정



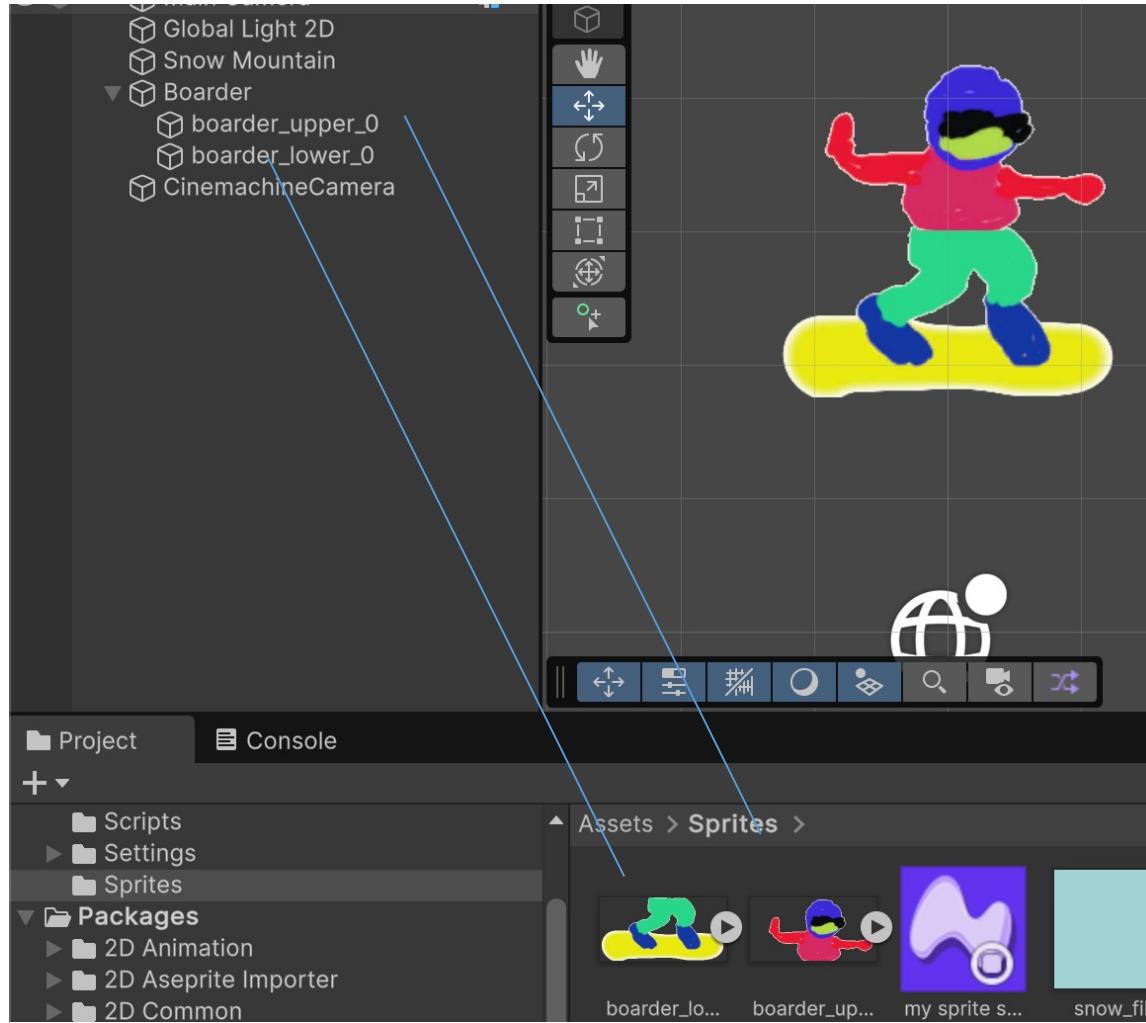
캐릭터에 필요한
애셋 추가

각각의 객체에 Collider 컴포넌트 추가



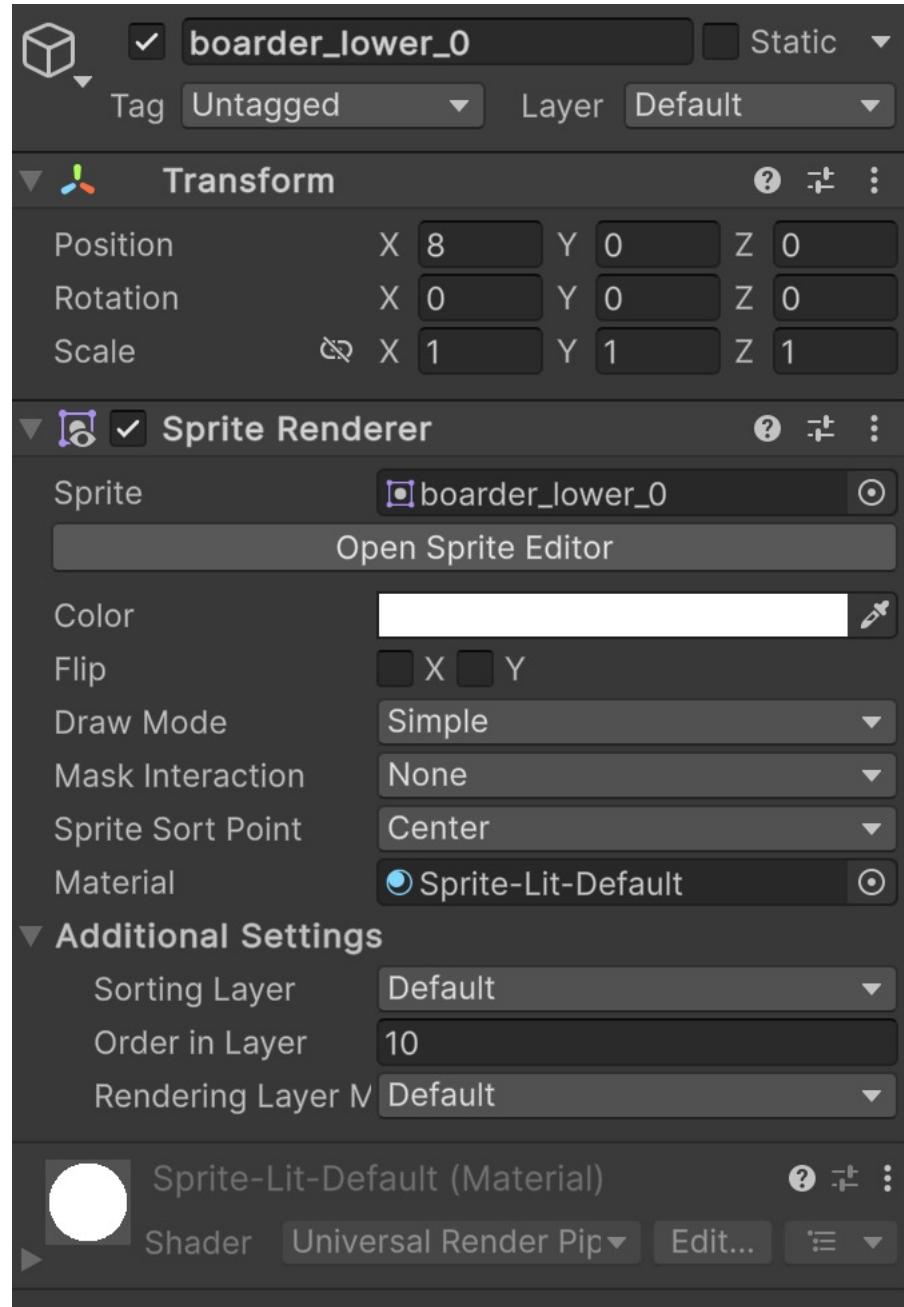
Ball을 Boarder로 이름 바꾸기
- Sprite Renderer 제거
- Circle Collider 제거

Hierarchy에 스프라이트 끌어서 배치



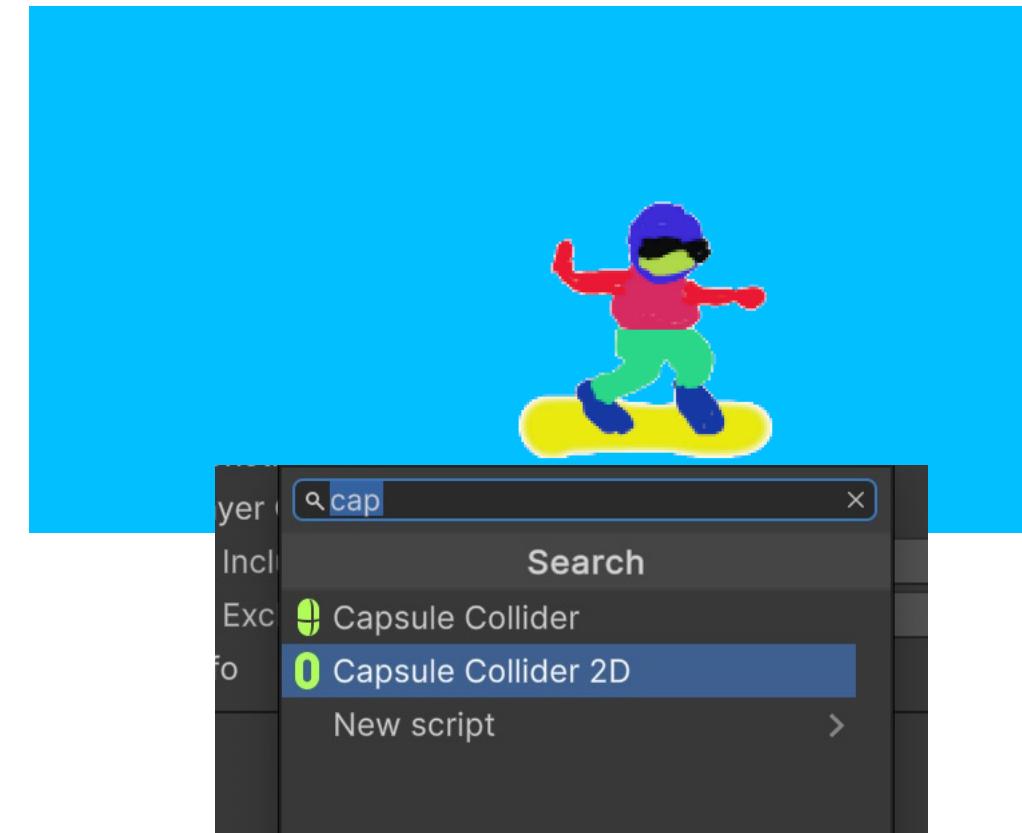
Boarder의 자식 객체로 추가
- 위의 변환을 그대로 물려받음

문제점



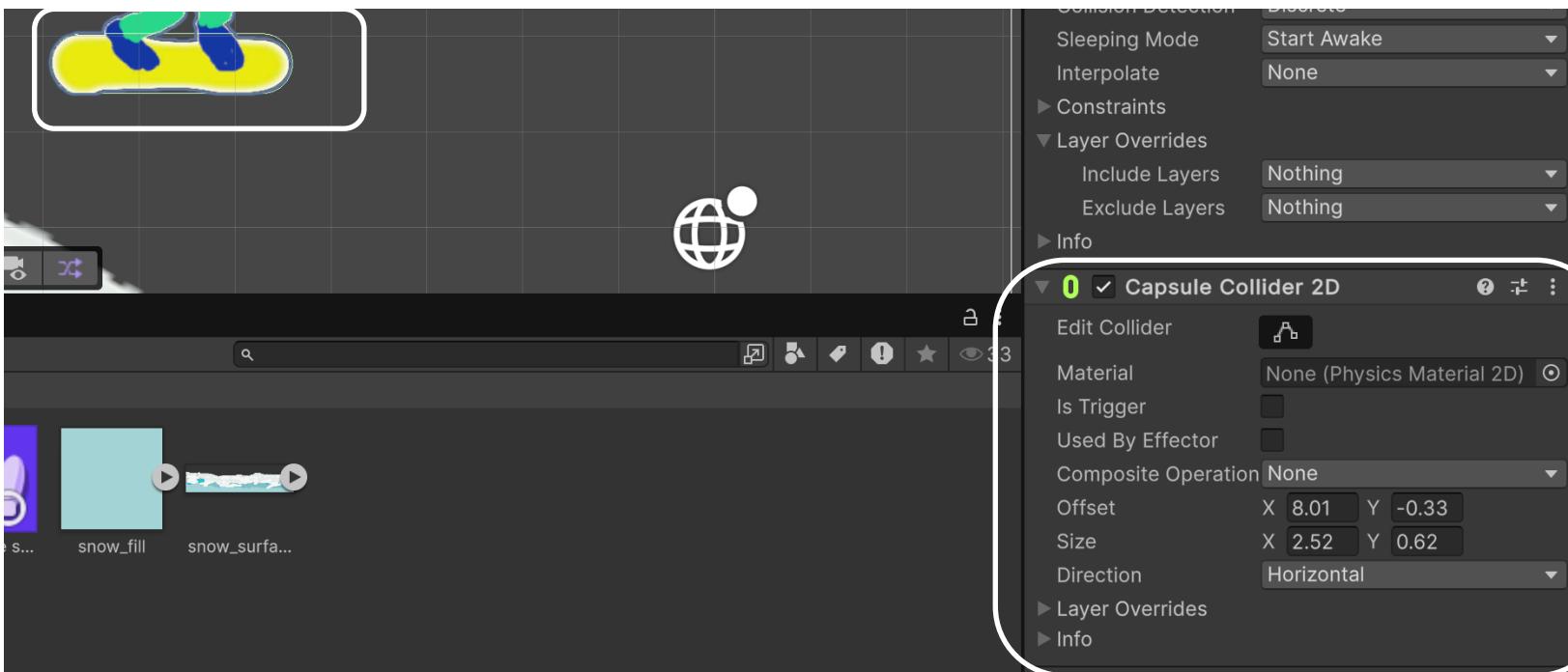
보더의 모양이 눈 산 뒤로 떨어질 것

- 레이어 순서 값을 큰 값으로 변경
- 앞에 나타나지만 계속 떨어짐
- 콜라이더 필요

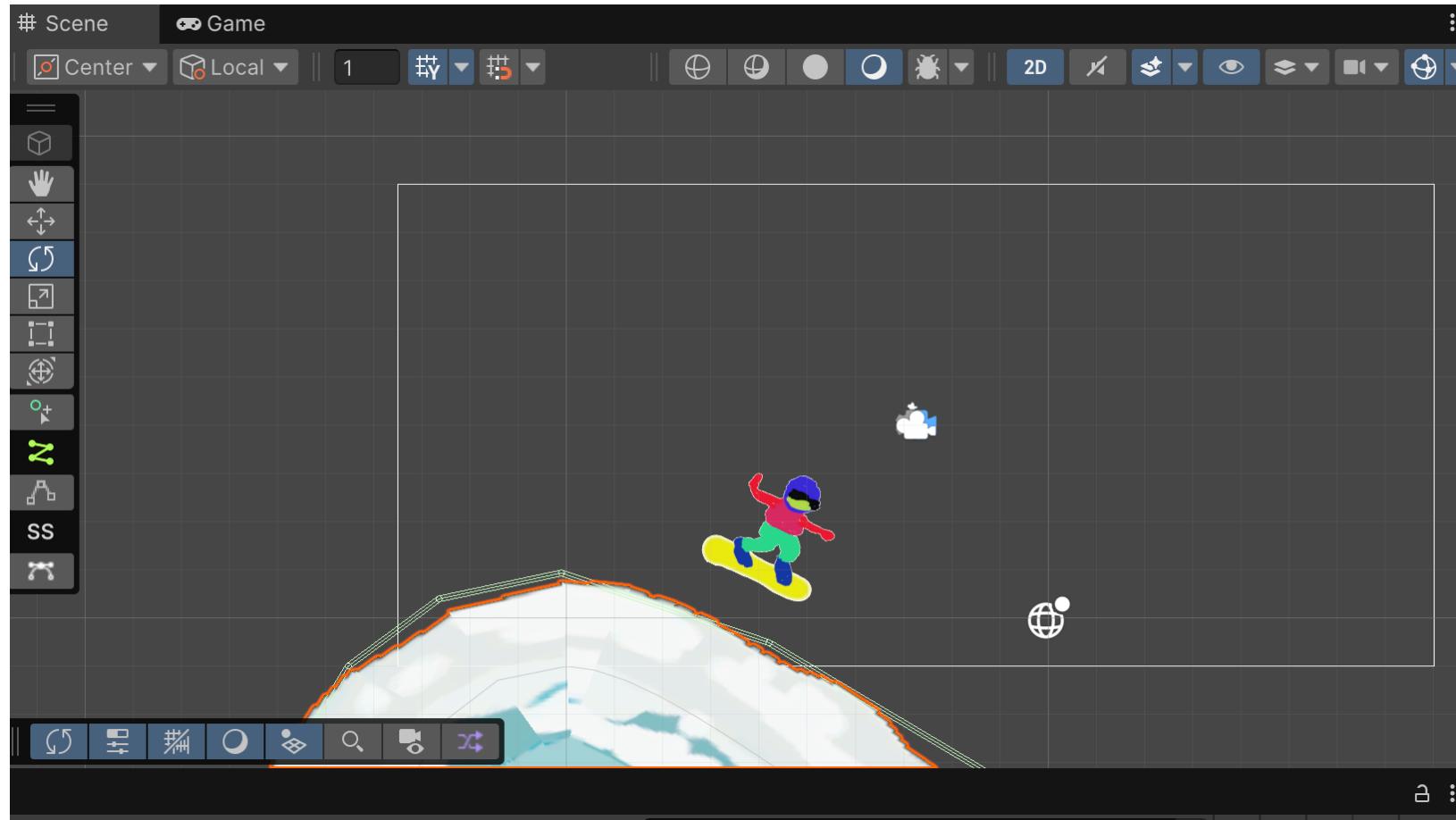


보더 객체에 콜라이더 추가

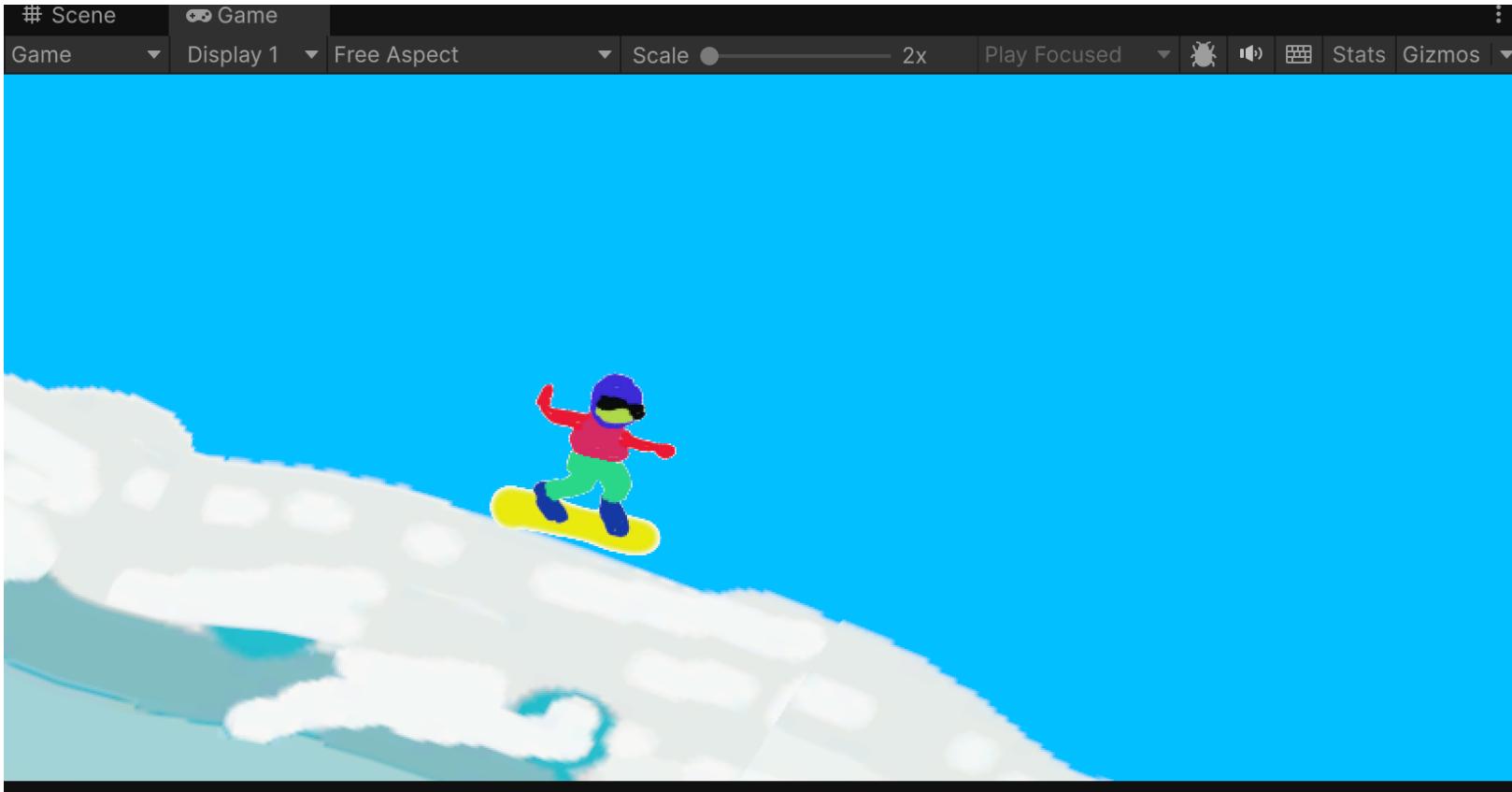
콜라이더의 모양 조정



카메라와 캐릭터 회전 등 튜닝



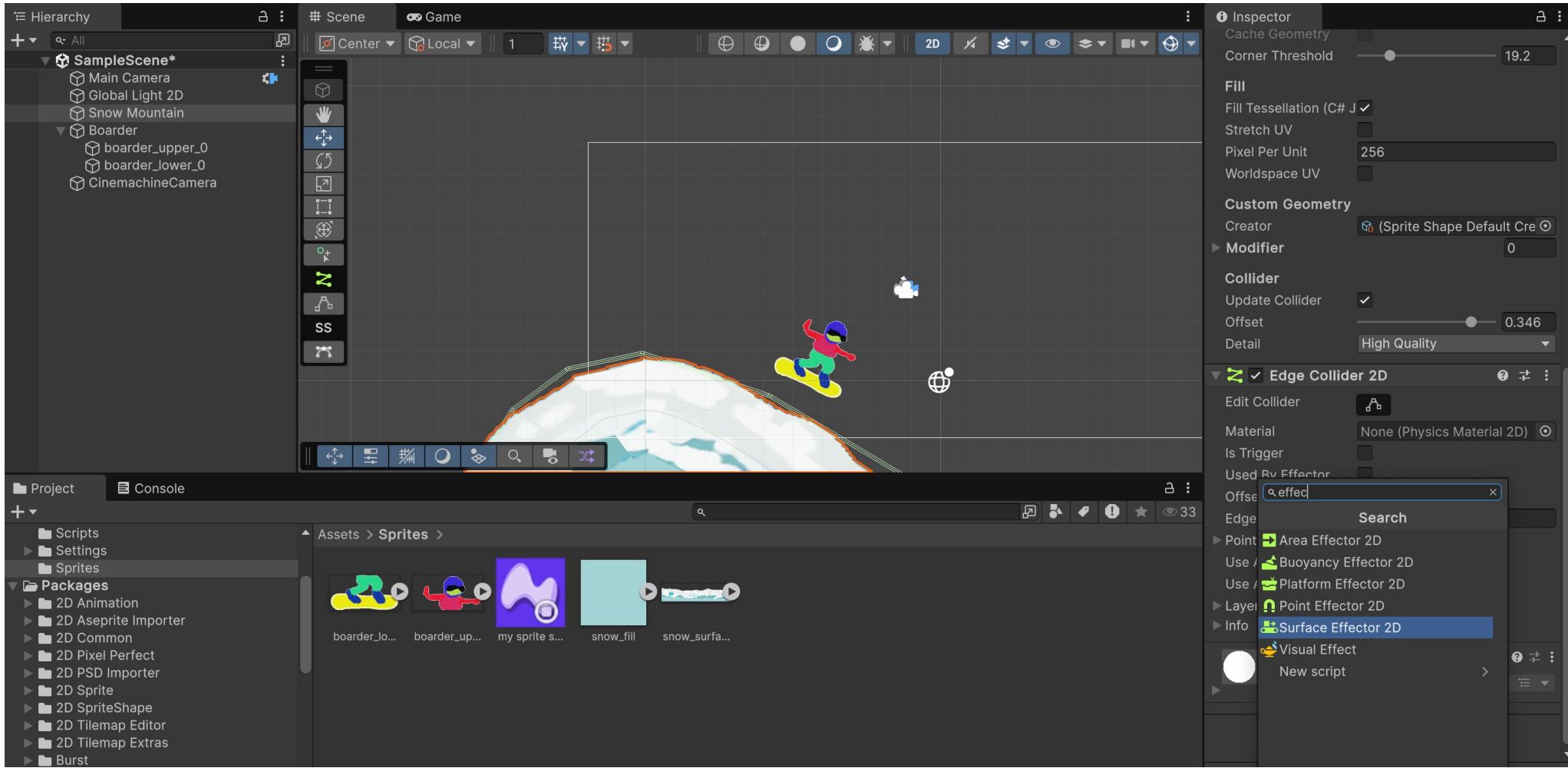
중력만으로는 보더를 계속 보내기 힘들다 → 이펙터 사용



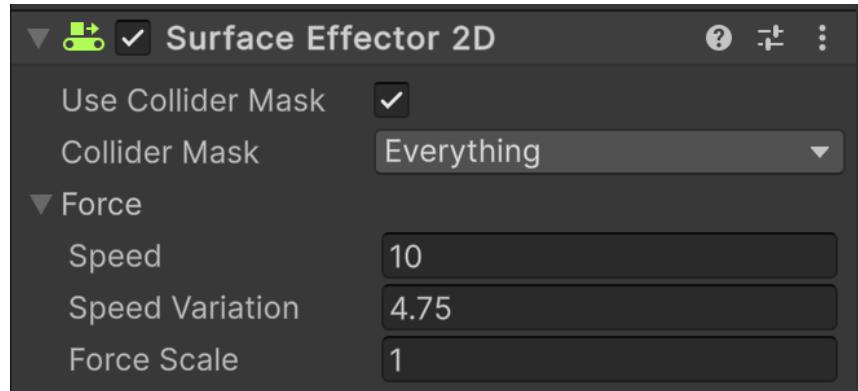
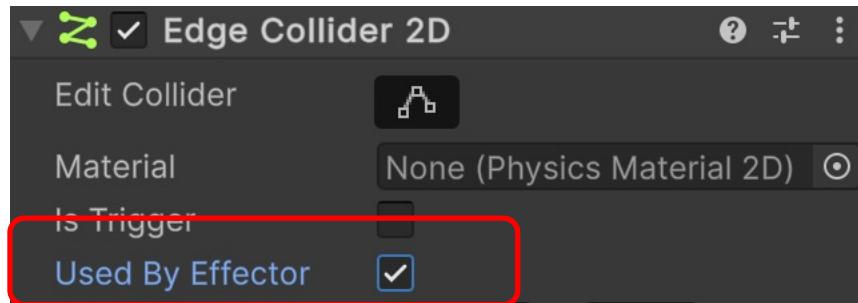
Surface Effector

- Collider 표면에서 힘이 발생하게 함
 - 콜라이더 표면이 일정의 컨베이어 벨트처럼 되게 할 수 있음
- 이펙터의 종류
 - 서피스 이펙터
 - 부력 이펙터
 - 포인트 이펙터 등

설산에 이펙터 컴포넌트 추가



콜라이더 설정 – Used by Effector



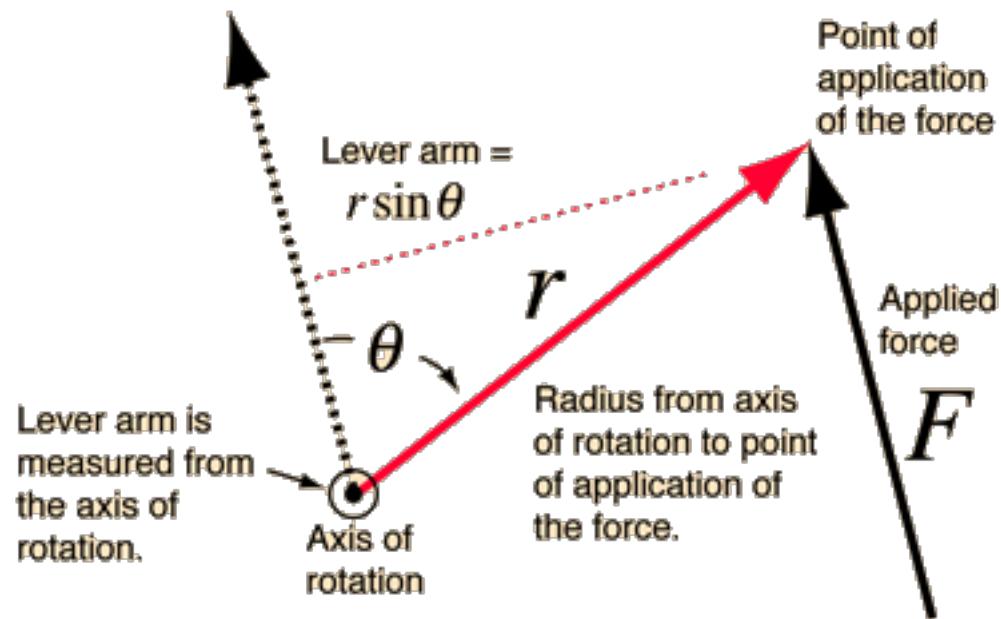
Speed: 벨터의 속도

Force Scale: 중력에 저항해서
캐릭터에 힘을 주는 정도

보더가 앞 뒤로 회전할 수 있게 하자

회전은 토크로 바꿀 수 있다

$$\text{torque} = \tau = rF \sin \theta$$

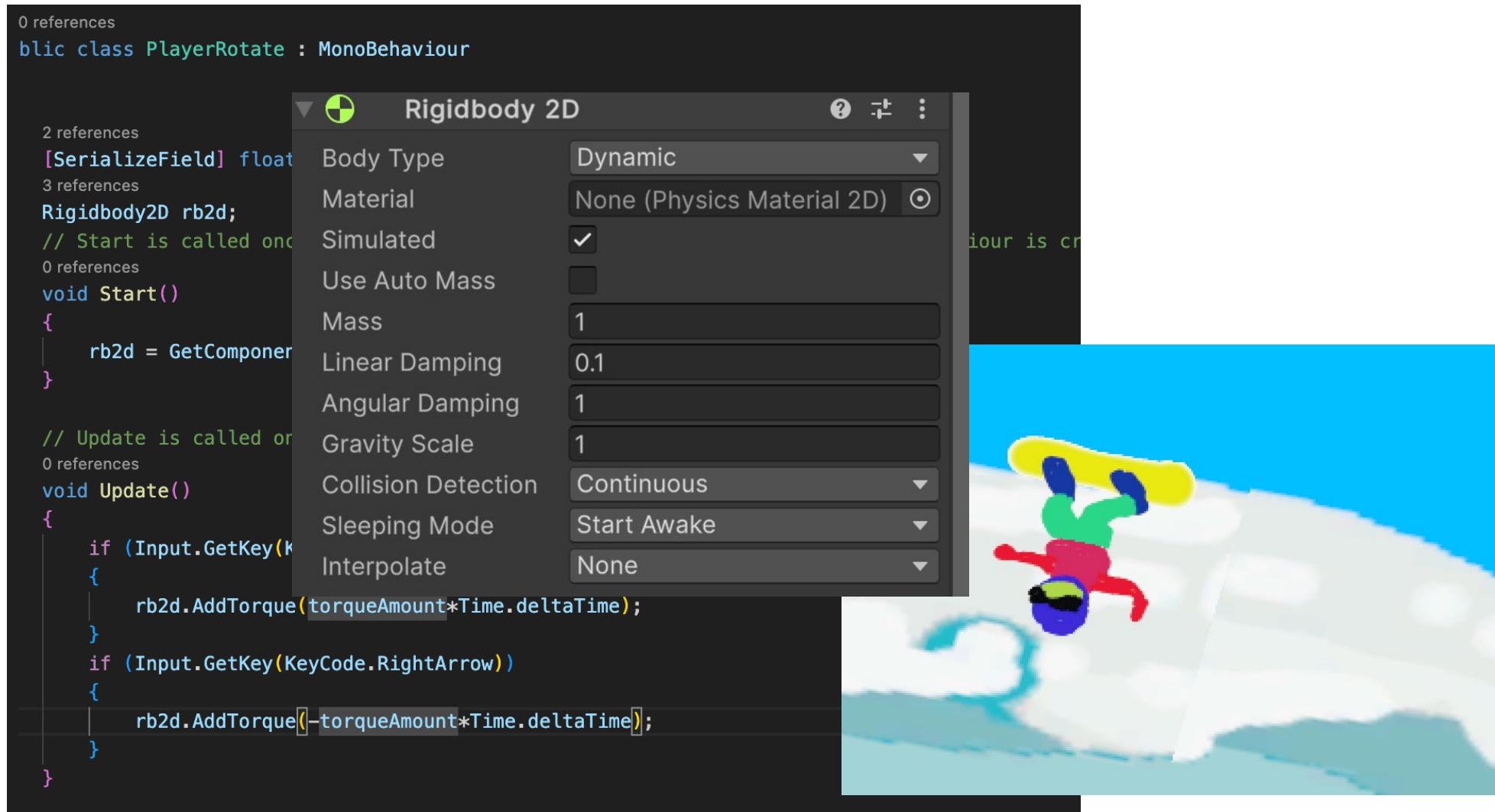


토크 표현방법:

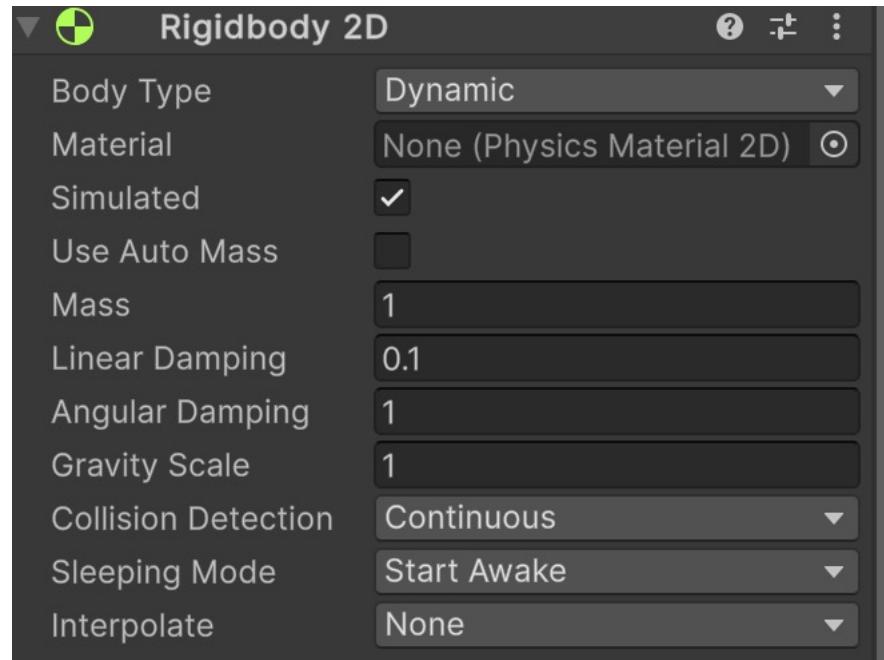
회전력의 크기 x 회전 축

2D 게임에서는 언제나 z축 회전
(0, 0, t)

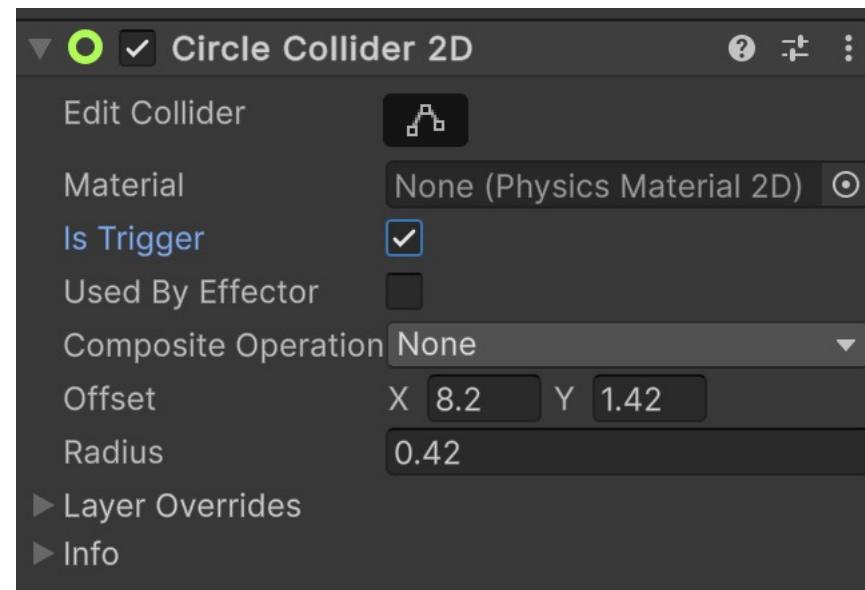
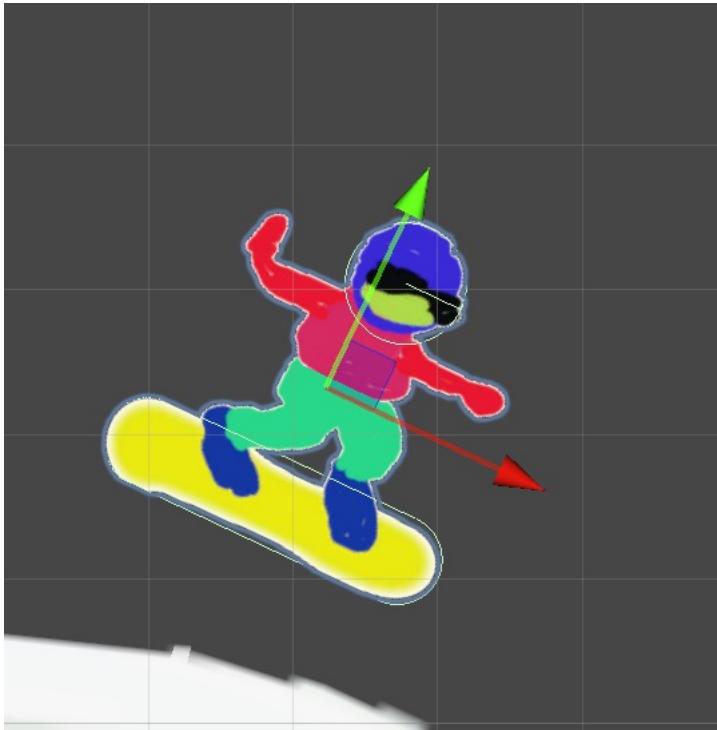
Boarder에 스크립트 연결



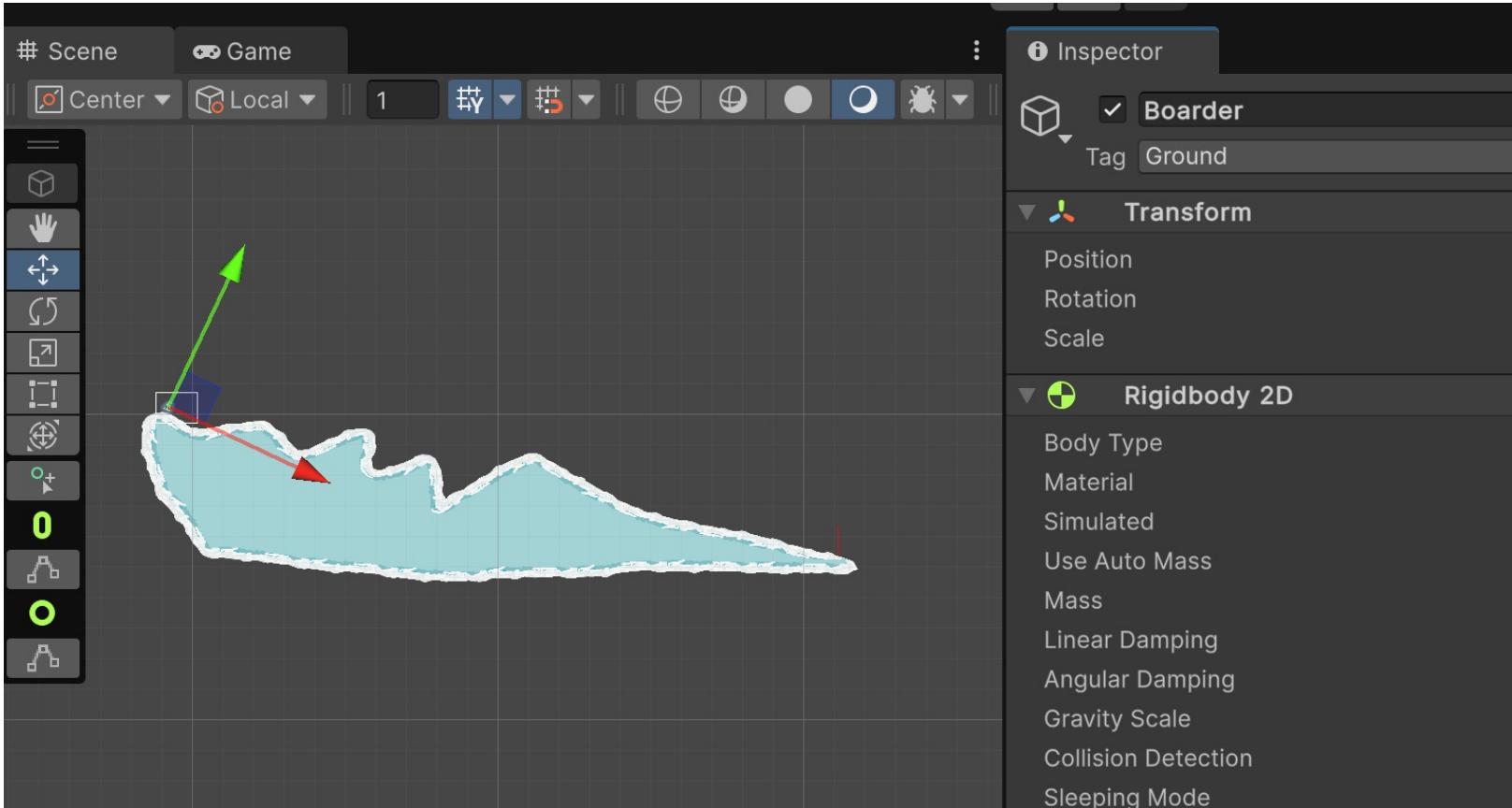
Linear & Angular Damping



실패를 감지하기 위해 보더의 머리에 트리거 설치

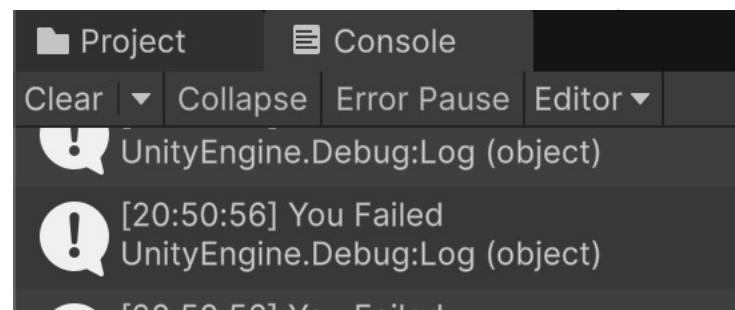


Sprite Shape에 태그를 “Ground”로 설정

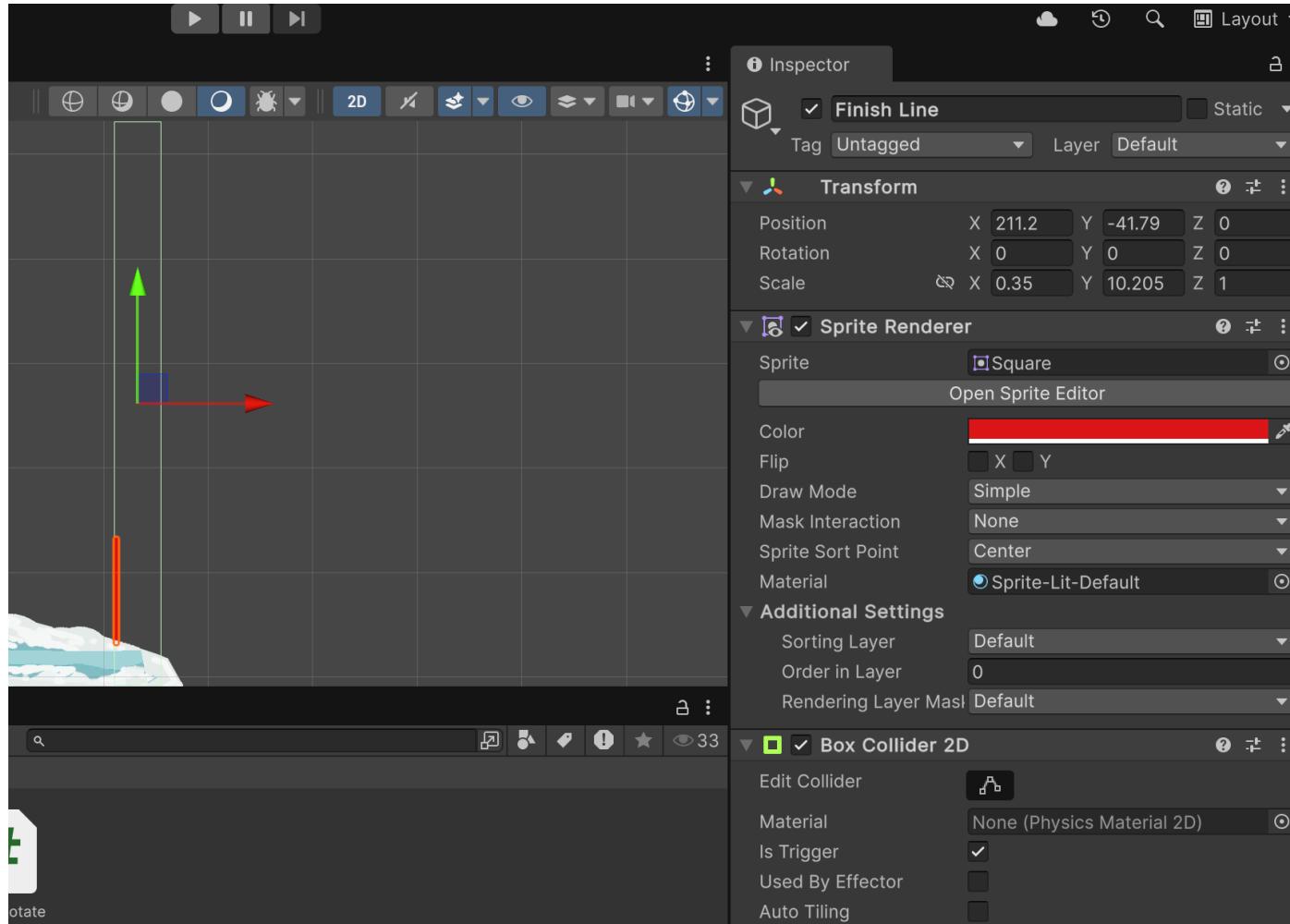


CrashDetector.cs 스크립트를 보더에 연결

```
0 references
public class CrashDetector : MonoBehaviour
{
    // Start is called once before the first execution of Update
    0 references
    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "Ground")
        {
            Debug.Log("You Failed");
        }
    }
}
```



Finish Line에 객체 설치하고 Trigger 콜라이더 설치



```
0 references
public class Finish : MonoBehaviour
{
    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "Player")
        {
            Debug.Log("You Won!!!");
        }
    }
}
```

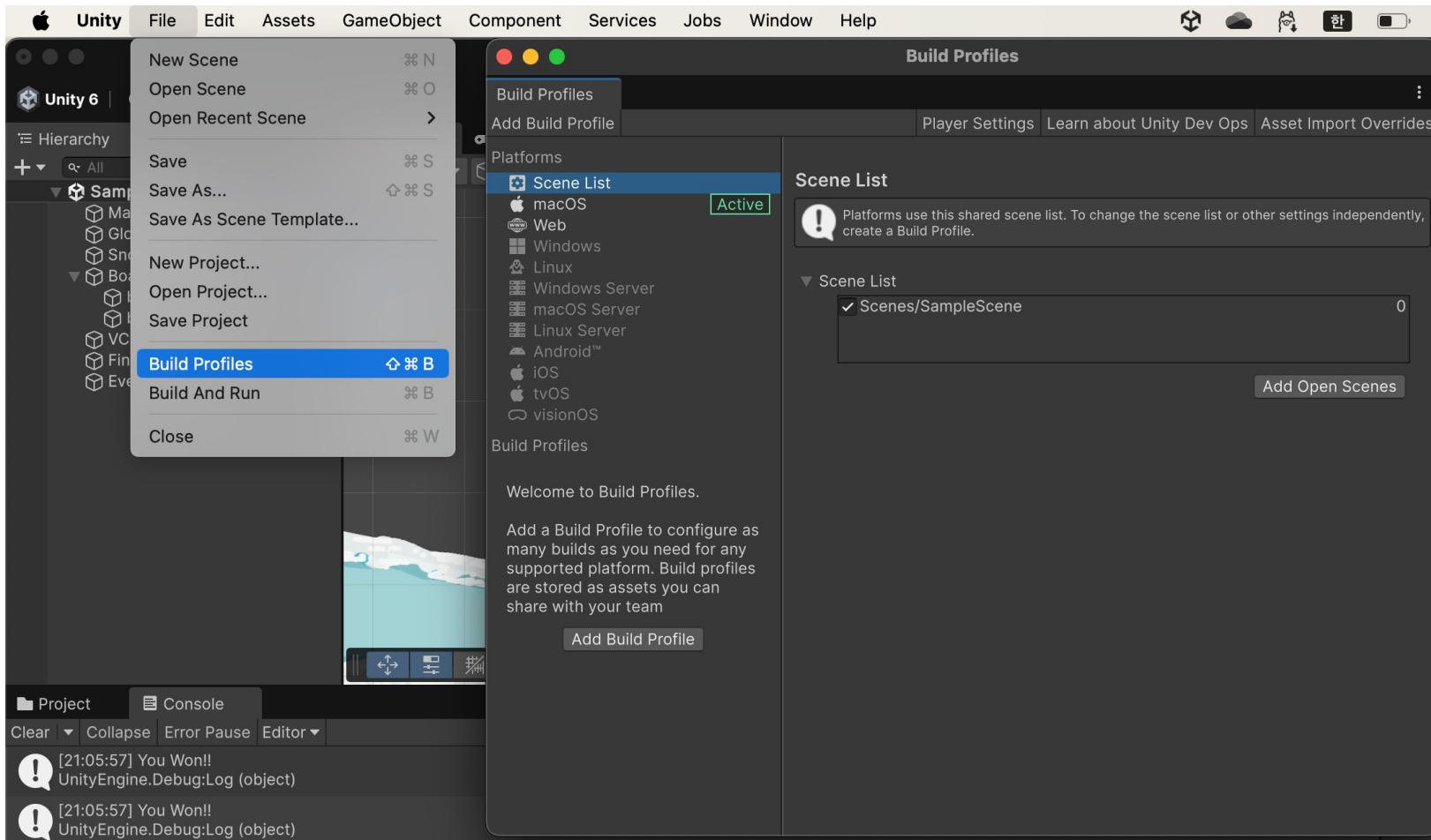
Name Space – UnityEngine.SceneManager

```
using UnityEngine;
using UnityEngine.SceneManagement;

0 references
public class Finish : MonoBehaviour
{
    0 references
    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "Player")
        {
            SceneManager.LoadScene();
        }
    }
}
```

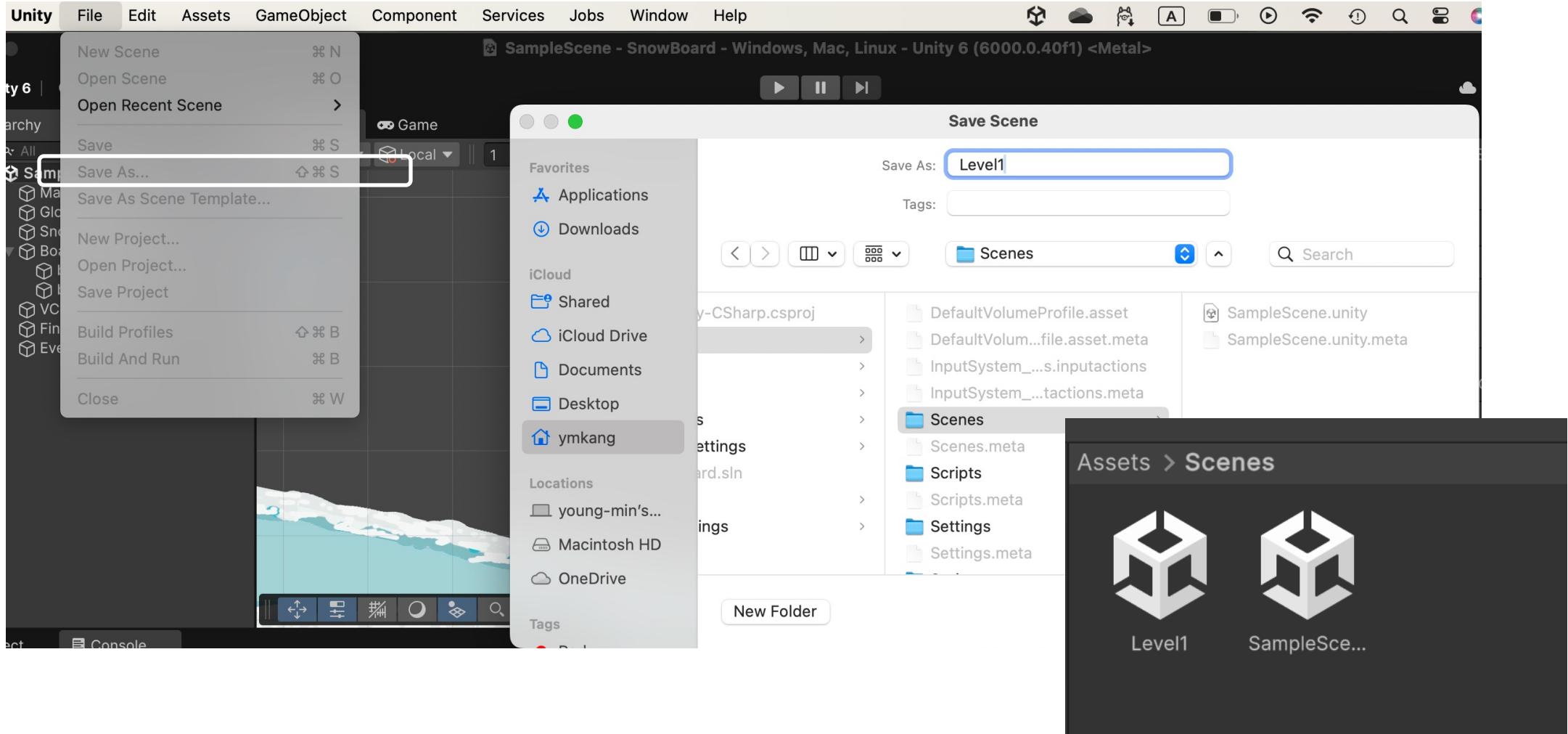
장면 전환을 위해 구현된 클래스를 사용할 수 있도록 특정 네임스페이스 사용

Scene 관리

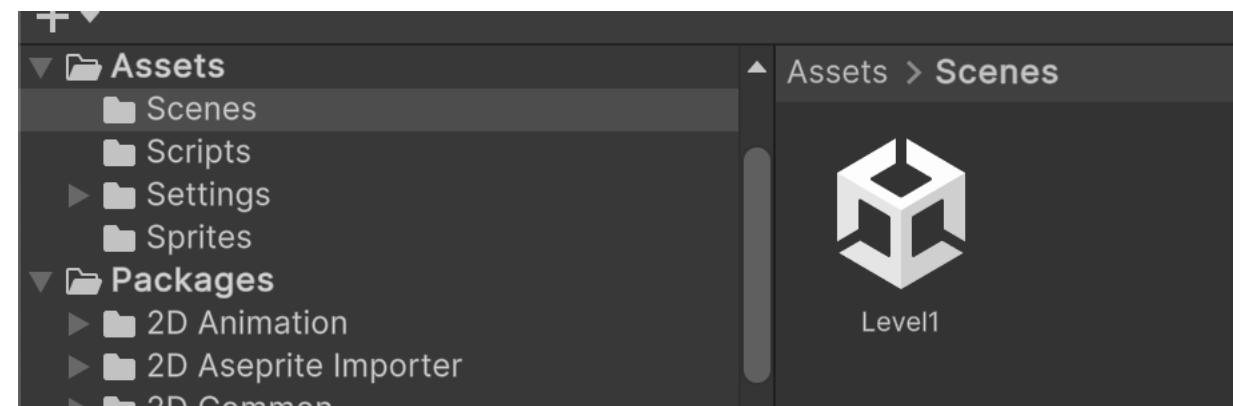
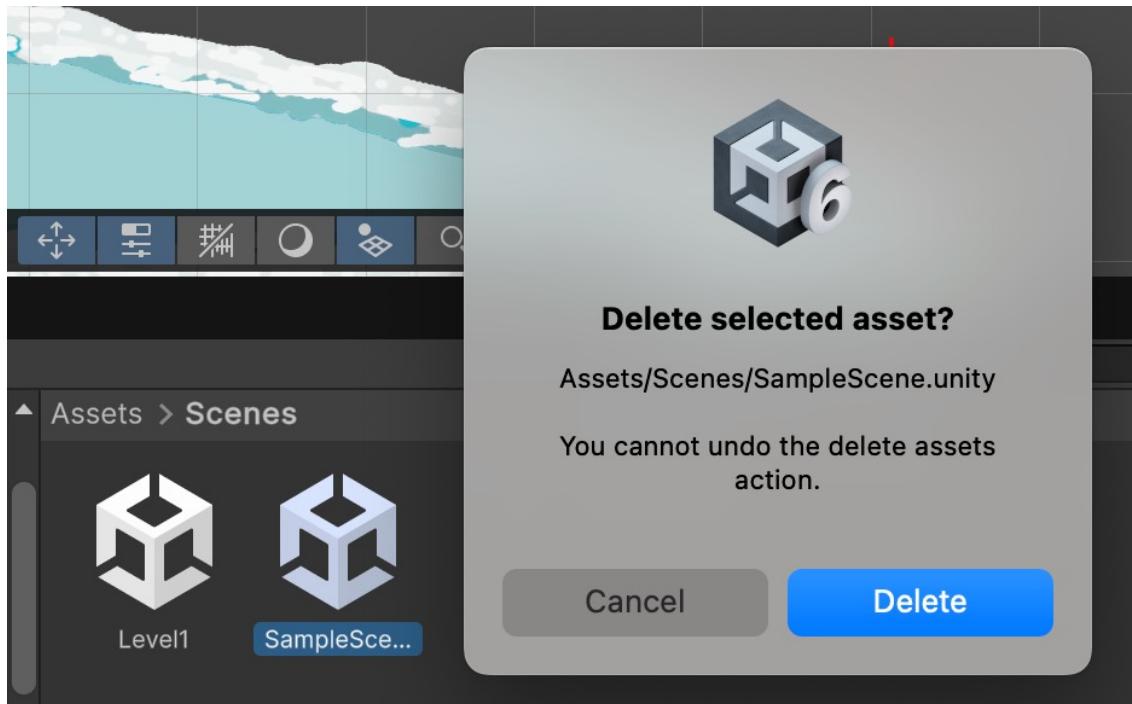


File → Build Profile → Scene List : SampleScene 하나가 있는 것을 확인

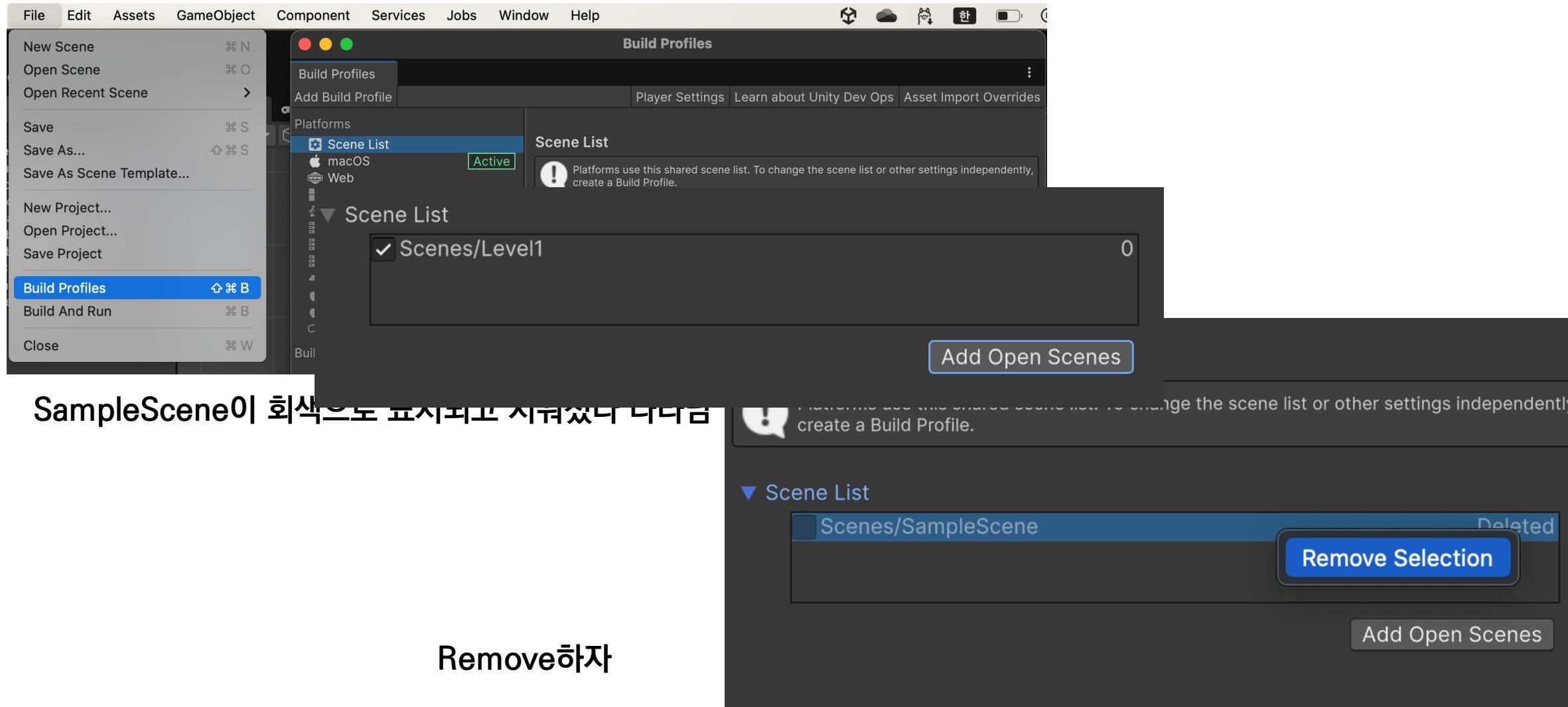
우리가 만든 장면을 Level1으로 저장하자



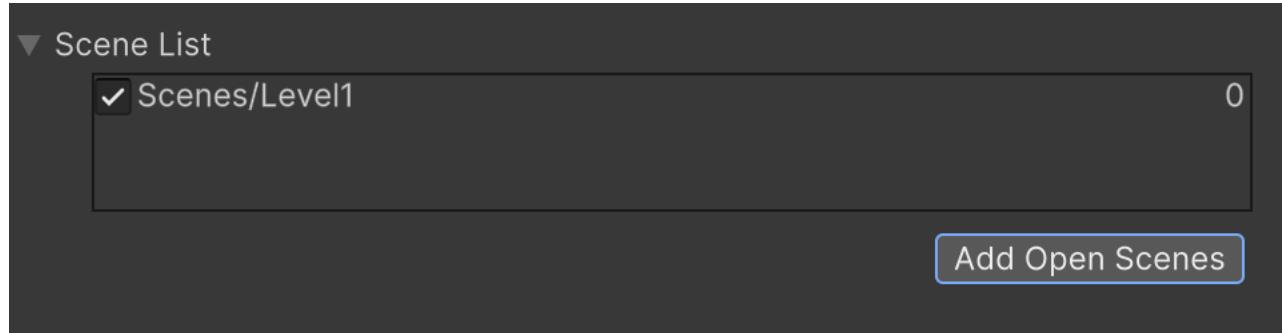
SampleScene은 삭제하자



Build Profile 다시 확인



Build Profile 다시 확인



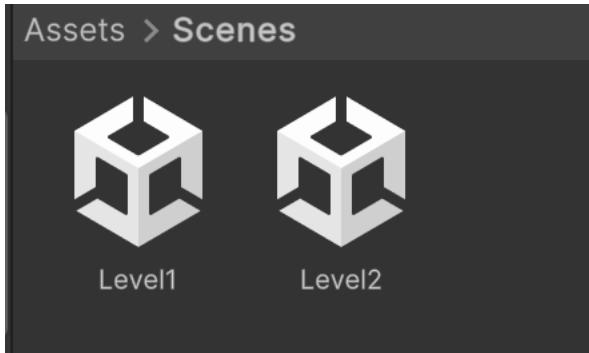
Add Open Scenes를 이용하여 저장한 장면을 장면 리스트에 추가

끝에 있는 번호는 장면 리스트의 인덱스로 이 장면은 0번 인덱스로 접근 가능

성공시 원래의 장면을 다시 로드

```
Assets > Scripts > C# Finish.cs > ...
1  using UnityEngine;
2  using UnityEngine.SceneManagement;
3
4      0 references
5  public class Finish : MonoBehaviour
6  {
7      0 references
8      void OnTriggerEnter2D(Collider2D other)
9      {
10         if (other.tag == "Player")
11         {
12             SceneManager.LoadScene(0);
13         }
14     }
```

성공시 다음 레벨로 가려면?



Level2를 복사하여 만들고, Build Profile에 장면 추가

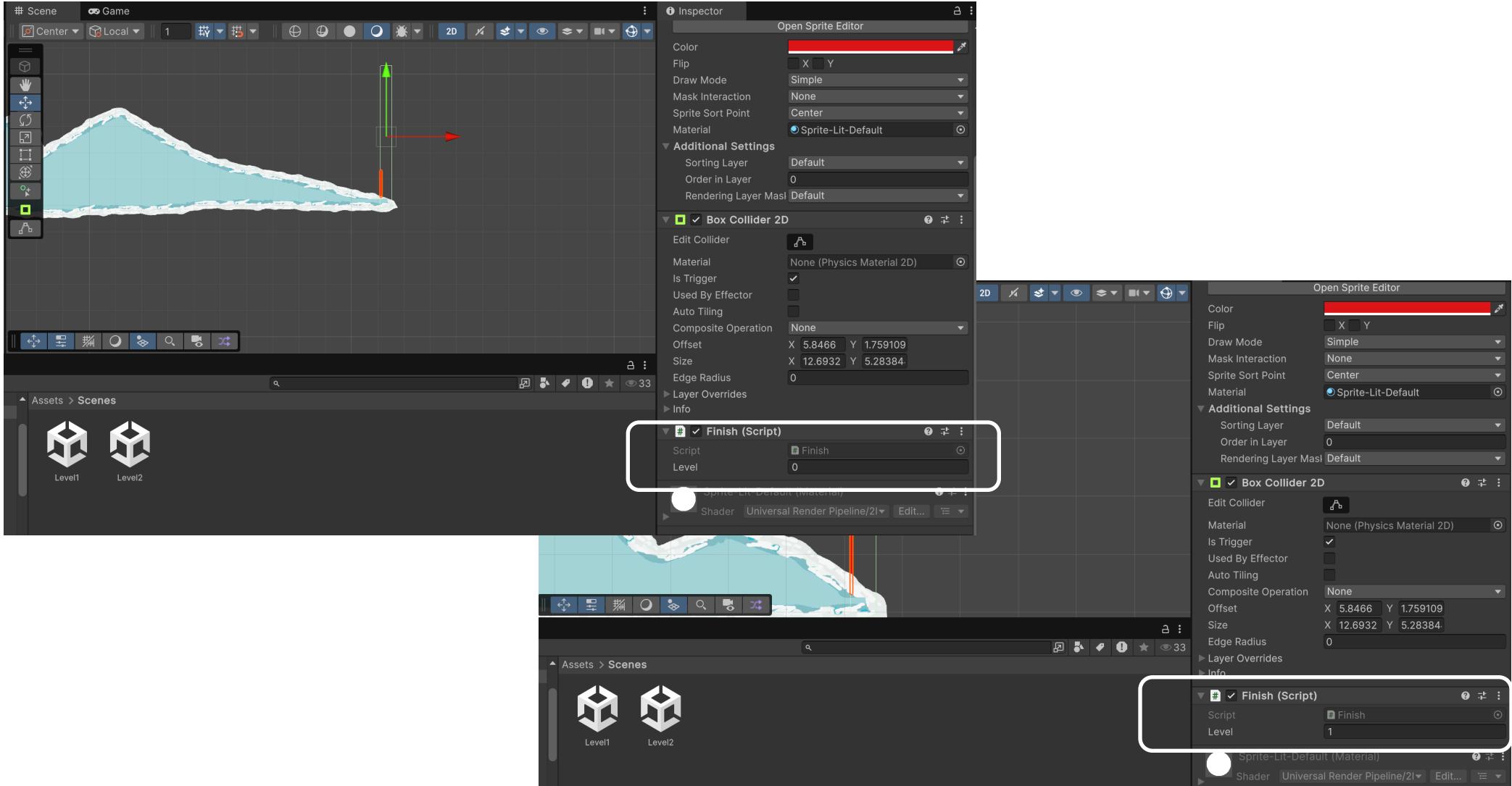
각 레벨의 모양을 변경

Finish 코드를 수정하여 레벨별로 다음 레벨을 호출하게 함

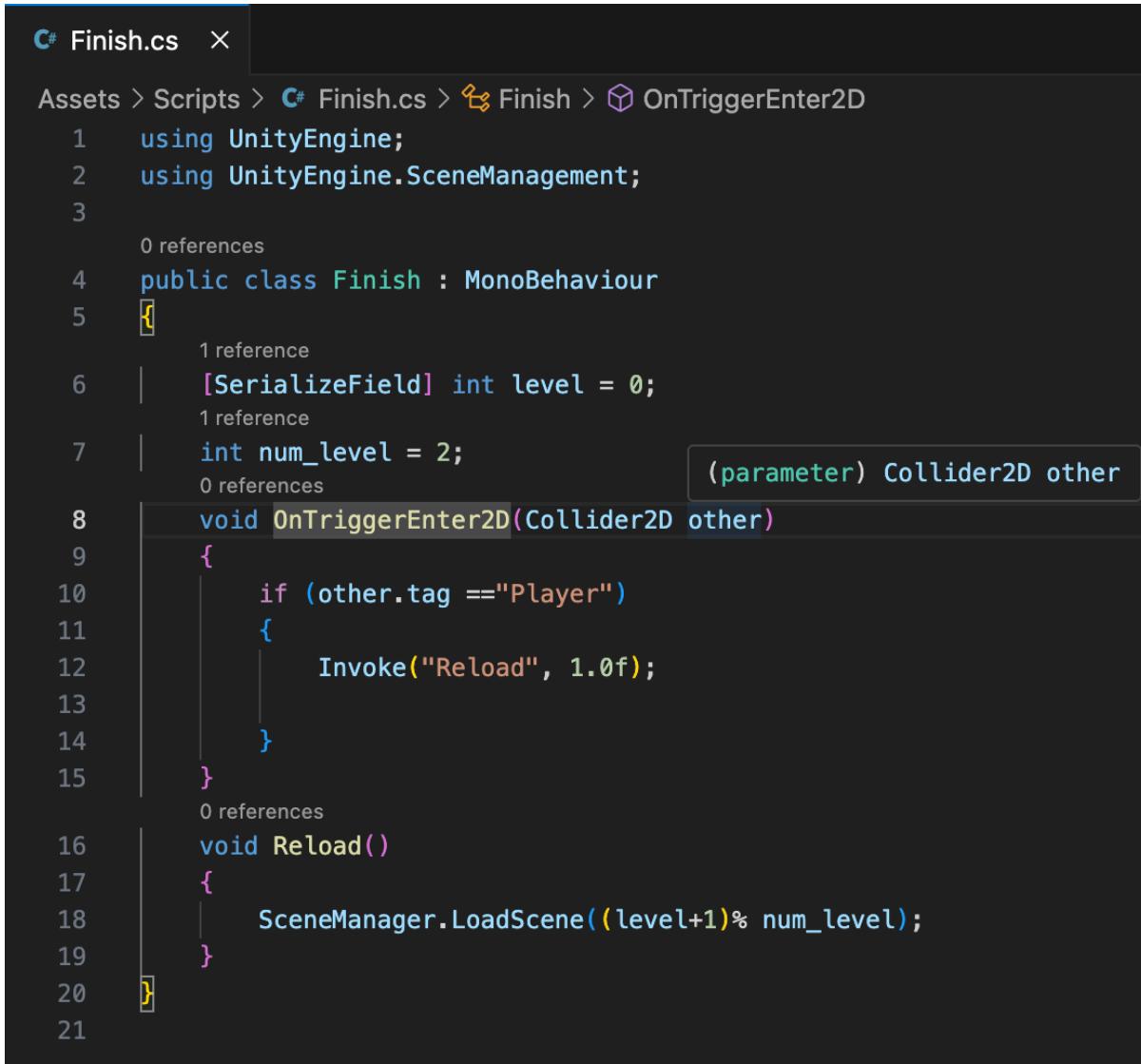
```
using UnityEngine;
using UnityEngine.SceneManagement;

public class Finish : MonoBehaviour
{
    [SerializeField] int level = 0;
    int num_level = 2;
    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "Player")
        {
            SceneManager.LoadScene((level+1)% num_level);
        }
    }
}
```

성공시 다음 레벨로 가려면?



약간의 자연을 발생시키기



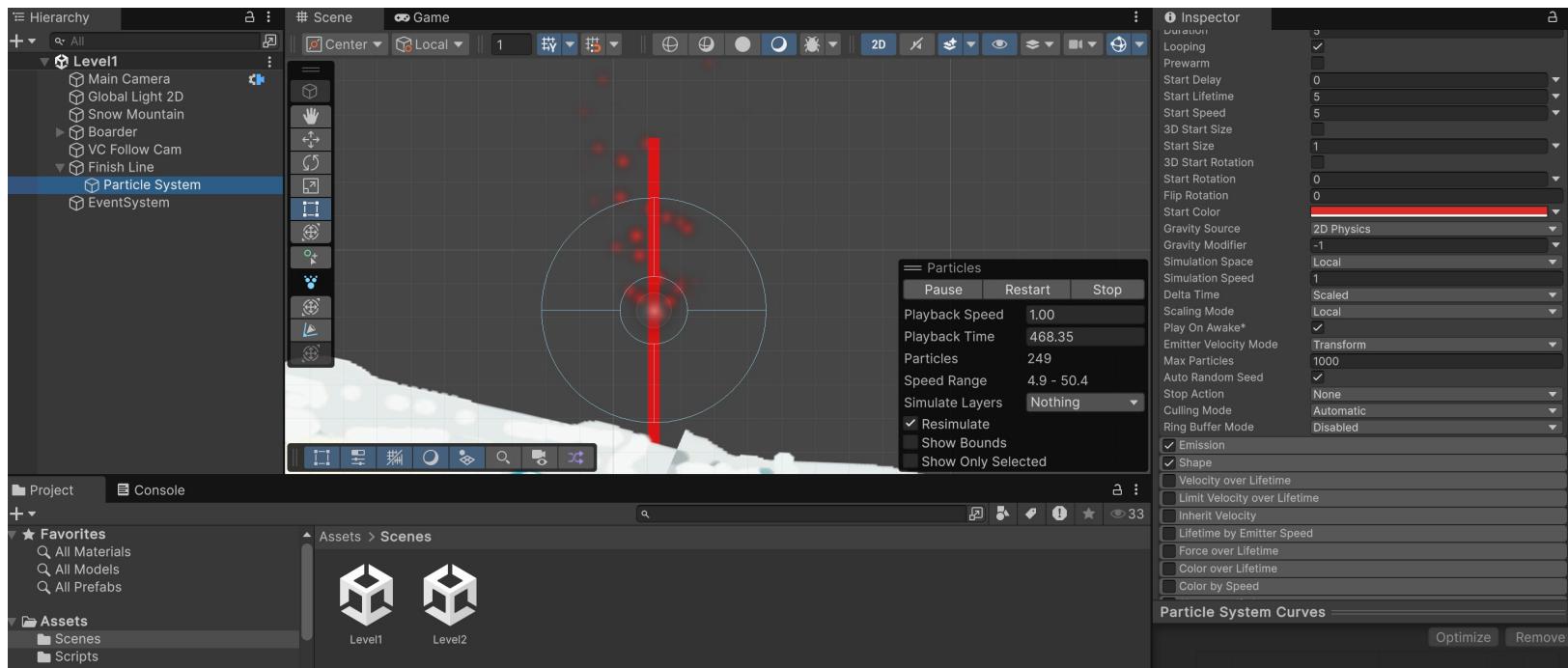
The screenshot shows a Unity code editor window titled "C# Finish.cs". The file path is "Assets > Scripts > C# Finish.cs > Finish > OnTriggerEnter2D". The code is as follows:

```
1  using UnityEngine;
2  using UnityEngine.SceneManagement;
3
4  public class Finish : MonoBehaviour
5  {
6      [SerializeField] int level = 0;
7      int num_level = 2;
8      void OnTriggerEnter2D(Collider2D other)
9      {
10         if (other.tag == "Player")
11         {
12             Invoke("Reload", 1.0f);
13         }
14     }
15 }
16 void Reload()
17 {
18     SceneManager.LoadScene((level+1)% num_level);
19 }
20 }
```

A tooltip for the "other" parameter in the `OnTriggerEnter2D` method is visible, stating "(parameter) Collider2D other".

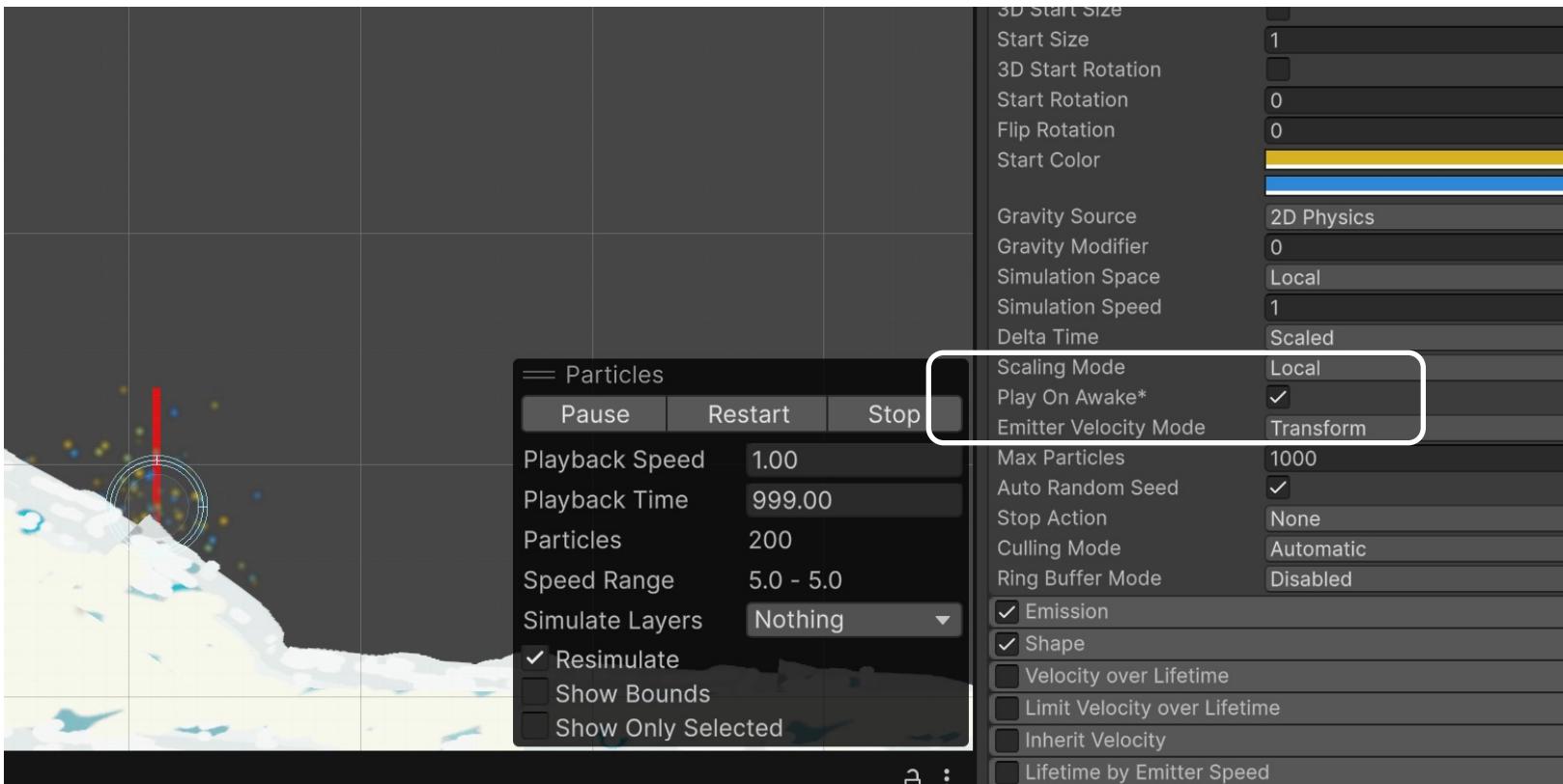
입자 시스템 (Particle System)

Create → Effect → Particle System을 Finish Line 객체에 달아 보자



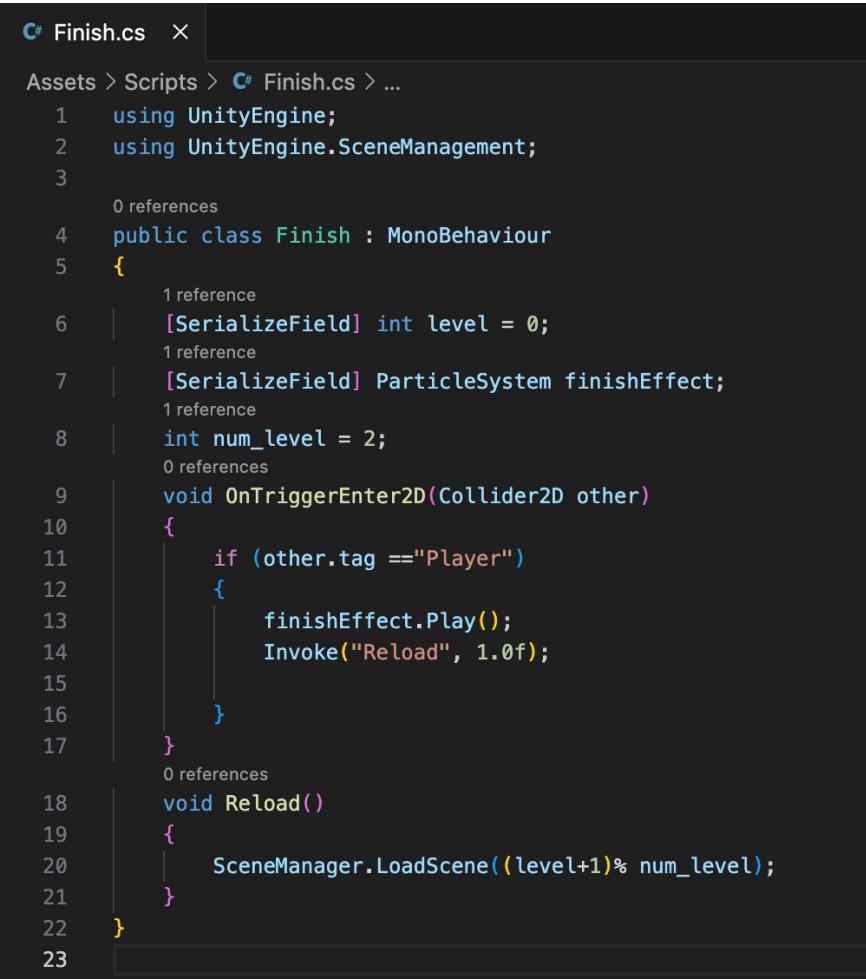
보더가 결승선을 통과할 때 입자 효과 발생시키기

입자 시스템의 Play on Awake 기능을 끈다: 자동으로 실행되지 않음



결승선 통과 감지 코드에서 입자 시스템 플레이

SerializeField를 이용하여 입자 시스템을 스크립트에서 사용할 수 있게 한다



The screenshot shows the Unity Editor's code editor with the file 'Finish.cs' open. The code defines a MonoBehaviour class named 'Finish' with two SerializeField properties: 'level' and 'finishEffect'. It includes methods for OnTriggerEnter2D and Reload.

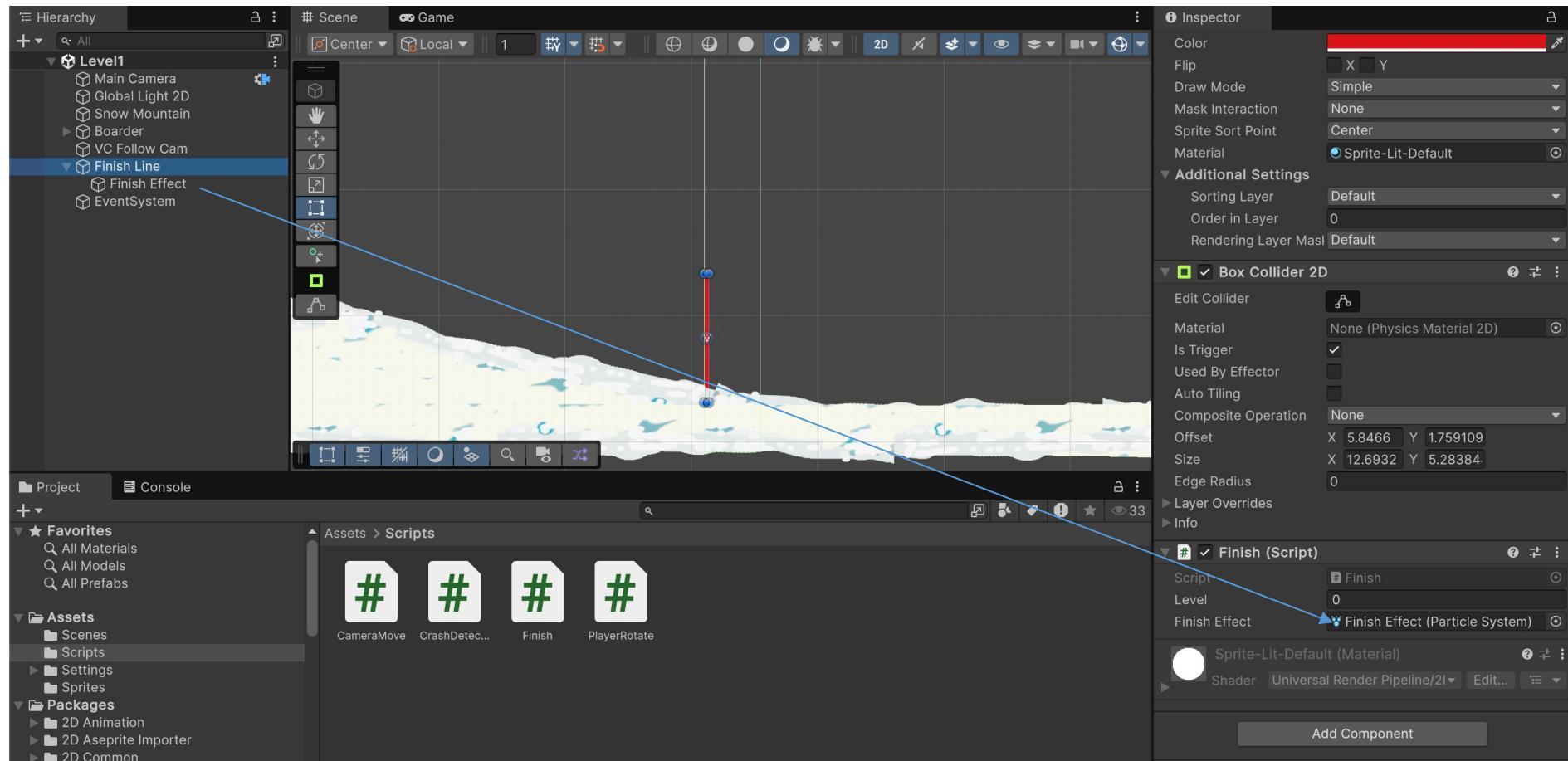
```
C# Finish.cs  ×

Assets > Scripts > C# Finish.cs > ...

1  using UnityEngine;
2  using UnityEngine.SceneManagement;
3
4  public class Finish : MonoBehaviour
5  {
6      [SerializeField] int level = 0;
7      [SerializeField] ParticleSystem finishEffect;
8      int num_level = 2;
9
10     void OnTriggerEnter2D(Collider2D other)
11     {
12         if (other.tag == "Player")
13         {
14             finishEffect.Play();
15             Invoke("Reload", 1.0f);
16         }
17     }
18     void Reload()
19     {
20         SceneManager.LoadScene((level + 1) % num_level);
21     }
22 }
```

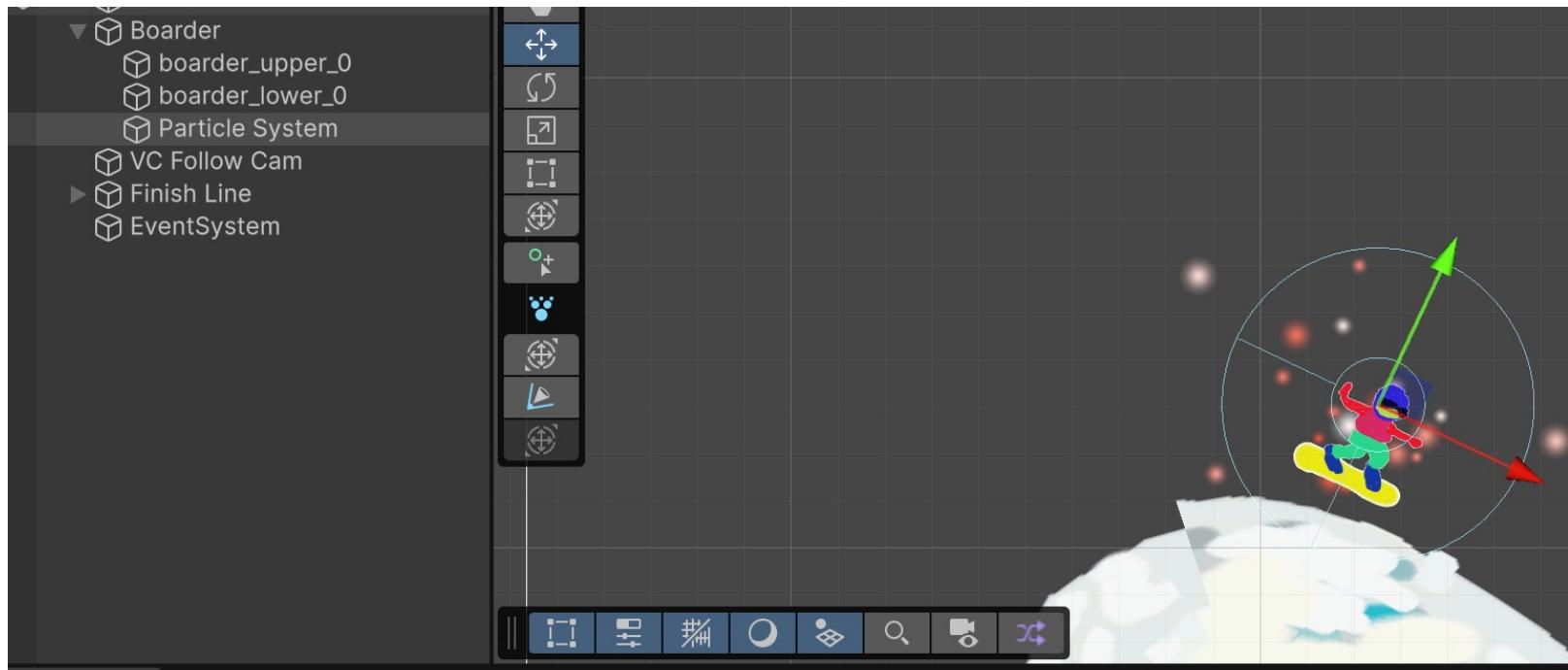
결승선 통과 감지 코드에서 입자 시스템 플레이

SerializeField 변수를 입자시스템과 연결 (Level1, Level2 모두)



보더의 머리가 바닥에 닿았을 때 효과를 줘보자

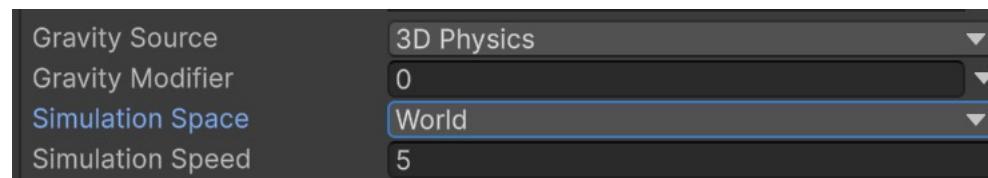
머리 근처에 파티클 시스템 설치



Play on Awake 등
앞에서 설정한 것들을 반복하고

충돌에 맞게 입자 효과 조정

시뮬레이션 공간을 전역(World)로 하면 입자가 분출된 뒤 캐릭터의 움직임에 영향받지 않음





게임에 가까워 지나요?