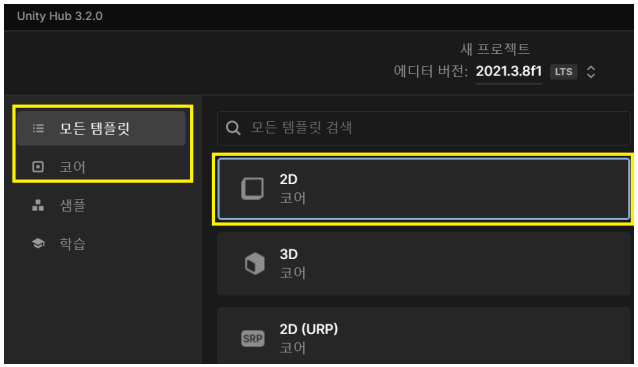
Unity
Uni-Run

참고:레트로의 유니티 게임 프로그래밍 에센스

목차

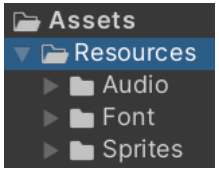
목차

프로젝트 준비

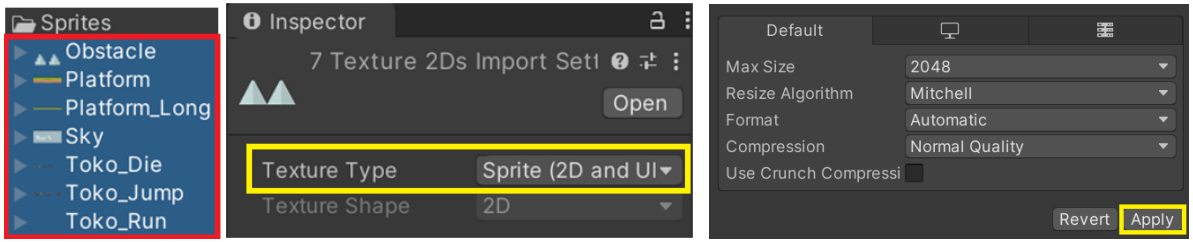


- **새 프로젝트 생성**
  - 새로 만드는 **프로젝트**의 템플릿을 **2D**로 선택

- **리소스 추가**
  - 프로젝트에 **Resources 폴더 생성**
  - **예제소스 폴더 => Uni-Run 폴더**에 있는 폴더 및 파일을 전부 **Resources 폴더에 복사**



- **Sprites**
  - Sprites 폴더의 **이미지 리소스들**의 **Texture Type**을 **Sprites(2D and UI)로 변경 후 [Apply]**



Game Manager

- **GameMgr**
  - Scene에 GameManager가 없다면 새로 만든다
  - Instance가 자신과 다르면(다른 GameManager 이미 존재) 자신을 Scene에서 제거

```
using UnityEngine.InputSystem;
public class GameMgr : MonoBehaviour
{
    static GameMgr instance = null;
    public static GameMgr Instance
    {
        get
        {
            if (null == instance)
            {
                instance = FindObjectOfType<GameMgr>();
                // new GameObject() : Scene에 새로운 GameObject를 생성.
                if (!instance) instance = new GameObject("GameManager").AddComponent<GameMgr>();

                instance.Initialize(); // 멤버 함수 Initialize()를 호출하여 Input System 초기화.
                DontDestroyOnLoad(instance.gameObject);
            }

            return instance;
        }
    }

    InputActionAsset inpuAsset;
```

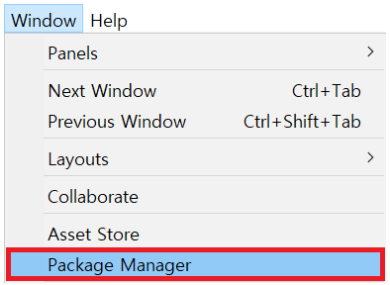
Game Manager

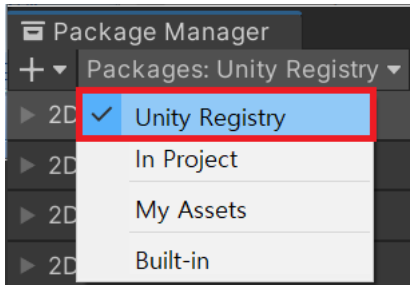
```
    private void Awake()
    {
        if (this != Instance)
        {
            Destroy(gameObject);
            return;
        }
    }

    void Initialize()
    {
        // InputActionAsset은 ScriptableObject로 만들어야 한다.
        inpuAsset = ScriptableObject.CreateInstance<InputActionAsset>();
        InputActionMap actionMap = inpuAsset.AddActionMap("Player");
        // 점프 입력은 Space Bar 또는 마우스 오른쪽 버튼을 사용.
        actionMap.AddAction("Jump", binding: "<Keyboard>/space");
        actionMap.AddBinding("<Mouse>/leftButton", action:"Jump");
        // Action Map을 활성화 해야 바인드한 입력이 사용 가능.
        actionMap.Disable();

        gameObject.AddComponent<PlayerInput>().actions = inpuAsset;
    }
} // class GameMgr
```

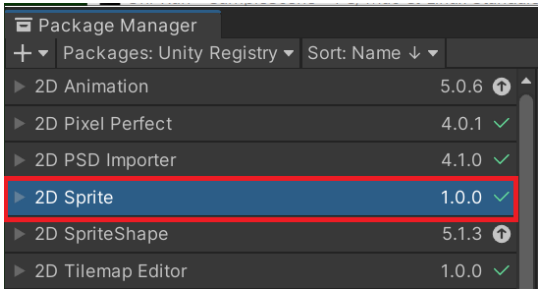
2D Sprite

2D Sprite Pakage

- Unity에서 2D Sprite를 그리는데 사용하는 도구 들의 모음
- 2D 기능을 **활용**하기 위해서는 반드시 추가 해야 하는 Package







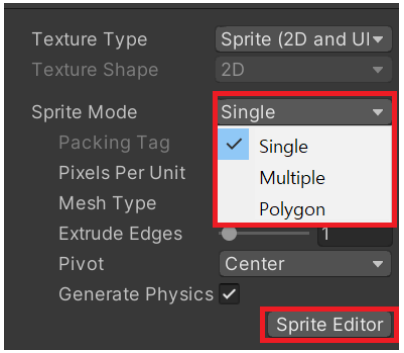
Draw Call

- **CPU가 GPU에게 그림을 그려달라는 요청**
- 드로우 콜이 **많이 발생할 수록 게임이 무거워지며 프레임 저하가 발생**하게 된다
- 2D 환경에서 드로우 콜을 줄이기 위해서 이미지 파일의 수를 줄이는 것이 중요
- **Image Sprites**를 사용하거나 Unity의 **Sprite Atlas**를 이용하여 **여러 이미지를 하나의 이미지로 취합**이 가능

Batch

- GPU에 의해 접근되는 리소스를 비슷한 것 끼리 묶어 처리하는 **렌더링 기법**
- **드로우 콜을 포함하는 상위 개념**
- ex) 하나의 오브젝트에 두 개의 Mesh를 이용하여 그리면 드로우 콜은 두 번 발생, 두 Mesh를 하나의 배치로 묶을 경우 드로우 콜은 한 번만 발생하게 된다

2D Sprite



Sprite Mode

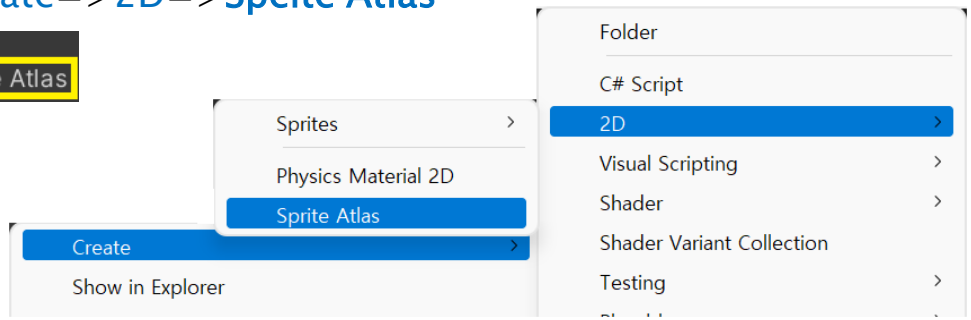
- Single : **Sprite 이미지**를 있는 그대로 **하나의 이미지로 사용**
- Multiple : **하나의 Sprite 이미지**를 **여러 개의 요소로 나누어 사용**
- Polygon : Sprite 이미지를 Mesh 처럼 만들어 사용

Sprite Editor

- **Multiple 또는 Polygon 모드**를 사용할 경우 **Sprite 이미지를 편집**하는데 사용

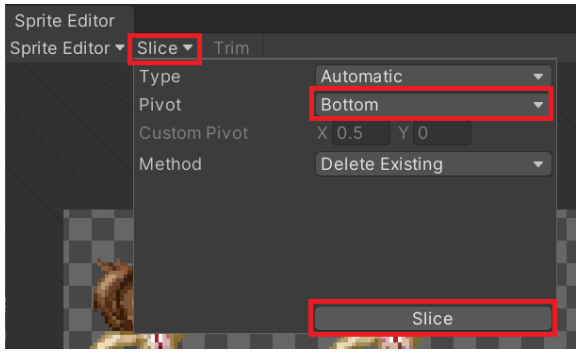
Sprite Atlas

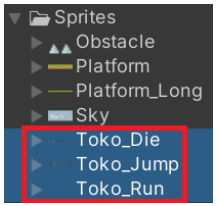
- 여러 개로 나누어 있는 이미지를 하나로 합쳐 하나의 배치로 만든다
- Asset=>Create=>2D=>Speite Atlas

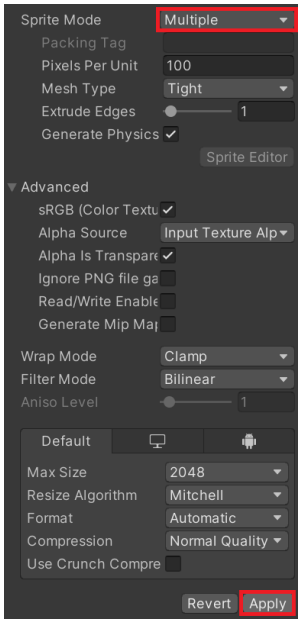


2D Sprite

- **Multiple Sprites**
  - Toko_Die, Toko_Jump, Toko_Run 세 Sprite 이미지를 **Sprite Mode를 Multiple로 변경**
  - Sprite의 Size를 키우기 위해서 **Pixel Per Unit**의 값을 **40으로 변경**(값이 작을수록 커진다)
  - Sprite Editor에서 Slice의 **Pivot을 Bottom으로 변경**하고 Slice 실행

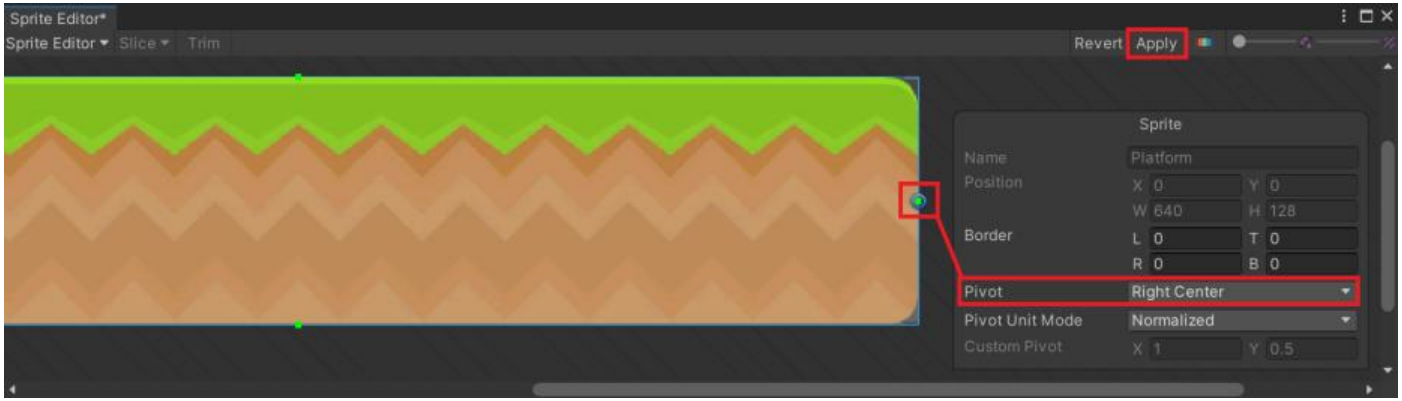






2D Sprite

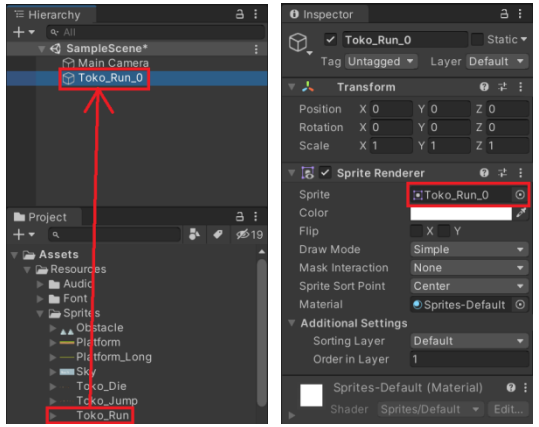
- **Pivot**
  - Platform, Platform_Long 이미지 **Sprite Editor** 선택
  - **Pivot**을 **RightCenter**로 변경



Sprite Animation

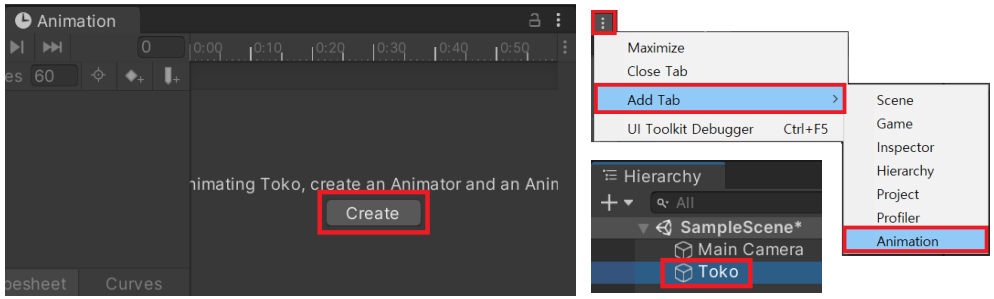
2D GameObject

- **Sprite로 설정한 이미지** 파일은 **하이어라키(Hierachy)로 등록이 가능**하다
- Toko_Run 이미지를 **하이어라키(Hierachy)로 드래그&드롭**, **자동으로** 잘라둔 이미지의 **첫 번째 이미지로 설정**된다
- 오브젝트 이름을 'Toko'로 변경
- Position : (-3, 0, 0)



Animation

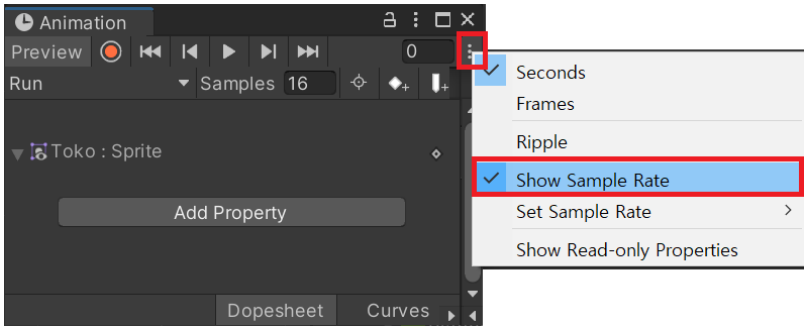
- **애니메이션**을 만들고 **편집**은 **Animation 창**에서 한다
- Window=>Animation=>Animation
- 또는 각 창의 오른쪽 상단의 [...]=>Add Tab=>Animation
- 애니메이션을 만들기 위한 **게임 오브젝트 선택**
- **Animation 창**의 **Create 버튼**을 선택

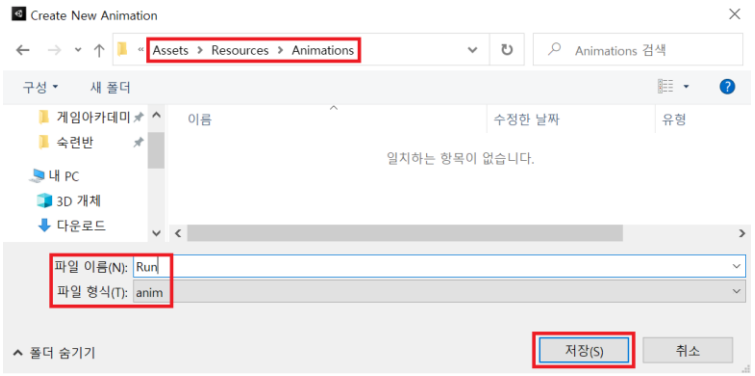


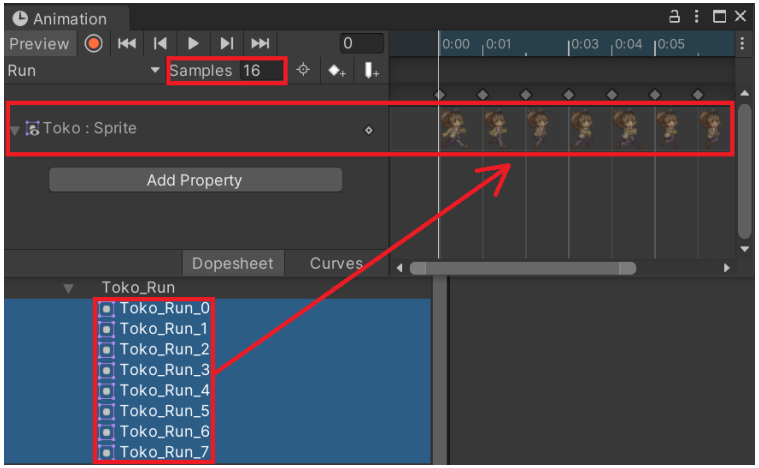
Sprite Animation

Player Animation

- Resources 폴더에 **Animations 폴더 추가**
- **'Toko'를 선택**하여 **Animation Create** 를 선택
- Animations 폴더에 **'Run'**으로 생성
- **'Toko_Run'** 이미지를 펼쳐 **Sprite를 전부 선택**
- Animation 창으로 **드래그&드롭**
- Animation 창 오른쪽 위의 [...]=>Show Sample Rate 체크
- **Samples** 값을 **16으로 변경**

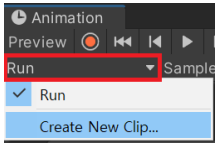




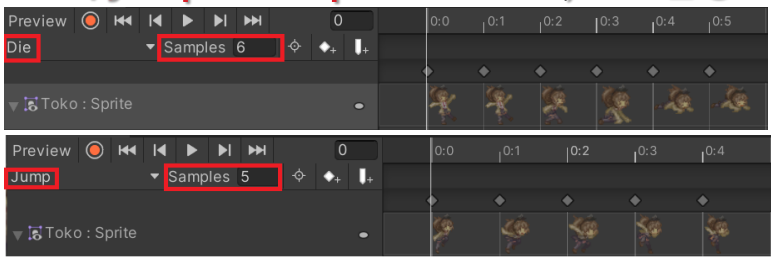


Sprite Animation

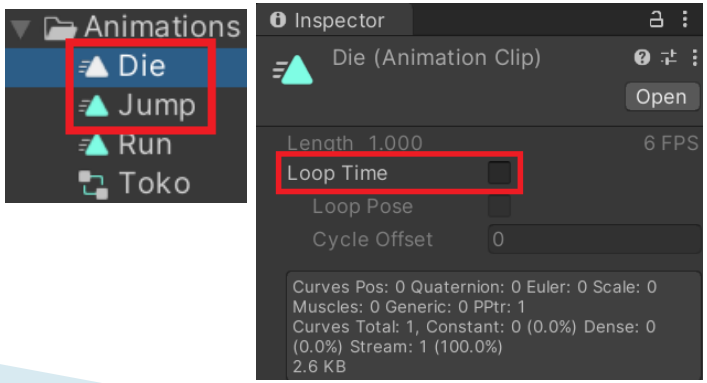
- Create New Clip…=>Die, Jump 추가



- **각 Animation Clip에 Sprite이미지 등록**
- **Die와 Jump의 Samples에 각각 6, 5로 변경**



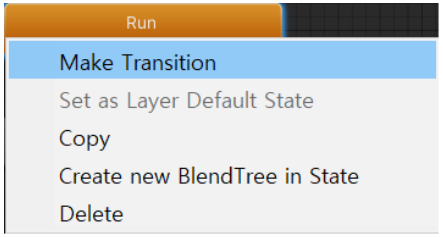
- 두 애니메이션은 **한 번만 재생**해야 되기 때문에 위해 **Animation Clip**을 선택하여 각각 **Loop Time 체크를 해제**

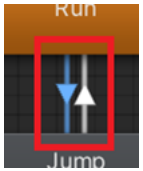


Animator

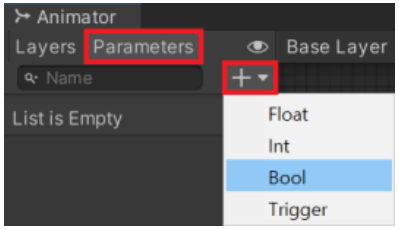
- ## Animator Controller

- Window=>Animation=>Animator 또는 Animator Controller 더블 클릭
- **애니메이션 클립을 만들면 자동으로 생성**, 추가 된다
- **애니메이션** 전환 및 정렬하고 **관리**할 수 있다
- **Make Transition**을 선택하여 **다른 애니메이션과 연결**, 조건을 설정하여 실행 중 애니메이션을 **다른 애니메이션으로 상태 변경**이 가능하다

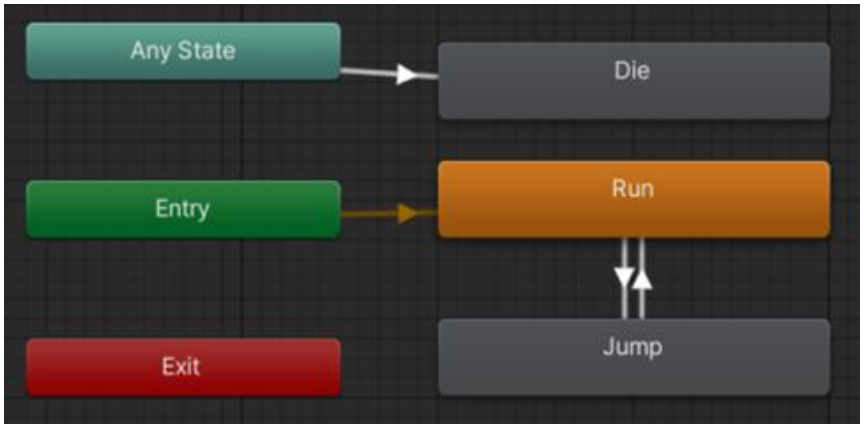




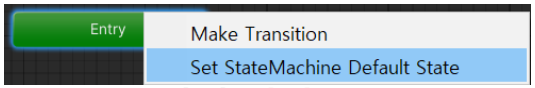
- **Parameters**를 [+]로 조건 **파라메터를 추가**할 수 있다



Animator



- 유한상태기계(FSM)을 기반으로 만들어져 있다



- **Entry :상태의 시작**
- 애니메이션을 실행하면 **연결된 애니메이션 클립(Default State)이 자동으로 실행**된다
- 제일 처음 적용된 애니메이션 클립이 Default로 적용된다
- **Set State Machine Default State**로 **Default 애니메이션을 변경**할 수 있다

- **Any State**
- Transition에 설정한 조건이 된다면 **어느 상태의 애니메이션이든 상관하지 않고 연결된 상태의 애니메이션으로 변경**한다

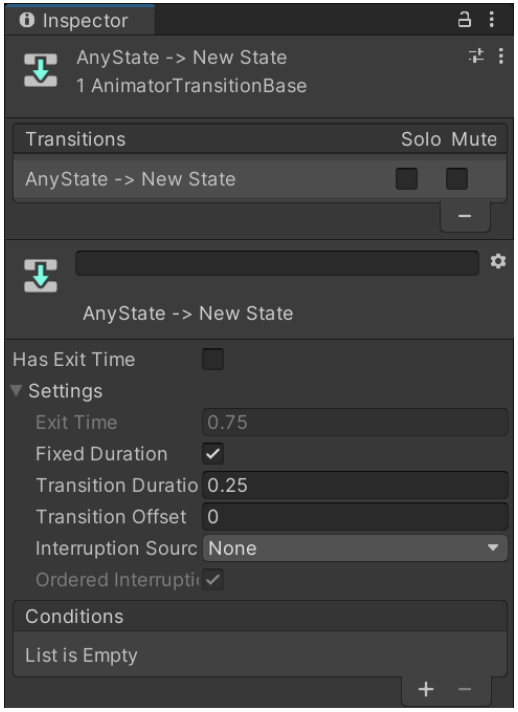
- **Exit : 현재 상태의 종료**
- **Exit로 연결된 애니메이션**(Any State Animation->Exit)이 끝나면 **Entry 상태로 변경**된다

Animator

Transition Inspector

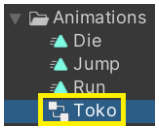
- Has Exit Time : **Exit Time을 참조**하여 해당 시간에 **전환 조건이 적용**되도록 한다
- Exit Time : **Has Exit Time이 활성화 인 경우 사용**(0.75 == 75%)
- Fixed Duration : **Transition Duration**의 값을 **초 단위**(true) 또는 **정규화된 단위**(false)로 사용
- Transition Duration : **애니메이션**을 부드럽게 **전환하기 위한 시간**(2D에서는 의미가 없다)
- Trasition Offset : 전환된 **애니메이션의 시작 지점**의 시간
- Conditions : **전환 조건**

  조건이 없다면 Exit Time을 참조하여 시간이 되면 곧 바로 상태(애니메이션)를 전환한다

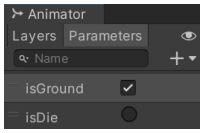


Animator

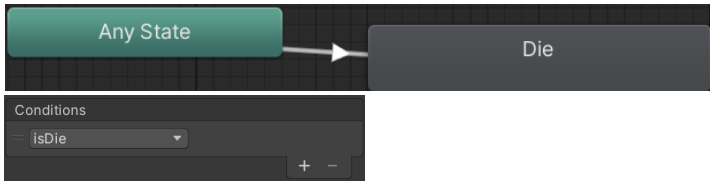
- **애니메이션 상태 편집**
  - 'Toko'의 **Animator Controller** 선택



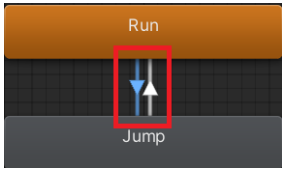
  - Parameters
    - isGround(Bool) : default True
    - isDie(Trigger)

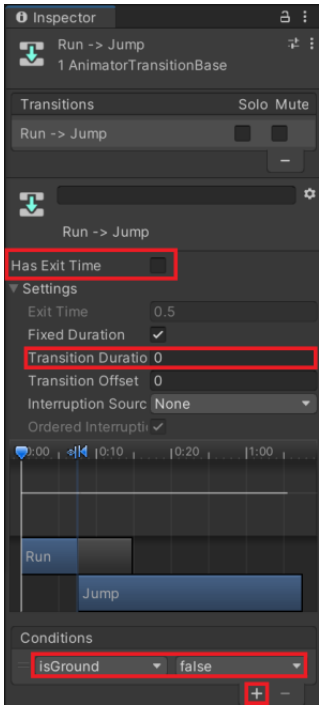


  - **Any State**에 **Die**를 **Transition으로 연결**
  - **Any State->Die** Transition 선택
  - **Has Exit Time 체크 해제**
  - **Transition Duration : 0**
  - Conditions 추가 **isDie**



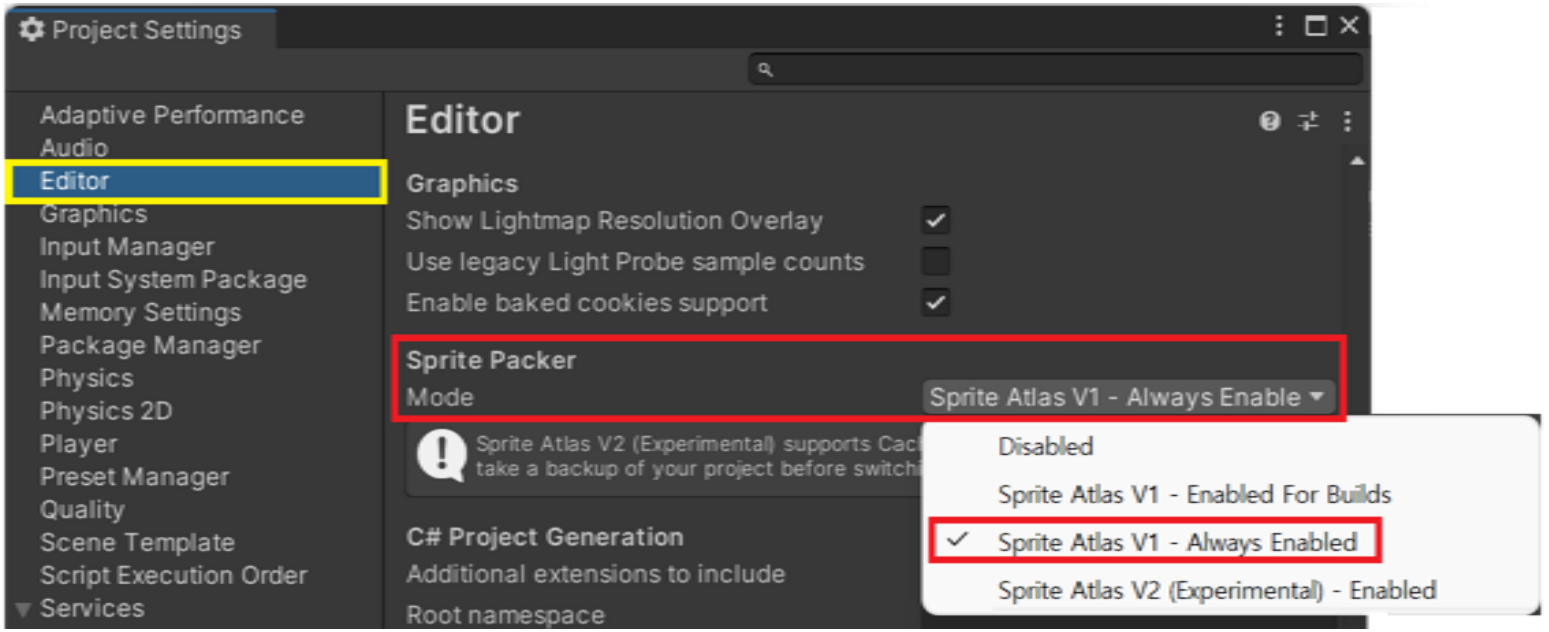
  - **Run과 Jump**에 서로 **Transition으로 연결**
  - 각 **Transition**(하얀색 화살표)을 **선택**
  - **Has Exit Time 체크 해제**
  - **Transition Duration : 0**
  - **Run->Jump** Conditions 추가 **isGround : false**
  - **Jump ->Run** Conditions 추가 **isGround : true**





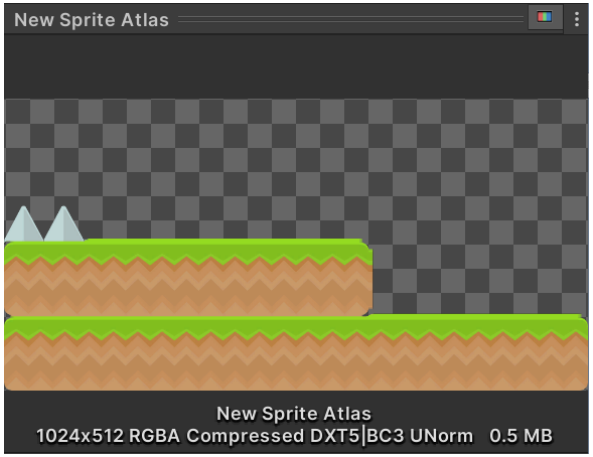
Sprites Atlas

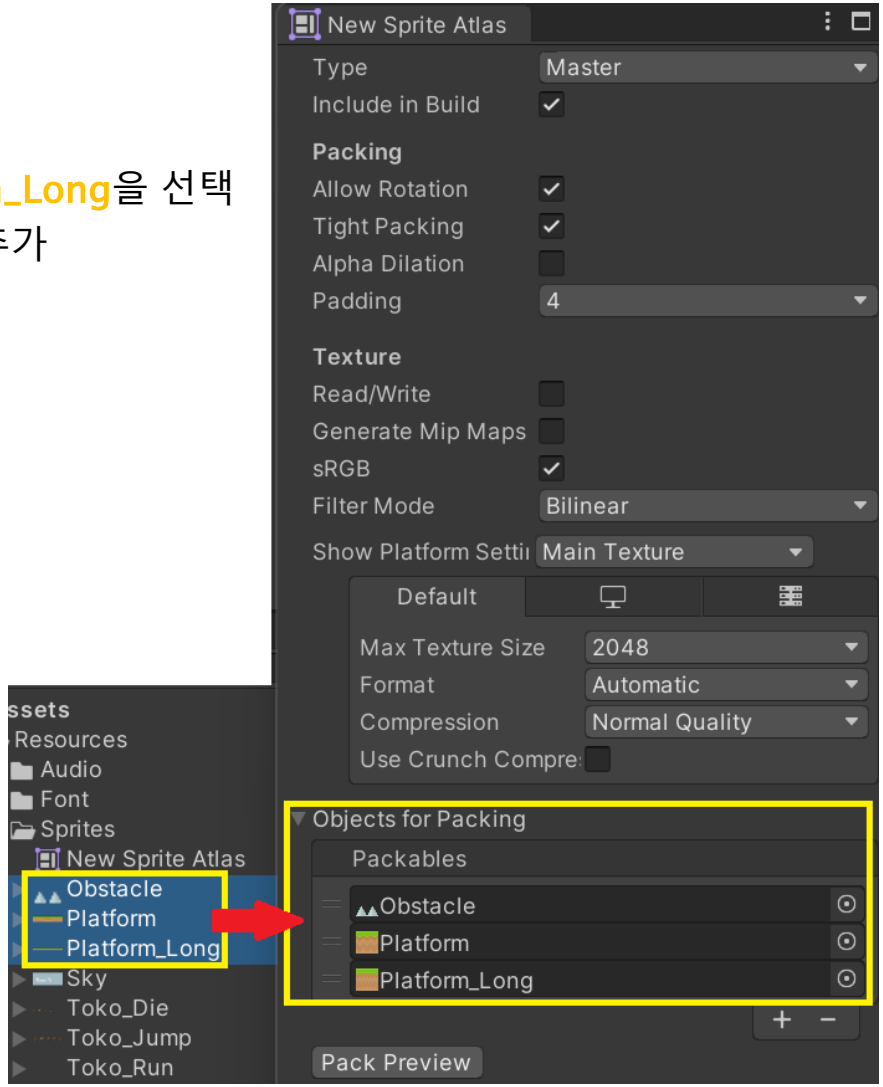
- **Sprites Atlas 만들기**
- Sprite Atlas를 사용하기 위해서 세팅 변경
- Edit=>Project Setting=>Editor=>Sprite Packer=>Model:Speite Atlas V1 – Always Enable



Sprites Atlas

- **새 Sprite Atlas 만들기**
- Sprites 폴더=>Obstacle, Platform, Platform_Long을 선택
- New Sprite Atlas=>objects for Packing에 추가
- [Pack Preview]로 합쳐진 이미지 확인 가능



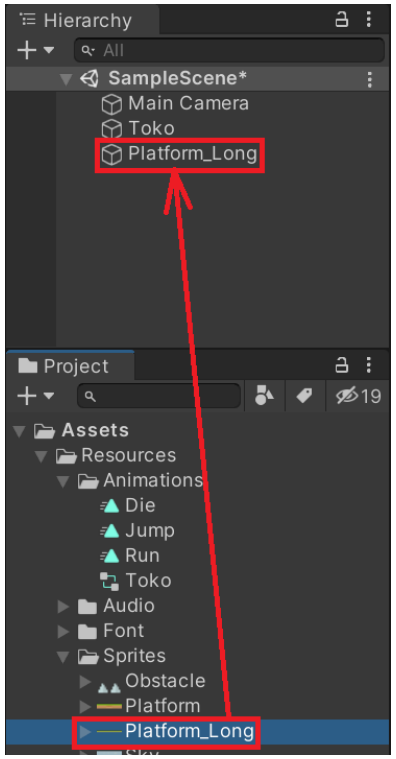


Run & Jump

- **Platform_Long**
  - Box Collider 2D 추가
  - Position : (5, -1, 0)

- **Toko**
  - **Scripts 폴더** 추가
  - C# **PlayerController** 생성, **Toko**에 추가 (AddComponent)

```
[RequireComponent(typeof(BoxCollider2D))] // 코드 작성 후 Script를 AddComponent해야,
[RequireComponent(typeof(Rigidbody2D))]   // 자동으로 컴포넌트들이 추가된다.
public class PlayerController : MonoBehaviour
{
    Rigidbody2D rigid;
    Animator anim;

    [Header("Move")]
    [SerializeField] float speed = 5.0f;
    [SerializeField] float jumpForce = 3.0f;

    readonly int limitJumpCount = 2;
    int jumpCount = 0;

    public float Speed { get { return speed; } }
```



Run & Jump

```
void Start()
{
    rigid = GetComponent<Rigidbody2D>();
    if (rigid) rigid.freezeRotation = true;

    anim = GetComponent<Animator>();
}

public void Jump(InputAction.CallbackContext context)
{
    if (!rigid) return;
    // started : 키, 버튼을 누른 프레임에 true
    if (context.started && (limitJumpCount > jumpCount))
    {
        jumpCount++;

        rigid.velocity = Vector2.zero;
        rigid.AddForce(Vector2.up * jumpForce * 100.0f);
    }
    // canceled : 키, 버튼을 땐 프레임에 true
    else if (context.canceled && (0 < rigid.velocity.y))
    {
        rigid.velocity *= 0.5f;
    }
}
```

Run & Jump

```
    private void OnCollisionEnter2D(Collision2D collision)
    {
        // normal : 충돌 지점(Platform)의 법선 벡터.
        if (collision.contacts[0].normal.y > 0.8f)
        {
            // animator에 설정해둔 Parameters를 변경.
            if (anim) anim.SetBool("isGround", true);
            // 땅에 닿으면 점프 카운터를 초기화.
            jumpCount = 0;
        }
    }

    private void OnCollisionExit2D(Collision2D collision)
    {
        if (anim) anim.SetBool("isGround", false);
    }
} // class PlayerController
```

Run & Jump

```
public class GameMgr : MonoBehaviour
{
    ...
    PlayerController player;

    private void Awake()
    {
        ...
        SetupInput ();
    }

    bool SetupInput()
    {
        player = FindObjectOfType<PlayerController>();
        if (player)
        {
            inpuAsset["Jump"].started += player.Jump;
            inpuAsset["Jump"].canceled += player.Jump;
            inpuAsset["Jump"].Enable();
            return true;
        }
        return false;
    }
}
```

Sprite Draw Layer

2D 그리기

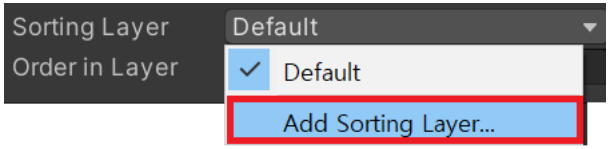
- 2D는 **깊이 값(z)**을 **사용하지 않는다**
- 다 같은 깊이(z)에 있기 때문에 **추가한 이미지에 의하여 보이지 않는 경우가 발생**한다
- Draw **Layer**를 설정하여 **그리는 순서**를 지정한다

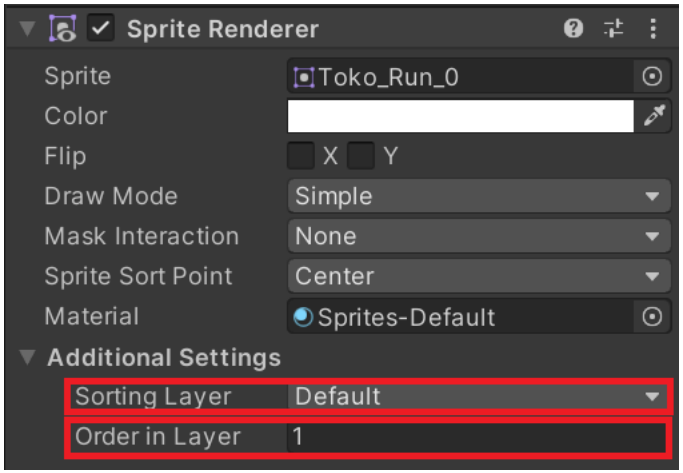
Sorting Layer

- **기본**적으로 **Default** Layer 하나밖에 없기 때문에 **같은 Layer(같은 깊이)에서 그려진다**
- **[Add Sorting Layer]**로 **Layer를 추가** 할 수 있다

Order in Layer

- Sorting Layer가 같은 경우 **그리는 우선 순위**

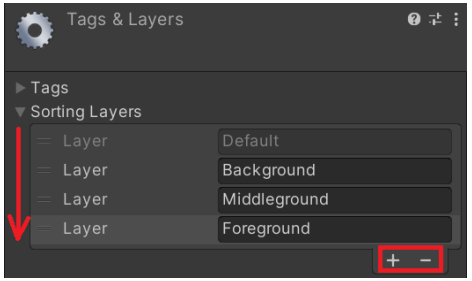


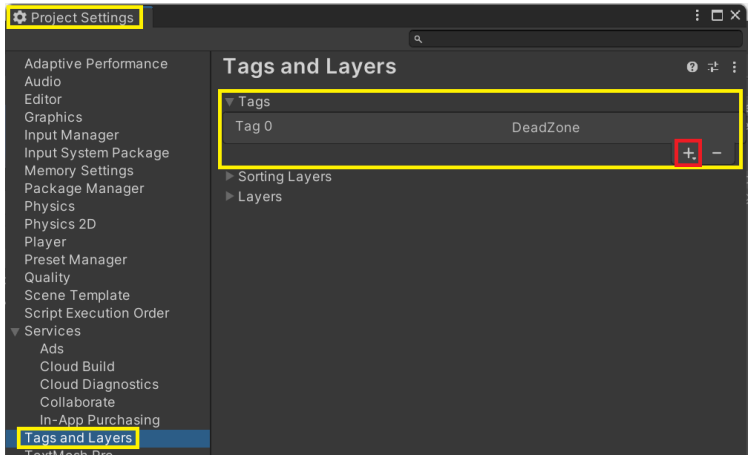


Sprite Draw Layer

- **Tag, Layer 추가**
  - Project Setting=>Tags and Layers=>Tags/Sorting Layers
  - 또는 Sprite Renderer의 Sorting Layer=>Add Sorting Layer를 선택
  - **[+], [-] 버튼**으로 **추가**, **삭제**가 가능

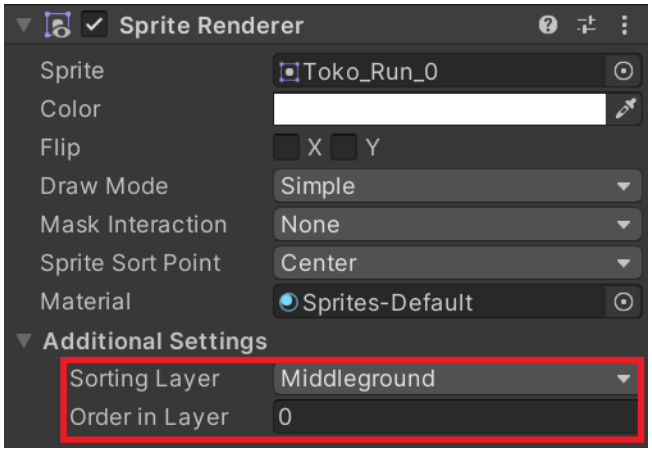
- **Tags**
  - DeadZone **추가**
- **Sorting Layers**
  - Background, Middleground, Foreground **추가**
  - Sorting Layers는 **순서에 주의**, **드래그&드롭으로 순서를 변경** 가능

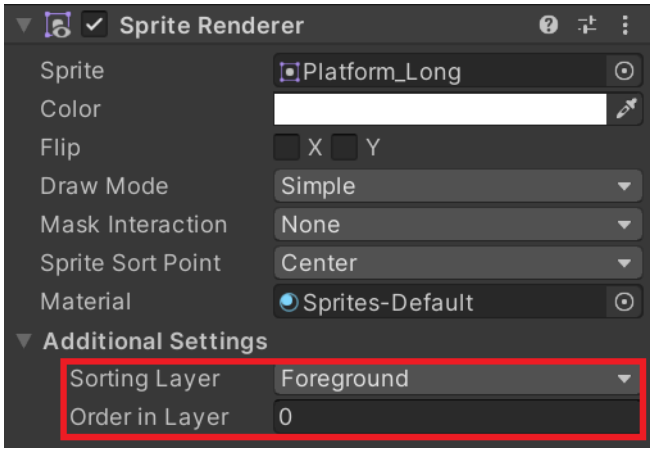




Sprite Draw Layer

- **Player(Toko)**
  - Sorting Layer : Middleground
  - Order in Layer : 0
- **Ground(Platform_Long)**
  - Sorting Layer : Foreground
  - Order in Layer : 0





배경

- **배경 스크롤**
  - Sky 이미지를 **Hierarchy에 추가**
  - Sprite Renderer에서 사용할 Sky(**Material**)을 **생성**
  - **새로 만든 Material**의 **Shader**를 **Unlit/Texture**로 변경
  - **Sky 이미지를 적용**
  - Sprite Rederer의 Material에 만들어둔 **Sky(Material)로 변경**
  - **Sorting Layer**를 **Background로 변경**
  - **Sprites 폴더**의 Sky 이미지의 **Mesh Type**을 **Full Rect로 변경**

    Tight:투명 영역을 최대한 제외

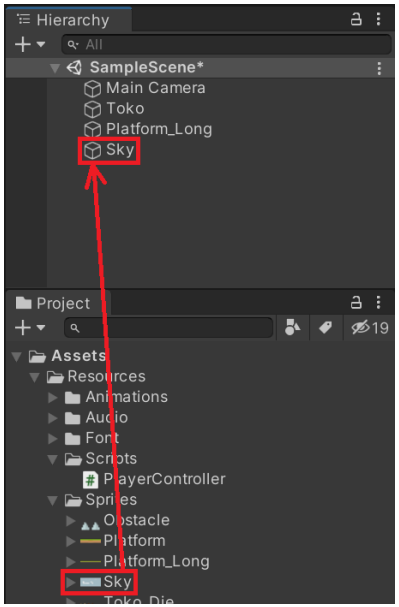
    Full Rect:원본 이미지 전체 크기
  - **Wrap Mode**를 **Repeat로 변경**

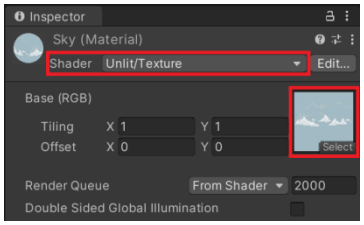
    Repeat:패턴 반복,

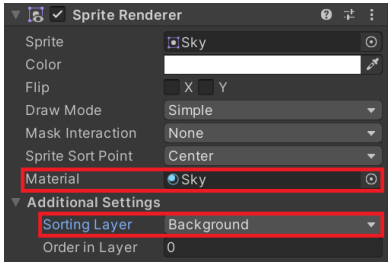
    Clamp:마지막 픽셀에 고정

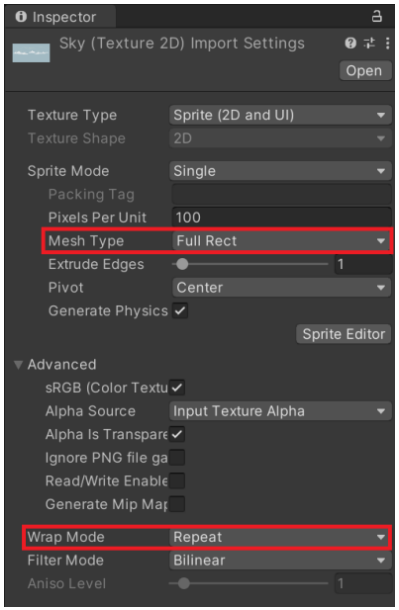
    Mirror:미러링 반복 패턴

    Mirror Once:한 번만 미러링 후 마지막 픽셀에 고정









배경

- **SpriteScroll**
- **Sky(GameObject)**에 스크립트 **생성 및 추가**
- 2D 게임이기 때문에 메인 카메라를 **Orthographic(원근감X)**으로 변경
- 카메라에 맞춰 이미지의 크기와 위치를 변경
- **Material**의 **offset** 을 변경하여 배경을 스크롤(**비용이 크기 때문에 사용하지 않는 것이 더 좋다**)
- **Mathf.Repeat()**를 이용하여 **Offset** 값의 범위를 **0.0f~1.0f로 제한**

```
public class SpriteScroll : MonoBehaviour
{
    SpriteRenderer background;
    Vector2 offset = Vector2.zero;

    float scrollSpeed = 0.0f;
    Coroutine coroutine = null;

    public void Initialize(float speed)
    {
        if (TryGetComponent(out background))
        {
            scrollSpeed = speed;

            Camera cam = Camera.main;
            cam.orthographic = true;
```

배경

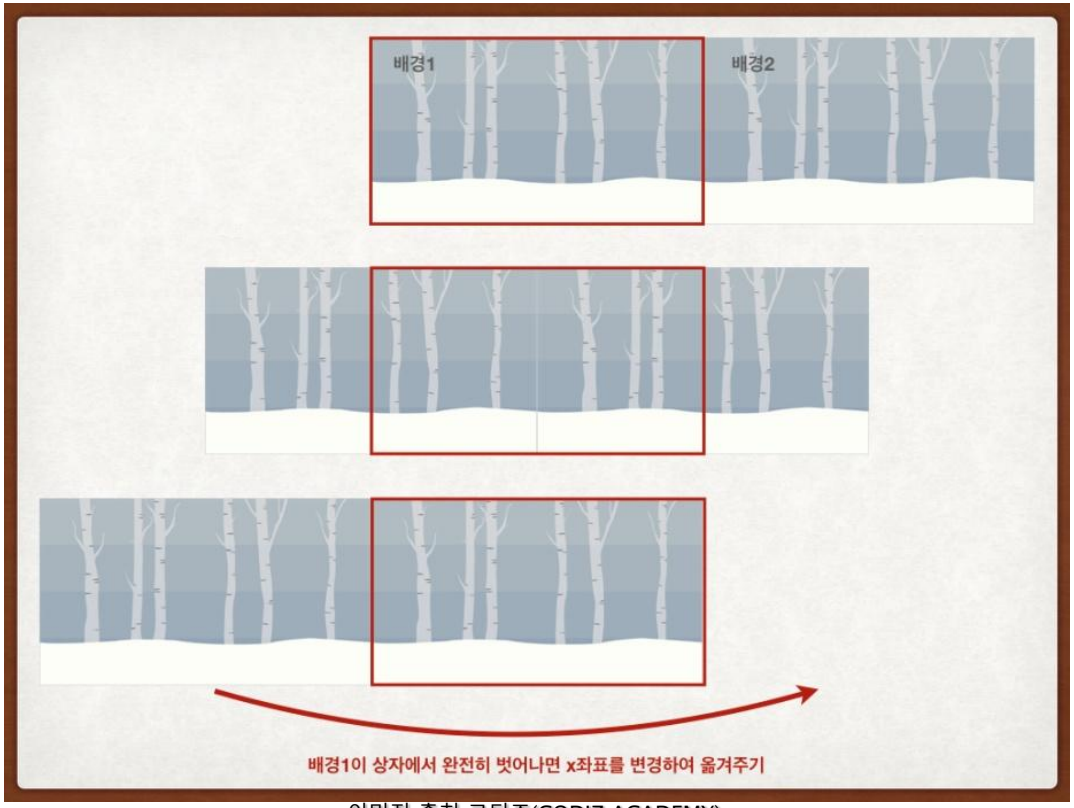
```
            Vector3 pos = cam.transform.position;
            pos.z += 10;
            transform.position = pos;

            float ratio = (float)Screen.width / Screen.height;
            float height = cam.orthographicSize * 2.0f;
            float width = height * ratio;

            background.drawMode = SpriteDrawMode.Tiled;
            background.size = new Vector2(width, height);
        } // if(TryGetComponent<>())
    } // void Initialize()

    IEnumerator RunScroll()
    {
        while (true)
        {
            yield return null;
            if (background)
            {
                offset.x = Mathf.Repeat(Time.time * scrollSpeed * 0.01f, 1.0f);
                background.material.mainTextureOffset = offset;
            }
        }
    }

    public void Run() { if (null == coroutine) coroutine = StartCoroutine(RunScroll()); }
    public void Stop() { if (null != coroutine) StopCoroutine(coroutine); }
} // SpriteScroll
```

배경

```
public class GameMgr : MonoBehaviour
{
    ...
    SpriteScroll scroll;
    bool isReady = false;

    private void Awake()
    {
        ...
        if (SetupInput())
        {
            isReady = SetupScroll();
            if (isReady) scroll.Run();
        }
    }
    ...
    bool SetupScroll()
    {
        scroll = FindObjectOfType<SpriteScroll>();
        if (scroll)
        {
            scroll.Initialize(player.Speed);
            return true;
        }
        return false;
    }
}
```

지형 이동

- Object Scroll
  - **일정한 개수의 지형을 배치**하여 두고 **꼬리물기를 하듯이 계속 이어지게** 처리한다



이미지 출처:코디즈(CODIZ ACADEMY)

지형 이동

- **SpriteScroll**
  - **지형의 위치** 이동과 지형 간의 거리 조절을 위한 **지형의 너비**를 지닌 GroundData **구조체**

```
struct GroundData
{
    public float xPos;
    public float width;
}

public class SpriteScroll : MonoBehaviour
{
    ...
    [Header("Ground Objects")]
    [SerializeField] SpriteRenderer[] grounds;
    [SerializeField] float bounds = 5.0f;

    GroundData[] groundDatas;
    int count = 0;
    float halfWidth = 0.0f;
    float prePosX = 0.0f;
```

지형 이동

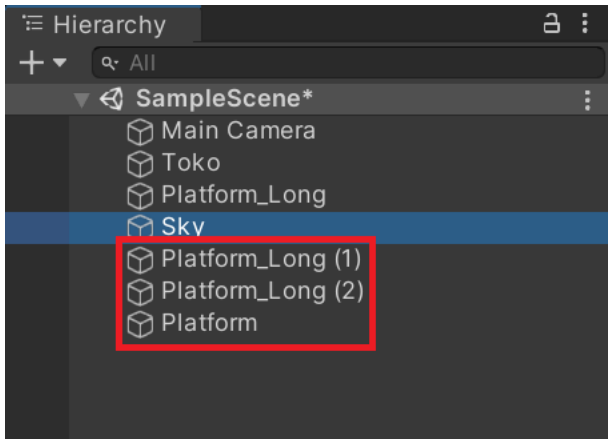
```
public void Initialize(float speed)
{
    if (TryGetComponent(out background))
    {
        ...
        halfWidth = width * 0.5f;
        count = grounds.Length;
        if (1 < count)
        {
            groundDatas = new GroundData[count];
            for (int i = 0; count > i; i++)
            {
                groundDatas[i].width = grounds[i].size.x;
                // 두 번째 지형부터 앞의 지형 위치와 설정 값들을 참조하여 시작 위치를 변경한다.
                if (0 < i)
                {
                    groundDatas[i].xPos = groundDatas[i - 1].xPos + bounds + groundDatas[i].width;
                    grounds[i].transform.position = Vector3.right * groundDatas[i].xPos + Vector3.down;
                }
                else groundDatas[i].xPos = grounds[i].transform.position.x;
            }

            prePosX = groundDatas[count - 1].xPos;
        } // if(1 < count)
    } // if(TryGetComponent<>())
} // void Initialize()
```

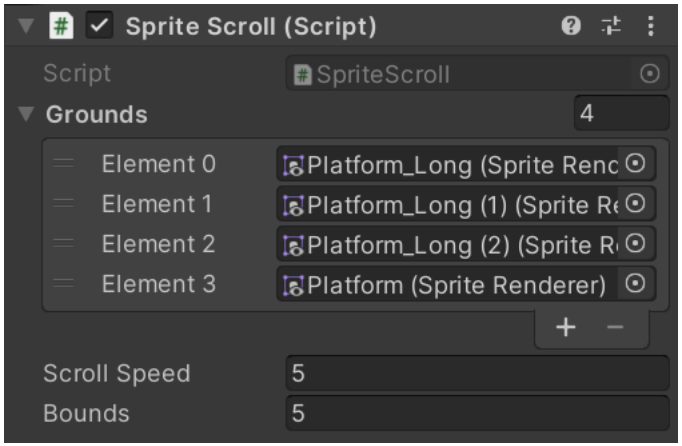
지형 이동

```
    IEnumerator RunScroll()
    {
        ...
            if (1 < count)
            {
                for (int i = 0; count > i; i++)
                {
                    groundDatas[i].xPos -= Time.deltaTime * scrollSpeed;
                    // 오브젝트가 카메라 영역을 벗어나면 해당 오브젝트를 제일 오른쪽 위치로 옮긴다.
                    // 이동 위치는 가장 오른쪽에 위치한 오브젝트의 x좌표 + 자신의 너비 + 각 오브젝트간의 거리.
                    if (-halfWidth >= groundDatas[i].xPos)
                    {
                        groundDatas[i].xPos = prePosX + bounds + groundDatas[i].width;
                    }

                    // 지형의 위치가 (x, -1, 0)이기 때문에 Vector3.down을 더한다.
                    grounds[i].transform.position = Vector3.right * groundDatas[i].xPos + Vector3.down;
                    // 카메라의 영역을 벗어날 경우 가장 오른쪽에 위치한 오브젝트의 위치를 저장하여 둔다.
                    // 자신의 앞에 위치했던 지형이 가장 오른쪽에 위치하게 되는 지형과 같다.
                    prePosX = groundDatas[i].xPos;
                } // for()
            } // if(1 < count)
        } // while(true)
    } // Ienumerator RunScroll()
} // class SpriteScroll
```

지형 이동

- **지형 배치**
  - **Hierarchy**에 **Platform_Long** 또는 **Platform**을 추가 후 **BoxCollider2D 컴포넌트** 추가
  - **Sprite Scroll**의 **Grounds**에 **등록**

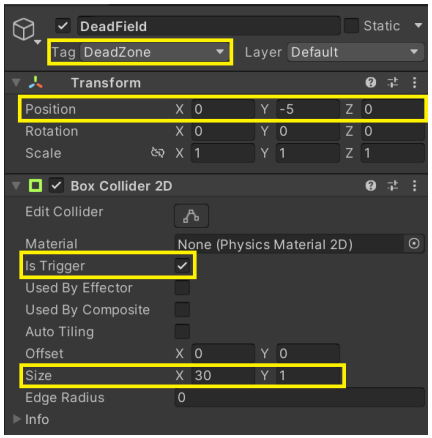


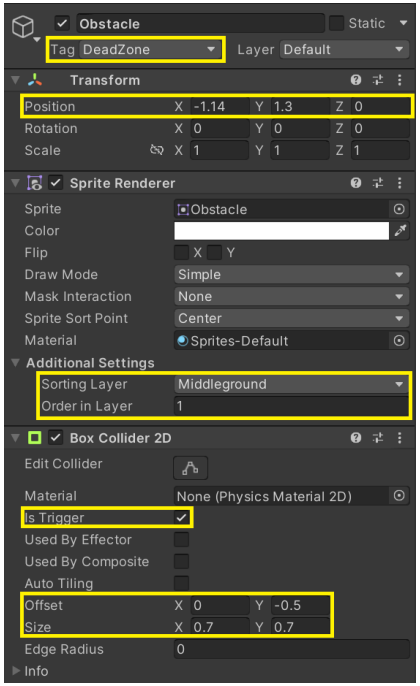


Dead Zone

DeadField

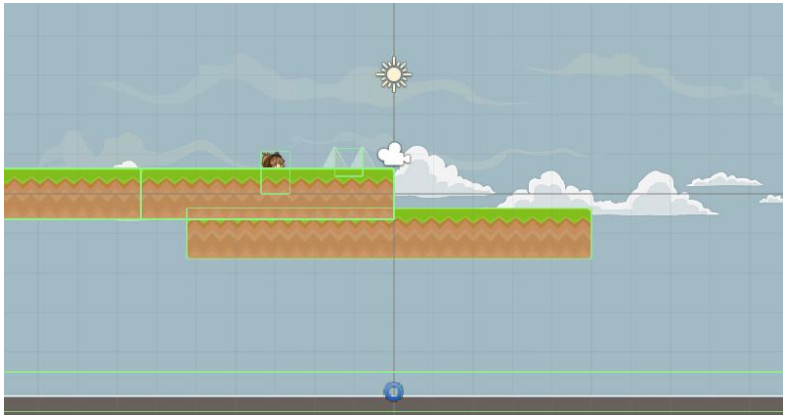
- **빈 오브젝트 생성**
- **BoxCollider2D 추가**
- Position : (0, -5, 0)
- Is Trigger : True
- Size : (30, 1)
- Tag : DeadZone





Obstacle

- Platform_Long(GameObject) 또는 Platform(GameObject)의 자식으로 **Obstacle**을 원하는 만큼 **추가**
- Sorting Layer : Middleground
- Order in Layer : 1
- **BoxCollider2D 추가**
- Position : (지형 범위 안의 원하는 x좌표, 1.3, 0)
- Is Trigger : True
- Offset : (0, -0.5)
- Size : (0.7, 0.7)
- Tag : DeadZone



Dead Zone

- **Tag**
  - **Tag(DeadZone)의 충돌** 확인하여 **플레이어의 움직임을 제한**
  - **Die 애니메이션**을 실행 후 현재 **상태**를 **변경**

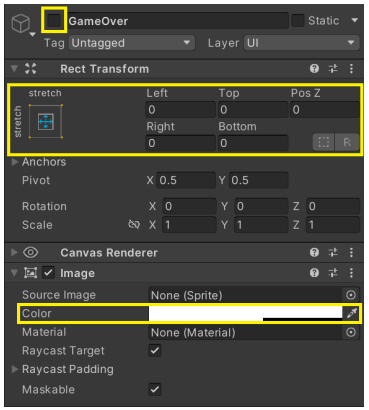
```
public class PlayerController : MonoBehaviour
{
    ...
    public event System.Action OnDead = null;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("DeadZone"))
        {
            // 장애물(DeadZone)에 닿으면 Physics를 정지하고 죽음 처리를 한다.
            if (rigid) rigid.simulated = false;
            if (anim) anim.SetTrigger("isDie");
            OnDead?.Invoke();
        }
    }
}
```

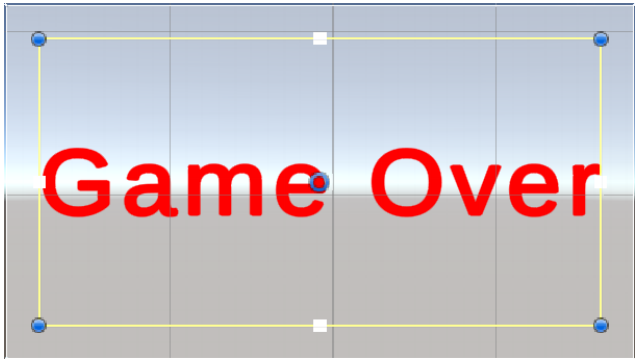
Dead Zone

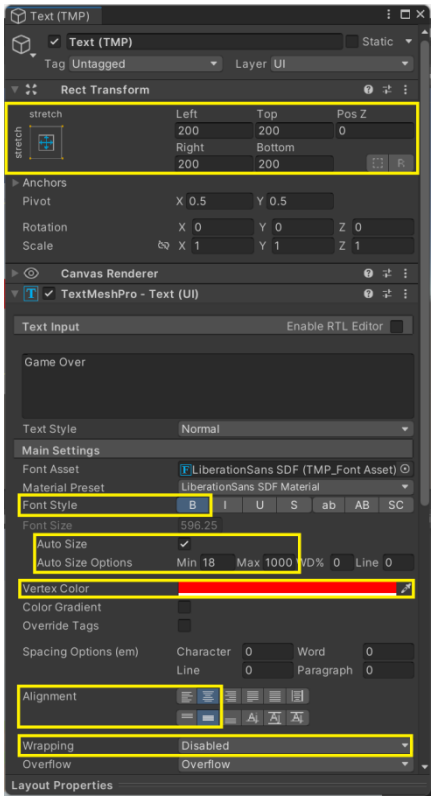
```
public class GameMgr : MonoBehaviour
{
    private void Awake()
    {
        ...
        if (SetupInput())
        {
            isReady = SetupScroll();
            if (isReady)
            {
                scroll.Run();
                player.OnDead += () =>
                {
                    inpuAsset["Jump"].Disable();
                    scroll.Stop();
                };
            }
        }
    }
}
```

UI

- **GameOver(Image)**
  - UI=>Image
  - Anchor : (stretch, stretch)
  - Left, Top, Pos z, Right, Bottom : 0
  - Color : (255, 255, 255, 150)
  - GameObject Active : false
  - **Child** : UI=>**TextMeshPro**
  - Anchor : (stretch, stretch)
  - Pos z : 0
  - Left, Top, Right, Bottom : 200
  - Font Style : Bold
  - Auto Size : True(Max : 1000)
  - Alignment : 가운데 정렬
  - Warpping : Disabled
  - Vertex Color : (255, 0, 0)



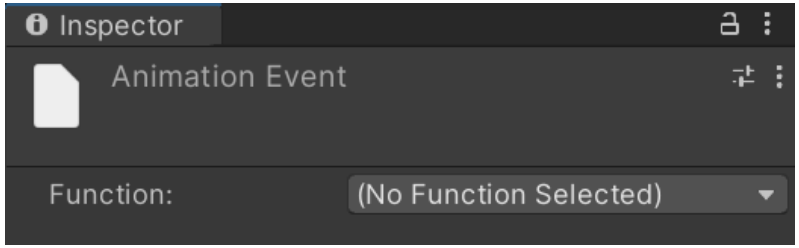


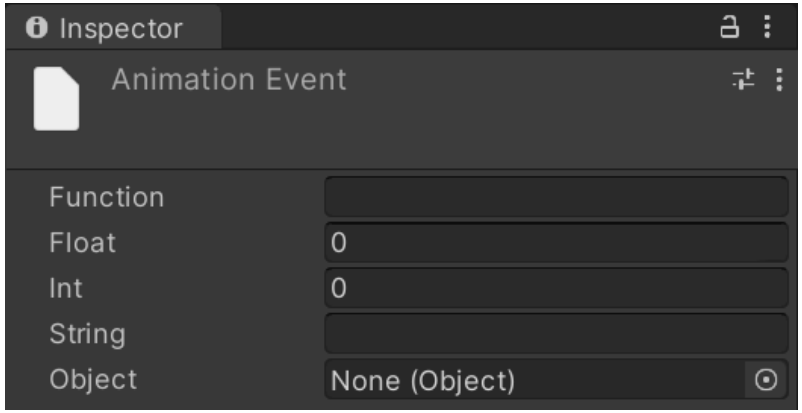


UI

- **Animation Clip Event**
  - 애니메이션 클립의 **특정 프레임**에 이벤트를 등록 가능
  - 이벤트를 등록할 오브젝트 선택
  - 애니메이션 창의 클립 선택
  - 특정 프레임에서 **[add event] 버튼** 또는 **[마우스 오른쪽 버튼 클릭=>Add Animation Event]**

- **Animation Event**
  - **Function**에 원하는 **호출 함수를 등록**(Function : 애니메이션이 있는 오브젝트에서 호출 가능한 함수 명)

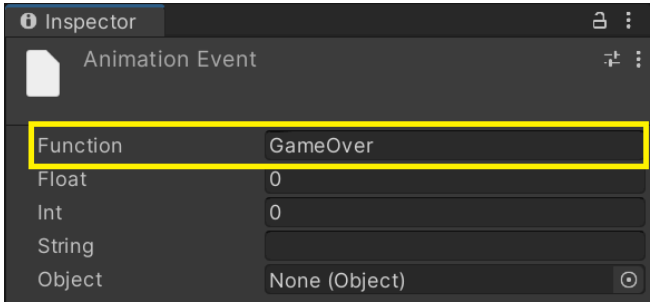


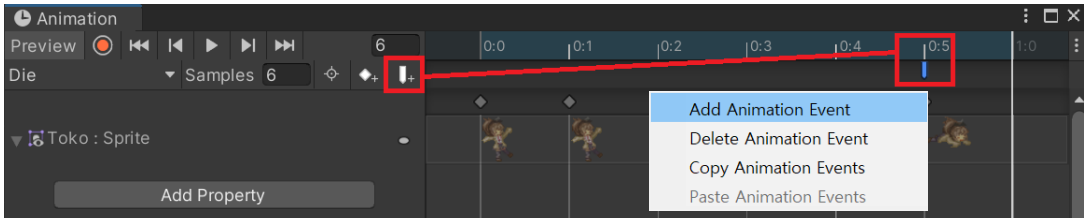


UI

GameMgr, PlayerController

- Die 애니메이션 실행이 끝나면 GameOver 화면을 출력한다
- Die animation Clip의 마지막 프레임에 Event를 등록하고 Function에 GamOver 등록



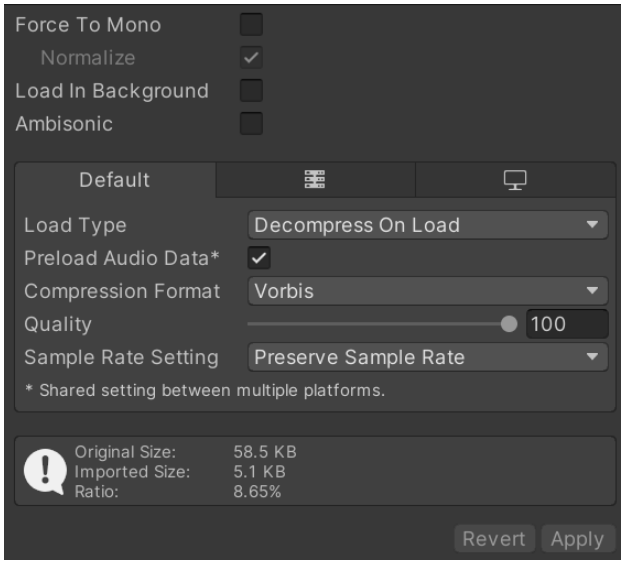



```
public class PlayerController : MonoBehaviour
{
    ...
    public event System.Action DieAnimationEvent = null;
    void GameOver() { DieAnimationEvent?.Invoke(); }
}

public class GameMgr : MonoBehaviour
{
    ...
    GameObject gameOver;
    private void Awake()
    {
        ...
        if (isReady)
        {
            var canvas = FindObjectOfType<Canvas>();
            if (canvas) gameOver = canvas.transform.GetChild(0).gameObject;
            player.DieAnimationEvent += () => { if (gameOver) gameOver.SetActive(true); };
        }
    }
}
```

Audio

- **die, jump Audio Clip**
  - Compression Format : **PCM**

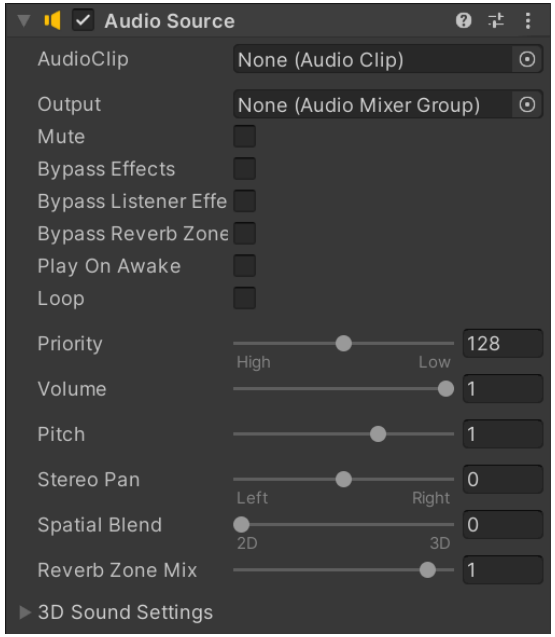
- **music Audio Clip**
  - Load Type : **Compressed In Memory**



Audio

- **SoundManager**
  - **Resources 폴더**에서 **Audio 파일을 로드**하여 **상황에 맞게 재생**

```
public class SoundMgr
{
    AudioClip[] audioClips;
    AudioSource audioSource;

    public void Initialize()
    {
        audioClips = Resources.LoadAll<AudioClip>("Audio");

        GameObject go = new GameObject("Main Audio Source");
        audioSource = go.AddComponent<AudioSource>();
        audioSource.loop = true;
        audioSource.playOnAwake = false;
        audioSource.transform.SetParent(GameMgr.Instance.transform);
    }
```



Audio

```
    AudioClip FindClip(string name)
    {
        foreach (AudioClip clip in audioClips)
        {
            if (clip.name.ToLower().Equals(name.ToLower()))
            {
                return clip;
            }
        }
        return null;
    }

    public void PlayBGM(string clipName)
    {
        audioSource.clip = FindClip(clipName);
        audioSource.Play();
    }

    public void StopBGM() { audioSource.Stop(); }

    public void PlaySFX(string clipName)
    {
        AudioClip clip = FindClip(clipName);
        if (clip) audioSource.PlayOneShot(clip);
    }
}
```

Audio

```
public class GameMgr : MonoBehaviour
{
    ...
    public SoundMgr soundMgr { get; private set; } = null;

    private void Awake()
    {
        ...
                player.OnDead += () =>
                {
                    inpuAsset["Jump"].Disable();
                    scroll.Stop();

                    soundMgr.StopBGM();
                    soundMgr.PlaySFX("die");
                };
            } // if(isReady)
        } // if(SetupInput())
        ...
    }

    void Initialize()
    {
        ...
        soundMgr = new SoundMgr();
        soundMgr.Initialize();
        soundMgr.PlayBGM("music");
    }
}
```

Audio

```
public class PlayerController : MonoBehaviour
{
    ...
    public void Jump(InputAction.CallbackContext context)
    {
        if (!rigid) return;

        if (context.started && (limitJumpCount > jumpCount))
        {
            GameMgr.Instance.soundMgr.PlaySFX("jump");
            jumpCount++;

            rigid.velocity = Vector2.zero;
            rigid.AddForce(Vector2.up * jumpForce * 100.0f);
        }
        else if (context.canceled && (0 < rigid.velocity.y))
        {
            rigid.velocity *= 0.5f;
        }
    }
}
```