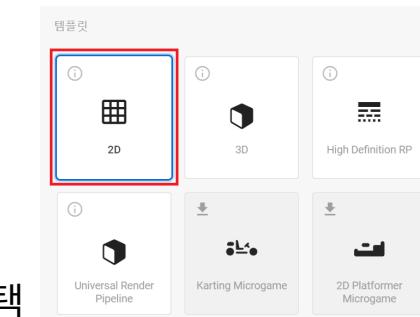


Unity Uni-Run

프로젝트 준비

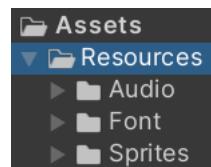
▶ 새 프로젝트 생성

- 새로 만드는 **프로젝트**의 템플릿을 **2D**로 선택
- 생성 후 프로젝트에서 변경을 하여도 좋지만 미리 선택하면 추가로 설정 할 필요가 없다



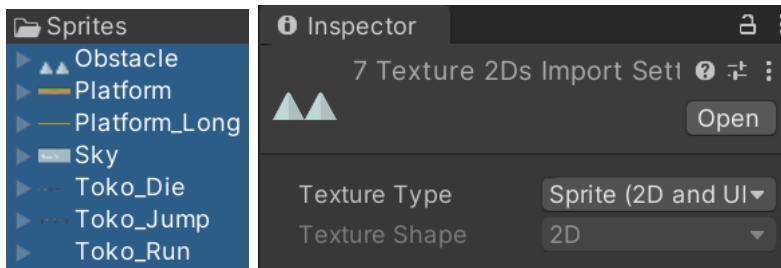
▶ 리소스 추가

- 프로젝트에 **Resources** 폴더를 만든다
- 예제소스 폴더 => Uni-Run 폴더에 있는 폴더 및 파일을 전부 **Resources** 폴더로 **복사**



▶ Sprites

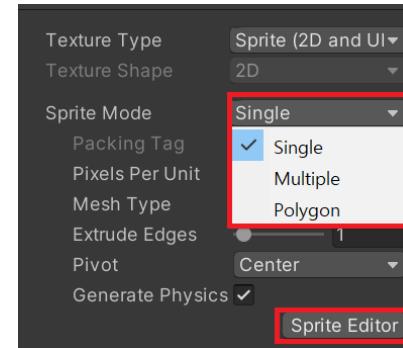
- Sprites 폴더의 이미지 리소스들의 Texture Type을 **Sprites(2D and UI)**로 변경



Multiple Sprites

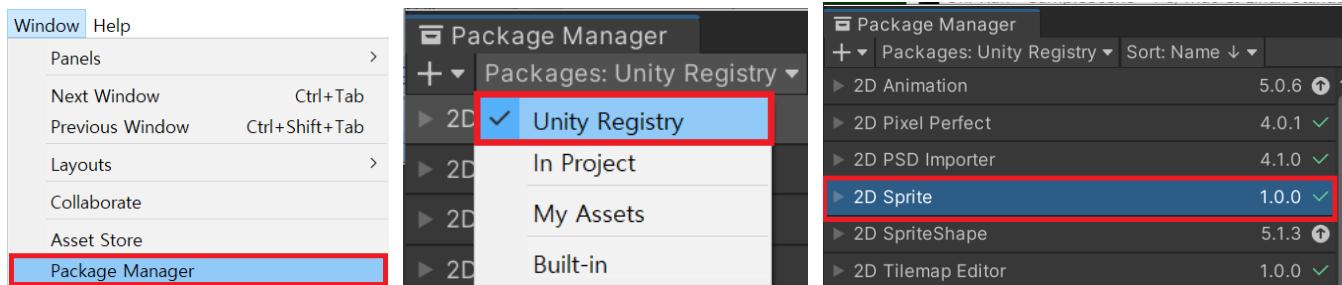
▶ Sprite Mode

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



▶ Sprite Editor

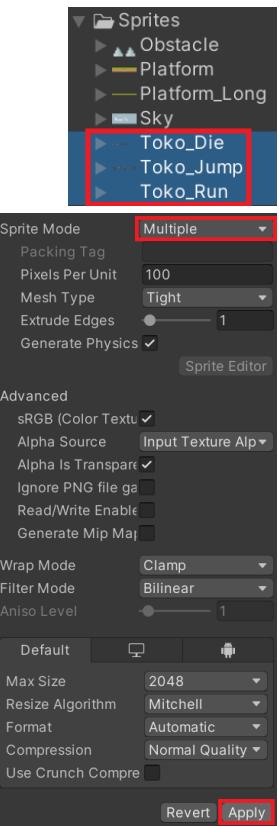
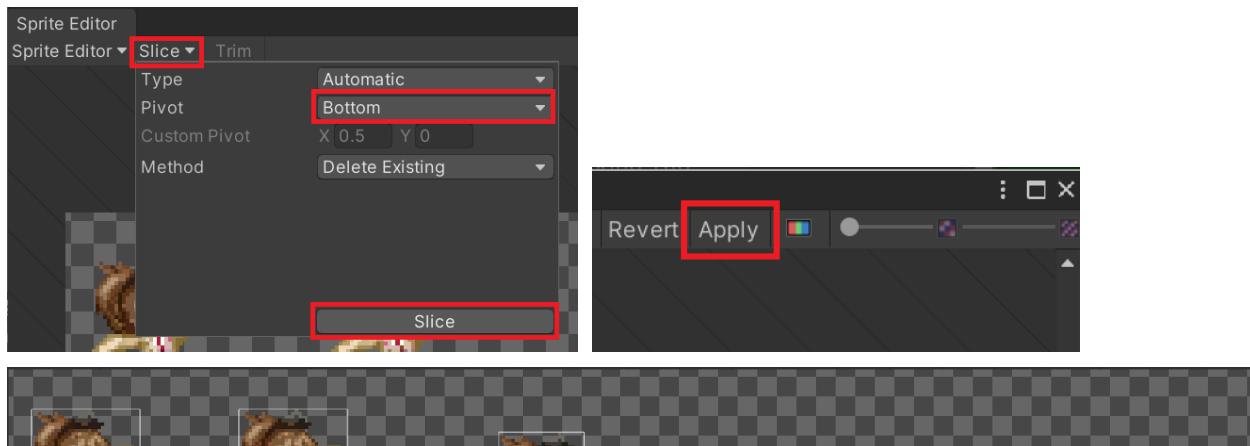
- **Multiple 또는 Polygon** 모드를 사용할 경우 **Sprite 이미지를 편집**하는데 사용
- **Sprite Editor**를 쓰기 위해서는 **Package Manager**에서 **2D Sprite**를 **설치**해야 한다
- **Window=> Package Manager=>Packages:Unity Registry**
- **2D Sprite**를 선택하고 오른쪽 아래의 **Install** 버튼으로 설치



Multiple Sprites

▶ Player Sprites

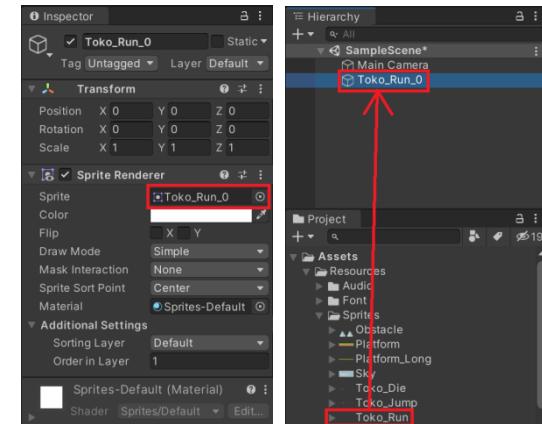
- Toko_Die, Toko_Jump, Toko_Run 세 Sprite 이미지를 **Sprite Mode**를 **Multiple**로 변경
- Sprite의 Size를 키우기 위해서 **Pixel Per Unit**의 값을 **40으로 변경**(값이 작을수록 커진다)
- Sprite Editor에서 Slice의 **Pivot**을 **Bottom**으로 변경하고 Slice 실행



Sprites Animation

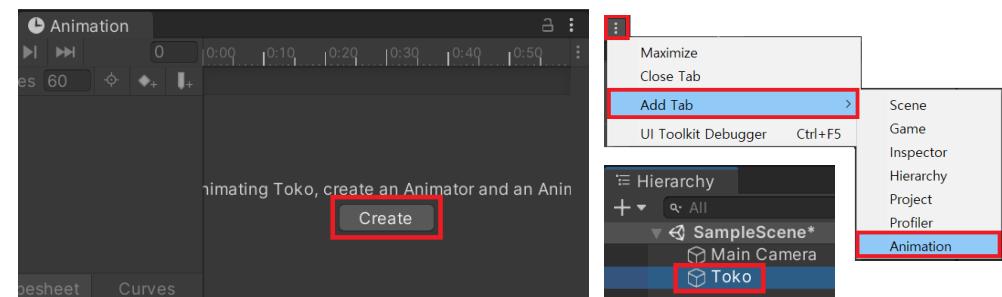
▶ 2D GameObject

- Sprite로 설정한 이미지 파일은 하이어라키(Hierarchy)로 등록이 가능하다
- Toko_Run 이미지를 하이어라키(Hierarchy)로 드래그&드롭, 자동으로 잘라둔 이미지의 첫 번째 이미지로 설정된다
- 오브젝트 이름을 'Toko'로 변경
- Position : (-3, 0, 0)



▶ Animation

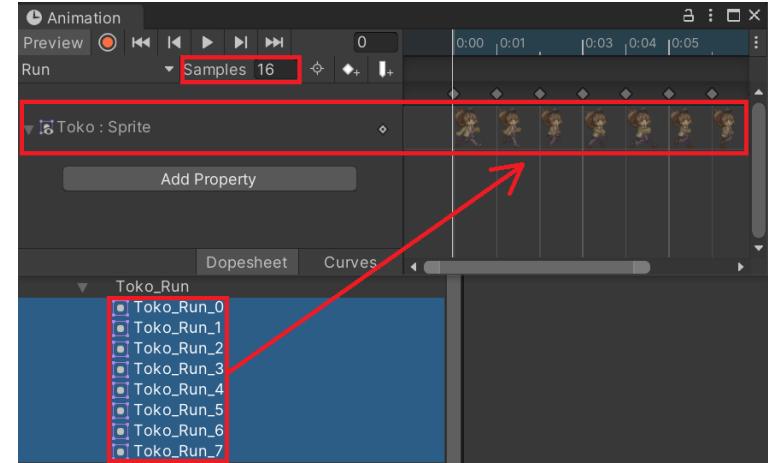
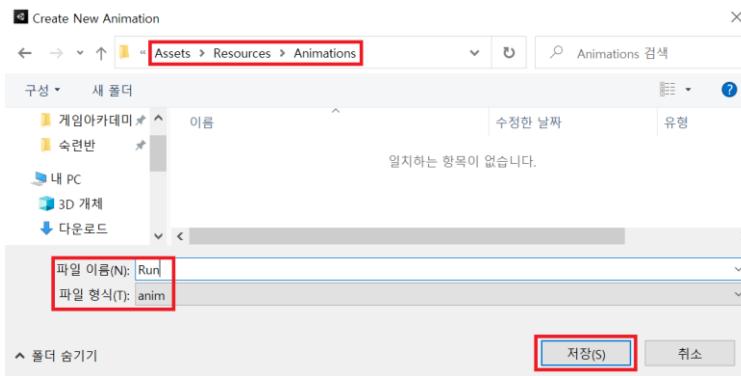
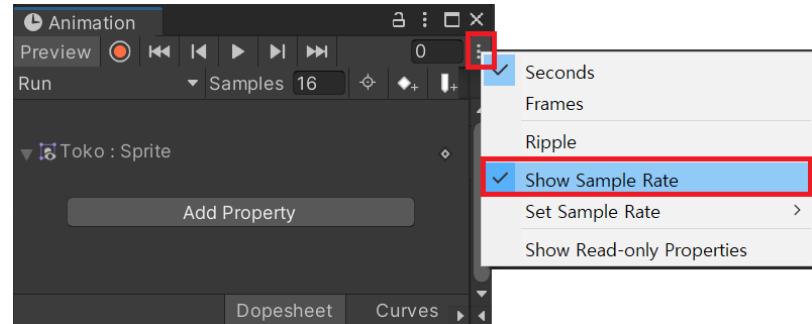
- 애니메이션을 만들고 편집은 Animation 창에서 한다
- Window=>Animation=>Animation
- 또는 각 창의 오른쪽 상단의 [...]=>Add Tab=>Animation
- 애니메이션을 만들기 위한 게임 오브젝트 선택
- Animation 창의 Create 버튼을 선택



Sprite Animation

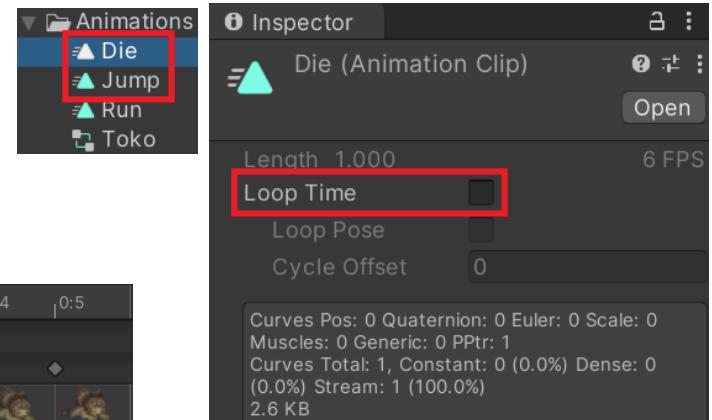
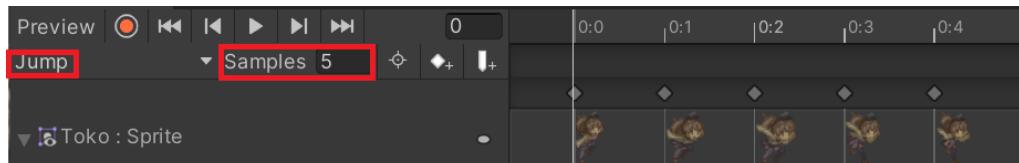
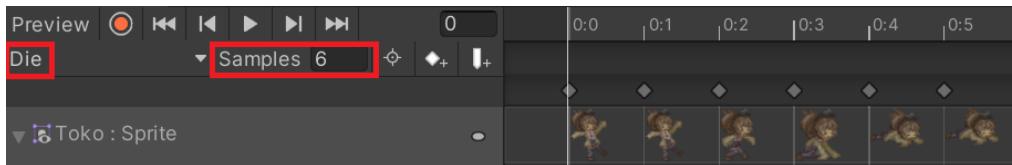
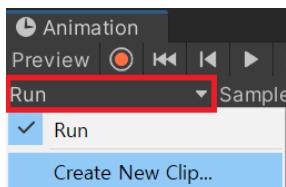
▶ Player Animation

- Resources 폴더에 Animations 폴더 추가
- 'Toko'를 선택하여 Animation Create 버튼을 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

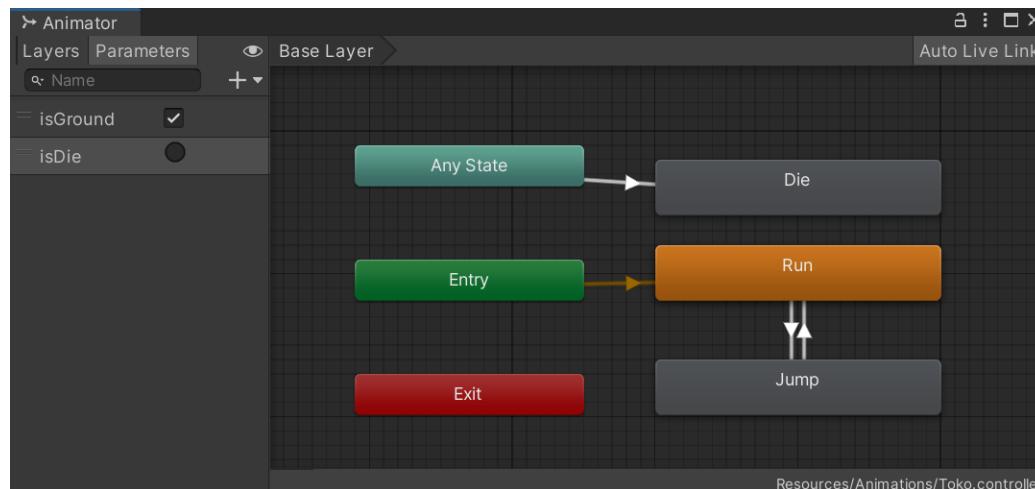
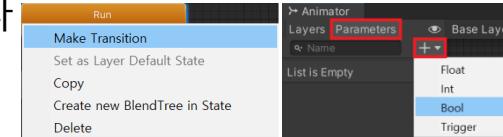
- Create New Clip...=>Die, Jump 추가
- 각 Animation Clip에 Sprite 이미지 등록
- Die와 Jump의 Samples에 각각 6, 5로 변경
- 두 애니메이션은 한 번만 재생해야 되기 때문에 위해 Animation Clip을 선택하여 각각 Loop Time 체크를 해제



Animator

▶ Animator Controller

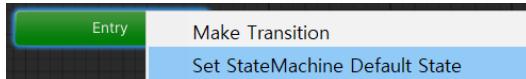
- Window=>Animation=>Animator 또는 Animator Controller 더블 클릭
- 애니메이션 클립을 만들면 자동으로 생성, 추가 된다
- 애니메이션 전환 및 정렬하고 관리할 수 있다
- 유한상태기계(FSM)을 기반으로 만들어져 있다
- Make Transition을 선택하여 다른 애니메이션과 연결, 조건을 설정하여 실행 중 애니메이션을 다른 애니메이션으로 상태 변경이 가능하다
- Parameters를 [+]로 조건 파라미터를 추가할 수 있다



Animator

▶ Entry

- **상태의 시작**
- 애니메이션을 실행하면 **연결된 애니메이션 클립(Default State)**이 자동으로 실행된다
- 제일 처음 적용된 애니메이션 클립이 Default로 적용된다
- **Set State Machine Default State**로 Default 애니메이션을 변경할 수 있다



▶ Any State

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



▶ Exit

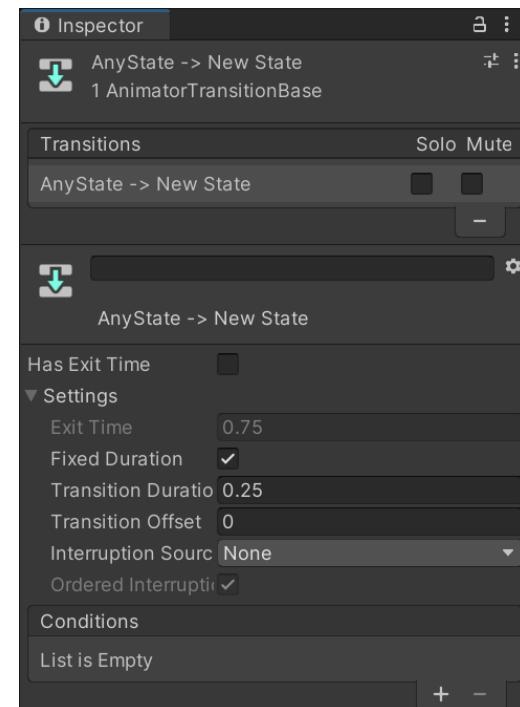
- **Exit로 연결된 애니메이션**(Any State Animation->Exit)이 끝나면 **Entry 상태로 변경**된다



Animator

▶ Transition Inspector

- Transitions
 - Solo : 현재의 상태에서 다른 상태로 변화하는 트랜지션은 Solo를 체크한 트랜지션만을 허용한다
 - Mute : 현재 트랜지션을 작동하지 못하게 한다
 - Has Exit Time : **Exit Time**을 참조하여 해당 시간에 **전환 조건이 적용되도록** 한다
 - Exit Time : **Has Exit Time**이 활성화 인 경우 사용(0.75 == 75%)
 - Fixed Duration : **Transition Duration**의 값을 **초 단위(true)**
또는 **정규화된 단위(false)**로 사용
 - Transition Duration : 애니메이션을 부드럽게 전환하기 위한 시간
(2D에서는 의미가 없다)
 - Trasition Offset : 전환된 애니메이션의 시작 지점의 시간
 - Conditions : **전환 조건**
조건이 없다면 Exit Time을 참조하여 시간이 되면
바로 상태(애니메이션)를 전환한다



Animator

▶ 애니메이션 상태 편집

- 'Toki'의 Animator Controller 선택

- Parameters에 **isGround(Bool)**, **isDie(Trigger)** 추가

- **isGround : True**

- Run과 Jump에 서로 Transition으로 연결

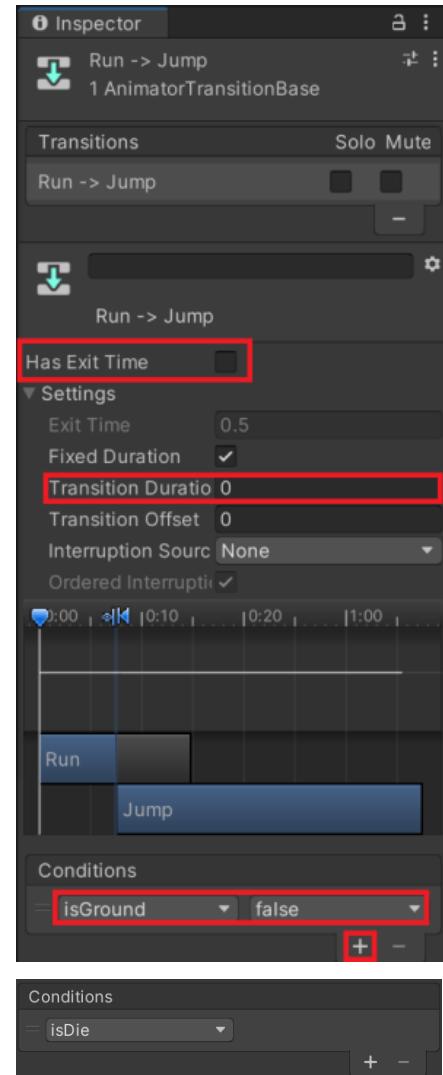
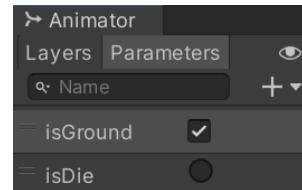
- 각 Transition(하얀색 화살표)을 선택

- Has Exit Time 체크 해제

- Transition Duration : 0

- Run->Jump Conditions 추가 **isGround : false**

- Jump ->Run Conditions 추가 **isGround : true**



- Any State에 Die를 Transition으로 연결

- Any State->Die Transition 선택

- Has Exit Time 체크 해제

- Transition Duration : 0

- Conditions 추가 **isDie**



Run & Jump

▶ Platform_Long

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

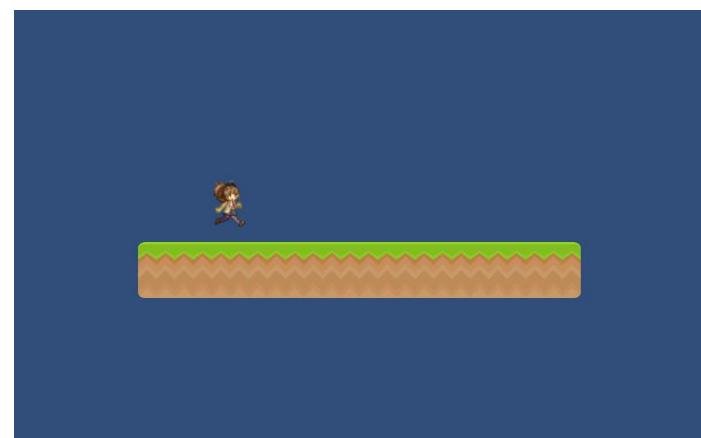
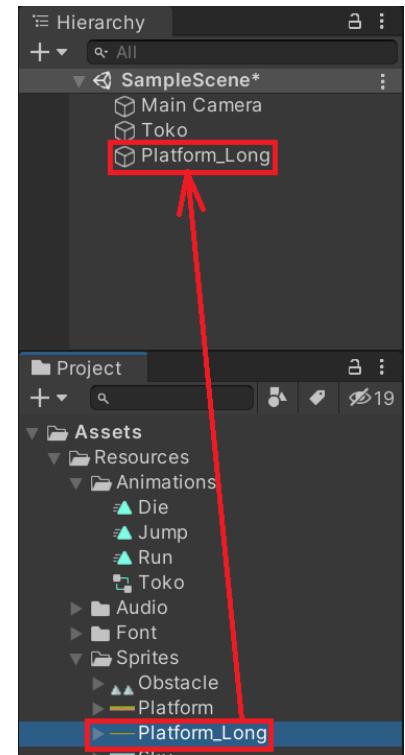
▶ Toko

- Box Collider 2D 추가
- Rigidbody 2D 추가

▶ PlayerController

- Scripts 풀더 추가
- C# PlayerController 생성, Toko에 추가

```
[RequireComponent(typeof(BoxCollider2D))]  
[RequireComponent(typeof(Rigidbody2D))]  
public class PlayerController : MonoBehaviour  
{  
    Rigidbody2D rigid;  
    Animator anim;  
  
    void Start()  
    {  
        rigid = GetComponent<Rigidbody2D>();  
        anim = GetComponent<Animator>();  
    }  
}
```



Run & Jump

- 특정 버튼을 누르는 시간에 비례하여 점프 높이가 다르게 한다
- 2단 점프 적용

```
public class PlayerController : MonoBehaviour
{
    [SerializeField] float jumpForce = 300f;

    readonly int limitJumpCount = 2;
    int jumpCount = 0;

    void Update()
    {
        if (!rigid) return;

        if (Input.GetKeyDown(KeyCode.Space) && limitJumpCount > jumpCount)
        {
            jumpCount++;

            rigid.velocity = Vector2.zero;
            rigid.AddForce(Vector2.up * jumpForce);
        }
        else if (Input.GetKeyUp(KeyCode.Space) && 0 < rigid.velocity.y)
        {
            rigid.velocity *= 0.5f;
        }
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        jumpCount = 0;
    }
}
```

Run & Jump

- 땅에 있을 경우 **달리는 애니메이션**, 점프를 하면 **점프 애니메이션을 실행**하도록 한다
- **SetXXX()** 함수로 설정한 **Parameters** 값을 수정, 제어할 수 있다

```
public class PlayerController : MonoBehaviour
{
    private void OnCollisionEnter2D(Collision2D collision)
    {
        // normal : 충돌 지점의 법선 벡터, 충돌 대상에서 나를 향하는 법선 벡터.
        if (collision.contacts[0].normal.y > 0.8f)
        {
            if (anim) anim.SetBool("isGround", true);
            jumpCount = 0;
        }
    }

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

Sprite Draw Layer

▶ 2D 그리기

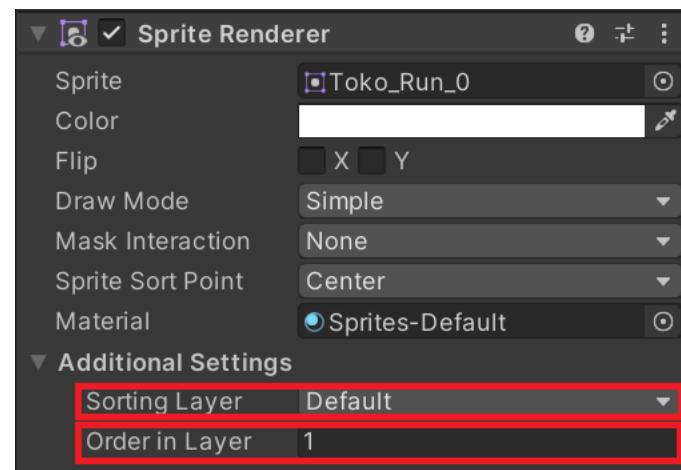
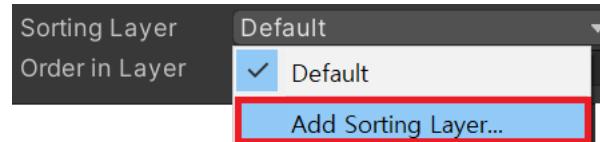
- 2D는 깊이 값(z)을 사용하지 않는다
- 다 같은 깊이(z)에 있기 때문에 추가한 이미지에 의하여 보이지 않는 경우가 발생한다
- Draw Layer를 설정하여 그리는 순서를 지정한다

▶ Sorting Layer

- 기본적으로 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



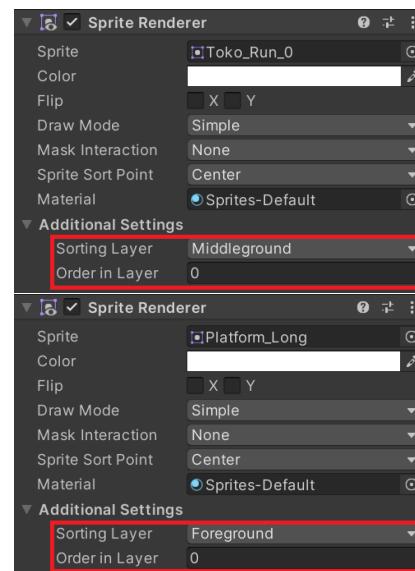
Sprite Draw Layer

▶ Sorting Layer 추가

- 툴 바(Tool Bar)의 Layers=>Edit Layer 또는 Sprite Renderer의 Sorting Layer=>Add Sorting Layer를 선택
- [+], [-] 버튼으로 Layer를 추가, 삭제가 가능하다
- **Background, Middleground, Foreground** 추가
- **순서에 주의**, 순서가 잘못되었다면 드래그&드롭으로 순서를 변경할 수 있다

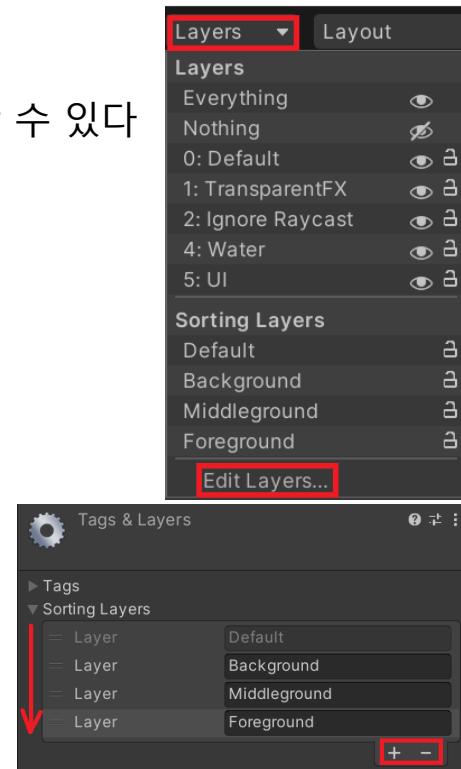
▶ Player(Toko)

- Sorting Layer : Middleground
- Order in Layer : 0



▶ Ground(Platform_Long)

- Sorting Layer : Foreground
- Order in Layer : 0



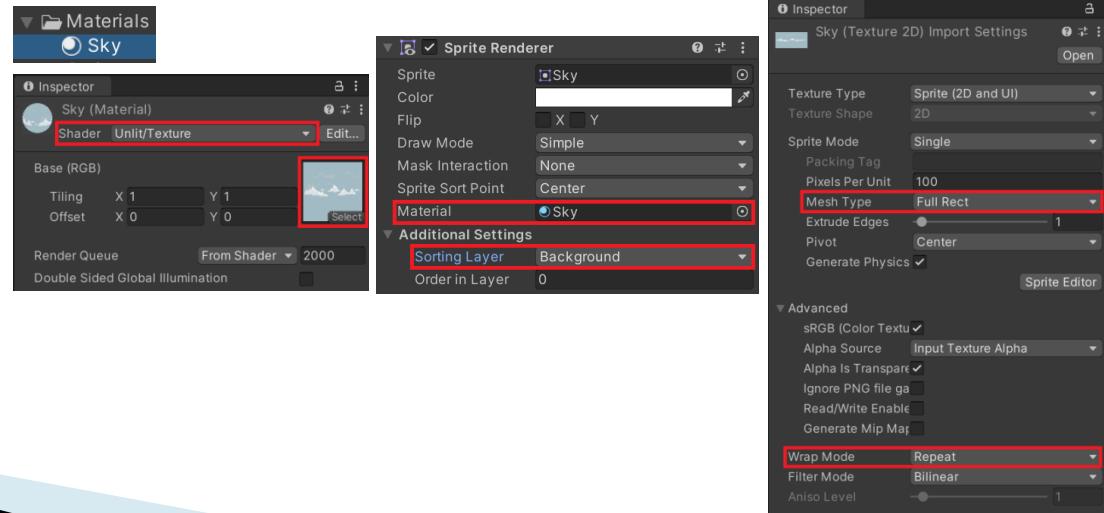
배경

▶ 배경 스크롤

- Sky 이미지를 Hierarchy에 추가
- Sprite Renderer에서 사용할 Sky Material을 새로 만든다
- Material의 Shader를 Unlit/Texture로 변경하고 Sky 이미지를 적용
- Sky Sprite Rederer의 Material을 만들어둔 Sky Material로 변경
- Sorting Layer를 Background로 변경
- Sky 이미지의 Mesh Type을 Full Rect로 변경
(Tight:투명 영역을 최대한 제외, Full Rect:원본 이미지 전체 크기)
- Wrap Mode를 Repeat로 변경

(Repeat:패턴 반복, Clamp:마지막 픽셀에 고정, Mirror:미러링 반복 패턴, Mirror Once:한 번만 미러링 후 마지막 픽셀에 고정)

- C# SpriteScroll 생성
- Sky에 SpriteScroll 추가



배경

▶ SpriteScroll

- 카메라의 영역 만큼 이미지의 크기(**width**)를 변경
- 오브젝트의 위치를 변경하지 않고 **Material**의 offset 값을 변경하여 배경이 흐르도록 한다
- Offset 값의 범위가 0~1이기 때문에 **Mathf.Repeat()** 함수를 이용하여 제한해 준다

```
public class SpriteScroll : MonoBehaviour
{
    [SerializeField] float scrollSpeed = 5f;

    SpriteRenderer background;
    Vector2 offset = Vector2.zero;

    private void Start()
    {
        var worldScreenHeight = Camera.main.orthographicSize * 2.0f;
        var worldScreenWidth = worldScreenHeight / Screen.height * Screen.width;
```

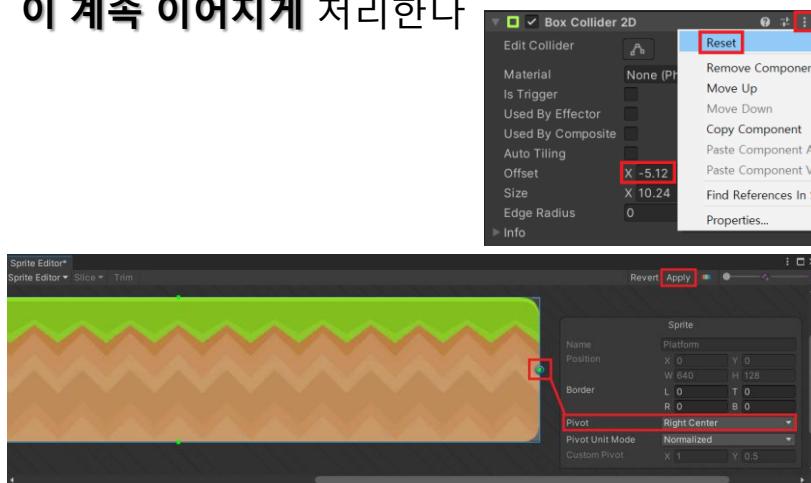
배경

```
background = GetComponent<SpriteRenderer>();
if (background)
{
    background.drawMode = SpriteDrawMode.Tiled;
    var size = background.size;
    size.x = worldScreenWidth;
    background.size = size;
}
void Update()
{
    if (background)
    {
        offset.x = Mathf.Repeat(Time.time * scrollSpeed * 0.01f, 1);
        background.material.mainTextureOffset = offset;
    }
}
```

지형 이동

▶ Pivot

- Platform, Platform_Long 이미지 **Sprite Editor** 선택
- **Pivot**을 **RightCenter**로 변경
- Apply하면 Pivot에 맞게 이미지의 위치가 자동으로 변경된다
- **BoxCollider**의 위치 변경을 위해 **Reset 버튼**을 선택
- 자동으로 이미지 위치만큼 Offset이 이동된다
- 배경 스크롤과 달리 **지형의 스크롤 방식은 일정한 개수의 지형을 배치하여 두고 꼬리물기를 하듯 이 계속 이어지게** 처리한다



이미지 출처: 코디즈(CODIZ ACADEMY)

지형 이동

▶ SpriteScroll

- 지형의 위치 이동과 지형 간의 거리 조절을 위한 지형의 너비를 지닌 **GroundData** 구조체를 만든다
- 각 지형이 카메라 영역을 벗어날 경우 가장 오른쪽에 배치된 지형 뒤로 이동한다

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

public class SpriteScroll : MonoBehaviour
{
    [SerializeField] SpriteRenderer[] grounds;

    [SerializeField] float bounds = 5f;

    GroundData[] groundDatas;
    float halfWidth = 0;
    float prePosX = 0;
```

지형 이동

```
private void Start()
{
    ...

halfWidth = worldScreenWidth * 0.5f;

var 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 값을 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;
}
}
```

지형 이동

```
void Update()
{
    ...

    var count = grounds.Length;
    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;
            }

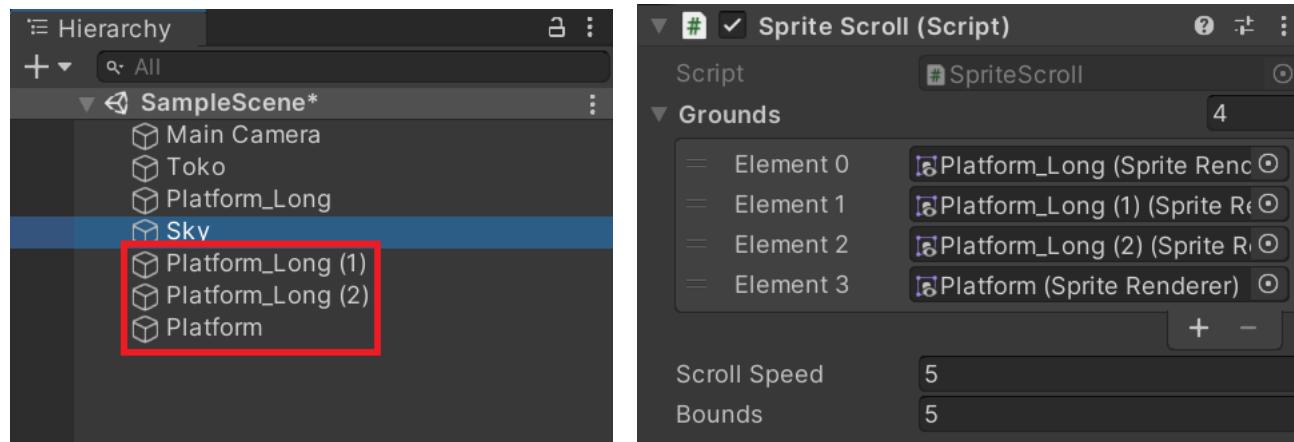
            // 지형의 위치가 (x, -1, 0)이기 때문에 Vector3.down을 더한다.
            grounds[i].transform.position = Vector3.right * groundDatas[i].xPos + Vector3.down;
            // 카메라의 영역을 벗어날 경우 가장 오른쪽에 위치한 오브젝트의 위치를 저장하여 둔다.
            // 자신의 앞에 위치했던 지형이 가장 오른쪽에 위치하게 되는 지형과 같다.
            prePosX = groundDatas[i].xPos;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

} // public class SpriteScroll
```

지형 이동

▶ 지형 배치

- Hierarchy에 Platform_Long 또는 Platform을 추가 후 BoxCollider2D 컴포넌트 추가
- Sprite Scroll의 Grounds에 등록



Dead Zone

▶ DeadZone

- Project Settings=>Tags and Layers=>Tags
- “DeadZone” **추가**

▶ DeadField

- [Create Empty] 또는 [Ctrl + Shift + N]으로 빈 오브젝트 생성
- **BoxCollider2D 추가**
- Position : (0, -5, 0)
- Is Trigger : True
- Size : (30, 1)
- Tag : DeadZone

▶ Obstacle

- Platform_Long 또는 Platform의 자식으로 Obstacle을 원하는 만큼 추가
- **BoxCollider2D 추가**
- Position : (원하는 x좌표, 1.3, 0)
- Is Trigger : True
- Offset : (0, -0.5)
- Size : (0.7, 0.7)
- Tag : DeadZone

Dead Zone

▶ GameMgr

- C# GameMgr 생성
- 현재 플레이어의 상태 정보를 게임 매니저에서 관리를 한다

```
public class GameMgr : MonoBehaviour
{
    private static GameMgr instance = null;
    public static GameMgr Instance {
        get
        {
            if (null == instance)
            {
                instance = new GameObject("GameMgr").AddComponent<GameMgr>();
                DontDestroyOnLoad(instance.gameObject);
            }
            return instance;
        }
    }

    private void Awake()
    {
        if (null == instance)
        {
            instance = this;
            DontDestroyOnLoad(instance.gameObject);
            return;
        }
        Destroy(gameObject);
    }

    public bool isDead { get; private set; }
    public void OnDie()
    {
        isDead = true;
    }
}
```

Dead Zone

▶ PlayerController

- “DeadZone”의 충돌 확인하여 플레이어의 움직임을 제한
- Die 애니메이션을 실행 후 현재 상태를 변경

```
public class PlayerController : MonoBehaviour
{
    void Update()
    {
        if (GameMgr.Instance.isDead || !rigid) return;
        ...
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag.Equals("DeadZone"))
        {
            if (rigid) rigid.simulated = false;
            if (anim) anim.SetTrigger("isDie");
            GameMgr.Instance.OnDie();
        }
    }
}
```

▶ SpriteScroll

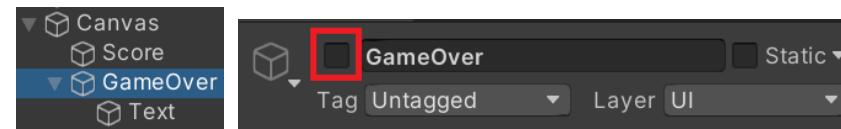
- 플레이어의 상태를 확인하여 스크롤을 멈춘다

```
public class SpriteScroll : MonoBehaviour
{
    void Update()
    {
        if (GameMgr.Instance.isDead) return;
        ...
    }
}
```

UI

▶ Score(Text)

- UI=>Text
- Anchor : (top, stretch)
- Pivot : (0.5, 1)
- Left, Pos Y, Pos z, Right : 0
- Height : 50
- Alignment : 가운데 정렬
- Best Fit : True



▶ GameOver(Image)

- UI=>Image
- Anchor : (stretch, stretch)
- Left, Top, Pos z, Right, Bottom : 0
- Image Color : (255, 255, 255, 150)
- **GameObject Active : false**
- **Child** : UI=>Text
- Anchor : (stretch, stretch)
- Pos z : 0
- Left, Top, Right, Bottom : 100
- Alignment : 가운데 정렬
- Best Fit : True
- Max Size : 170
- Color : (255, 0, 0)



UI

▶ GameMgr

- Score 데이터 출력

```
using UnityEngine.UI;
public class GameMgr : MonoBehaviour
{
    private Text score;
    private float scoreCount = 0;

    private void Awake()
    {
        var canvas = FindObjectOfType<Canvas>();
        if (canvas)
        {
            var scoreTr = canvas.transform.Find("Score");
            if (scoreTr) score = scoreTr.GetComponent<Text>();
        }
        ...
    }

    void Update()
    {
        if (false == isDead && score)
        {
            scoreCount += Time.deltaTime * 100;
            score.text = string.Format("Score : {0:N0}", Mathf.RoundToInt(scoreCount));
        }
    }
}
```

UI

▶ Animation Clip Event

- 애니메이션의 클립의 특정 프레임에 이벤트를 등록할 수 있다
- 이벤트를 등록할 오브젝트 선택
- 애니메이션 창의 클립을 변경
- 특정 프레임에서 [add event] 버튼 또는 [마우스 오른쪽 버튼 클릭=>Add Animation Event] 선택

▶ Animation Event

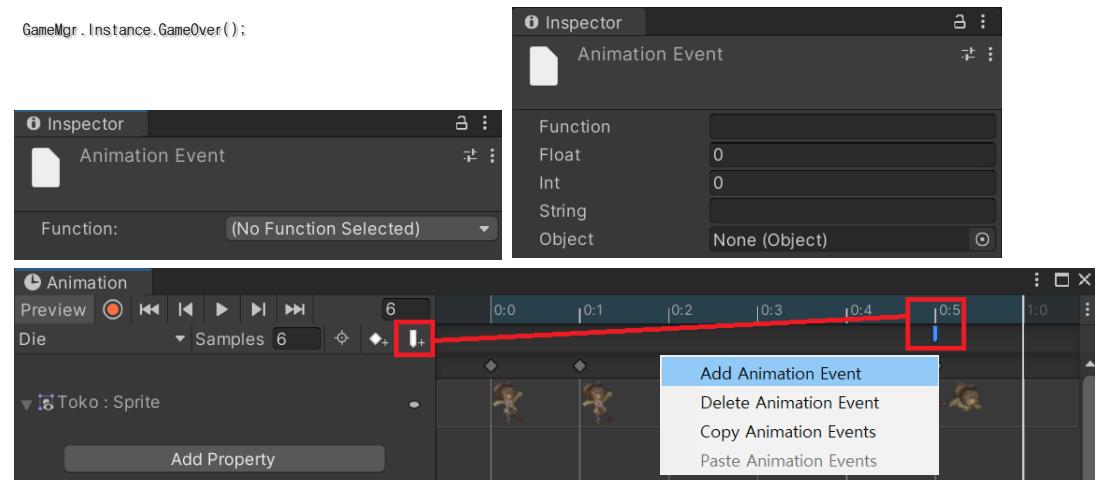
- 애니메이션 이벤트 등록 창은 두 종류가 있다
- Function에 원하는 호출 함수를 등록
(직접 입력하는 형식일 경우 해당 애니메이션이 있는 오브젝트에서 호출 가능한 함수 명을 적어야 한다)
- 인수가 필요할 경우 리스트 박스 형식에서는 자동으로 Parameters가 추가 되고 직접 호출 함수를 입력하는 형식일 경우는 하위에 있는 파라미터(float, int, string, object)에 데이터를 입력 및 등록하면 된다

▶ GameMgr, PlayerController

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

```
public class GameMgr : MonoBehaviour
{
    private GameObject gameOver;
    private void Awake()
    {
        var canvas = FindObjectOfType<Canvas>();
        if (canvas)
        {
            ...
            var gameOverTr = canvas.transform.Find("GameOver");
            if (gameOverTr) gameOver = gameOverTr.gameObject;
        }
    }
    public void GameOver()
    {
        if (gameOver) gameOver.SetActive(true);
    }
}
```

```
public class PlayerController : MonoBehaviour
{
    private void GameOver()
    {
        GameMgr.Instance.GameOver();
    }
}
```



Audio

▶ Audio Clip

- Mono, Stereo, 멀티 채널 오디오(최대 8개의 채널)
지원 포맷 : *.aif, *.wav, *.mp3, *.ogg
- 트래커 모듈(*.xm, *.mod, *.it, *.s3m)을 지원하며 일반 오디오 에셋과 같이 사용 가능

- Force To Mono : 멀티 채널 오디오를 Mono로 다운믹싱
- Normalize : Force To Mono 과정에서 정규화 한다
- Load In Background : 오디오 클립을 별도의 스레드에서 로드한다
- Ambisonic : 360도 동영상 및 XR 애플리케이션에서 사용
(앰비소닉 인코딩 오디오가 포함되어 있을 경우 활성화 된다)

Load Type

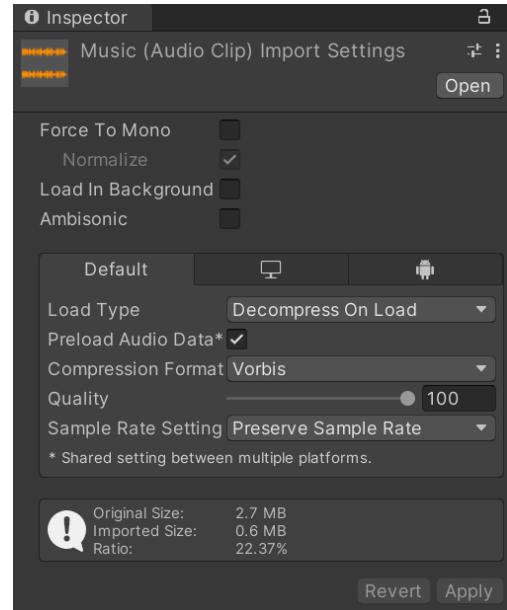
- > Decompress On Load : 로딩 후 오디오 파일의 압축을 해제(효과음과 같이 크기가 작은 파일에 사용)
- > Compressed In Memory : 메모리에서 압축 상태를 유지, 재생 시에 압축을 해제(BGM과 같은 크기가 큰 파일에 사용)
- > Streaming : Persistent Memory(HDD, Flash Driver)에 저장된 오디오 파일을 스트리밍 방식으로 재생(오디오 파일을 저장하기 위한 메모리가 필요 없다)
- Preload Audio Data : 씬(Scene)이 로딩될 때 오디오 클립을 미리 로드한다
(비활성화 상태일 경우 오디오 로딩:*.Play(), *.PlayOneShot(), *.LoadAudioData() 오디오 언로딩:*.UnloadAudioData())

Compression Format

- > PCM : 품질은 높아지는 대신 파일의 크기가 커진다(효과음에 적합)
- > ADPCM : 별소리, 충격음, 무기 소리와 같이 많은 노이즈를 포함한 대용량으로 재생되어야 하는 사운드에 적합
- > Vorbis/MP3 : 압축하여 크기가 줄어들지만 PCM 오디오에 비해 품질이 낮아진다
- Quality : 압축 비율을 지정(PCM/ADPCM/HEVAG 형식에는 사용되지 않는다)

Sample Rate Setting

- > Preserve Sample Rate : 샘플 레이트를 수정되지 않은 상태로 유지
- > Optimize Sample Rate : 하이 프리퀀시 콘텐츠에 따라 최적화
- > Override Sample Rate : 수동 오버라이드를 허용하여 실질적으로 주파수 성분을 제거할 때 사용



Audio

▶ die, jump Audio Clip

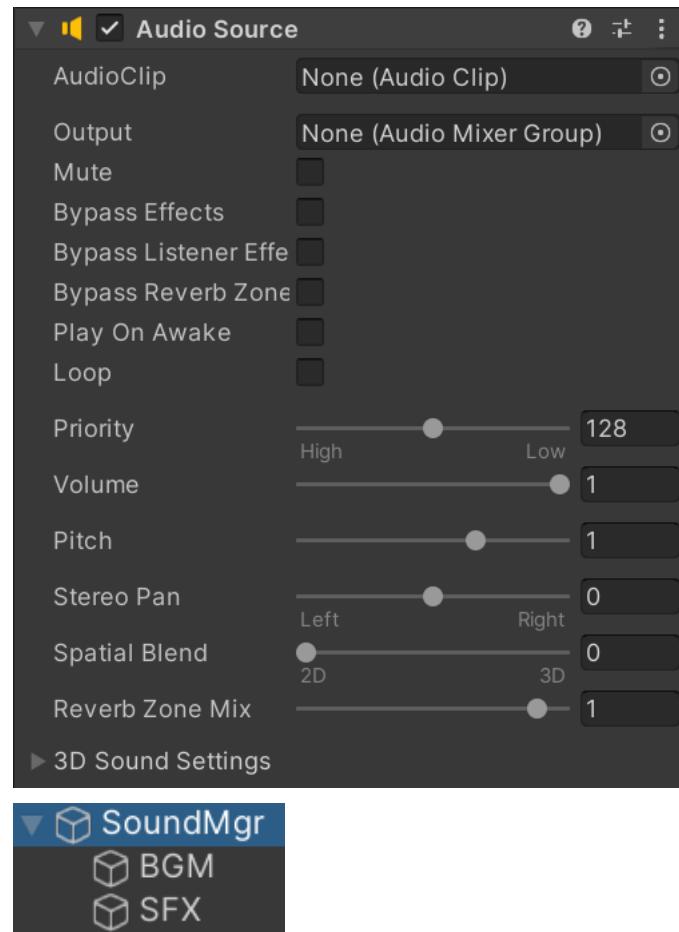
- Compression Format : PCM

▶ music Audio Clip

- Load Type : Compressed In Memory

▶ SoundMgr(GameObject)

- Create Empty
- C# SoundMgr 생성 및 등록
- Child : BGM(GameObject)
- AudioSource 추가
- Play On Awake : False
- Child : SFX(GameObject)
- AudioSource 추가
- Play On Awake : False



Audio

▶ SoundMgr(C#)

- Inspector 창에서 **BGM, SFX**을 등록
- **BGM은 반복재생**을 하고 SFX는 반복재생을 하지 않는다
- **Resources 폴더**에서 **Audio 파일**을 로드하여 **상황에 맞게 재생**한다

```
public class SoundMgr : MonoBehaviour
{
    public static SoundMgr Instance { get; private set; }
    private void Awake()
    {
        if (null == Instance)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);
            return;
        }
        Destroy(gameObject);
    }

    [SerializeField] AudioSource BGM;
    [SerializeField] AudioSource SFX;
    private AudioClip[] audioClips;
    private void Start()
    {
        audioClips = Resources.LoadAll<AudioClip>("Audio");
        if (BGM) BGM.loop = true;
        if (SFX) SFX.loop = false;
    }
}
```

Audio

```
public void PlayBGM(string name)
{
    if (BGM && 0 < audioClips.Length)
    {
        foreach (var clip in audioClips)
        {
            if (clip.name.ToLower().Equals(name.ToLower()))
            {
                BGM.clip = clip;
                BGM.Play();
                break;
            }
        }
    }
}

public void StopBGM()
{
    if (BGM)
    {
        BGM.Stop();
    }
}
```

Audio

```
public void PlaySFX(string name)
{
    if (SFX && 0 < audioClips.Length)
    {
        foreach (var clip in audioClips)
        {
            if (clip.name.ToLower().Equals(name.ToLower()))
            {
                SFX.PlayOneShot(clip);
                break;
            }
        }
    }
}

// public class SoundMgr
```

Audio

▶ GameMgr

```
public class GameMgr : MonoBehaviour
{
    private void Awake()
    {
        SoundMgr.Instance.PlayBGM("music");
        ...
    }
    public void OnDie()
    {
        isDead = true;
        SoundMgr.Instance.StopBGM();
    }
}
```

Audio

▶ PlayerController

```
public class PlayerController : MonoBehaviour
{
    void Update()
    {
        ...
        if (Input.GetKeyDown(KeyCode.Space) && limitJumpCount > jumpCount)
        {
            ...
            SoundMgr.Instance.PlaySFX("jump");
        }
        else if ...
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag.Equals("DeadZone"))
        {
            ...
            SoundMgr.Instance.PlaySFX("die");
        }
    }
}
```