(개발 시연 영상 보기)

(깃허브 바로가기)

좌측 아이콘 클릭 시, 해당 페이지로 이동합니다.

프로젝트 소개

2D Game Engine

(Vega Engine)

기간 | 2024.11.15 ~ 2024.12.15 (4주)

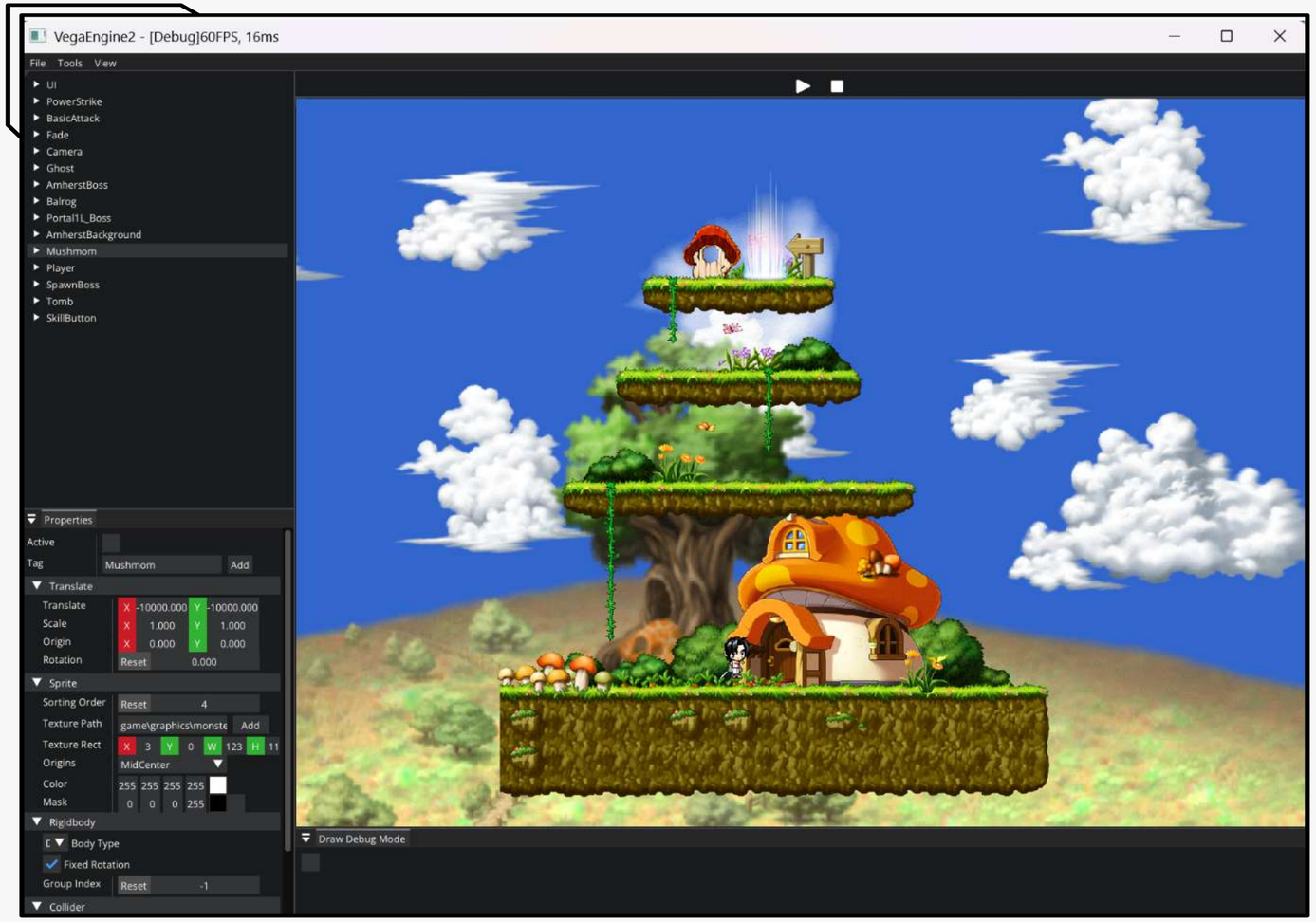
인원 | 1인 개발

장르 | 개인 / 2D 게임 엔진 / C++

언어 | C++11

주요 역할

1. 2D 게임을 위한 Unity 모방 게임 엔진 개발
2. 팀프로젝트에서 팀원에게 엔진 사용법 설명

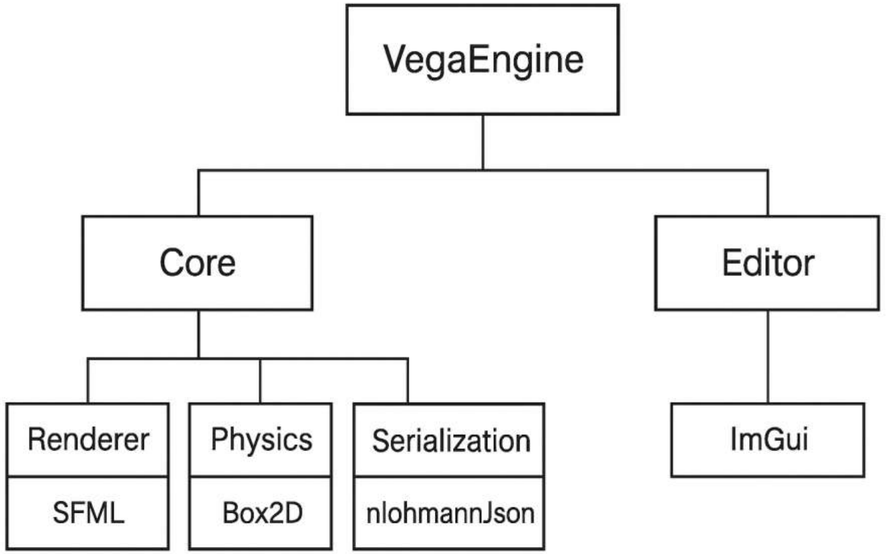


01 엔진의 구조

01

기능 별 라이브러리 목록

1. **Entity Component System**
   - EnTT (유니티와 같은 ECS 구현을 위한 기능)
2. **2D Renderer**
   - SFML 2.6.1 (윈도우 관리 및 렌더링 기능)
3. **Graphical User Interface (GUI)**
   - ImGui (에디터 UI를 구성하기 위한 기능)
4. **2D Physics System**
   - Box2D (게임의 물리를 담당하는 기능)
5. **Serialization**
   - nlohmanJson (GameObject, Scene등을 저장하기 위한 기능)



한 달이라는 제한된 기간 안에 완성하기 위해, 오픈소스 라이브러리를 적극 활용하여 게임 엔진을 구현했습니다.

02 Prefab 기능

01

목표

- Unity의 Prefab 기능을 모방하여 구현한다.

02

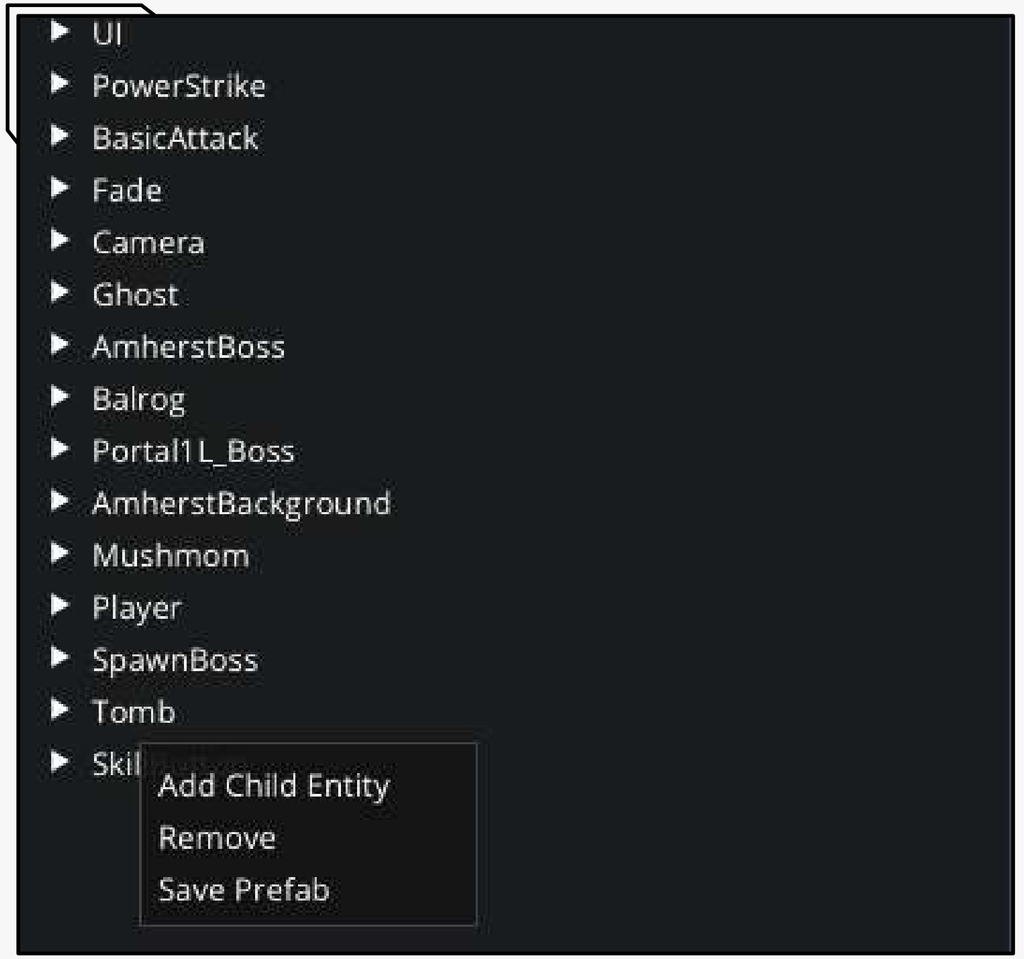
방법

- 각 오브젝트에 대한 정보를 Json으로 직렬화
- UUID도 저장하여 데이터에 대한 상속 관계도 저장

03

결과

- 기존의 오브젝트를 저장하거나 런타임에 생성할 수 있게 되었습니다.



02 Prefab 기능

04

구현 흐름

1. 연결된 모든 컴포넌트들을 Json으로 직렬화

↳ 프로그램이 종료되어도 데이터들이 유지

2. 자식 게임 오브젝트에 대한 정보들을 UUID로 저장

↳ 상속 관계와 연결 관계는 역직렬화 시, UUID가 필요

3. Scene을 Load하거나 프로젝트 로드 시, 역직렬화

↳ Instanciate, Scene 기능에 활용 가능

(총알 Prefab으로 런타임에 Instance 생성)

02 Prefab 기능

05

코드 설명

instantiate 기능은 Entity들을 런타임에 객체화 하는 방식입니다.

instance가 되어 씬에 들어왔을 때, 실제 물리 엔진 World에 바인딩되는 구조

Instantiate를 호출하면 원하는 prefab으로 런타임에 새로운 인스턴스를 생성할 수 있습니다.

```
public GameObject Instantiate(string path, Transform parent = null)
{
    GameObject prefab = Load<GameObject>($"Prefabs/{path}");
    if (prefab == null)
    {
        Debug.Log($"Failed to load prefab : {path}");
            return null;
    }
    return Object.Instantiate(prefab, parent);
}
```

```
if (AttackElap >= AttackTime && InputManager::IsKeyPressed(KeyType::Z))
{
    AttackElap = 0.0f;
    InputKey.key1 = KeyType::Z;
    GameObject obj = FZ_CURRENT_SCENE->GetEntityFromTag("Bullet");
    float angle = Utils::Angle({ fx, fy });
    sf::Vector2f pos = transform.Transform.GetTranslate();
    pos.y += -15.f;
    sf::Vector2f dir = Utils::GetRotateVector(angle, sf::Vector2f(20.f, 0.0f));
    FZ_CURRENT_SCENE->Instantiate(obj, pos + dir, angle);
}
```

03 Hierarchy

01

목표

- Unity의 Hierarchy 기능을 모방하여 구현한다.

02

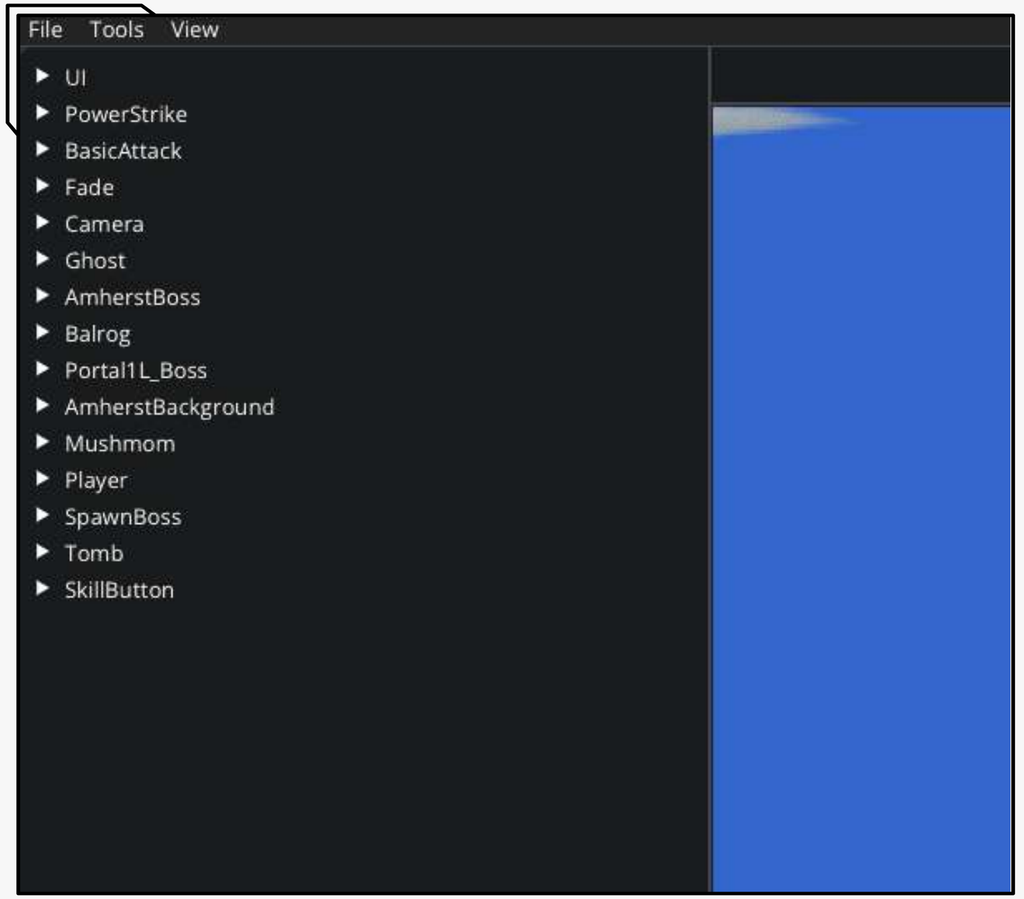
방법

- GameObject들을 생성, 삭제 가능
- 부모-자식 관계로 Child GameObject 생성 가능

03

결과

- Hierarchy에 있는 정보들로 Scene 데이터 구성 가능



03 Hierarchy

04

구현 흐름

1. Prefab으로 직렬화된 데이터를 UI으로 표시

↳ ImGui를 활용해서 Hierarchy 구성

2. 부모-자식 관계는 Stack과 재귀를 활용하여 구현

↳ ImGui의 Child 생성 코드를 직렬화 데이터랑 동기화

3. Save/Load

↳ 3. 게임의 Save/Load 시, 해당 데이터들로 Scene 저장

```
bool AnimationClip::loadFromFile(const std::string& filePath)
{
    if (!Utils::CanFileOpen(filePath))
        return false;

    Database::LoadFromJson(filePath);
    auto& json = Database::GetJsonObject(filePath);
    auto& target = json["AnimationClip"];

    std::string loopStr;
    id = target["ClipName"];
    loopStr = target["LoopType"];
    path = target["TexturePath"];
    int size = target["FrameCount"];
    auto& translate = target["Transform"]["Translate"];
    auto& Rotation = target["Transform"]["Rotation"];
    auto& Scale = target["Transform"]["Scale"];
    transform.SetTranslate({ translate[0], translate[1] });
    transform.SetRotation(Rotation);
    transform.SetScale({ Scale[0], Scale[1] });
```

(가중치를 이용하여 랜덤 생성을 하기 위한 랜덤 매니저의 Static 메서드)

04 Components

01

목표

- Unity의 Component 기능들을 모방하여 구현한다.

02

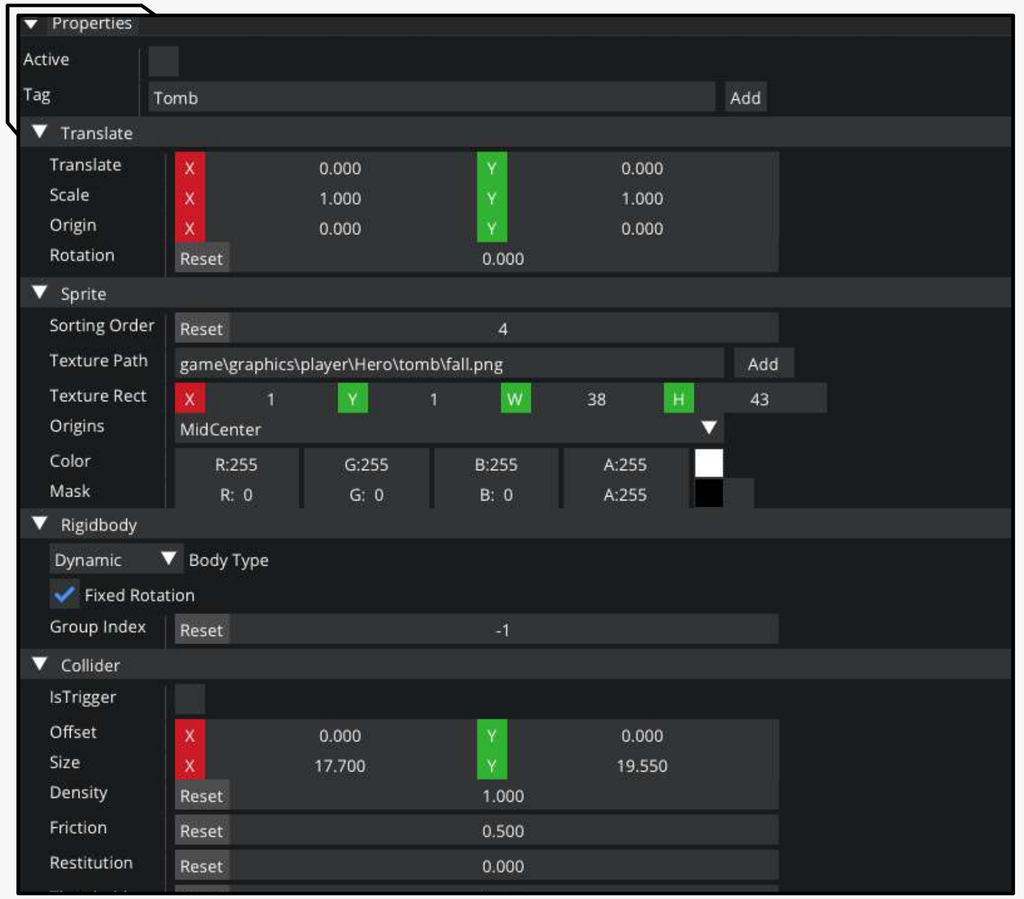
방법

- Entt 라이브러리를 활용하여 ECS 시스템 구성
- 커스텀 Component들을 구현

03

결과

- Transform, Rigidbody, Collider 등 다양한 컴포넌트 구현



04 Components

04

구현 흐름

1. Prefab으로 직렬화된 데이터를 UI으로 표시

↳ ImGui를 활용해서 Hierarchy 구성

2. 부모-자식 관계는 Stack과 재귀를 활용하여 구현

↳ ImGui의 Child 생성 코드를 직렬화 데이터랑 동기화

3. Save/Load

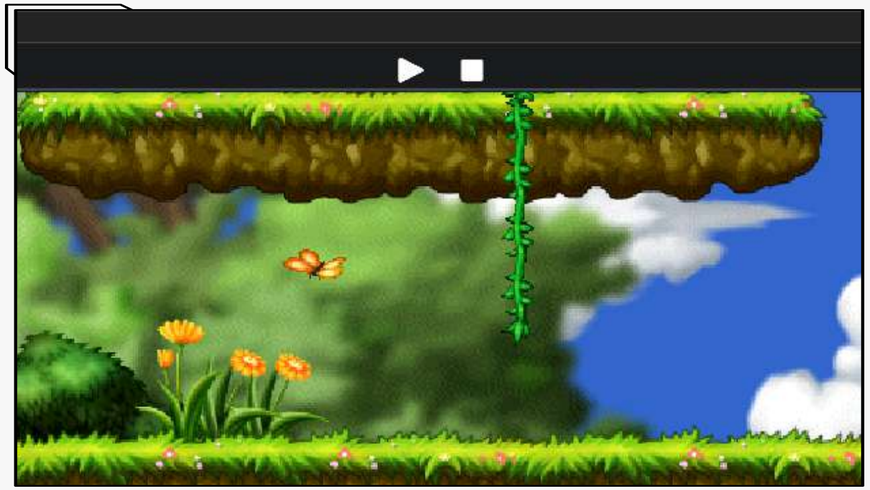
↳ 3. 게임의 Save/Load 시, 해당 데이터들로 Scene 저장

```
void OnUpdate(float dt)
{
    auto& transform = GetComponent<TransformComponent>();
    auto& body = GetComponent<RigidbodyComponent>();
    body.SetGravityScale(0.0f);

    float angle = transform.Transform.GetRotation();
    sf::Vector2f dir = Utils::Angle(angle);
    body.AddPosition({ Speed * dir.x, Speed * dir.y });
}
```

(게임 오브젝트를 이동시키는 스크립트 코드)

05 그외 구현 사항



```
mespace fz {

  class GhostScript : public VegaScript
  {
  public:
      GameObject player;
      float speed = 100.f;
      float radius = 20.0f;
      fz::Transform transform;
      float angle = 0;
      sf::Vector2f centor;
```

테스트 플레이 기능

Game Start/Stop 버튼을 통해서 에디터 상에서 게임을 테스트할 수 있습니다.

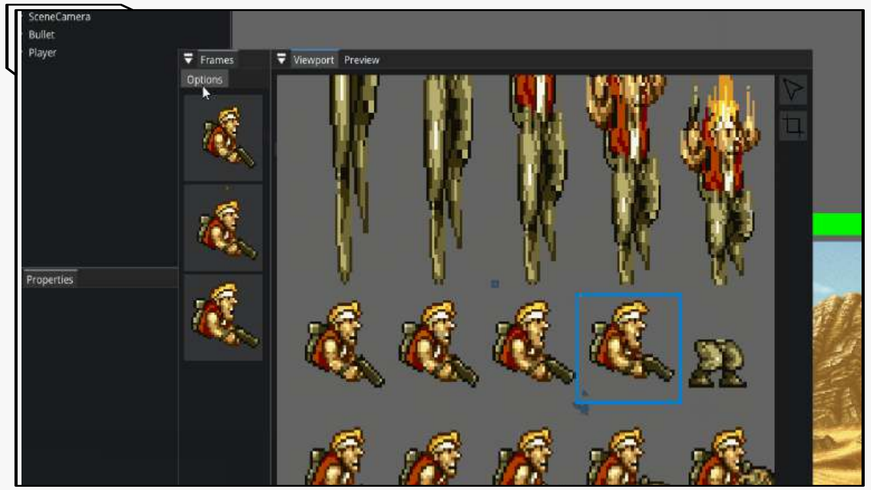
스크립팅 시스템

Vega-Engine은 Rttr 라이브러리의 Reflection 기능을 활용한 스크립팅 시스템을 제공합니다.

05 그외 구현 사항

```
void Start() override
{
    clips["Portal"].loadFromFile("game/animations/effect/p
    TransformComponent& transform = GetComponent<Transform
    sf::Sprite& sprite = GetComponent<SpriteComponent>();
    animator.SetTarget(GetCurrentEntity());
    animator.Play(&clips["Portal"]);

    auto& comp = AddComponent<PortalComponent>();
    comp.NextPlayerPos = { 671.f, 228.f };
    comp.NextScenePath = "game/scene/Stage1_hunt.vega";
    SoundMgr::Instance().PlayBgm("game/sound/Hunt_bgm.mp3"
}
```



Animation Clip

아틀라스 이미지를 활용해서 애니메이션을 구성할 수 있는 기능입니다.

Sprite Editor

Animation Clip을 프레임 단위로 편집할 수 있는 에디터 기능입니다.