SurvivalShooter チュートリアル

Survival Shooter チュートリアル
https://learn.unity.com/project/survival-shooter-tutorial

Survival Shooter Project Assets
https://assetstore.unity.com/packages/essentials/tutorial-projects/survival-shooter-tutorial-legacy-40756

Slides
https://oc.unity3d.com/index.php/s/xQbGL7Fm3mF0ySs

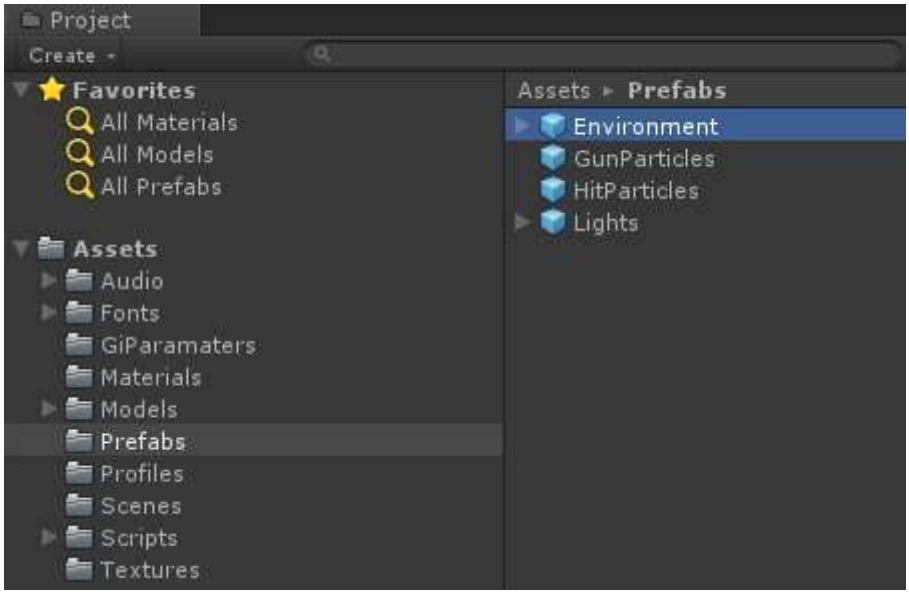
Phase1 環境のセットアップ

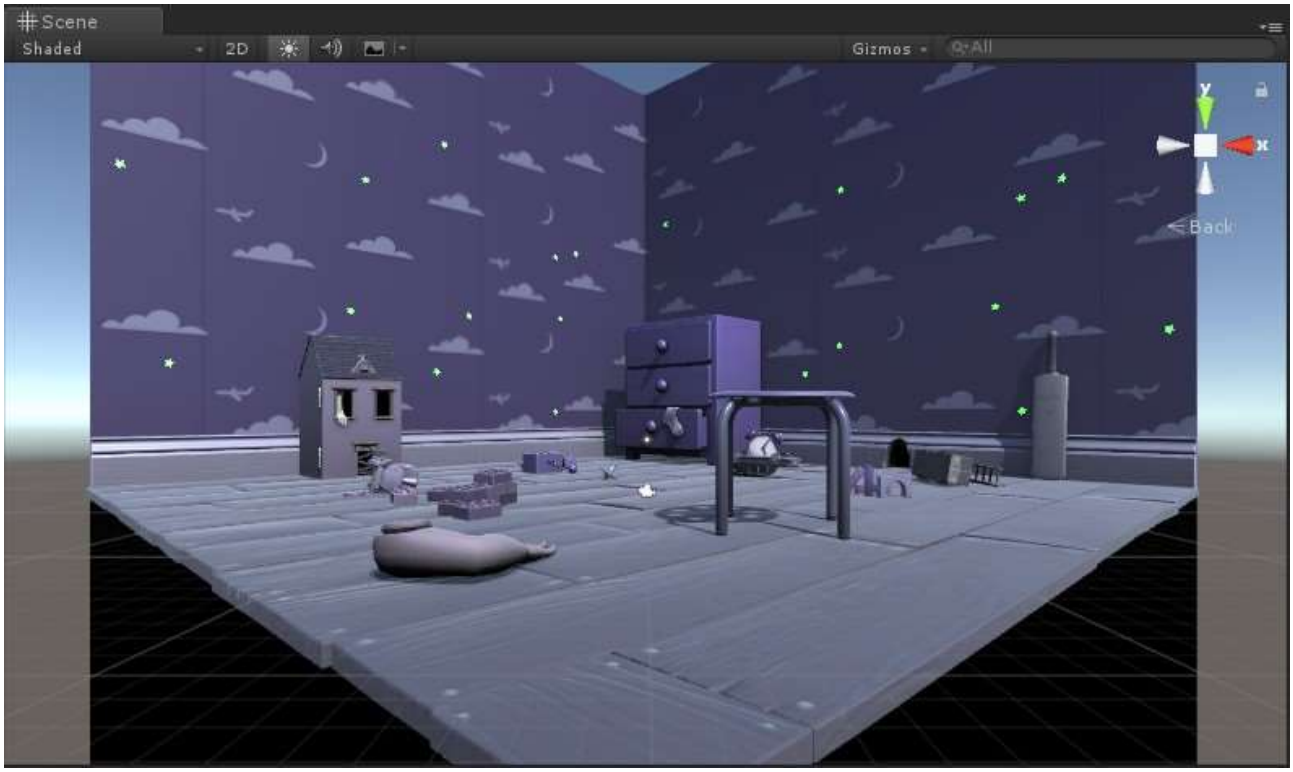
１．シーンの作成

File > Save scene as をクリックし，シーンの名前を「Level01」として，Scenes フォルダに保存。

２．ステージの構成要素配置

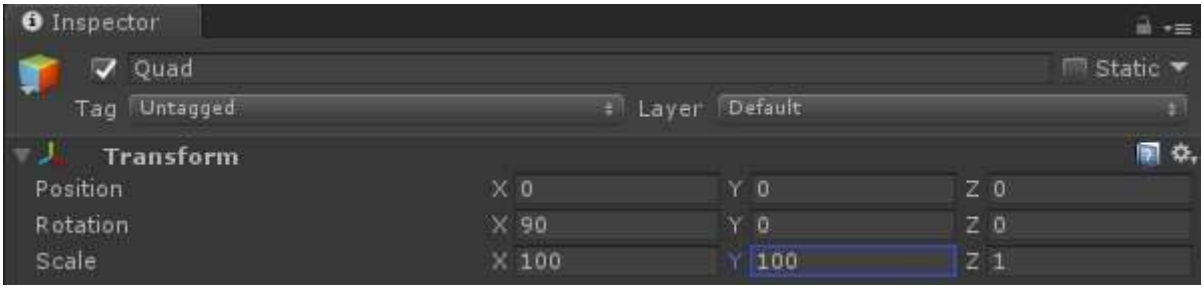
Prefabs フォルダ内の Environment プレハブを Scene または Hierarchy いずれかにドラッグする。Transform の position は 0，0，0 にする。



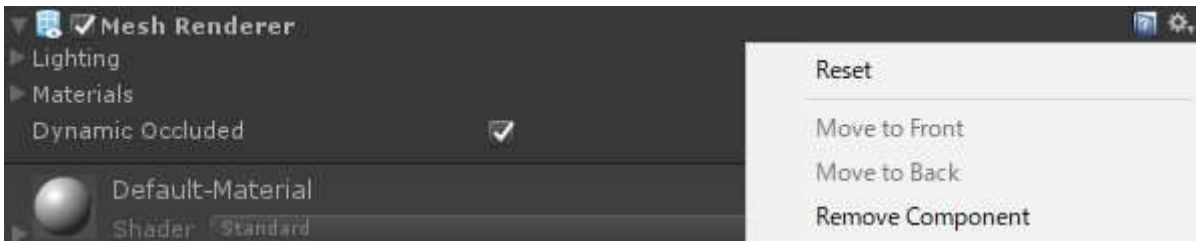


Prefabs フォルダ内の Light プレハブを Scene または Hierarchy いずれかにドラッグする。シーンにあらかじめ配置されている Directional Light は必要ないので削除する。

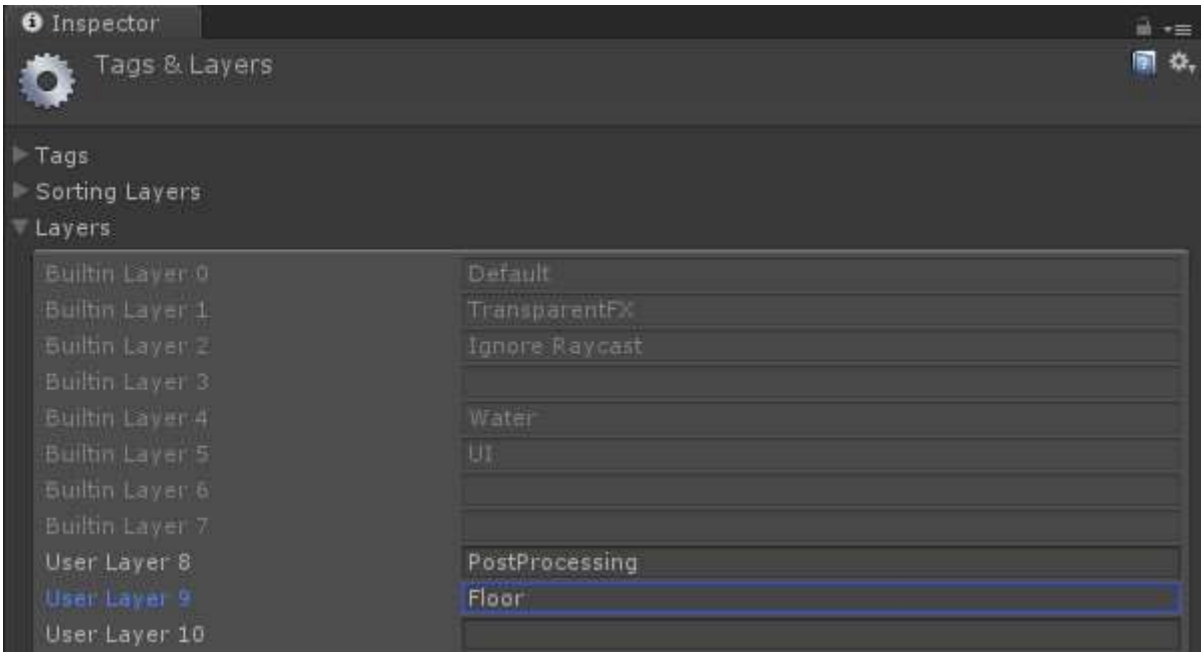
Hierarchy > Create >3D Object > Quad をクリックし，当たり判定用の床オブジェクトを追加する。
Transform を以下のように設定。



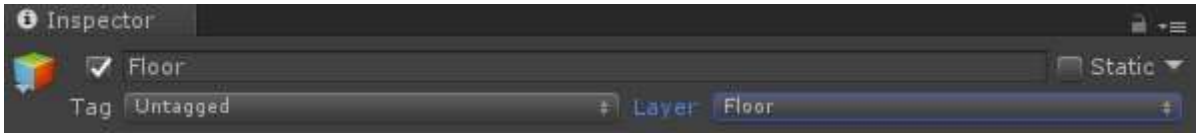
オブジェクト名を Floor に変更し，Mesh Render コンポーネントを削除（Remove Component）する。



新しく Floor レイヤーを追加する。

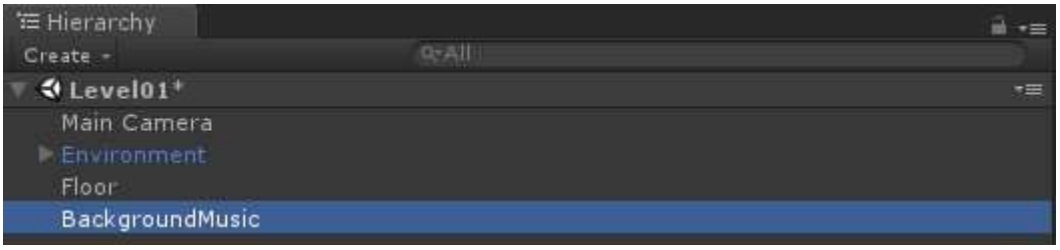


Floor オブジェクトのレイヤーを Floor に変更する。

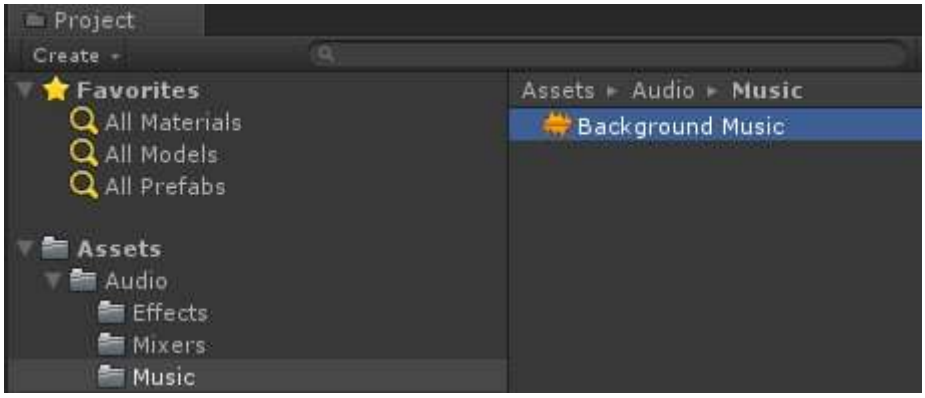


３．BGM の設定

Hierarchy > Create > Create Empty で空のオブジェクトを作成し, BackgroundMusic に名前を変更する。



Add Component > Audio > Audio Source で Audio Source コンポーネントを追加し，BGM として BackgroundMusic を AudioClip に設定する。



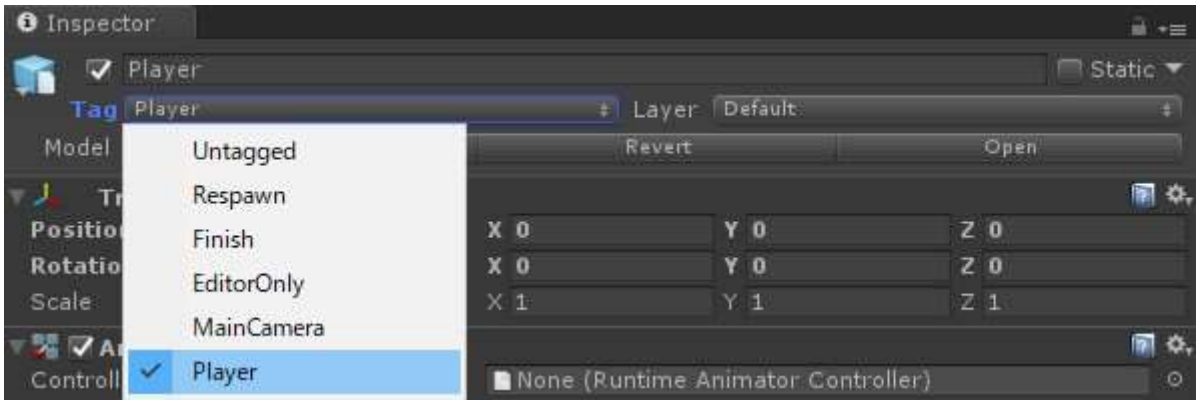
BGM は繰り返し再生させるので，Loop にチェックを入れ，Volume は 0.1 程度に設定する。Volume は実際に実行後，調整する。

Phase2　プレイヤーキャラクタ

1．プレイヤーキャラクタの配置

Project > Assets > Models > Character フォルダ内の Player を Scene または Hierarchy いずれかにドラッグする。Transform の position は 0，0，0 にする。

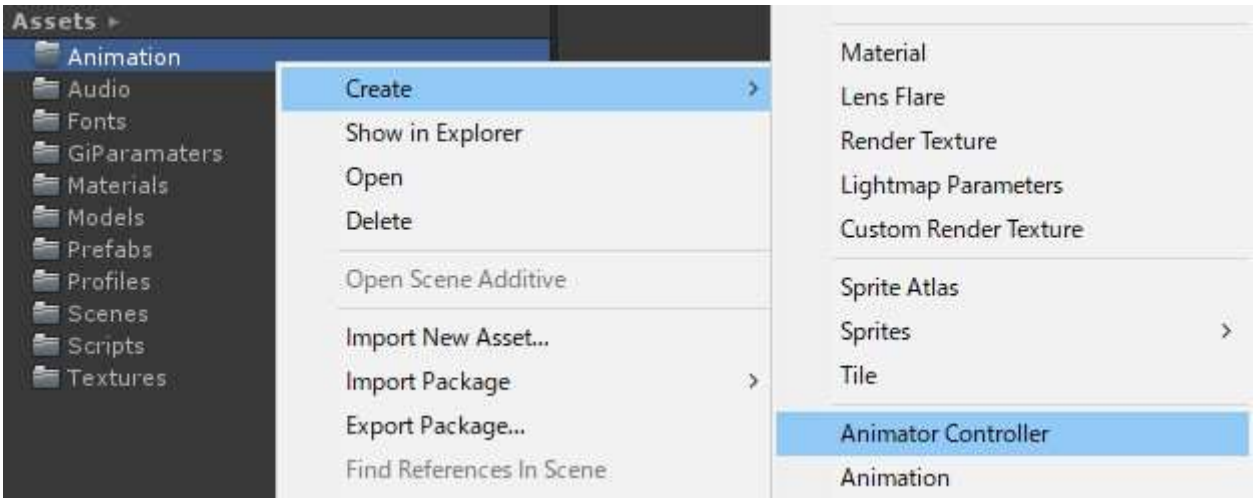
Player オブジェクトの Tag を Player（デフォルトで設定済みのタグ）に変更する。

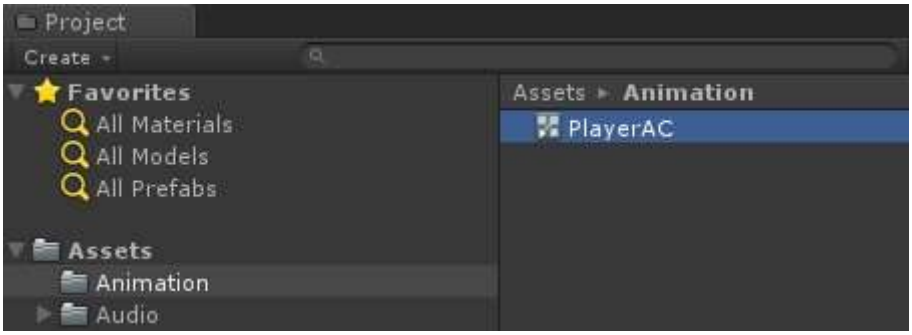


２．AnimatorController の作成

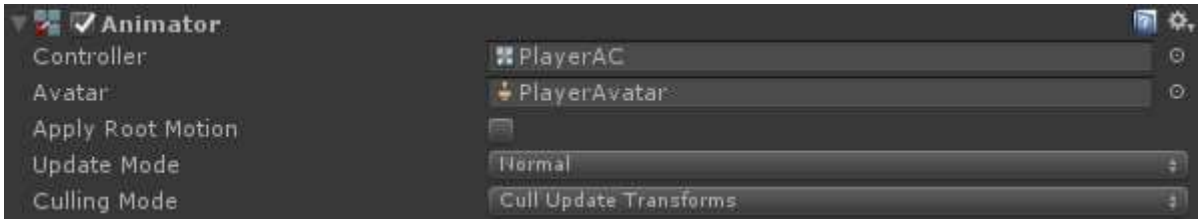
アニメーションを制御するためのアセットが AnimatorController で，アニメーションさせたいオブジェクトに，Animator コンポーネントを追加し，AnimatorController を設定することで，オブジェクトのアニメーションを制御する。

Project に Animation フォルダを新たに作成し，Animation フォルダ内に AnimatorController を作成する。名前は PlayerAC とする。



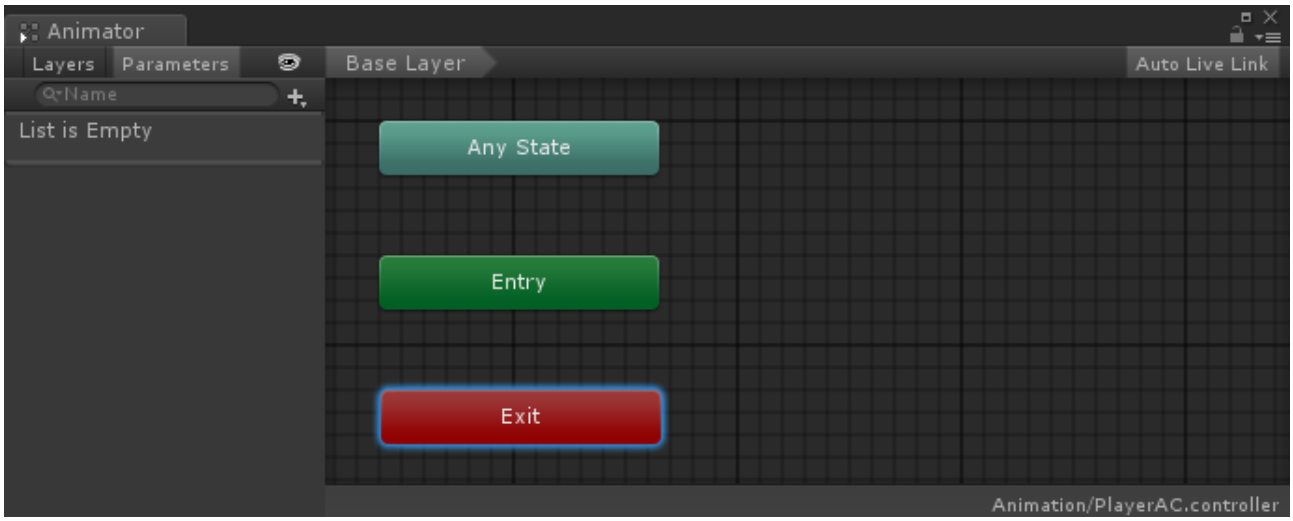


作成した PlayerAC を Hierarchy の Player にドラッグする。Animator コンポーネントの Contoller に PlayerAC が設定される。

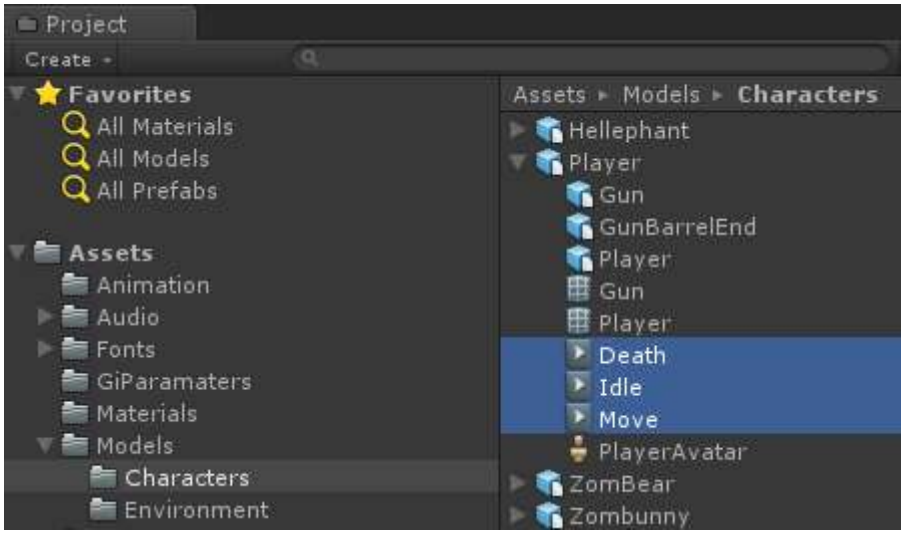


３．PlayerAC の設定

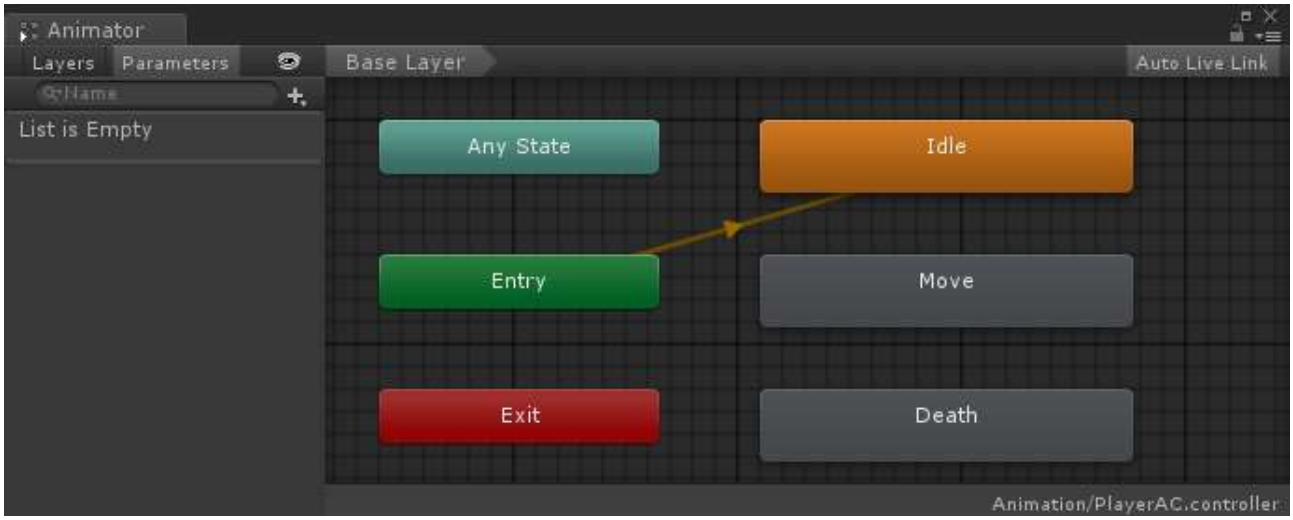
Project 内の PlayerAC をダブルクリックするか，Hierarchy でオブジェトを選択後，Window > Animator で，Animator ウインドウを開く。



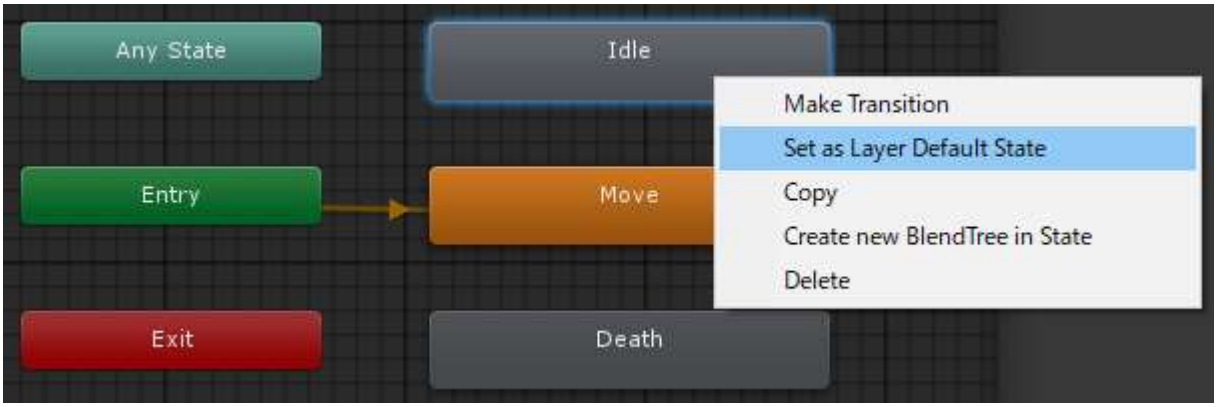
Project > Asset > Models > Characters > Player の「Idle」「Move」「Death」を Animator ウインドウの空白部分にドラッグする。



Idle アニメーションをデフォルトのアニメーションにする。



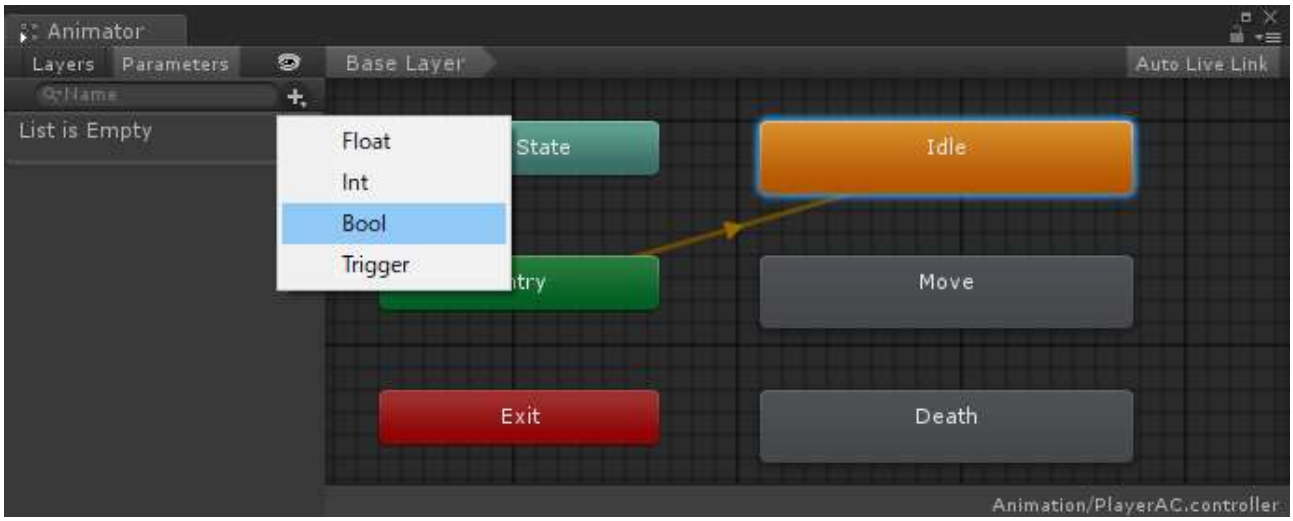
アニメーションは Entry から開始され，最初に開始されるアニメーションがデフォルトのアニメーションになる。デフォルトのアニメーションを変更する場合は，デフォルトにしたいアニメーションを右クリックし，Set as Layer Default State をクリックする。

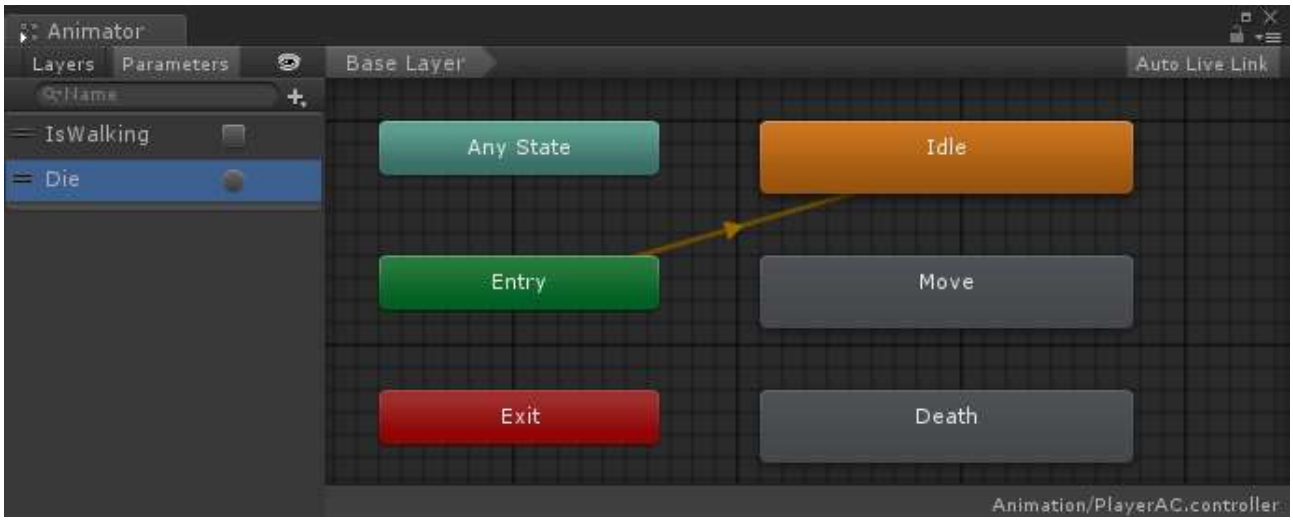


アニメーションを切り替えるためのパラメータを作成する。

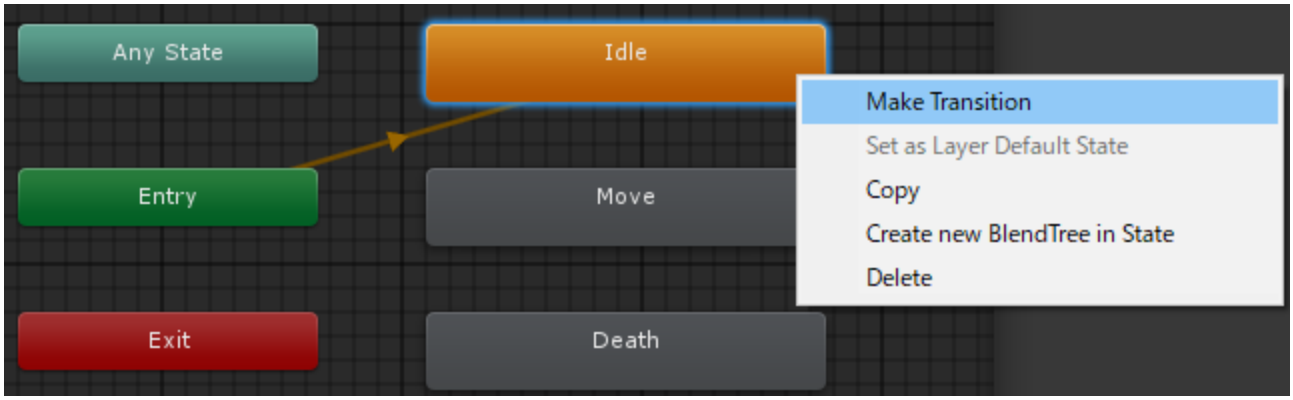
Parameters > + をクリックし，リストから Bool をクリックする。これで，Bool 型のパラメータが作成されるので，名前を IsWalking とする。

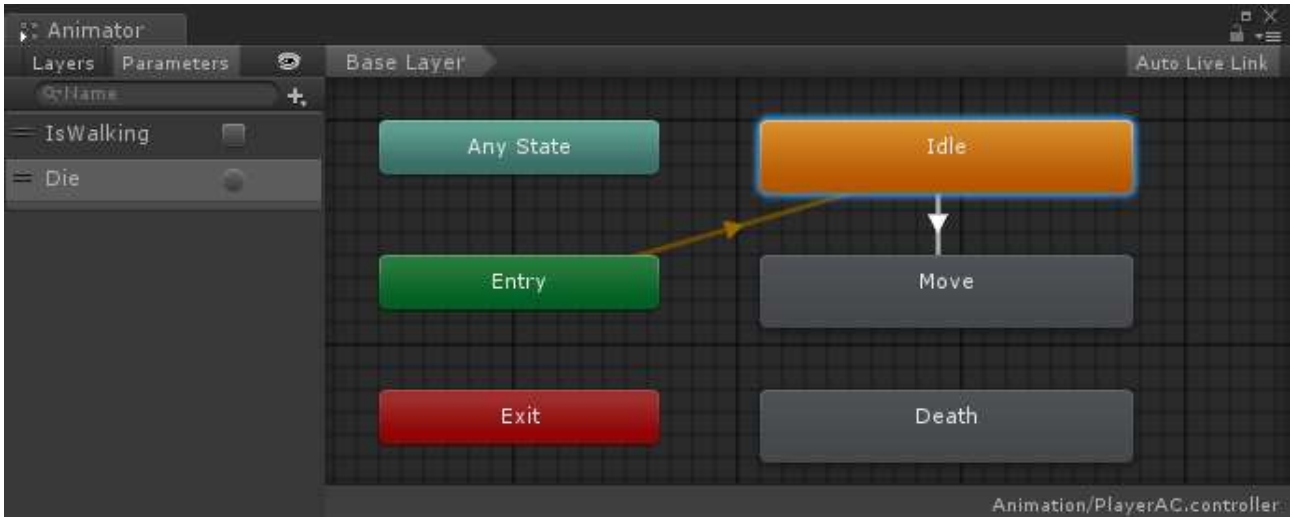
同様に，Tigger 型のパラメータ Die を作成する。





アニメーションの状態を遷移させるため，Idel アニメーションを右クリックし，Make Transition を選択する。矢印が表示されるので Move アニメーションにつなぐ。

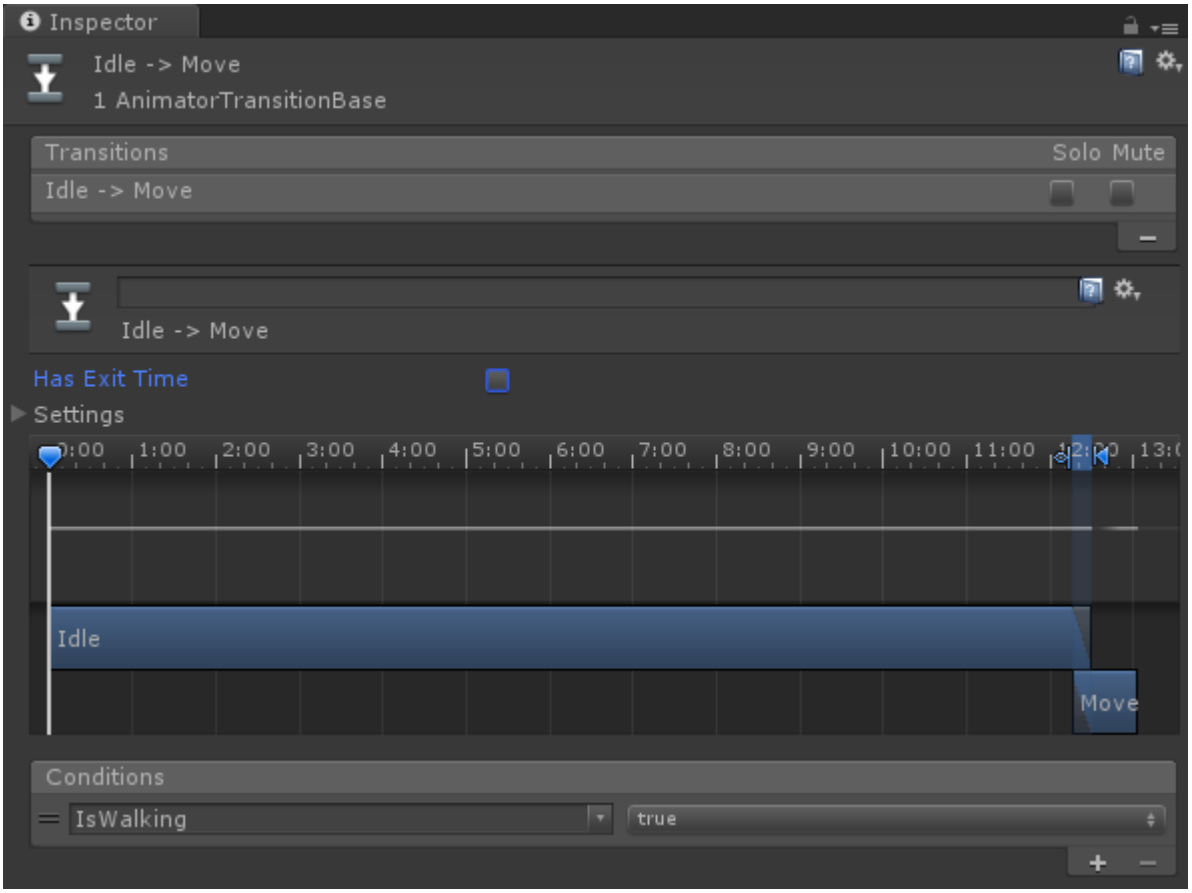


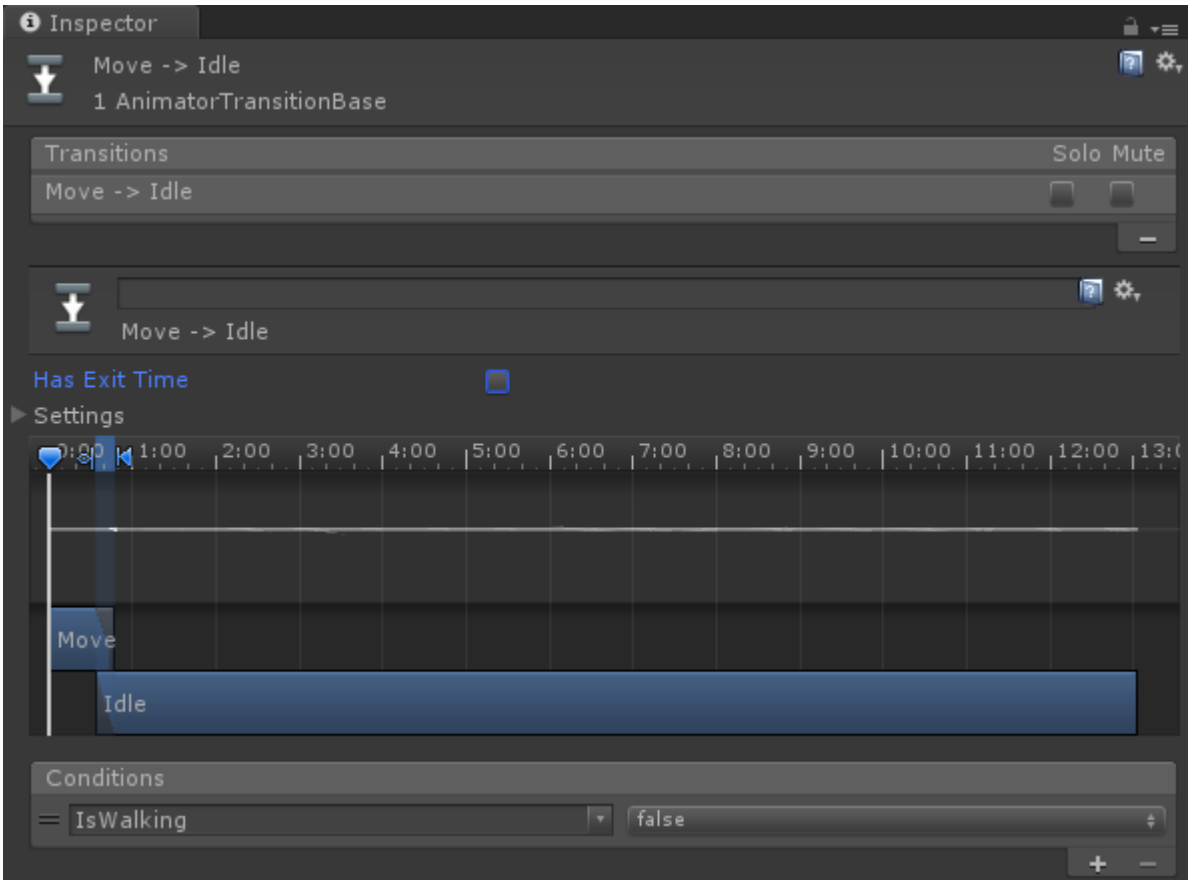


Idele アニメーションから Move アニメーションへの矢印をクリックし，Inspector に設定項目を表示させる。

Has Exit Time のチェックを外し，Conditions の ＋ をクリックし，アニメーションが切り替わる条件として，IsWalking が true を追加する。

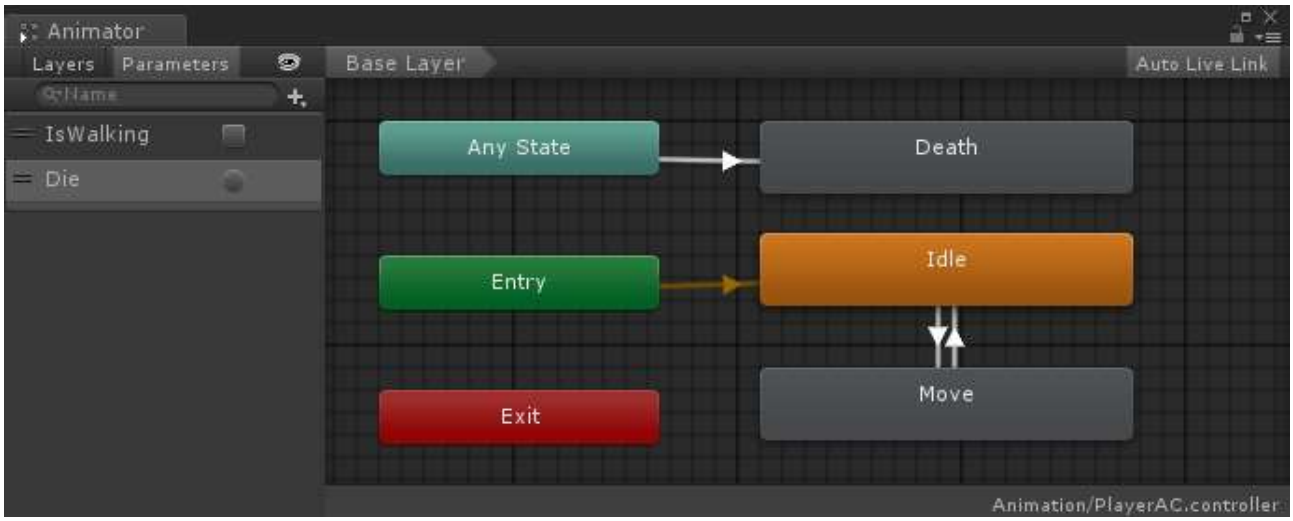
同様に，Move アニメーションから Idle アニメーションへの遷移を作成し，Conditions に IsWalking が false を設定する。Has Exit Time のチェックも外す。

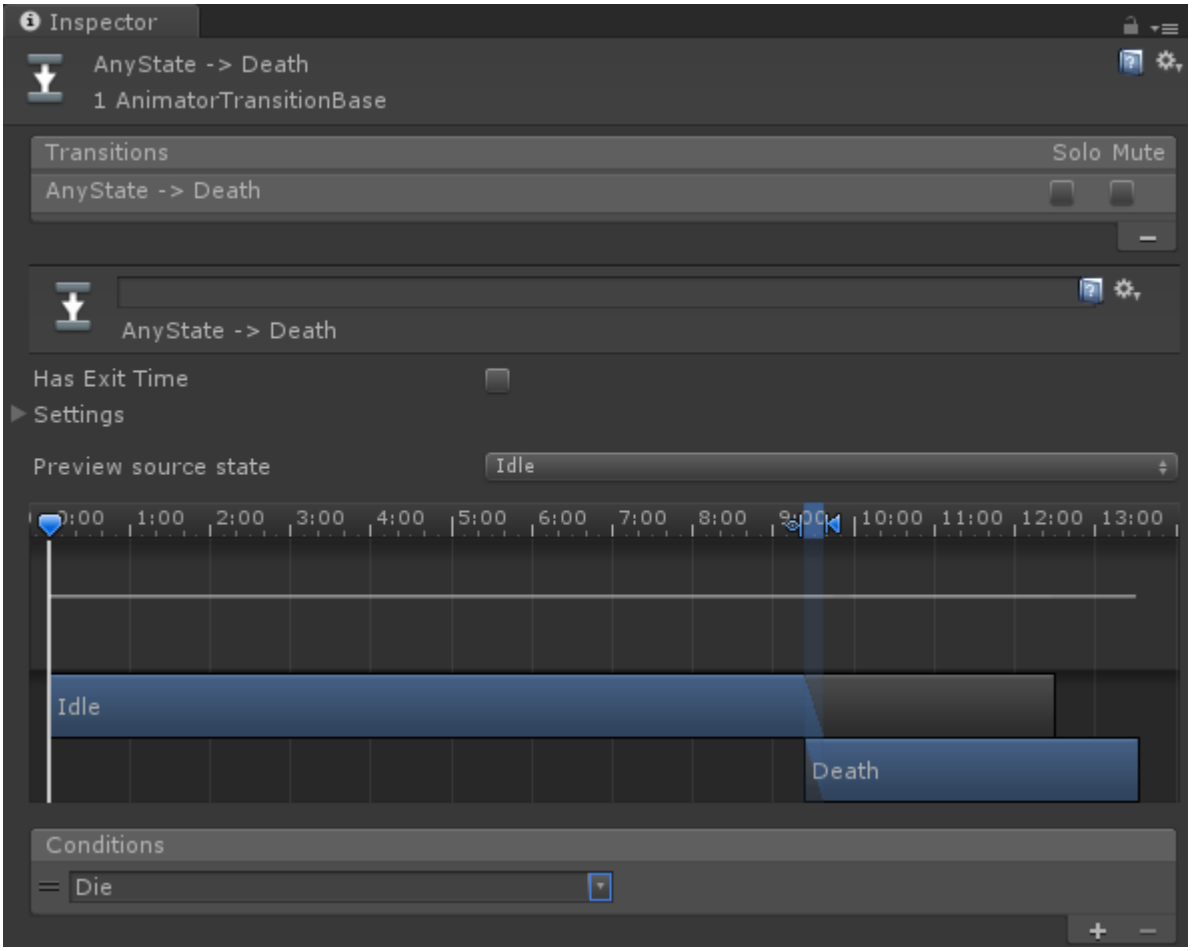

Inspector
Idle -> Move
1 AnimatorTransitionBase
Transitions
Solo Mute
Idle -> Move
Idle -> Move
Has Exit Time
Settings
1:00
2:00
3:00
4:00
5:00
6:00
7:00
8:00
9:00
10:00
11:00
Idle
Move
Conditions
IsWalking
true


Inspector
Move -> Idle
1 AnimatorTransitionBase
Transitions
Solo Mute
Move -> Idle
Move -> Idle
Has Exit Time
Settings
1:00
2:00
3:00
4:00
5:00
6:00
7:00
8:00
9:00
10:00
11:00
12:00
Move
Idle
Conditions
IsWalking
false

Die アニメーションは Any State から遷移するようにし，遷移の条件として Die を追加する。

Any State はその名の通り，いずれかのアニメーション中でも条件を満たした場合にアニメーションを遷移させる場合に使用する。





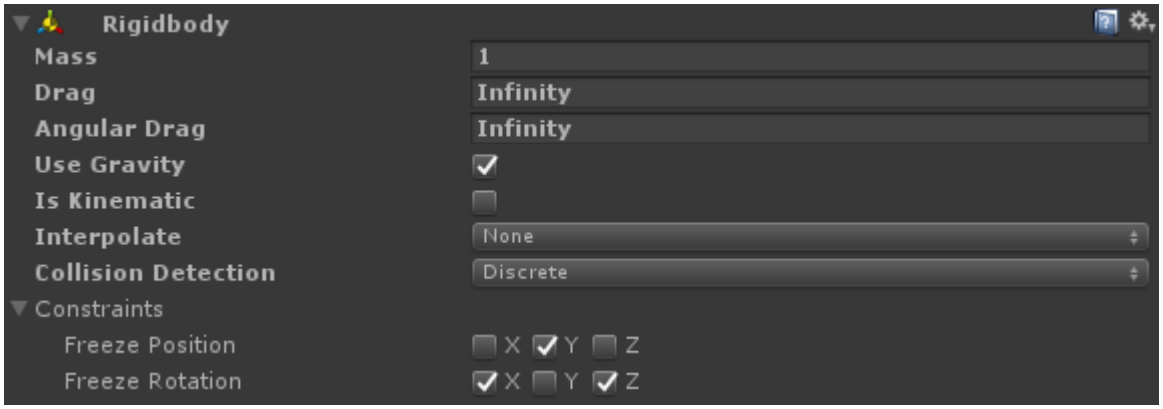
４．プレイヤーキャラクタへのコンポーネントの追加

プレイヤーキャラクタを動かすのに必要なコンポーネント，Rigidbody と Capsule Collider，Audio Source を Player オブジェクトに追加する。

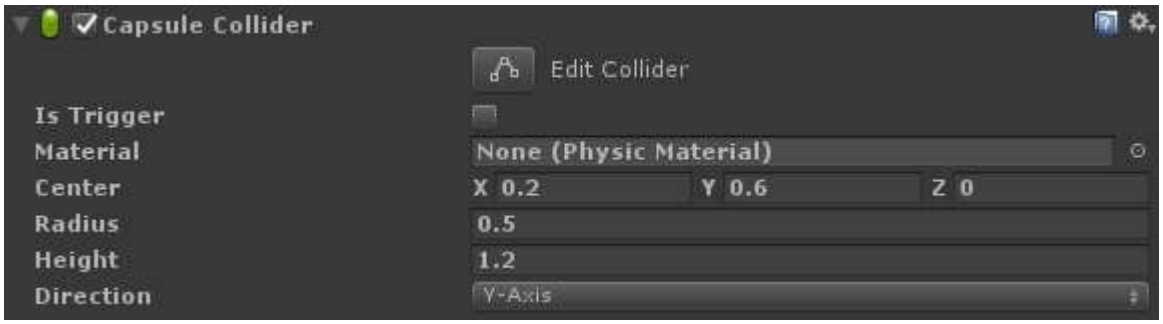
Rigidbody コンポーネントの Drag と Angular Drag に Infinity を設定する。Constraints を展開し，Freeze Position の Y，Freeze Rotation の X と Z をチェックする。

Drag は空気抵抗を示し，Infinity は無限大の意味になる。Drag に Infinity を設定するとオブジェクトは直ちに動きを止める。Angular Drag は回転時の空気抵抗を示す。

Freeze Position と Freeze Rotation では，移動や回転を停止する軸にチェックを入れる。



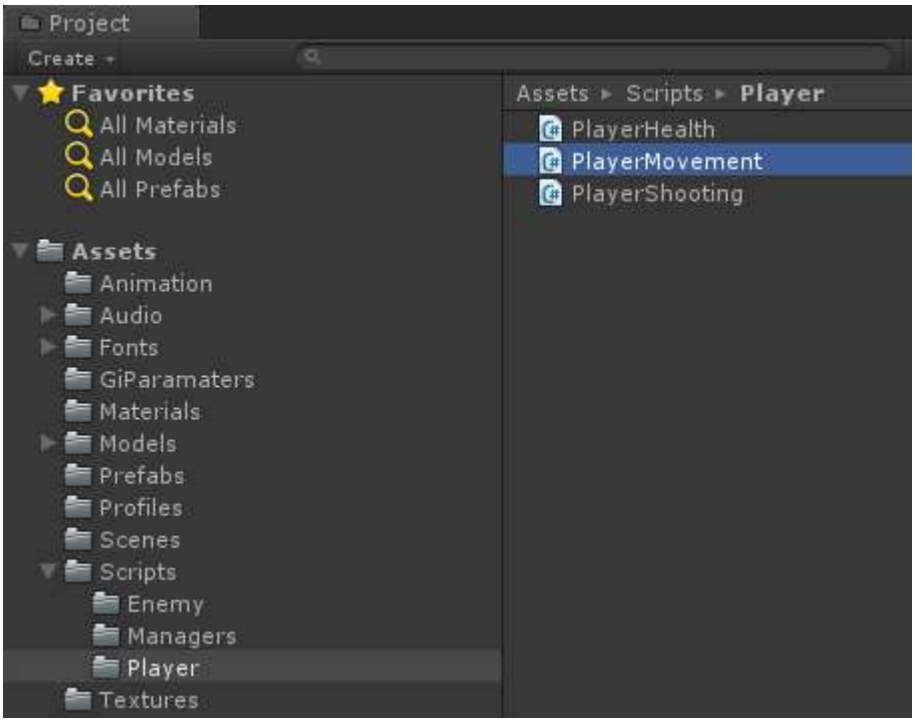
Capsule Collider コンポーネントの Center X には 0.2 を Y には 0.6 を設定する。Height は 1.2 にする。



Audio Source コンポーネントには，プレイヤーが攻撃を受けた時の効果音 Player Hurt を AudioClip に設定する。Play On Awake のチェックは外す。

５．プレイヤーキャラクタを動かすスクリプトの作成

Project > Assets > Scripts > Player > PlayerMovement を Player にドラッグする。



PlayerMovement スクリプトをダブルクリックし開く。

以下のコードを追加する。

```
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    public float speed = 6f;   // プレイヤーの移動速度

    Vector3 movement;          // プレイヤーの移動方向
    Animator anim;             // アニメーションコンポーネント
    Rigidbody playerRigidbody;      // Rigidbody コンポーネント
    int floorMask;                  // Floor と rayermask を使う
    float camRayLength = 100f;      // カメラからの ray

    void Awake()
    {
        floorMask = LayerMask.GetMask("Floor");      // Floor の rayermask を作成

        anim = GetComponent<Animator>();             // Animator コンポーネントを取得
        playerRigidbody = GetComponent<Rigidbody>(); // Rigidbody コンポーネントを取
得
    }

    void FixedUpdate()
    {
        // インプットから左右上下の移動量を-1 もしくは 1 で受け取る
        float h = Input.GetAxisRaw("Horizontal");
        float v = Input.GetAxisRaw("Vertical");
```

```
        Move(h, v);         // プレイヤーを動かす Move()を呼ぶ
        Turning();          // プレイヤーの方向を動かす Turning()を呼ぶ
        Animating(h, v);  // プレイヤーのアニメーションを設定する Animating()を呼ぶ
    }

    void Move(float h,float v)
    {
        movement.Set(h,0f,v);      //移動量を設定

        // 移動するベクトルを 1 にし，移動する距離を設定する
        movement = movement.normalized * speed * Time.deltaTime;

        // プレイヤーのポジションを動かす
        playerRigidbody.MovePosition(transform.position + movement);
    }

    void Turning()
    {
        // カメラから，マウスで指している方向の ray を取得する
        Ray camRay = Camera.main.ScreenPointToRay(Input.mousePosition);
        Debug.DrawRay(camRay.origin, camRay.direction * 100, Color.yellow);

        // ray が衝突した情報を取得する
        RaycastHit floorHit;

        // ray を飛ばして，床に衝突した場合の処理
        if (Physics.Raycast(camRay, out floorHit, camRayLength, floorMask))
        {
            // マウスで指している場所と，プレイヤーの場所の差分を取得
            Vector3 playerToMouse = floorHit.point - transform.position;

            // プレイヤは y 座標には動かさない
            playerToMouse.y = 0f;

            // プレイヤーの場所から，マウスで指している場所への角度を取得
            Quaternion newRotation = Quaternion.LookRotation(playerToMouse);

            // プレイヤーの角度(プレイヤーの向き)を、新しく設定
            playerRigidbody.MoveRotation(newRotation);
        }
    }

    void Animating(float h, float v)
    {
        // プレイヤーの移動量が 0 以外の場合，walking を true にする
        bool walking = h != 0f || v != 0f;

        // アニメーションのパラメータ IsWalking を walking の値で設定する
        anim.SetBool("IsWalking", walking);
    }
}
```

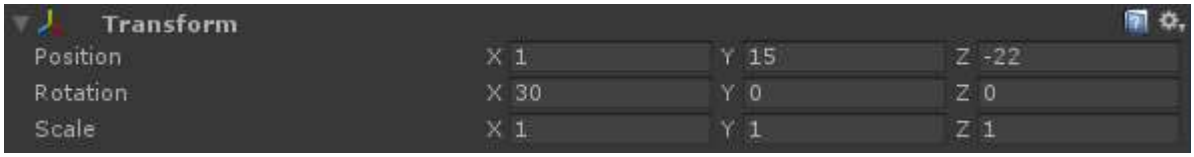
スクリプトを保存し，ゲームを実行し動作を確認する。
矢印キーで移動し，マウスを追って向きを変えれば OK。

Phase3 カメラのセットアップ

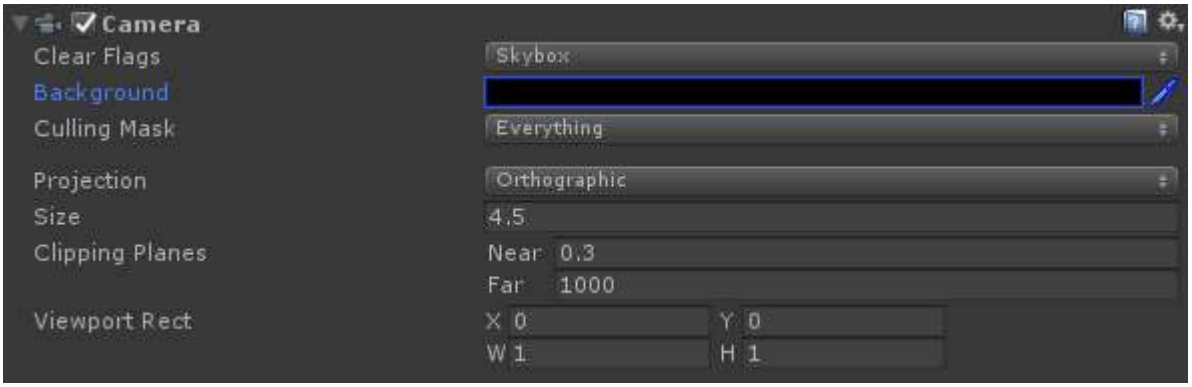
1．カメラの設定

カメラがプレイヤーキャラクタを，後ろから追いかけるようにする。

Hierarchy の Main Camera を選択し，Transform コンポーネントの Position を 1，15，-22 に，Rotation を 30，0，0 に設定する。

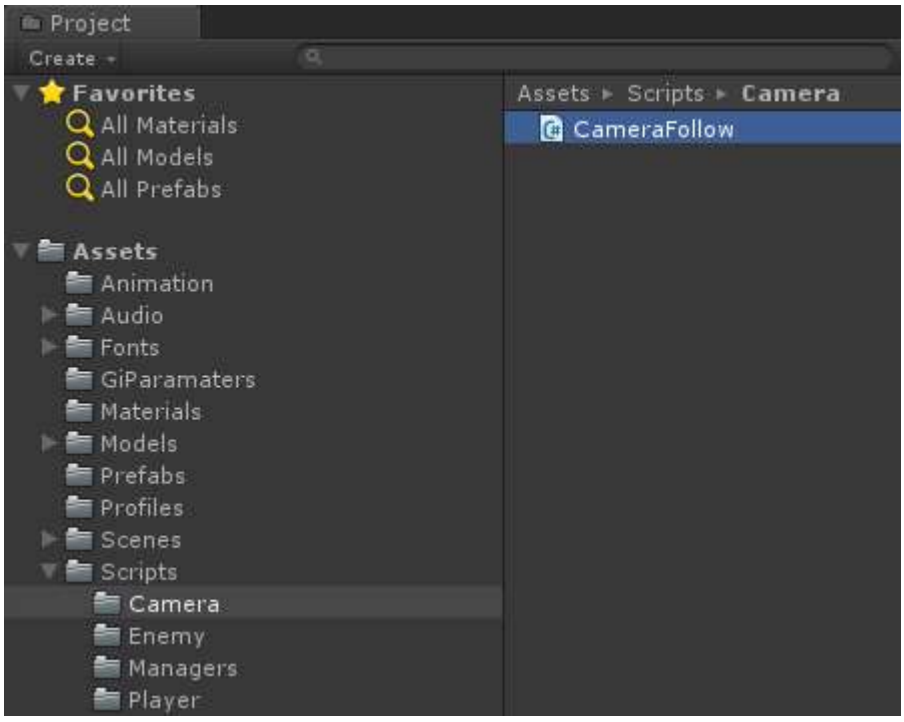


Camera コンポーネントの Background を黒，Projection を Orthographic（平行投影），size を 4.5 に設定する。



2．カメラの移動スクリプトの作成

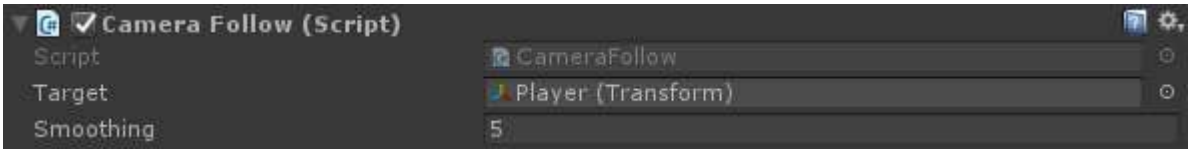
Project > Assets > Scripts フォルダ内に Camera フォルダを作成する。作成した Camera フォルダ内に CameraFollow スクリプトを作成する。作成した CameraFollow スクリプトを Main Camera にドラッグする。



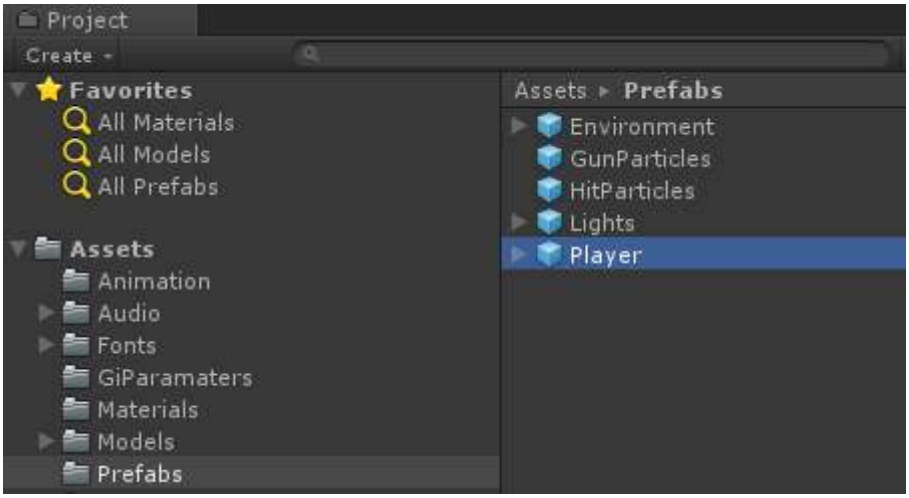
CameraFollow スクリプトに以下のコードを追加する。

```
using UnityEngine;
using System.Collections;

public class CameraFollow : MonoBehaviour
{
    public Transform target;     //カメラがついていくターゲットのポジション
    public float smoothing = 5f; //カメラのついていくスピード

    Vector3 offset; // カメラとターゲットの間の距離

    void Start()
    {
        // カメラとターゲットの間の距離を算出
        offset = transform.position - target.position;
    }

    // 物体が動く毎に呼び出される
    void FixedUpdate()
    {
        // カメラの移動先
        Vector3 targetCamPos = target.position + offset;

        // カメラを移動する
        transform.position = Vector3.Lerp(transform.position, targetCamPos,
            smoothing * Time.deltaTime);
    }
}
```

Camera Follow の Target 変数に Player オブジェクトをアサインする。



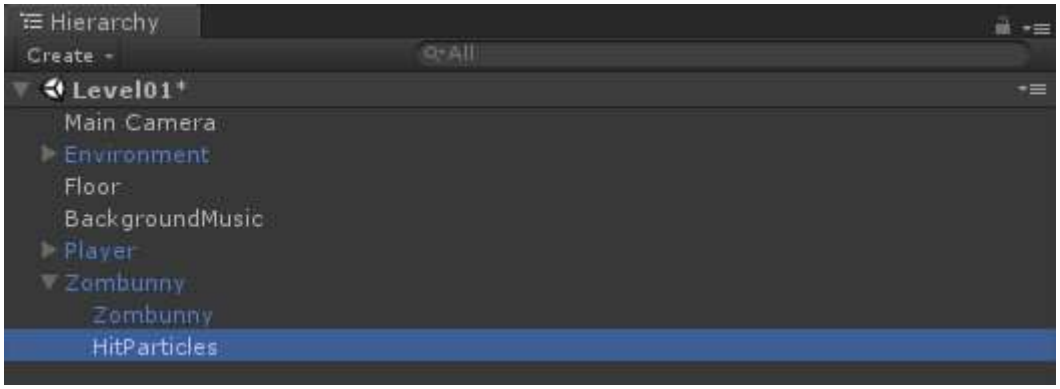
Hierarchy の Player オブジェクトを Prefabs フォルダにドラッグしプレハブ化する。



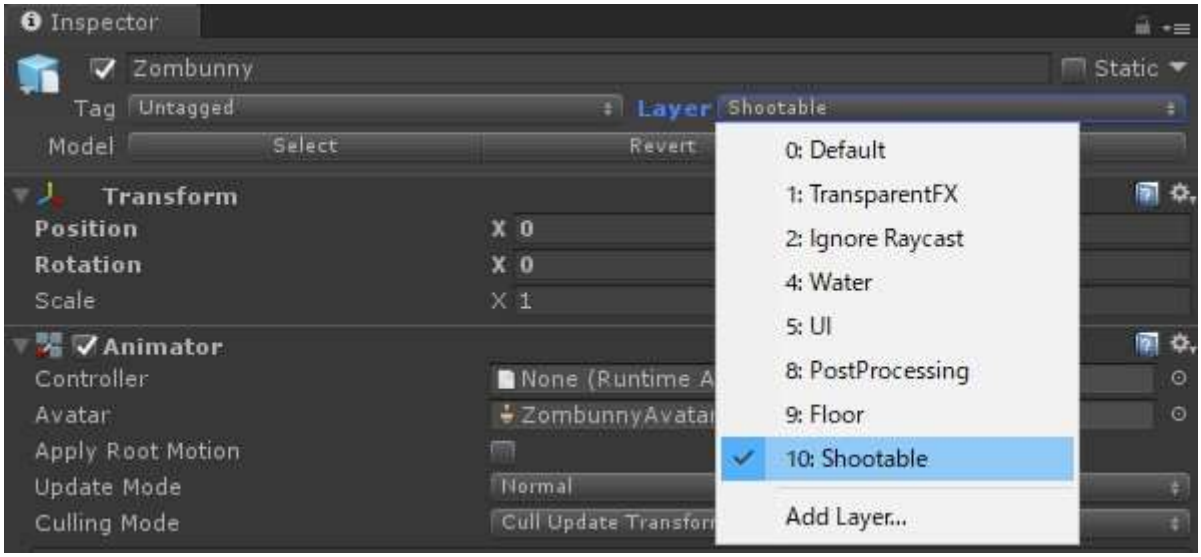
Phase4　敵の作成

1．敵の配置

Project > Assets > Models > Character フォルダ内の Zombunny を Scene または Hierarchy いずれかにドラッグする。Projects > Assts > Prefabs フォルダ内の HitParticles を Zombunny にドラッグする。



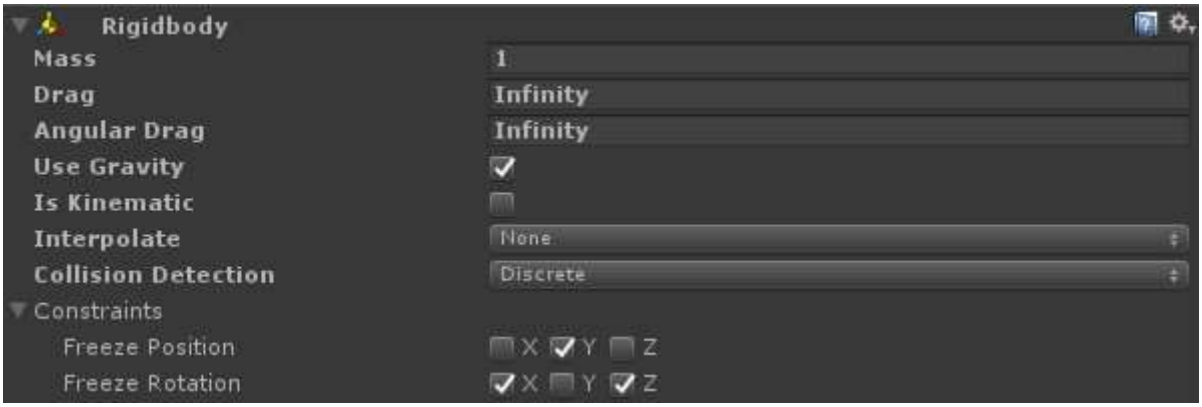
新しく Shootable レイヤーを追加し，Zombunny の Layer を Shootable に設定する。



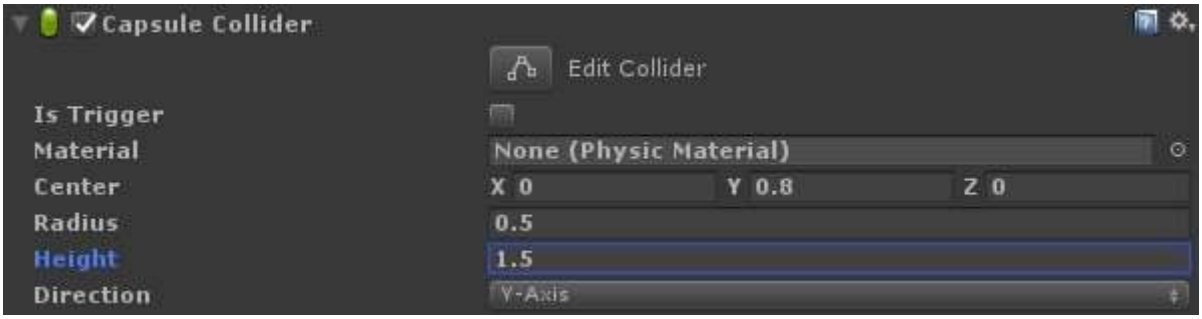
2．敵キャラクタへのコンポーネントの追加

敵キャラクタを動かすのに必要なコンポーネント，Rigidbody と Capsule Collider，Sphere Collider，Audio Source を Zombunny オブジェクトに追加する。

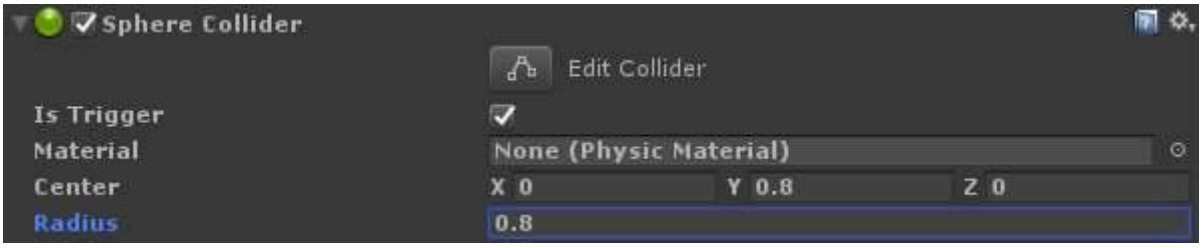
Rigidbody コンポーネントの Drag と Angular Drag に Infinity を設定する。Constraints を展開し，Freeze Position の Y，Freeze Rotation の X と Z をチェックする。



Capsule Collider コンポーネントの Center Y には 0.8 を設定する。Height は 1.5 にする。



Sphere Collider（Player との接触検知用）コンポーネントの Is Trigger をチェックし，Center Y に 0.8, Radius は 0.8 にする。

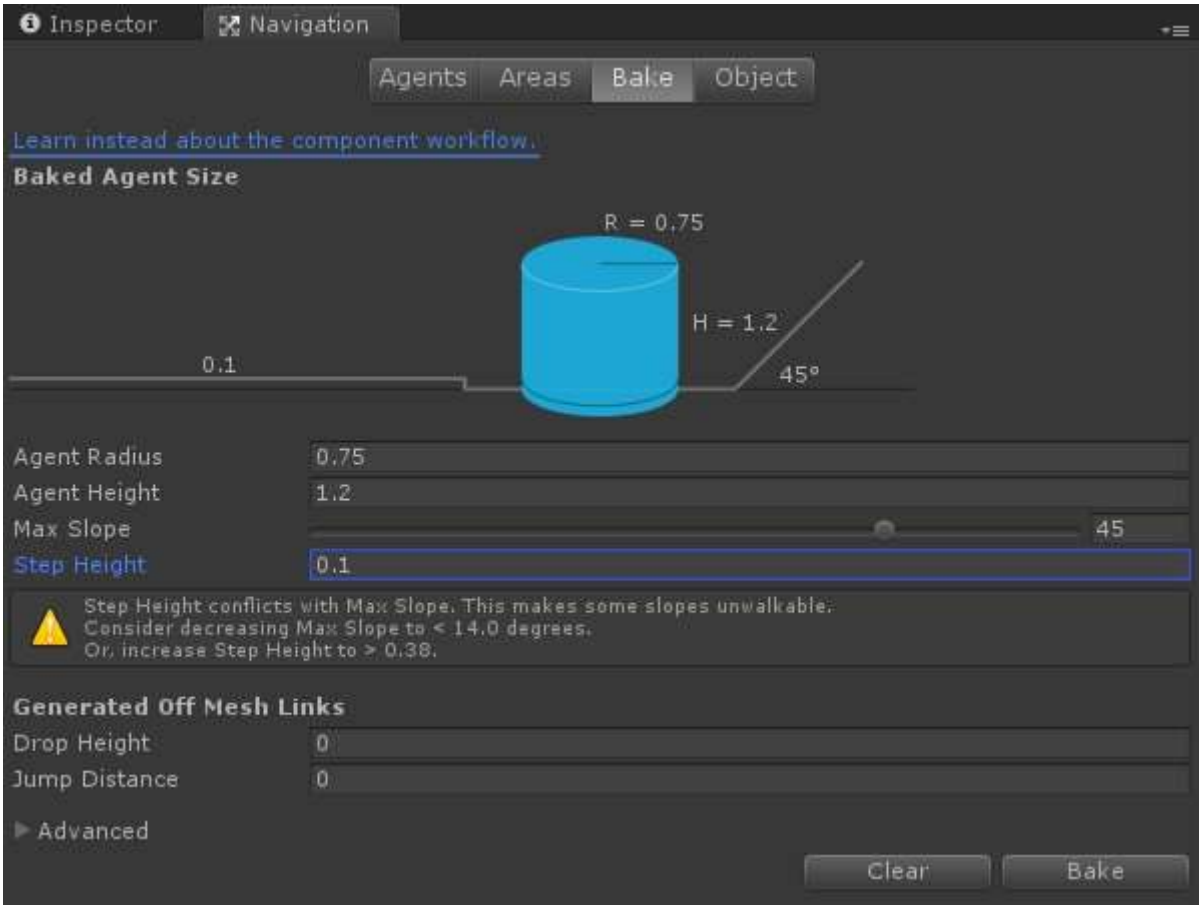


Audio Source コンポーネントには，敵キャラクタが攻撃を受けた時の効果音 Zombunny Hurt を AudioClip に設定する。Play On Awake のチェックは外す。

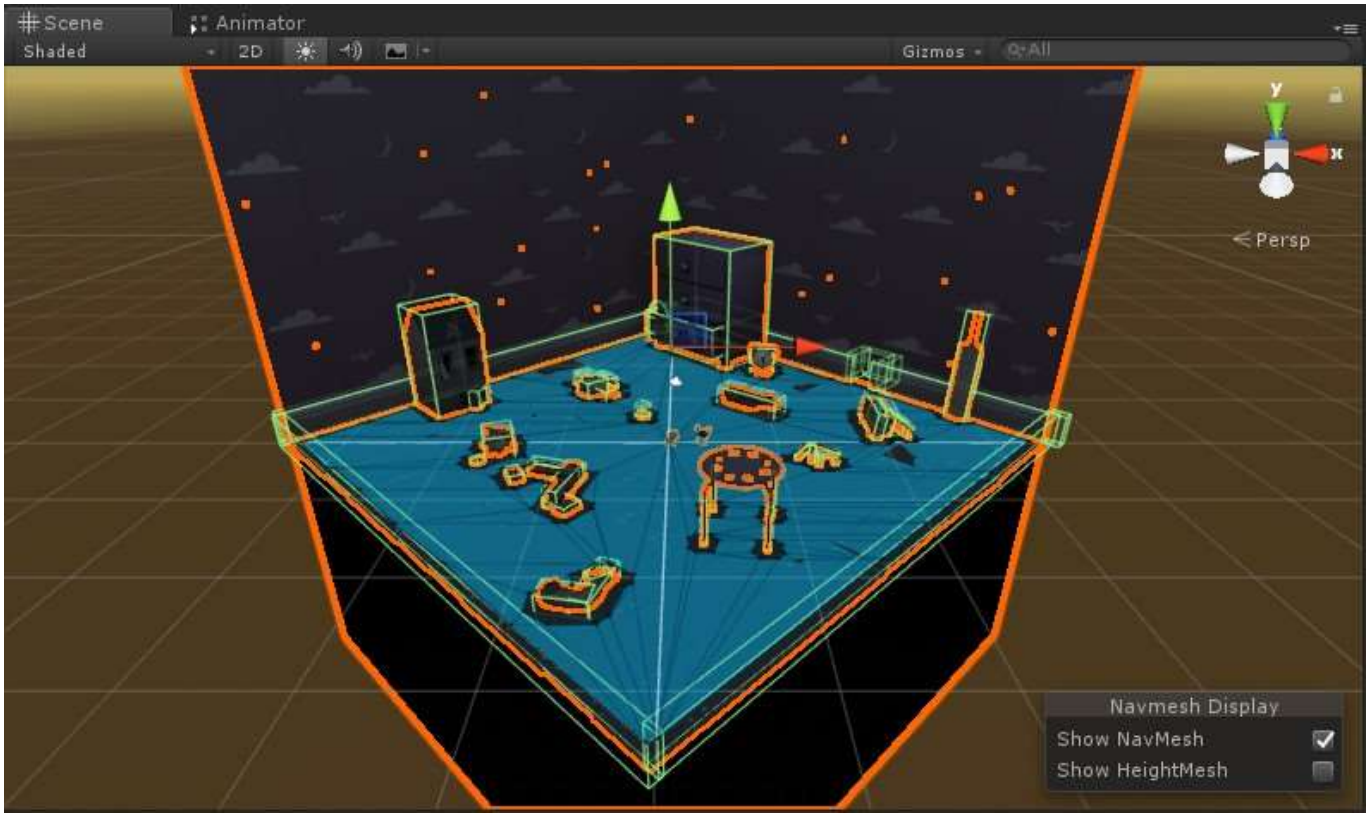
４．NaviMesh の設定

NaviMesh とは，NaviMesh Agent コンポーネントを付けたオブジェクトが移動することのできる範囲のこと。

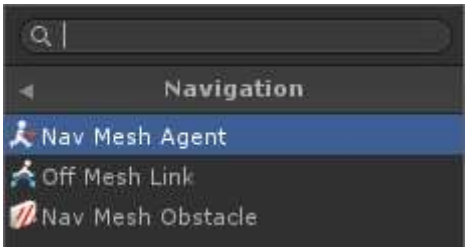
Windows > Navigation をクリックし Navigation ウインドウを開く。Bake タブをクリックし，移動できる範囲を設定する。Agent Radius に 0.75，Agent Hight に 1.2，Step Height に 0.1 を設定し，Bake ボタンをクリックする。

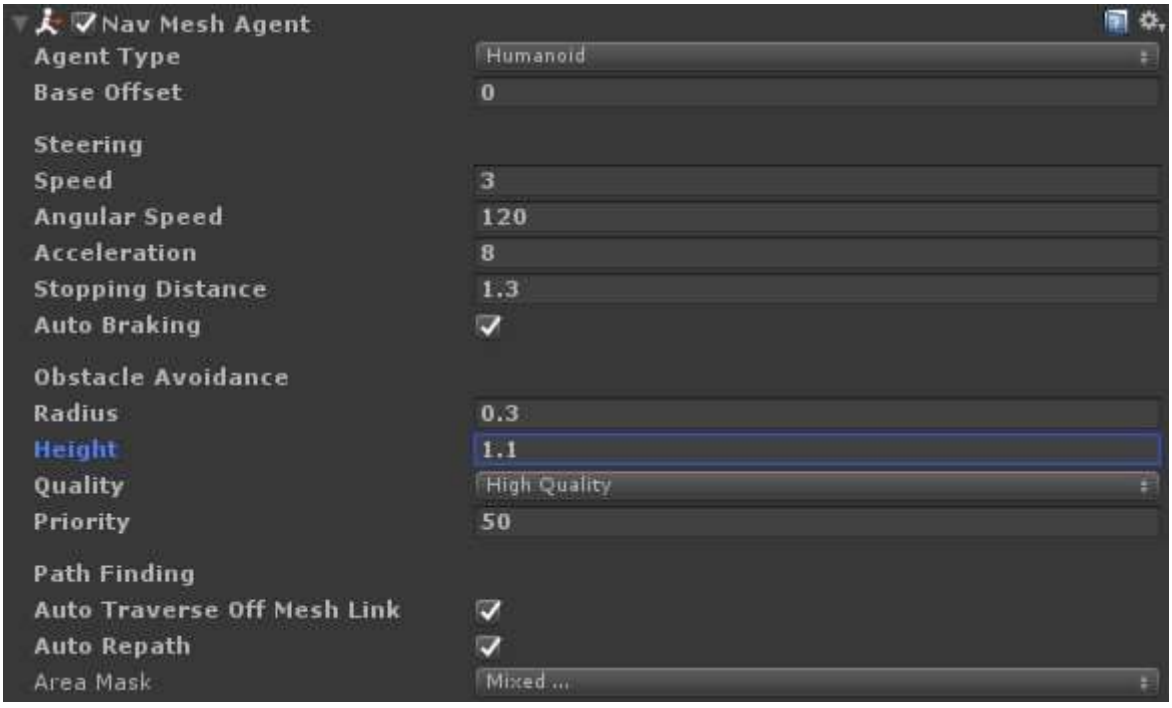


Scene ビューで移動できる範囲が青く表示される。



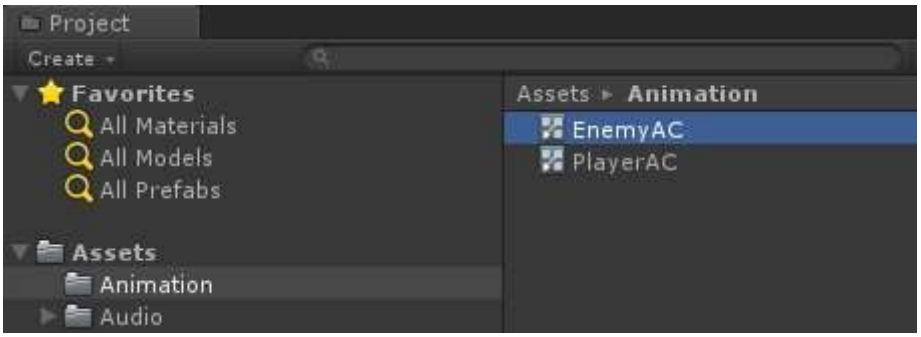
Zombunny に Add Component > Navigation > Nav Mesh Agent コンポーネントを追加し，Radius は 0.3, Speed は 3，Stopping Distance は 1.3，Height は 1.1 を設定する。



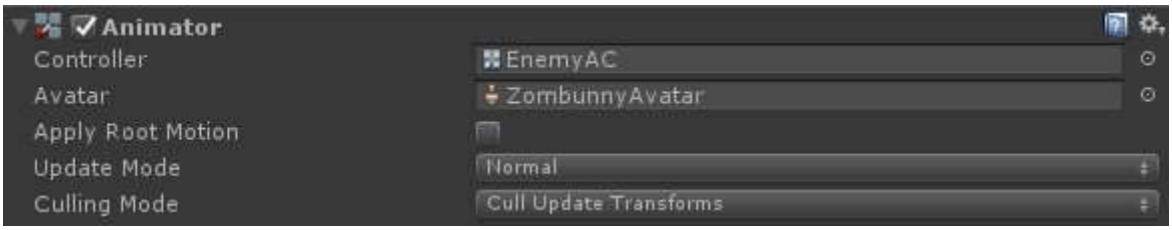


５．敵アニメーションの作成

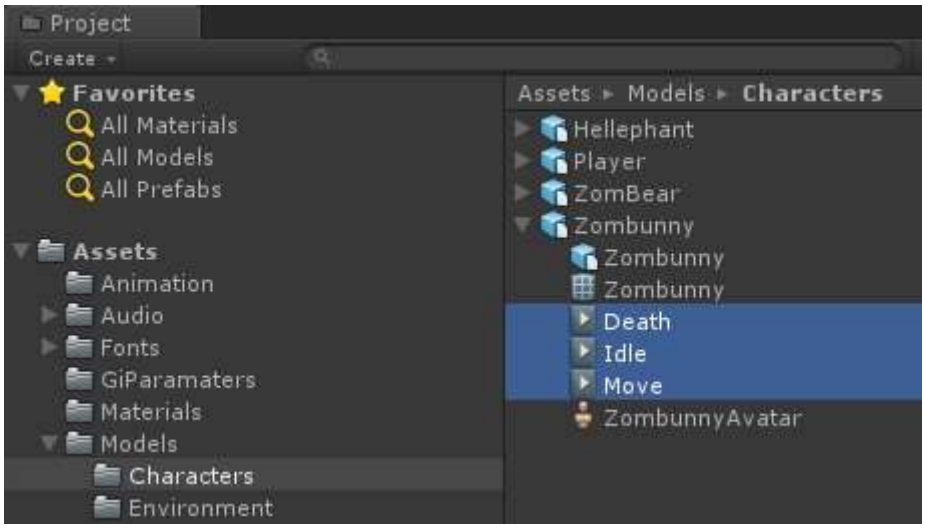
Animation フォルダ内に AnimatorController を作成する。名前は EnemyAC とする。



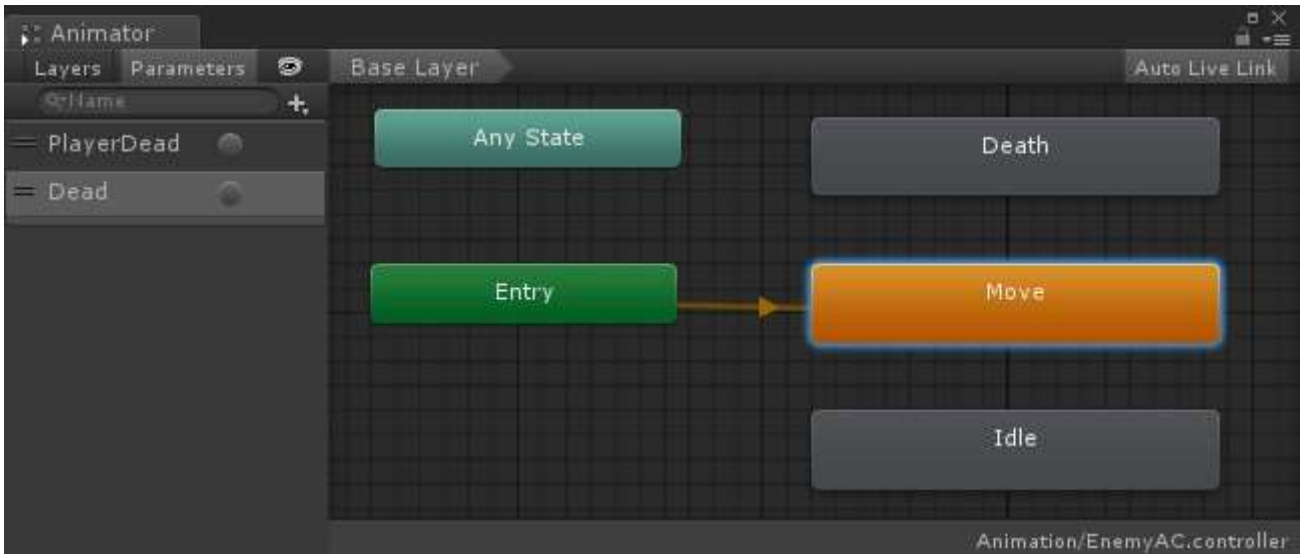
作成した EnemyAC を Hierarchy の Zombunny にドラッグする。Animator コンポーネントの Contoller に EnemyAC が設定される。



EnemyAC をダブルクリックし，Animator ウインドウを開き，Project > Asset > Models > Characters > Zombunny の Idle，Move，Death を Animator ウインドウの空白部分にドラッグする。



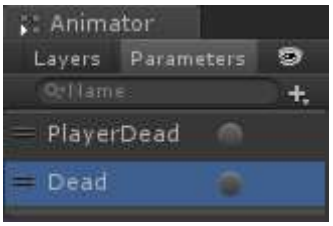
Move アニメーションがデフォルトのアニメーションになるよう設定する。



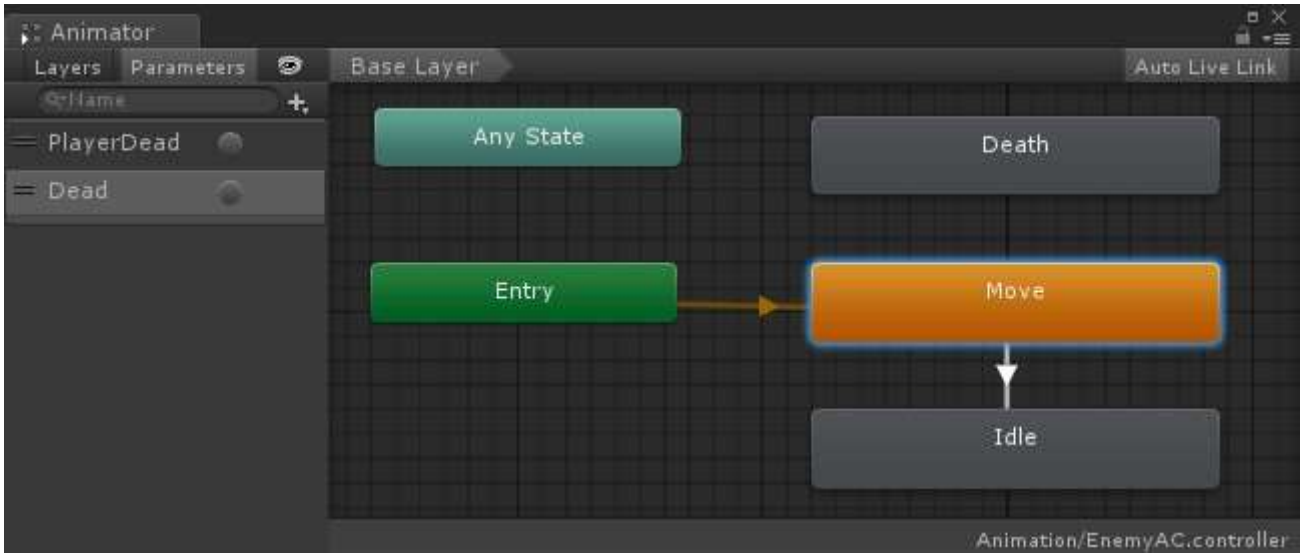
Parameters タブをクリックし，+ を押して，以下のパラメータを追加する。

Trigger 型：PlayerDead（Player が倒された場合のトリガー）

Trigger 型：Dead

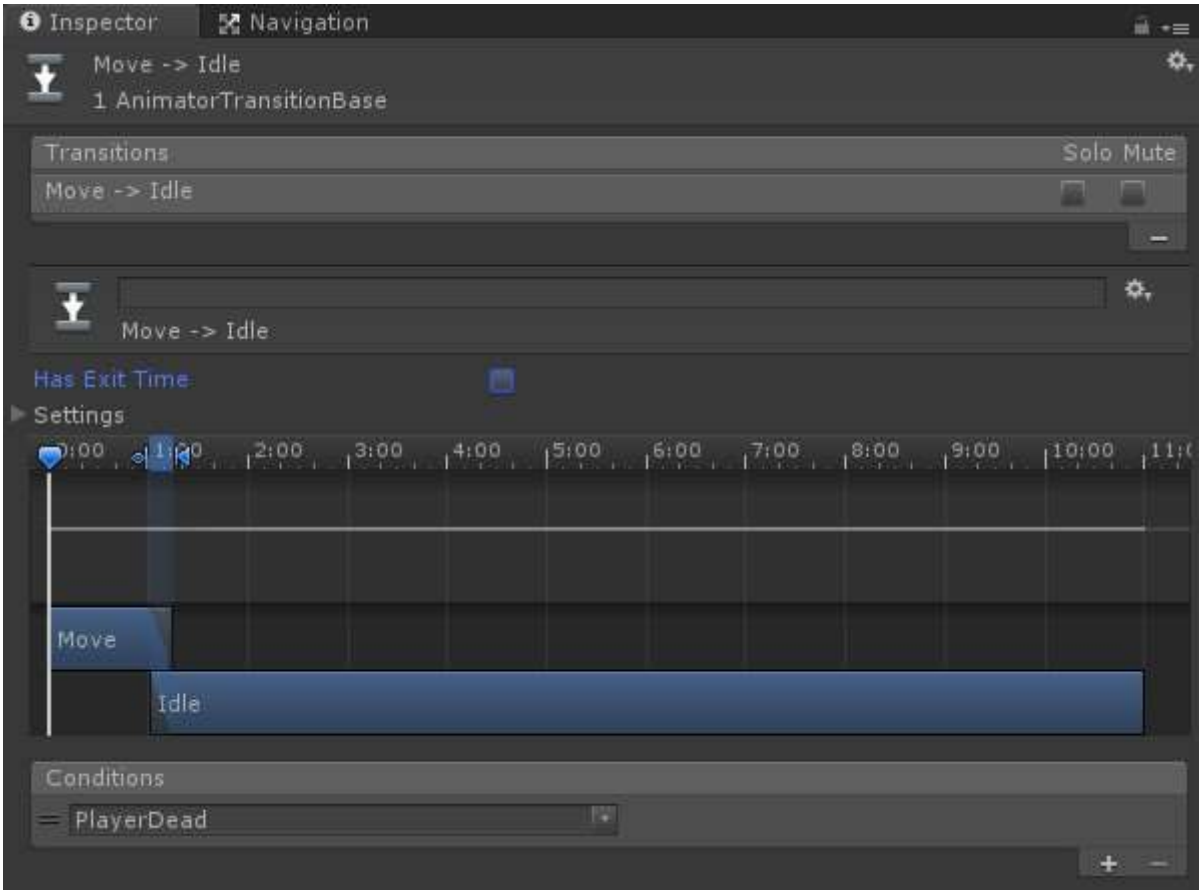


Move を右クリックし，Make Transition を選択し，Idle へ矢印を繋げる。

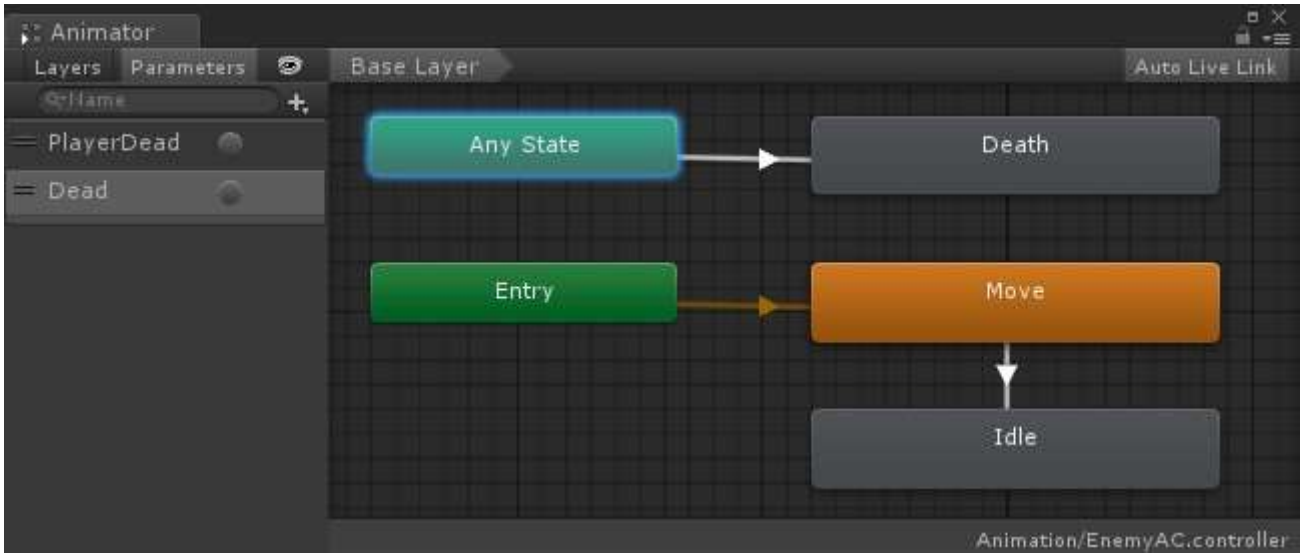


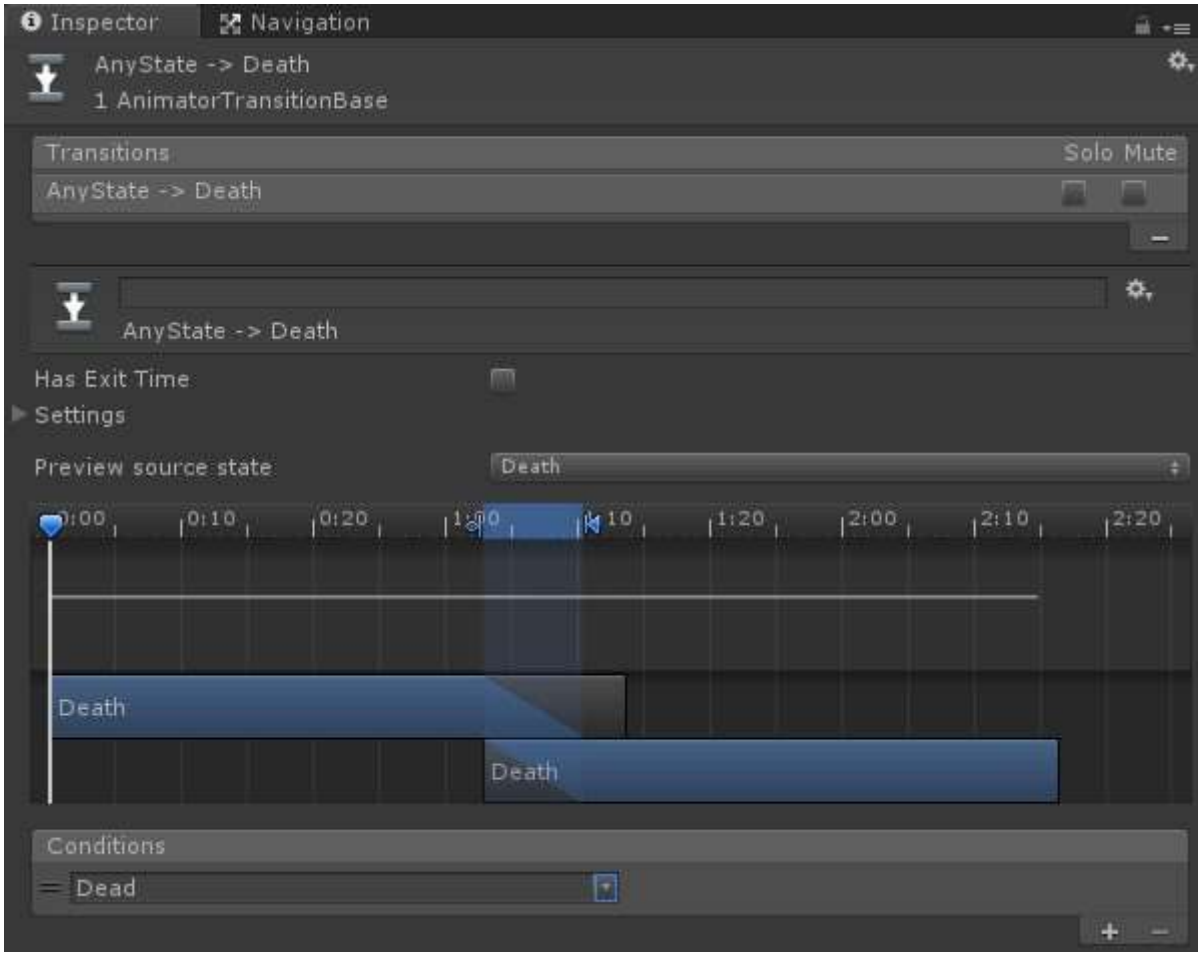
Move から Idle への矢印をクリックし，Inspector に設定項目を表示させる。

Has Exit Time のチェックを外し，Conditions の + をクリックし，アニメーションが切り替わる条件として，PlayerDead を追加する。



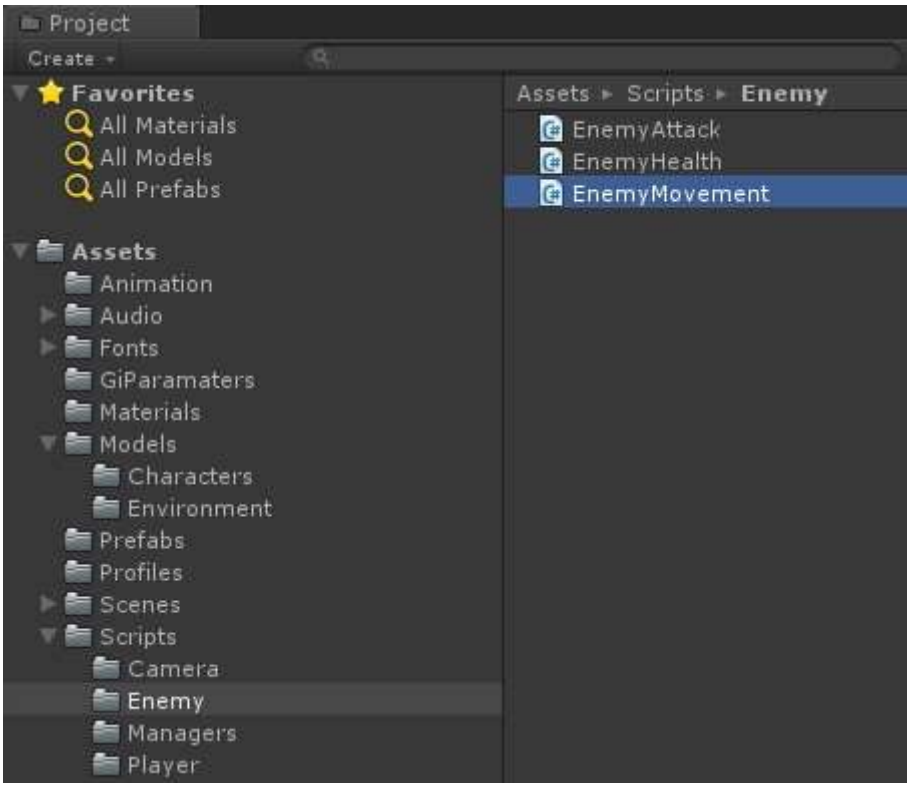
同様に，Any State から Dead に矢印をつなぎ，矢印をクリックし，Has Exit Time のチェックを外し，Conditions の Dead を追加する。




Inspector
Navigation
AnyState -> Death
1 AnimatorTransitionBase
Transitions
Solo Mute
AnyState -> Death
AnyState -> Death
Has Exit Time
Settings
Preview source state
Death
0:00
0:10
0:20
1:10
1:20
2:00
2:10
2:20
Death
Death
Conditions
Dead

6．Player を追いかけるスクリプトの作成

Project > Assets > Scripts > Enemy > EnemyMovement を Zombunny にアタッチする。



EnemyMovement スクリプトは以下の内容。一部のコードはコメントアウトされており，今後のチュートリアルで使用する。

```
using UnityEngine;
using System.Collections;

public class EnemyMovement : MonoBehaviour
{
    Transform player;
    //PlayerHealth playerHealth;
    //EnemyHealth enemyHealth;
    UnityEngine.AI.NavMeshAgent nav;     // NavMeshAgent コンポーネットへアクセスするための変数


    void Awake ()
    {
        // Player タグの付いたオブジェクトを探索し，参照を取得する
        player = GameObject.FindGameObjectWithTag ("Player").transform;
        //playerHealth = player.GetComponent<PlayerHealth>();
        //enemyHealth = GetComponent<EnemyHealth>();

        // NavMeshAgent コンポーネントへの参照の取得
        nav = GetComponent<UnityEngine.AI.NavMeshAgent>();
    }
```

```
    void Update ()
    {
        //if(enemyHealth.currentHealth > 0 && playerHealth.currentHealth > 0)
        //{
            // Player の position を設定し，Player に向けて動かす
            nav.SetDestination(player.position);
        //}
        //else
        //{
        //    nav.enabled = false;
        //}
    }
}
```

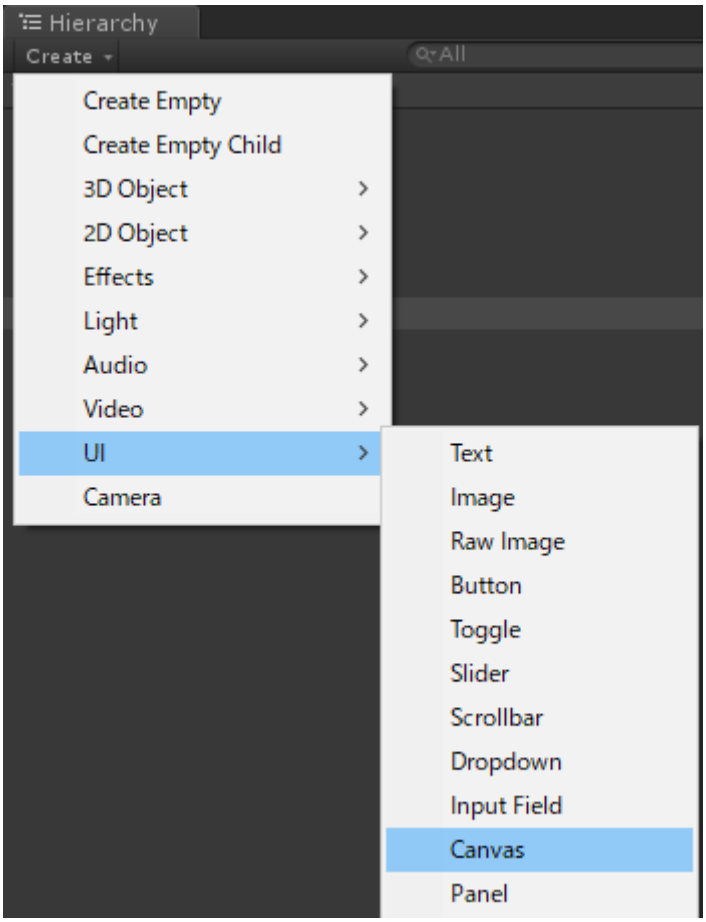
実行して，Player が移動すると Zombunny が Player を追いかけるか確認する。

Phase5　UI（ユーザーインタフェース）の作成

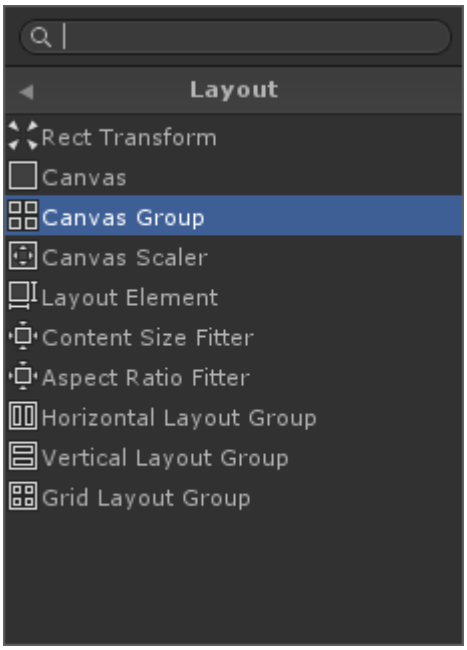
1．ハート画像（イメージ）の表示

ゲーム画面の左下に表示される，プレイヤーの HP を表すハートの画像を表示させます。

Hierarchy > Create > UI > Canvas をクリックし，Canvas を追加する。Canvas は HUDCanvas にリネームする。

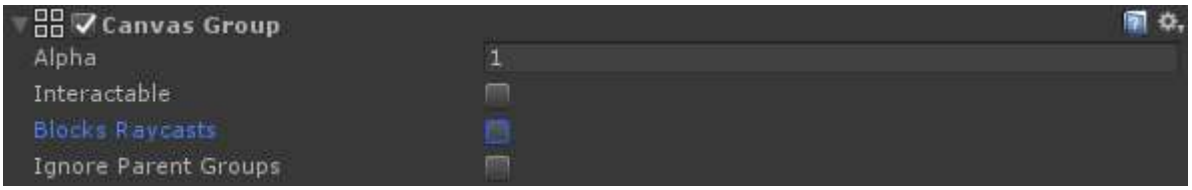


HUDCanvas に Canvas Group コンポーネントを追加（Add Component > Layout > Canvas Group）する。

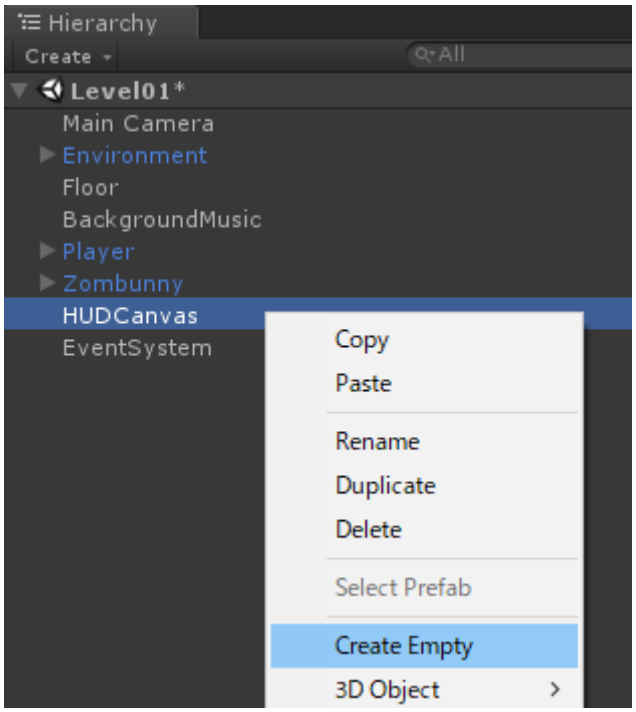


Canvas Group コンポーネントの，Interactable と Block Raycasts のチェックを外す。

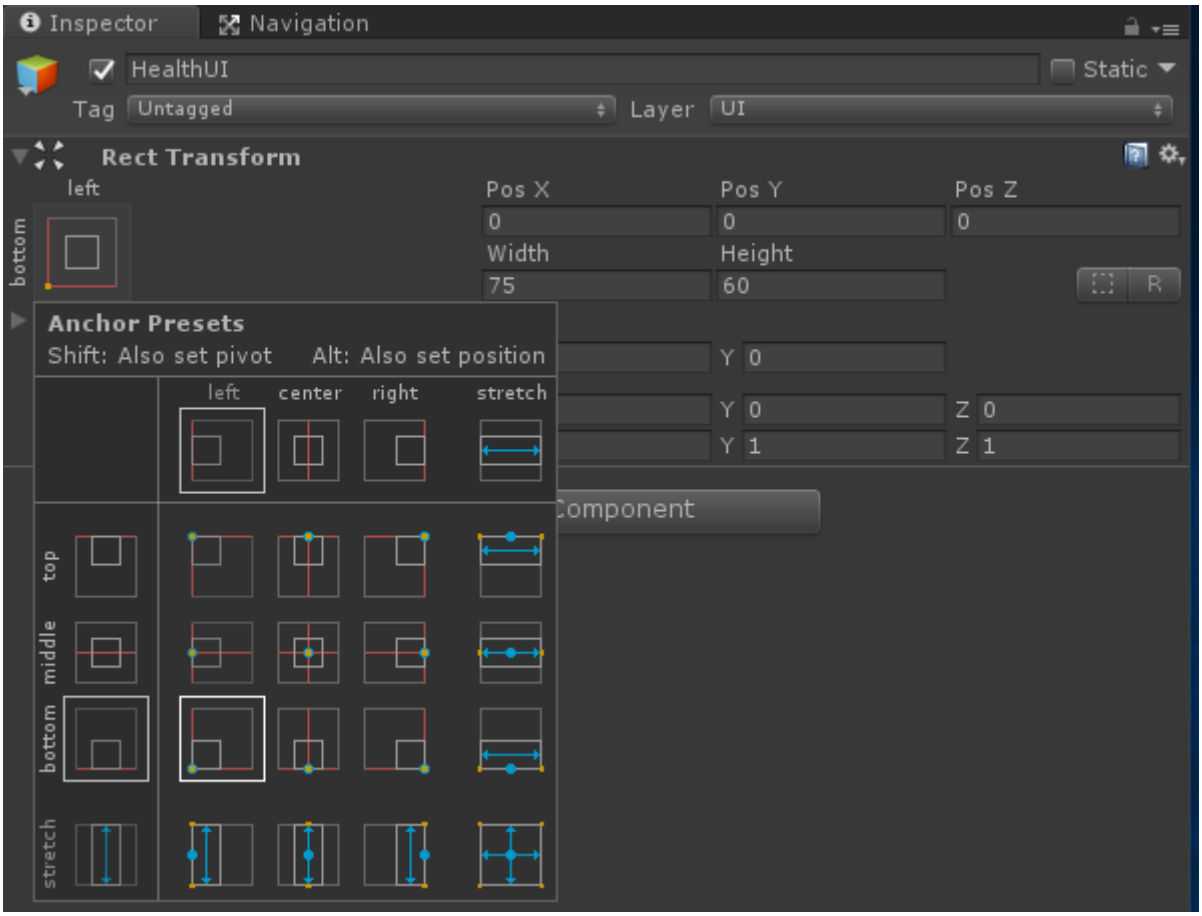
Canvas Group コンポーネントは，このコンポーネントがアタッチされているオブジェクトと，その子オブジェクトをグループ化し，共通の機能を与えることができる。Interactable は入力を受け付けるかどうかの設定で，チェックが外れている場合，入力などは無効になる。Block Raycasts は Ray が当たるかどうかの設定で，チェックが外れている場合，Ray にはぶつからない。



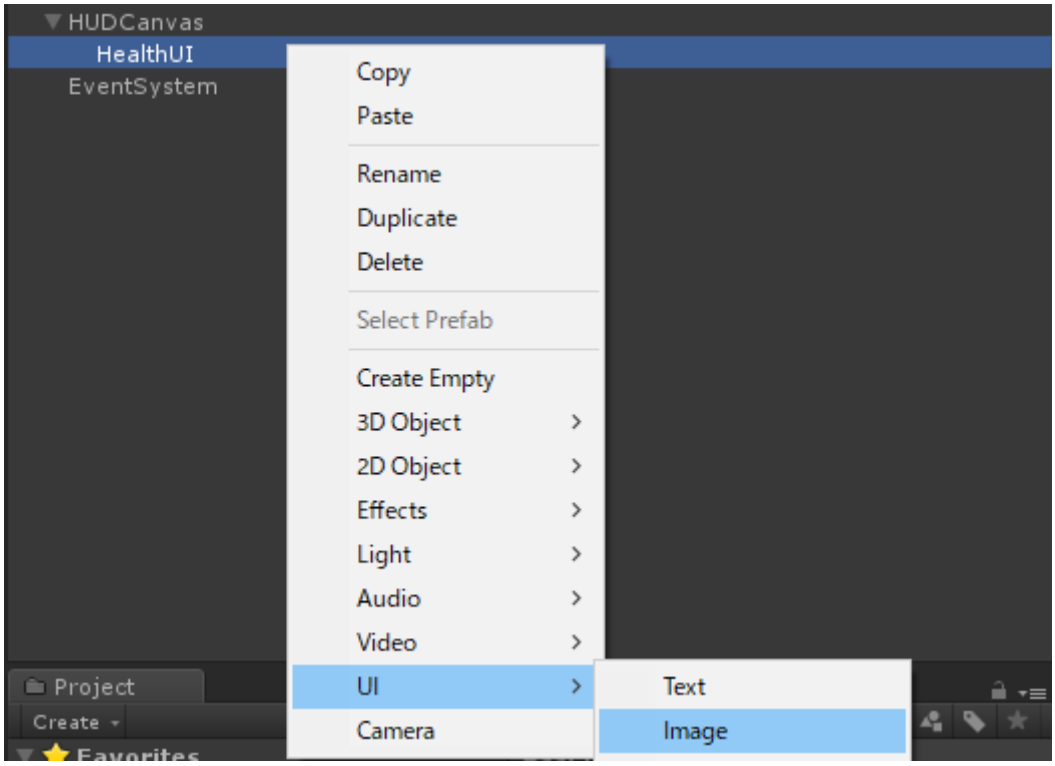
HUDCanvas を右クリックし，空の子オブジェクトを作成し，名前を HealthUI にリネームする。



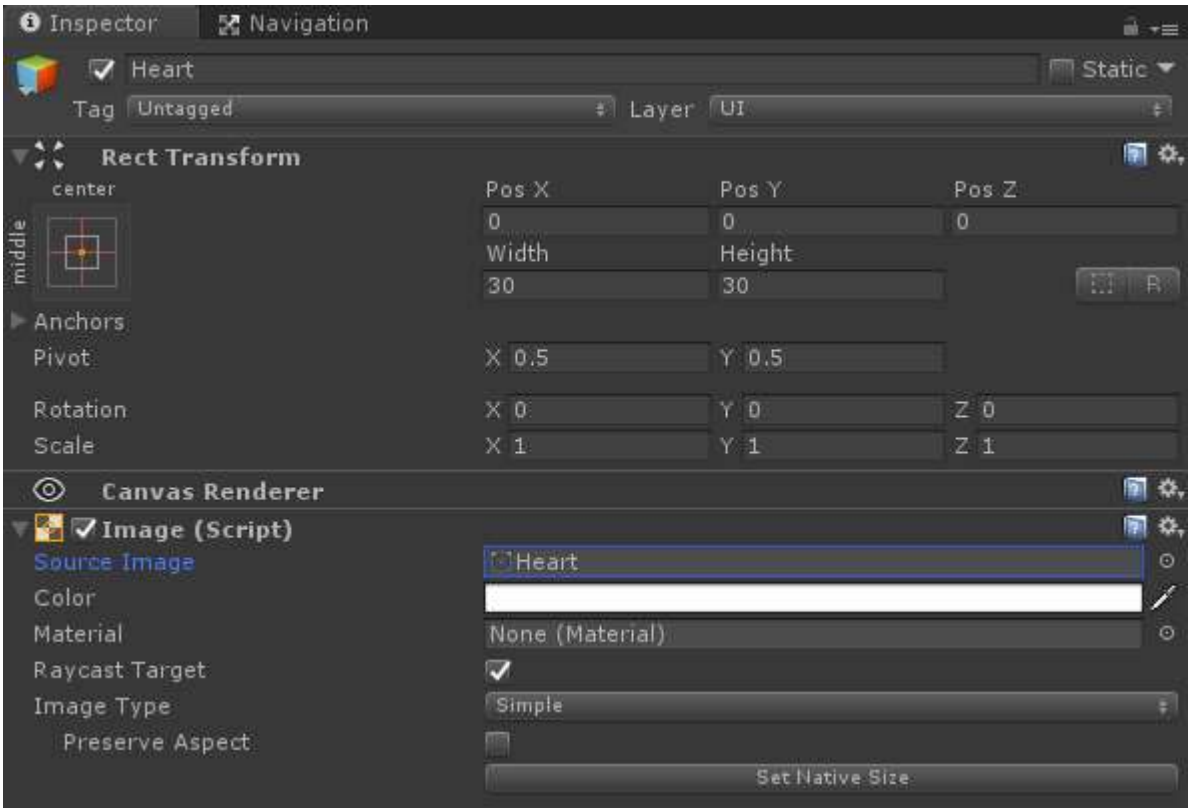
HealthUI をクリックし，Inspector の Rect　Transform の Anchor Presets をクリックし，Shift + Alt を押したまま，左下の Bottom，left の部分をクリックする。Width と Height にそれぞれ 75 と 60 を設定する。

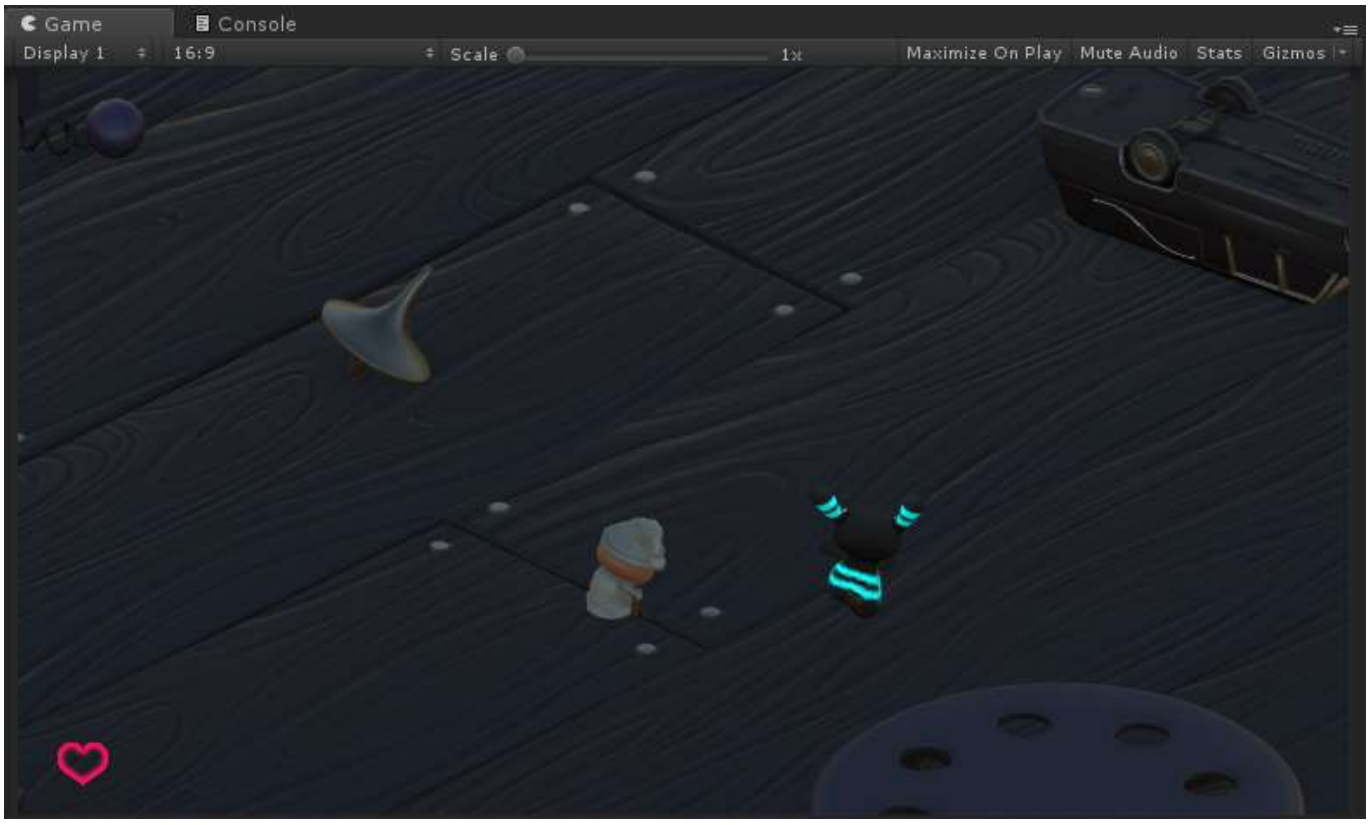


HealthUI を右クリックし，HeaithUI の子オジェクトとして Image オブジェクト（UI > Image）を作成する。名前は Heart にリネームする。



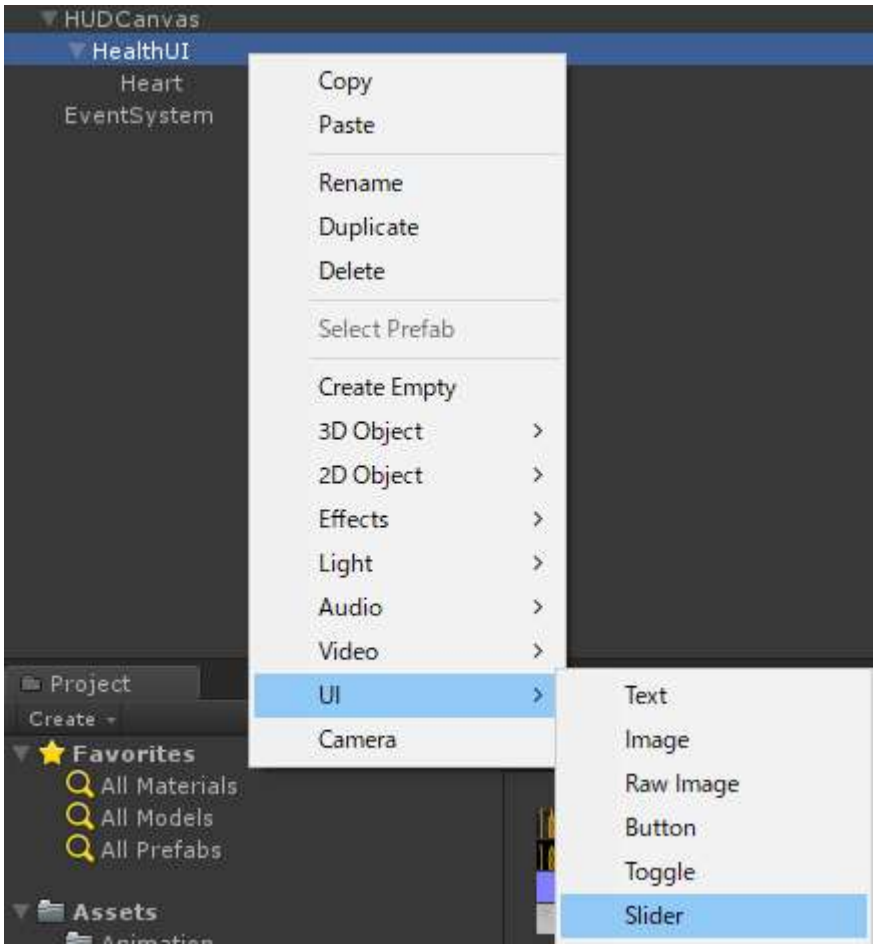
Heart をクリックし，Inspector の Rect Transform の Position，X と Y を 0 に設定する。Width と Height には 30 を設定する。Image コンポーネントの Source Image にハートの画像（Project > Textures > Heart）を設定する。

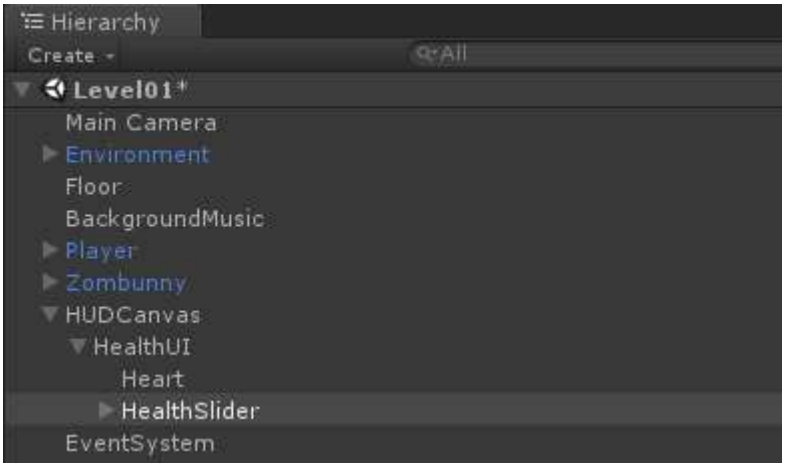




２．HP ゲージ（Slider）の表示

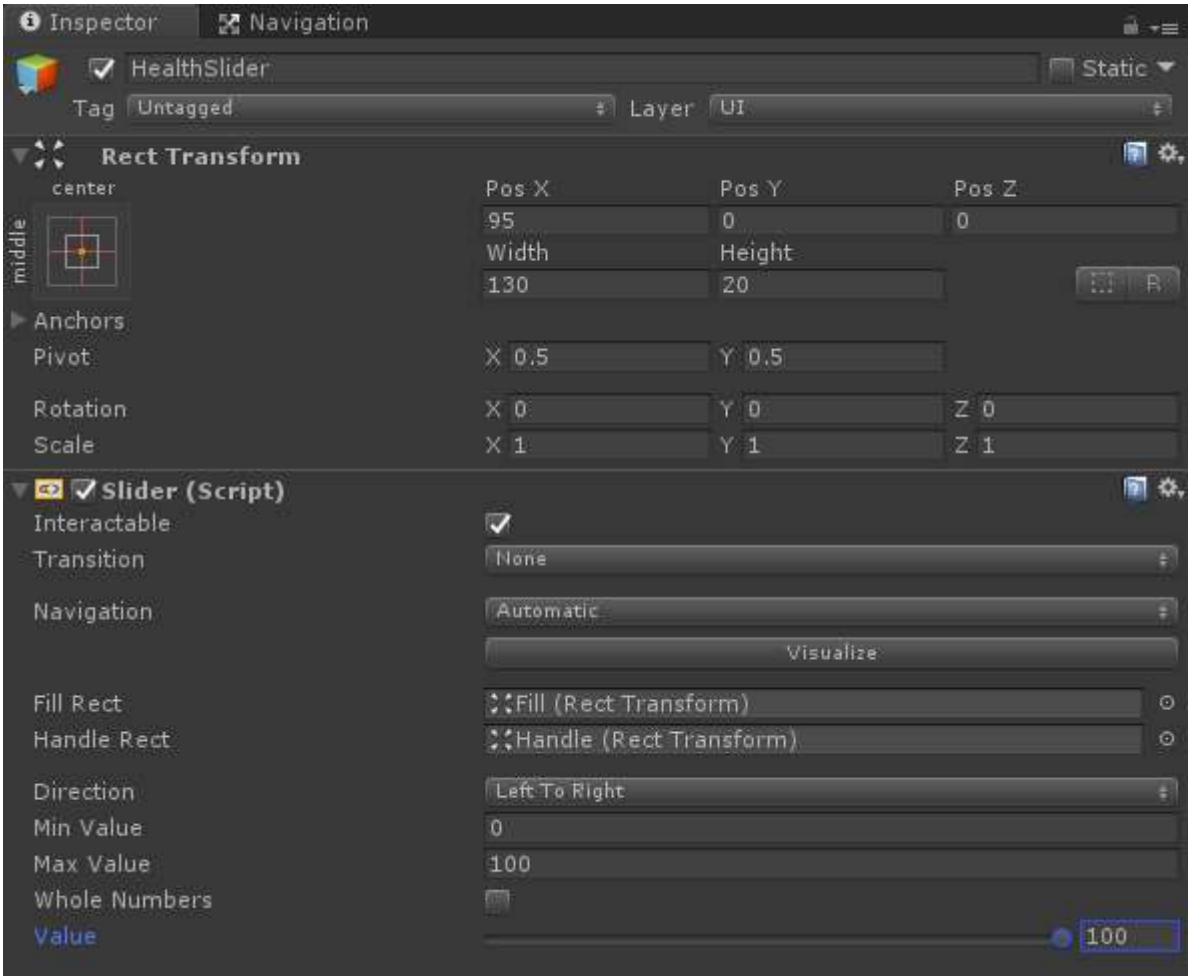
HealthUI を右クリックし，子オブジェクトとして HP ゲージとなる Slider（UI > Slider）を作成する。名前を HealthSlider にリネームする。



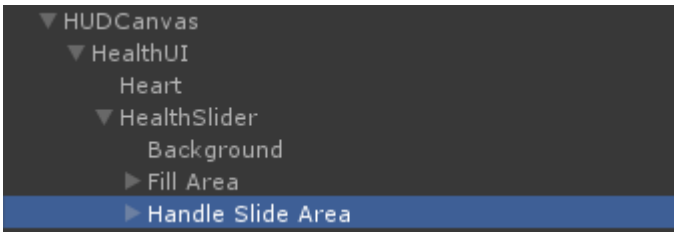


HealthSlider をクリックし，Inspector の Rect Transform の Position X を 95 に，Width と Height をそれぞれ 130 と 20 に設定する。

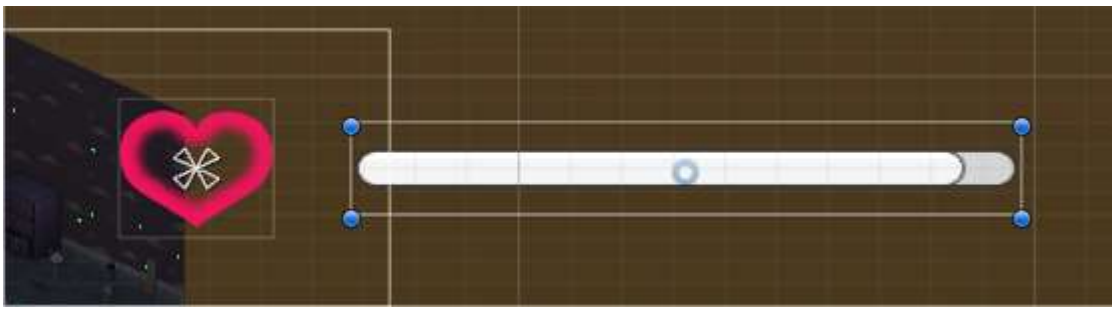
Slider(Script)の Transition を None，Max Value と Value には 100 を設定する。



HealthSlider を展開し，子オブジェクトの Handle Slide Area を削除する。

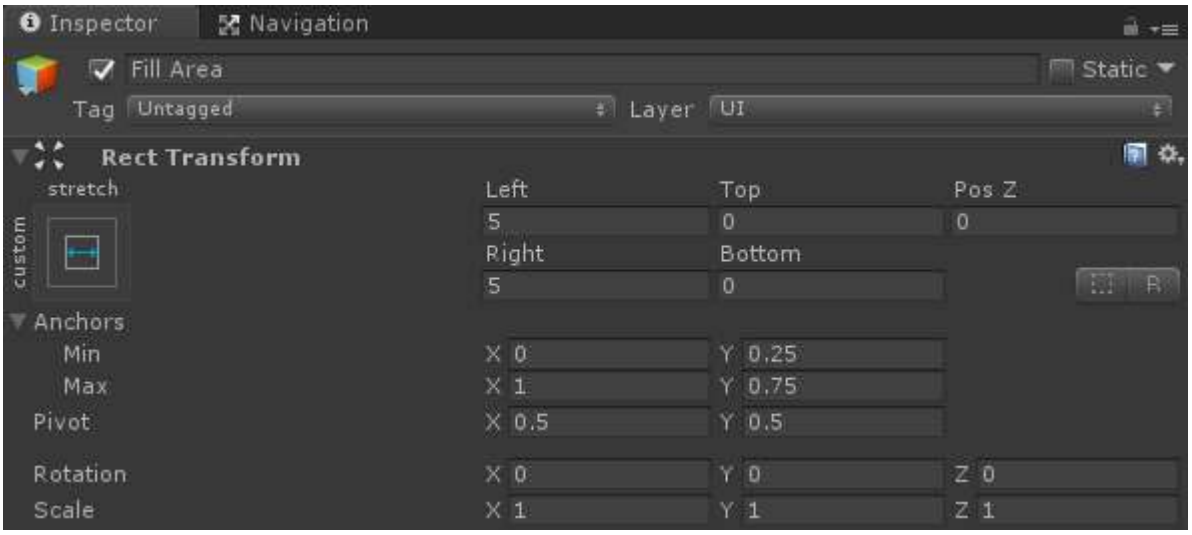


削除すると以下のとおり，ハンドル部分がなくなり，バーが右端まで届かなくなる。



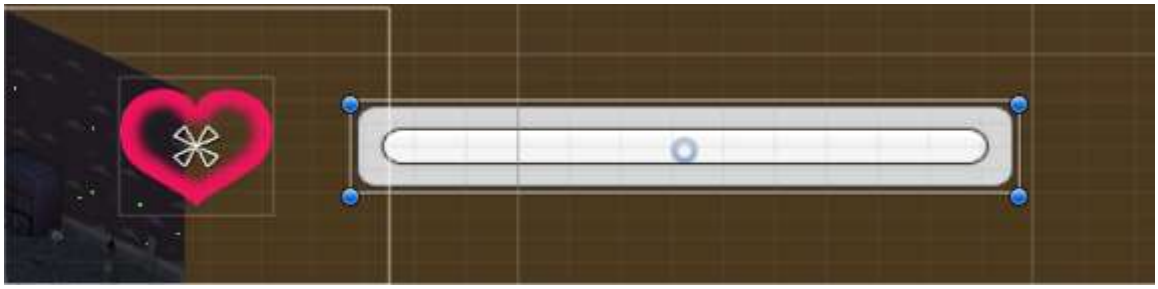
Background オブジェクトをクリックし，Inspector の Rect Transform で，Top と Bottom の値を 0 に設定する。Anchors を展開し，Min の X と Y には 0，Max の X と Y には 1 を設定する。

Fill Area オブジェクトをクリックし, Inspector の Rect Transform の Left と Right を 5 に, Top と Bottom を 0 に設定する。



Fill Area オブジェクトを展開し, Fill オブジェクトをクリックする。Inspector の Rect Transform の Left, Right, Top, Bottom すべてを 0 に設定する。

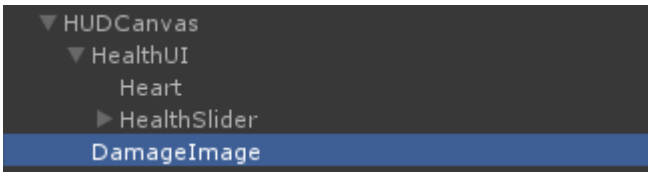
ここまでの設定で，画面左下に HP ゲージが表示される。



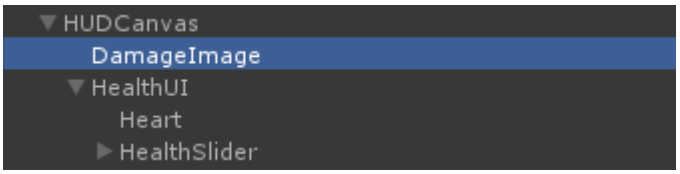
３．ダメージを受けたときに画面をフラッシュさせる

プレイヤーキャラクタがダメージを受けたときに，画面全体をフラッシュさせる UI を作成する。

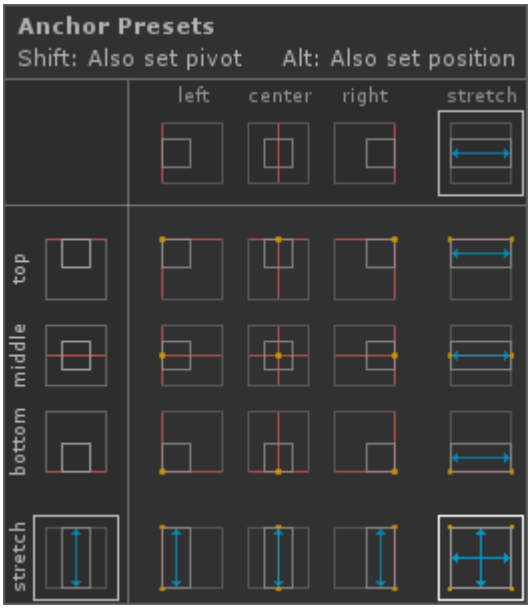
HUDCanvas を右クリック，UI > Image をクリックし Image オブジェクトを追加する。名前は DamageImage にリネームする。



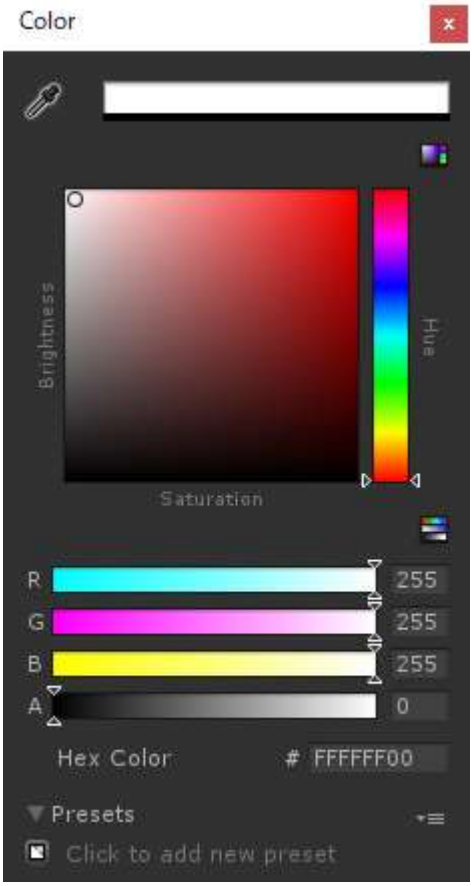
uGUI では Canvas 内のオブジェクトは，Hierarchy の上から順に表示される。DamageImage は HUDCanvas 直下に移動させ，最初に描画するように変更する。



Inspector の Anchor Presets をクリックする。Alt キーを押しながら，右下の縦横方向にストレッチをクリックし，HUDCanvas のサイズに自動的ストレッチされるように設定する。



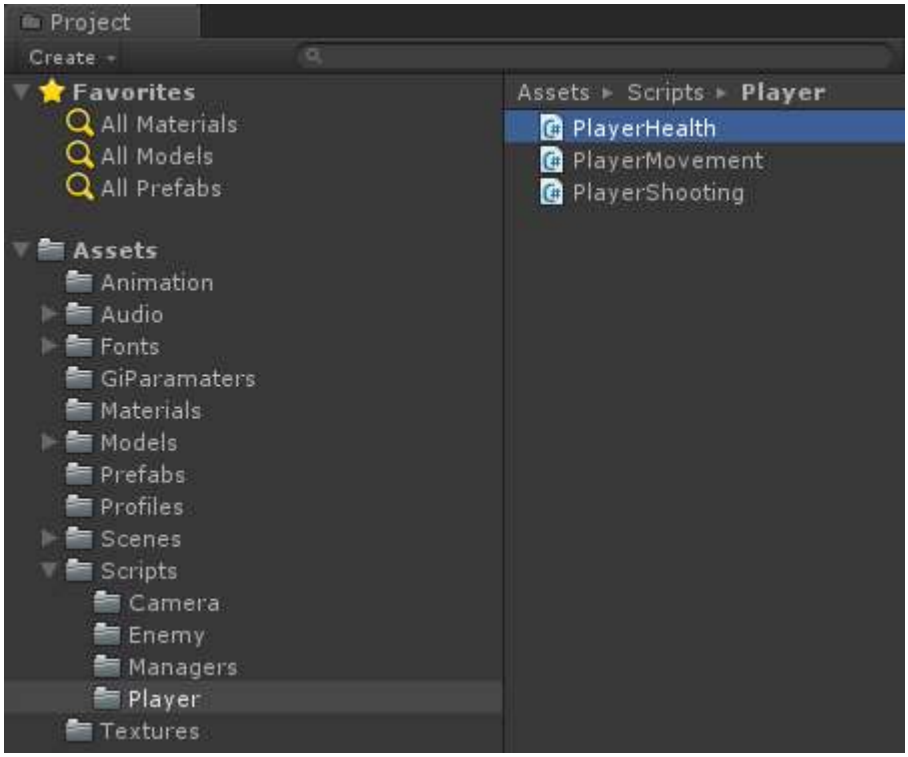
Image(Script)コンポーネントの Color をクリックし，A（アルファ値）を 0（透明）に設定する。ダメージを受けたときに，色とアルファ値を変更し画面全体をフラッシュさせる効果を与えるようにする。


Color
Brightness
Hue
Saturation
R
255
G
255
B
255
A
0
Hex Color
FFFFFF00
Presets
Click to add new preset

Phase6　プレイヤーキャラクタの体力

1．プレイヤキャラクタの体力管理スクリプトの設定

Project > Assets > Scripts > Player > PlayerHealth を Hierarchy の Player にドラッグする。



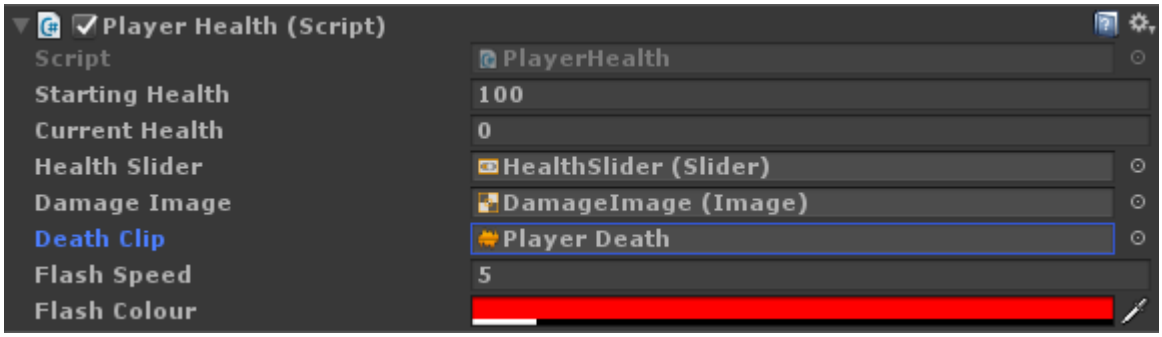
PlayerHealth をダブルクリックし，コードの内容を確認する。

```
using System.Collections;
using UnityEngine.SceneManagement;

public class PlayerHealth : MonoBehaviour
{
    public int startingHealth = 100; // スタート時の HP
    public int currentHealth;        // 現在の HP
    public Slider healthSlider;      // HP ゲージ
    public Image damageImage;        // 攻撃を受けたときの画像
    public AudioClip deathClip;      // 倒されたときの効果音
    public float flashSpeed = 5f;    // 攻撃されたときの画像の表示速度
    public Color flashColour = new Color(1f, 0f, 0f, 0.1f); // 攻撃されたときの
画像の色

    Animator anim;                   // アニメーション
    AudioSource playerAudio;         // 音楽
    PlayerMovement playerMovement;   // PlayerMovement スクリプト
    //PlayerShooting playerShooting;
    bool isDead;     // プレイヤーが倒されているかのフラグ
    bool damaged;    // プレイヤーがダメージを受けているかのフラグ

    void Awake ()
    {
        // コンポーネントを設定
```

```
        anim = GetComponent <Animator> ();
        playerAudio = GetComponent <AudioSource> ();
        playerMovement = GetComponent <PlayerMovement> ();
        //playerShooting = GetComponentInChildren <PlayerShooting> ();

        //体力を初期化
        currentHealth = startingHealth;
    }

    void Update ()
    {
        // プレイヤーがダメージを受けたか判定
        if(damaged)
        {
            // 攻撃された時の画像の色を設定する
            damageImage.color = flashColour;
        }
        else
        {
            // 攻撃された時の画像の色をクリアする
            damageImage.color = Color.Lerp (damageImage.color, Color.clear,
                flashSpeed * Time.deltaTime);
        }
        // ダメージを受けているかのフラグをオフにする
        damaged = false;
    }

    // プレイヤーが攻撃されたときの処理
    public void TakeDamage (int amount)
    {
        // ダメージを受けているかのフラグをオンにする
        damaged = true;

        // プレイヤーの HP を減らす
        currentHealth -= amount;

        // HP ゲージを減らす
        healthSlider.value = currentHealth;

        // 攻撃を受けたときの効果音を鳴らす
        playerAudio.Play ();

        // HP が 0 以下に，かつ既に倒されていない場合に倒される
        if(currentHealth <= 0 && !isDead)
        {
            Death ();
        }
    }

    // プレイヤーが倒された場合の処理
    void Death ()
    {
        // プレイヤーが倒されているかのフラグをオンにする
        isDead = true;
```

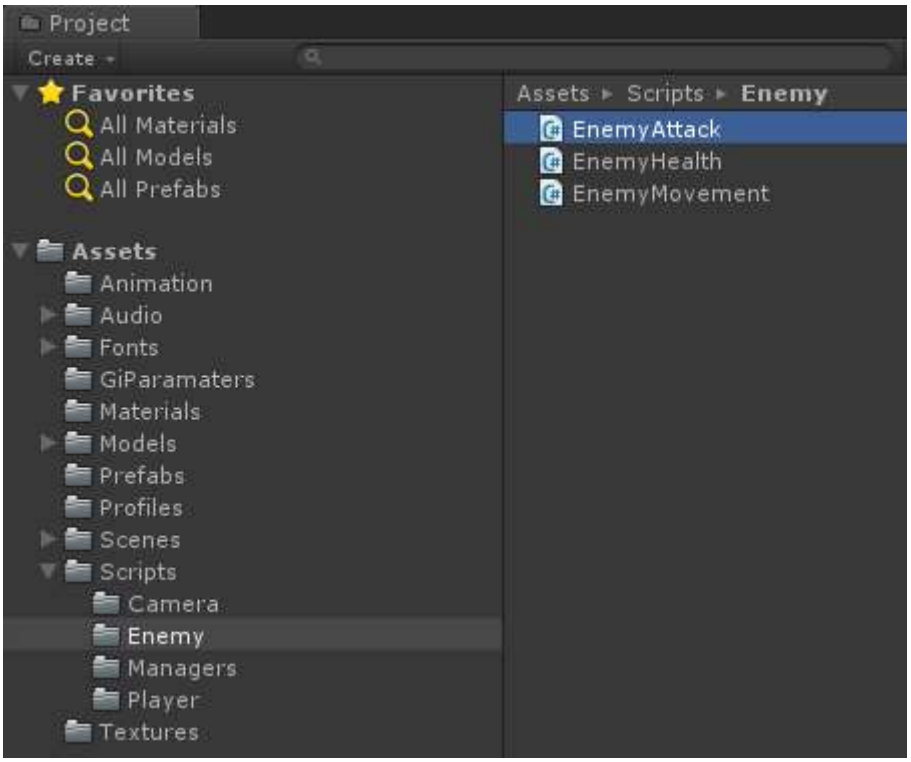
```
        // playerShooting.DisableEffects ();

        // 倒された時のアニメーションのフラグをオンにする
        anim.SetTrigger ("Die");

        // 倒された時の効果音を設定する
        playerAudio.clip = deathClip;
        // 倒された時の効果音を鳴らす
        playerAudio.Play ();

        // プレイヤーを動けなくする
        playerMovement.enabled = false;
        //playerShooting.enabled = false;
    }

    public void RestartLevel ()
    {
        SceneManager.LoadScene (0);
    }
}
```

Player にアタッチした PlayerHealth コンポーネントの Health Slider に HealthSlider を，Damge Image には DamageImage を，Death Clip には PlayerDeath を設定する。Flash Colour がダメージを受けたときにフラッシュする DamageImage の色になる。



２．敵キャラクタの攻撃スクリプトの設定

Project > Assets > Scripts > Enemy > EnemyAttack を Hierarchy の Zombunny にドラッグする。



EnemyAttack をダブルクリックし，スクリプトの内容を確認する。

```
using UnityEngine;
using System.Collections;

public class EnemyAttack : MonoBehaviour
{
    public float timeBetweenAttacks = 0.5f;    // 攻撃する時間間隔
    public int attackDamage = 10;              // 攻撃のダメージ量

    Animator anim;                  // アニメーション
    GameObject player;              // プレイヤー
    PlayerHealth playerHealth;      // PlayerHealth スクリプト
    //EnemyHealth enemyHealth;
    bool playerInRange;             // 攻撃する範囲にプレイヤーが存在するかのフラグ
    float timer;                    // 経過時間

    void Awake ()
    {
        // プレイヤーを取得
        player = GameObject.FindGameObjectWithTag ("Player");
        // プレイヤーの HP を取得
        playerHealth = player.GetComponent <PlayerHealth> ();
        //enemyHealth = GetComponent<EnemyHealth>();
        // アニメータコンポーネントを取得
        anim = GetComponent <Animator> ();
    }

    // 物体と衝突したときの処理
```

```
    void OnTriggerEnter (Collider other)
    {
        // 衝突した物体がプレイヤーの場合
        if(other.gameObject == player)
        {
            // 攻撃する範囲にプレイヤーが存在するかのフラグをオンにする
            playerInRange = true;
        }
    }

    // 衝突した物体が離れた時の処理
    void OnTriggerExit (Collider other)
    {
        // 衝突した物体がプレイヤーの場合
        if(other.gameObject == player)
        {
            // 攻撃する範囲にプレイヤーが存在するかのフラグをオフにする
            playerInRange = false;
        }
    }

    void Update ()
    {
        // 経過時間を加算する
        timer += Time.deltaTime;

        // 攻撃の時間間隔よりも経過時間が長い、かつプレイヤーが攻撃範囲に存在する場合
        if(timer >= timeBetweenAttacks && playerInRange
            /* && enemyHealth.currentHealth > 0*/)
        {
            // プレイヤーを攻撃する
            Attack ();
        }

        // プレイヤーの HP が 0 以下になった場合
        if(playerHealth.currentHealth <= 0)
        {
            // プレイヤーが倒された時のアニメーションフラグをオンにする
            anim.SetTrigger ("PlayerDead");
        }
    }

    // 敵が攻撃する処理
    void Attack ()
    {
        // 経過時間を 0 にする
        timer = 0f;

        // プレイヤーの HP が 0 よりも大きい場合
        if(playerHealth.currentHealth > 0)
        {
            // プレイヤーにダメージを与える
            playerHealth.TakeDamage (attackDamage);
        }
```

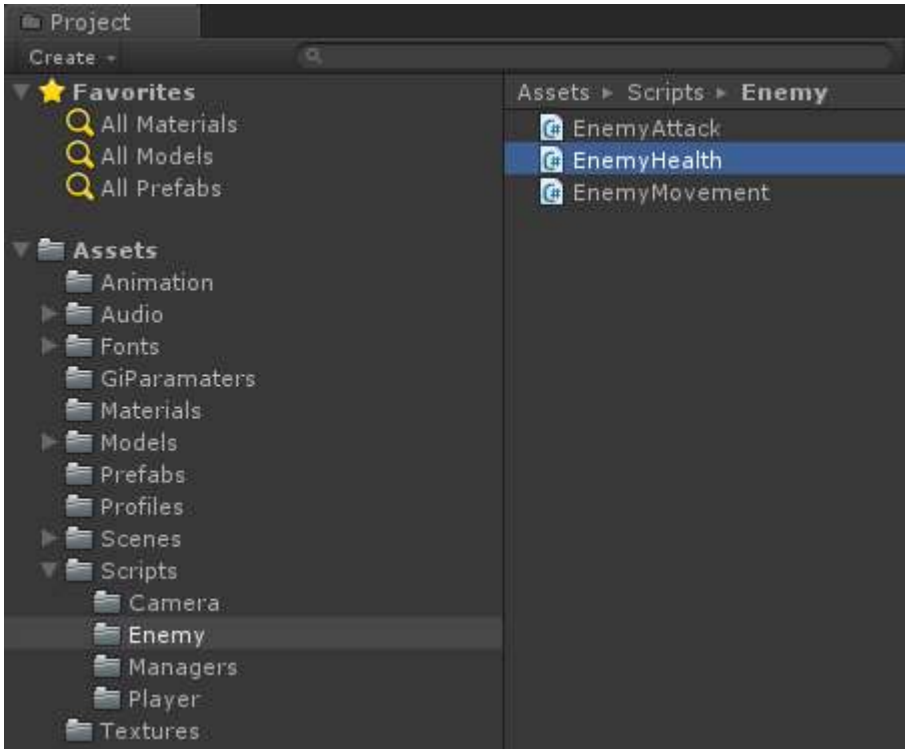
```
    }
}
```

これで敵キャラクタがプレイヤーキャラクタに接触すると，画面が赤くフラッシュし，HP ゲージが減っていく。HP ゲージが 0 になると，アニメーションが Death アニメーションに変わる。

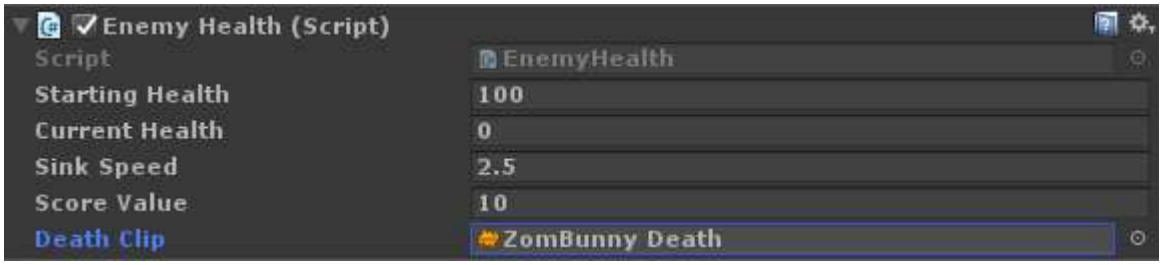
Phase7　敵をやっつける

1．敵の体力管理用スクリプトの設定

Project > Assets > Scripts > Enemy > EnemyHealth を，Hierarchy の Zombunny にドラッグする。



Zombunny にアタッチした EnemyHealth コンポーネントの Death Clip に Zombunny Death を設定する。



EnemyHealth をダブルクリックし，スクリプトの内容を確認する。

```
using UnityEngine;

public class EnemyHealth : MonoBehaviour
{
    public int startingHealth = 100;    // 敵の HP の初期値
    public int currentHealth;           // 敵の最新の HP
    public float sinkSpeed = 2.5f;      // 敵の消滅時間
    public int scoreValue = 10;         // 敵を倒した時のスコア
    public AudioClip deathClip;

    Animator anim;                      // アニメータ
    AudioSource enemyAudio;             // 敵の効果音
    ParticleSystem hitParticles;        // 攻撃されたときのエフェクト
    CapsuleCollider capsuleCollider;    // Collider
    bool isDead;                        // 敵が倒されたかフラグ
```

```
    bool isSinking;                         // 敵の消滅フラグ

    void Awake ()
    {
        // コンポーネントの設定
        anim = GetComponent <Animator> ();
        enemyAudio = GetComponent <AudioSource> ();
        hitParticles = GetComponentInChildren <ParticleSystem> ();
        capsuleCollider = GetComponent <CapsuleCollider> ();

        // 最新の HP を初期化
        currentHealth = startingHealth;
    }

    void Update ()
    {
        // 消滅フラグがオンの場合
        if(isSinking)
        {
            // 敵が沈んでいく
            transform.Translate (-Vector3.up * sinkSpeed * Time.deltaTime);
        }
    }

    // 攻撃を受けた時の処理
    public void TakeDamage (int amount, Vector3 hitPoint)
    {
        // 倒されたフラグがオンの場合
        if(isDead)
            return;
        // 攻撃を受けた時の音を鳴らす
        enemyAudio.Play ();
        // HP を減らす
        currentHealth -= amount;
        // 攻撃を受けた時のエフェクトを表示する
        hitParticles.transform.position = hitPoint;
        hitParticles.Play();
        // 敵の HP が 0 以下になった場合
        if(currentHealth <= 0)
        {
            // 敵が倒される
            Death ();
        }
    }

    // 敵が倒された場合の処理
    void Death ()
    {
        // 敵が倒されたかフラグをオンにする
        isDead = true;

        capsuleCollider.isTrigger = true;
        // 倒された時のアニメーショントリガーをオンにする
        anim.SetTrigger ("Dead");
```

```
        // 倒された時の効果音を設定する
        enemyAudio.clip = deathClip;
        // 敵が倒された時の効果音を鳴らす
        enemyAudio.Play ();
    }

    // 敵を消滅させる処理
    // Zombunny のアニメーション(Death)の中から呼び出されます
    // この呼び出す設定は既に完了しています
    public void StartSinking ()
    {
        // NavMesh Agent コンポーネントをオフにする
        GetComponent <UnityEngine.AI.NavMeshAgent> ().enabled = false;
        // Rigidbody コンポーネントの Kinematic をオンにする(重力の効果を受けない)
        GetComponent <Rigidbody> ().isKinematic = true;
        // 消滅フラグをオンにする
        isSinking = true;
        // ScoreManager.score += scoreValue;
        // 敵のオブジェクトを破壊する
        Destroy (gameObject, 2f);
    }
}
```

EnemyAttack スクリプトを開き，コメントアウトしている以下の 2 行//を消し，スクリプトを有効にする。

```
    //EnemyHealth enemyHealth;
    //enemyHealth = GetComponent<EnemyHealth>();
```

同様に，コメントアウトしている以下の条件を，/* と */ を消して条件を有効にする。

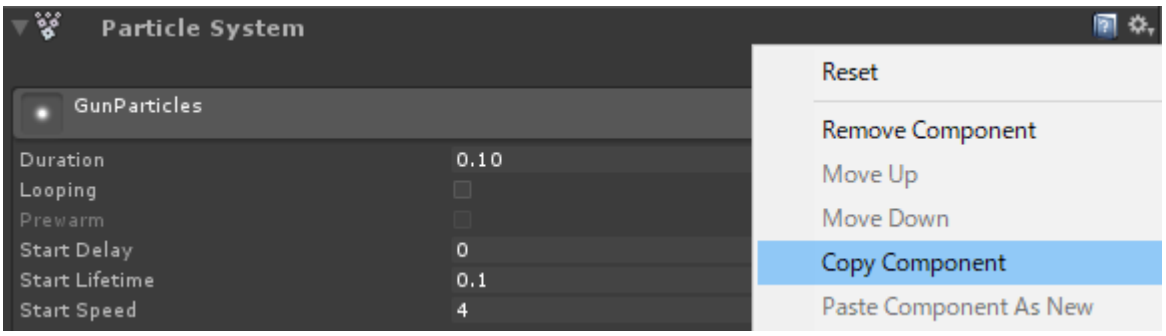
```
    if(timer >= timeBetweenAttacks && playerInRange/* && enemyHealth.currentHealth
> 0*/)
```

２．銃を撃つエフェクトの設定

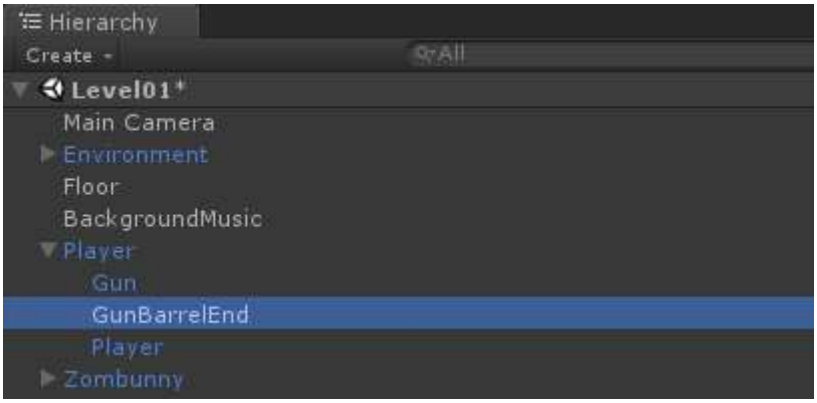
銃を撃ったときのエフェクトを設定する。

Project > Assets > Prefabs > GunParticles をクリックし，Inspector の Particle System コンポーネントのギアマークをクリックする。

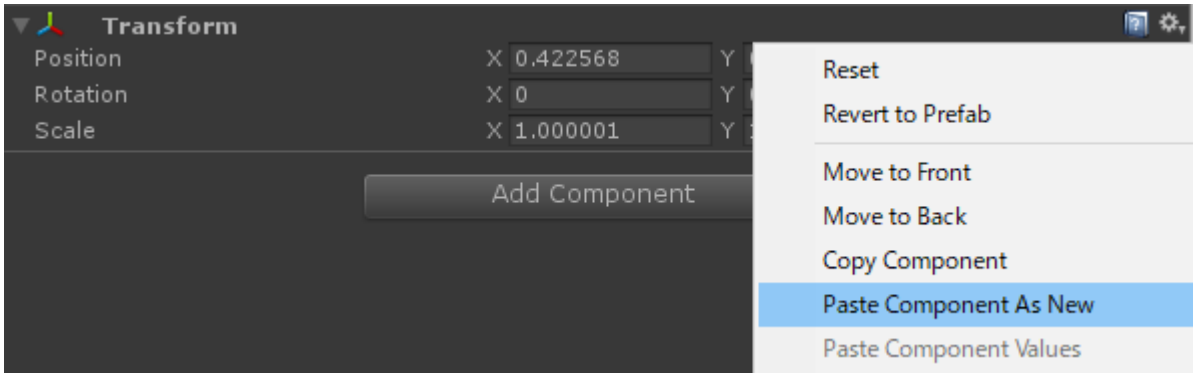
Copy Component をクリックし，このコンポーネントのコピーを作成する。



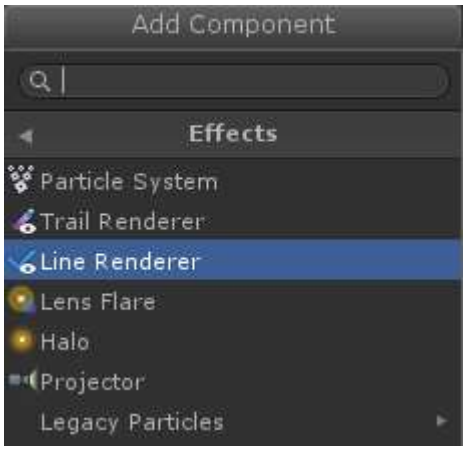
Hierarchy の Player を展開し，GunBulletEnd オブジェクトをクリックする。



Inspector の Transform コンポーネントのギアマークをクリックし，Paste Component As New をクリックし，先ほどコピーした Particle System コンポーネントをコピーする。



GunBulletEnd オブジェクトに射線を描画するための，Line Renderer コンポーネントを追加する。

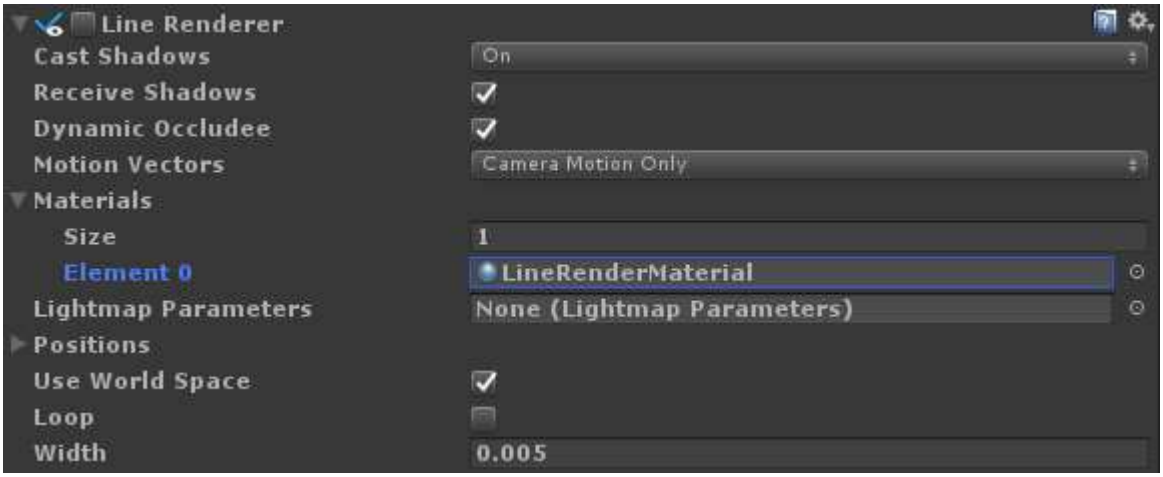


Line Renderer コンポーネントの以下の項目を設定する。

・Line Renderer コンポーネントを無効にする。（銃を撃ったときだけ射線を描画するため）

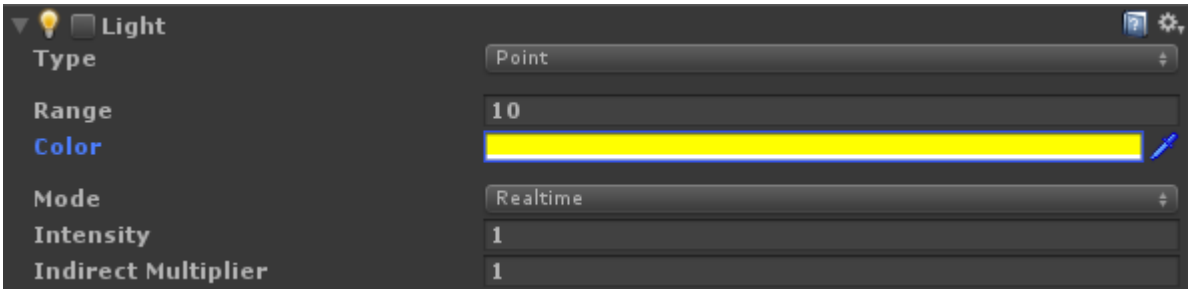
・Materials を展開し，Element0 に LineRenderMaterial を設定する。

・Width に 0.05 を設定する。



GunBulletEnd オブジェクトに，銃を発射したときに銃口を光らせる Light コンポーネントを追加する。

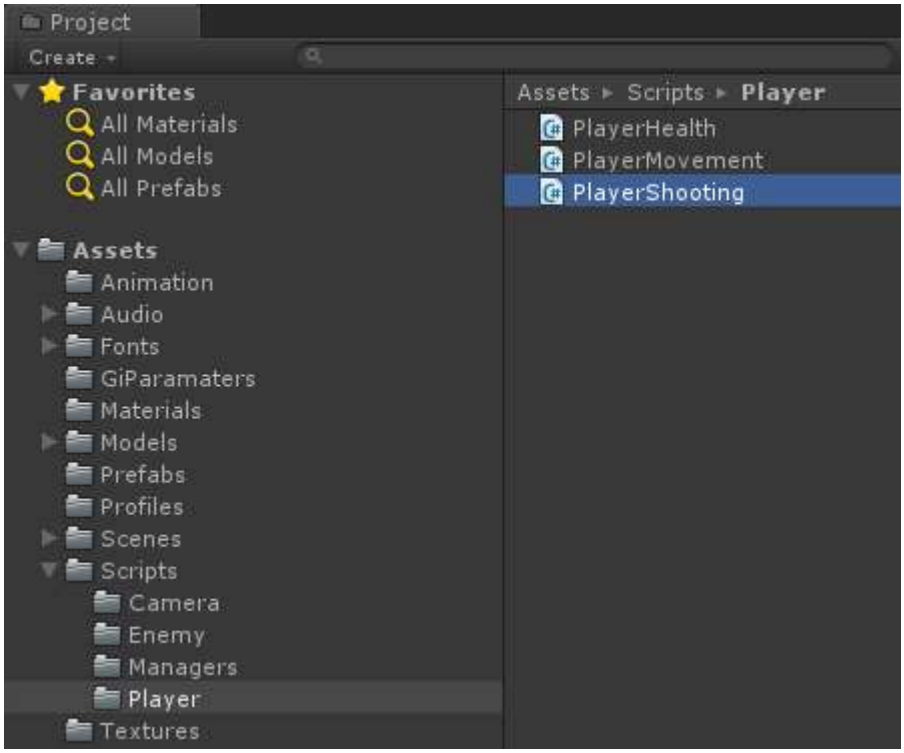
Light コンポーネントは銃を撃ったときに光るようにするため，デフォルトで無効にし，Color を黄色に設定する。



GunBulletEnd オブジェクトに銃を撃ったときの効果音を鳴らすための Audio Source コンポーネントを追加する。Audio Clip に Player GunShot を設定し，Play On Awake のチェックを外す。

３．銃を撃つスクリプトの設定

GunBulletEnd オブジェクトに PlayerShooting スクリプトをドラッグする。



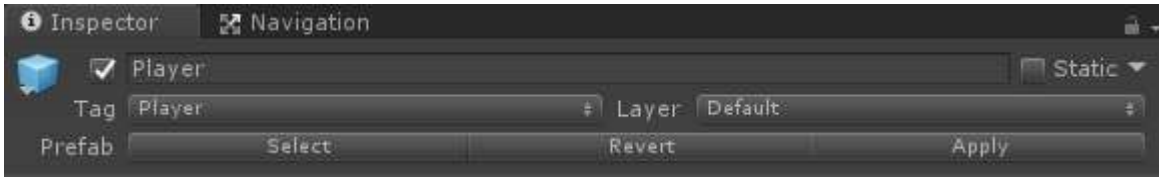
PlayerShooting スクリプトを開いて，スクリプトの内容を確認する。

```
using UnityEngine;

public class PlayerShooting : MonoBehaviour
{
    public int damagePerShot = 20;                  // 弾のダメージ
    public float timeBetweenBullets = 0.15f;    // 弾を撃つ間隔
    public float range = 100f;                      // 弾の飛距離

    float timer; //経過時間
    Ray shootRay = new Ray();                   // ray を，弾の攻撃範囲とする
    RaycastHit shootHit;                        // 弾が当たった物体
    int shootableMask;                          //撃てるもののみ識別する
    //弾を打った時のエフェクト
    ParticleSystem gunParticles;
    LineRenderer gunLine;
    AudioSource gunAudio;
    Light gunLight;
    float effectsDisplayTime = 0.2f;

    void Awake ()
    {
        // Shootable Layer を取得
        shootableMask = LayerMask.GetMask ("Shootable");
        // コンポーネントを取得
        gunParticles = GetComponent<ParticleSystem> ();
        gunLine = GetComponent <LineRenderer> ();
        gunAudio = GetComponent<AudioSource> ();
```

```
        gunLight = GetComponent<Light> ();
    }

    void Update ()
    {
        // 経過時間を計測
        timer += Time.deltaTime;

        // 弾を打つボタンを押したとき，かつ経過時間が弾を打つ間隔よりも大きい場合
                if(Input.GetButton ("Fire1") && timer >= timeBetweenBullets &&
Time.timeScale != 0)
        {
            // 弾を撃つ
            Shoot ();
        }

        // 経過時間がエフェクトの表示時間よりも大きくなった場合
        if(timer >= timeBetweenBullets * effectsDisplayTime)
        {
            // エフェクトを非表示にする
            DisableEffects ();
        }
    }

    // 銃を撃つエフェクトをオフにする処理
    public void DisableEffects ()
    {
        gunLine.enabled = false;
        gunLight.enabled = false;
    }

    // 弾を撃つ処理
    void Shoot ()
    {
        // 経過時間を初期化
        timer = 0f;

        // 弾を撃つエフェクトをオンにする
        gunAudio.Play ();
        gunLight.enabled = true;

        // 弾を連写することを想定して，オフにしてからオンにする
        gunParticles.Stop ();
        gunParticles.Play ();

        // 射線のスタート位置を設定する
        gunLine.enabled = true;
        gunLine.SetPosition (0, transform.position);

        // 弾の攻撃範囲のスタート位置を設定する
        shootRay.origin = transform.position;
        // 弾の飛んでいく方向を設定する
        shootRay.direction = transform.forward;
```

```
        // Ray を飛ばし，障害物に当たった場合
        if(Physics.Raycast (shootRay, out shootHit, range, shootableMask))
        {
            // 弾が当たった障害物の EnemyHealth スクリプトコンポーネントを取得する
            EnemyHealth enemyHealth =
                shootHit.collider.GetComponent<EnemyHealth>();

            // EnemyHealth スクリプトコンポーネントが null ではない場合(敵に弾が当たった場合)
            if(enemyHealth != null)
            {
                // 敵にダメージを与える
                enemyHealth.TakeDamage (damagePerShot, shootHit.point);
            }
            // 射線を障害物で当たった場所で止める
            gunLine.SetPosition (1, shootHit.point);
        }
        // 障害物に当たらなかった場合
        else
        {
            // 射線を弾の飛距離分表示する
            gunLine.SetPosition (1, shootRay.origin + shootRay.direction *
                range);
        }
    }
}
```

Hierarchy の Player をクリックし，Player オブジェクトに加えた変更をプレハブに反映させるため，Inspector の Apply ボタンをクリックする。

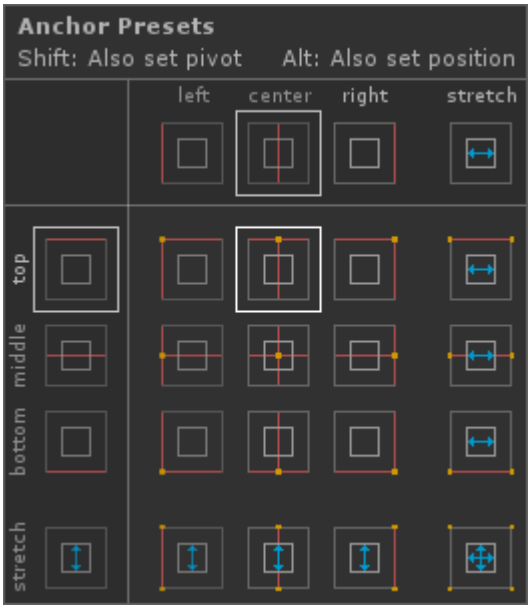


EnemyMovemennt スクリプトを開き，コメントアウトしてある箇所をすべて解除する。PlayerHealth スクリプトを開き，コメントアウトしてある箇所をすべて解除する。

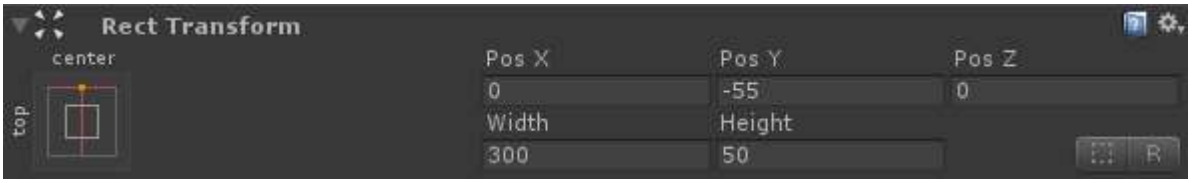
Phase8　得点を付ける

1．スコア UI の作成

Hierarchy の HUDCanvas を右クリックし UI > Text で Text オブジェクトを追加し，ScoreText にリネームする。Inspector の AnchorPresets をクリックし，中央（center）上（top）を選択する。

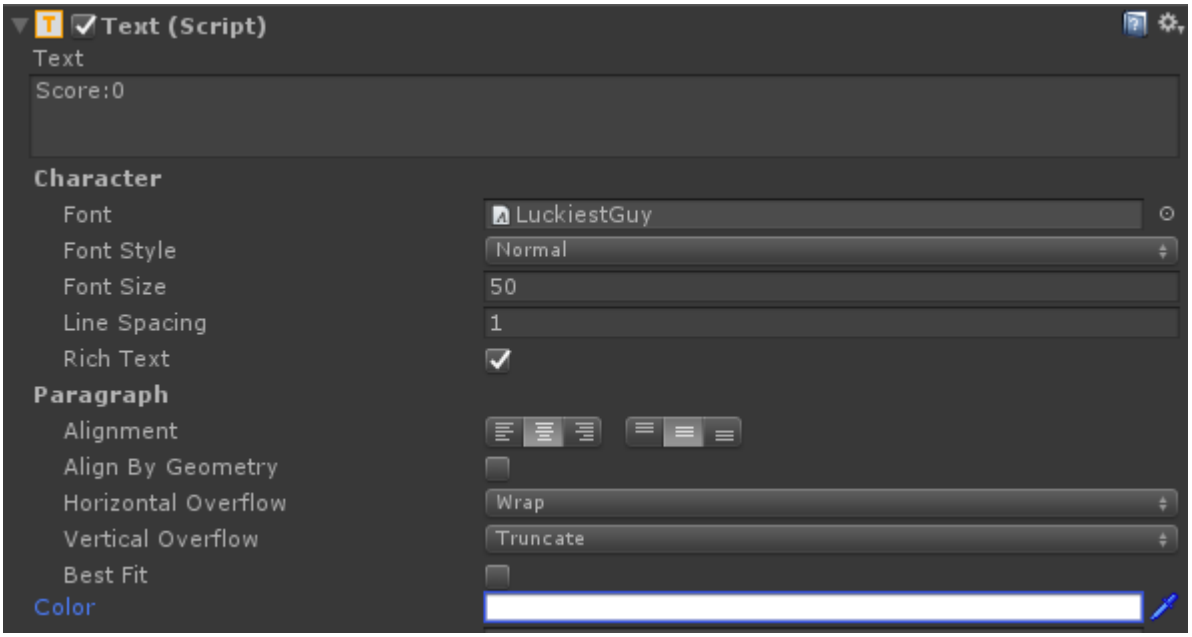


Transform コンポーネントの Position X を 0，Y を-55，Width を 300，Height を 50 に設定する。



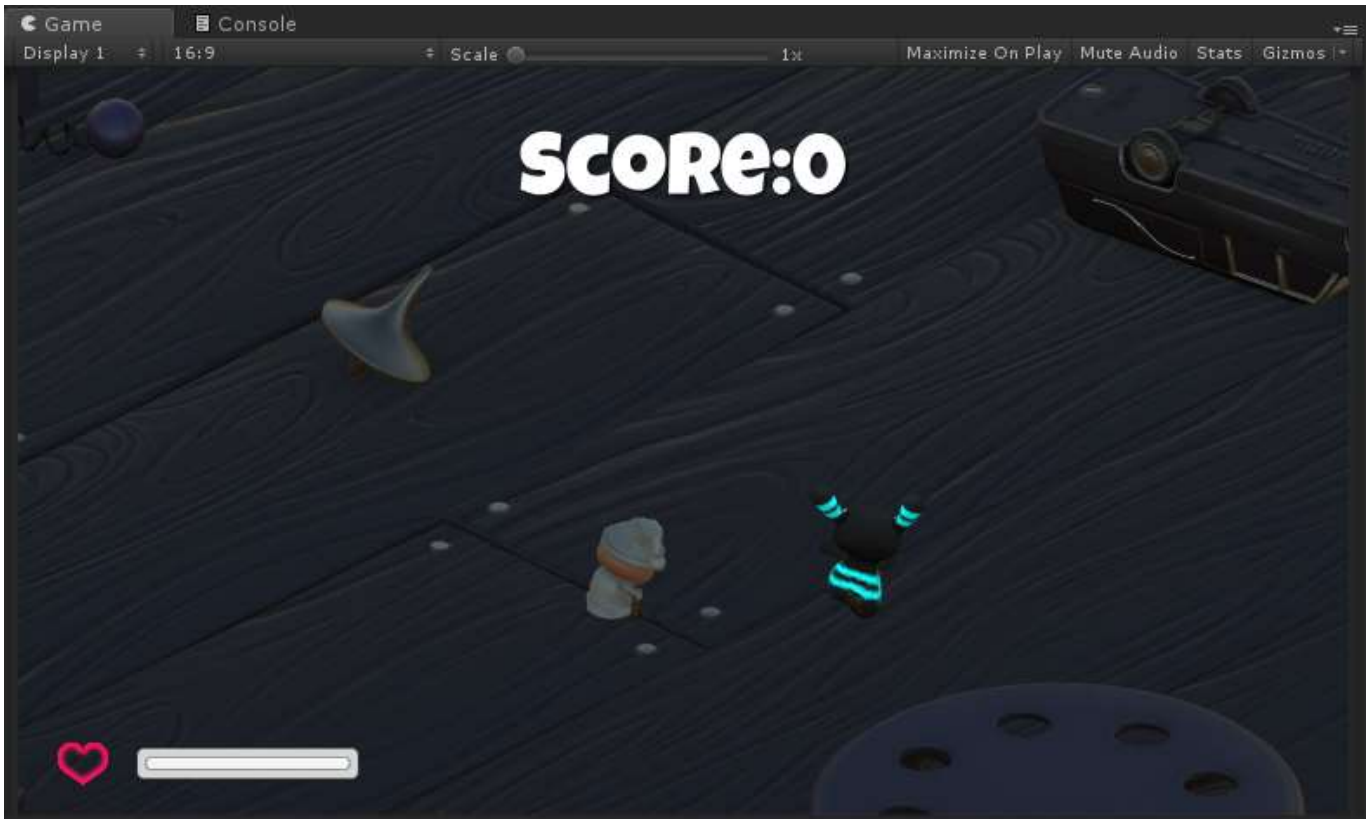
Text コンポーネント以下の値を設定する。

・Text：“Score:0”
・Font：LuckiesGuy
・Font Size：50
・Alignment：Center，Middle
・Color：白



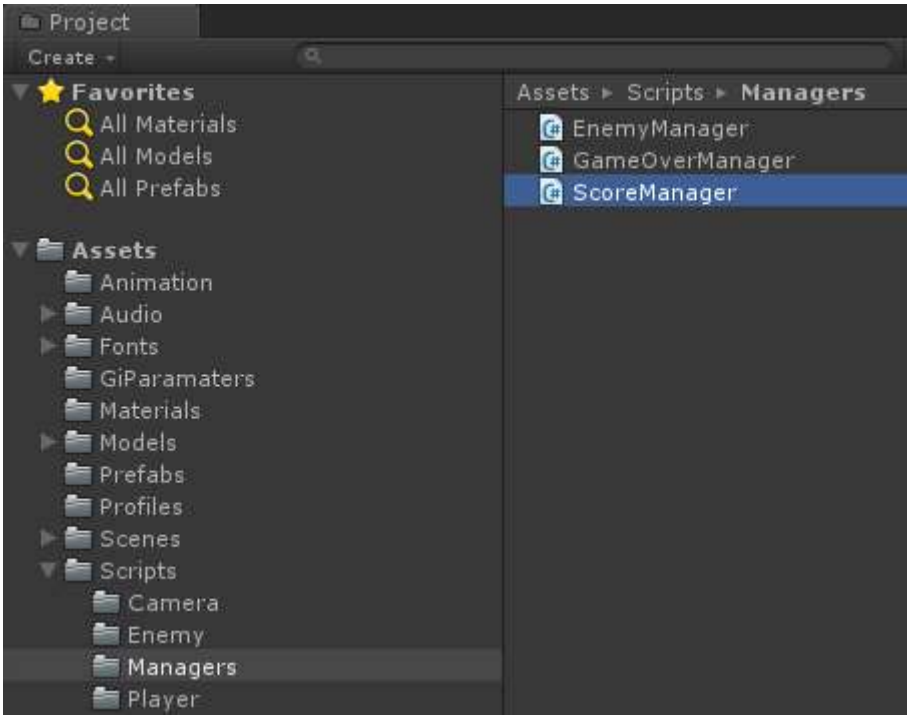
Text オブジェクトに Shadow コンポーネントを追加し, 影を描画することで立体的に見えるようにする。Add Component > UI > Effects > Shadow で Shadow コンポーネント追加。Shadow コンポーネントの Effect Distance X を 2，Y を-2 に設定する。

これで，スコアが画面上部中央に表示される。



２．スコアを表示するスクリプトの設定

Project > Assets > Scripts > Manager > ScoreManager を ScoreText にドラッグする。

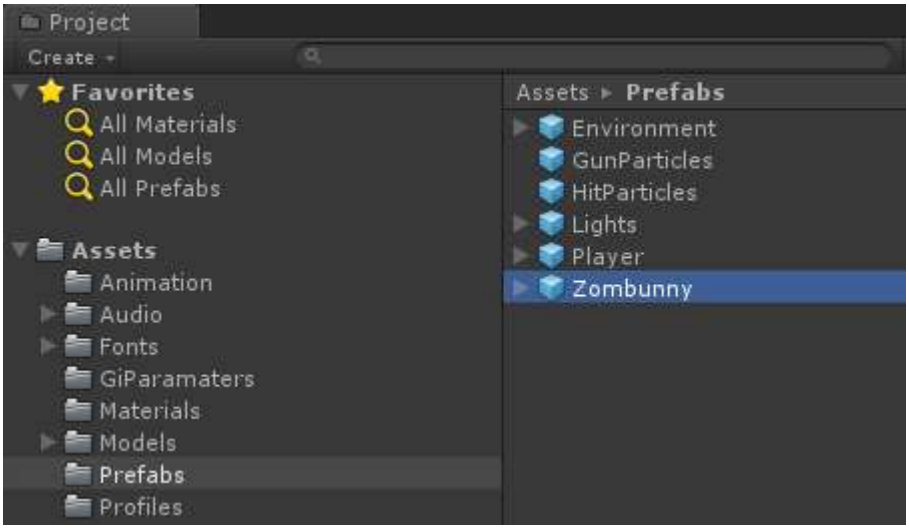


ScoreManager をダブルクリックし，スクリプトの内容を確認する。

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class ScoreManager : MonoBehaviour
{
    public static int score;     //スコア

    // スコア表示用のテキスト
    Text text;

    void Awake ()
    {
        // テキストコンポーネントのとの取得
        text = GetComponent <Text> ();
        // スコアの初期化
        score = 0;
    }

    void Update ()
    {
        // テキストコンポーネントのテキストに表示する文字列を設定
        text.text = "Score: " + score;
    }
}
```

Hierarchy の Zombunny オブジェクトの EnemyHealth スクリプトを開く。//ScoreManager.score +=

scoreValue;のコメントを削除し，敵キャラがやられたときに得点が加算されるようにする。
テストプレイし，Zombunny がやられたとき，得点が入るか確認する。
Zombunny オブジェクトを Prefabs フォルダにドラッグしプレハブ化する。

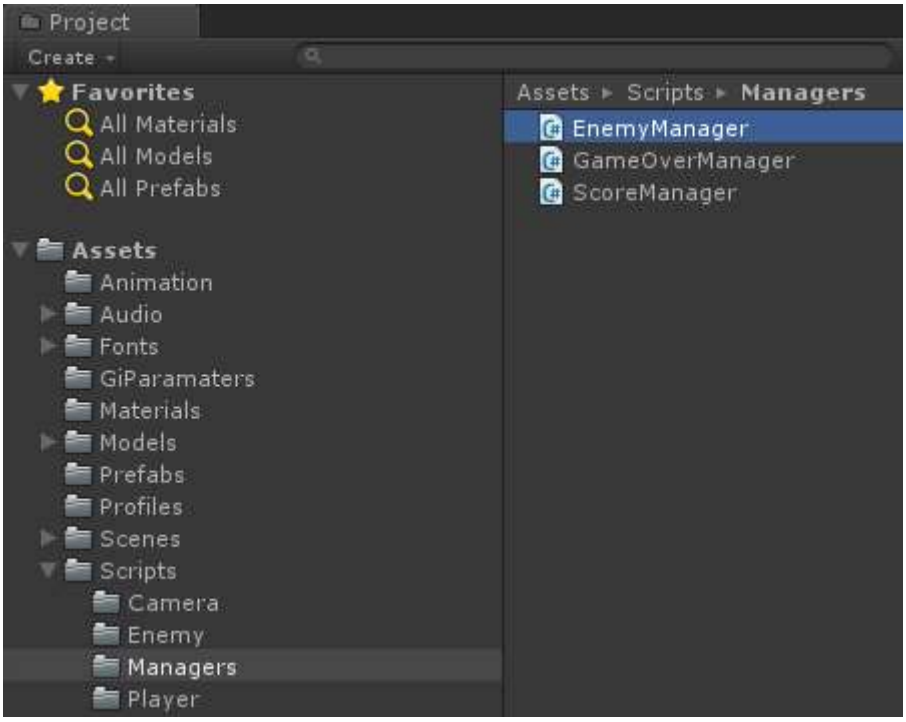


Hierarchy の Bombunny は削除する。

Phase9　もっと敵を出現させる

1．敵を自動生成する

Hierarchy に空のオブジェクトを追加し，EnemyManager にリネームする。Project > Assets > Scripts > Manager > EnemyManager スクリプトを，EnemyManager にドラッグする。

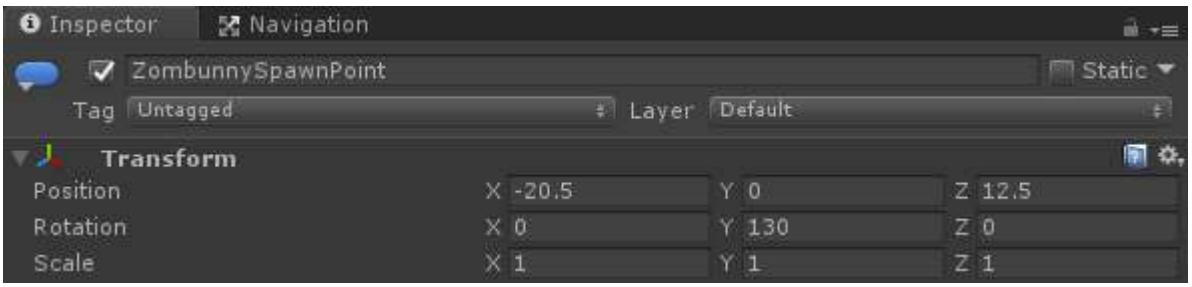


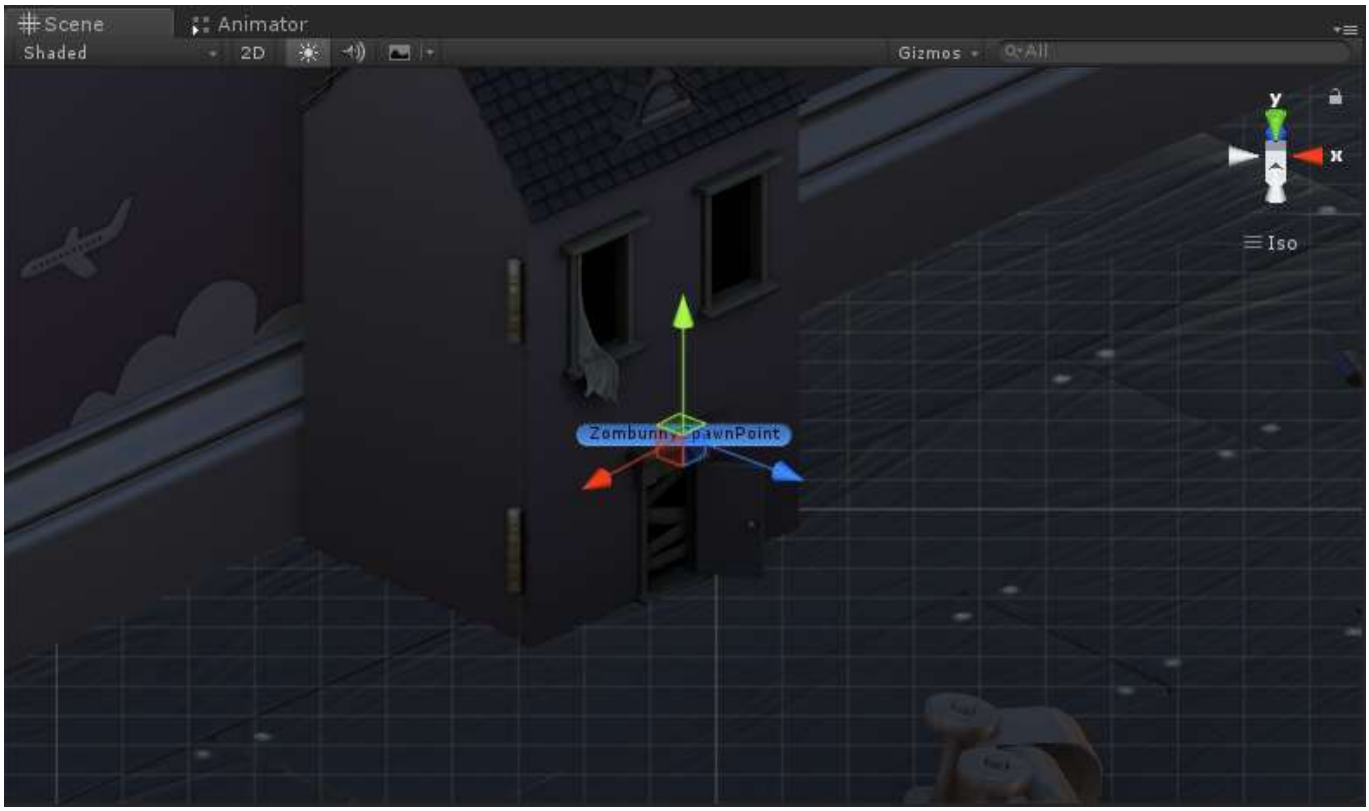
EnemyManager を開き，スクリプトの内容を確認する。

```
using UnityEngine;

public class EnemyManager : MonoBehaviour
{
    public PlayerHealth playerHealth;   // プレイヤーの HP
    public GameObject enemy;            // 敵のオブジェクト
    public float spawnTime = 3f;        // 敵の生成間隔
    public Transform[] spawnPoints;     // 敵を生成する場所

    void Start ()
    {
        // 敵を生成する間隔で，敵を生成する処理を呼ぶ
        InvokeRepeating ("Spawn", spawnTime, spawnTime);
    }

    // 敵を生成する処理
    void Spawn ()
    {
        // プレイヤーの HP が 0 以下の場合
        if(playerHealth.currentHealth <= 0f)
        {
            return;
        }

        // 敵を生成する場所をランダムに決める
```

```
        int spawnPointIndex = Random.Range (0, spawnPoints.Length);

        // 敵を生成する
        Instantiate (enemy, spawnPoints[spawnPointIndex].position,
            spawnPoints[spawnPointIndex].rotation);
    }
}
```

Hierarchy に空のオブジェクトを作成し，ZombunnySpawnPoint にリネームする。Inspector のオブジェクトのアイコンを青のカプセルアイコンに変更する。Transform コンポーネントの Position X を-20.5, Y を 0，Z を 12.5 に設定する。Rotation Y を 130 に設定する。



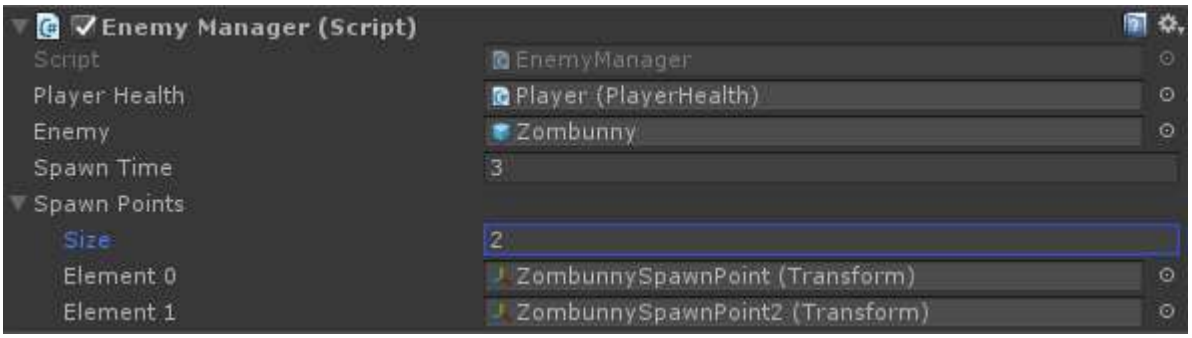


Hierarchy の EnemyManager をクリックし，Inspector の EnemyManager コンポーネントに以下の値を設定する。

・Player Health：Player（Hierarchy の Player をドラッグ）
・Enemy：Zombunny（Prefabs の Zombbuny をドラッグ）
・Spawn Points Size：1
・Spawn Ponts Element 0：ZombunnySpawnPoint（Hierarchy の ZombunnySpawnPoint をドラグ）

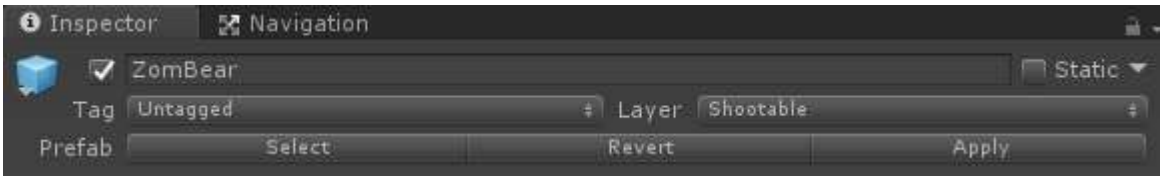
テストプレイして，敵が自動生成されるか確認する。

ZombunnySpawnPoint と同様のオブジェクト（敵の生成位置が異なる）を作成し，EnenyManger コンポーネントの，Spawn Points に追加することで，複数の Spawn Point のいずれかからランダムに敵が生成されるようになる。

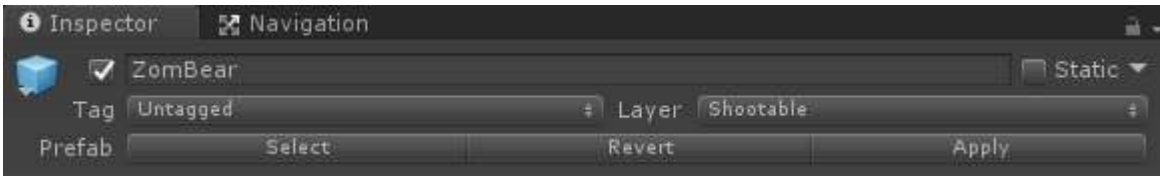


２．ZomBear を追加する

Project > Assets > Models > Characters フォルダ内の ZomBear を Hierarchy にドラッグする。Project > Prefabs フォルダ内の HitParticles を Zombear にドラッグする。



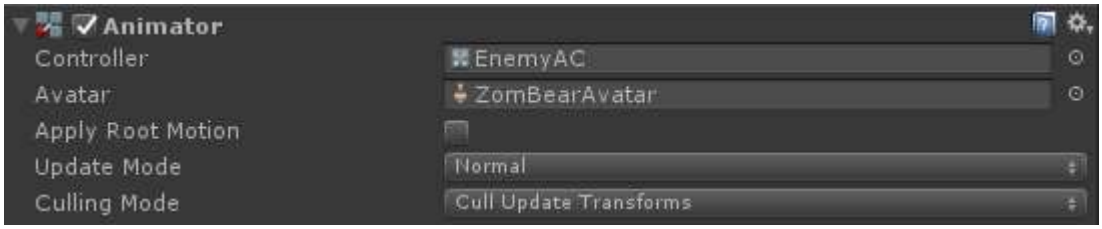
ZomBear の Layer を Shootable に設定する。



Zombunny プレハブを参考に，Zombear に以下のコンポーネントを追加，設定をする。

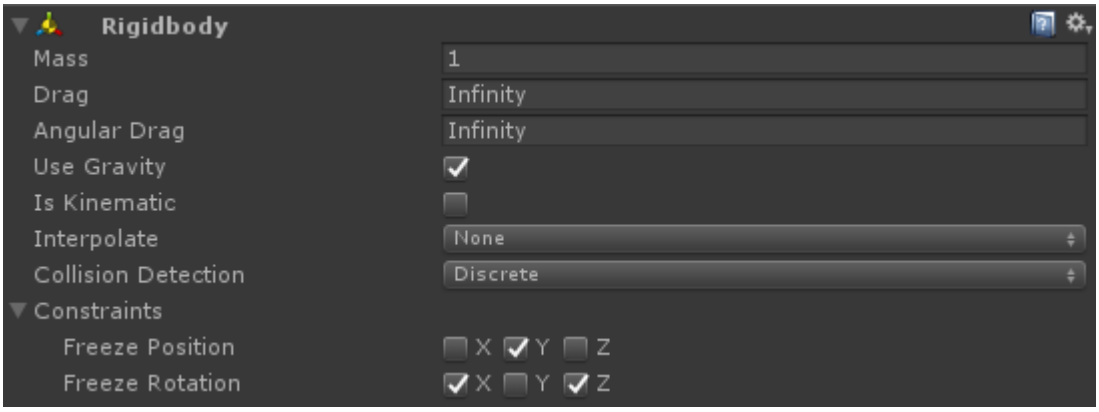
・Animator

Controller に Zombunny と同じ EnemyAC を設定する。



・Rigidbody

Zommbunny と同じく Drag，Angular Drag，Freeze Position，Freeze Rotation を設定する。

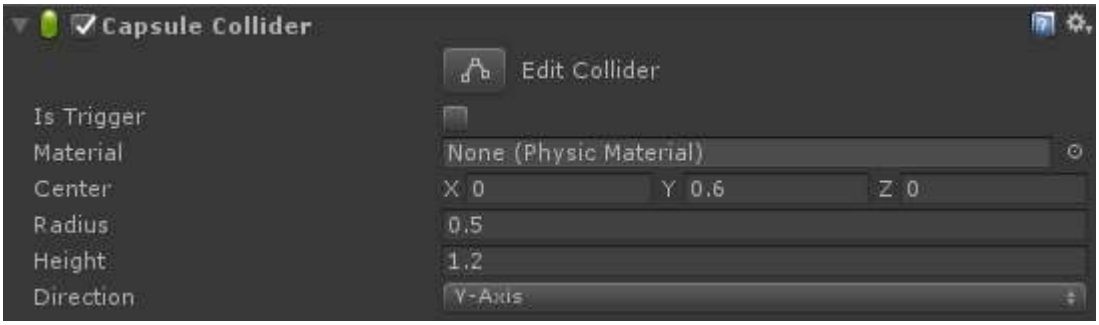


・AudioSource

ZomBear が攻撃を受けた時の効果音 ZomBear Hurt を設定する。

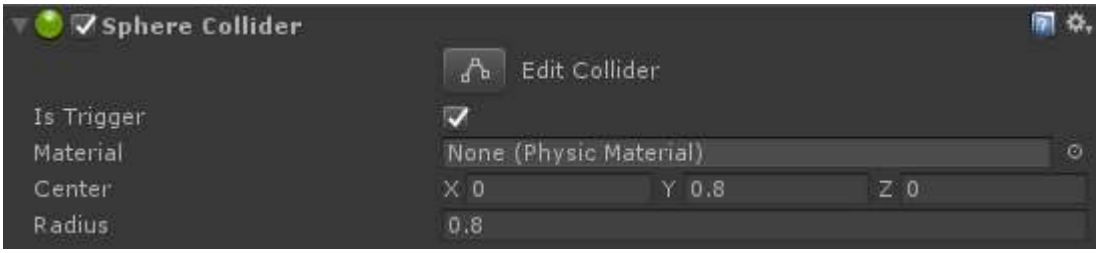
・Capsule Collider

衝突判定の範囲を設定する。



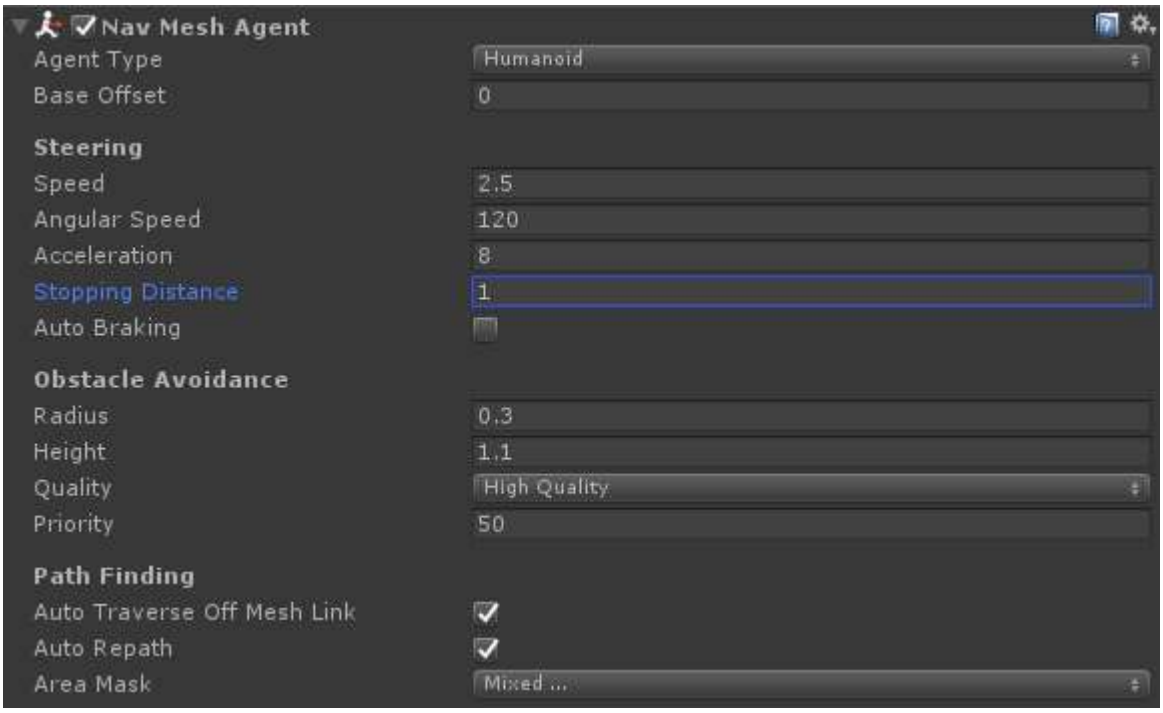
・Sphere Collider

ZomBear の攻撃範囲を設置し，Is Trigger をチェックする。



・Nav Mesh Agent

Radius，Speed，Stopping Distance，Height を設定する。



・Enemy Attack
攻撃を受けたときのダメージの値を設定する。

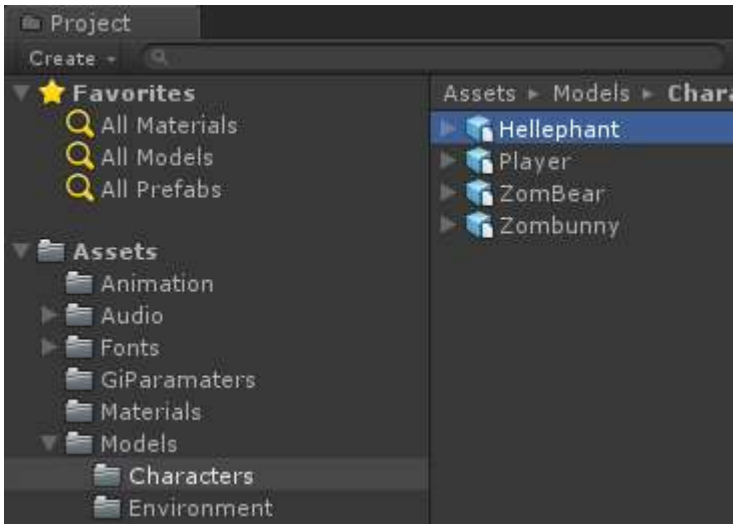
・Enemy Movement

・Enemy Health
やられたときにスコアに加算する得点と効果音を設定する。

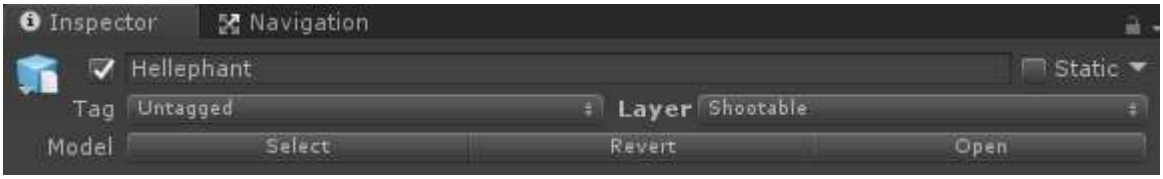
ZomBear オブジェクトを Prefabs フォルダにドラッグしプレハブ化し，Hierarchy から ZomBear を削除する。

３．Hellephant を追加する。

Project > Assets > Models > Characters フォルダ内の Hellephant を Hierarchy にドラッグする。Project > Prefabs フォルダ内の HitParticles を Hellephant にドラッグする。



Hellephant の Layer を Shootable に設定する。

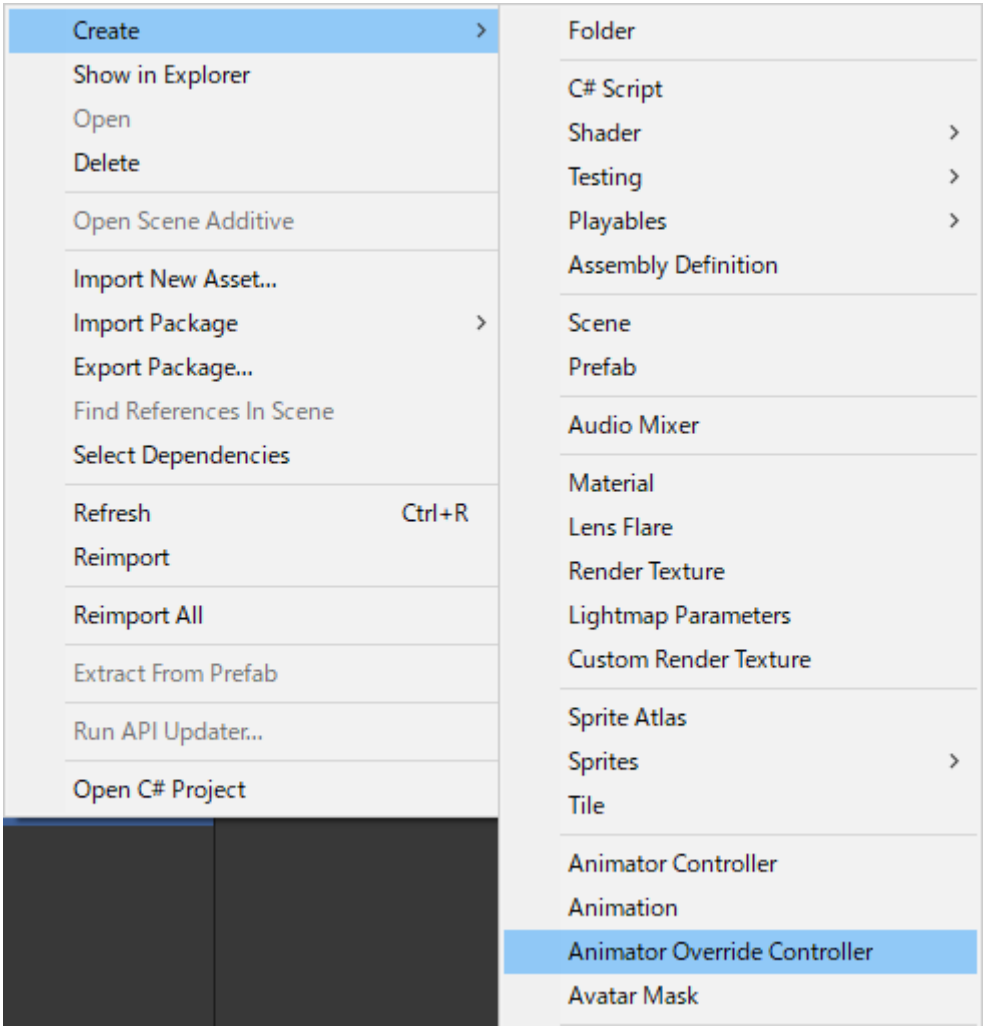


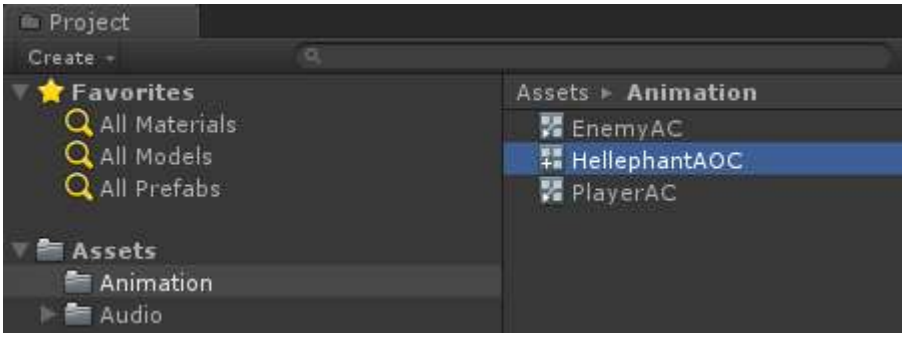
Zombunny プレハブを参考に，Hellephant に以下のコンポーネントを追加，設定をする。

・Animator

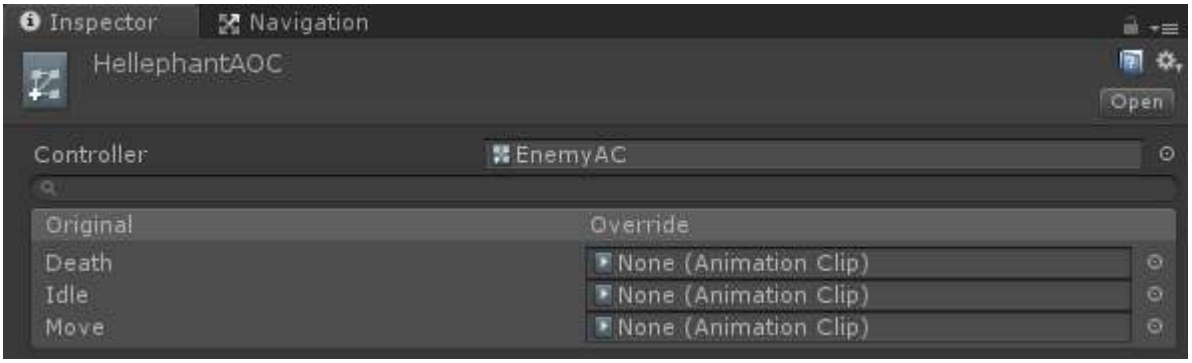
Animation フォルダ何に, EnemyAC の状態遷移を変えずにアニメーションを変更して Hellephant 用 AnimatorController を作成する。

Animation フォルダを右クリックし，Create > Animator Override Controller で Hellepahnt 用 AnimatorController を作成し，HellepahntAOC にリネームする。

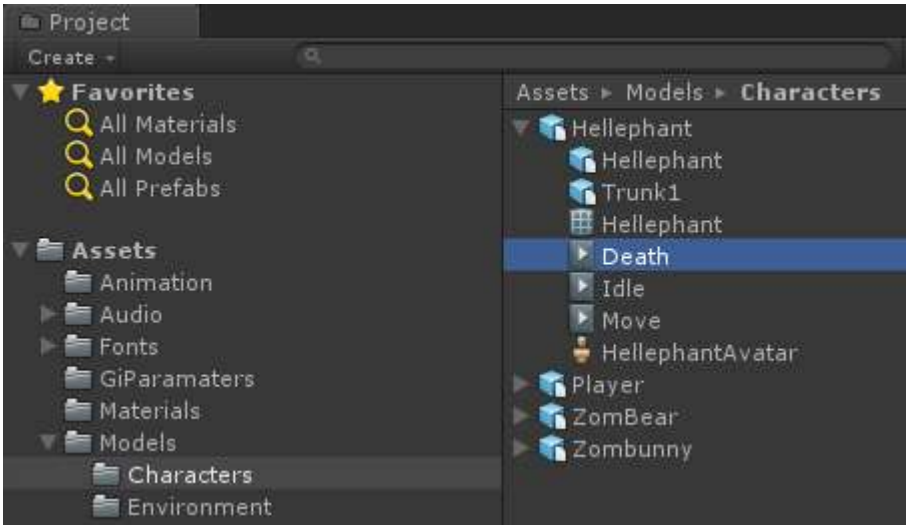




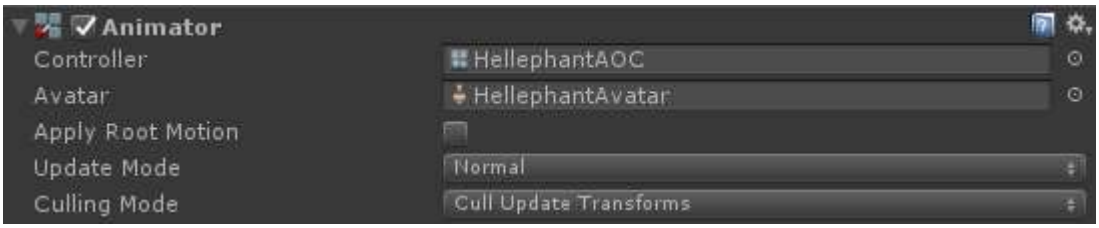
Project 内の HellephantAOC を選択した状態で，Inspector の Controller に EnemyAC を設定する。これで，アニメーションの状態遷移は EnemyAC と同じになる。



Originarl の Death，Idle，Move アニメーションを，Ellephant 用アニメーションに上書きする。

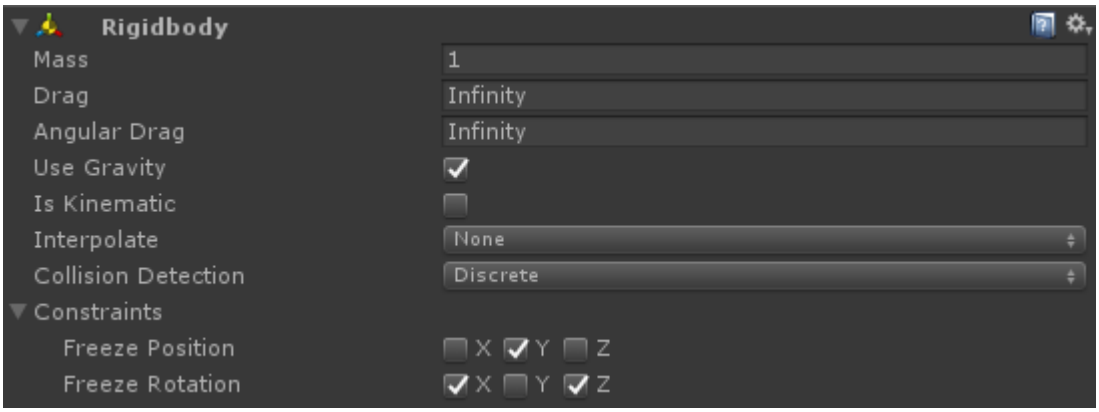


Controller に作成した HellephantAOC を設定する。



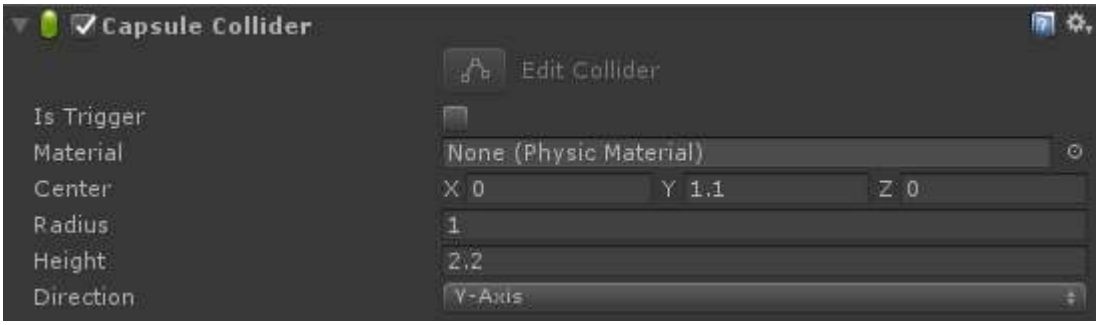
・Rigidbody

Zommbunny と同じく Drag，Angular Drag，Freeze Position，Freeze Rotation を設定する。

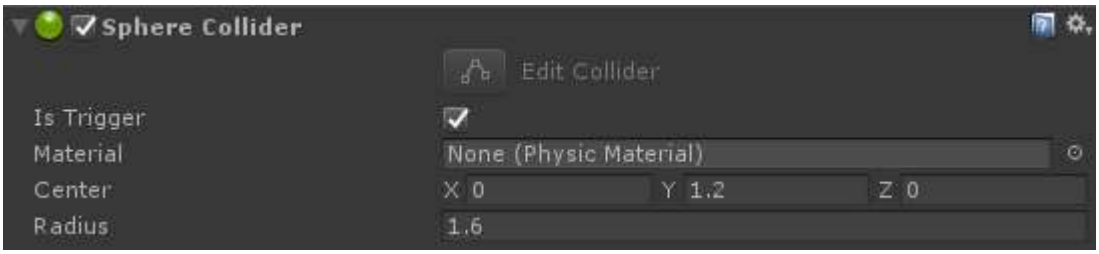


・AudioSource
Hellephant が攻撃を受けた時の効果音 Hellephant Hurt を設定する。

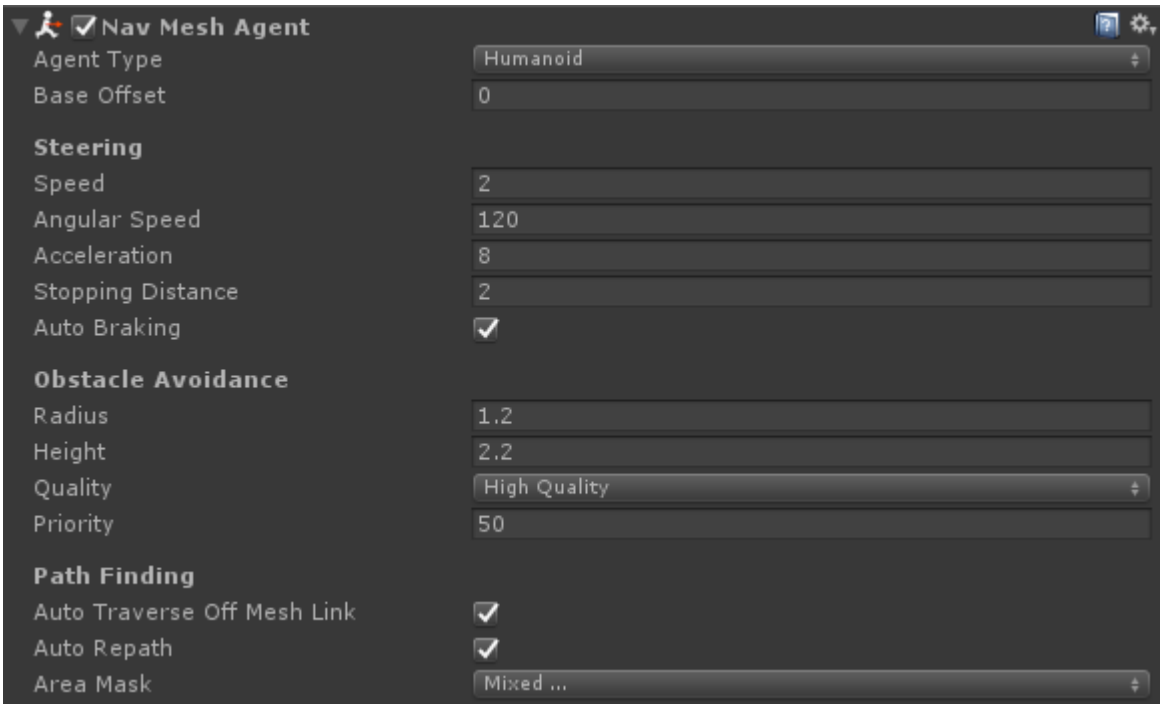
・Capsule Collider
衝突判定の範囲を設定する。



・Sphere Collider
Hellephant の攻撃範囲を設置し，Is Trigger をチェックする。



・Nav Mesh Agent
Radius，Speed，Stopping Distance，Height を設定する。



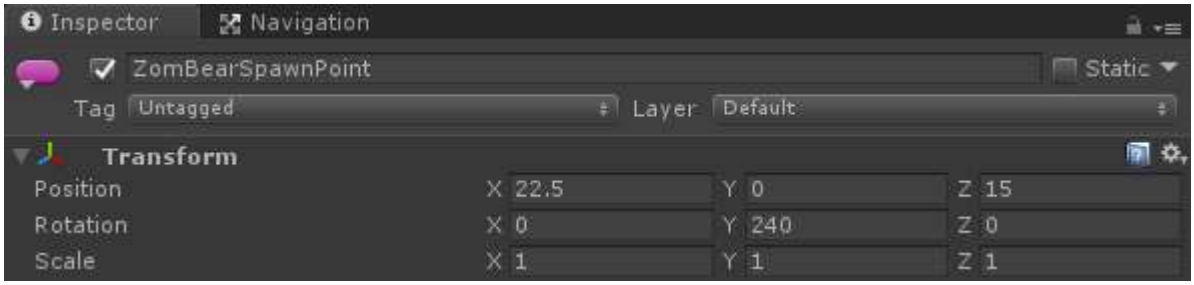
・Enemy Attack
攻撃を受けたときのダメージの値を設定する。

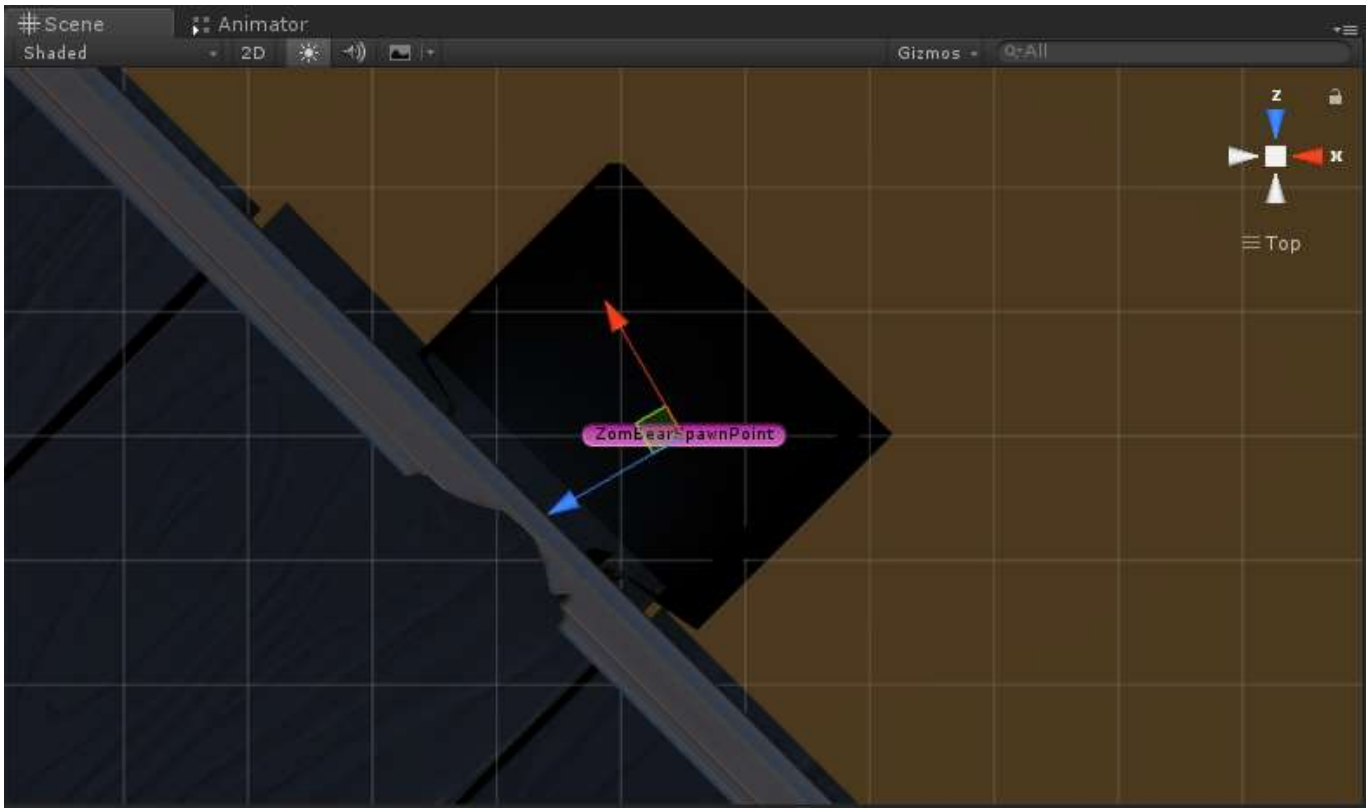
・Enemy Movement
・Enemy Health
Hellephant の体力，やられたときにスコアに加算する得点と効果音を設定する。

Hellephant オブジェクトを Prefabs フォルダにドラッグしプレハブ化し, Hierarchy から Hellepant を削除する。

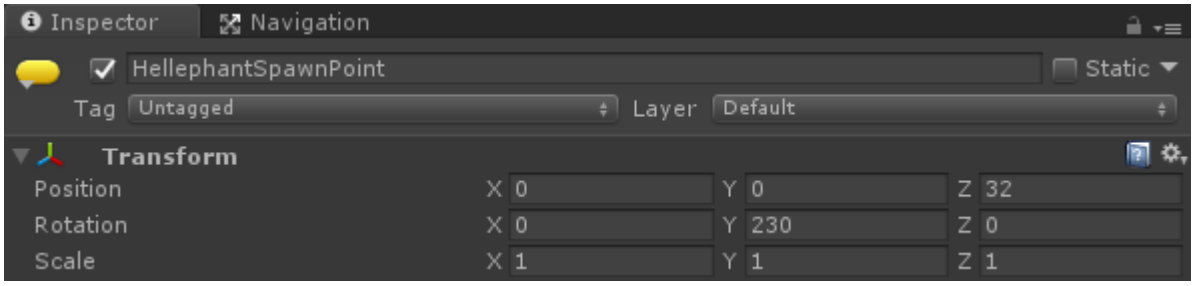
４．ZomBear と Hellephant の自動生成

Hierarchy に空のオブジェクトを作成し，ZomBearSpawnPoint にリネームする。Inspector のオブジェクトのアイコンをピンクのカプセルアイコンに変更する。Transform コンポーネントの Position X を 22.5，Y を 0，Z を 15 に設定する。Rotation Y を 240 に設定する。

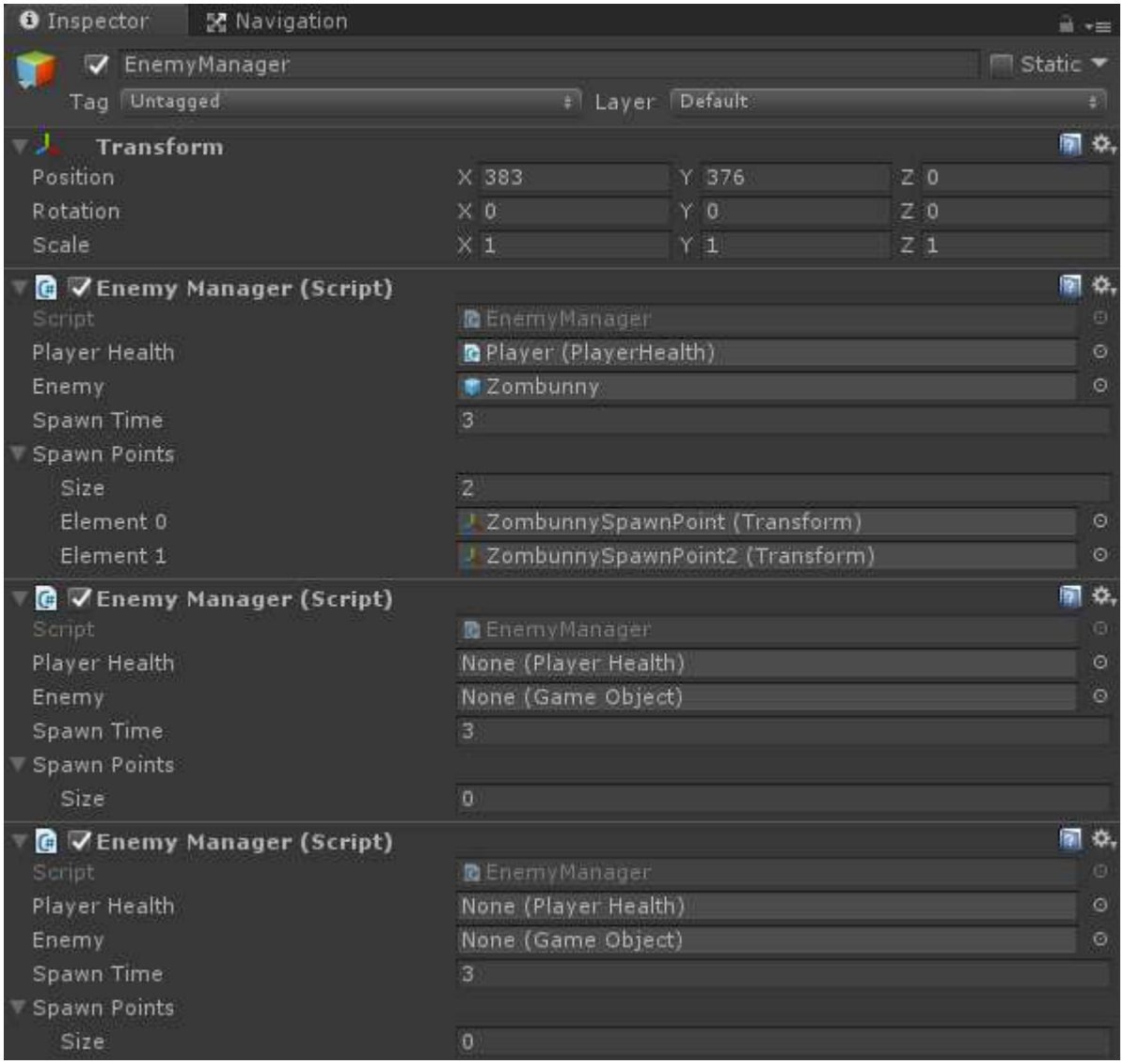




Hierarchy に空のオブジェクトを作成し，HellephantSpawnPoint にリネームする。Inspector のオブジェクトのアイコンを黄色のカプセルアイコンに変更する。Transform コンポーネントの Position X を 0，Y を 0，Z を 32 に設定する。Rotation Y を 230 に設定する。



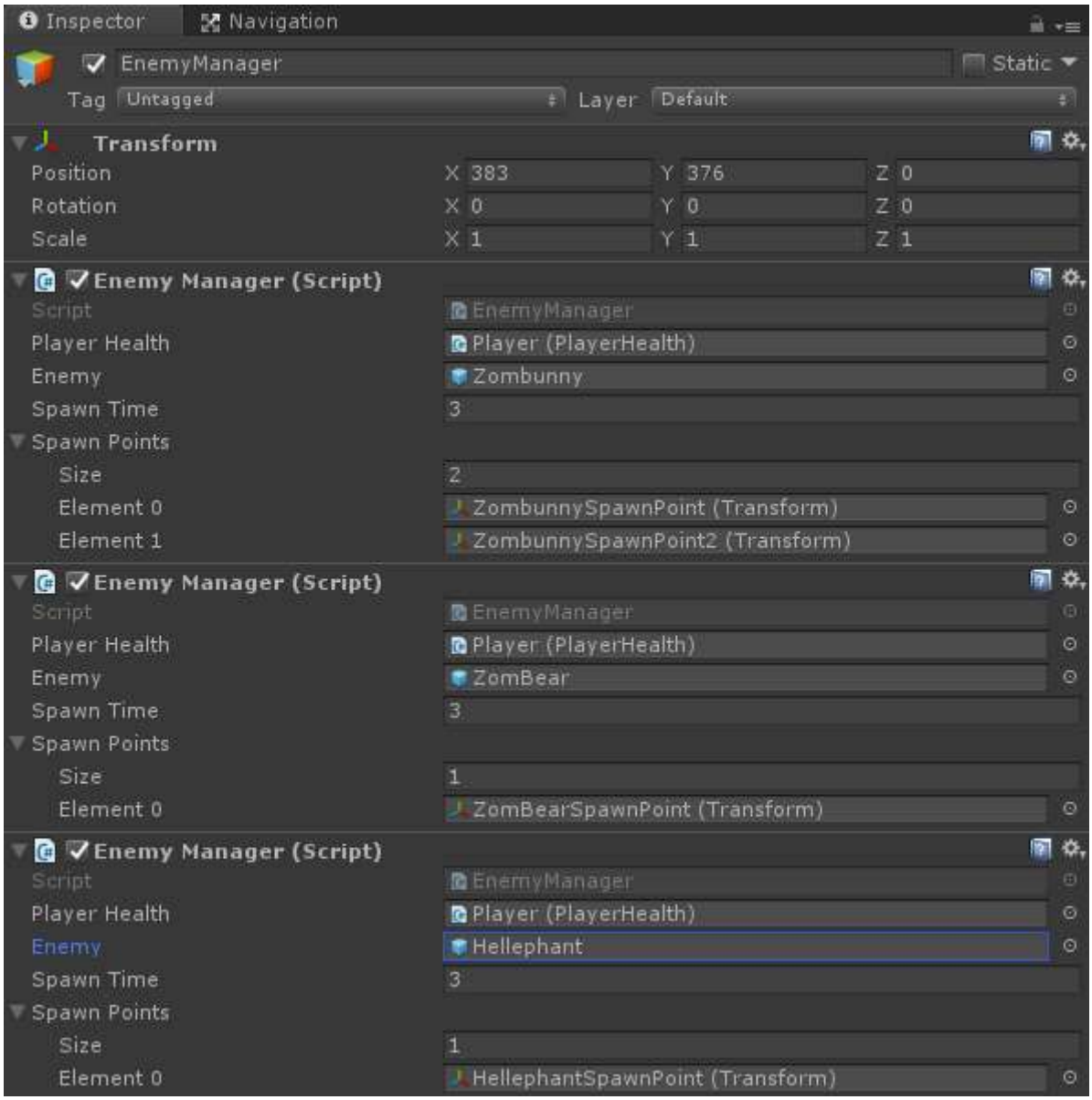
Priject > Assets > Scripts > Managers > EnemyManager スクリプトを Hierarchy の EnemyManager オブジェクトに 2 回（ZomBear 生成用と Helephant 生成用）ドラッグする。Inspector で EnemyManager コンポーネントを 3 つ持っているか確認する。



追加した EnemyManager コンポーネントの Player Health に Hierarchy から Player をドラッグする。

Prefabs フォルダから ZomBear を 2 番目の EnemyManager の Enemy にドラッグし，Hellephant は 3 番目の EnemyManager の Enemy にドラッグする。

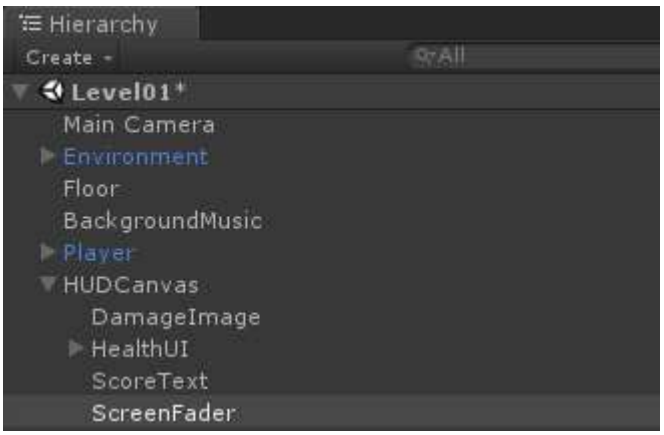
Hierarchy の ZomBearSpawnPoint を 2 番目の EnemyManager の SpawnPoints にドラッグし，HellepahntSpawnPoint を 3 番目の EnemyManager の SpawnPoints にドラッグする。


Inspector
Navigation
EnemyManager
Static
Tag Untagged
Layer Default
Transform
Position X 383 Y 376 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Enemy Manager (Script)
Script EnemyManager
Player Health Player (PlayerHealth)
Enemy Zombunny
Spawn Time 3
Spawn Points
Size 2
Element 0 ZombunnySpawnPoint (Transform)
Element 1 ZombunnySpawnPoint2 (Transform)
Enemy Manager (Script)
Script EnemyManager
Player Health Player (PlayerHealth)
Enemy ZomBear
Spawn Time 3
Spawn Points
Size 1
Element 0 ZomBearSpawnPoint (Transform)
Enemy Manager (Script)
Script EnemyManager
Player Health Player (PlayerHealth)
Enemy Hellephant
Spawn Time 3
Spawn Points
Size 1
Element 0 HellephantSpawnPoint (Transform)

Phase10　ゲームオーバー

1．ゲームオーバー画面の作成

HUDCanvas を右クリック, UI > Image で Image オブジェクトを追加し, ScreenFader にリネームする。これが，ゲームオーバー時の背景画面となる。



Rect Transform コンポーネントの Anchor Presets をクリックし，Alt キーを押しながら右下の縦横方向にストレッチをクリックする。

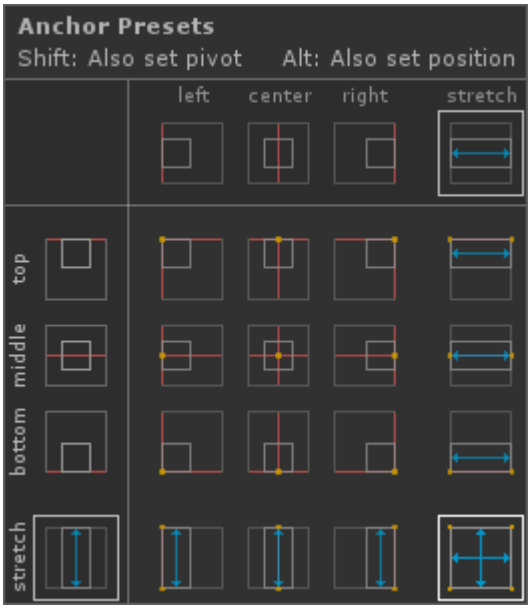
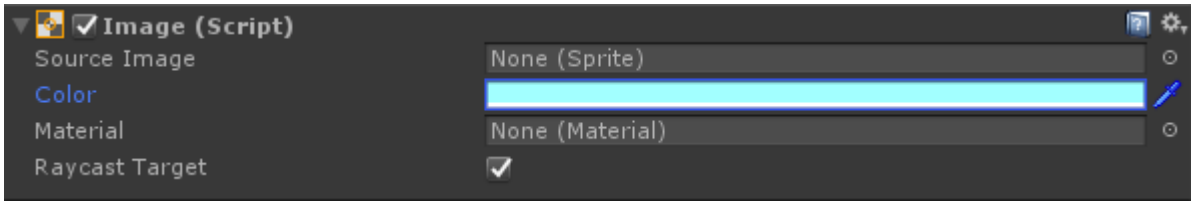
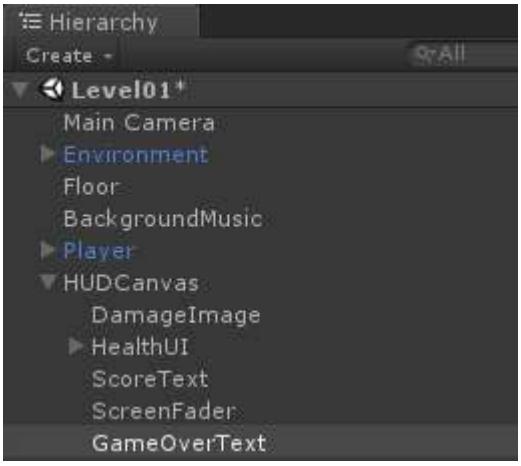


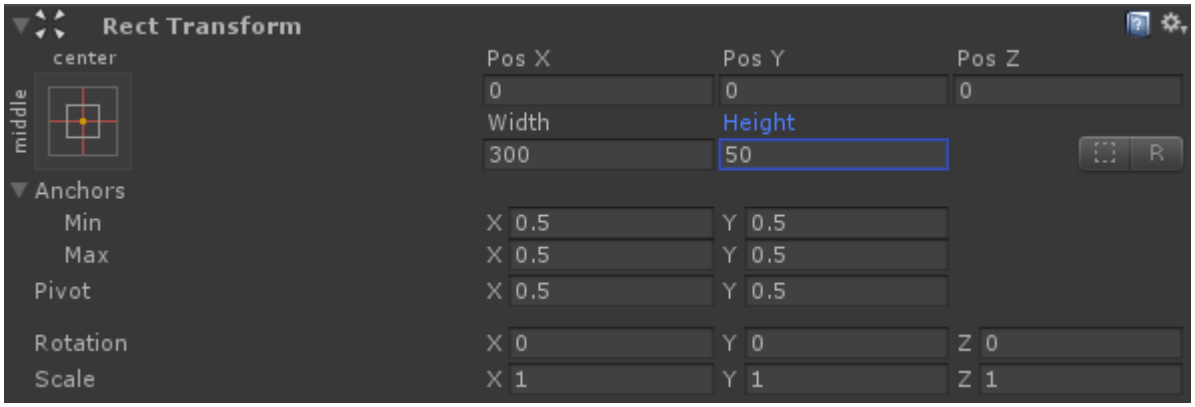
Image コンポーネントの Color を薄い青色に設定する。



HUDCanvas を右クリック，UI > Text で Text オブジェクトを追加し，GameOverText にリネームする。



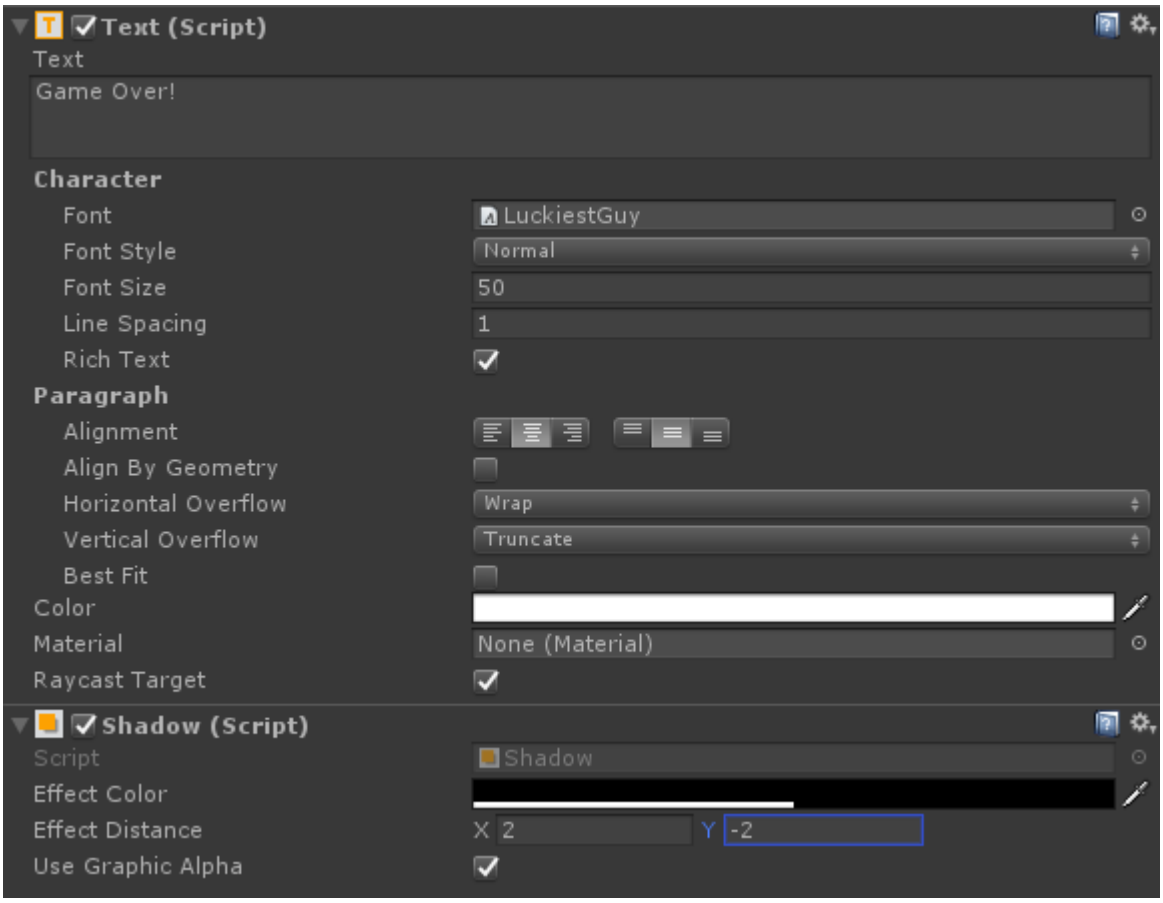
Rect Transform コンポーネントの Anchor Presets をクリックし，Alt キーを押しながら Middle，center をクリックする。Width を 300，Height を 50 に設定する。



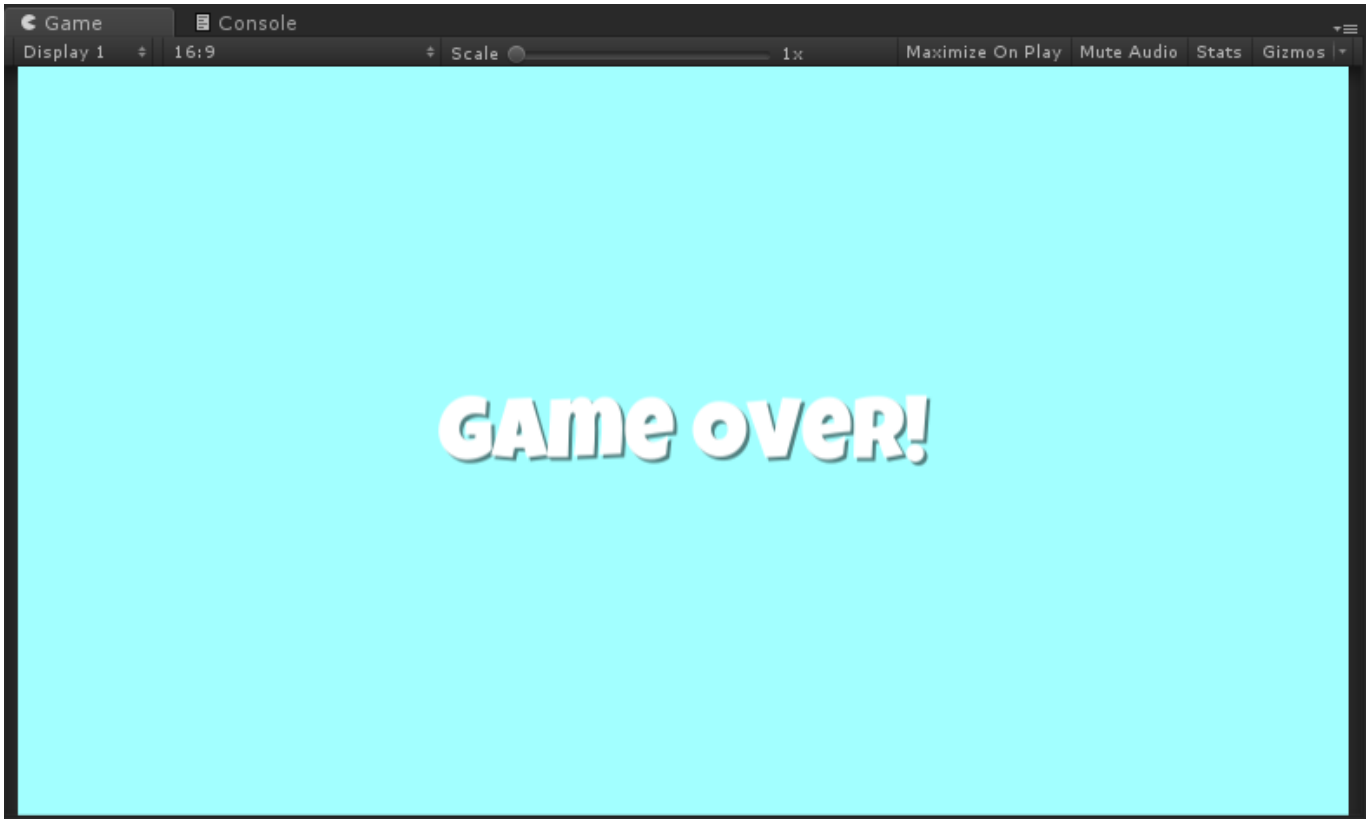
Text コンポーネントに以下の内容を設定する。

・Text：”Game Over!”
・Font：LuckiesGuy
・Font Size：50
・Alignment：Middle，Center
・Color：白

Shadow コンポーネントを追加（Add Component > UI > Effects > Shadow）し影を設定する。



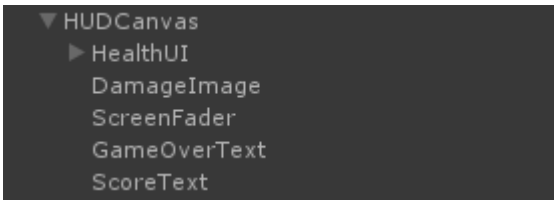
これで，ゲームオーバー画面が表示される。



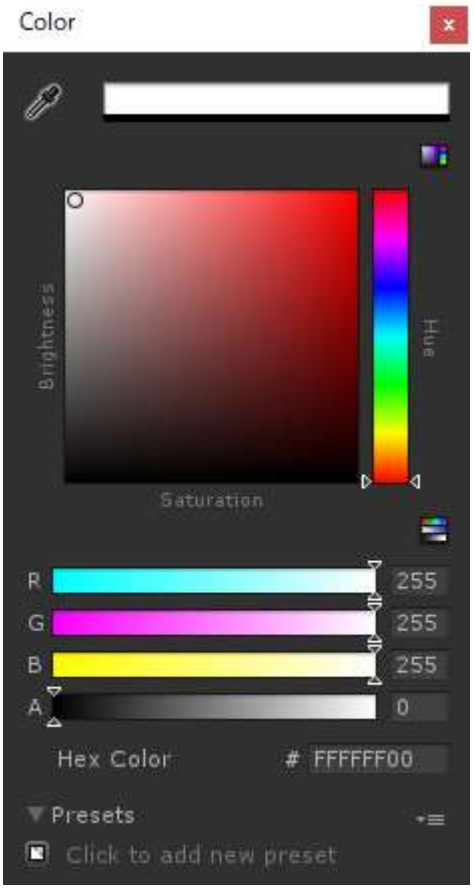
ゲームオーバー時に表示する UI オブジェクトを調整するため，HUDCanvas の子オブジェクトの並びを次の順番にする。

・HeaithUI

・DamageImage
・ScreenFader（これ以降がゲームオーバー時に表示される）
・GameOverText
・ScoreText

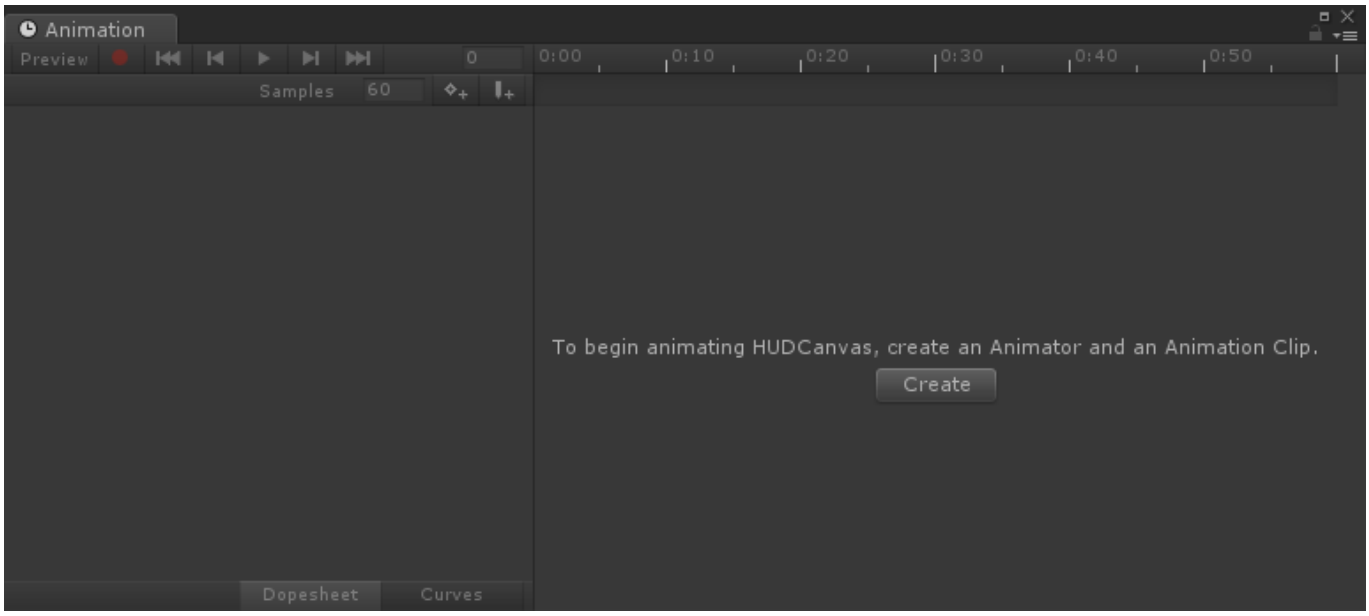


ScreenFader と GameOverText はゲームオーバーになったときだけ表示するため，ゲームのプレイ中は表示させない。Text コンポーネントの Color のアルファ値（A）を 0 に設定しておく。

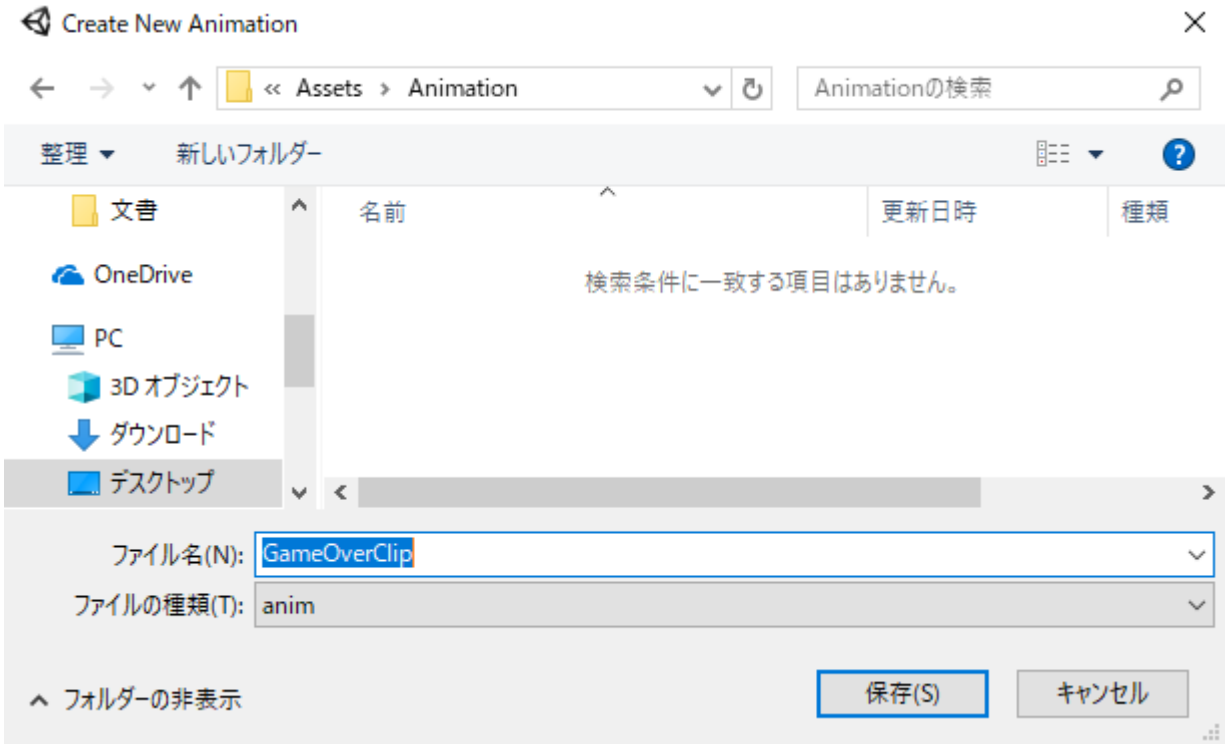


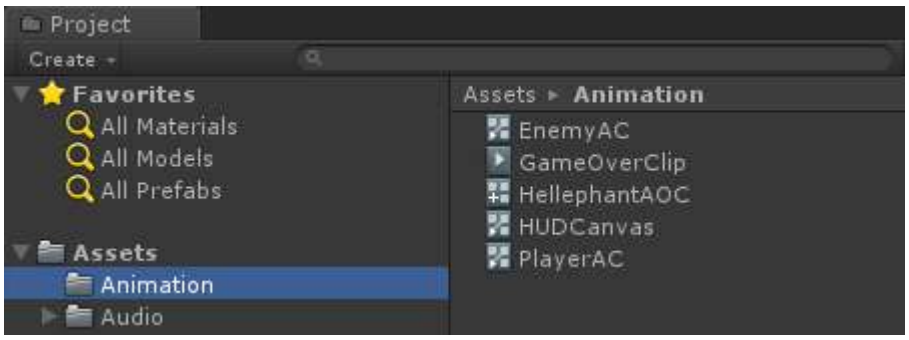
２．ゲームオーバー画面のアニメーション作成

Hierarchy の HUDCanvas をクリックした状態で，メニューの Window > Animation をクリックし，Animation ウインドウを開く。



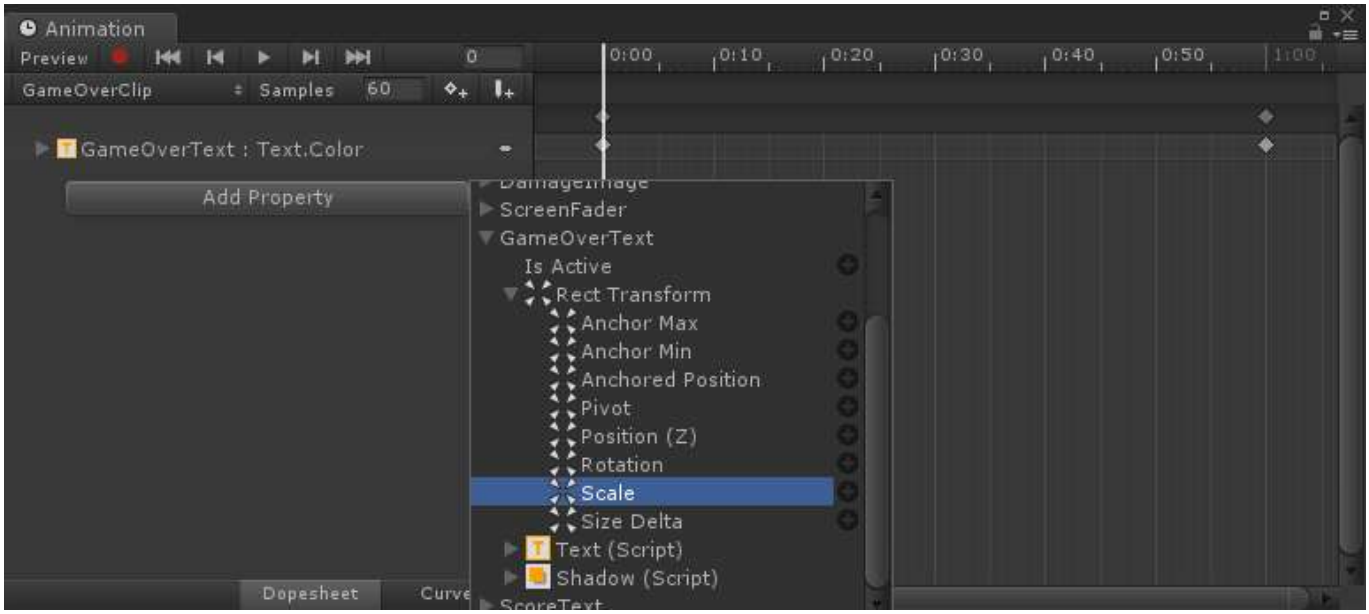
Create ボタンをクリックし，プロジェクトの Animation フォルダ内に GameOverClip アニメーションを作成する。AnimatorController も同時に作成される。



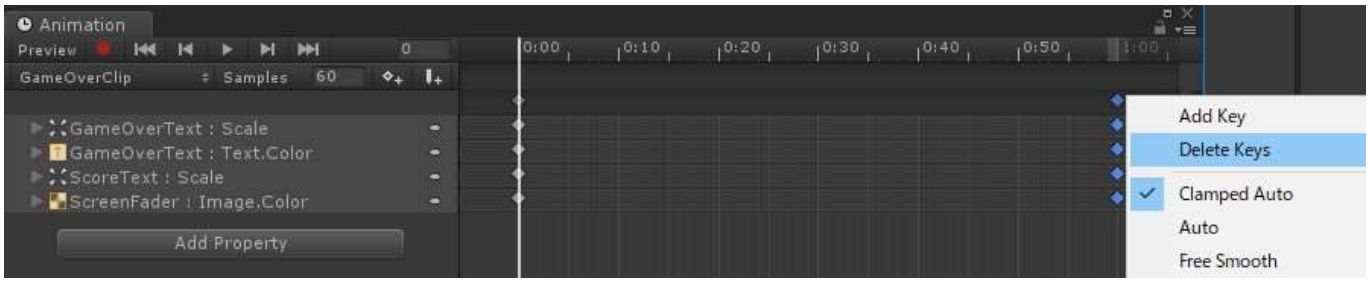


Animation ウインドウの Add Property ボタンをクリックし，アニメーションさせる要素を追加していく。今回アニメーションさせるのは，以下の 4 つの要素になる。

・GameOverText > Text > Color +
・GameOverText > RectTransform > Scale +
・ScreenFader > Image > Color +
・ScoreText > RectTransform > Scale +



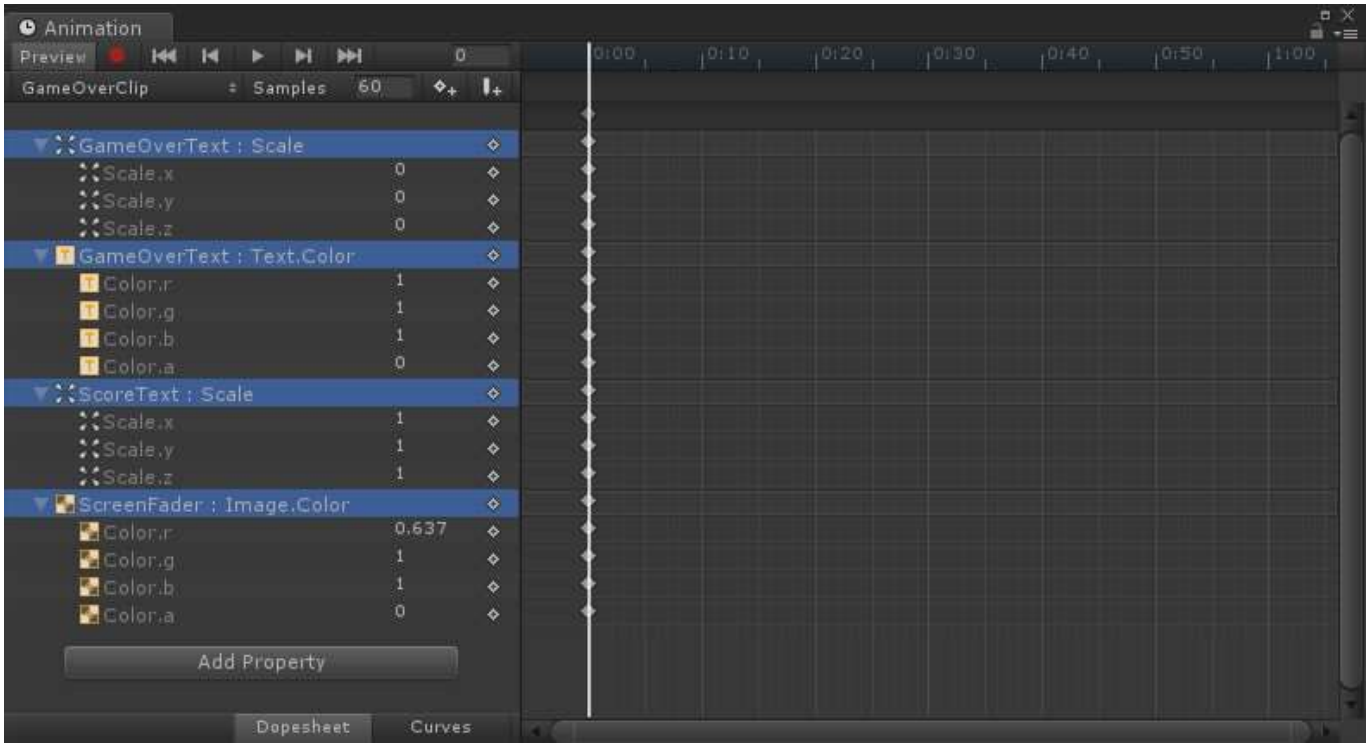
60（1:00）フレームのキーフレームを右クリックし Delete Keys で削除する。



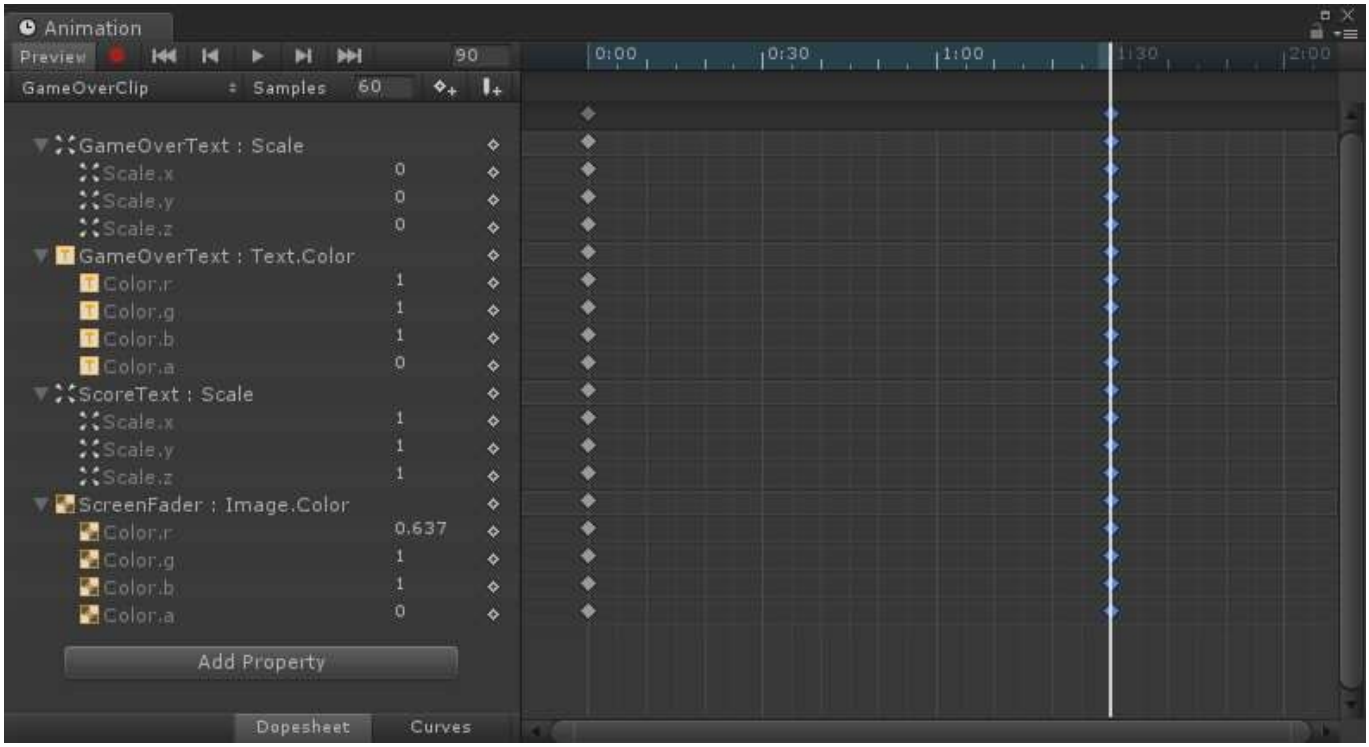
各要素に以下のとおりアニメーションを設定する。

・GameOverText > Text > Color	120 フレーム：アルファ値を 1
・GameOverText > RectTransform > Scale	0 フレーム：0，0，0
	110 フレーム：1.2，1.2，1.2
	120 フレーム：1，1，1
・ScreenFader > Image > Color	120 フレーム：でアルファ値を 1
・ScoreText > RectTransform > Scale	120 フレーム：0.8，0.8，0.8

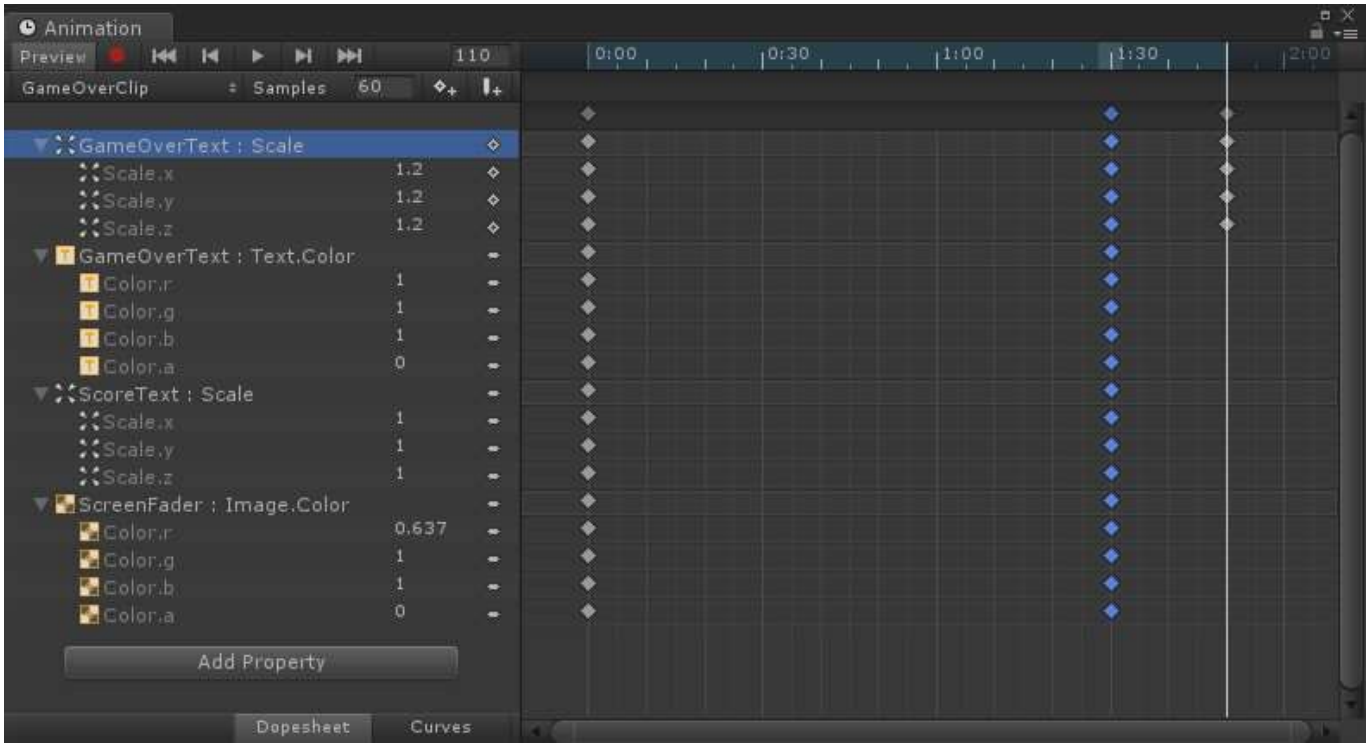
0 フレームのキーフレームをクリックし，GameOverText の Scale.x，y，z を 0 に設定する。



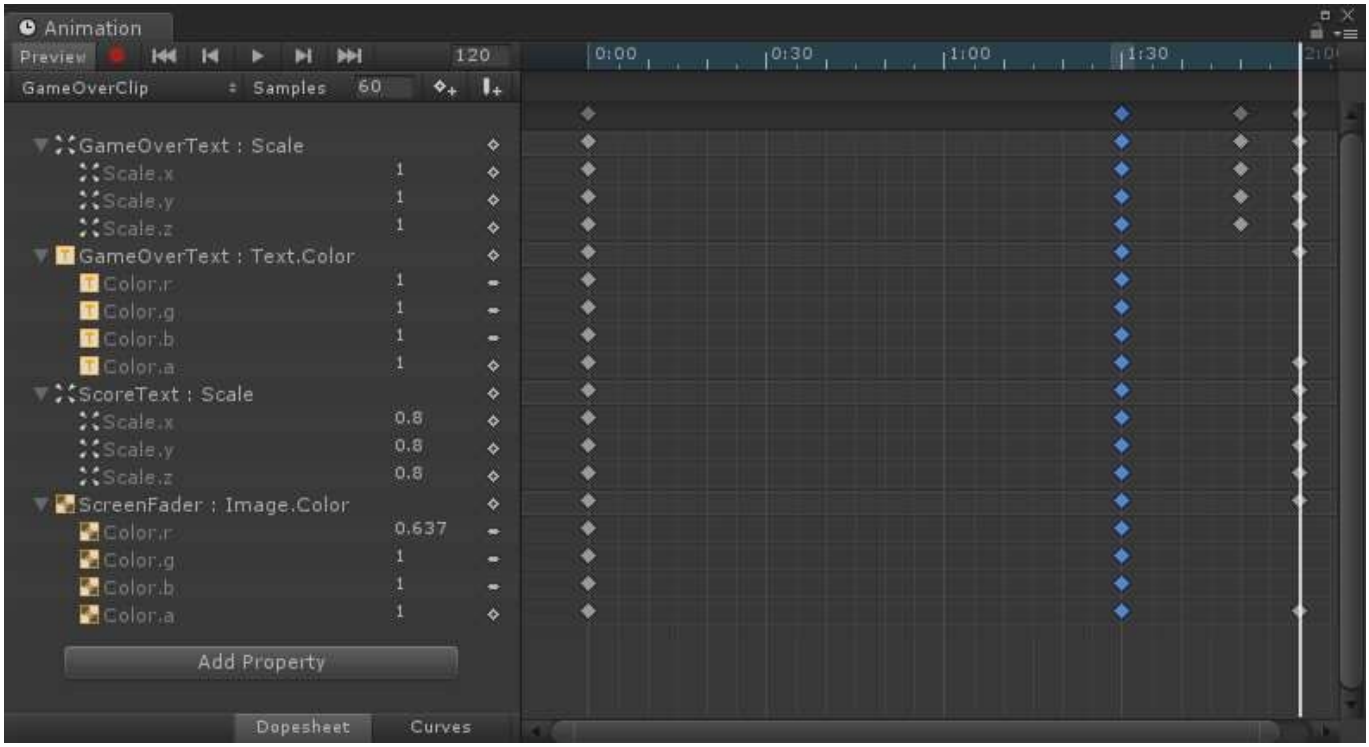
90（1:30）フレームの位置をクリックし，Add Keyframe ボタン  をクリックしキーフレームを作成する。



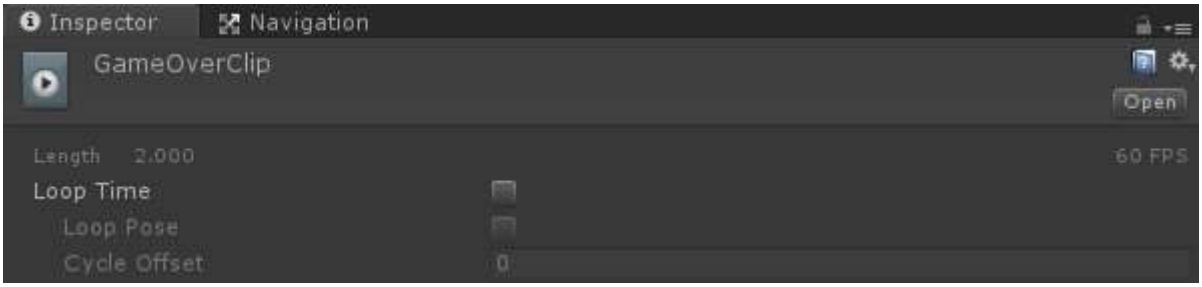
110 フレームをクリックし，GameOverText の Scale.x，y，z を 1.2 に設定する。自動的にキーフレームが設定される。



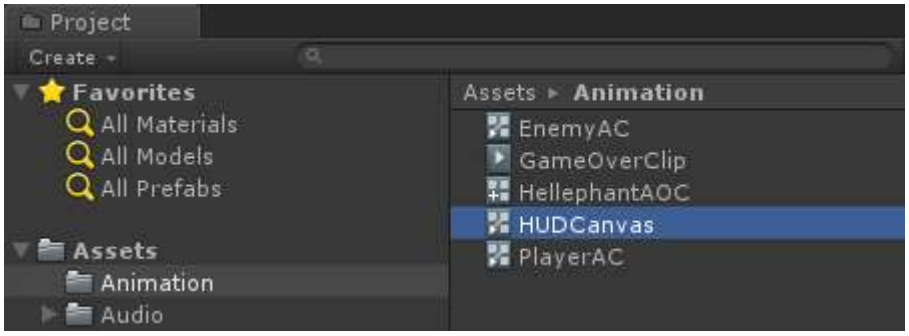
120（2:00）フレームをクリックし，GameOverText の Color.a を 1，GameOverText の Scale.x，y，z を 1，ScreenFader の Color.a を 1，Score.Text の Scale.x，y，z を 0.8 に設定する。



GameOverClip をクリックし Inspector の Loop Time のチェックを外す。

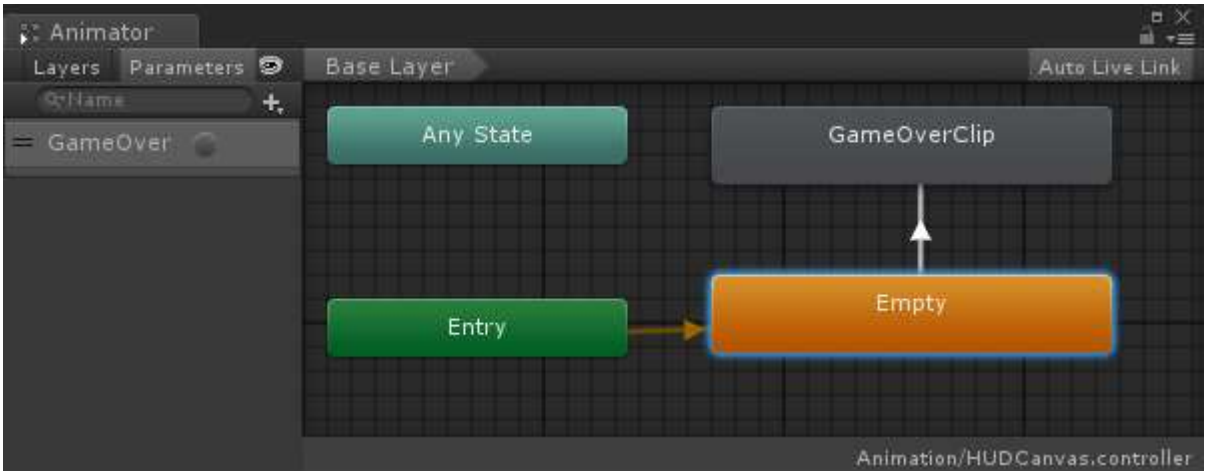


Purject > Assets > Animation フォルダの HUDCanvas をダブルクリックして，Animator ウインドウを開く。

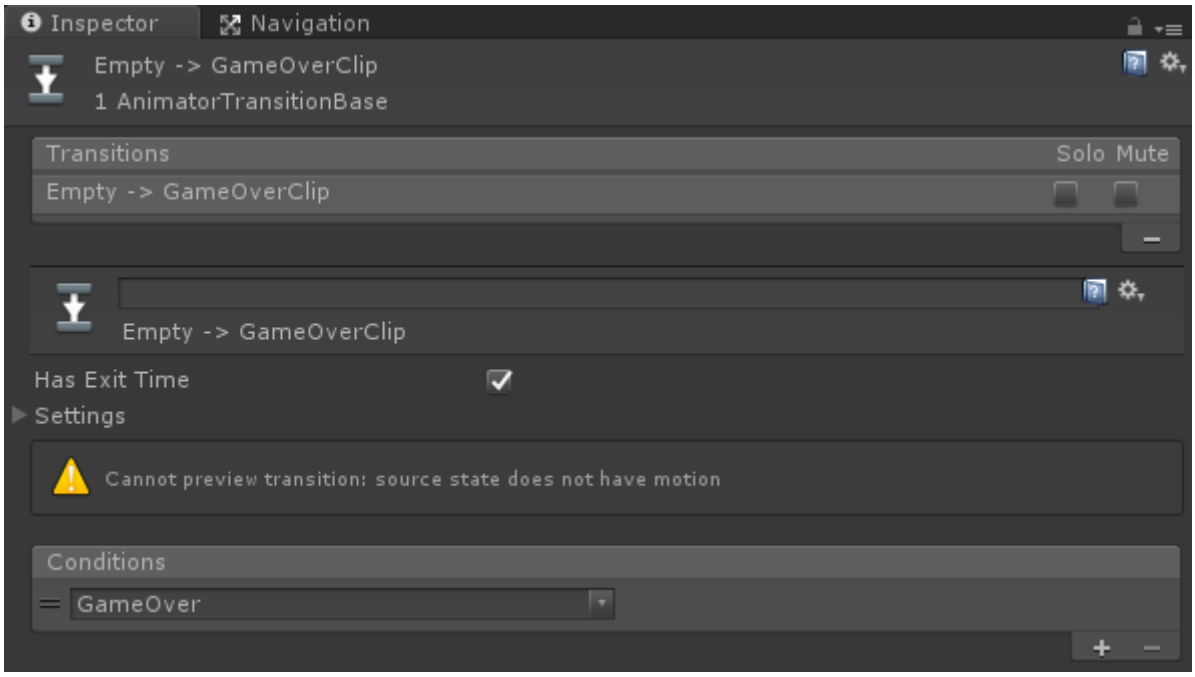


右クリックし Create State > Empty をクリックし，空の State を作成し，Empty にリネームする。Empty を右クリックし Set as Layer Default State を選択し，Empty をデフォルトの状態にする。

Empty を右クリックし，Make Transition を選択し，Empty から GameOverClip に Transition を追加する。

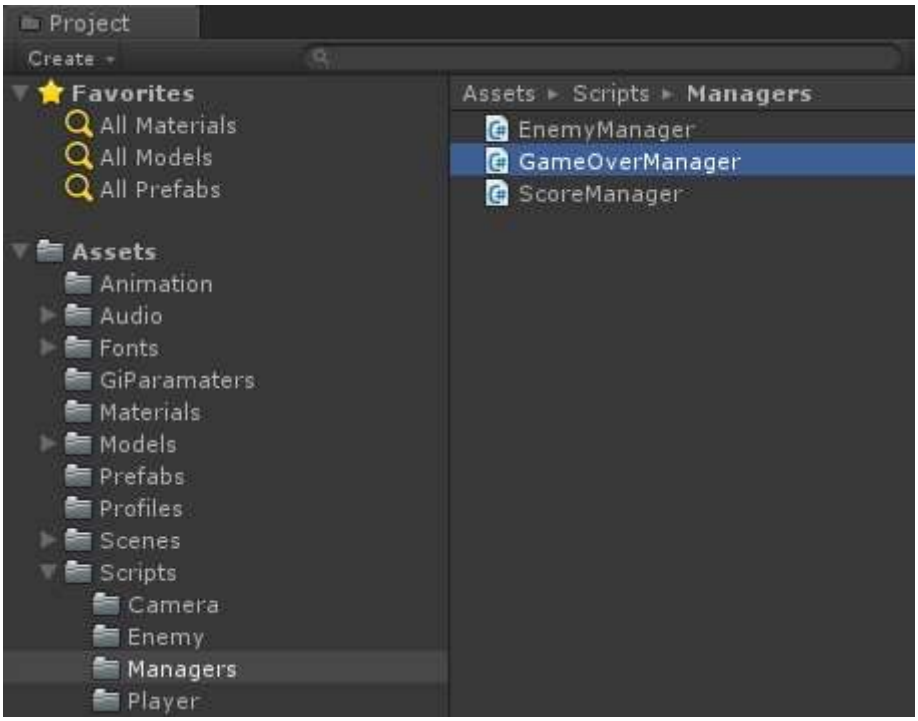


Trigger 型のパラメータ GameOver を作成し，Empty から GameOverClip への Transition の条件として追加する。



３．ゲームオーバースクリプトの設定

Project > Assets > Scripts > Managers フォルダの GameOverManager スクリプトを HUDCanvas にドラッグする。



GameOverManager スクリプトをダブルクリックして開き，内容を確認する。

```
using UnityEngine;

public class GameOverManager : MonoBehaviour
{
    public PlayerHealth playerHealth;    // Player Health スクリプト

    Animator anim;                       // Animator Controller

    void Awake()
    {
        // AnimatorController コンポーネントへの参照取得
        anim = GetComponent<Animator>();
    }

    void Update()
    {
        // プレイヤー体力 0 以下か判定
        if (playerHealth.currentHealth <= 0)
        {
            // GameOver トリガーセット
            anim.SetTrigger("GameOver");
        }
    }
```

Hierarchy の Player を GameOverManager コンポーネントの Player Health にドラッグする。

シーンを保存し，実行して確認する。