

Unity 2D 講習

戸高

0. Unity 2D

今回からは Unity で 2D のゲームを作ってもらいます。とはいえ基本は 3D のゲームと変わりません。シーンやオブジェクトの考え方は変わりませんし、プレハブ化も Instantiate もできますし、AddForce すれば動きます。しかし、一部のコンポーネントや命令が変わるほか、描画の方法が「スプライト」というものになります。これからの講習は、そういったことについて教えていきます。

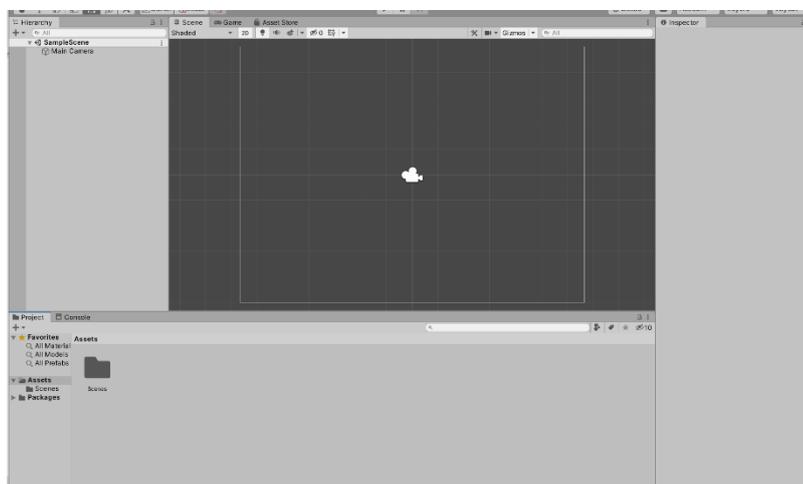
1. プロジェクトの作成

2D のプロジェクトを作成します。

- ① Unity Hub で新規作成ボタンを押します。
- ② テンプレートを「2D」にして、適当な名前をつけて、プロジェクトを作成します。



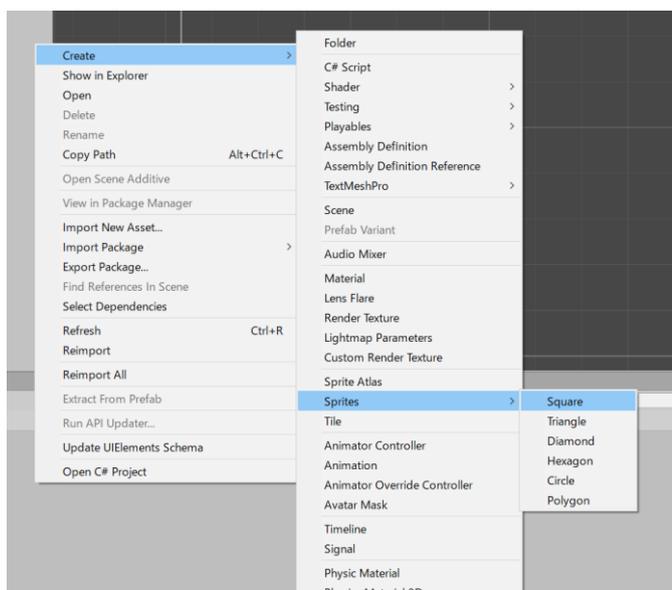
- ③ しばらく待ちます。
- ④ 2D のプロジェクトができます。3D とはすこし違うのが分かります。



2. スプライト

スプライトは、画像を張り付けるための2Dのオブジェクトです。まずは作ってみましょう。

- ① プロジェクトビュー上で右クリックし、**Create>Sprites>Square** を選択します。名前の入力を求められますが、そのままにしてください。



すると白いものができます。これがスプライトの元となる画像です。

- ② 今の画像をシーンにドラッグアンドドロップします。すると白い四角がシーンにあらわれます。これで、シーンに白い四角の画像が張り付けられたオブジェクト＝スプライトができました。

3. 外部ファイルの取り込み

今まで、Unity 内部にデフォルトで存在するデータを使ってゲームを作ってきました。これも立派なゲームですが、プレイヤーがただの球体とか四角だと味気ないです。そこで、外部からキャラの絵のデータを取り込み、かわいいキャラを動かせるようにしましょう。

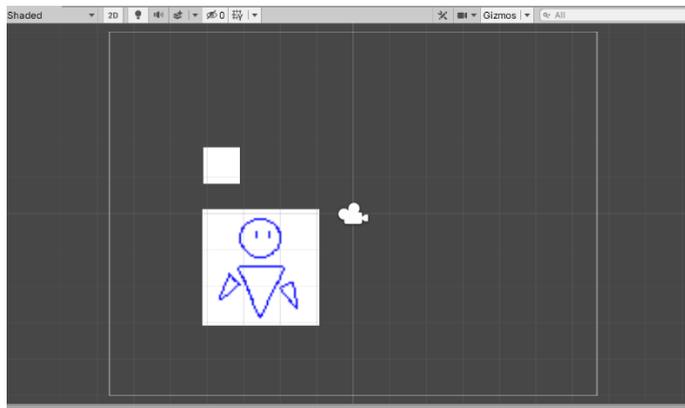
- ① 画像を用意します。Windows 標準のペイントっていうソフトで書けるので適当に描いてください（タスクバーの検索窓にペイントって打ったらでてきます。使い方はシンプルなのでここでは書きません。わかんなかったら聞いて）。描くのがだるいって人は僕に言ってくれたら一応素材あげます。また、自分の PC でいいソフト持ってるぜって人はぜひそっちを使ってください。
- ② 自分で描いた人はわかりやすい場所にその絵を保存してください。このとき、ファイル名に日本語を使うとバグる可能性があるのでアルファベットでにしてください。
- ③ 画像を調達する方法を書いておきます。飛ばしてください。

方法は主に4つあります。

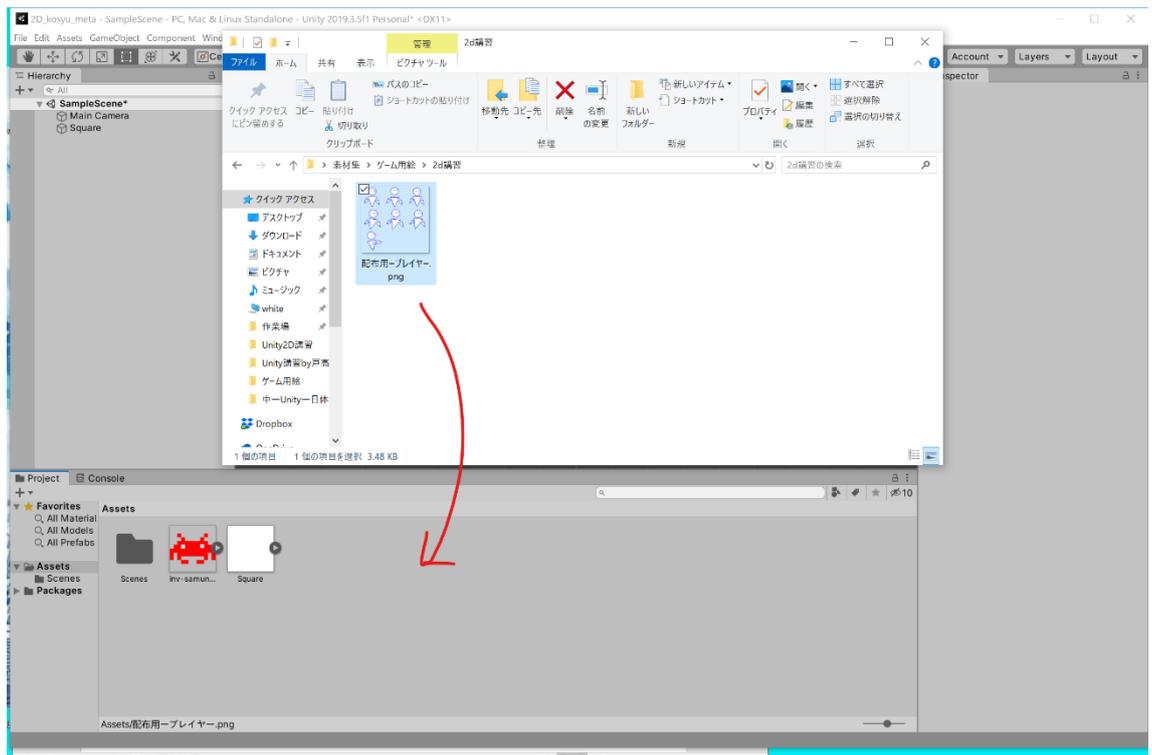
- 1：ネットでフリー素材を検索する（著作権には注意）。
- 2：自分で描く。
- 3：人に頼む。
- 4：Unity の AssetStore を使う。（素材を販売している場所。有料のものが多
い）。

飛ばしてください 終わりです。

画像を描いたら、背景を透過させます。「手軽に透明 png」なるソフトをあげるの
で、それを使ってください（ここまで来たら僕を呼んでください）。なぜ透過させ
るかという、そうしないと以下のように白い四角ができてしまうからです。ペ
イント以外のツールで自分で描いた人はそういう設定をして保存しましょう。



- ④ 画像を保存したフォルダを PC から開き、画像をプロジェクトビューにドラッグ
アンドドロップします。

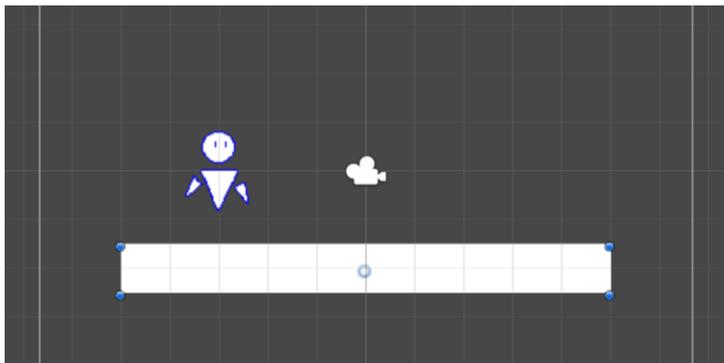


⑤ その画像をプロジェクトビューからシーンにドラッグアンドドロップします。

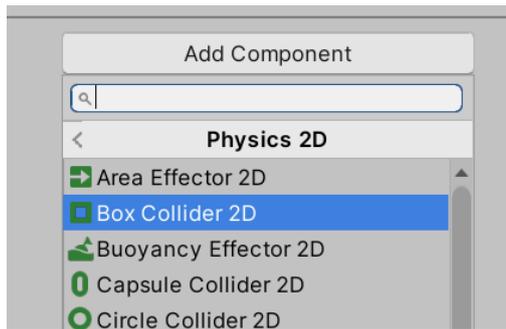
4. 準備をする

今回作るゲームは、初回の一日体験で作ったような横2Dアクションにします。まずはフィールドを作りましょう。

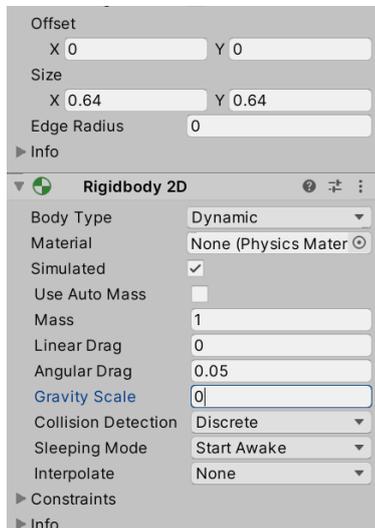
① スクリーンショットのように白い四角とプレイヤーを配置します。プレイヤーの座標は (-3,0)、大きさは (3,3)、四角は座標 (0,-2)、大きさ (10,1) です。



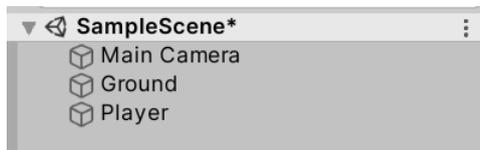
② 当たり判定をつけます。四角とプレイヤーの両方に、Add Component から Physics2D→BoxCollider2D を選択して当たり判定をつけてください。2D がついていることに注意してください。



- ③ プレイヤーに Add Component→Physics2D→Rigidbody2D を選択して Rigidbody2D コンポーネントを追加してください。こいつは Rigidbody の 2D 版です。そのうえで Gravity Scale を 0 にしてください。これは重力を働かせないということです。重力は後で自前で実装します。



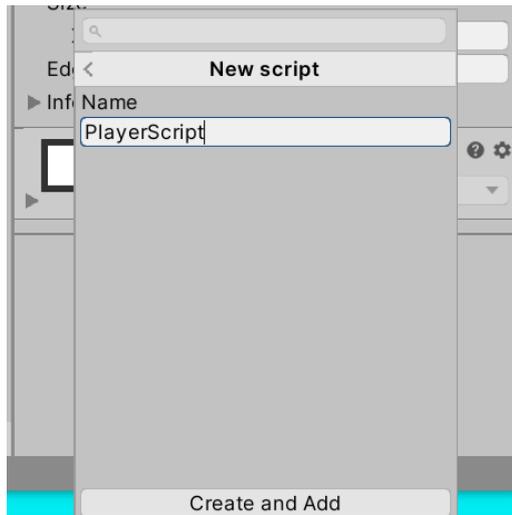
- ④ オブジェクトの名前を変更しましょう。Square は Ground に、取り込んだ画像は Player に名前を変えてください。



5. プレイヤーを動かす

プレイヤーを動かしましょう。

- ① Player に Add Component→New Script で、PlayerScript という名前のスクリプトを追加してください。



② 以下のように打ち込んでください。

```
Assets > PlayerScript2.cs > ...
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  0 references
6  public class PlayerScript2 : MonoBehaviour
7  {
8      2 references
9      private Rigidbody2D rigid;
10     2 references
11     private float XForce;
12     1 reference
13     public float Speed;
14     // Start is called before the first frame update
15     0 references
16     void Start()
17     {
18         rigid = GetComponent<Rigidbody2D>();
19     }
20     // Update is called once per frame
21     0 references
22     void Update()
23     {
24         XForce = Speed * Input.GetAxis("Horizontal");
25         rigid.velocity = new Vector2(XForce,0);
26     }
27 }
```

GetComponent<>(): 移動の際に使っていたプログラムの一部ですが、これの本
当の意味は「コンポーネントを取得する」です。<>の中に取得したいコンポーネ
ントの名前を入れると、そのコンポーネントの情報を取得できます。() を書き忘
れないようにしましょう。

Input.GetAxis("Horizontal")：右矢印キーが押されていたら 1、左矢印キーが押されていたら -1、どちらも押されていない場合は 0 を返す命令。滑らかに数値が変化するので、移動の「だんだん速くなる」感が出ます。

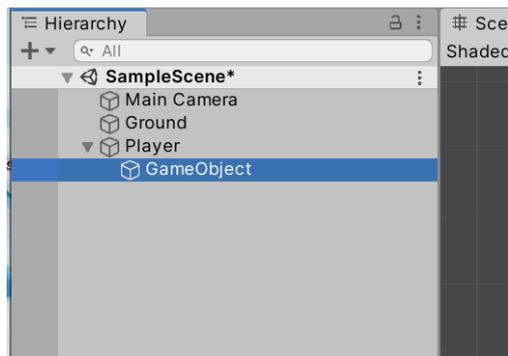
rigid.velocity = new Vector2(XForce,0)：rigidbody2D コンポーネントが持つ情報の中の、速度 (velocity) を (XForce,0) にする、という意味。Vector2 は Vector3 の仲間で、2つの数値情報を持ちます。

- ③ インスペクターから Speed を適当に (5 くらい) 編集して実行すると、横移動ができるようになります。

6. 着地判定

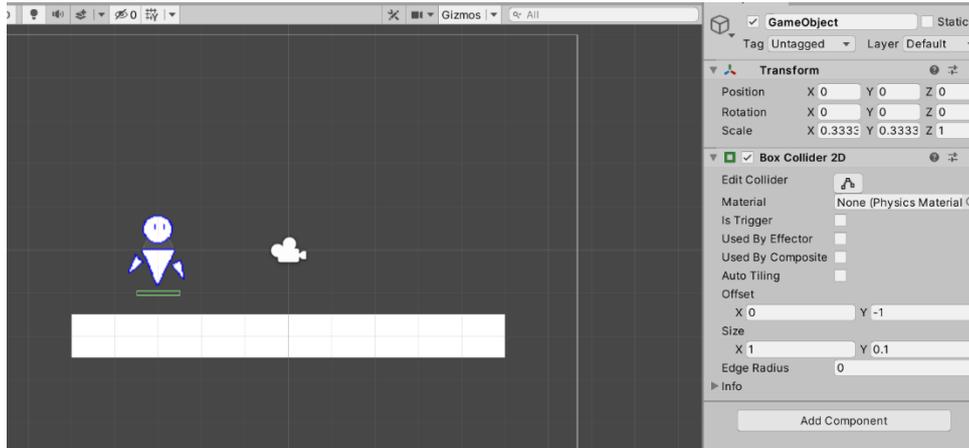
次は着地ができるようにします。このためには地面にあたっているかを判断することが必要です。いくつか方法がありますが、今回は接地判断用のオブジェクトを作ってそれに判断をさせます。

- ① ヒエラルキーで右クリック→Create Empty をクリックします。すると GameObject という名前のオブジェクトができます。こいつには Transform 以外のコンポーネントがついていません (ので形もありませんし見えません)。このオブジェクトはゲームの管理によく使います。
- ② 子オブジェクトというものを教えます。子オブジェクトは親となるオブジェクトとの位置関係が固定されます。つまり、親が動くと子もついてくるということです。ほかにも関連づいたオブジェクトを親子にすることで、見やすく管理しやすくなるというメリットがあります。
- ③ ヒエラルキー上で、GameObject を Player にドラッグアンドドロップしてください。すると以下のようになると思います。

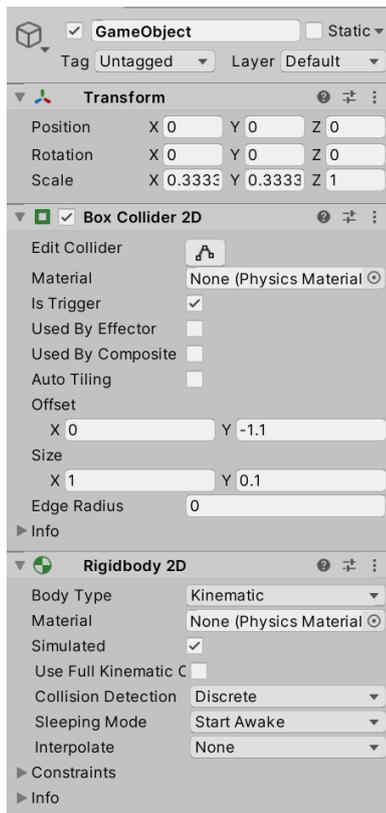


これは GameObject が Player の子オブジェクトとなったことを表します。

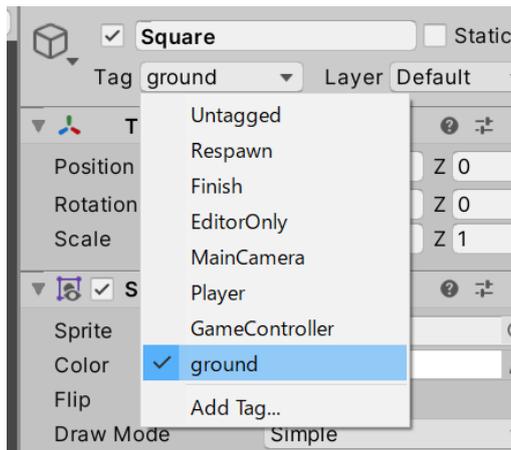
- ④ Add Component から Box Collider 2D を追加してください。そして、Player の下の適切な位置と大きさに Collider を移動変形してください。Offset と Size をいじればできます。このとき・Player の当たり判定と当たらないような位置・Player の当たり判定の横の長さと同じ横の長さにしてください。また、Is Trigger にチェックを入れてください。



ついでに Add Component→Physics2D→Rigidbody2D で Rigidbody 2D コンポーネントを追加してください。追加したら BodyType を Kinematic にしてください。これを付けると重力などの影響を受けなくなります。



- ⑤ Ground の Tag を変更して ground にします。Add Tag から Tag を追加する方法は 3D 講習資料を見てください。



- ⑥ Add Component→New Script から LandScript という名前でスクリプトを付けてください。中身は以下のように。

```
5 public class LandScript : MonoBehaviour
6 {
7     // Start is called before the first frame update
8     void Start()
9     {
10    }
11 }
12
13 // Update is called once per frame
14 void Update()
15 {
16 }
17
18
19 public bool IsLand;
20 void OnTriggerEnter2D(Collider2D Hit)
21 {
22     if(Hit.CompareTag("ground")){
23         IsLand = true;
24     }
25 }
26 void OnTriggerExit2D(Collider2D Hit)
27 {
28     if(Hit.CompareTag("ground")){
29         IsLand = false;
30     }
31 }
32 }
33
```

Bool 型の変数 IsLand を定義し、足場にタグが ground の当たり判定が入ったらそれを true に、出たらそれを false にするという意味になります。

OnTriggerEnter2D(Collider2D Hit) : OnTriggerEnter の 2D 版。

Hit.CompareTag("ground") : 当たったもののタグが ground なら、の意味。

OnTriggerExit2D(Collider2D Hit) : OnTriggerEnter2D の仲間。Enter の方は当たり判定に何かが入った時に処理が行われますが、こっちは当たり判定から何かが出た時に処理されます。ほかに Stay っていうのもあるけど今は無視。

7. 着地とジャンプ

着地しているかの判断ができるようになったので、着地していなければ重力を働かせ、していれば下に動かさないというプログラムを書いていきます。この機能は Rigidbody にありますが、物理演算は予期せぬ動作の温床となる他、自由度が低いので、演算機能を切って手動で重力を実装します。

① PlayerScript を書きかえます。

```
5 public class PlayerScript2 : MonoBehaviour
6 {
7     2 references
8     private Rigidbody2D rigid;
9     2 references
10    private LandScript landSc;
11    2 references
12    private float XForce;
13    3 references
14    private float YForce;
15    1 reference
16    public float gravity;
17    1 reference
18    public float Speed;
19    // Start is called before the first frame update
20    0 references
21    void Start()
22    {
23        rigid = GetComponent<Rigidbody2D>();
24        landSc = GetComponentInChildren<LandScript>();
25    }
26
27    // Update is called once per frame
28    0 references
29    void Update()
30    {
31        XForce = Speed * Input.GetAxis("Horizontal");
32        if(landSc.IsLand){
33            YForce = 0;
34        }else{
35            YForce -= gravity;
36        }
37        rigid.velocity = new Vector2(XForce,YForce);
38    }
39 }
```

private LandScript : LandScript を変数の型として扱っています。Unity ではすべてのコンポーネント（の名前）を型にできるので、自作スクリプトの名前も型にできます。

GetComponentInChildren<>() : 子オブジェクトについているコンポーネントを

取得する命令です。

LdScript.IsLand : LdScript の中には LandScript の情報が入っています。そこにはさらに IsLand という bool 型の変数の情報も入っています。ここではそれを取り出しています。

if(～){A}else{B} : ～だった時は A、～ではなかった時は B の処理を実行します。

- ② ジャンプできるようにします。さらに PlayerScript を編集します。

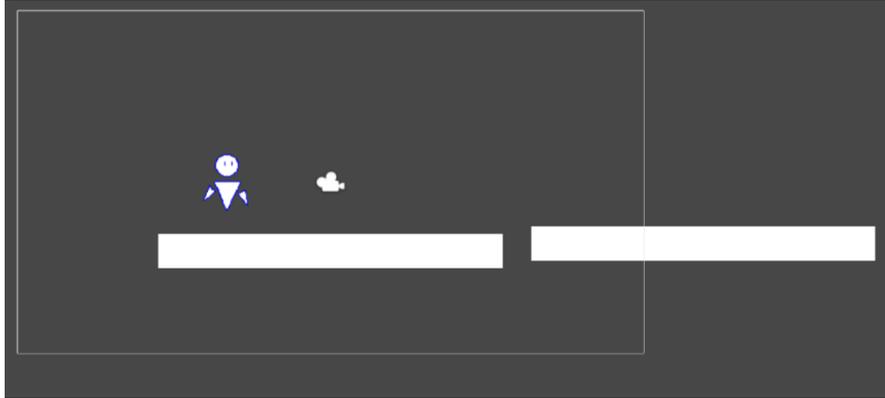
```
9     private float XForce;
      4 references
10    private float YForce;
      1 reference
11    public float gravity;
      1 reference
12    public float jumpForce;
      1 reference
13    public float Speed;
14    // Start is called before the first frame update
      0 references
15    void Start()
16    {
17        rigid = GetComponent<Rigidbody2D>();
18        landSc = GetComponentInChildren<LandScript>();
19    }
20
21    // Update is called once per frame
      0 references
22    void Update()
23    {
24        XForce = Speed * Input.GetAxis("Horizontal");
25        if(landSc.IsLand){
26            if(Input.GetKey(KeyCode.UpArrow)){
27                YForce = jumpForce;
28            }else{
29                YForce = 0;
30            }
31        }else{
32            YForce -= gravity;
33        }
34        rigid.velocity = new Vector2(XForce,YForce);
35    }
36 }
37
```

- ③ インスペクターから jumpForce を 20、Gravity を 1 くらいにしてください。すると ↑ キーでジャンプができます。

8. カメラ制御

基本的な移動ができるようになったので、ステージの作成に移ります。

- ① 足場となっているオブジェクト Ground をコピーアンドペーストして、以下のよ
うな位置に移動させます。適当で結構です。



シーンビューでの視点の移動はマウスホイール押し込み+ドラッグまたは矢印キーでできます。拡大縮小はマウスホイールを回せばできます。

- ② このまま実行すると、プレイヤーが画面外へ出られてしまいます。これを回避するために、視点を動かしましょう。
- ③ ゲーム実行中の視点は Main Camera（正確にいうと Camera コンポーネントのついたオブジェクト）の位置です。Main Camera に CameraScript という名前でスクリプトを追加しましょう。

```
5 public class CameraScript : MonoBehaviour
6
7
8     2 references
9     private GameObject Player;
10    5 references
11    private float difX;
12    4 references
13    private Transform trans;
14    // Start is called before the first frame update
15    0 references
16    void Start()
17    {
18        Player = GameObject.FindGameObjectWithTag("Player");
19        trans = GetComponent<Transform>();
20    }
21
22    // Update is called once per frame
23    0 references
24    void Update()
25    {
26        difX = Player.transform.position.x - trans.position.x;
27
28        if(difX > 3){
29            trans.Translate(difX - 3,0,0);
30        }
31        if(difX < -3){
32            trans.Translate(difX + 3,0,0);
33        }
34    }
35 }
```

GameObject.FindGameObjectWithTag("Player") : Player タグがついたオブジェクトを一つ取得する命令。FindObjectsWithTag("~)との違いは、一つ取得

するか複数取得（して配列で返す）かです。

自身の X 座標が Player の座標と 3 以上離れたら Player についていくように移動するスクリプトです。

- ④ 実行すると常にプレイヤーが画面に映り続けます。

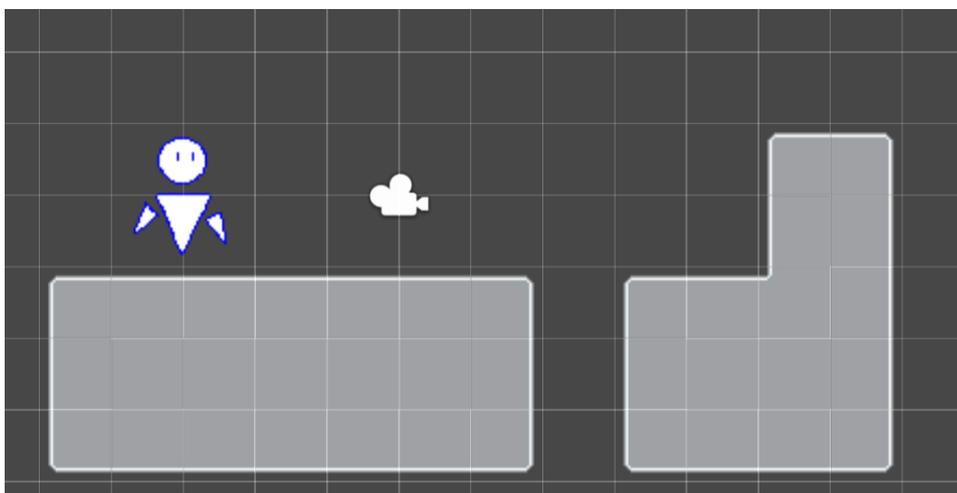
9. スプライトの分割

カメラの準備も終わったので本格的にステージを作っていきます。

- ① 8.①みたいにコピペやなんやらで足場とか壁とかギミックを作るのもいいですが、もっといい方法があるのでそれを教えます。まずは最終的にやりたいことを見せます。



こういう絵から

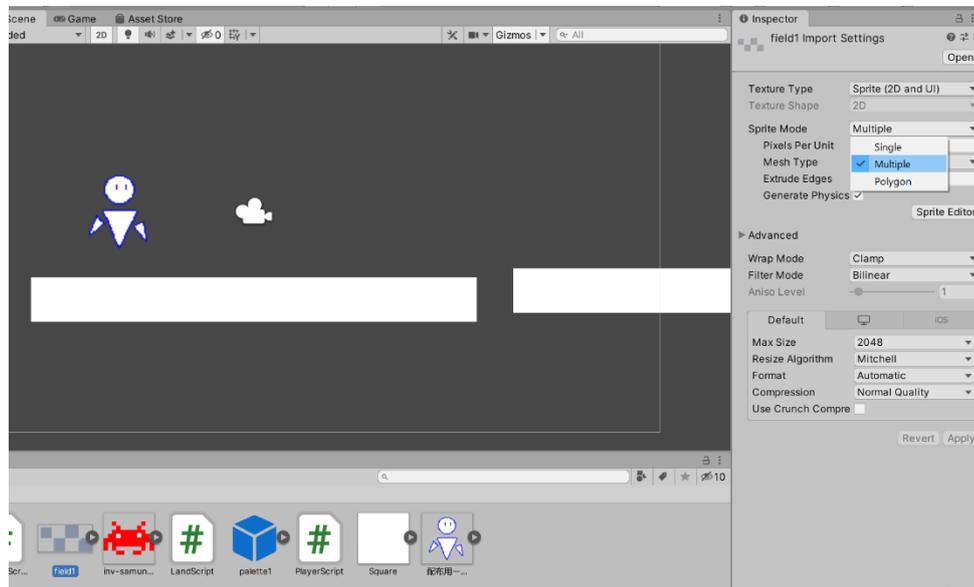


こんなステージを作ります。

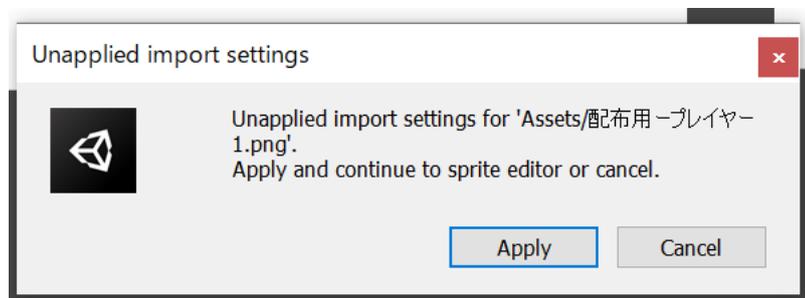
絵をパーツとして、いくつかのパーツを組み合わせてステージをつくるつもりです。

- ② まずは絵を描きましょう。↑のように、・サイズと形が同じ・つなげたときにきれいに繋がる絵を・複数枚一つの絵に・それぞれが接しないように・きれいに配置して描いてください。（うまく説明できてる気がしない、わからなかったら呼んで）描くのがだるかったら呼んでくれたらデータあげます。
- ③ 描いた絵を取り込みます。

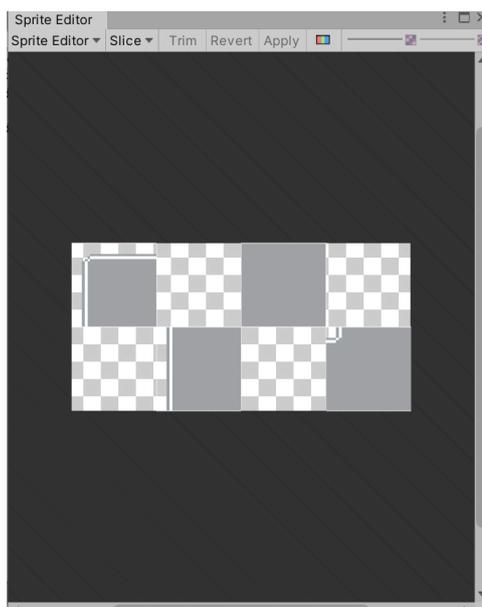
- ④ 取り込んだ絵をクリックして、インスペクターから Sprite Mode を Single から Multiple にします。こうすると一枚の絵を複数のスプライトとして扱うことができます。



- ⑤ Sprite Mode のちょっと下にある Sprite Editor ボタンを押してください。この時下ののようなダイアログが出たら Apply を押してください。

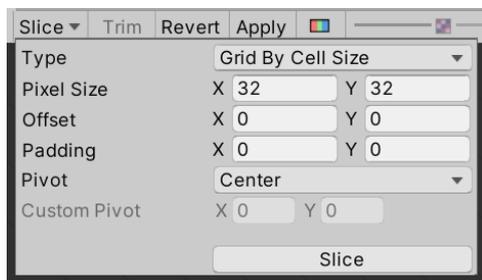


すると下ののようなウィンドウが出ます。



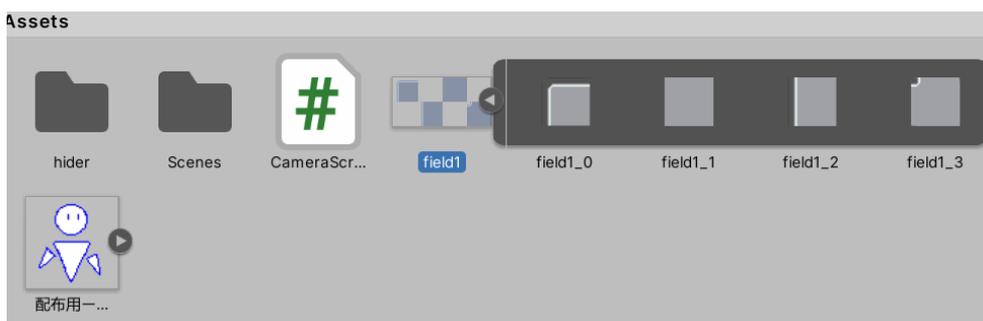
この画面で絵の分割を行います。

- ⑥ 絵がきれいに配置されている人は、右上の Slice ボタンから Type に Automatic を選択、または Slice by Cell Size を選択したうえで一パーツの大きさを指定して、最後に Slice ボタンを押してください。



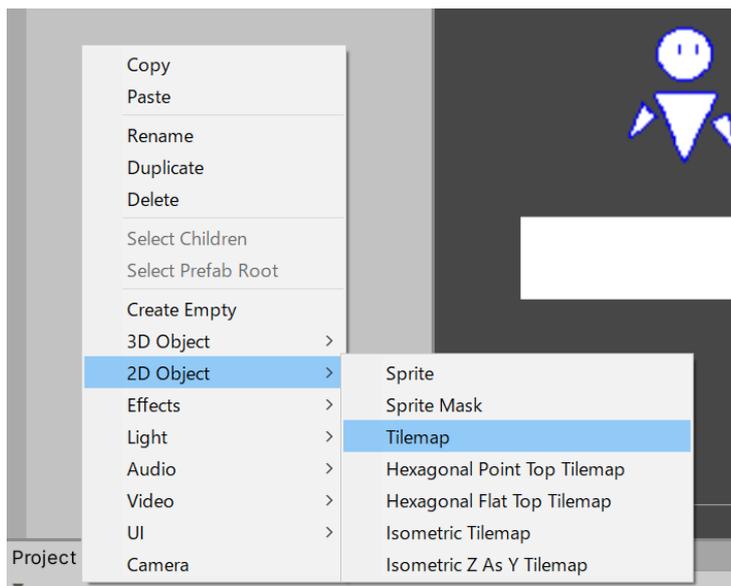
そうでない人は、絵の 1 パーツにしたい部分の上でドラッグをすることを繰り返してください。

- ⑦ その画面を閉じてください。すると先ほどと同じようなダイアログが出るので、Apply を押してください。
- ⑧ もとの画面に戻って絵をクリックすると、分割されたデータが出てきます。これらのデータは一つ一つをスプライトとして扱えます。

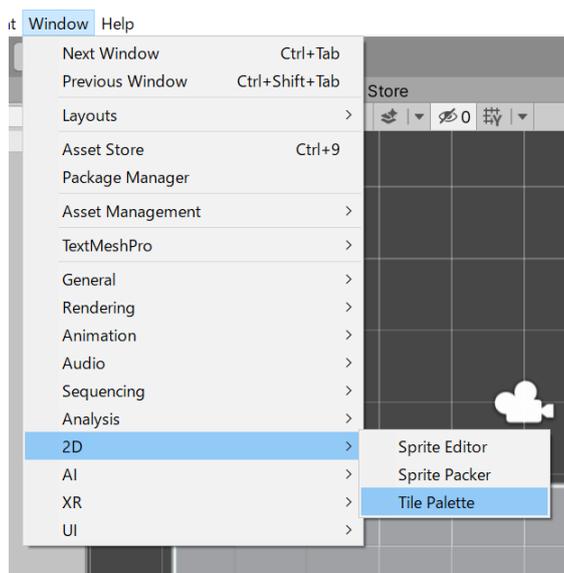


10. Tilemap の準備

- ① 用意したスプライトを使っていよいよステージを作ります。これから使うのはタイルマップという機能です。この機能は、スプライトを「タイル」とみなし、シーン上に敷き詰めることでステージの見た目をつくる機能です。
- ② まずはヒエラルキー上に 2D Object→Tilemap からオブジェクトをつくります。Grid という名前です。

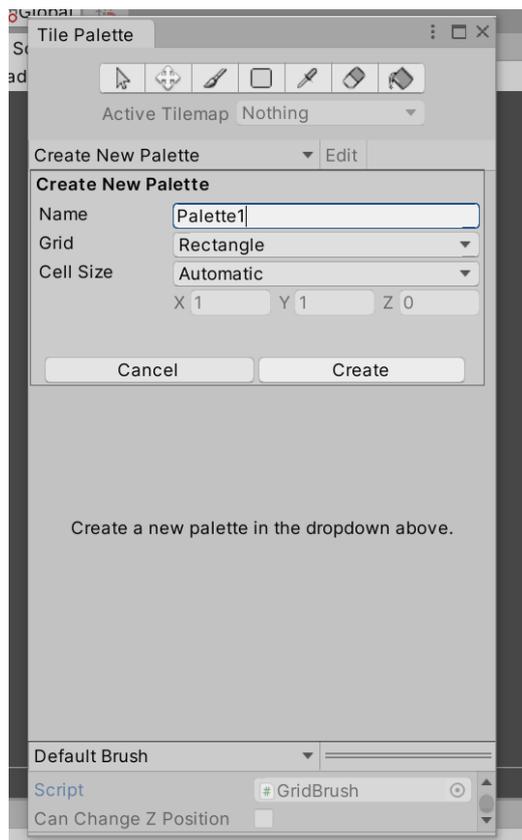


- ③ これからの作業に邪魔なので、Ground は消してください。
- ④ 画面上の Window→2D→Tile Palette を押してください。

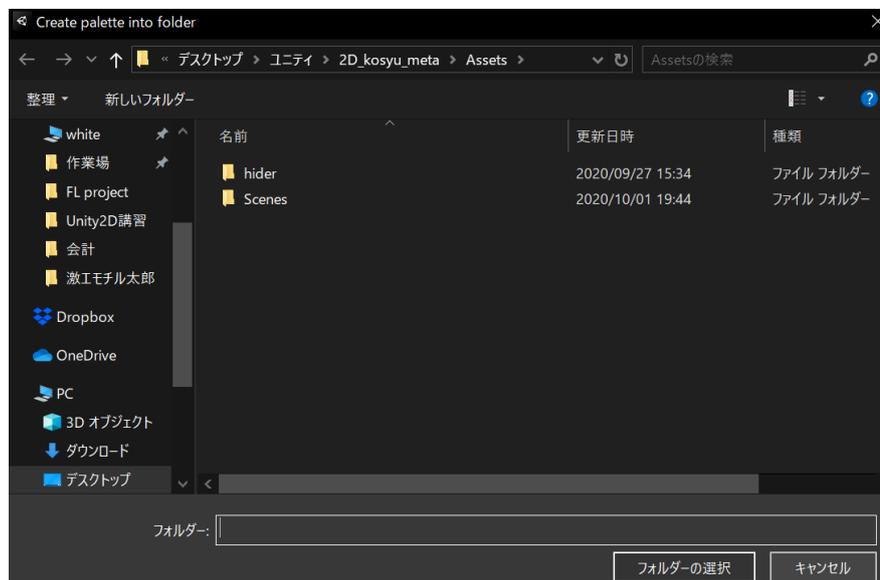


すると新しいウィンドウが出てきます。

- ⑤ Create new palette を選択して名前を Palette1 にして create を押してください。

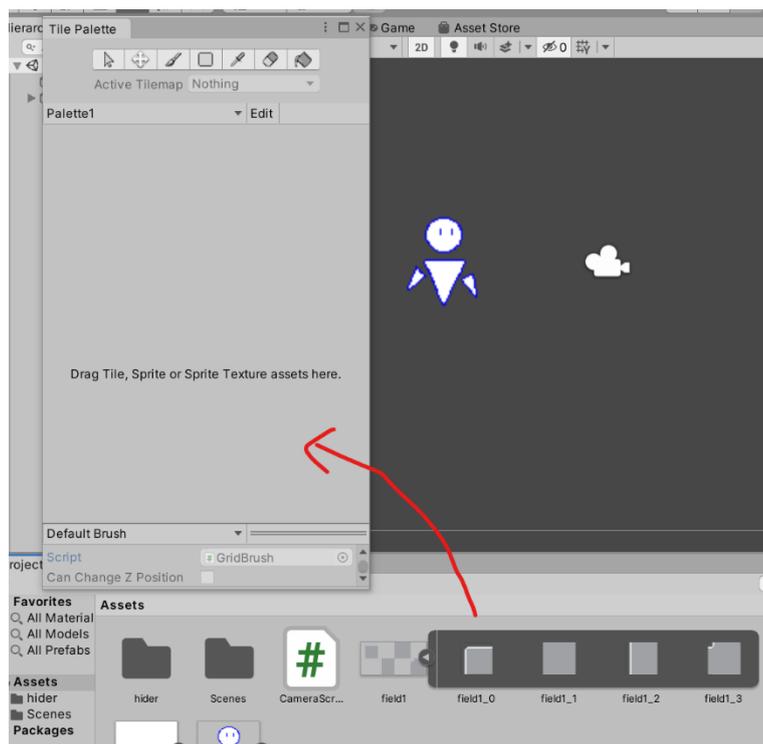


セーブ画面が出たらフォルダーの選択ボタンを押してください。

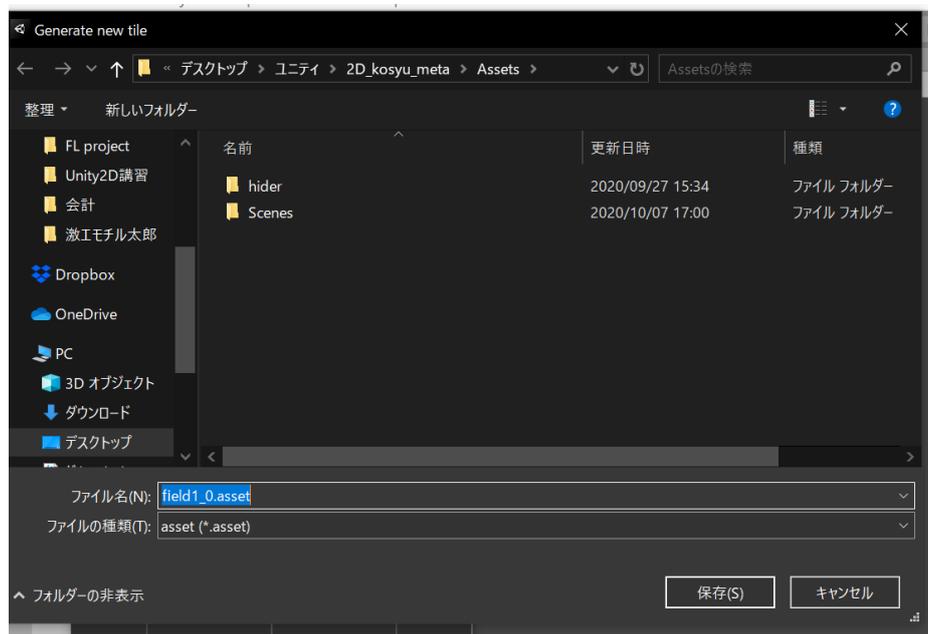


このウィンドウをタイルパレットといい、ここにタイルを登録します。

- ⑥ タイルパレットに、先ほど分割したスプライトをドラッグアンドドロップします。

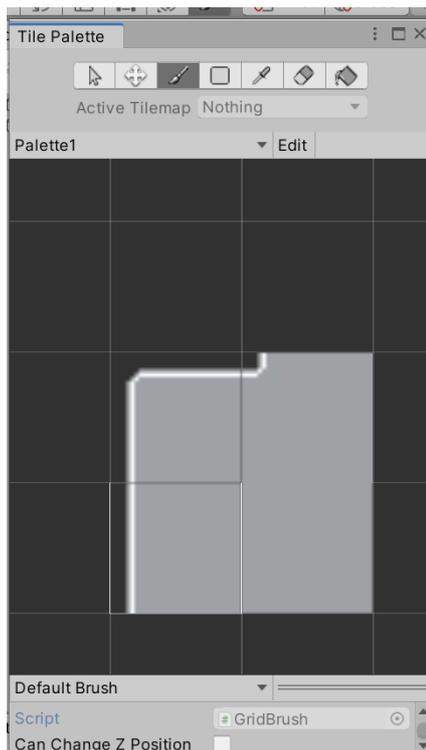


すると保存画面が出るので、そのまま保存してください。



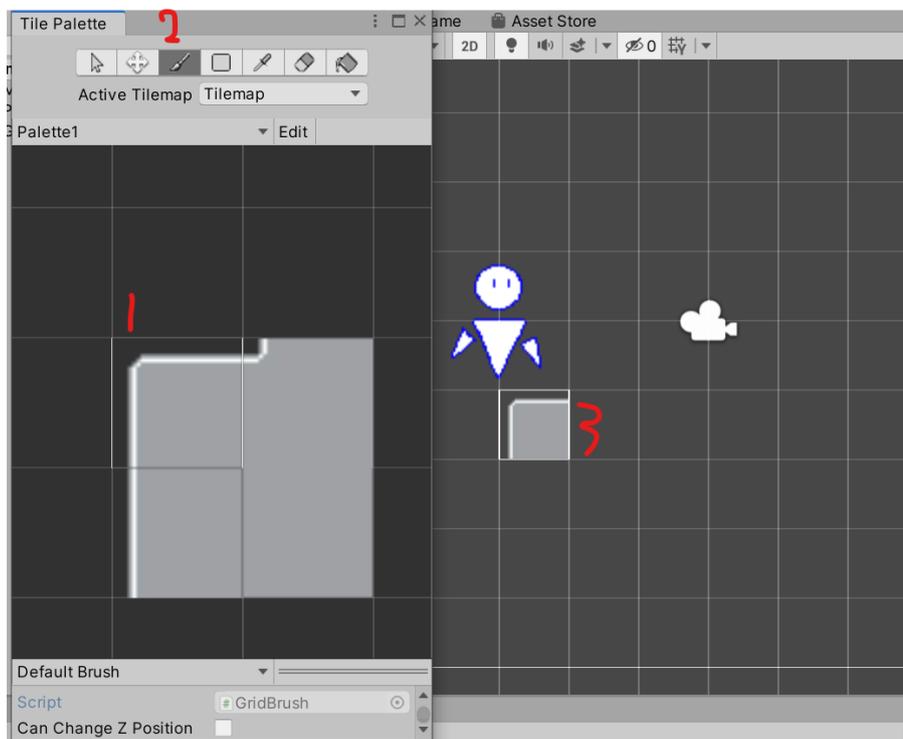
これで、スプライトをタイルという形にして、パレットに登録したことになります。

- ⑦ ほかの分割したスプライトについても同様にドラッグアンドドロップして保存してください。

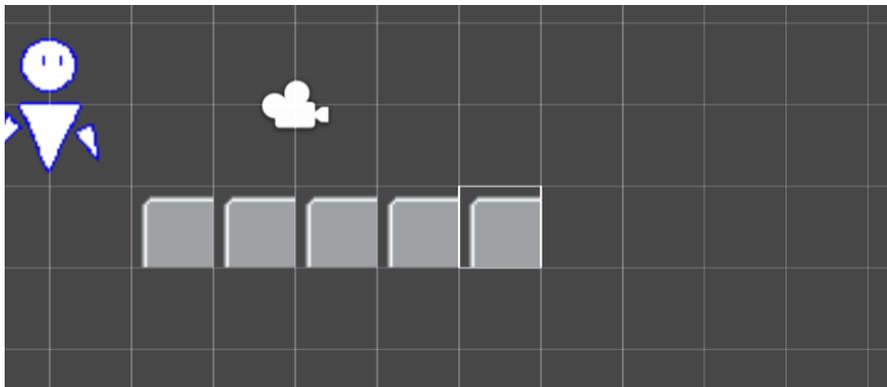


1 1. Tilemap

- ① パレットの用意ができたので、ステージを作ります。
 - ・パレット上から一つタイルを選んでクリックし、
 - ・パレット上部のペンのマークが黒くなっていることを確認し、
 - ・シーン上にカーソルをもって行ってください。

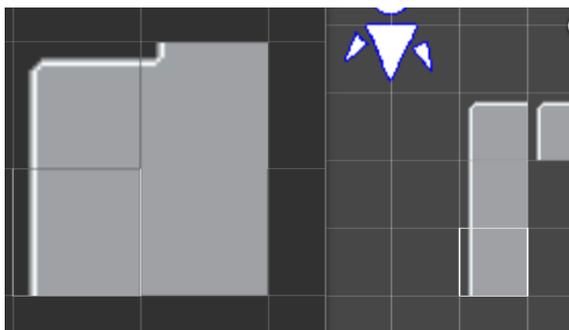


- ② クリックするとその場にタイルが置かれます。ドラッグすると通った範囲をすべてそのタイルで塗ることが可能です。

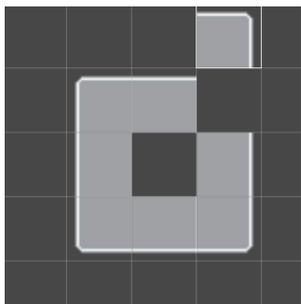


このように、ペイントツール感覚でステージを作ることができます。
以下、使い方を説明していきます。(大きさが違う人は⑦から見て)

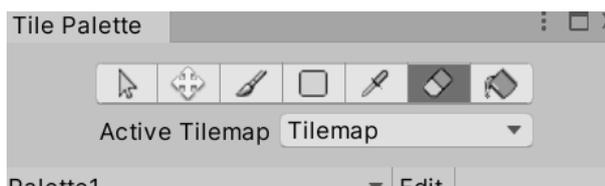
- ③ 塗るタイルの種類を変えたいときはパレットから別のタイルを選択します。



- ④ タイルを回転させたいときは、『⌂』ボタンか『⌋』ボタンを押してください。



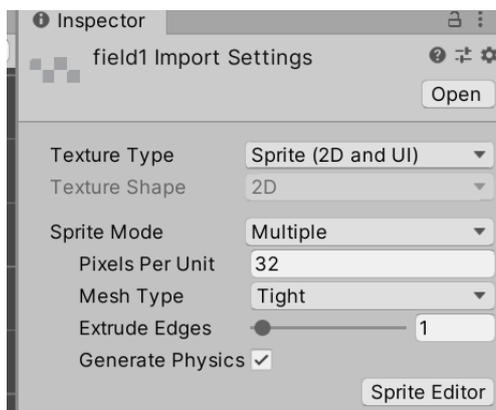
- ⑤ タイルを消したいときはパレット上部の消しゴムのボタンを押してください。



- ⑥ では自由にステージをつくってみてください。

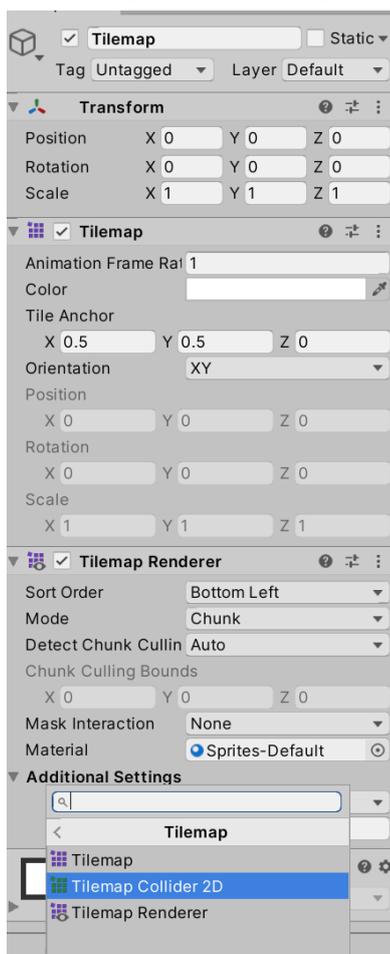
- ⑦ サイズが合わないという人は、画像データの Pixels per unit を変更してみてください。

さい。



1 2. 当たり判定の追加

- ① Tilemap で作ったステージには当たり判定がないので、当たり判定を作ります。Grid の子オブジェクトに Tilemap という名前のオブジェクトがあるので、それに Add Component>Tilemap>Tilemap Collider 2D で当たり判定を追加します。



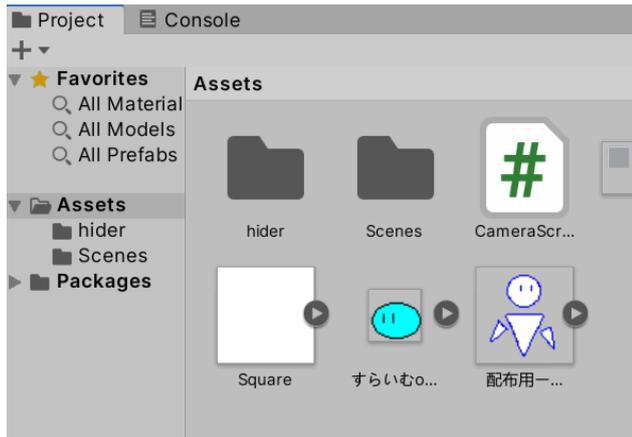
- ② Tag を ground にすれば当たり判定の完成です。

1 3. 敵を作る

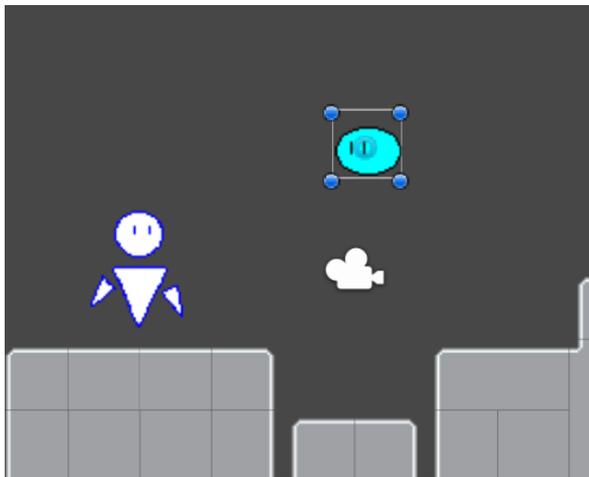
- ① ステージができたので敵をつくります。
- ② まずは絵を用意してください。一枚でいいです。



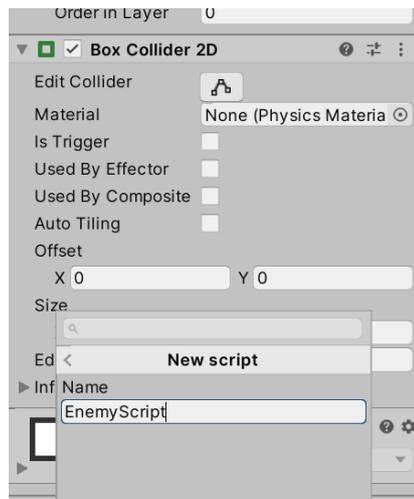
- ③ プロジェクトに放り込みます。



- ④ プロジェクトからシーンに放り込みます。



- ⑤ Box Collider 2D と New Script から EnemyScript を Add Component します。



- ⑥ 敵の仕様は色々考えられると思いますが、とりあえず当たったら HP が減ることにしてしましましょう。まず自分に HP をつけます。Player Script に以下のように加筆してください。

```
public class PlayerScript2 : MonoBehaviour
{
    private Rigidbody2D rigid;
    private LandScript landSc;
    private float XForce;
    private float YForce;
    public float gravity;
    public float jumpForce;
    public float Speed;
    public int HP = 10;
    // Start is called before the first frame update
    void Start()
    {
        rigid = GetComponent<Rigidbody2D>();
        landSc = GetComponentInChildren<LandScript>();
    }
}
```

変数「HP」を追加しました。

- ⑦ 次に、敵と当たったら HP を減らすようにします。この機能は、PlayerScript 中に OnCollisionEnter2D をつけてタグが Enemy だったら……という感じに実装することもできますが、今回は EnemyScript から HP を減らすようにしましましょう。EnemyScript を編集します。

```

public class EnemyScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
    }

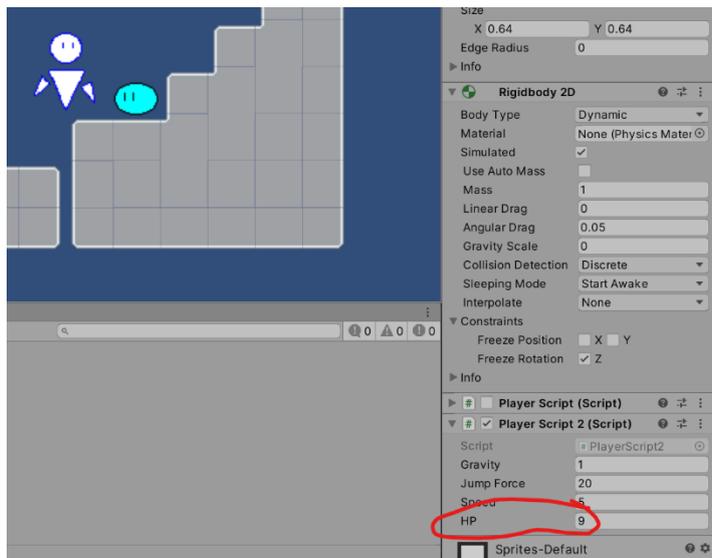
    // Update is called once per frame
    void Update()
    {
    }

    void OnCollisionEnter2D(Collision2D Hit){
        if(Hit.gameObject.CompareTag("Player")){
            Hit.gameObject.GetComponent<PlayerScript>().HP -= 1;
        }
    }
}

```

敵の当たり判定は IsTrigger ではないので、OnTriggerEnter 2 D ではなく OnCollisionEnter 2 D になっています。Collision だと Hit.CompareTag ができないので gameObject の情報の取得を間にかませていることに気をつけてください。

- ⑧ 当たると HP が減るようになりました。



敵の機能はほかにもたくさん考えられると思うので（動くなど）、自分でつけてみてください。（分からなかったら聞いてね）

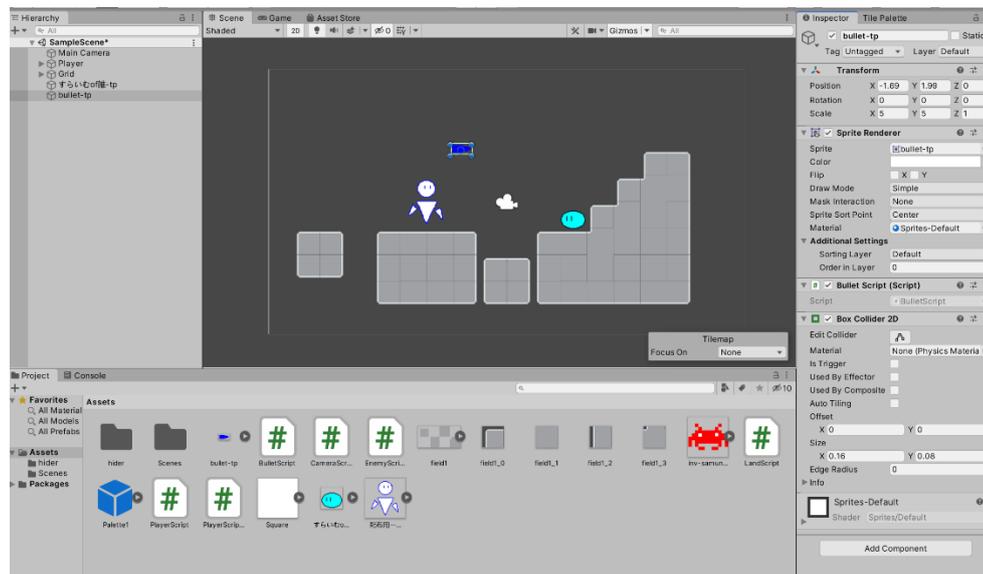
1 4. 弾をつくる

- ① 敵を倒すために弾を発射できるようにしましょう。まずは絵をかきます。



- ② 敵の時と同じ要領でプロジェクトとシーンに放り込みます。
 ③ 敵のときと同じく BoxCollider2D と New Script から BulletScript を追加します。

BoxCollider2D は IsTrigger を On にしてください。



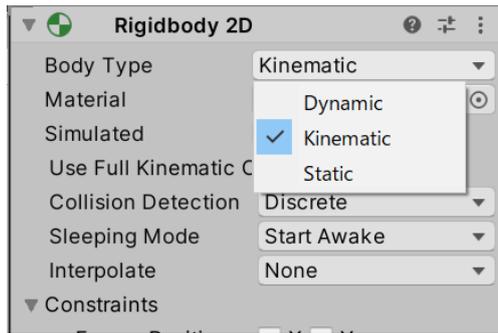
- ④ 弾は自動で動くようにしましょう。Bulletscript を編集します。

```
5 public class BulletScript : MonoBehaviour
6 {
7     // Start is called before the first frame update
8     void Start()
9     {
10
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
16         GetComponent<Transform>().Translate(0.1f,0,0);
17     }
18 }
```

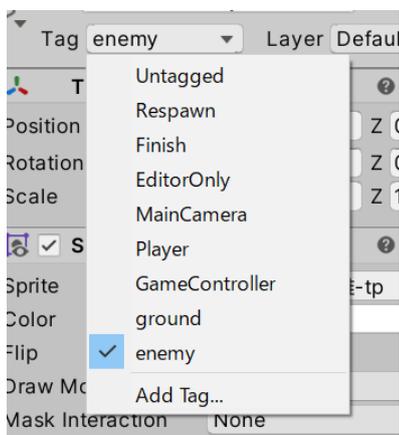
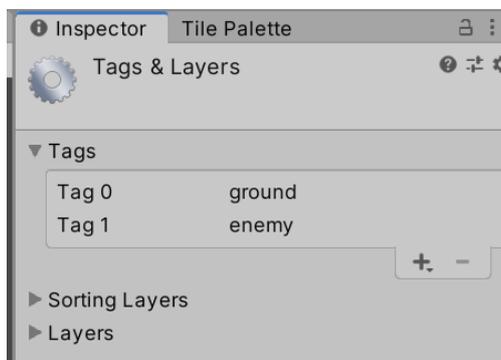
Transform.Translate(x,y,z) : オブジェクトの位置を x,y,z だけ動かす命令。

GetComponent<Transform>() で自分自身の Transform コンポーネントを取得しているの、自身の位置を毎フレーム x 方向に 0.1 だけ動かす、という意味になります。

- ⑤ 弾が敵に当たったら敵が消えるようにしましょう。当たり判定を検知するために弾に rigidbody2D をつけ、Body Mode を kinematic にしてください。



- ⑥ 次に、当たったのが敵かどうかを判断するため、敵に Tag をつけます。敵オブジェクトのタグを Add Tag から enemy にしてください。



- ⑦ 次に BulletScript を編集します。

```
void Update()
{
    GetComponent<Transform>().Translate(0.05f,0,0);
}

0 references
void OnTriggerEnter2D(Collider2D Hit){
    if(Hit.CompareTag("enemy")){
        Destroy(Hit.gameObject);
        Destroy(gameObject);
    }
}
```

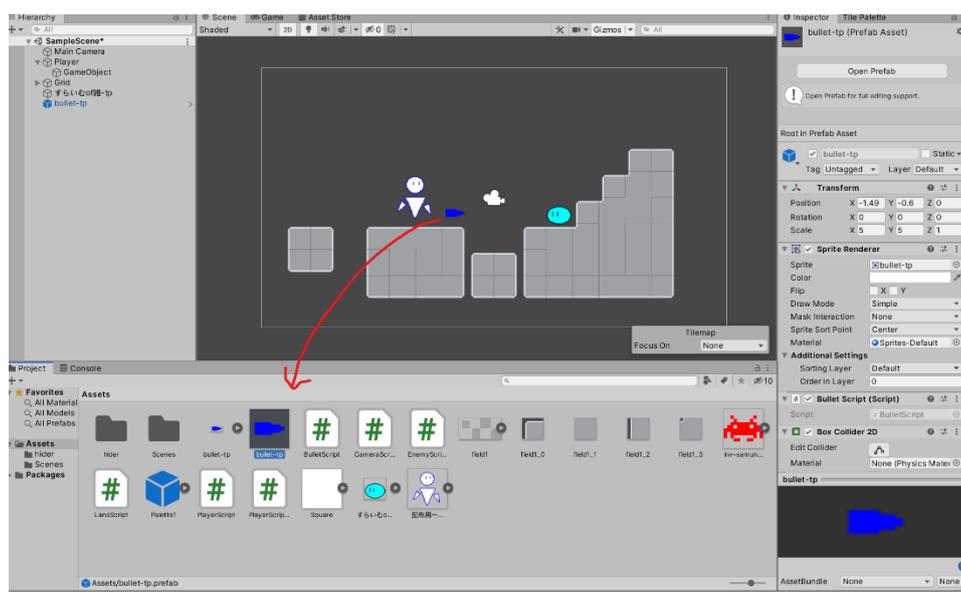
Destroy(Hit.gameObject)は当たったオブジェクト、Destroy(gameObject)は自分自身を消す命令です。

- ⑧ 実行すると、弾が敵にあると弾も敵も消えることがわかります。

15. 弾の発射

この弾をプレイヤーが発射できるようにします。

- ① シーン上の弾をプロジェクトビューにドラッグアンドドロップし、プレハブ化します。



- ② もとにした弾は削除してください。
- ③ PlayerScript を編集します。

```

14     public int HP = 10;
15     public GameObject bullet;
16     // Start is called before the first frame update
17     void Start()
18     {
19         rigid = GetComponent<Rigidbody2D>();
20         landSc = GetComponentInChildren<LandScript>();
21     }
22
23     // Update is called once per frame
24     void Update()
25     {
26         XForce = Speed * Input.GetAxis("Horizontal");
27         if(landSc.IsLand){
28             if(Input.GetKey(KeyCode.UpArrow)){
29                 YForce = jumpForce;
30             }else{
31                 YForce = 0;
32             }
33         }else{
34             YForce -= gravity;
35         }
36         rigid.velocity = new Vector2(XForce,YForce);
37
38         if(Input.GetKeyDown(KeyCode.Space)){
39             Instantiate(bullet,transform.position,Quaternion.Euler(0,0,0));
40         }
41     }

```

変数 bullet を追加し、スペースキーが押されたら弾が発射されるようにしました。

if(Input.GetKeyDown())：そのキーが押されたときだけかっこ内の命令を実行する。GetKey()との違いは、GetKey は押されている間ずっと、だが、GetKeyDown は押され始めたときのみ。

- ④ インスペクターから PlayerScript の bullet のところに先ほどのプレハブをドラッグアンドドロップします。



- ⑤ 実行すると、スペースキーが押されるたびに弾が発射されます。

16. 弾の削除

- ① このままだと弾は敵に当たらない限り消えないため、弾を何発も撃つと処理落ちするようになります。そのため、弾に射程を持たせ、射程を越えたら弾が消えるようにします。BulletScript を編集します。

```

14     void Update()
15     {
16         GetComponent<Transform>().Translate(0.05f,0,0);
17         if(transform.position.x > 10){
18             Destroy(gameObject);
19         }
20         if(transform.position.x < -5){
21             Destroy(gameObject);
22         }
23     }
24

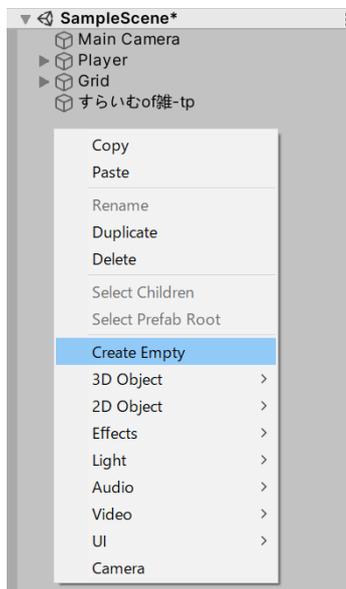
```

弾の x 座標が 10 以上になるか -5 以下になった時に弾が消えるようにしました。ステージの作り方によっては、x 座標が 10 以上になっても弾には消えないでほしい、ということもあると思います。そういう時はこの数値を変えるなり別の基準（弾が発射されてからの秒数など）で弾を消すなりしてください。

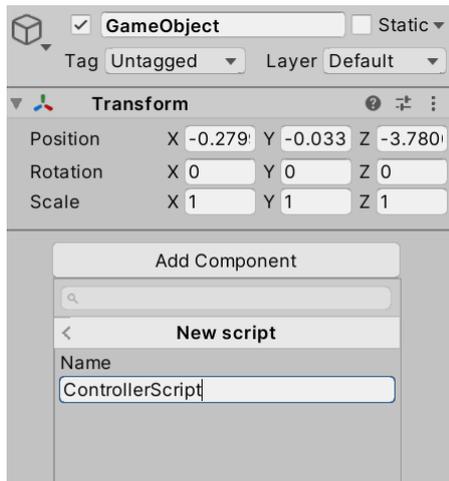
1.7. ゲームオーバーの実装

先ほどプレイヤーに HP をつけましたが、このままでは HP が 0 になっても何も起こらず、マイナスになることもあります。そこで HP が 0 になったとき、ゲームオーバーになるようにしましょう。

- ① ゲーム進行にかかわる処理をやらせようオブジェクトを作ります。Create Empty で空のオブジェクトを作ります。



- ② New Script から ControllerScript を追加して、以下のように編集します。



```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5
6  0 references
7  public class ControllerScript : MonoBehaviour
8  {
9      // Start is called before the first frame update
10     2 references
11     private GameObject Player;
12     2 references
13     private PlayerScript ps;
14     0 references
15     void Start()
16     {
17         Player = GameObject.FindGameObjectWithTag("Player");
18         ps = Player.GetComponent<PlayerScript>();
19     }
20
21     // Update is called once per frame
22     0 references
23     void Update()
24     {
25         if(ps.HP <= 0){
26             SceneManager.LoadScene(SceneManager.GetActiveScene().name);
27         }
28     }
29 }

```

4行目も追加していることに注意してください。

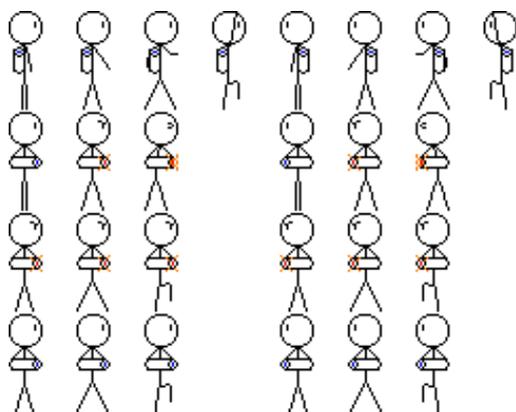
SceneManager.GetActiveScene().name：今実行されているシーンの名前を取得します。SceneManager.LoadScene と組み合わせて、現在のシーンを再読み込みし、初めからやり直すことができます。

- ③ 実行すると、敵に10回当たると初めからやり直しになりました。ゲームオーバーはほかの実装法も思いつくと思うので（シーンを遷移させリトライするかゲームを終わるかを選ばせるなど）ぜひ他の方法でも試してください。

18. アニメーション

ゲームはだいたい形になりました（バグもありますしゲームクリアも作ってませんが、自分でどうにかできるはずです）。ここからはプレイヤーにモーションをつけていきたいと思います。

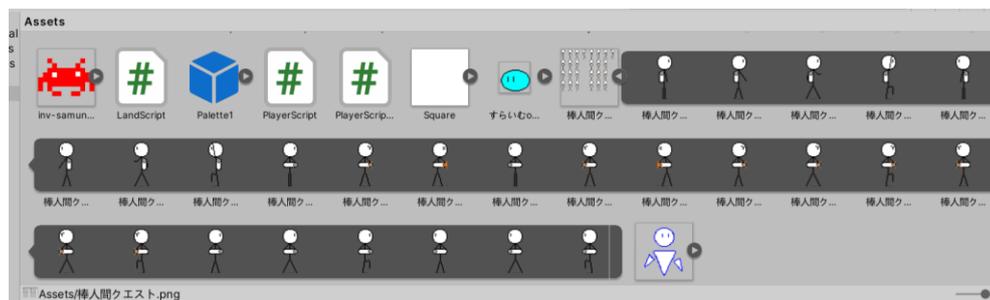
- ① 作業をする前にモーション＝アニメーションの仕組みを知っておきましょう。アニメーションは、パラパラ漫画のように、画面に映る絵を前の絵とすこしずらした絵にすることで動いたように見せる技術です。UnityにはAnimationという機能があり、これでスプライトを変更することでキャラが動いているように見せます（ボーンアニメーションというものもありますが高度なので今回は割愛）。
- ② ではモーションをつくっていきましょう。まずは絵を用意します。



（絵が下手なのはごめんね）

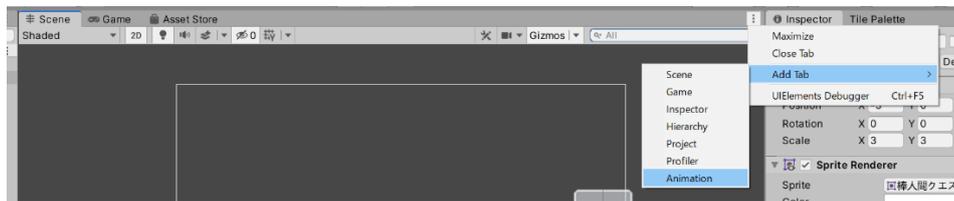
立っている絵、歩いている絵、ジャンプしている絵、攻撃している絵など、欲しい絵を用意してください。絵のサイズは同じだとよいです。また、歩いている絵は2種類以上あるとよいです（足を出している絵とその途中の絵など）。上の画像はそれぞれに左右反転の絵も用意していますが、これは用意しなくても大丈夫です。

- ③ この絵をプロジェクトに放り込み、分割します。

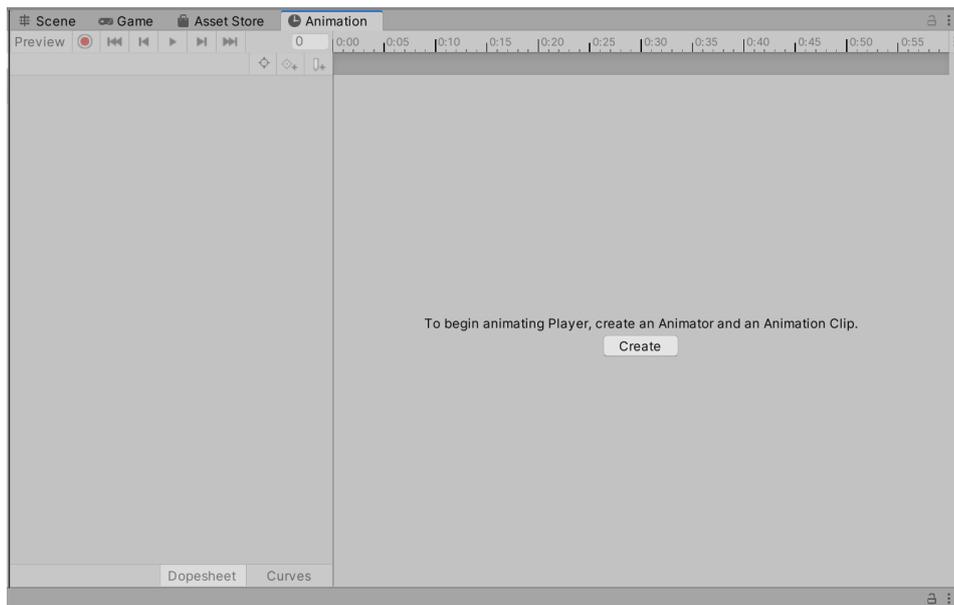


- ④ シーンビュー右上の：ボタン（点は本当は3つ）を押して、Add Tab→Animation

を選択します。

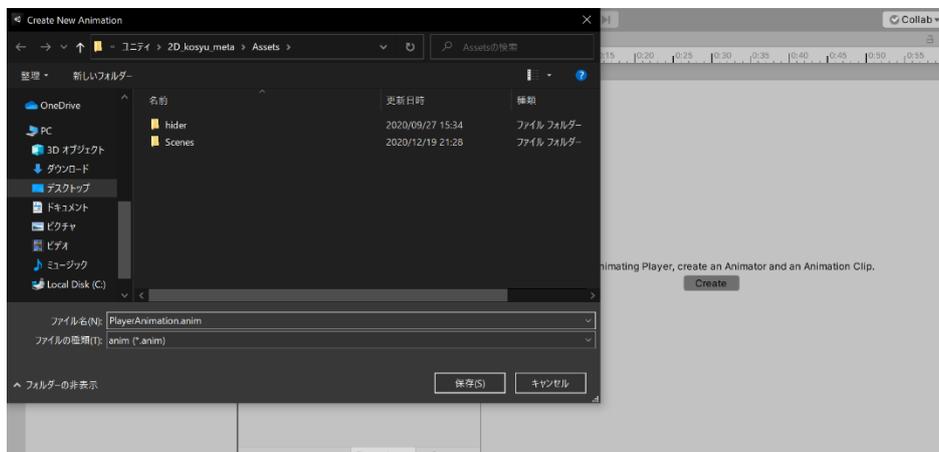


すると Animation タブができます。これはインスペクターやシーンビューなどと同系列のもので。

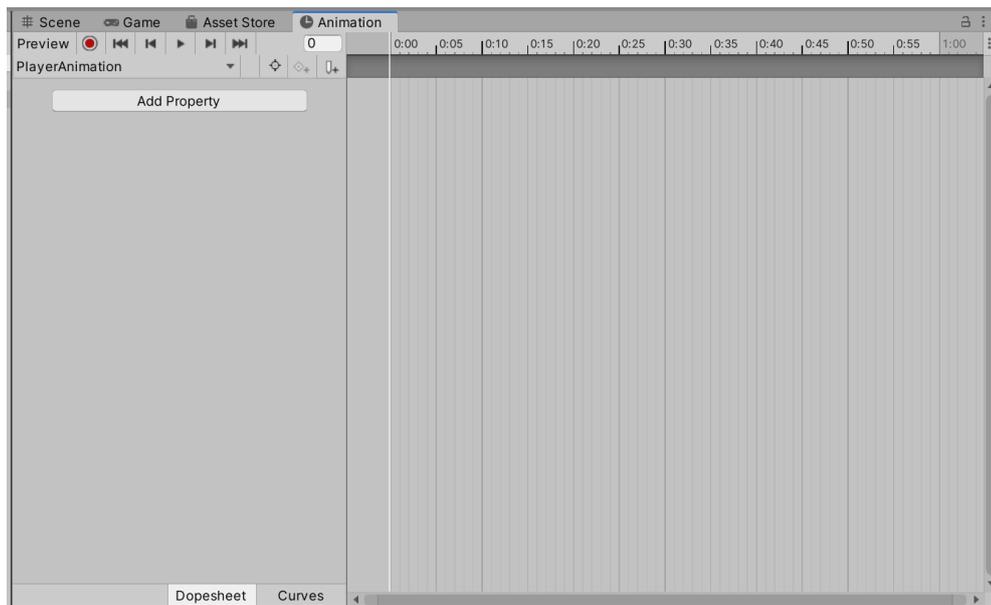


ゲームビューにもどりたときは、左上の Scene ボタンを押すともどれます。

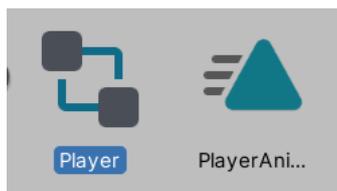
- ⑤ Player を選択した状態で、Animation タブの真ん中にある create ボタンを押してください。すると保存画面にうつるので、PlayerAnimation.anim という名前で保存してください。



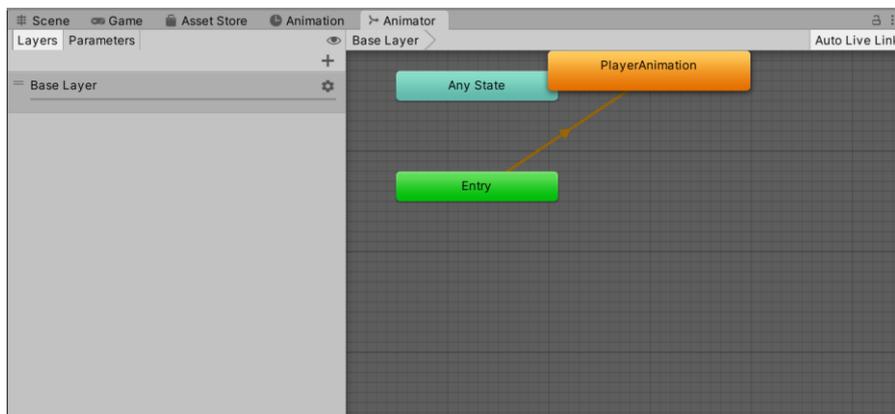
すると Animation タブの中身が変化します。



- ⑥ また、プロジェクトに Player と PlayerAnimation という名前のファイルができていと思うので、Player の方をクリックしてください。

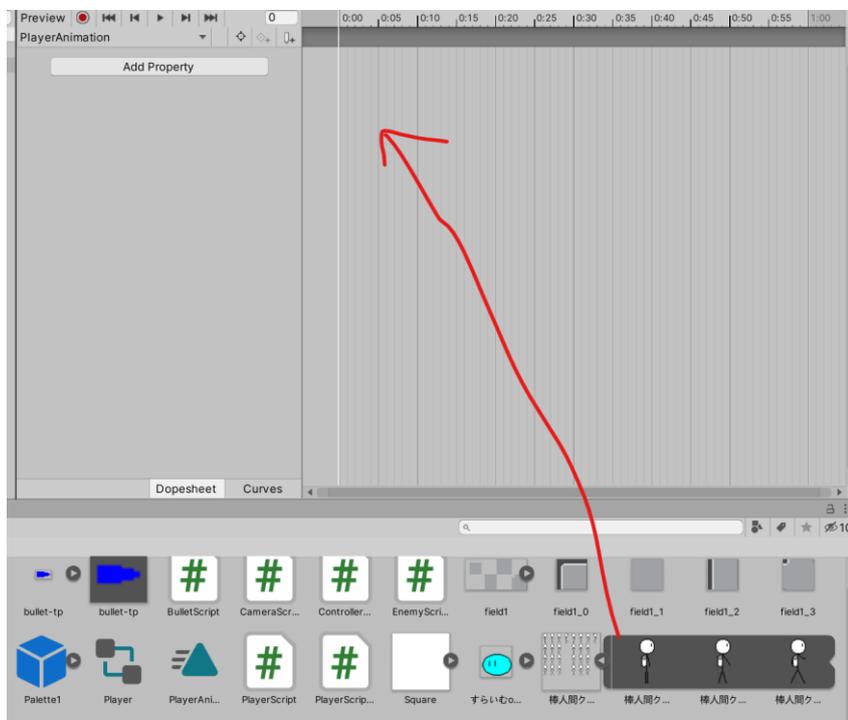


すると Animator という名前のタブが開かれます。

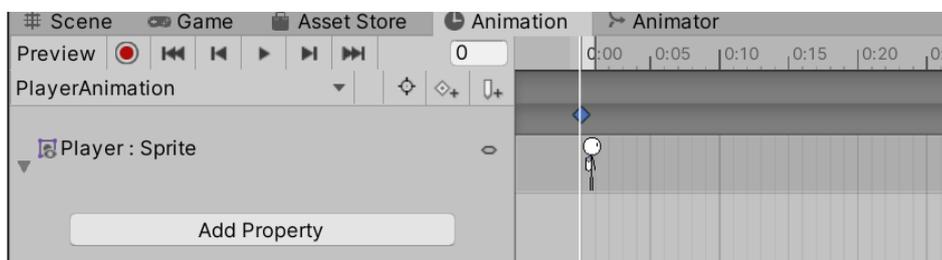


- ⑦ ここまできたところで Animation 機能について説明します。まず、Animation というのは、スプライトを何秒かごとに切り替えて一つのまとまった動きをあらわすものです。Animation は一つのオブジェクトに対して複数つくることができ、役割ごとに複数つくるのが基本です（歩いているときのモーション、ダメージを受けたときのモーションなど）。そして、Animation を管理するものがあり、それが Animator です。主に Animation の切り替えをします。では実際にやってみましょう。

- ⑧ まずは止まっているモーションを作ります。Animation タブに移動し、止まっているときに表示したいスプライトをタブの右側にドラッグアンドドロップしてください。

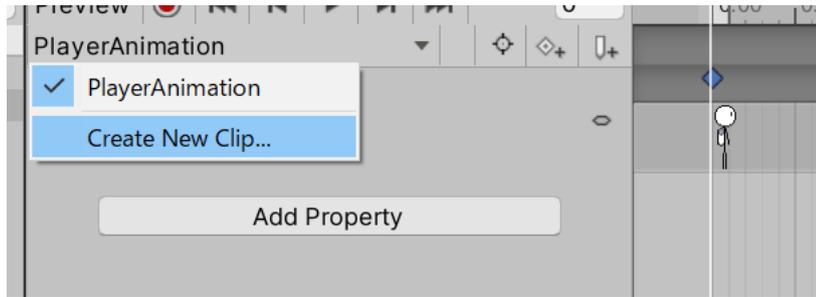


すると以下ようになります。



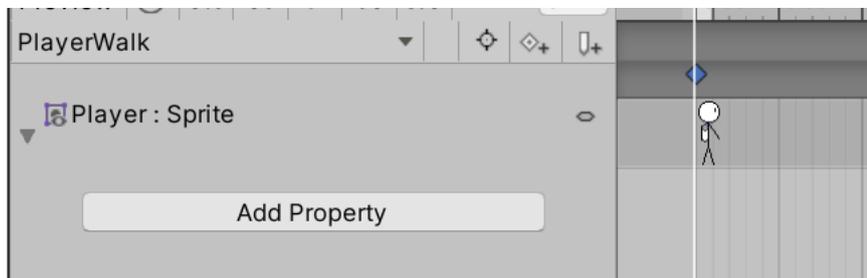
これで、PlayerAnimation の一番最初の絵が登録されました。この画面は時間経過とその時のスプライトを表示するタイムラインなので、0秒目に止まった絵が表示されることになります。止まっているときにキャラは動かない（待機モーションとかもありますが今回はそういう仕様にしました）ので、止まっているモーションはこれで完成です。

- ⑨ 次に歩いているときのモーションを作ります。タブ左側の PlayerAnimation と書かれた部分の右側の▼をクリックし、Create new clip を選択→PlayerWalk という名前で保存してください。

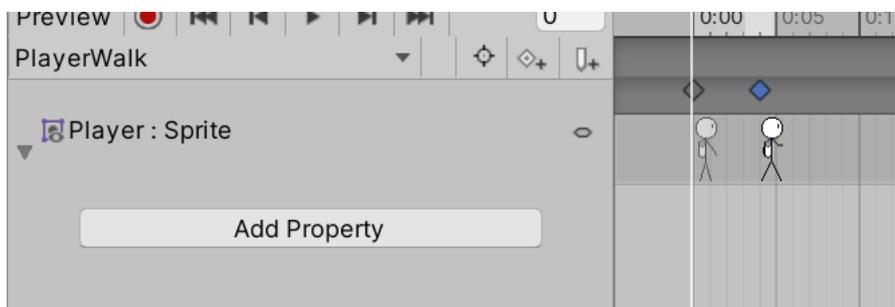


すると編集している Animation が PlayerAnimation から PlayerWalk になります。先ほどのモーションをもう一度編集したいときは、今の▼を押して編集したい Animation へ移動してください。

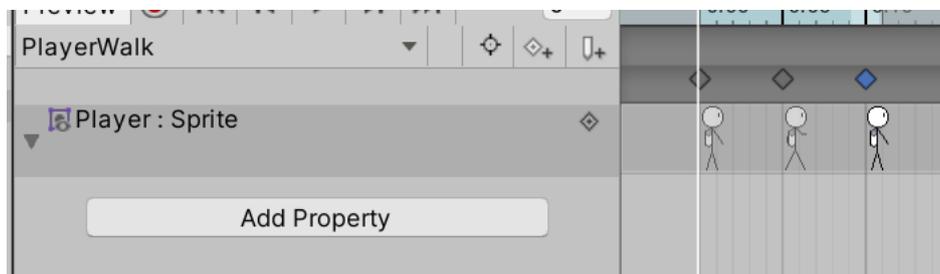
- ⑩ 歩いているときのスプライトを一つドラッグアンドドロップしてください。



- ⑪ ⑩とは別の歩いているときのスプライトをその後ろ側にドラッグアンドドロップしてください。

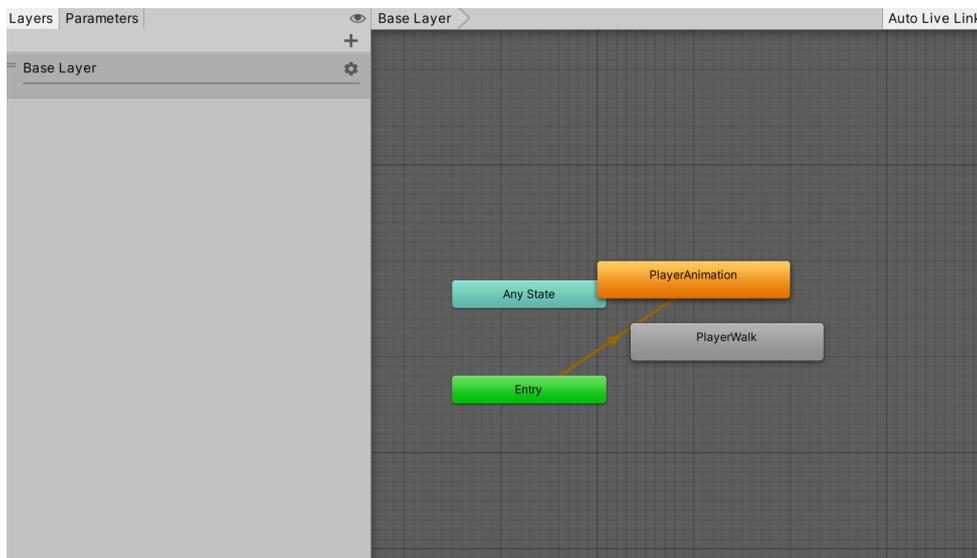


- ⑫ ⑩と同じスプライトをそのさらに後ろにドラッグアンドドロップしてください。



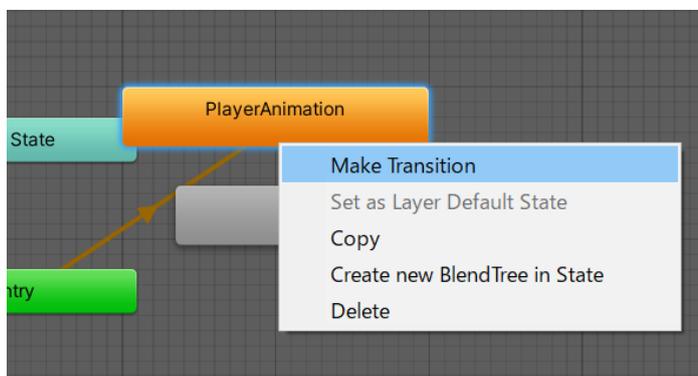
これは、この Animation を再生したときに最後に来たら一瞬で最初に戻ってしまい、最後に配置したスプライトが見えなくなってしまうからです。これで歩きモーションは完成とします。

- ⑬ ではこれを Player に適応してみましょう。Animator タブに移動します。

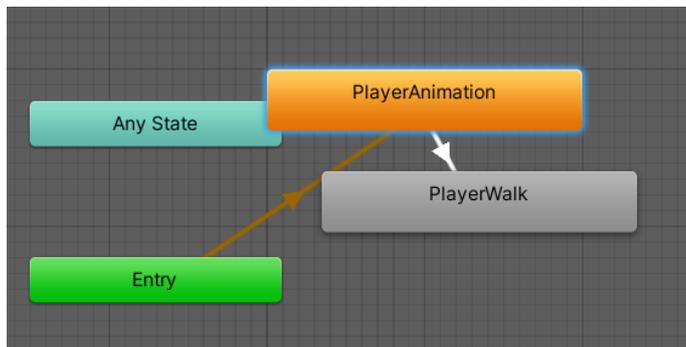


この画面では先ほど作った Animation をどのようなときに切り替えるのかの設定をします。Entry から PlayerAnimation に矢印が出ていますが、これは始まった時は PlayerAnimation の状態にする、という意味です。

- ⑭ この画面ではある Animation から別の Animation への遷移を矢印で表します。PlayerAnimation を右クリックし、Make Transition をクリックした後、PlayerWalk をクリックしてください。



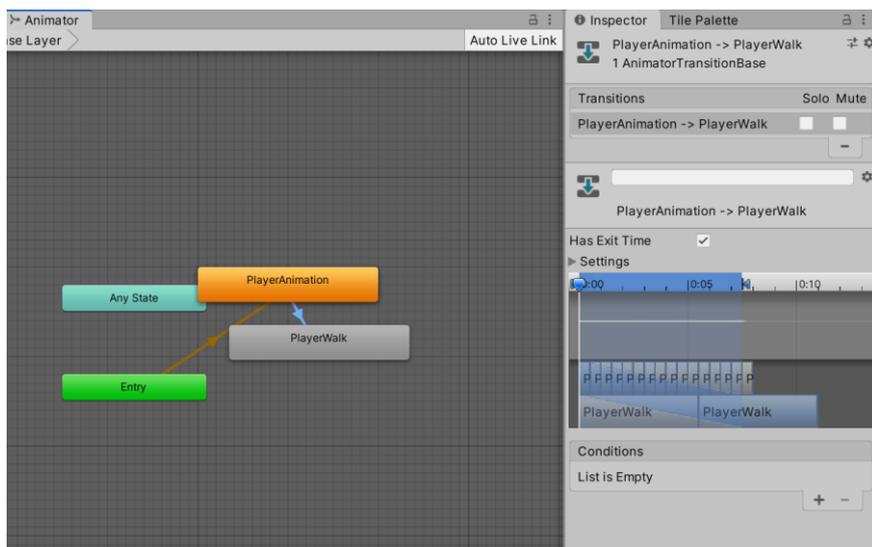
すると矢印が作られました。



- ⑮ ここでゲームを実行してみましょう。Player が PlayerWalk で設定したように動いているのがわかるはずです。これは、ゲームが始まった後 PlayerAnimation を再生し、その再生が終わったら PlayerWalk を再生したことになります。

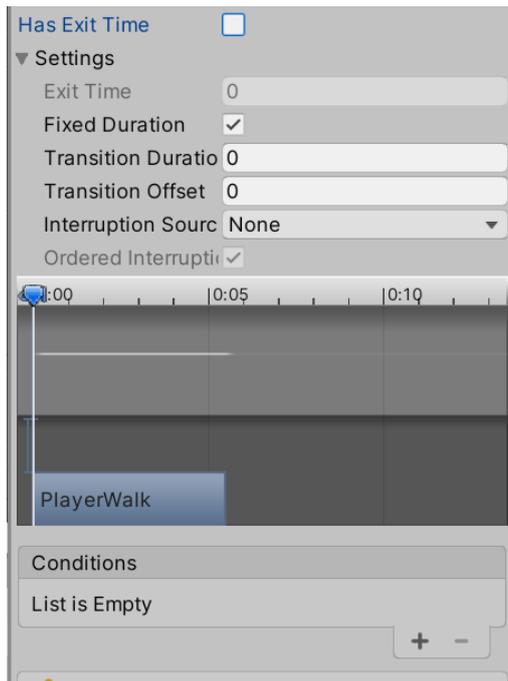
19. アニメーション2

- ① 歩いているときだけ PlayerWalk を表示してもらいたいのので、そのように設定をします。まずは Animator タブの先ほど作った矢印をクリックしてください。



インスペクターに矢印の設定が出ました。これをいじることでアニメーション遷移の設定が行えます。

- ② まずは Has Exit Time のチェックをオフにしてください。これは「アニメーションが終わったら次のアニメーションに遷移する」という意味です。また、Settings を開いたところにある Transition Duration を 0 にしてください。この項目は 3D の時に使うものです。

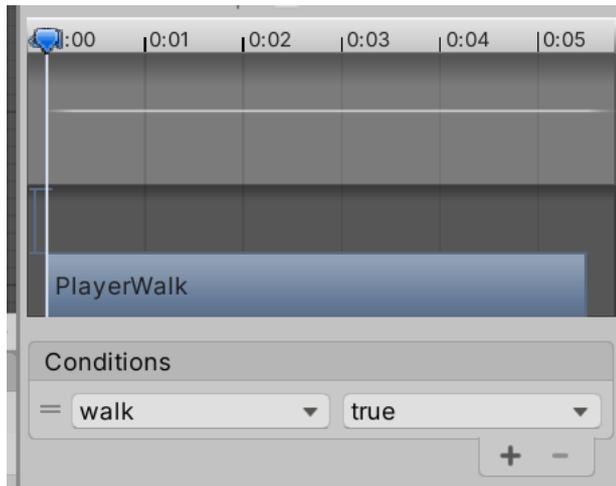


- ③ いまのままでは遷移する条件がないので、それをつくります。右上の Parameter をクリックして、左下に出てきた+ボタンをクリックして、Bool を選択してください。すると名前を付けるように言われるので、walk としてください。



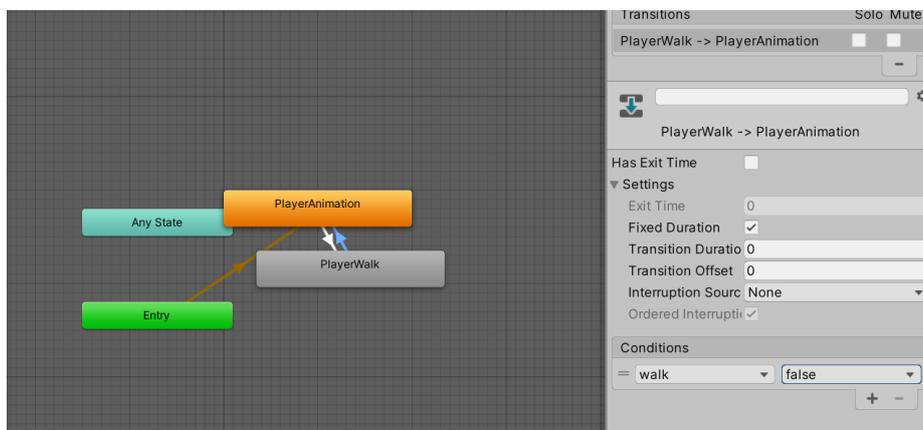
これは、Bool 型の変数 walk を作ったということになります。この walk が true の時に PlayerWalk のアニメーションを再生し、false の時に PlayerAnimation を再生することにします。

- ④ 矢印の設定の Conditions の下にある+ボタンを押してください。すると以下のようになります。



Conditions は何がどうなったらアニメーションを遷移させるかの条件の設定です。ここでは変数 walk が true になったらアニメーションが遷移する、という意味です。

- ⑤ このままだと PlayerAnimation から PlayerWalk に遷移すると、PlayerAnimation にもどることができません。そのため PlayerWalk から PlayerAnimation に新しい矢印を作ってもどれるようにします。



先ほどと同じように矢印を作り、設定を上画像のように変更してください。先ほどとの違いは Conditions が false になっていることです。

- ⑥ これで変数 walk の値でアニメーションを管理できるようになりました。しかし、変数 walk は勝手には変わらないので、スクリプトから変えてやる必要があります。PlayerScript を編集します。

```

public class PlayerScript2 : MonoBehaviour
{
    private Rigidbody2D rigid;
    private LandScript landSc;
    private Animator anime; //変更
    private float XForce;
    private float YForce;
    public float gravity;
    public float jumpForce;
    public float Speed;
    public int HP = 10;
    public GameObject bullet;
    // Start is called before the first frame update
    void Start()
    {
        anime = GetComponent<Animator>();
        rigid = GetComponent<Rigidbody2D>();
        landSc = GetComponentInChildren<LandScript>();
    }

    // Update is called once per frame
    void Update()
    {
        XForce = Speed * Input.GetAxis("Horizontal");
        if(Input.GetAxis("Horizontal") != 0){ //変更
            anime.SetBool("walk",true); //
        }else{ //
            anime.SetBool("walk",false); //
        } //ここまで
        if(landSc.IsLand){
            if(Input.GetKey(KeyCode.UpArrow)){

```

変数 anime にアニメーターの情報を取得し、SetBool という命令で walk の値を変えています。

anime.SetBool(“パラメータの名前”,値)でパラメータの値を変えられます。

- ⑦ 実行すると、プレイヤーが移動しているときだけ歩くアニメーションをしています。これでアニメーションの基本は完成です。

これで Unity 2 D 講習は終わりです。

ここまで来たあなたに

講習お疲れ様でした！この資料を最後まで読み切ったあなたはある程度の Unity の実力を得たはずですよ。そこで、最後に実力を確かめる課題を出します。この課題の提出をもって講習を卒業したと認めます。いままでの資料を見ながらでも、ネットや本を見ながらでも

よいので、やって提出してみてください。

課題

この Unity 2D 講習で作ったゲームは多くの改善点があります。自分が気になった点を改善し、私に見せてください。

(改善点の例：敵の上に乗ると行動不可能になる、ジャンプのアニメーションがない、など)