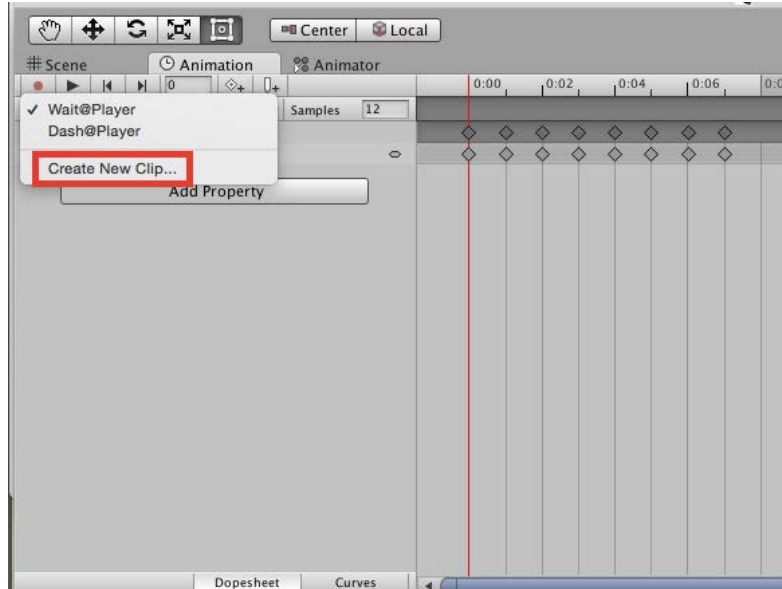


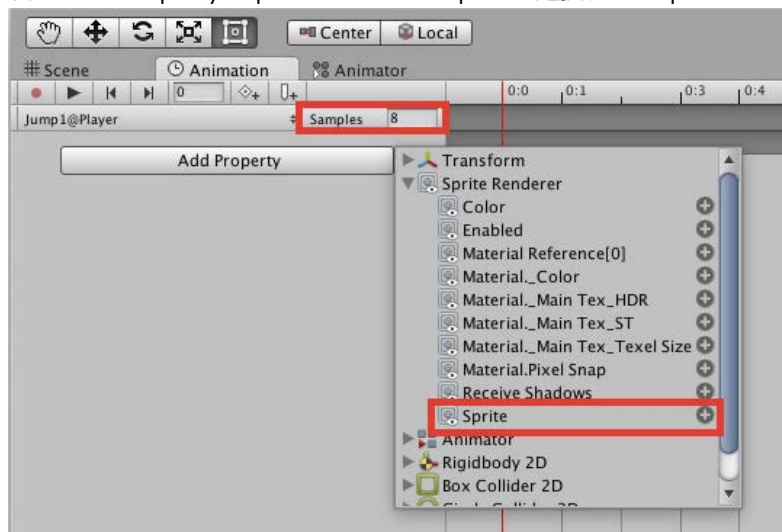
Unity ちゃんをジャンプさせる。

1. ジャンプアニメーションを作成する

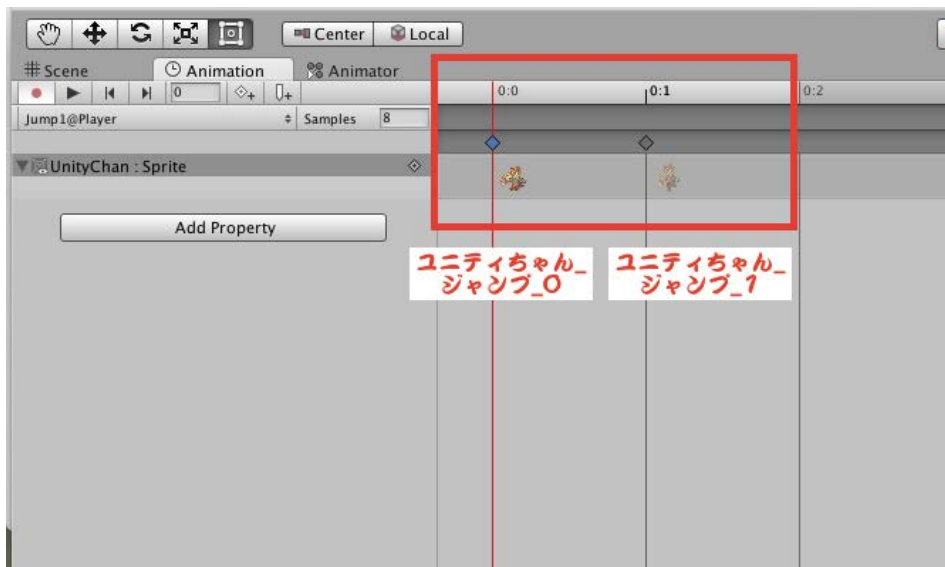
Hierarchy ビューでユニティちゃんを選択した状態で、Animation ビューを開き、Create New Clip を選択します。
アニメーション名は「Jump1@Player」とします。



次に、Add Property>Sprite Renderer>Sprite を選択し、Samples を 8 に変更します。



続いて Sprite を配置していきます～ユニティちゃん_ジャンプ_0 を 0.0 地点、ユニティちゃん_ジャンプ_1 を 0.1 地点に配置します



同じ要領で、以下の設定で他のアニメーションも作成していきます。

名前:Jump2@Player,

Samples: 8,

0.0: ユニティちゃん_ジャンプ_2, 0.1: ユニティちゃん_ジャンプ_3

名前:Jump3@Player,

Samples: 8,

0.0:ユニティちゃん_ジャンプ_4, 0.1: ユニティちゃん_ジャンプ_5

名前:Jump4@Player,

Samples: 8,

0.0:ユニティちゃん_ジャンプ_6, 0.1: ユニティちゃん_ジャンプ_7

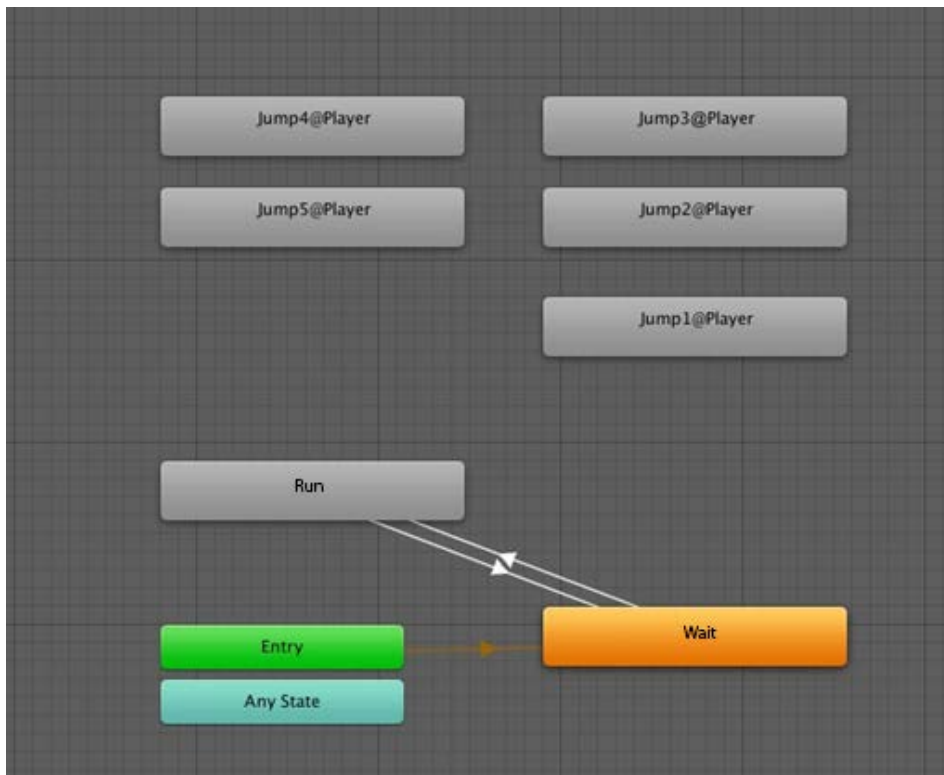
名前:Jump5@Player,

Samples: 8,

0.0:ユニティちゃん_ジャンプ_8, 0.1: ユニティちゃん_ジャンプ_0

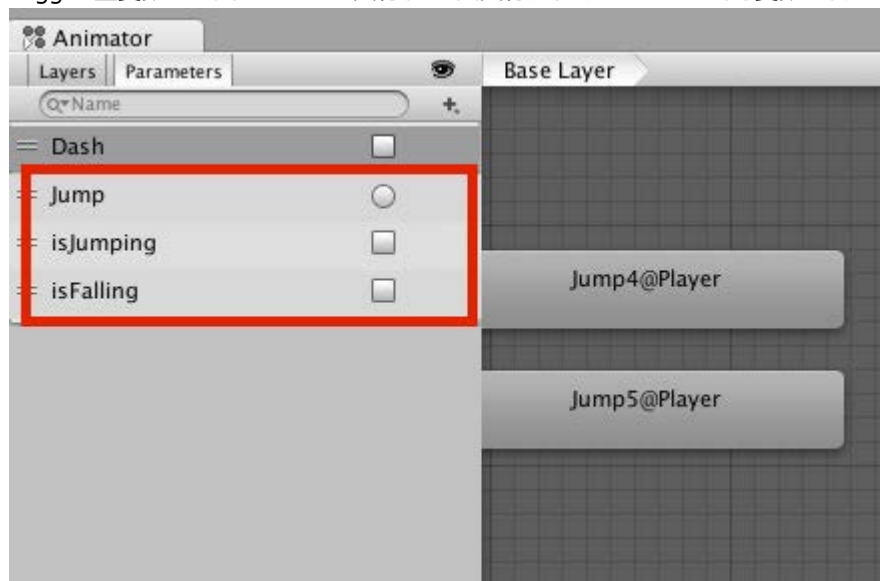
2. ジャンプアニメーションを設定する

続いてアニメーターの設定をしていきます。AnimatorControllers フォルダの Player を開き、アニメーションの配置を整頓します。開くと下記のようになっているはずです。

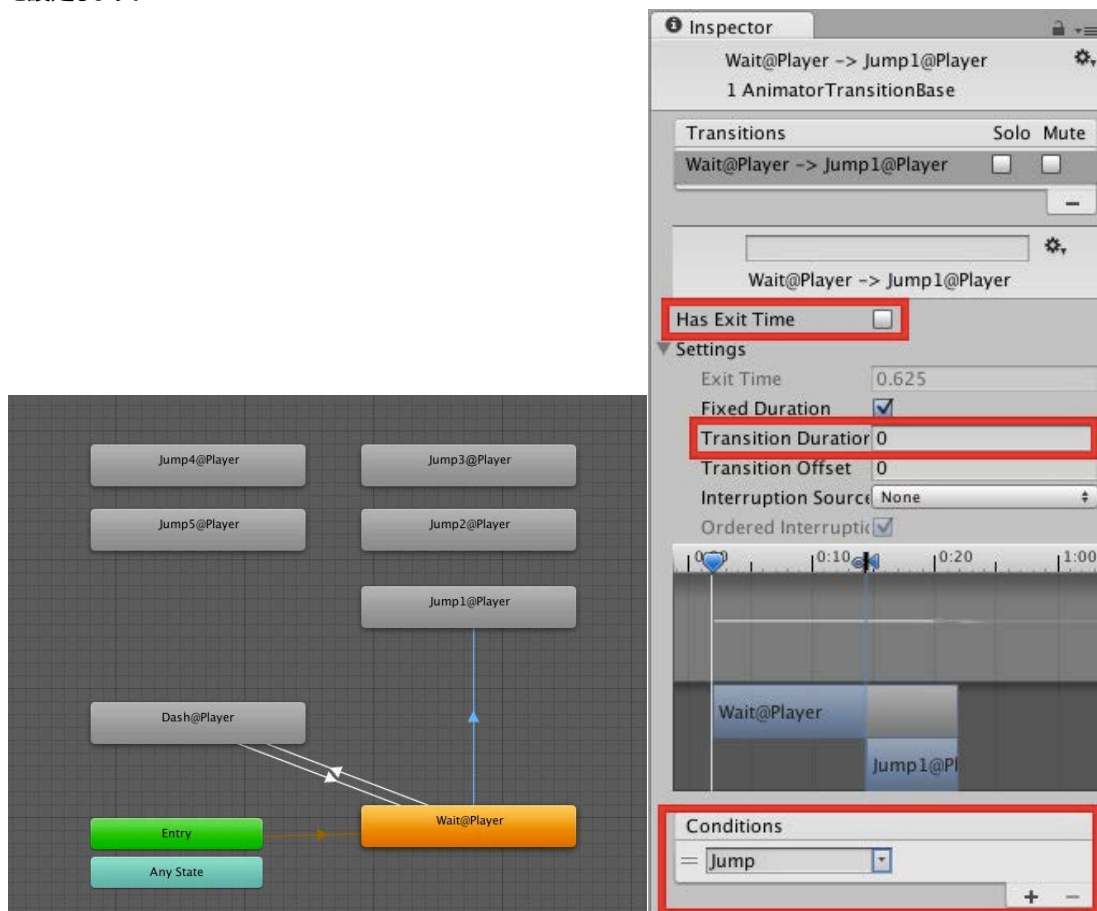


続いてアニメーター内に新たに変数を 3 つ作成します。ジャンプしたかどうかの判定に Trigger 型の Jump 変数、もう 1 つはジャンプ中に下から上へ移動しているかの判定を行うため Bool 型の isJumping 変数、最後にジャンプ中に上から下へ移動しているかの判定のための Bool 型の isFalling 変数になります。

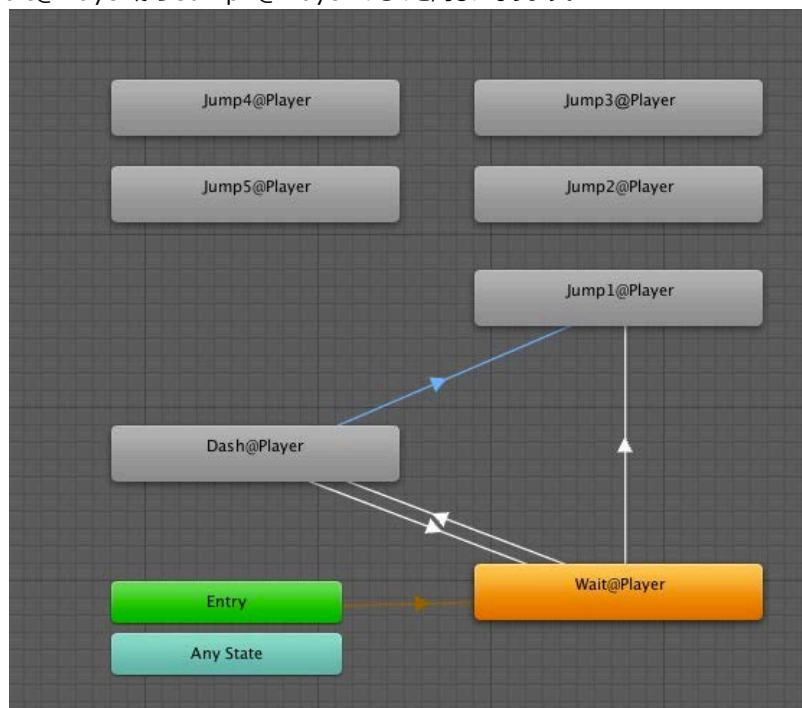
Trigger 型変数は 1 回だけ true を実行し、1 回実行するとすぐに false となる変数になります。



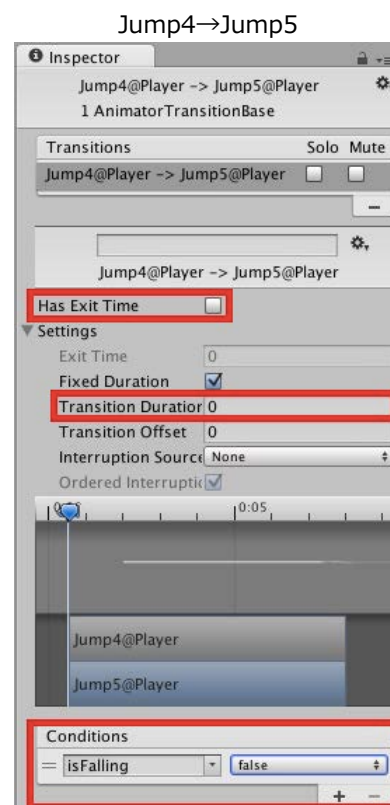
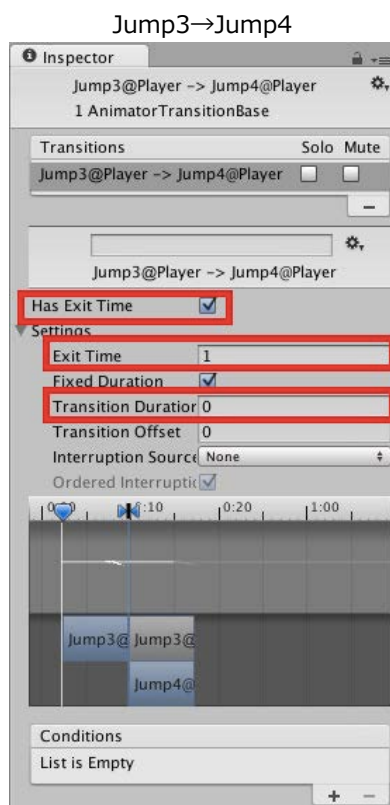
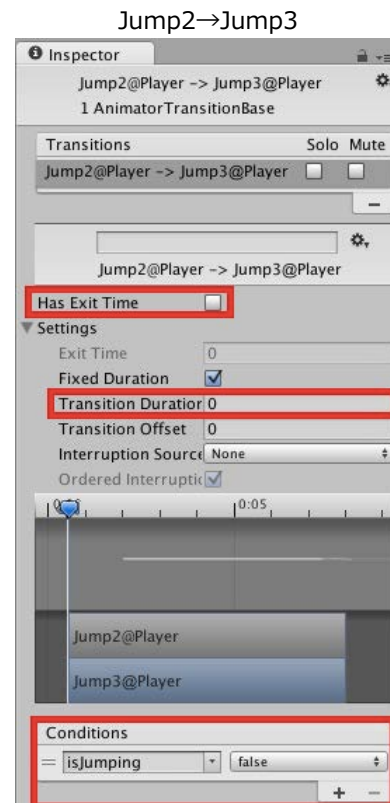
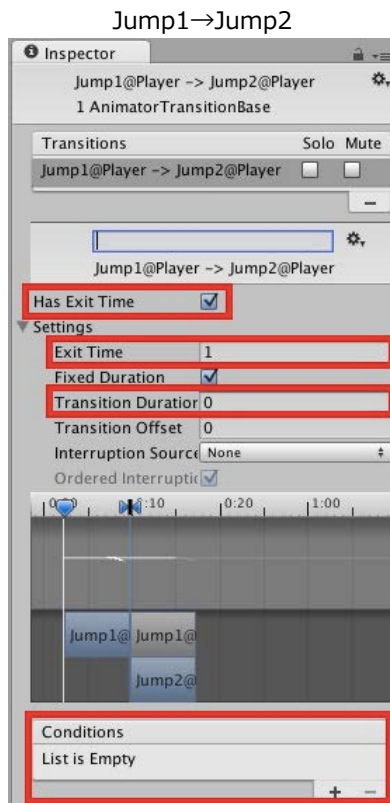
変数の作成が終わったら、アニメーション遷移の設定を行います。Wait@Player から Jump1@Player へ Make Transition し、Inspector ビューを Has Exit Time を false、Transition Duration を 0、Conditions に Jump を設定します。

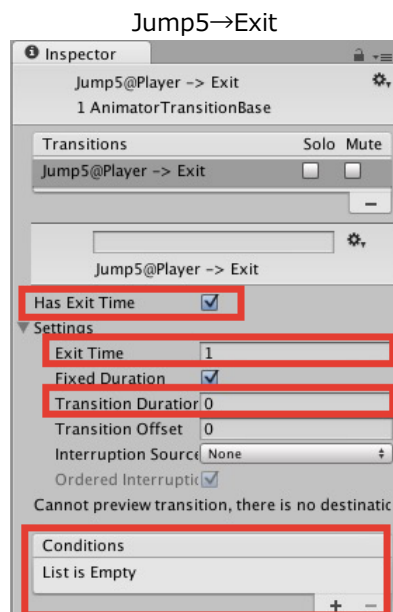


また、Dash@Player から Jump1@Player へも Make Transition を行います。Inspector 内容は Wait@Player から Jump1@Player のものと同一になります。



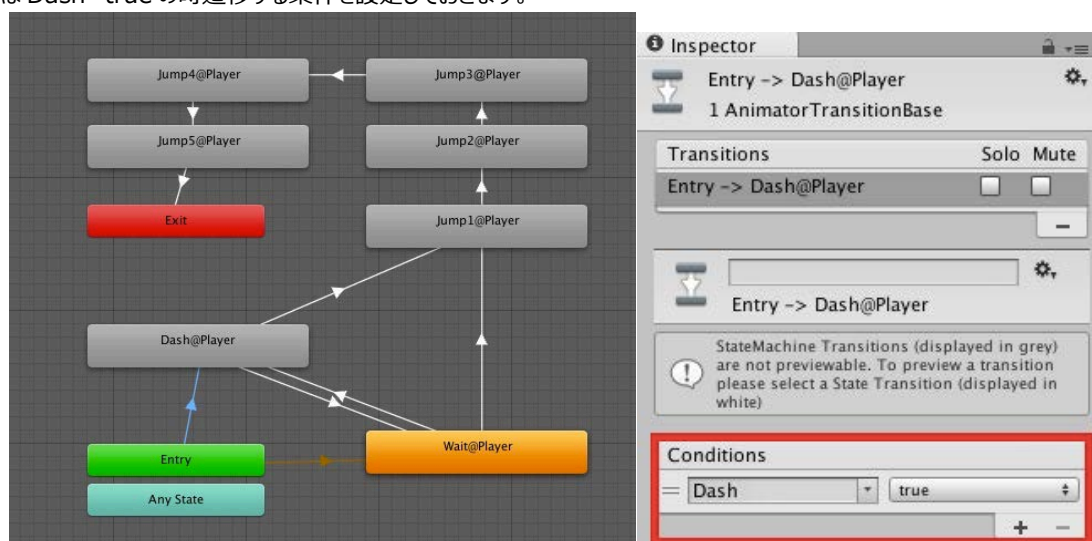
それでは同じ要領で、以下の Inspector ビュー画像を参考に Jump アニメーションの遷移設定をしていきましょう。





アニメーター上で Exit までいくと、Entry へ戻ります。

Entryに戻った時点で左右どちらかのボタンを押していた時は Wait@Player を挟まずに Dash@Player へ遷移してほしいため、Entry から Dash@Player へも Make Transition を行います。また、Inspector ビューの Conditions には Dash=true の時遷移する条件を設定しておきます。



3. ジャンプ用スクリプトを追加する

Player.cs

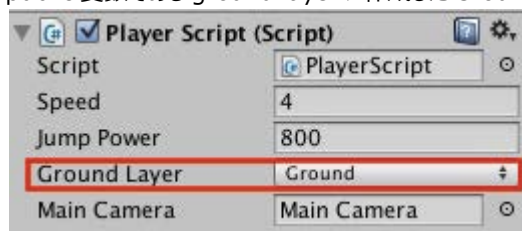
```
1  using UnityEngine;
2  using System.Collections;
3
4  public class PlayerScript : MonoBehaviour {
5
6      public float speed = 4f;
7
8      public float jumpPower = 700; //ジャンプ力
9      public LayerMask groundLayer; //Linecast で判定する Layer
10
11     public GameObject mainCamera;
12     private Rigidbody2D rigidbody2D;
13     private Animator anim;
14
15     private bool isGrounded; //着地判定
16
17     void Start () {
18         anim = GetComponent<Animator>();
19         rigidbody2D = GetComponent<Rigidbody2D>();
20     }
21
22     void Update ()
23     {
24         //Linecast でユニティちゃんの足元に地面があるか判定
25         isGrounded = Physics2D.Linecast (
26             transform.position + transform.up * 1,
27             transform.position - transform.up * 0.05f,
28             groundLayer);
29         //スペースキーを押し、
30         if (Input.GetKeyDown ("space")) {
31             //着地していた時、
32             if (isGrounded) {
33                 //Dash アニメーションを止めて、Jump アニメーションを実行
34                 anim.SetBool("Dash", false);
35                 anim.SetTrigger("Jump");
36                 //着地判定を false
37                 isGrounded = false;
38                 //AddForce にて上方向へ力を加える
39                 rigidbody2D.AddForce (Vector2.up * jumpPower);
40             }
41         }
42         //上下への移動速度を取得
43         float velY = rigidbody2D.velocity.y;
44         //移動速度が 0.1 より大きければ上昇
```

```
45     bool isJumping = velY > 0.1f ? true:false;
46     //移動速度が-0.1 より小さければ下降
47     bool isFalling = velY < -0.1f ? true:false;
48     //結果をアニメータービューの変数へ反映する
49     anim.SetBool("isJumping",isJumping);
50     anim.SetBool("isFalling",isFalling);
51 }
52
53 void FixedUpdate ()
54 {
55     float x = Input.GetAxisRaw ("Horizontal");
56     if (x != 0) {
57         rigidbody2D.velocity =
58             new Vector2 (x * speed, rigidbody2D.velocity.y);
59         Vector2 temp = transform.localScale;
60         temp.x = x;
61         transform.localScale = temp;
62         anim.SetBool ("Dash", true);
63         if (transform.position.x > mainCamera.transform.position.x - 4) {
64             Vector3 cameraPos = mainCamera.transform.position;
65             cameraPos.x = transform.position.x + 4;
66             mainCamera.transform.position = cameraPos;
67         }
68         Vector2 min = Camera.main.ViewportToWorldPoint(new Vector2(0, 0));
69         Vector2 max = Camera.main.ViewportToWorldPoint(new Vector2(1, 1));
70         Vector2 pos = transform.position;
71         pos.x = Mathf.Clamp(pos.x, min.x + 0.5f, max.x);
72         transform.position = pos;
73     } else {
74         rigidbody2D.velocity = new Vector2 (0, rigidbody2D.velocity.y);
75         anim.SetBool ("Dash", false);
76     }
77 }
78
```

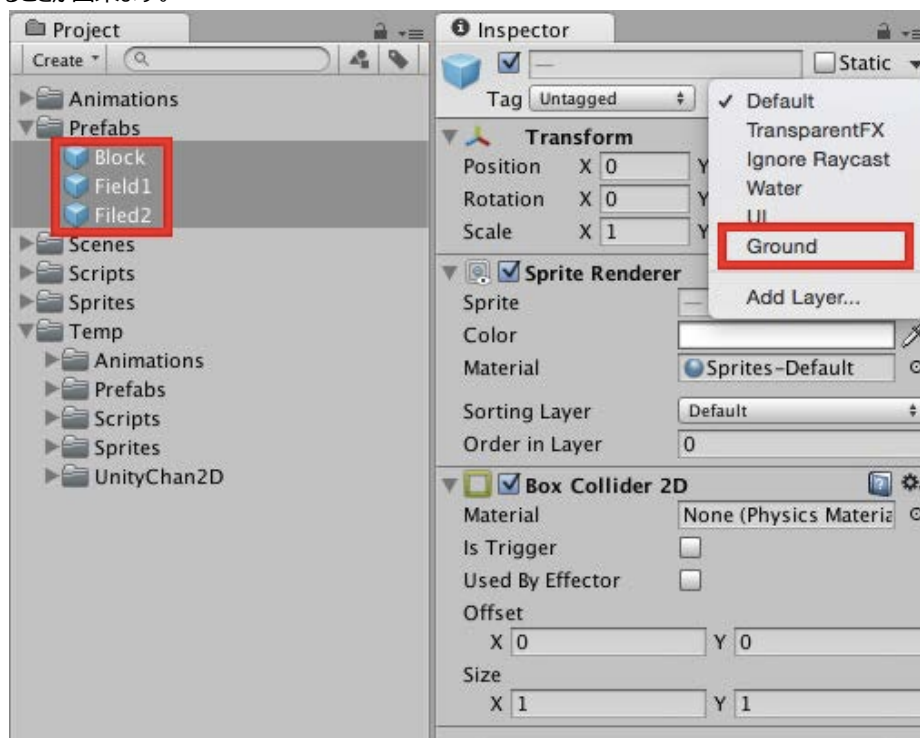

ソース入力後、まずは Edit > Project Settings > Tag & Layers Layers から Layers に Ground を追加し、



public 変数である groundLayer に作成した Ground レイヤーを設定します。



また、用意した Prefab 全てに Ground レイヤーを設定します。その際下記のようにまとめて Ground レイヤーを設定することが出来ます。



スクリプトの流れとして下記の通りになります。

① Linecast で足元に Ground レイヤーがあるか判定

Sprite Editor で画像の中心を Bottom に設定したため、足元を基準として、足元から上へ+1 した位置から足元から-0.05 した位置へ線を引き、その線が Ground レイヤー付きのオブジェクトとぶつかっていれば isGrounded=true、ぶつかっていなければ isGrounded=false とします。

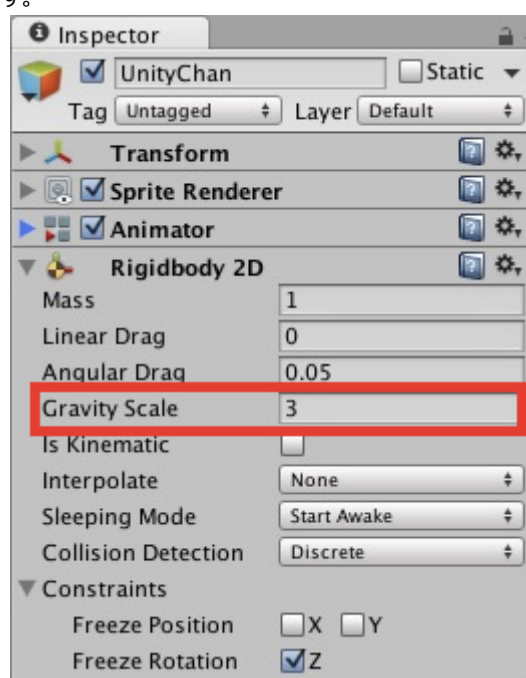


② Jump アニメーションの実行

③ AddForce にて上方向へ力を加える

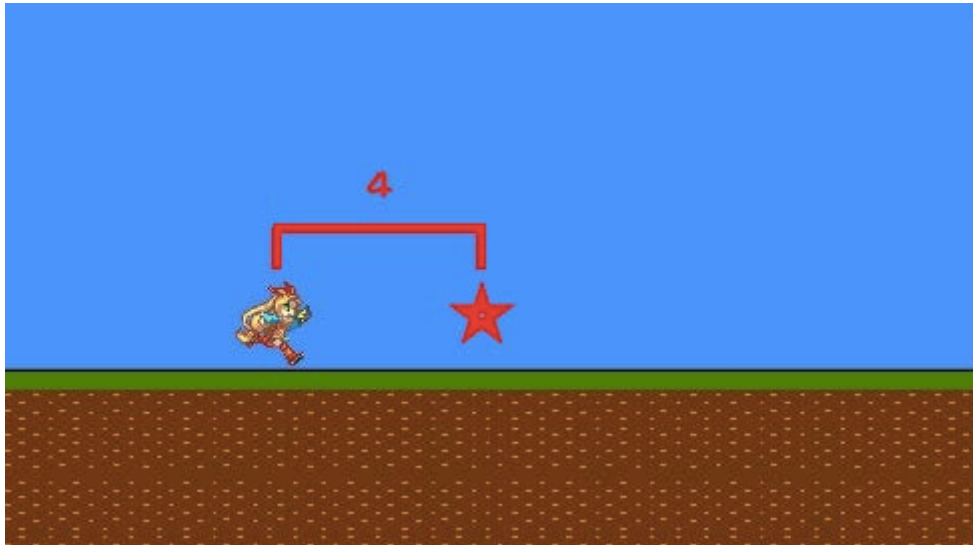
④ 上昇していれば isJumping を true、下降していれば isFalling を true にしアニメーションの切替

最後に、このままだとユニティちゃんが高くジャンプしすぎてしまうので、Rigidbody2D の Gravity Scale を 3 に変更します。

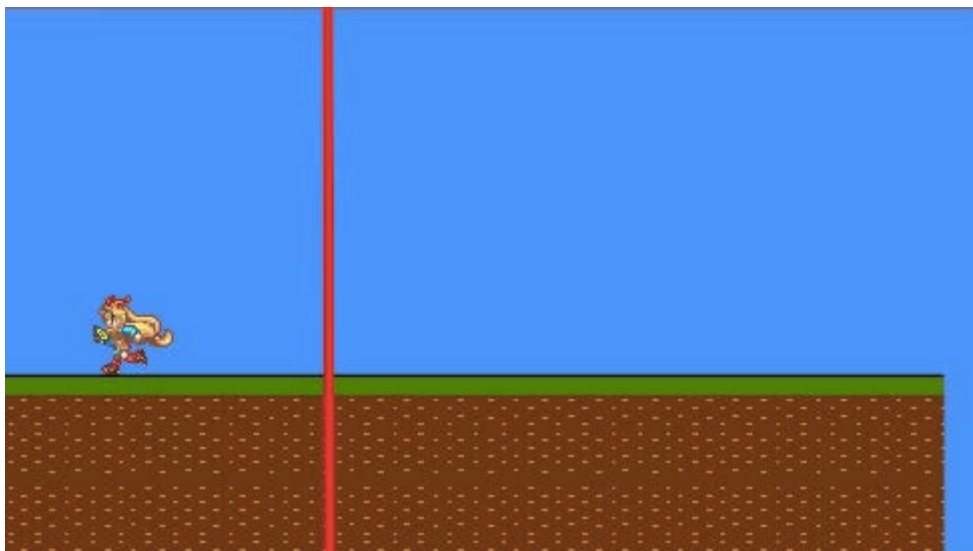


これで、Unity ちゃんがジャンプできるはずです。

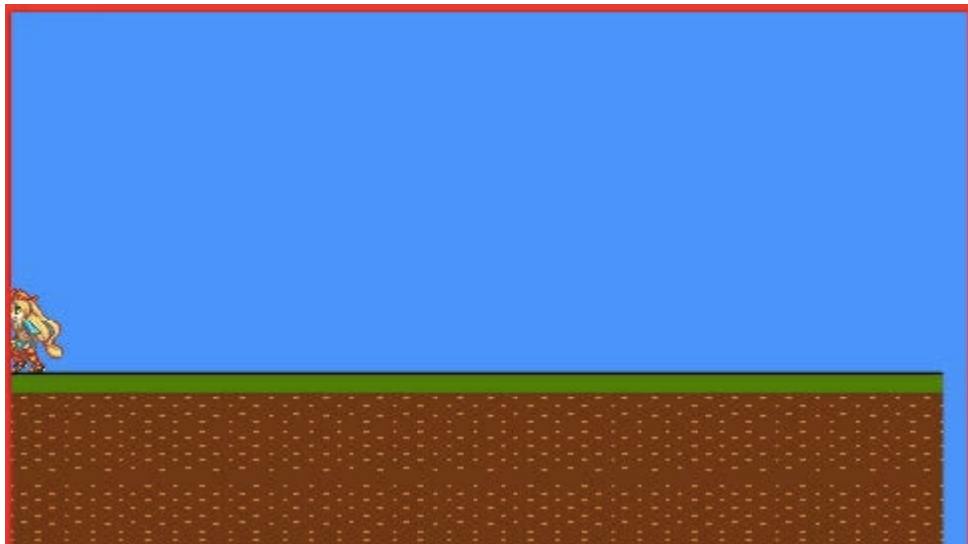
これで、画面中央から CAMERA_OFFSET 離れた位置をユニティちゃんを超えると、Main Camera もそれに合わせて移動するようになります。



また左に移動した場合、カメラ位置は動かしません



そして Mathf.Clamp メソッドで Unity ちゃんの移動位置を制限しているため画面からはみ出ません。



ポイント

Transform : オブジェクトの位置、回転、スケールを扱うクラス

Unity Documents

<https://docs.unity3d.com/jp/current/ScriptReference/Transform.html>

シーン内のすべてのオブジェクトは Transform を持ちます。Transform はオブジェクトの位置、回転、スケールを格納し、操作するために使用されます。すべての Transform は親を持ち、階層的に位置、回転、スケールを適用することができます。これはヒエラルキーウィンドウで階層を見ることができます。

Mathf.Clamp : 与えられた最小 float 値と最大 float 値の範囲に値を制限します。

Unity Documents

<https://docs.unity3d.com/ja/540/ScriptReference/Mathf.Clamp.html>

読み込み専用変数の設定。

const と static readonly の使い分け

const フィールドは、コンパイル時定数の扱い。(MSDN より)

- 変数のように扱える定数 (暗黙的 static)
- 宣言時にのみ初期化可能 (コンパイル時に値が埋め込まれる)
- readonly より実行速度が速い
- switch 文やデフォルト引数に使える
- インスタンスを new した結果は割り当てられない (C# の組み込み型のみ)

readonly フィールドは、実行時定数の扱い。(MSDN より)

- 実際は、読み取り専用の代入不可な変数
- 宣言時の他に、コンストラクタ内でも初期化可能
- 定数である const よりは、僅かに実行速度が遅い
- switch 文やデフォルト引数には使えない
- インスタンスを new した結果を割り当てられる

static readonly

- const が使いたいけど、使えない場合に、static readonly を使用する。

定数値のシンボル名が必要で、その値の型を const 宣言で使用できない場合、またはその値をコンパイル時に計算できない場合は、static readonly フィールドが役に立ちます。(MSDN - 定数用の static readonly フィールド)

まとめ

- 基本的に、static readonly を使用する。
- const は、属性に指定するパラメータや列挙型の定義など、コンパイル時に値が必要な場合にのみ使用する。
- Effective C# でも、const よりも readonly の使用が推奨されている。

高いパフォーマンスが求められていて、なおかつ将来にわたって変更されることがないことが明らかな場合にのみコンパイル時定数を使用すべき。