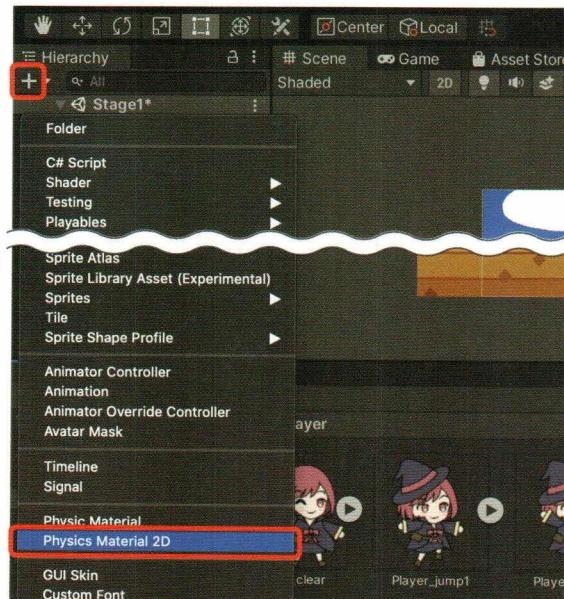


ジャンプ動作を調整しよう (Physics Material 2D の追加)



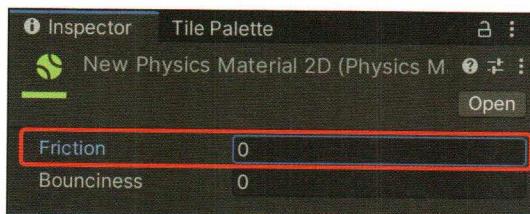
これでプレイヤーキャラクターをジャンプさせることができます。しかし、このままでは1つ問題があります。ジャンプ中に左右キーを押した状態で壁やブロックに接触すると、プレイヤーが壁にくっついて落ちてこなくなるのです。

これはキャラクターとブロックの間に摩擦抵抗が発生しているために起こっています。つまり摩擦をゼロにすればくっつかなくなるというわけです。



そのためにはCapsule Collider 2Dコンポーネントにマテリアルを設定します。

プロジェクトビュー左上の[+]ボタンから、Physics Material 2Dを探して選択してください。これは物体の物理特性を調整するためのパラメーターです。



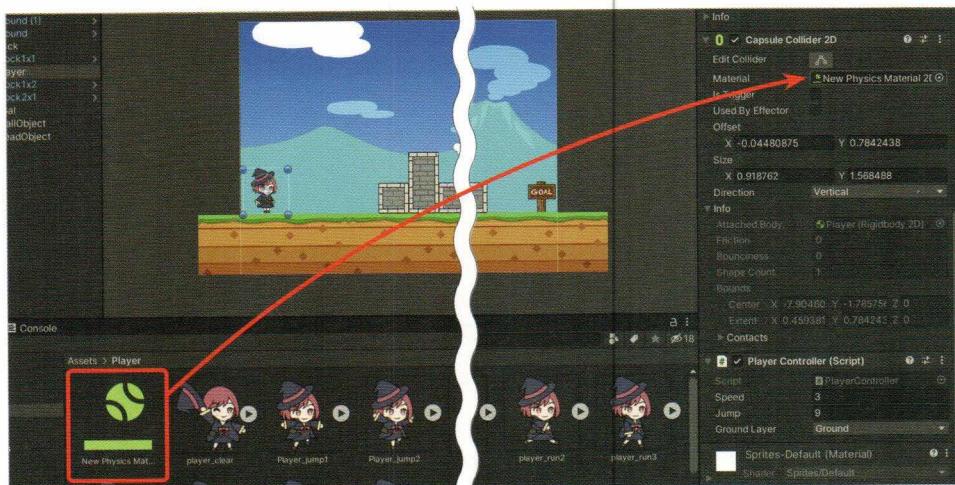
すると、プロジェクトビューにアイコンが追加されます。アイコンを選択して、インスペクタービューを見てください。「Friction」という項目があります、これは「0.0」～「1.0」の間で物体どうしの摩擦係数を表すものです。ここではこの値を0にしましょう。



摩擦係数

物体の滑りにくさを0.0～1.0の小数で表す数値です。1.0では他の物体に接触したときにまったく滑らなくなり、0にすれば、他の物体に接触したときになめらかに滑るようにできます。

「Friction」の値を「0」に設定したら、プレイヤーキャラクターを選択し、プロジェクトビューから Capsule Collider 2D の Material に、Physics Material 2D をドラッグ & ドロップすれば完了です。このマテリアルデータは Player フォルダーに保存しておきましょう。



4

5

7

サイドビューゲームの基本システムを作ろう

ジャンプ挙動の調整

Tips

ジャンプの動作が少し「ふわっ」とした感じがするなら、Rigidbody 2D の「Gravity Scale」を調整してみましょう。

「Gravity Scale」はゲームオブジェクトにかかる重力の数値です。「1」では1G、つまり地球上と同じ重力となります。この値を1よりも大きくすればかかる重力が大きくなり、ジャンプ力が抑えられ、「0.5」のように1よりも小さくすれば軽くなります。また、「0」にすると無重力状態になります。

サンプルでは、「Gravity Scale」の値を「1.5」に調整しています。

プレイヤーキャラクターのアニメーションを作ろう

次は、キャラクターにアニメーションを付けていきます。キャラクターは7枚の画像を順に切り替えて表示することで移動中にアニメーションするようにしましょう。

アニメーションはUnityの**メカニム**(Mecanim)という仕組みを使って作ります。まずはメカニムの仕組みを簡単に説明しておきましょう。

アニメーションをさせるためには、以下の4つのデータを作ってメカニムを使用します。

◆ スプライト (Sprite)

アニメーションさせるための画像データがスプライトです。Unityで画像を表示するSprite Rendererというコンポーネントがありましたね。Sprite Rendererコンポーネントはこのスプライトデータを使って画像を表示しているというわけです。

◆ アニメーションクリップ (Animation Clip)

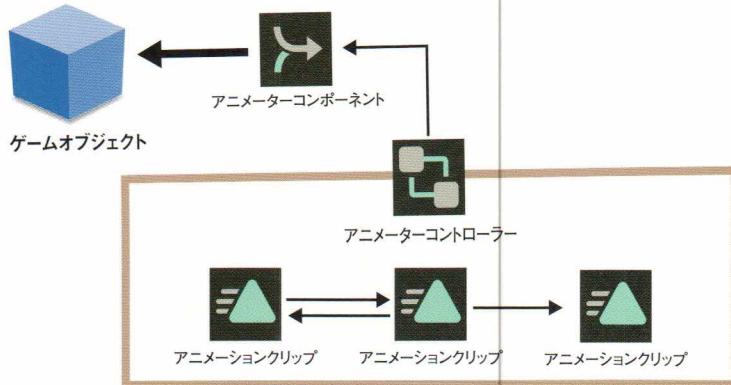
アニメーションクリップは、複数のスプライトを使って画像を切り替えて、アニメーションさせるためのデータです。1アニメーションに対する再生時間や再生速度などの管理が行えます。

◆ アニメーターコントローラー (Animator Controller)

複数のアニメーションクリップを管理するデータです。ゲームキャラクターには立ち止まり、走ったりジャンプしたり、いろいろなアニメーションをさせたいですよね。アニメーターコントローラーを使うと、それら個々のアニメーションをさせるアニメーションクリップを一つ、どこで切り替えるのかを管理できます。

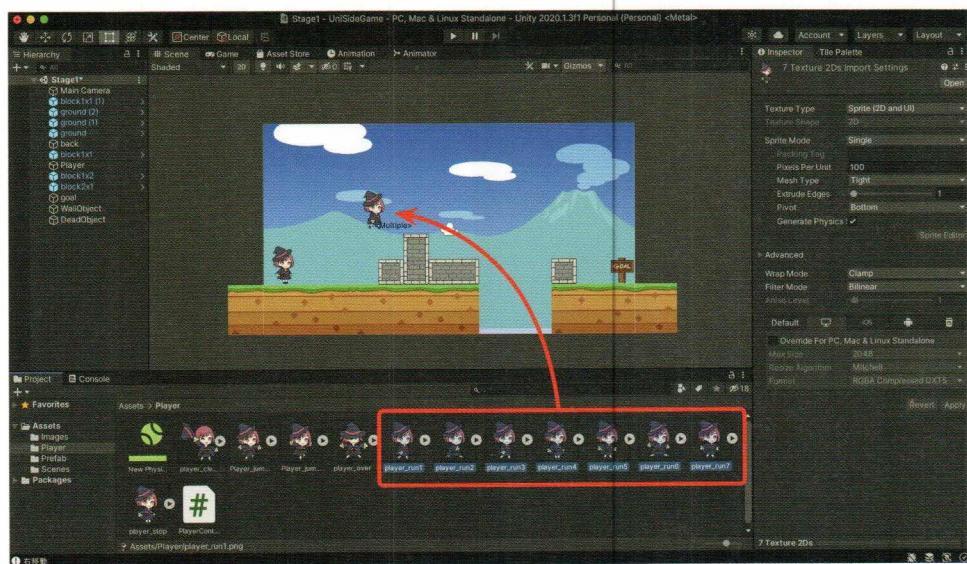
◆ アニメーターコンポーネント (Animator Component)

アニメーターコンポーネントは、ゲームオブジェクトにアタッチしてアニメーションさせるコンポーネントです。アニメーターコンポーネントにアニメーターコントローラーを設定してアニメーションをさせるのです。データの名前が似ていて少しわかりにくいですね。アニメーションデータの関係を表したのが次の図になります。アニメーターコンポーネントが一番外側にある入れ子構造になっています。

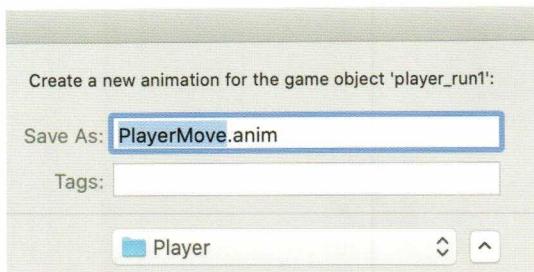


移動アニメーションを作ろう

それでは実際に、キャラクターの移動アニメーションを作ってみましょう。一番簡単なアニメーションの作り方は、「プロジェクトビューにある複数の画像アセットをシーンビューにドラッグ＆ドロップすること」です。そうすればUnityが自動的に必要なアニメーションデータを作ってくれます。

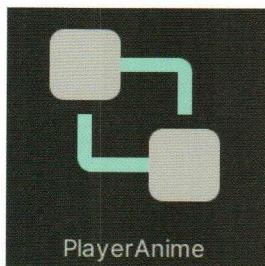


まず、プロジェクトビューで移動の画像「player_run1」～「player_run7」をすべて選択してシーンビューにドラッグ＆ドロップしてください。

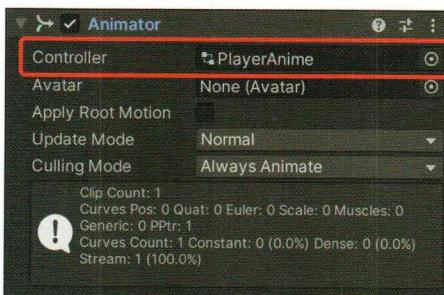


このように複数の画像をシーンビューに配置すると、アニメーションクリップでファイルの名前と保存場所を指定するためのダイアログが開くので、ファイル名と場所を決めて保存しましょう。ここでは移動アニメーションを作るため「PlayerMove」というファイル名を付けてPlayerフォルダーに保存します。

プロジェクトビューには2つの新しいアイコンができています。順に見ていきましょう。



このアイコンはアニメーターコントローラーで、複数のアニメーションクリップを管理するデータです。アニメーターコントローラーの名前は画像ファイルの名前になっていますが、後々わかりやすいように「PlayerAnime」に変更しておきましょう。プロジェクトビューでアイコンを選択して、[Return]キーを押せば名前を編集することができます。

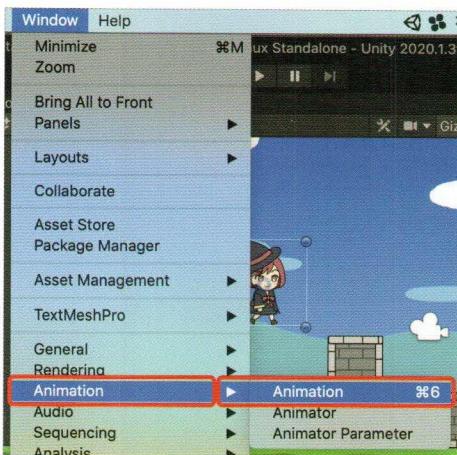


シーンビューに配置したプレイヤーキャラクターのゲームオブジェクトを選択して、インスペクタービューを見てください。Animatorというコンポーネントが付いていますね。このうち「Controller」という項目に入っているものがアニメーターコントローラーです。



このアイコンはアニメーションクリップで、複数のスプライトをまとめてアニメーションさせるデータです。先ほどPlayerMoveという名前で保存したデータがこれになります。

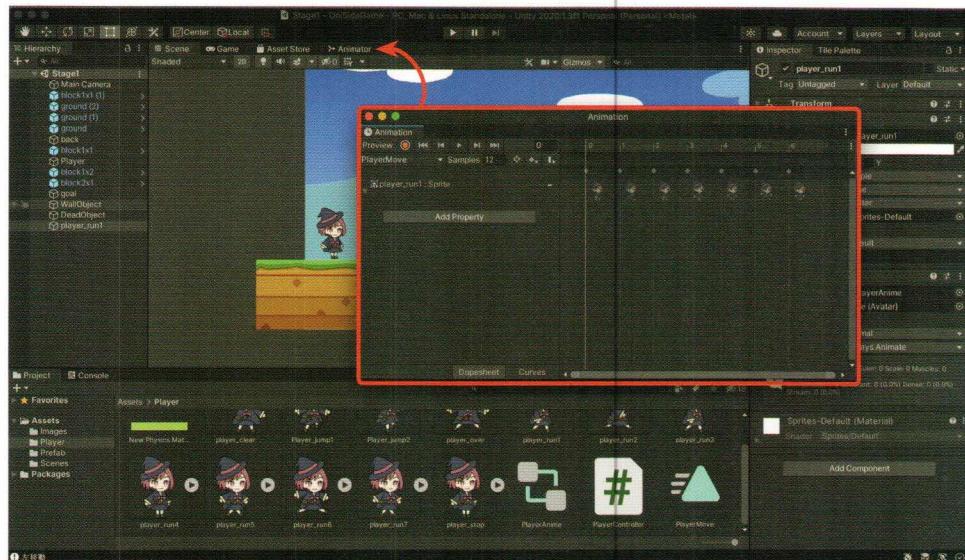
◆ 「Animation」 ウィンドウ



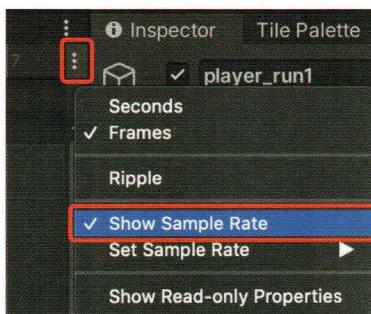
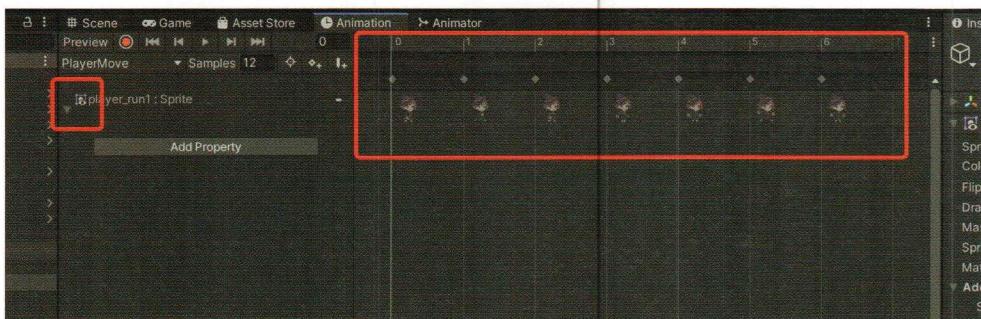
先ほど7枚の画像アセットをドラッグ＆ドロップして作ったゲームオブジェクトを選択して、[Window] メニューから [Animation] → [Animation] を選択してください。 「Animation」(アニメーション) ウィンドウが開きます。

「Animation」 ウィンドウは Animation Clip の内容を表示するウィンドウです。全体が見えにくい場合はウィンドウのサイズを調整するといいでしょう。

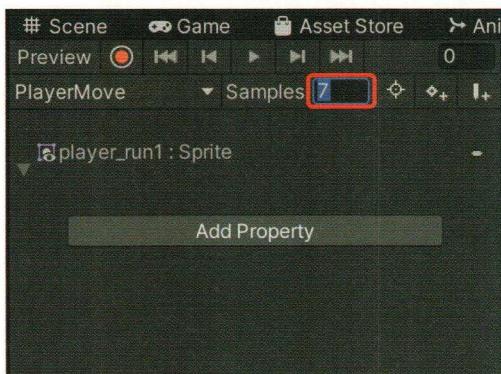
また、「Animation」 ウィンドウはタブ化して Unity ウィンドウに組み込むこともできます。左上のタブをつかんで、シーンビュー や プロジェクトビュー などにドラッグ & ドロップしてみてください。個人的には、シーンビュー のエリアにタブ化しておくと使いやすいです。



それでは、「Sprite」と書かれている箇所の左側にある、三角形のボタンをクリックしてください。右側のビューに登録されているスプライトが時間軸に沿って表示されます。時間の表示スケールはマウスのスクロールホイールを上下させたり、タッチパッドを上下させたりすることで変更できるので、ちょうどいい見え方に調整しましょう。



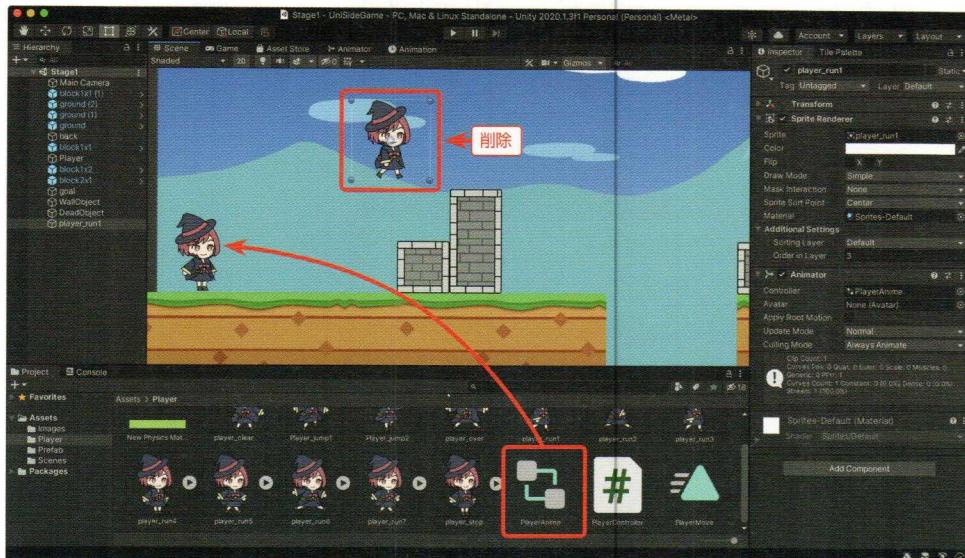
ここで、「Samples」という項目に注目してください。もし表示されていない場合は、ウィンドウ右上のボタンから [Show Sample Rate] を選択しましょう。



これは1秒間に何コマ表示するのかを決める値です。ここでは「12」になっています。これは「1秒間に12コマ表示する」という意味です。この走りアニメーションは7コマなので、1秒間に1.7回ループすることになります。少し速すぎるので、「Samples」の値を「7」に書き換えて、[Return] キーを押しましょう。7に変更されます。

これで1秒間に7コマのアニメーションになり、1秒でちょうど1ループすることになります。ちょうどいいと思う速度になるよう調整してみてください。

今回はアニメーションデータを作るためにゲームオブジェクトを作りました。シーンビューに配置されたゲームオブジェクトは必要ないので削除してください。



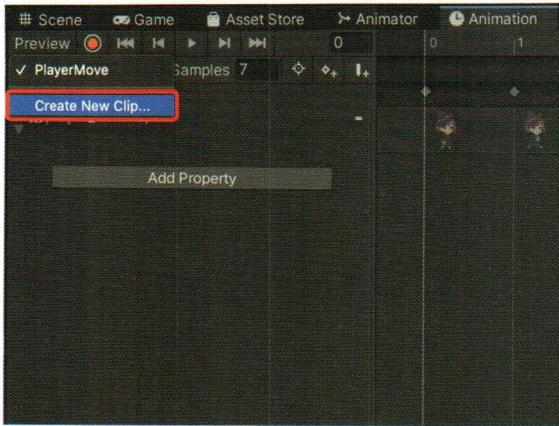
最後に、プロジェクトビューに残されたアニメーターコントローラーをヒエラルキービューまたはシーンビューのPlayerにドラッグ＆ドロップしてアタッチしてください。これで元々配置されていたプレイヤーキャラクターのゲームオブジェクトに、このアニメーターコンポーネントがアタッチされます。

ジャンプアニメーションを作る

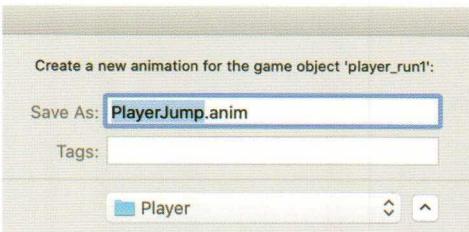
続いて、ジャンプ中のアニメーションを作ってみましょう。移動アニメーションは複数の画像アセットをシーンにドラッグすることで自動的に作りましたが、今回は手動で作ってみましょう。

プレイヤーキャラクターを選択して、「Animation」ウィンドウを開いてください。

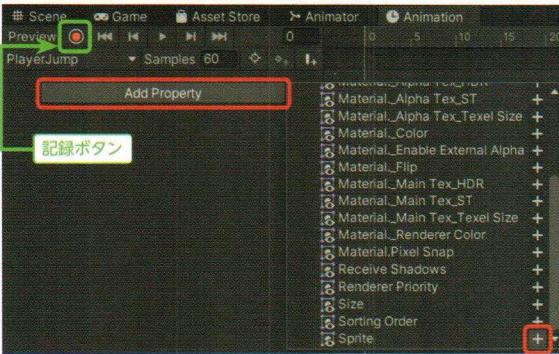
参照 → 「Animation ウィンドウ」 [121 ページ](#)



すると、PlayerMoveの移動アニメーションが1つだけ設定されているはずです。左上のメニューから、[Create New Clip...] を選択してください。



アニメーションクリップを保存するダイアログが表示されます。「PlayerJump」と名前を付けてPlayerフォルダーに保存しましょう。

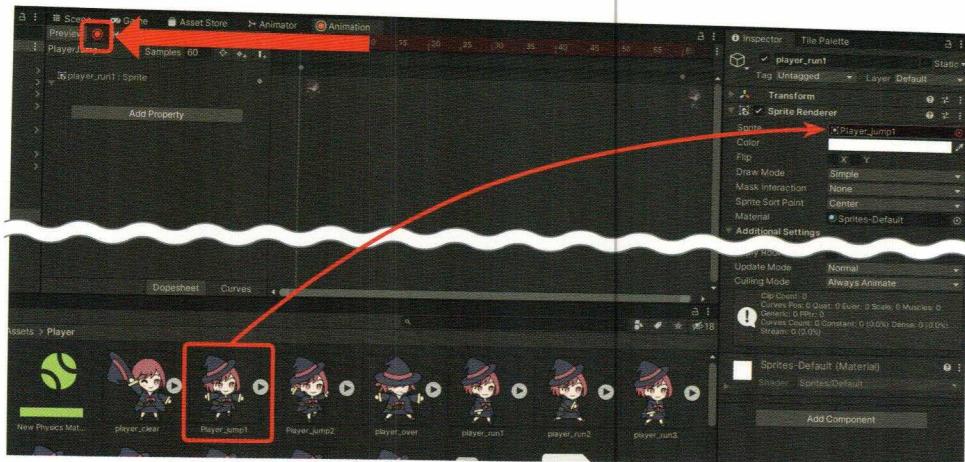


プロジェクトビューに新しいアニメーションクリップが追加され、「Animation」 ウィンドウが空の状態になります。[Add Property] ボタンをクリックすると追加メニューが開くので、そこから[Sprite Renderer] → [Sprite] の右にある [+] ボタンを押してください。これでスプライトを切り替えるコ

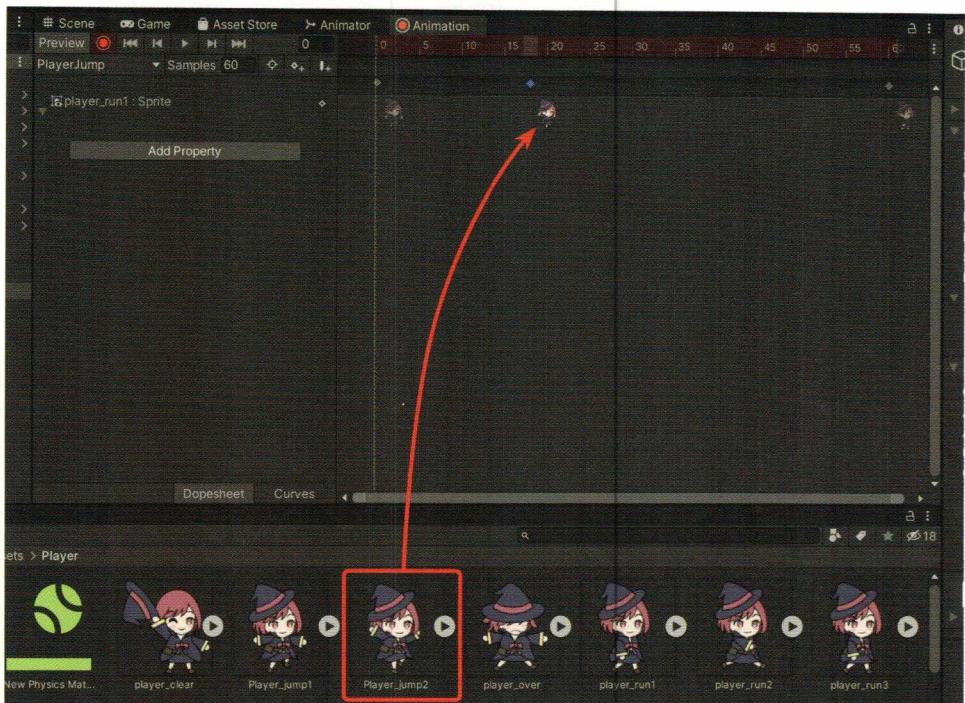
マアニメーションを作ることができます。

現在、Sprite Rendererコンポーネントに使われている画像を使って1秒間のアニメーションが作られています。この画像をジャンプ用の画像に差し替えましょう。ウィンドウ左上にある記録ボタンを押すことで記録モードとなり、画像の変更ができるようになります。

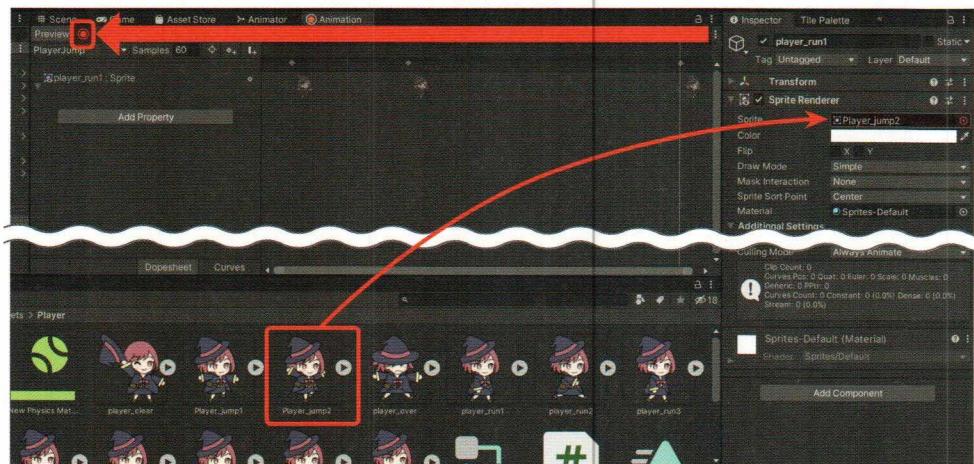
Sprite Rendererコンポーネントを見ると、「Sprite」の項目が赤くなっています。そこにジャンプのパターンである「Player_jump1」をドラッグ&ドロップしましょう。これでアニメーションの画像が変更されます。



ジャンプアニメーションは2コマで用意しているので、もう1コマ追加しましょう。まずプロジェクトビューの「Player_jump2」を追加したいタイムライン上にドラッグ＆ドロップします。これでその位置にコマが1つ追加されます。ここでは0.20秒に追加しています。このキーフレームはあとで移動させることもできます。今は好きなところにキーフレームを追加してみてください。



最後のコマも同じく「Player_jump2」に変更しておきます。タイムライン上部で最後のコマを選択します。記録ボタンを押して記録モードにし、Sprite Rendererコンポーネントの「Sprite」に「Player_jump2」をドラッグ＆ドロップしてください。

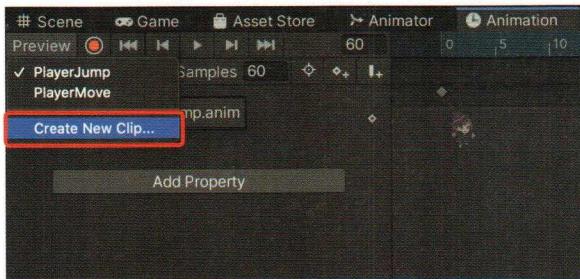


待機、ゴール、ゲームオーバーのアニメーションを作ろう

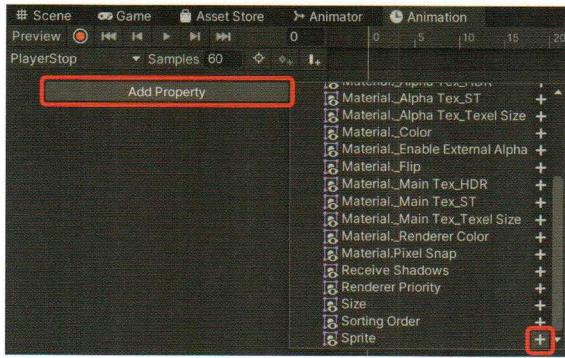
さらに、「移動せずに停止している状態」である待機アニメーションと、ゴール時とゲームオーバー時にポーズを付けるアニメーションを作りましょう。これらはそれぞれ1コマだけのアニメーションとしてアニメーションクリップを作ります。

まずは待機アニメーションクリップを作ります。キャラクターを選択して、「Animation」ウィンドウを開いてください。

参考 → 「Animation ウィンドウ」 [121 ページ](#)



「Animation」ウィンドウ左上のプルダウンメニューから「[Create New Clip...]」を選択して新しいアニメーションクリップを作ります。名前は「PlayerStop」としておきましょう。



それから [Add Property] ボタンをクリックして、Sprite Rendererから「Sprite」の右側にある〔+〕ボタンをクリックしてライトアニメーションを作りましょう。

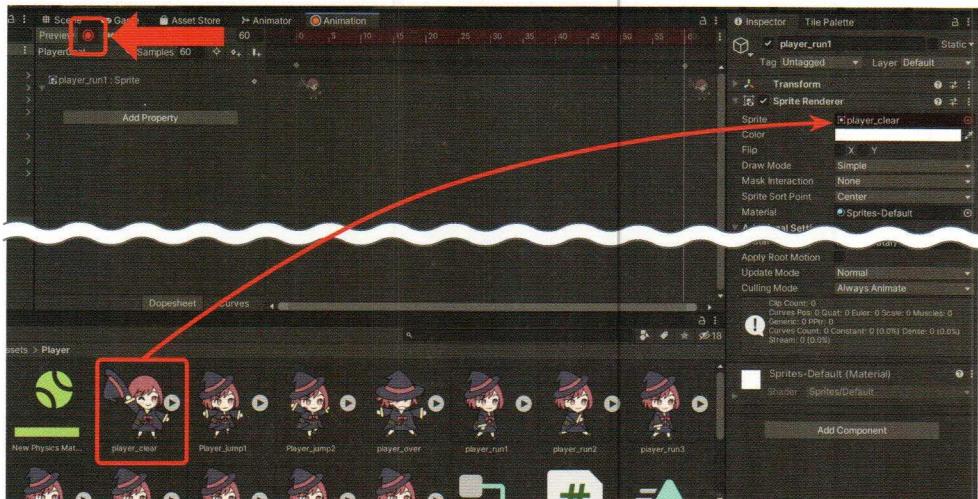
プレイヤーキャラクターの画像は元々待機用の画像 (player_stop) を使っています。そのため、

ジャンプのように変更する必要はなく、これで完成です。



ゴールのアニメーションクリップも待機と同じように作りましょう。使用する画像は「player_clear」です。アニメーションクリップの名前は「PlayerGoal」にしておきます。

記録ボタンを押し、キーフレームとして登録されている画像2つを「player_clear」に変更します。

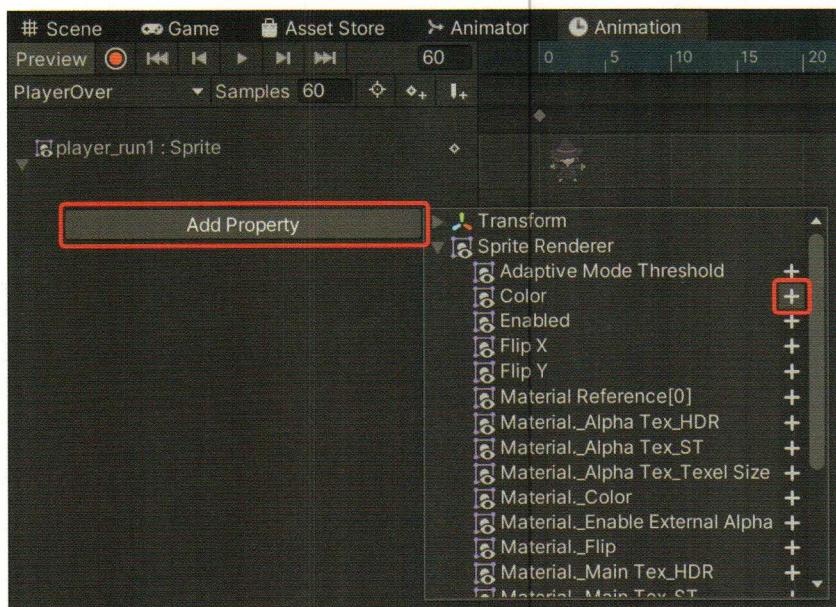




続いて、ゲームオーバーになったときのアニメーションも作りましょう。アニメーションクリップの作り方は「PlayerGoal」と同じです。アニメーションクリップの名前は「PlayerOver」をしておきます。ゲームオーバーの画像は「player_over」を使ってください。

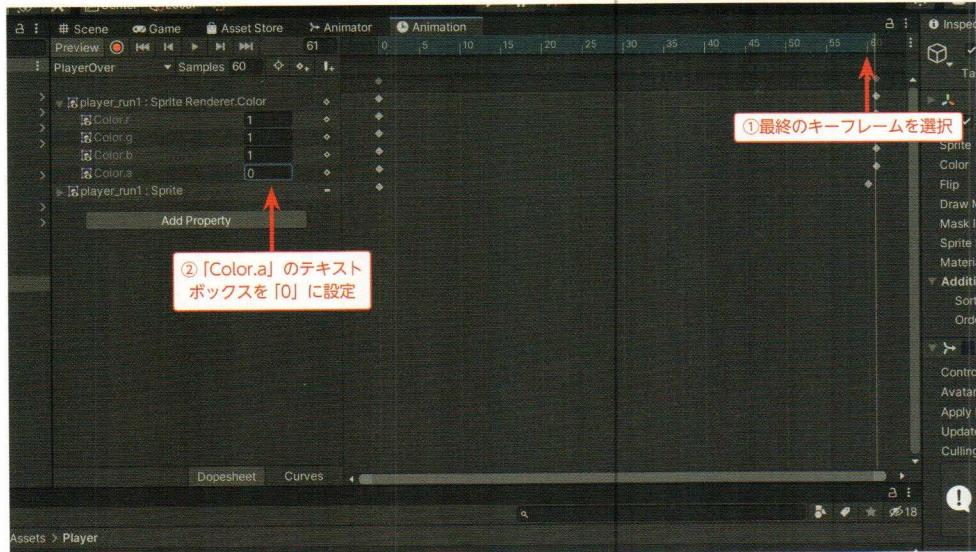
ゲームオーバーはスプライトアニメーションの他に、カラーの透明度をアニメーションさせてフェードアウトしていく演出を加えることにしましょう。

[Add Property] ボタンをクリックして、Sprite Rendererから「Color」の右側にある[+]ボタンを押してカラーアニメーションを追加してください。これでアニメーションにカラーが追加され、Sprite Rendererコンポーネントのカラーをアニメーションさせることができます。



ここでは、カラーを以下の手順で設定してみてください。

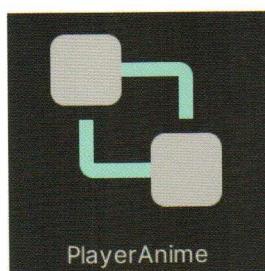
- ① タイムをクリックして、最終のキーフレームを選択します
- ② 「Color.a」のテキストボックスを「0」に設定します



Color.a は不透明度の設定です。ここを「0」に設定することで指定の時間（この場合なら1秒）かけてゲームオブジェクトが透明になっていきます。

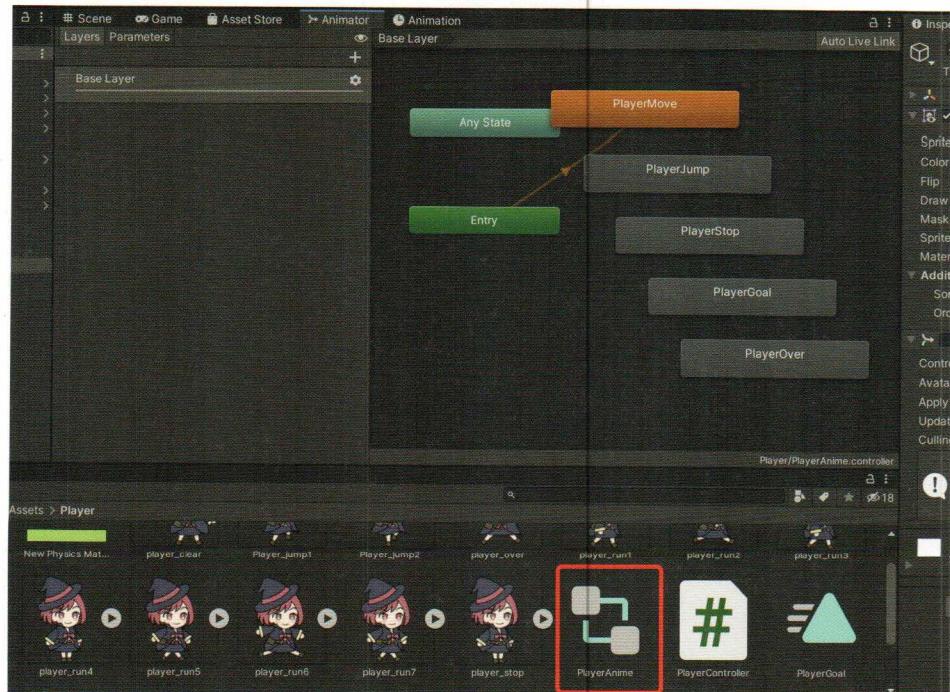
プレイヤーのアニメーションを切り替えよう

ここまでで、「移動」「ジャンプ」「待機」「ゴール」「ゲームオーバー」のアニメーションクリックができます。これらをアニメーターコントローラーを編集し、動作や操作に合わせて切り替えられるようにしましょう。



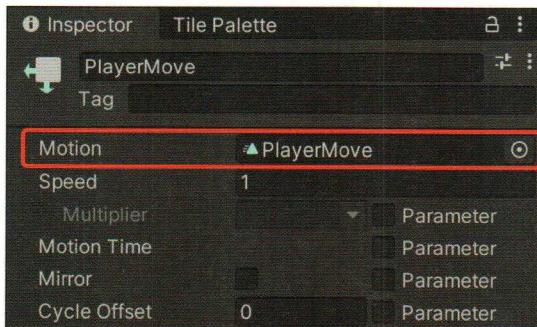
アニメーターコントローラー（PlayerAnime と名前を付けて保存したファイルです）をダブルクリックして開いてください。シーンビューの表示が切り替わり、アニメーターコントローラーが表示されます。

今まで作ったアニメーションクリップが四角いアイコンとして表示されています。これが「アニメーターコントローラー」のデータをUI化したものです。



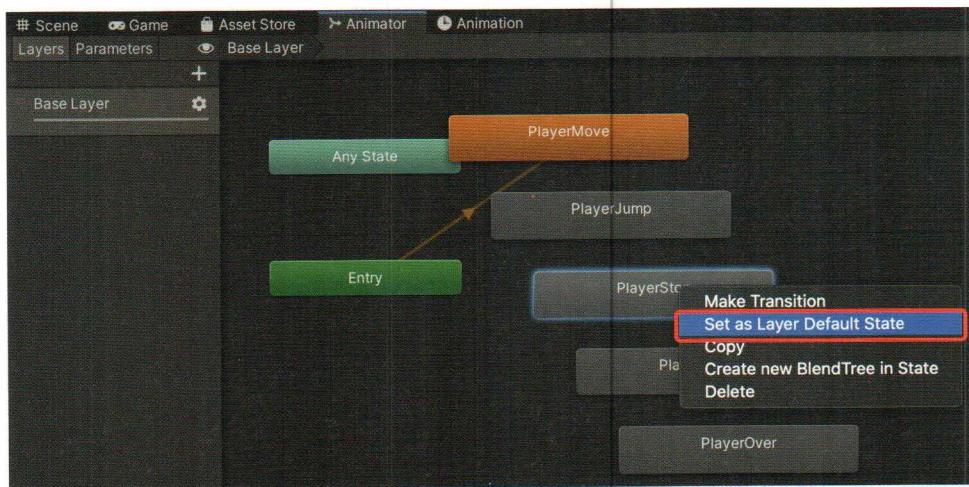
「Entry」と「PlayerMove」が矢印でつながっていますね。そのうち「Entry」はアニメーションが始まったときの開始点です。この画面はつまり、「開始直後に PlayerMove アニメーションクリップが再生される」ということを表しているわけです。

このように、アニメーターコントローラーはアニメーションクリップどうしを矢印でつなぐことでアニメーションの切り替えを行うことができるアニメーションエディターです。

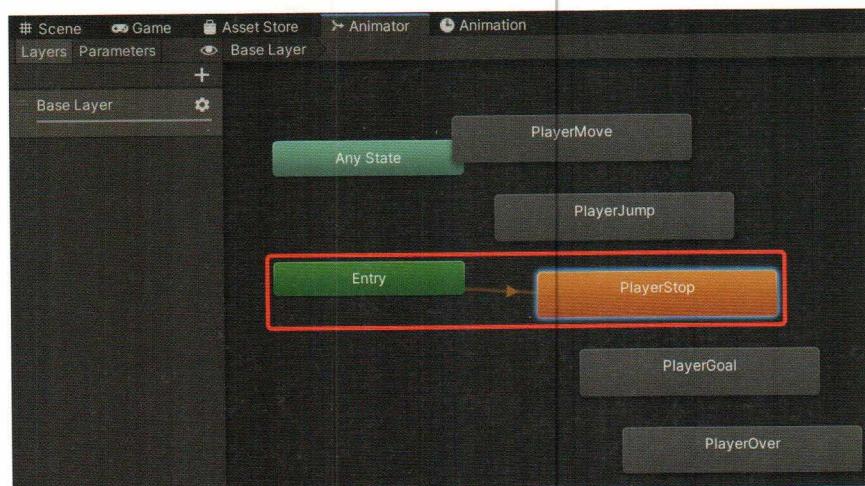


それでは、オレンジのアイコン(「PlayerMove」)を選択して、インスペクタービューを見てみましょう。「Motion」という箇所にアニメーションクリップが設定されています。これが再生されるアニメーションクリップです。

それでは、プレイヤーキャラクターのアニメーションを待機から開始されるようにしましょう。「PlayerStop」を選択してマウスを右クリックするとメニューが表示されるので、[Set as Layer Default State] を選択してください。



「Entry」からの矢印が「PlayerStop」に移動しオレンジ色に変わりました。これでプレイヤーキャラクターのアニメーションは「PlayerStop」(待機) から開始されるようになります。



アニメーションさせるためにスクリプトを変更しよう

それでは、アニメーションを切り替えるためのスクリプトを書いていきます。PlayerControllerスクリプトを変更するのでPlayerControllerを開いてください。ハイライト部が、PlayerControllerの変更内容です。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerController : MonoBehaviour
{
    Rigidbody2D rbody; // Rigidbody2D 型の変数
    float axisH = 0.0f; // 入力
    public float speed = 3.0f; // 移動速度

    public float jump = 9.0f; // ジャンプ力
    public LayerMask groundLayer; // 着地できるレイヤー
    bool goJump = false; // ジャンプ開始フラグ
    bool onGround = false; // 地面に立っているフラグ

    // アニメーション対応
    Animator animator; // アニメーター
    public string stopAnime = "PlayerStop";
    public string moveAnime = "PlayerMove";
    public string jumpAnime = "PlayerJump";
    public string goalAnime = "PlayerGoal";
    public string deadAnime = "PlayerOver";
    string nowAnime = "";
    string oldAnime = "";

    // Start is called before the first frame update
    void Start()
    {
        // Rigidbody2D を取ってくる
        rbody = this.GetComponent<Rigidbody2D>();
        // Animator を取ってくる
        animator = GetComponent<Animator>();
        nowAnime = stopAnime;
        oldAnime = stopAnime;
    }

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

```

void FixedUpdate()
{
    ~ 省略 ~

    if (onGround)
    {
        // 地面の上
        if (axisH == 0)
        {
            nowAnime = stopAnime; // 停止中
        }
        else
        {
            nowAnime = moveAnime; // 移動
        }
    }
    else
    {
        // 空中
        nowAnime = jumpAnime;
    }

    if (nowAnime != oldAnime)
    {
        oldAnime = nowAnime;
        animator.Play(nowAnime); // アニメーション再生
    }
}

// ジャンプ
public void Jump()
{
    ~ 省略 ~
}

// 接触開始
void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.tag == "Goal")
    {
        Goal(); // ゴール！ !
    }
    else if (collision.gameObject.tag == "Dead")
    {
        GameOver(); // ゲームオーバー！ !
    }
}

// ゴール
public void Goal()
{
    animator.Play(goalAnime);
}

```

```
// ゲームオーバー
public void GameOver()
{
    animator.Play(deadAnime);
}
}
```

◆ 変数

まず、変数が8つ追加されています。`animator`変数はAnimatorコンポーネントを保持する変数です。それ以下の`string`型の変数は先ほど作ったアニメーションデータを切り替えるためのパラメーターナンバーを定義したものです。作ったアニメーションクリップと同じ名前にしておいてください。いずれも、`public`を付けておくことで、あとからUnityエディター上で変更ができるようにしています。

◆ Start メソッド

`Start`メソッドでは、`GetComponent`メソッドを使って`animator`変数に値を入れています。

◆ FixedUpdate メソッド

`FixedUpdate`メソッドでは、`onGround`（地上にいるフラグ）と`axisH`（移動値）を`if`文でチェックして、アニメーション名を`nowAnime`変数に設定しています。

その際、もし前のフレームとアニメーション名が違うなら、Animatorコンポーネントの`Play`メソッドでアニメーションを再生しています。`Play`メソッドは、引数としてアニメーションクリップ名を指定することで、そのアニメーションを再生してくれるメソッドです。

◆ OnTriggerEnter2D メソッド

`OnTriggerEnter2D`メソッドは何かが`Collider`に接触したときに呼ばれるメソッドです。引数の`collision`が接触した`Collider`コンポーネントで、それが持つ`gameObject`変数が「`Collider`コンポーネントがアタッチされているゲームオブジェクト」です。さらに、ゲームオブジェクトが持つ`tag`変数は、「設定されているTag」です。`tag`が「Goal」であれば`Goal`メソッドを呼び、「Dead」であれば`GameOver`メソッドを呼んでいます。

◆ Goal メソッド／GameOver メソッド

ゴールとゲームオーバーのアニメーションを外部から更新できるように`public`を付けたメソッドにあります。`Goal`メソッドでは、ゴールのアニメーションに切り替えます。`GameOver`メソッドでは、ゲームオーバーのアニメーションに切り替えています。

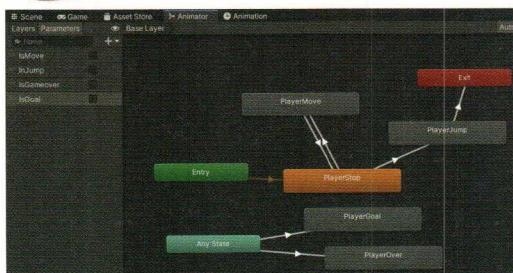


イベント当たり判定のメソッド

`OnTriggerEnter2D` メソッド以外に、`OnTriggerStay2D`、`OnTriggerExit2D` という当たり判定のメソッドがあります。この3つのメソッドは、「In Trigger」にチェックを付けたコライダーが他のコライダーに接触したとき (`OnTriggerEnter2D`)、接触中 (`OnTriggerStay2D`)、接触終了時 (`OnTriggerExit2D`) に呼ばれるメソッドで、引数の `collision` 変数は接触したコライダーコンポーネントです。



アニメーションの切り替え



今回はスクリプトで `Animator` クラスの `Play` メソッドを呼ぶことでアニメーションの切り替えをしていますが、Unity のアニメーションシステムには、アニメーターコントローラー内でアニメーションクリップの切り替わりと、切り替え条件の

変数を設定し、その変数をスクリプトで利用して切り替えを制御できる仕組みがあります。

アニメーションパターンがもっとたくさんあり、複雑な切り替えを必要とする場合は便利ですが、今回のようにパターンが少ない場合は、スクリプトだけで切り替えを行うほうがシンプルなため使用していません。

ゲーム終了判定スクリプトを書こう

最後にゴール当たり（「Goal」タグの付いたゲームオブジェクト）とゲームオーバー当たり（「Dead」タグの付いたゲームオブジェクト）に接触したときに対応するように、`PlayerController` スクリプトを変更しましょう。ハイライト部が、`PlayerController` の変更内容です。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerController : MonoBehaviour
{
    ~ 省略 ~
```

```
public static string gameState = "playing"; // ゲームの状態

// Start is called before the first frame update
void Start()
{
    ~ 省略 ~

    gameState = "playing"; // ゲーム中にする
}

// Update is called once per frame
void Update()
{
    if (gameState != "playing")
    {
        return;
    }

    ~ 省略 ~

}

void FixedUpdate()
{
    if (gameState != "playing")
    {
        return;
    }

    ~ 省略 ~
}

// ジャンプ
public void Jump()
{
    ~ 省略 ~
}

// 接触開始
void OnTriggerEnter2D(Collider2D collision)
{
    ~ 省略 ~
}

// ゴール
public void Goal()
{
    animator.Play(goalAnime);

    gameState = "gameclear";
    GameStop(); // ゲーム停止
}
```

```
// ゲームオーバー
public void GameOver()
{
    animator.Play(deadAnime);

    gameState = "gameover";
    GameStop(); // ゲーム停止
    // =====
    // ゲームオーバー演出
    // =====
    // プレイヤー当たりを消す
    GetComponent<CapsuleCollider2D>().enabled = false;
    // プレイヤーを上に少し跳ね上げる演出
    rbody.AddForce(new Vector2(0, 5), ForceMode2D.Impulse);
}

// ゲーム停止
void GameStop()
{
    // Rigidbody2D を取ってくる
    Rigidbody2D rbody = GetComponent<Rigidbody2D>();
    // 速度を 0 にして強制停止
    rbody.velocity = new Vector2(0, 0);
}
```

それでは、詳しく見ていきましょう。

◆ 変数

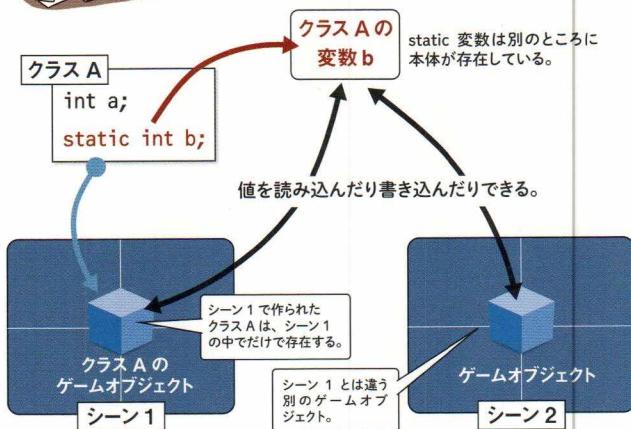
`gameState` はプレイヤーキャラクターの状態を表す `string` 型の変数です。`gameState` の値が "playing" (ゲームプレイ中) でない場合に、「プレイヤーの操作」など一切の処理をできないようにします。この変数には、先頭に `public` と `static` を付けています。このうち `public` は外部から参照するために必要なキーワードでしたね。なお `static` については、次ページの Tips で解説します。

ゲームの状態は以下の 4 つの文字列で表すようにします。

- "playing" : ゲーム中。プレイヤーキャラクターを操作できる状態です
- "gameclear" : ゲームクリア。ゴールに接触した状態です
- "gameover" : ゲームオーバー。「Dead」タグのゲームオブジェクトに接触した状態です
- "gameend" : ゲーム終了。`gameclear` と `gameover` の次に来る状態として使います



終了するまで値が保持される static 変数



static が付けられた変数は「static 変数（静态変数）」と呼ばれます。通常のクラス内にある変数はアッチセスされたゲームオブジェクトのコンポーネントとして存在し、シーンが切り替わればゲームオブジェクトと一緒に消えてしまいますが、

static 変数はクラスそのものに所属し、ゲーム全体を通して存在する変数となります。つまり、static 変数に書き込まれた値はゲーム全体が終了されるまで消えないのです。

gameState 変数には public を付けているので、外部からもアクセスできるようになっています。このような static 変数に外部からアクセスするには、

```
PlayerController.gameState  
(クラス名) . (変数名)
```

のように、クラス名と変数名をドットでつないで書きます。

◆ Start メソッド

gameState 変数は、Start メソッドで "playing"（ゲームプレイ中）で初期化しています。 gameState 変数は static 変数のため、ゲームの状況によって書き換わり、ゲームが終了するまでそのままになっているので、Start メソッドで初期化する必要があります。

◆ Update メソッド / FixedUpdate メソッド

ゲームクリア、ゲームオーバーになった場合、gameState が "playing" 以外になります。ゲームが終了した場合、キャラクターを操作したり移動させたりする必要がなくなります（むしろできないほうがいい）。

そのため、Update メソッドと FixedUpdate メソッドの先頭で gameState をチェックして、"playing" でなければ return で即座にメソッドを抜けて中断するようにしています。これでキャラクターの操作移動ができなくなります。

◆ Goal メソッド／GameOver メソッド／GameStop メソッド

Goal メソッドと GameOver メソッドはゲーム終了時の処理をまとめたメソッドです。外部からも呼べるように `public` 指定しています。共通処理として、GameStop メソッドを呼んで、移動速度を 0 にして停止させ、`gameState` を設定しています。

ゲームオーバーの演出としては、キャラクターの速度を 0 にし、アタッチしている `CapsuleCollider2D` の `enabled` 変数（bool 型）を `false` にすることで当たり判定を無効にしています、つまり地面当たりをすり抜けるようにしているわけです。

そして、`RigidBody2D` の `AddForce` メソッドを使い、上方向に 5 の力を加え少し跳ね上げています。これによりゲームオーバーになったプレイヤーキャラクターはポーズが切り替わり、少し上にジャンプしたあと、透明になりながら落下して消えていきます。

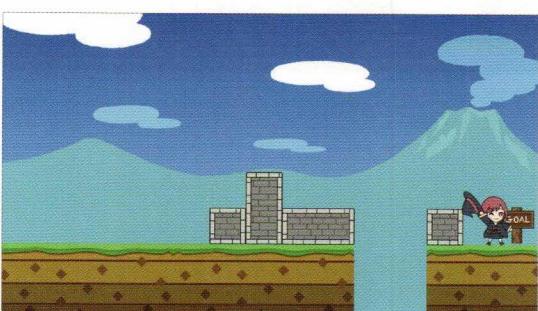
ゲームを実行しよう



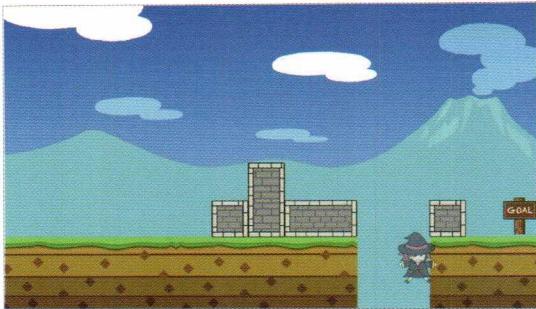
らジャンプします。着地後、入力をなくすと待機ポーズに戻ります。

ここまでできたら、ゲームを実行してプレイヤーキャラクターを動かしてみましょう。

待機ポーズで始まり、パソコンの左右キーで移動アニメーションしながら、左右に移動操作することができます。またスペースキーを押すとポーズがジャンプに切り替わりながら

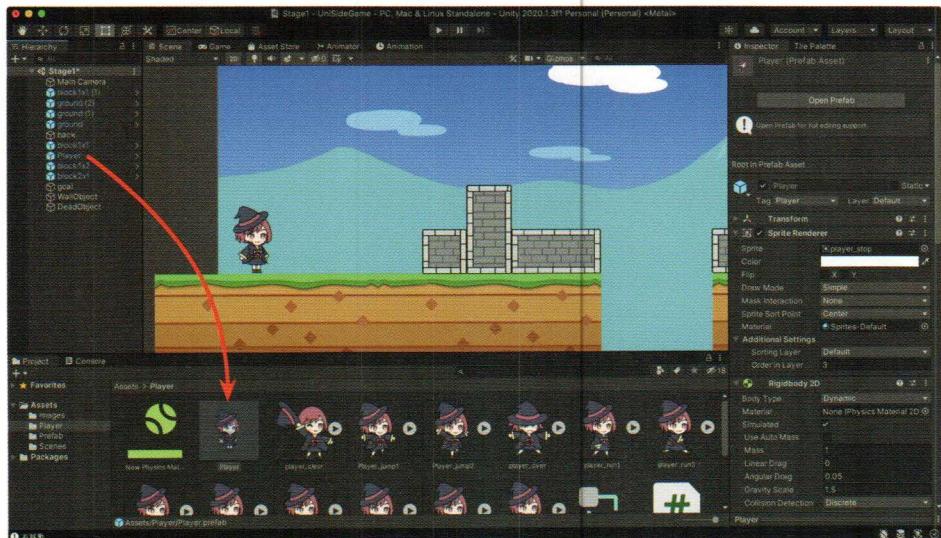


画面右端に配置したゴールに接触すると操作できなくなり、キャラクターはゴールのポーズになります。



真ん中の穴に落ちると、操作できなくなり、キャラクターはゲームオーバーのポーズになります。その後は落下しながら消えていきます。

ここまでで、プレイヤーキャラクターは完成です。ヒエラルキービューのPlayerをプロジェクトビューのPlayerフォルダーにドラッグ＆ドロップしてプレハブ化しておきましょう。



これで他のゲームステージを追加した場合でも、このプレハブを配置することで同じプレイヤーキャラクターのゲームオブジェクトを作ることができるようにになります。