応用テキスト
A02

アニメーション

ver.1.0.0

アニメーション概要
2-1　キャラクターアニメーション
2-2　Tweenアニメーション
2-3　背景＆カメラ

バージョン4.2.1をベースに作成し、4.3.0にも対応しています。

本書はGodotエンジンの普及、発展を目的に
プログラボ教育事業運営委員会が制作したものです。

アニメーション概要

2Dゲーム内でアニメーションを使う要素を考えてみます。
必ずしも記載したノードを使う必要はないですが、特に
AnimationPlayerはゲームオブジェクトのいろいろな状態を
変化させることができるため、使いこなしたいノードです。

ノードだけでなく、Tweenクラスや、シェーダーリソースを
使うことでより表現力の高いアニメーションをつくることが
できます。

スクロールする背景

- ParallaxLayer
- ParallaxBackground
- Parallax2D 4.3

キャラクター・アイテム

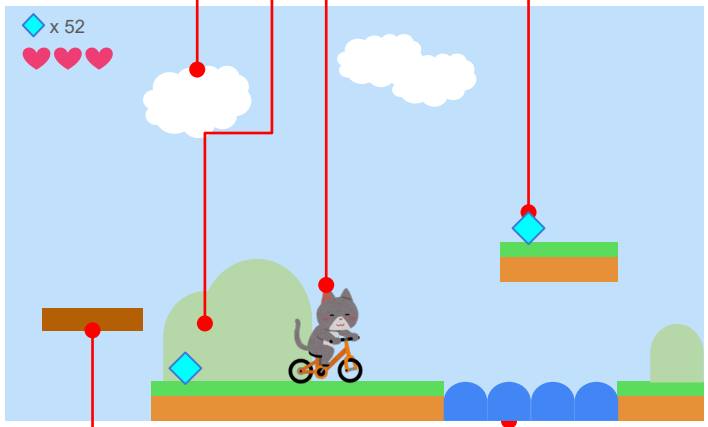
- AnimatedSprite2D
- AnimationPlayer
- AnimationTree

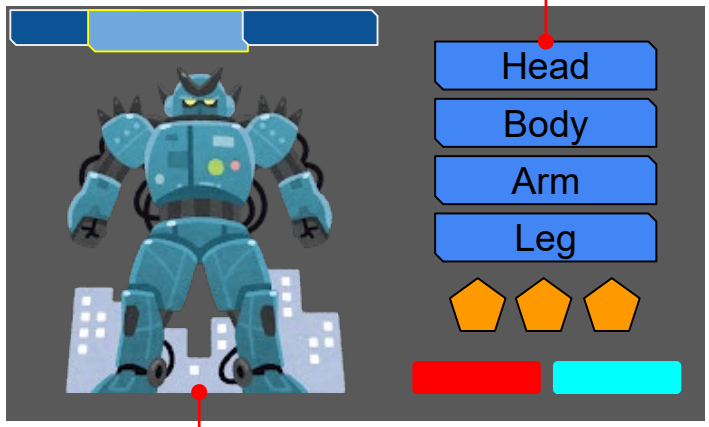
UI（ユーザーインターフェース）

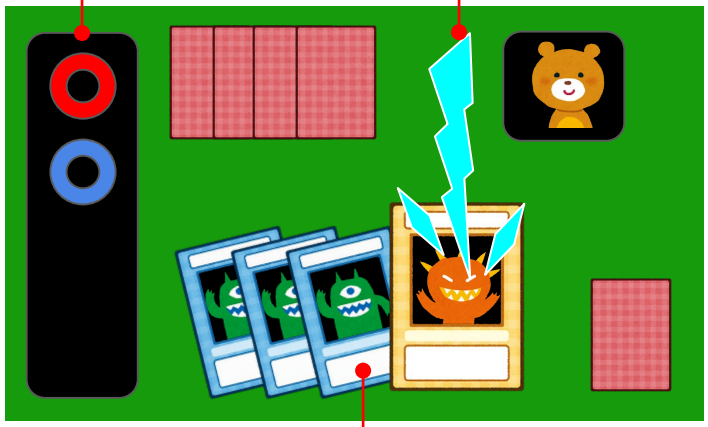
- Tweenクラス
- Button

エフェクト

- AnimationPlayer
- CPUParticles2D
- GPUParticles2D







動く足場

- AnimationPlayer
- Path2D
- PathFollow2D

マップ

- TileMap

画面切り替え効果

- Tweenクラス
- CanvasLayer

ゲームオブジェクト

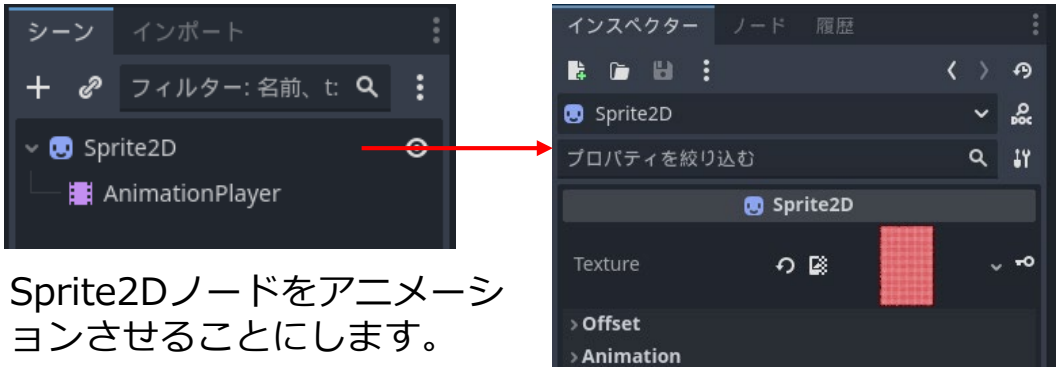
- AnimationPlayer
- Tweenクラス

2-1 キャラクターアニメーション

- AnimationPlayer（１）
- AnimationPlayer（２）
- AnimationPlayer（３）
- Spriteアニメーション（１）
- Spriteアニメーション（２）
- AnimationPlayerを使ったSpriteアニメーション（１）
- AnimationPlayerを使ったSpriteアニメーション（２）
- AnimationPlayerを使ったSpriteアニメーション（３）
- AnimationPlayerの活用（１）
- AnimationPlayerの活用（２）
- AnimationTree【StateMachine】（１）
- AnimationTree【StateMachine】（２）
- AnimationTree【StateMachine】（３）

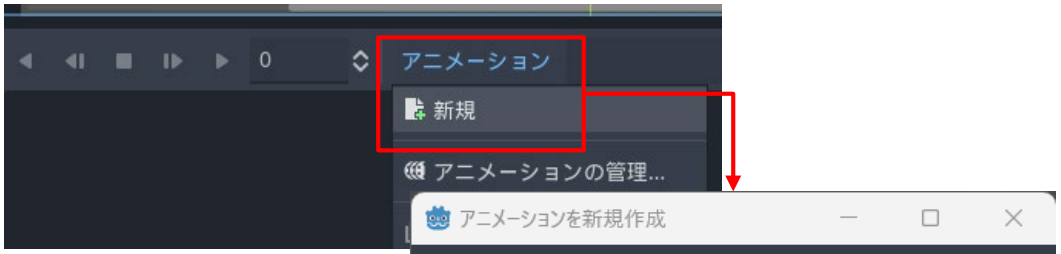
AnimationPlayer（１）

AnimationPlayerの基本的な使い方を解説します。

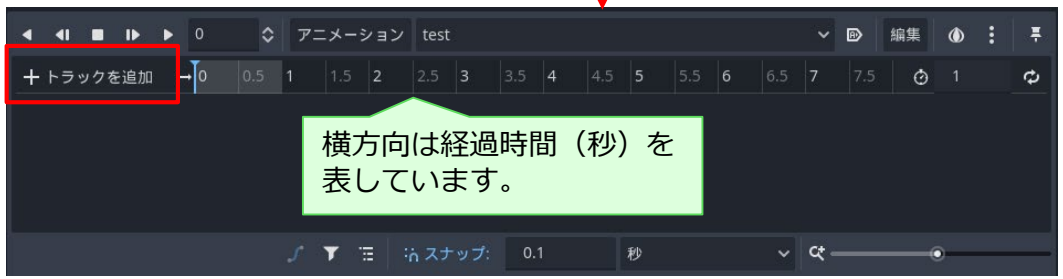


Sprite2Dノードをアニメーションさせることにします。

ボトムパネルのアニメーションメニュー＞新規を選びます。



名前をつけると、タイムラインができあがります。

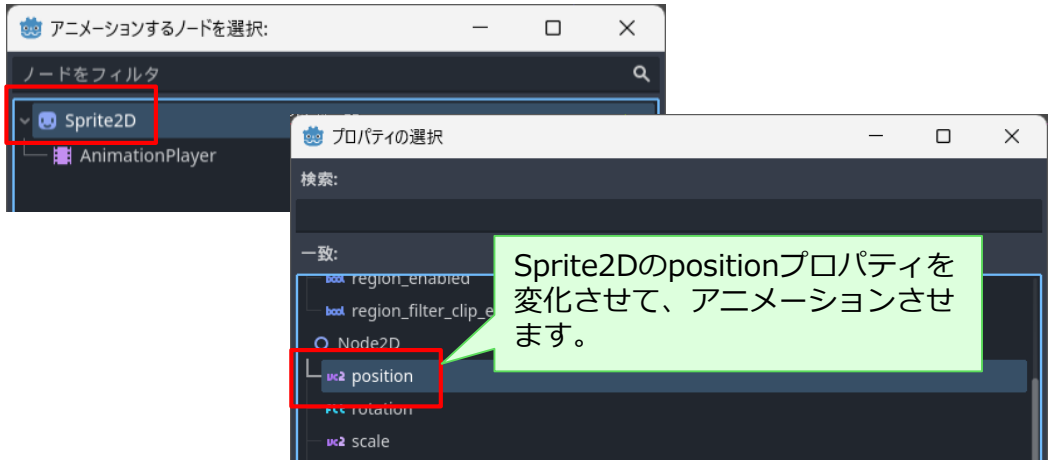


トラックを追加をクリックすると、メニューが表示されます。

トラックとは、何を操作するかで分かれています。プロパティなら位置や回転、サイズなどを操作できますし、メソッド呼び出しはアニメーションの特定のタイミングでメソッドを実行できます。

ここでは、プロパティトラックを選んで進めます。

アニメーションさせるノードと、プロパティを選びます。

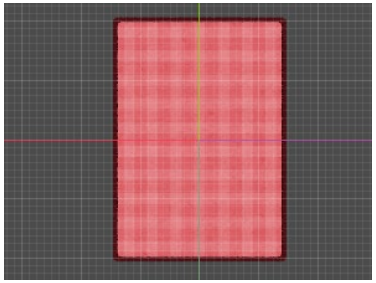


これでアニメーションの準備ができました。

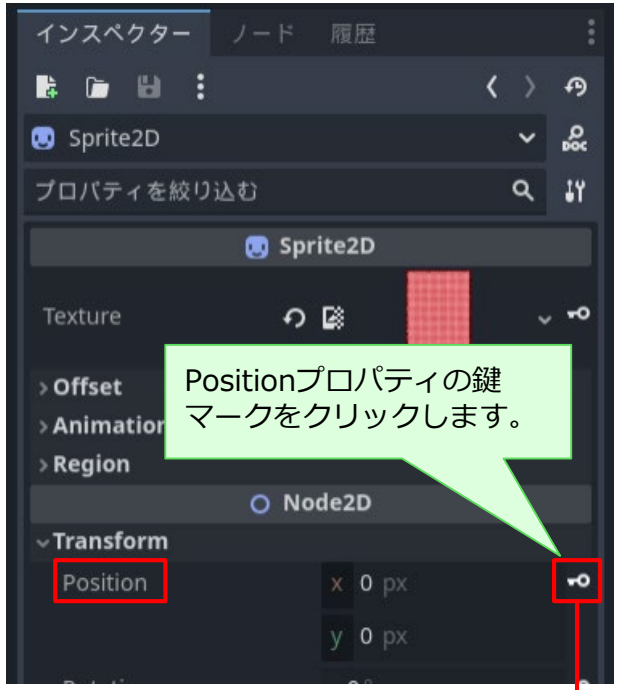
Next Page

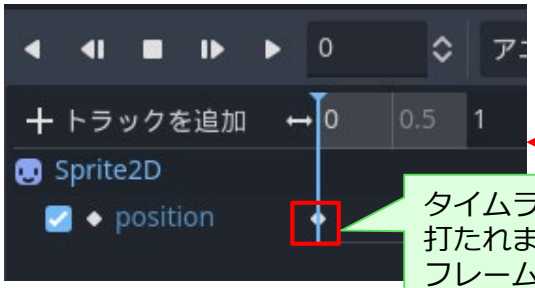
AnimationPlayer（2）

AnimationPlayerでアニメーション設定していきます。

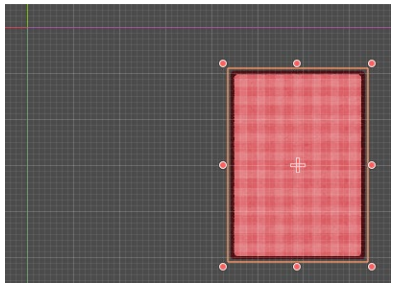


x:0 y:0 の原点に配置されています。



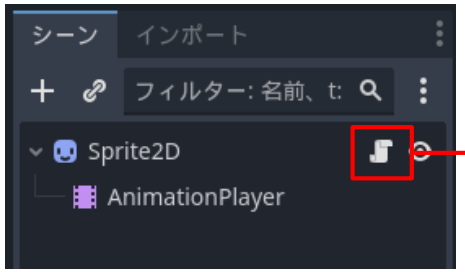


Sprite2Dを画面上で少し離れた位置まで移動させます。位置は適当で良いです。



左上のボタンを使って、アニメーションを再生して動作を確認します。

キーフレームアニメーションは、始点と終点にそれぞれ設定したキーフレームの間は自動的にスムーズに補完するアニメーションがつくられるのが特徴です。



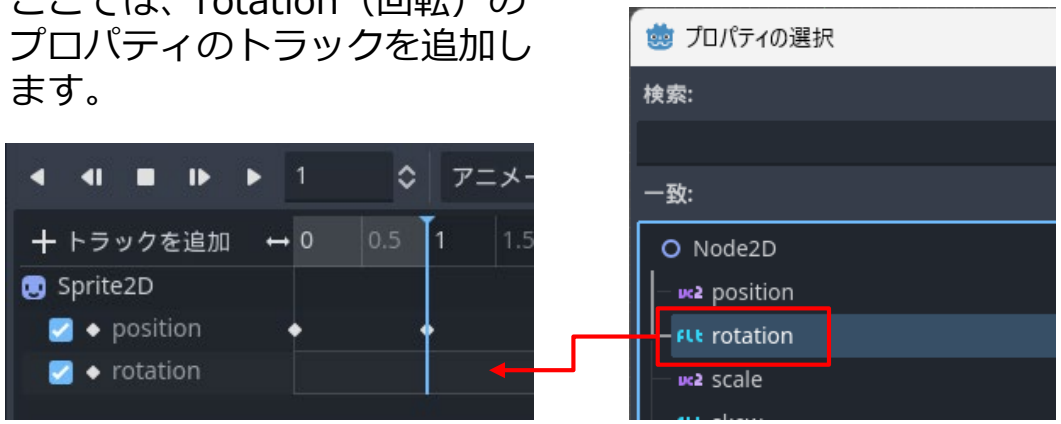

```
$AnimationPlayer.play("test")
```

スクリプトからアニメーションを再生するには、play()メソッドでアニメーションの名前を指定して実行します。

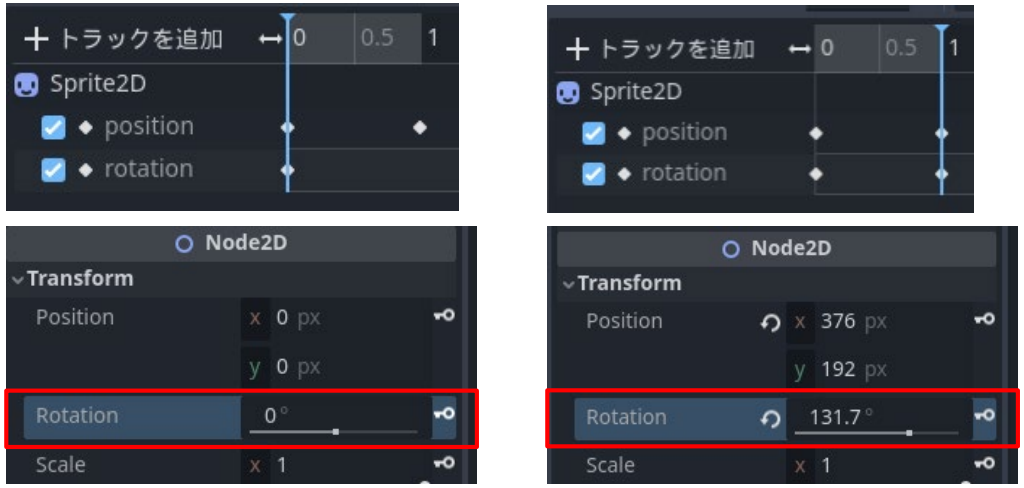
Next Page

AnimationPlayer（3）

トラックを追加して、複合的なアニメーションにします。
ここでは、rotation（回転）のプロパティのトラックを追加します。

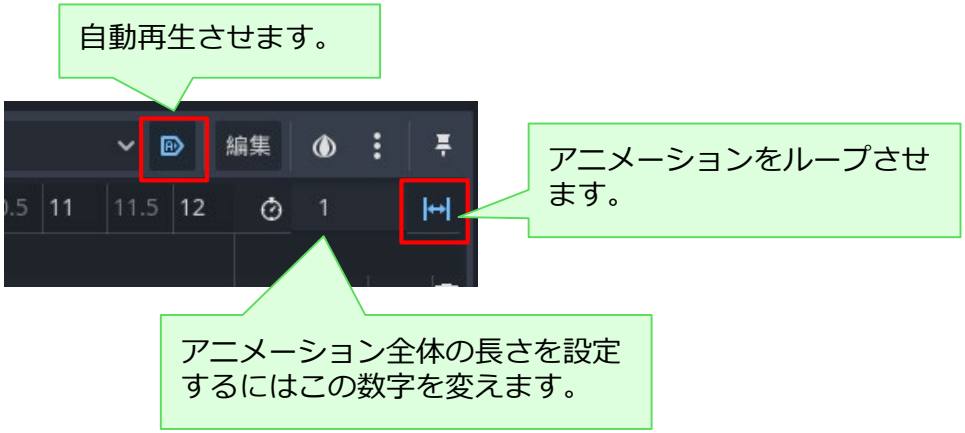


rotationトラックが追加されました。positionのときと同じように、再生ヘッドの位置を動かしてから、キーフレームを設定していきます。



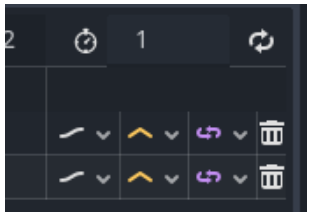
再生すると、回転しながら移動するアニメーションになりました。このようにトラックを組み合わせて複雑なアニメーションをつくっていきます。

実際にシーンを実行した際にすぐにアニメーションをさせる場合や、アニメーションをループさせる場合は右上のボタンで設定します。



キーフレームのポイントはマウスのドラッグで移動させることもでき、アニメーションを微調整することが可能です。

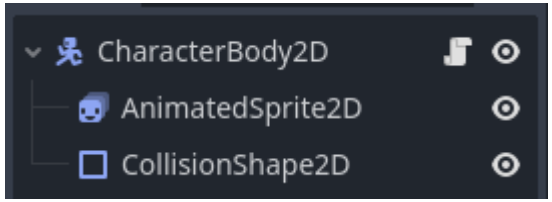
トラック右端には、アニメーションを補完する方法を変更するメニューもあります。いろいろ試してみよう。



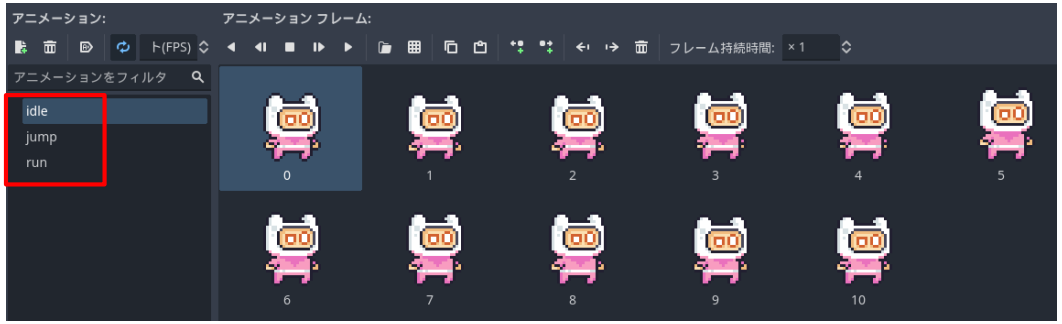
Spriteアニメーション（１）

2Dゲームで手軽にアニメーションをつくることができるのが、AnimatedSprite2Dです。スプライトシートを使ってキャラクターアニメーションを作成します。

CharacterBody2Dノードでプレイヤーキャラクターをアニメーションさせることができます。



素材：https://pixelfrog-assets.itch.io/pixel-adventure-1



この例では、idle（待機）、jump（ジャンプ）、run（走る）の3つの状態のアニメーションを設定しています。

これはプレイヤーキャラクターをアニメーションをさせる、標準的なコードです。

```gdscript
if is_on_floor():
    if velocity:
        if velocity.x > 0:
            $AnimatedSprite2D.flip_h = false
        elif velocity.x < 0:
            $AnimatedSprite2D.flip_h = true
        $AnimatedSprite2D.play("run")
    else:
        $AnimatedSprite2D.play("idle")
else:
    $AnimatedSprite2D.play("jump")
```

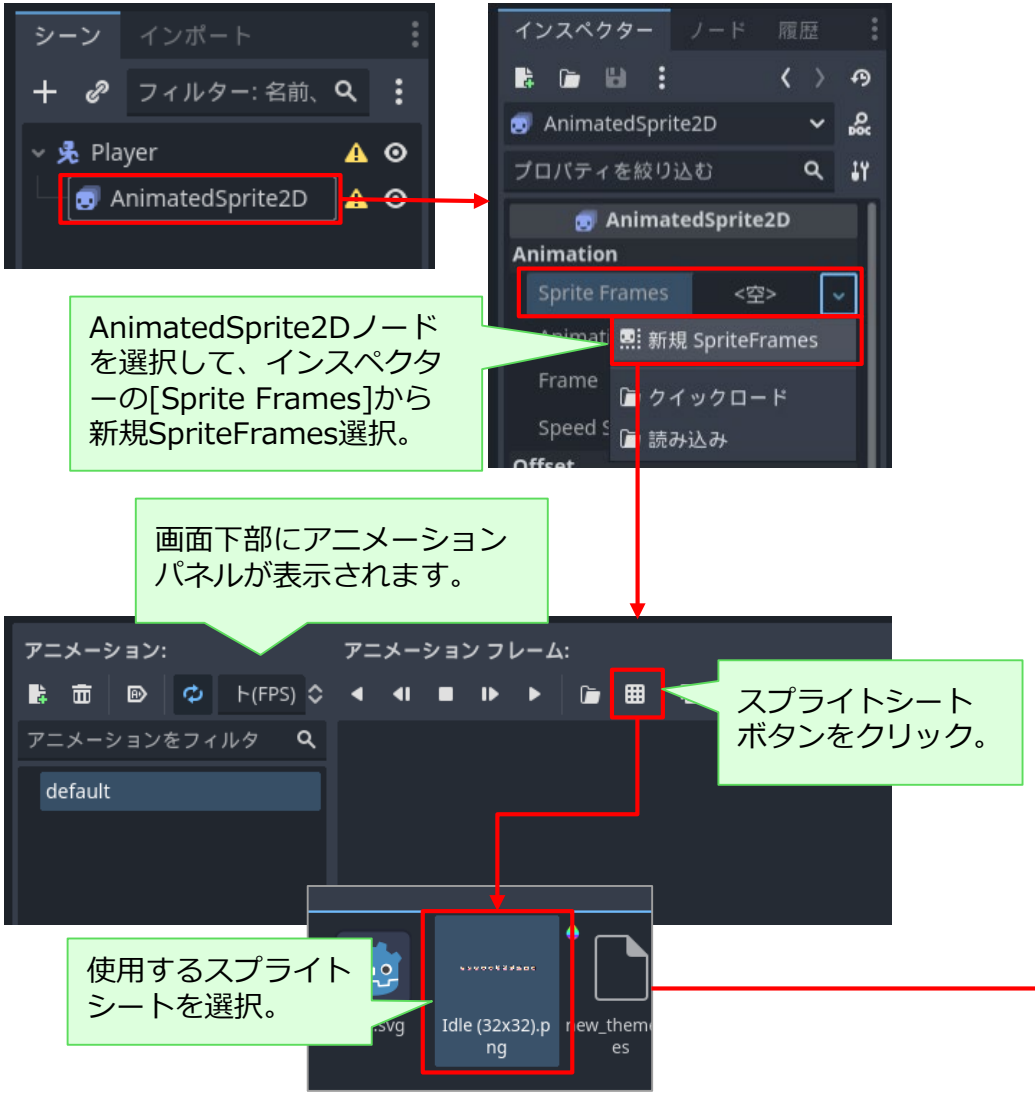
is_on_floor()メソッドは、CharacterBody2Dのメソッドで、キャラクターの足元あるコリジョンシェイプが他のコリジョンシェイプ（地面など）に接触している場合に True を返します。

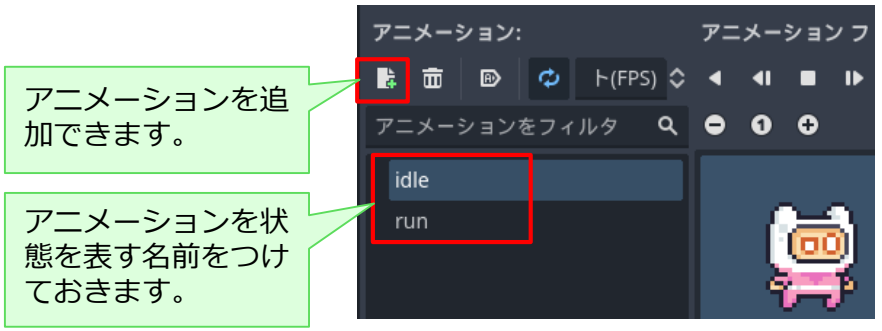
Velocityプロパティは、CharacterBody2Dのプロパティで、X軸（左右の移動）とY軸（上下の移動）の成分を持つ2Dベクトルです。これがゼロの場合は移動をしていないと判定でき、そうでない場合は、X軸のどちらの方向に向いているかを判別できます。

この2つを使ってキャラクターの状態の判定を行い、AnimatedSprite2Dに設定したアニメーションの再生を使い分けています。

Spriteアニメーション（2）

AnimatedSprite2Dを使った、キャラクターアニメーションを作ります。

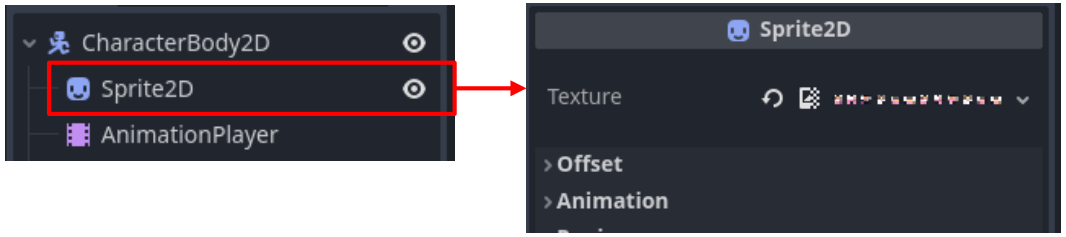




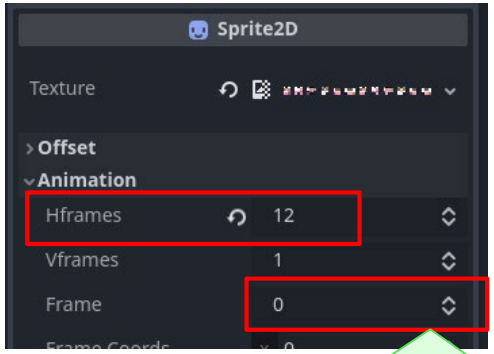
AnimationPlayerを使ったSpriteアニメーション（１）

AnimationPlayerを使っても、AnimatedSprite2Dのようにスプライトシートを使ったアニメーションをつくれます。

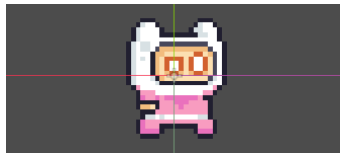
ノードの構成です。Sprite2DにはTextureとしてスプライトシートを割り当てています。



画面上ではこのようになっています。



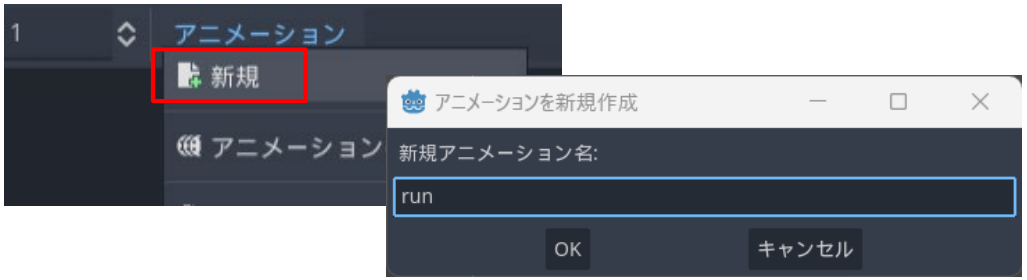
横に12フレームのシートになっているので、Hframesに12を指定します。



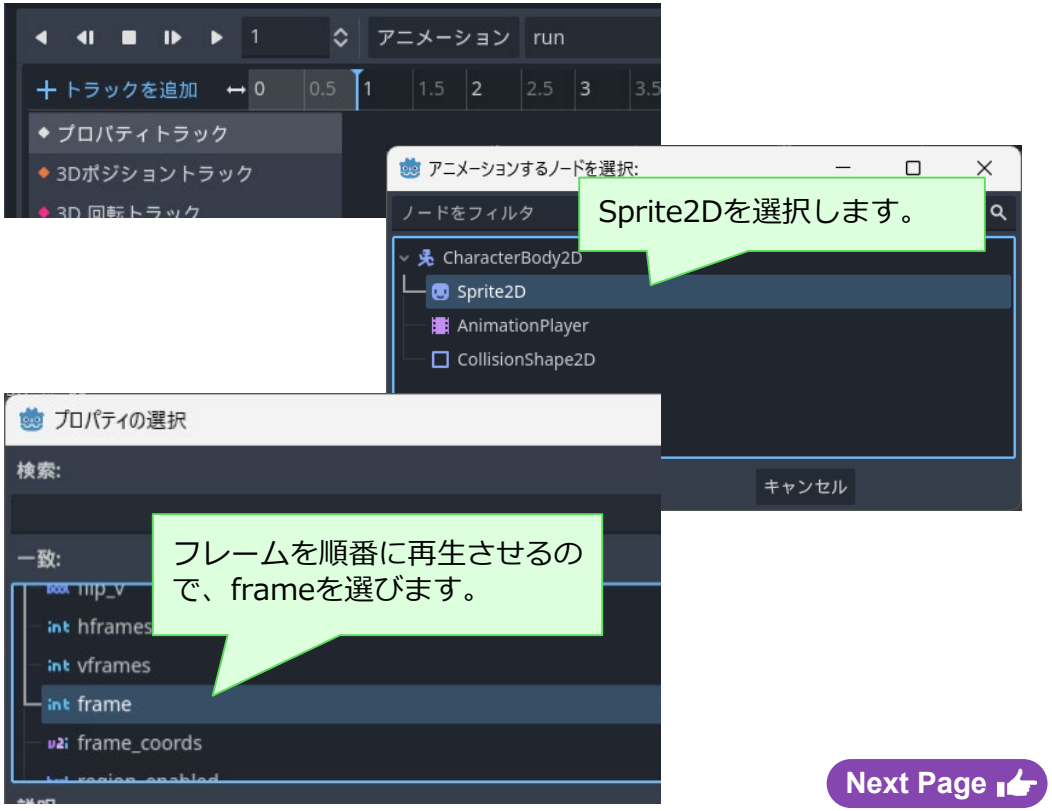
フレーム番号は0から始まります。右端の上下矢印をクリックすると各フレームの画像を確認できます。

これで各画像が、1フレームずつ格納されました。フレームを順番に再生すればアニメーションになります。

アニメーション>新規追加ボタンをクリックします。AnimationPlayerノードがあると、アニメーションを新規作成できるようになります。



アニメーション名を設定すると、タイムラインができあがるので、続いてプロパティトラックを追加します。



Next Page

AnimationPlayerを使ったSpriteアニメーション（２）

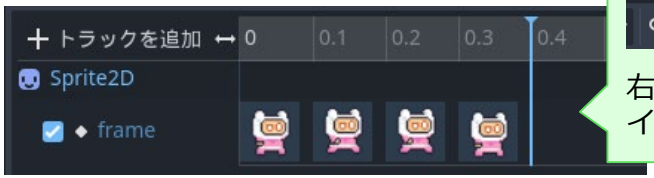
タイムラインにキーフレームを追加していきます。インスペクタードックで操作します。

Sprite2Dノードのインスペクターから、Frameの鍵マークをクリックします。

0フレーム目の画像がキーフレームとして設定されました。

鍵マークを数回クリックするとこのようにキーフレームが追加されていきます。

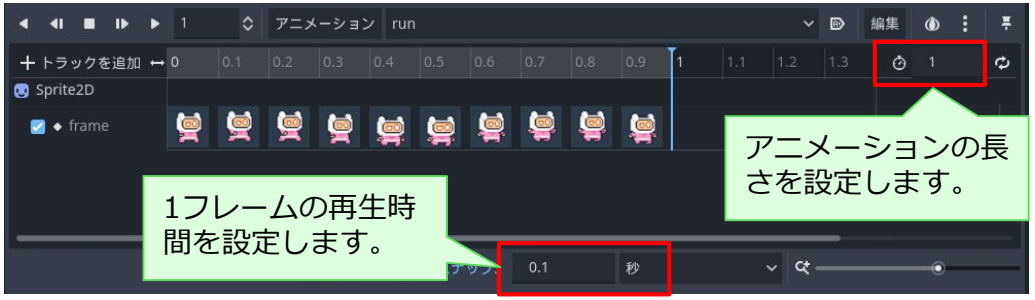
鍵マークをクリックするたびに、フレーム番号は１ずつ増えていくため、順番に次のフレームが追加されます。



右下のスライダーを操作してタイムラインを見やすくします。

4フレーム分追加しました。全部で12フレームあるので残りを追加していきます。

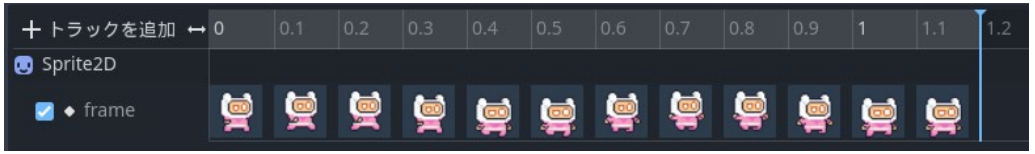
10フレーム分追加しましたが、デフォルトではアニメーションの長さが1秒で、1フレームが0.1秒の設定になっているため、これ以上フレームを増やせられません。



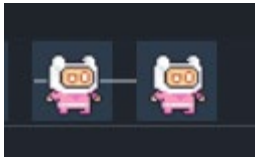
1フレームの再生時間を設定します。

アニメーションの長さを設定します。

アニメーションの長さを1.2秒に変更して、残りの2フレームを追加します。



下図のようにフレーム間にグレーの線が表示された場合は、同じフレーム番号が連続して設定されています。

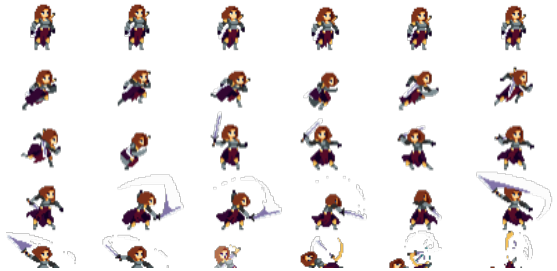


フレームを右クリックしてキーを削除できます。

その場合はキーを削除してから、インスペクターでフレーム番号を選びなおし、キーフレーム追加をやり直します。

AnimationPlayerを使ったSpriteアニメーション（３）

AnimatedSprite2Dと同様に、複数のアニメーションを作成し、Scriptからアニメーションを切り替えて再生することもできます。
スプライトシートに全ての状態が含まれている場合は、それぞれの状態のアニメーションを追加し、適切なフレームを設定することで、複数のアニメーションを作成できます。



素材：
https://clembod.itch.io/warrior-free-animation-set

一方、下記のようにアクションごとにスプライトシートが分かれている場合は、個別に読み込む方法をとります。

Idle（待機）

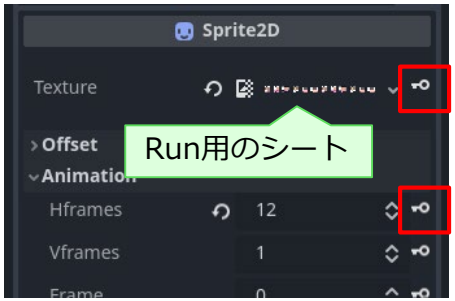
Run（走り）

基本的な作成手順は前ページまでの通りですが、Textureにスプライトシートを設定した後に、TextureとHframesの鍵アイコンをクリックしておきます。

タイムラインではこのようにトラックが追加されます。

その後、frameを前ページの手順通り追加して、idleアニメーションができあがった状態です。

次にrunアニメーションを追加、選択し、Texturesにrun用のスプライトシートを設定し、同じくTextureとHframesの鍵アイコンをクリックします。



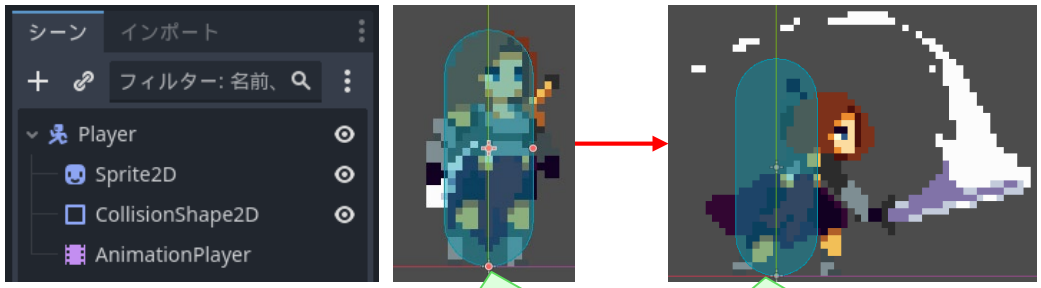
frameを追加してrunアニメーションを設定します。

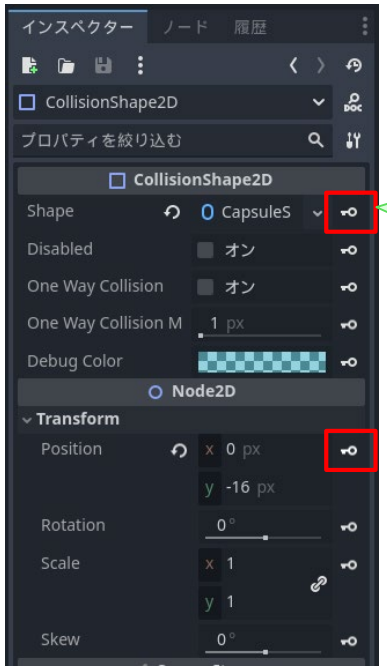
こうすることで、1つのSprite2Dに複数のスプライトシートを設定し、それぞれ個別のアニメーションを作成することができます。

AnimationPlayerの活用（１）

AnimationPlayerを使うと、複数のノードやプロパティをフレームごとに変更することができます。

このケースでは、スプライトのアニメーションによってコリジョンの位置がずれてしまうことを解消するために、フレームごとにコリジョンの形（サイズ）と位置を変更します。



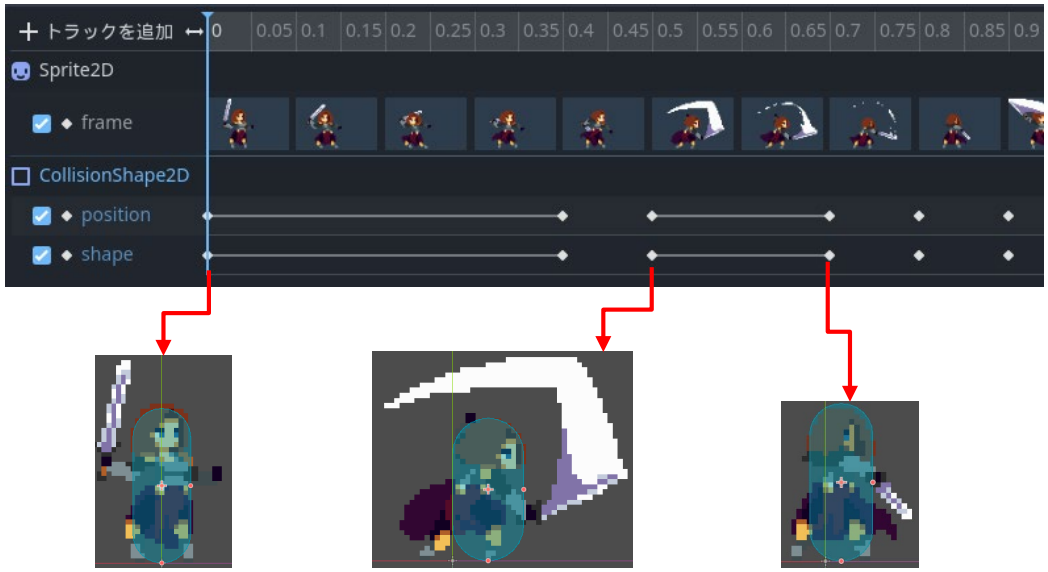


コリジョンのインスペクターでは、各プロパティに鍵アイコンが表示されるため、これでキーフレームを設定します。

コリジョンの形やサイズは、Shapeプロパティを新たに設定することで変更し、位置はPositionプロパティを変更します。

これによりフレームごとに微調整することができます。

実際にコリジョンをフレームごとに変更した例です。



下は追加で剣の部分にコリジョンを設定した例です。プレイヤーの画像上で実際に剣の部分にのみ攻撃判定をつけます。

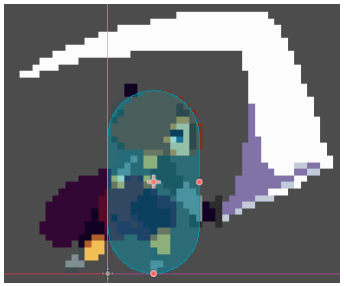
AnimationPlayerを使うことで、様々な状態変化を細かにコントロールすることができます。

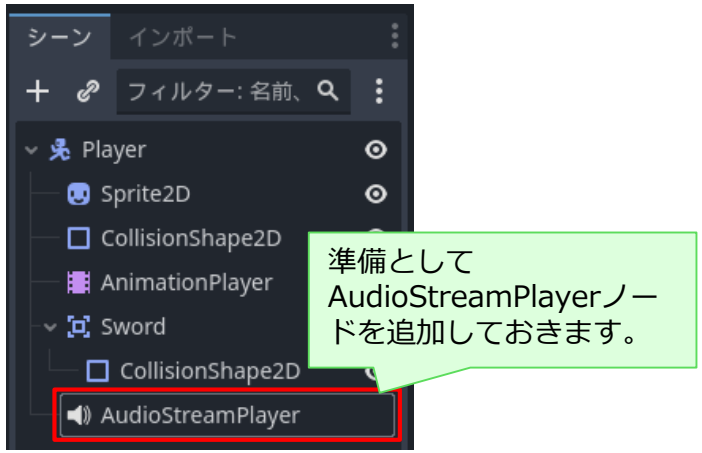
AnimationPlayerの活用（２）

AnimationPlayerはトラックを追加すれば、任意のタイミング（キーフレーム）で、様々なことを実行できます。

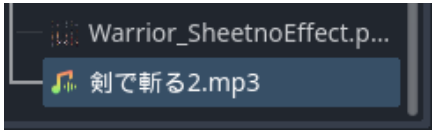
■オーディオ再生トラック

アニメーションの任意のタイミングで効果音を再生させます。
ここでは、剣を振る音を鳴らせてみましょう。



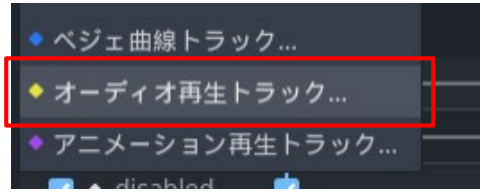


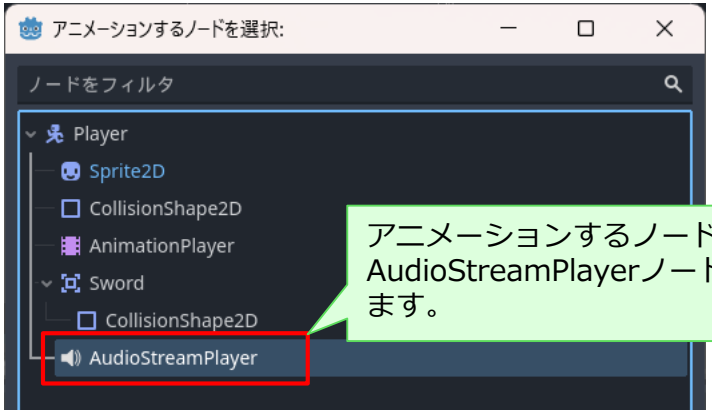
効果音素材もファイルシステム内に準備しておきます。

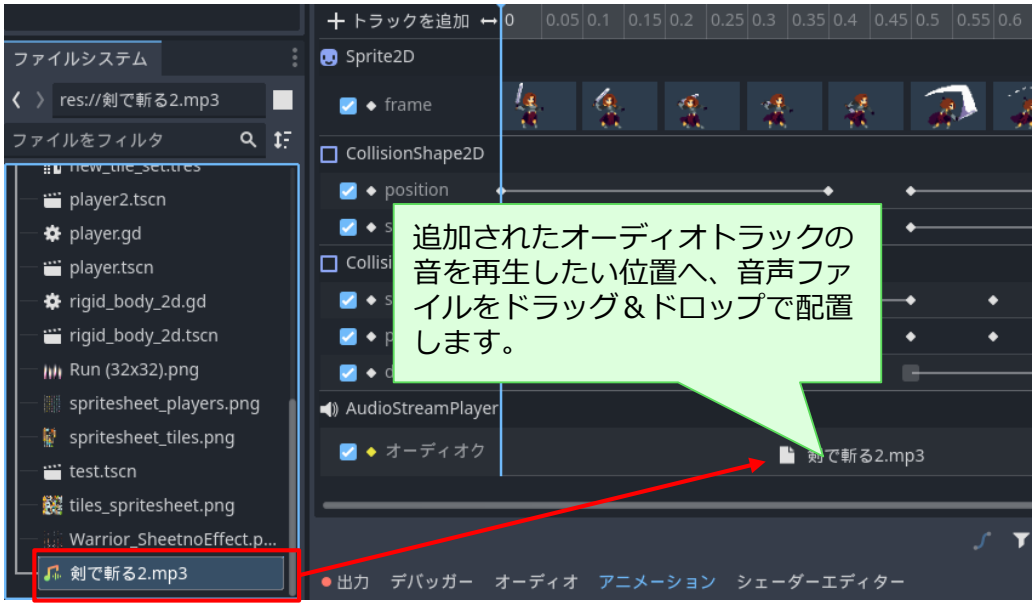


素材：効果音ラボ
https://soundeffect-lab.info

AnimationPlayerにオーディオ再生トラックを追加します。







音声ファイルが配置され、アニメーションのタイミングに合わせて音が再生されます。

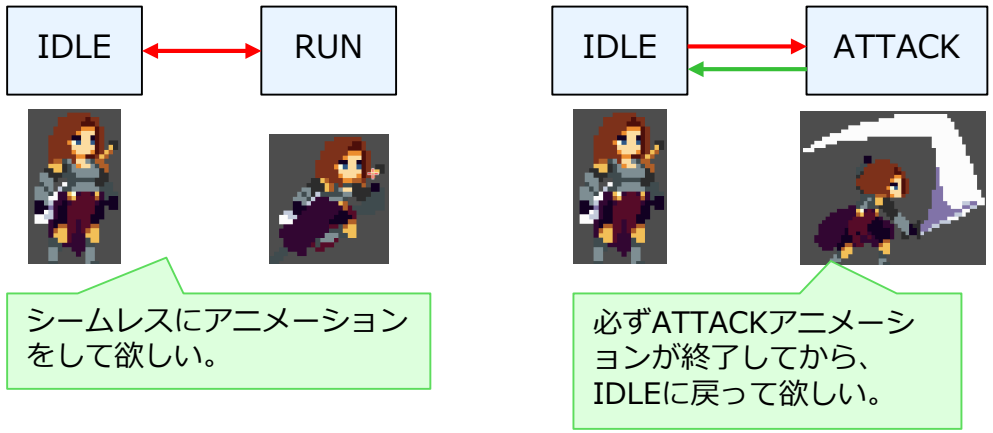
AnimationTree【StateMachine】（1）

AnimationTreeは、AnimationPlayerで設定したアニメーションを制御するためのノードです。

■StateMachine（ステートマシン）

State（ステート）とは「状態」のことで、ステートマシン自体はGodot以外でも使われる用語です。AnimationTreeのStateMachineを使って、複数のアニメーションの状態を制御することができます。

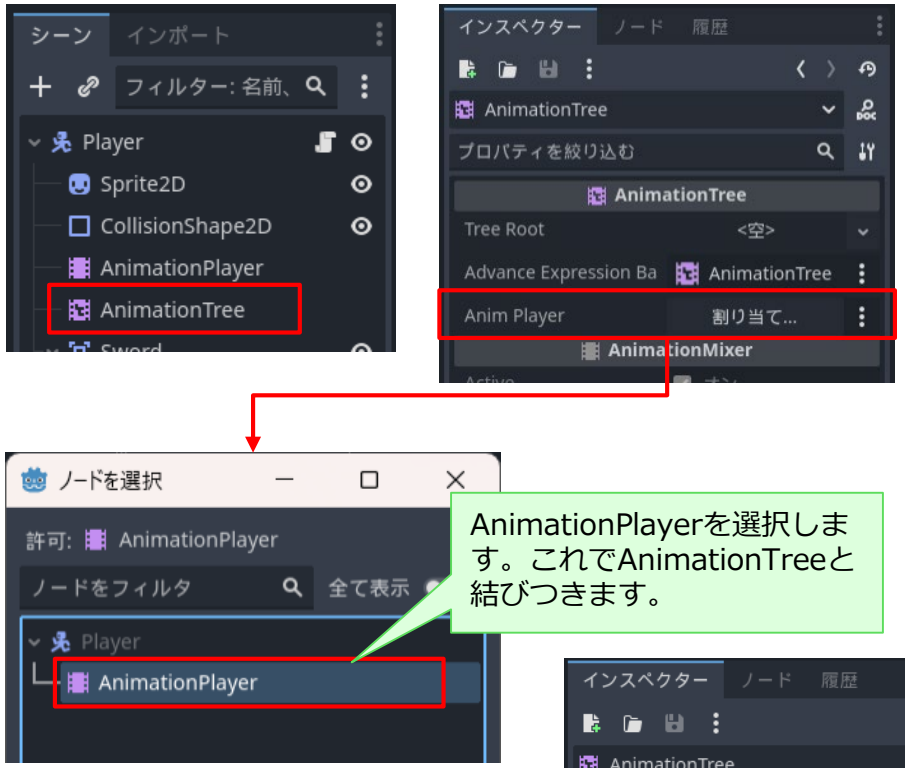
例えば、IDLE（待機）からRUN（走り）状態に移るときは、それぞれのアニメーションが終了せずに、継ぎ目なく切り替わって欲しいですが、攻撃（攻撃）アニメーションは、途中で終了すると困ります。



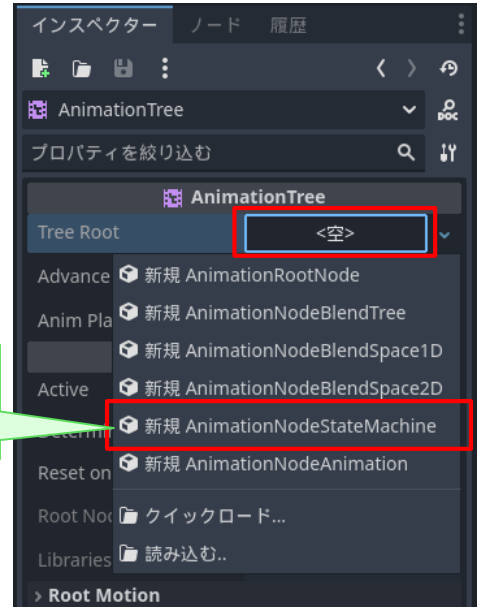
スクリプトでアニメーションの終了状態などを都度確認すれば、できなくはないですが、複雑なコードになってしまいますし直感的ではありません。

AnimationTreeを使えばエディタ画面上で、各アニメーション状態の関係性を設定できるため、管理が分かりやすく簡単になります。

AnimationTreeノードを追加したら、インスペクターのAnimPlayerプロパティの [割り当て...] をクリックします。



インスペクターのTree Rootを設定します。



新規AnimationNodeStateMachineを選択します。

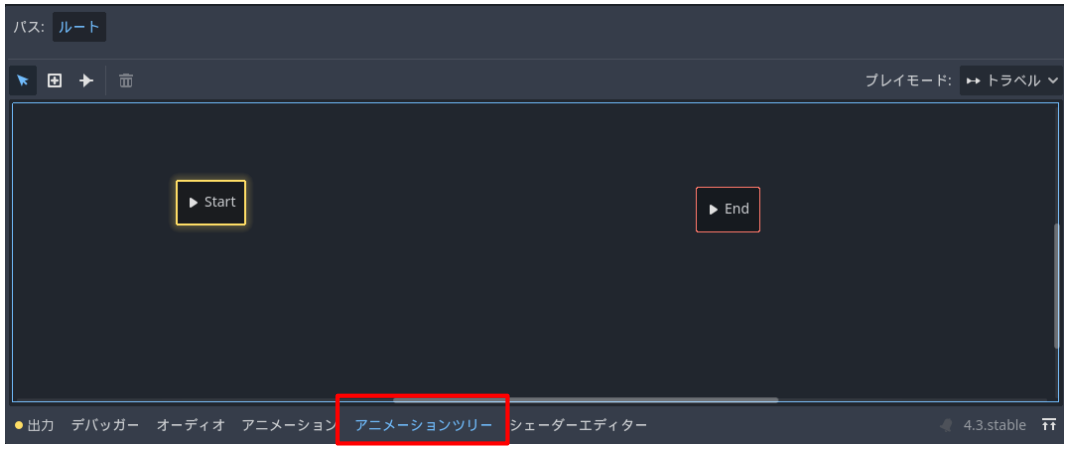
基本の設定はこれで完了です。

Next Page

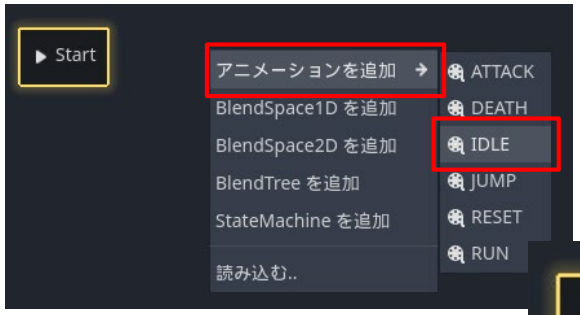
AnimationTree【StateMachine】（2）

AnimationPlayerにいくつかのアニメーションが設定されている状態から進めます。

AnimationTreeノードを選択した状態で、ボトムパネルから「アニメーションツリー」を選択します。このパネルでアニメーション状態の関連付けを設定していきます。



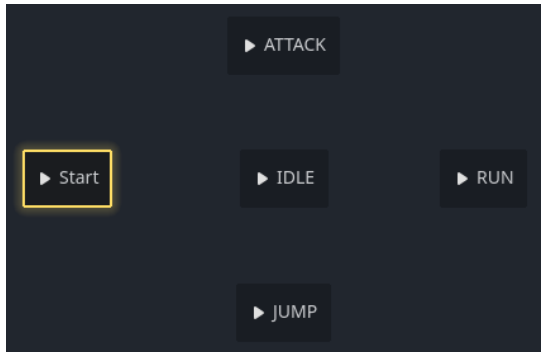
パネル内の適当な位置で右クリックするとメニューが現れるので、[アニメーションを追加] → [IDLE] を選択します。



[IDLE] のボタンのようなものができあがります。
※これもノードとよびます

同じようにして、他の状態を表すノードを追加していきます。

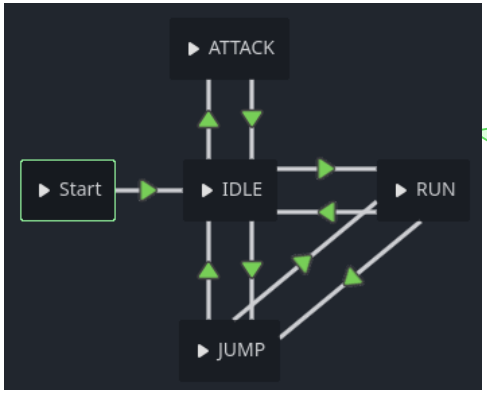
位置はあとで変えられるので、ここでは少し離れている程度で良いです。



左上のボタンを右向き矢印のものに変更します。

ノードからノードへマウスドラッグすると図のように矢印がつながります。

下の図を参考に、ノード間を矢印で結んでいきます。接続されているノード同士のアニメーションは、アニメのつなぎ方を設定できるようになります。



この例だと、RUNとATTACKのアニメーションは、つなぎ目を設定しないことになります。

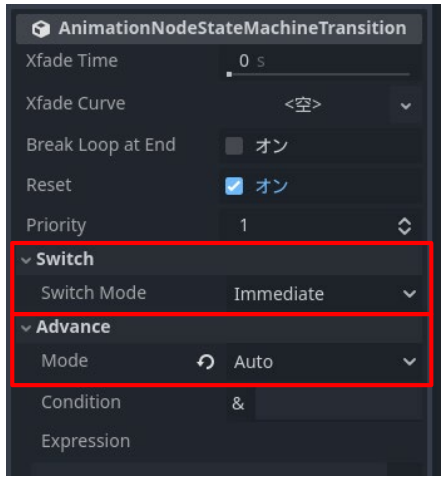
次は接続した各ラインに設定を行っていきます。

Next Page

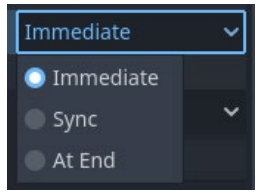
AnimationTree【StateMachine】（3）

各ラインの設定していくため、ノードをつないだラインをクリックして選択します。

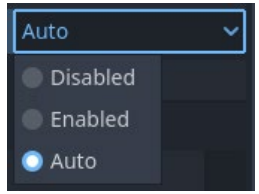
インスペクターにAnimationNodeStateMachineTransitionと表示されたプロパティが現れます。



いくつか項目がありますが、
Switch：状態を変えるタイミング
Advance：詳細な動作
を設定していきます。

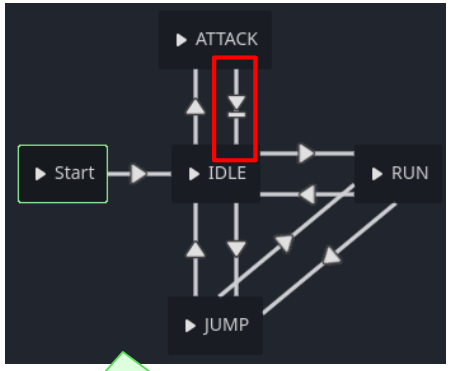


Switch Mode
- Immediate（イミディエイト）
  すぐに次の状態に変更する。
- Sync（シンク）
  すぐに次の状態に変更しつつ、前後の状態を同じタイミングで同期させる。
- At End（アット・エンド）
  現在のアニメーションが終了してから次の状態にする。



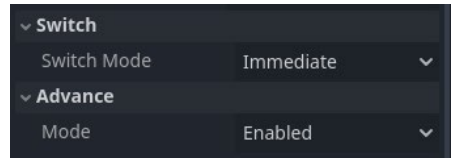
AdvanceのMode
- Disabled（ディスエイブルド）
  無効にする。
- Enabled（イネーブルド）
  有効にする
- Auto（オート）
  自動的に決定する。

ここでは例として以下のような設定にしました。

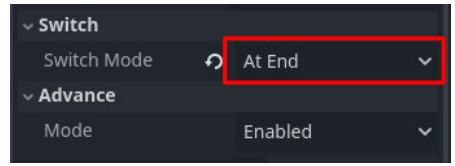


ゲーム自体の動作のルールや、アニメーションのスプライト、ユーザー入力に対するスクリプトによっても、最適な設定内容は変わります。

基本的には、ImmediateとEnabledにしています。



ATTACKからIDLEに戻るラインのみ、アニメーションが終わってからにしたいため、At Endにしています。



スクリプト例です。

```gdscript
@onready var animation_player = $AnimationPlayer
```

```gdscript
func updateAnimation():
    if Input.is_action_pressed("attack") and is_on_floor():
        state_machine.travel("ATTACK")
        return

    if is_on_floor():
        if abs(velocity.x) >= 0.1:
            state_machine.travel("RUN")
        else:
            state_machine.travel("IDLE")
    else:
        state_machine.travel("JUMP")
```

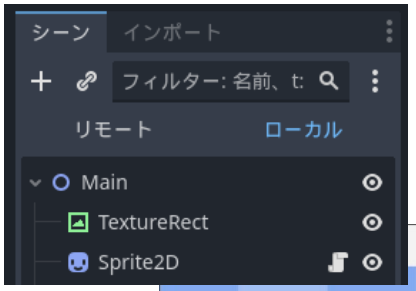
2-2 Tweenアニメーション

- Tweenクラス（１）
- Tweenクラス（２）
- Tweenクラス（３）
- Path（パス）

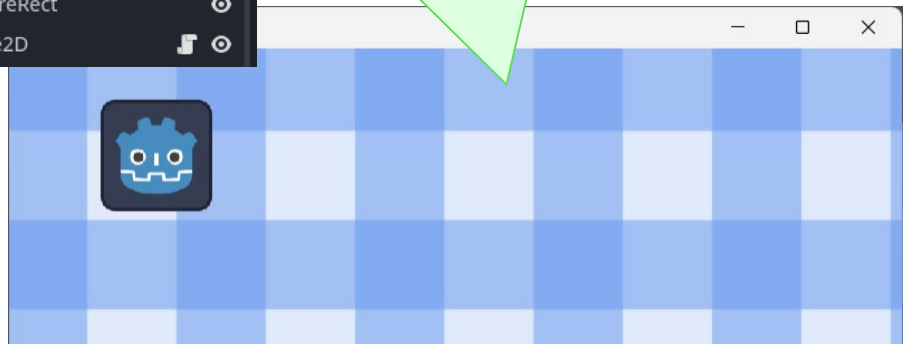
Tweenクラス（1）

AnimationPlayerはあらかじめアニメーションを作成しておき、ゲーム内の状況によってアニメーションを再生させることができます。

TweenクラスはScriptでゲーム内オブジェクトのプロパティを変化させてアニメーションさせることができます。ゲーム内の状況に合わせてリアルタイムにアニメーションをさせる場合などはTweenを使うといいでしょう。



Sprite2Dの変化が分かりやすいように、TextureRectにチェック模様を敷いています。

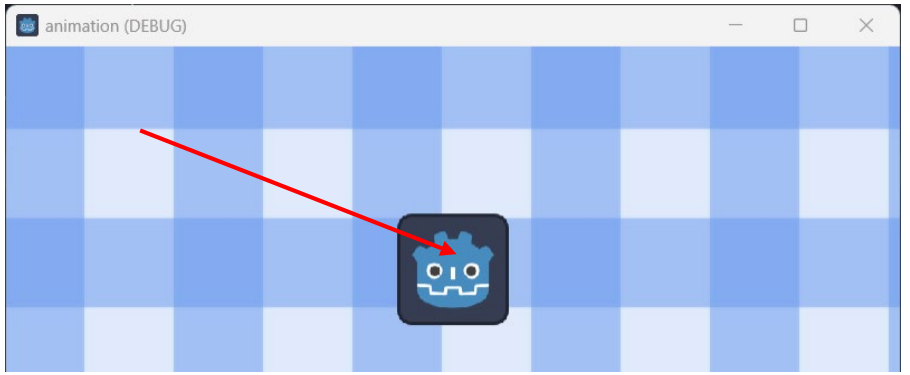


この例は、Tweenクラスを生成し、任意の座標へ移動するスクリプトです。

```
func _ready():
    var tween = create_tween()
    tween.tween_property(self, "position", Vector2(200,100), 1)
```

移動先の座標を指定しています

実行すると、1秒間をかけて右下（200,100）の位置へ移動します。

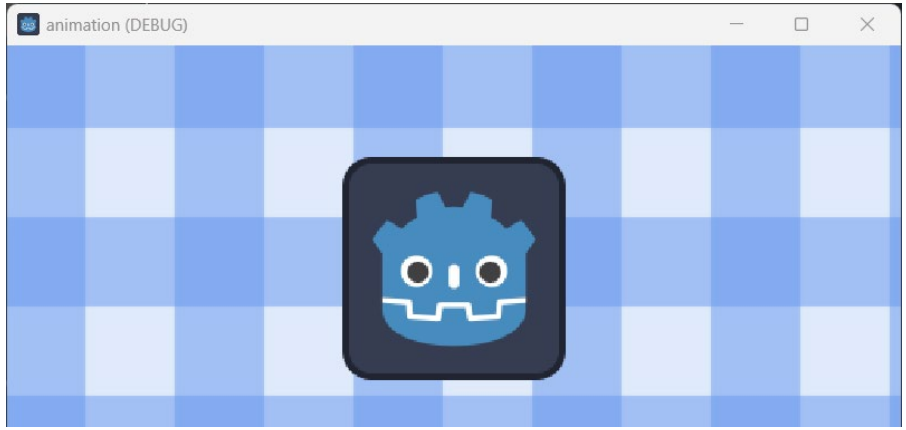


tween_property(オブジェクト, プロパティ, 変化後, 動作間隔)

オブジェクトのプロパティを変化させてアニメーションを行います。

移動した後に、大きさを2倍にする処理を追加しました。

```
var tween = create_tween()
tween.tween_property(self, "position", Vector2(200,100), 1)
tween.tween_property(self, "scale", scale*2, 1)
```



Tweenクラス（２）

Tweenクラスのアニメーションは**非同期**で実行されます。例えばさきほどのアニメーションの前後に、positionを表示する処理を入れて実行します。

```
print(position)
var tween = create_tween()
tween.tween_property(self, "position", Vector2(200,100), 1)
tween.tween_property(self, "scale", scale*2, 1)
print(position)
```

実行結果

```
(66, 48)
(66, 48)
```

結果は移動前の座標が連続表示されました。これはTweenアニメーションをしている間も他の処理は走り続けているということです。

Tweenアニメーションが終わってから何かの処理をしたい場合は、**tween_callback()**メソッドを使います。

```
func _ready():
    print(position)
    var tween = create_tween()
    tween.tween_property(self, "position", Vector2(200,100), 1)
    tween.tween_property(self, "scale", scale*2, 1)
    tween.tween_callback(finish_tween)

func finish_tween():
    print(position)
```

実行結果

```
(66, 48)
(200, 100)
```

2つのTween処理が終わってから、tween_callbac(メソッド名)で指定されたメソッドを呼び出します。

他にもTweenを制御できる処理がいくつかあります。

```
tween.tween_property(self, "position", Vector2(200,100), 1)
tween.parallel().tween_property(self, "scale", scale*2, 1)
```

parallel()をつけたアニメーションは、並行して実行されます。この場合は移動しながら大きさも同時に変更されます。

```
var tween = create_tween().set_parallel()
tween.tween_property(self, "position", Vector2(200,100), 1)
tween.tween_property(self, "scale", scale*2, 1)
```

また、tweenクラス作成時に、**set_parallel()**を指定しておくと、全てが並行処理されます。

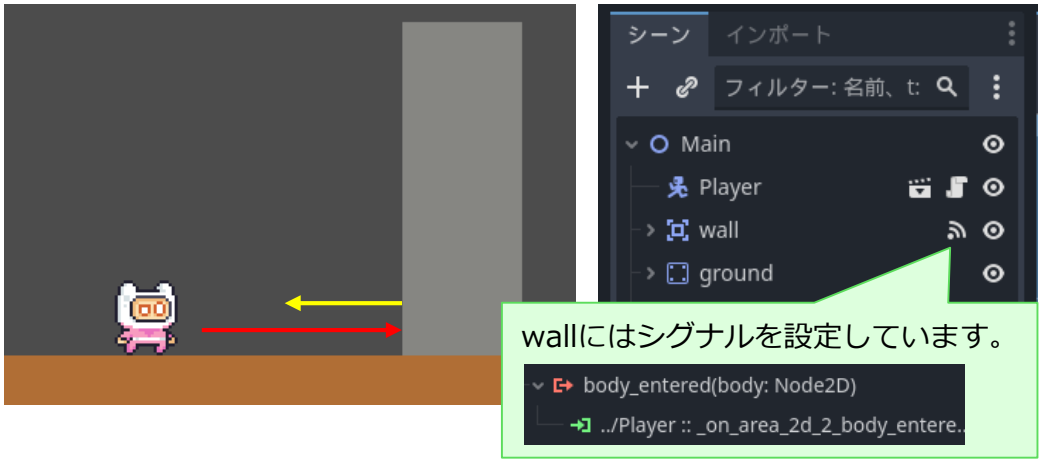
```
var tween = create_tween().set_parallel()
tween.tween_property(self, "position", Vector2(200,100), 1)
tween.chain().tween_property(self, "scale", scale*2, 1)
tween.tween_property(self, "rotation", deg_to_rad(90), 1)
```

set_parallel()が指定されていても、**chain()**が指定されていれば1つ前のアニメーションが終わってから実行されます。

```
tween.tween_property(self, "position", Vector2(200,100), 1)
tween.tween_interval(1)
tween.tween_property(self, "scale", scale*2, 1)
```

tween_interval(時間)は、指定した時間だけアニメーション処理を待機します。

プレイヤーのアクションで使う例として、壁にぶつかって跳ね返る（ノックバック）する動作をTweenでつくります。



プレイヤーが壁にぶつかるとシグナルにより、ノックバックするメソッドを呼び出すようにします。

まず、ノックバックの変数を設定します。kb_distanceは跳ね返る距離、kb_durationは跳ね返る時間です。

player.gd

```
var kb_distance = 50
var kb_duration = 0.2
```

※kbはノックバック（knock back）の意味です。

壁にぶつかったあとの処理です。

player.gd

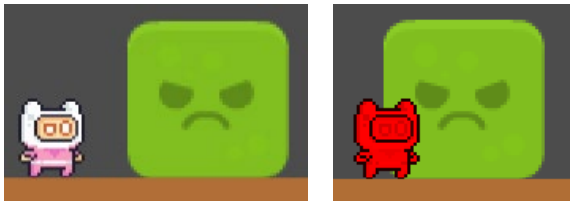
```
func _on_area_2d_2_body_entered(body):
    var direction = -transform.x.normalized()
    var kb_position = position + direction * kb_distance

    var tween = create_tween()
    tween.tween_property(self, "position", kb_position, kb_duration)
```

directionは移動方向にマイナスをつけることで、逆方向のベクトルになります。右向きに進んでいる場合はVector2（-1,0)になります。

kp_positionは、移動後のpositionを算出します。現在のpositionに方向と距離を掛けたものが格納されます。
その後、tweenクラスを生成し跳ね返る処理をします。

もう一つの例では、キャラクターのダメージ表現として、色をTweenで変化させています。



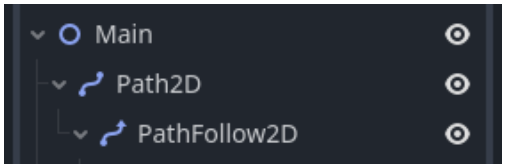
player.gd

```
func change_color():
    var original_color = self.modulate
    var tween = create_tween()
    tween.tween_property(self, "modulate", Color(1.0, 0.0, 0.0), 0.1)

    await get_tree().create_timer(0.2).timeout
    tween = create_tween()
    tween.tween_property(self, "modulate", original_color, 0.1)
```

Path（パス）

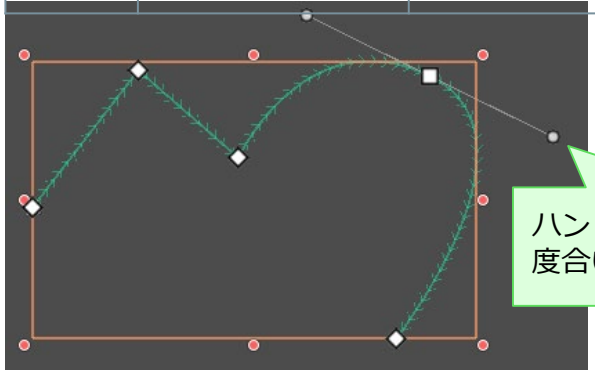
プラットフォーマーの足場を一定のルートに沿って移動させる場合など、「経路」が必要な場合にPathを使用します。



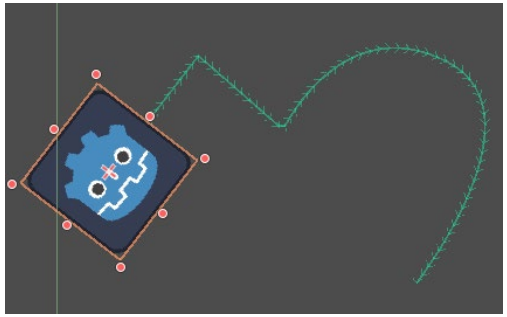
Path2Dノードで経路となるデータを作成し、経路に沿って動かすためにPathFollow2Dを使用します。

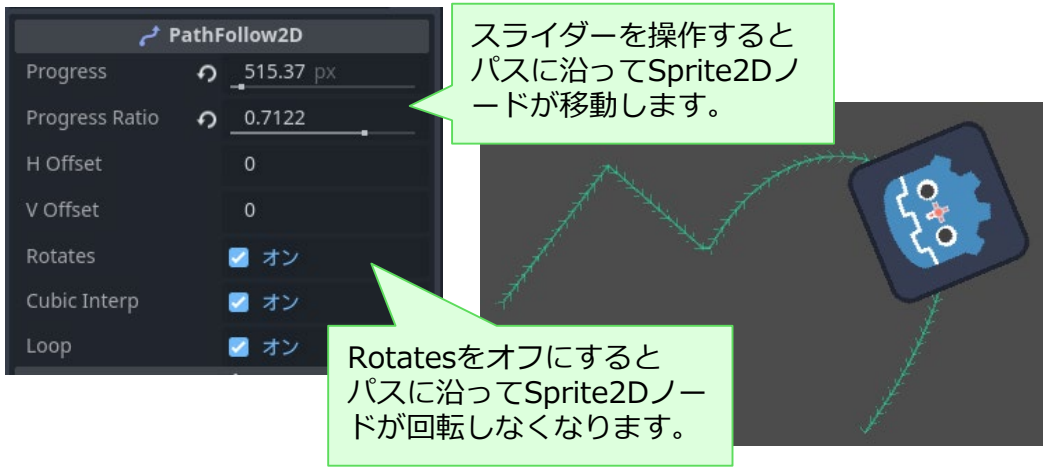
Path2Dノードを選択すると、ツールバーにパスを描画するためのツールが表示されます。

ポイント選択	パスのポイントを選択、移動する
ポイント編集	ポイントのカーブを編集する
ポイント追加	ポイントを描画する
ポイント削除	ポイントを削除する
パスを閉じる	パスの終端同士をつないで閉じます

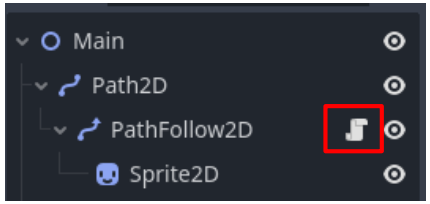


Sprite2DノードをPathFollow2Dノードの子ノードとして作成します。





スクリプトから動かすには、PathFollow2Dにコードを書きます。



PathFollow2D.gd

```
func _process(delta):
    progress_ratio += 0.005
```

progress_ratio（プログレスレイシオ）を毎フレーム増加させることで移動させることができます。

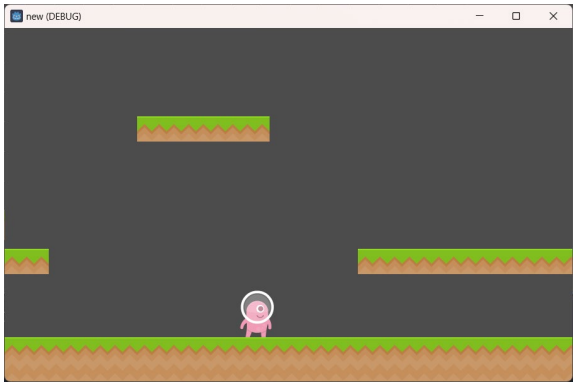
2-3 背景&カメラ

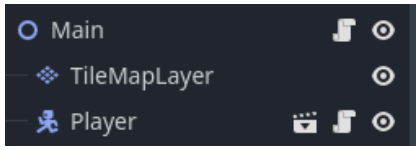
- 背景
- Parallax2D（１）
- Parallax2D（２）
- カメラと背景
- カメラ設定

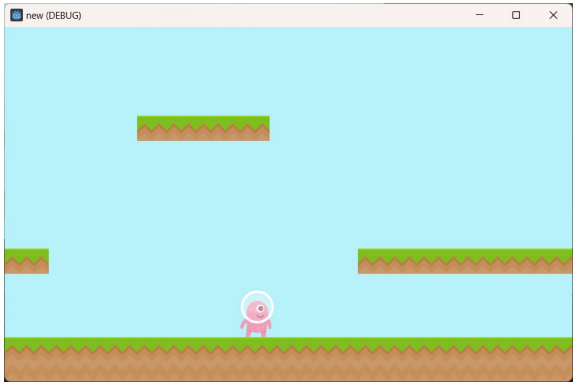
ゲームの種類にもよりますが、ゲームの背景は視覚的に世界観を伝える手段でもあり、奥行などの空間を表現したりすることもできます。

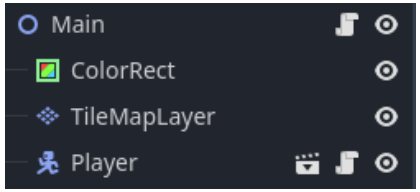
■ColorRect

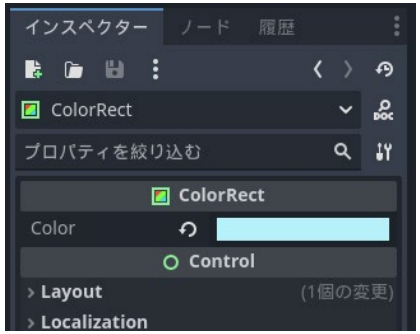
最もシンプルな方法は、ColorRect（カラー・レクト）ノードを使って、単色の四角形で背景を表現するやり方です。







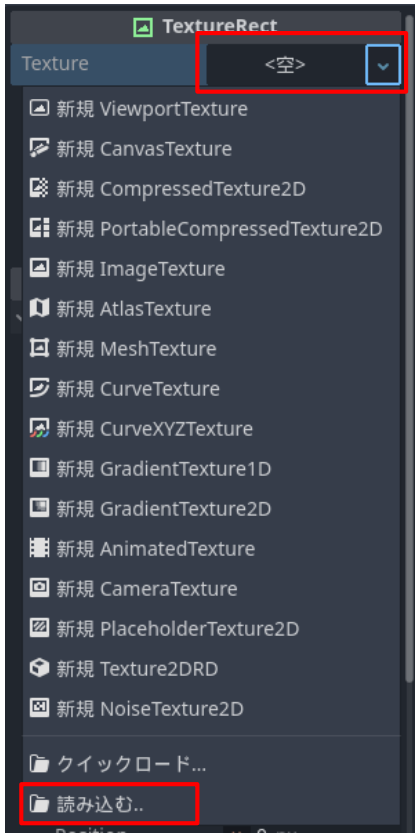




これだけでもゲームの雰囲気が変わるのが分かります。

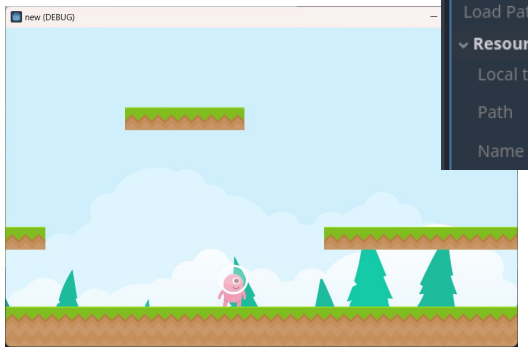
■TextureRect

画像を背景に使いたい場合は、TextureRect（テクスチャ・レクト）ノードを使います。



TextureRectのインスペクターでは、使用するTextureをいくつかの種類から選択します。

ここでは画像を使うだけですので、一番下の [読み込む...] から、使用する画像ファイルを読み込むか、<空>のところに、直接画像ファイルをドラッグします。



画像ファイルが読み込まれて、背景となりました。

Parallax（パララックス）とは、視点の移動によって近くの物体と遠くの物体が異なる速度で動くように見える現象のことです。日本語では「視差」とも呼ばれます。

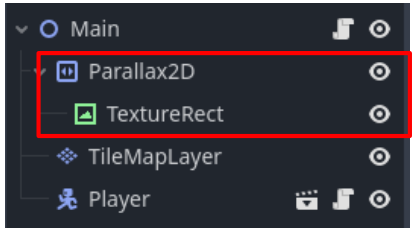
2Dゲームでは、パララックス効果を使うことで奥行き感や立体感のある背景を演出できます

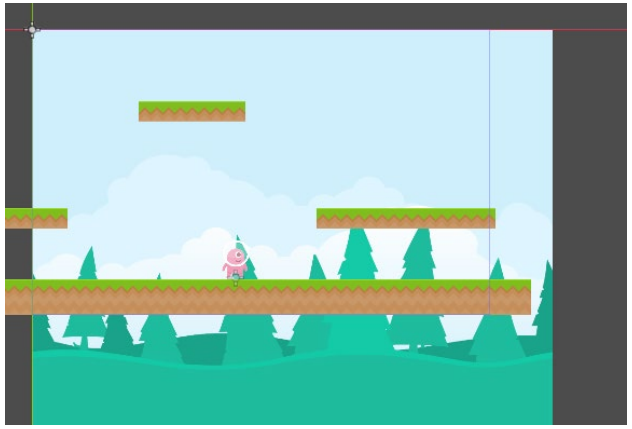
GodotでもParallaxを実装するためのノードがあり、バージョン4.3からは「Parallax2D」ノードが追加され、それ以前のバージョンで使用していたノードを使わずに簡単に背景効果を表現できるようになりました。

奥行表現を使わなくても、シンプルにスクロールする背景としても使えます。

■Parallax2D

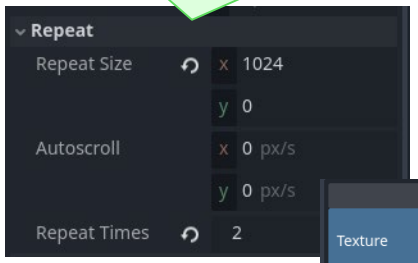
Parallax2Dノードを追加し、背景画像としてTextureRectを子ノードとして配置しています。

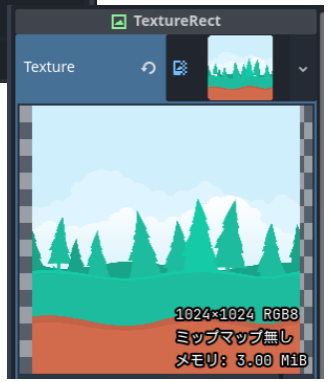




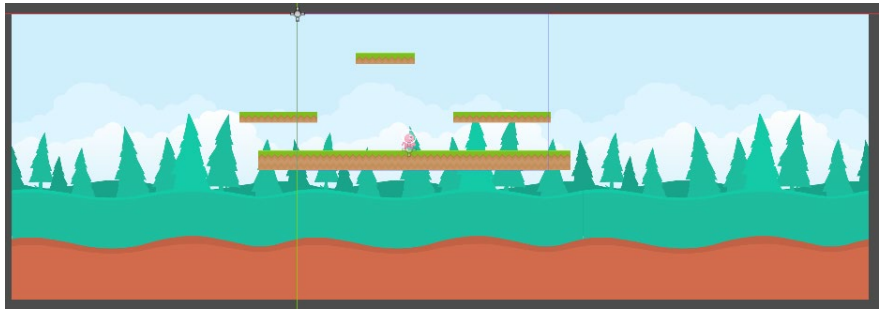
インスペクターでは背景のスクロールや繰り返しの設定ができます。

この例では、背景画像が1024x1024ピクセルのため、Repeat Sizeには1024を指定し、Repeat Timesには2を設定しています。





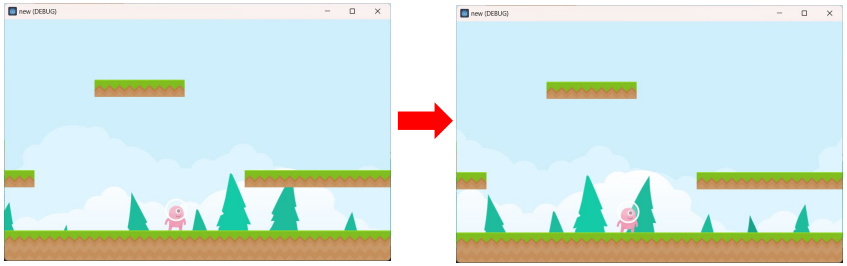
背景画像が左右にリピートされますので、これだけで背景画像を拡張することができます。



■背景スクロール

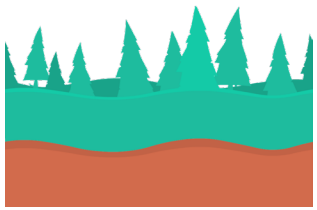
シューティングゲームやランゲームでは背景が自動的にスクロールすることで、ステージの進行やスピードを表現することができます。Parallax2Dノードを使うことで、背景が継ぎ目なくリピートされます。

Autoscrollを設定すれば背景が自動的にスクロールするようになります。
この例では１秒間に100px左方向にスクロールします。

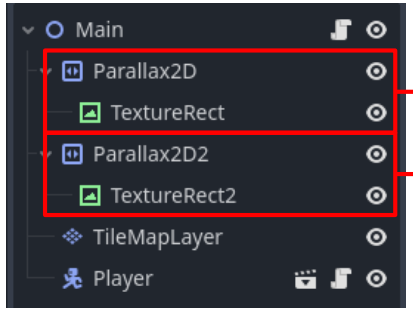


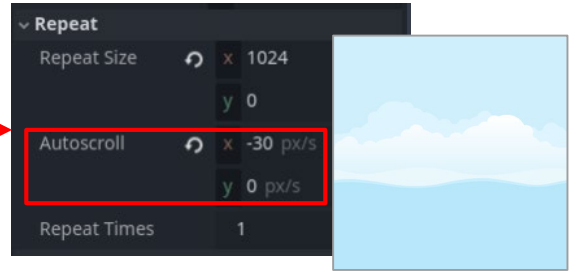
■多重スクロール

複数の背景を用意し、スクロールスピードを変えることで奥行感を演出することができます。例として遠景と近景の２枚の素材を用意します。



Parallax2Dノードを２組用意します。



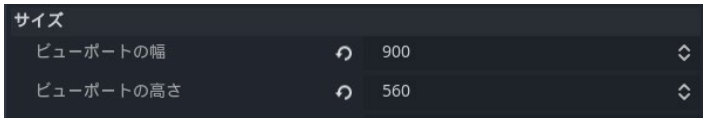


手前に表示される近景の森は、遠景の空の風景よりもスクロールスピードを速くすることで遠近感を演出しています。

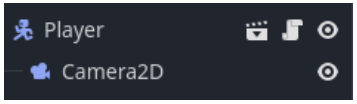
実際に画面で動かしてみないと分かりにくいですが、背景素材さえ用意できれば、とても簡単につくることができ、ゲームの奥行き感を出すことができます。

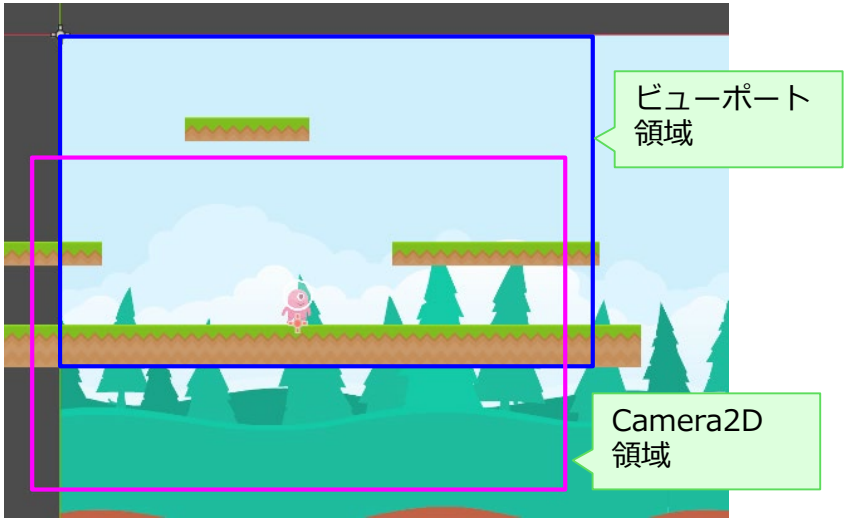
カメラと背景

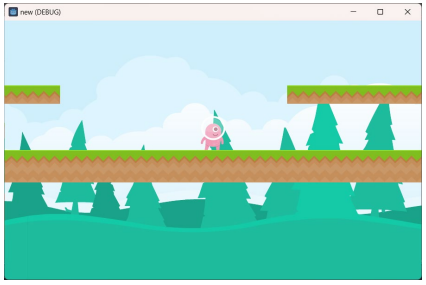
ゲーム画面に見える領域は、基本的にビューポートで設定された範囲が表示されます。



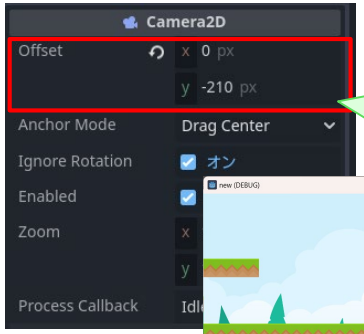
プラットフォーマーゲームのように、プレイヤーは画面中央付近に固定されており、背景やステージが移動するためには、カメラを使います。
Godotの2Dゲームでは、Camera2Dノードを使用します。



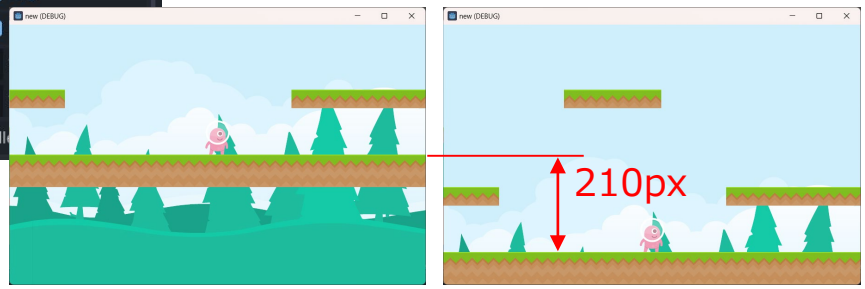




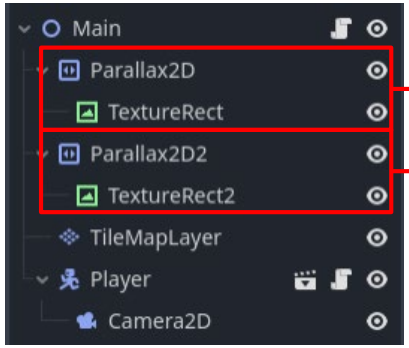
デフォルトのままではプレイヤーが中央に表示されるため、パラメータ設定を変更します。

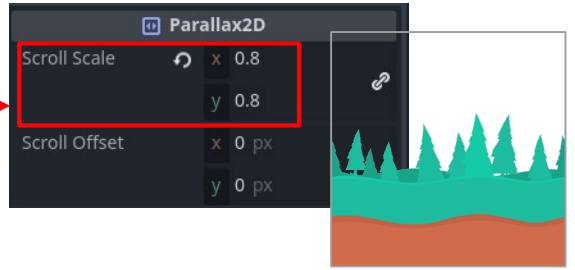


縦方向のyを-210pxに設定しています。カメラが上方向に210px移動することで、210px下がって表示されるようになりました。



プレイヤーの動きに合わせて、2つの背景がそれぞれ異なるスピードでスクロールするように設定します。





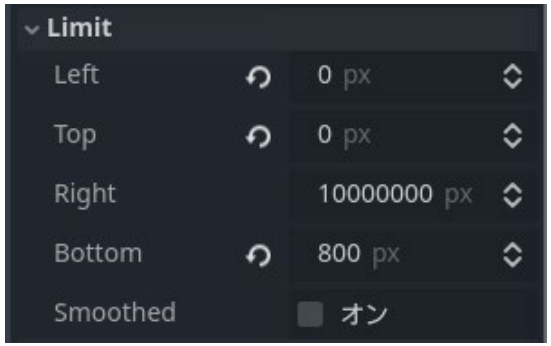
Scroll Scaleをそれぞれ、0.5と0.8に設定しました。遠くの背景ほど数値を小さくするのがポイントです。

カメラ設定

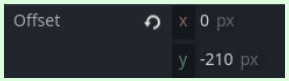
Camera2Dには他にも多くのパラメータがあります。よく使われるものを解説します。

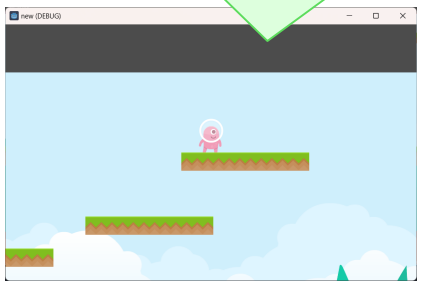
Limit

カメラがプレイヤーに追随する範囲を制限します。

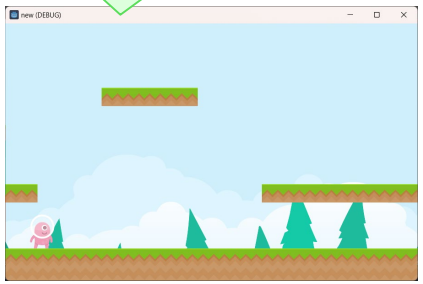


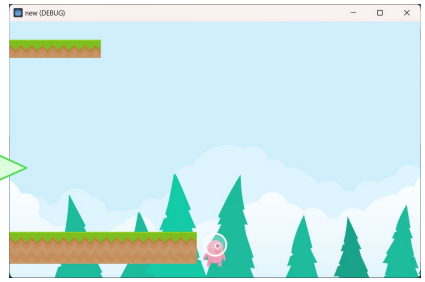
Topを0にして、上端のスクロールをコントロールします。この場合はOffsetも指定しているため、切れ目が見えてしまっています。





Leftを0にすることで、左端にはスクロールしません。

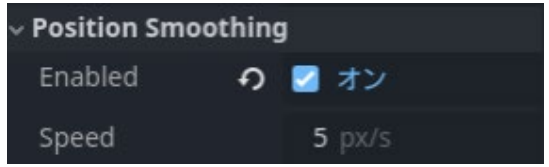




Bottomを設定することで、プレイヤーが落ちていってもカメラは追随しないようにできます。

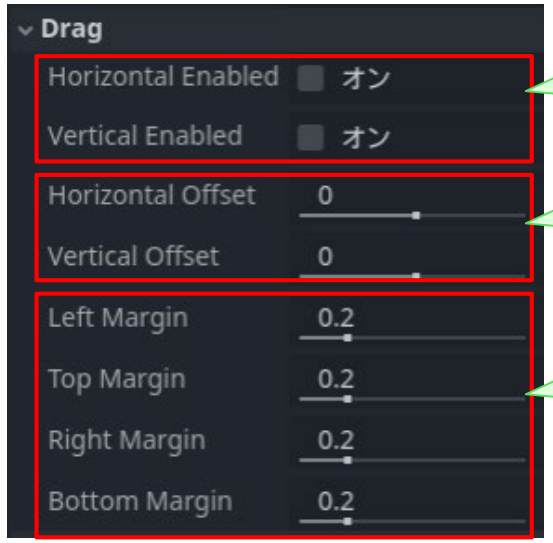
Position Smoothing

この設定にチェックを入れると、カメラの追随を滑らかにすることができます。



Drag

プレイヤーに追随しない範囲を設定します。



横方向、縦方向の追随しない設定を切り替えます。

横方向、縦方向のカメラ表示位置をずらします。

上下左右のマージン（追随するまでの余裕）割合。

Editor

エディタ画面上での表示設定をします。

カメラの表示範囲枠などを表示のオンオフを切り替えます。