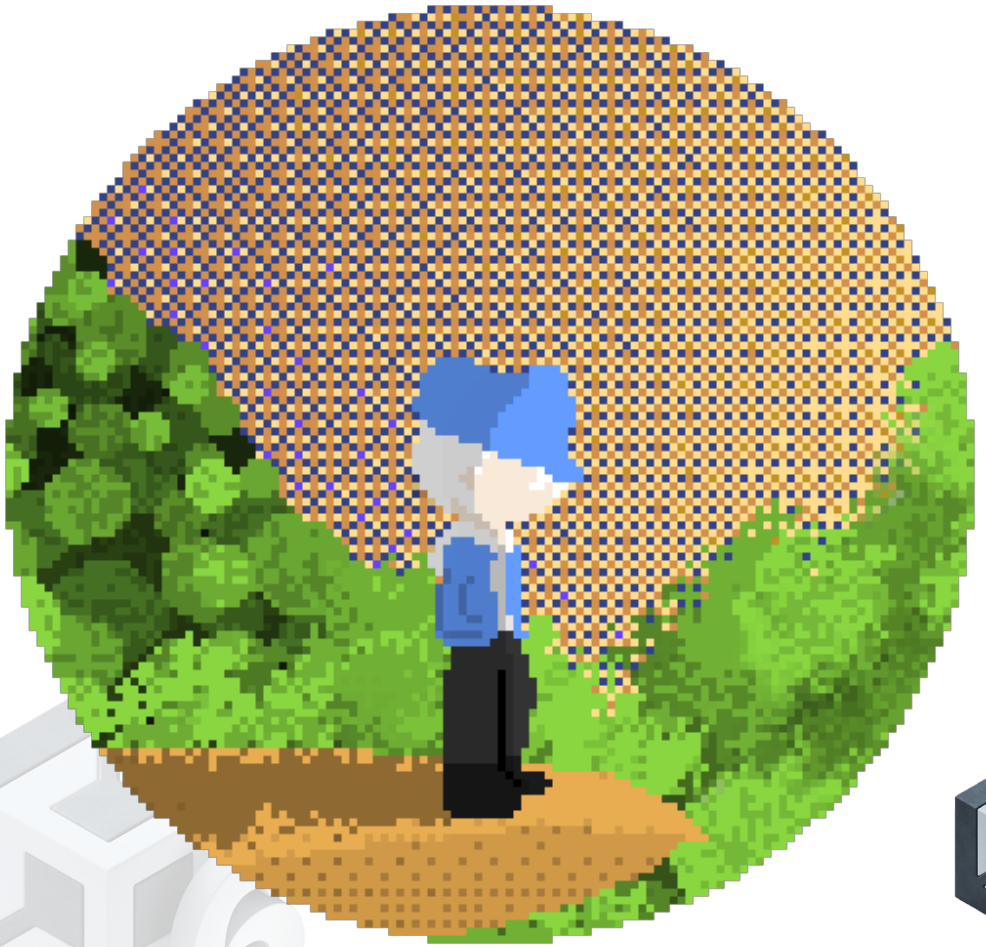
Unity初心者講座

1.プレイヤーの操作

自己紹介



21st/部長/インフラ

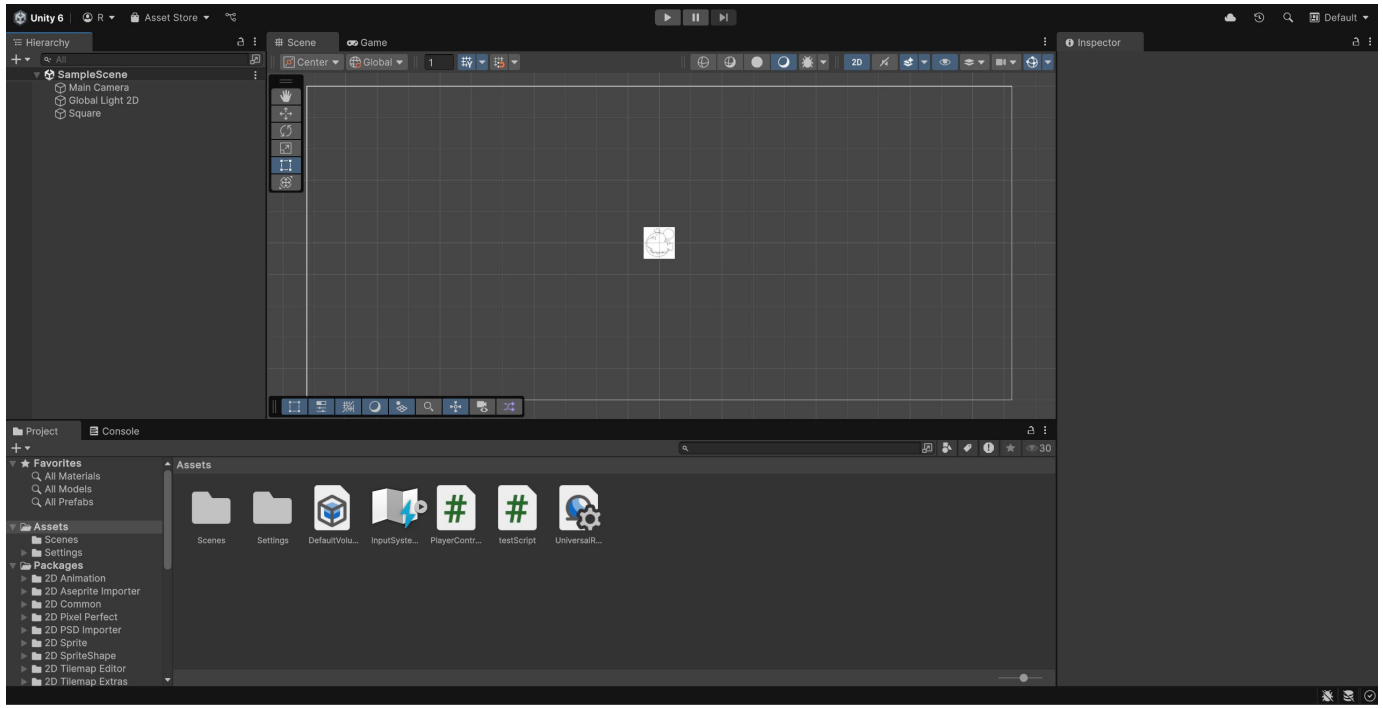
ロペ

Unityでゲームを作りながら
ドット絵でお絵描きしている
電子情報システム学科の2年

概要

1：プレイヤーの移動

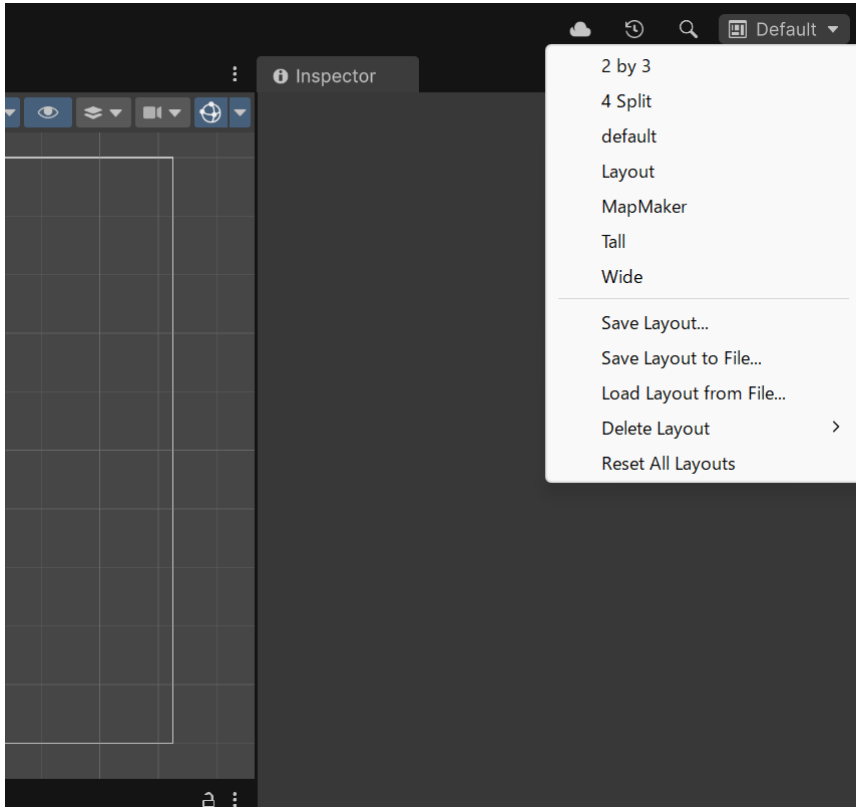
前回作成したプジェクトを
UnityHubで開きなおします



1：プレイヤーの移動

初期画面だと少し操作しにくいので、レイアウトを変えます

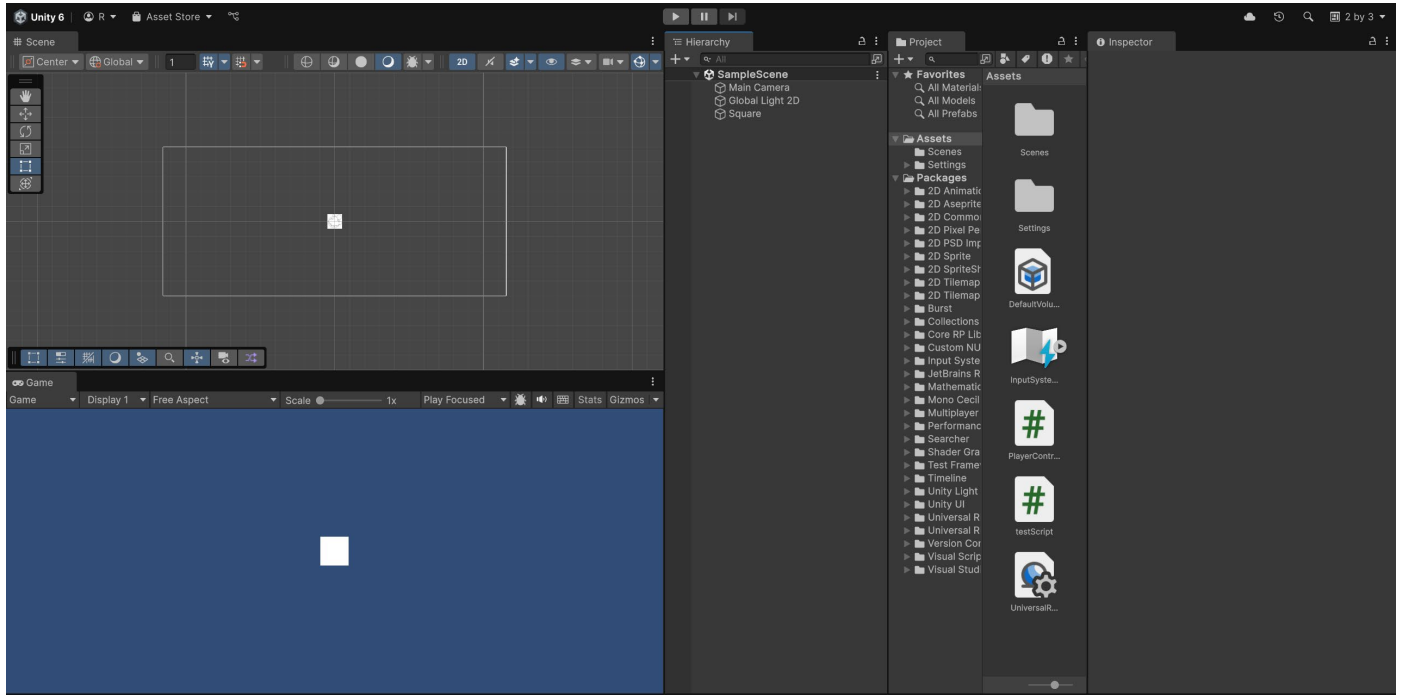
右上の”Default”から
“2 by 3”を選択



1：プレイヤーの移動

いい感じに変わったかと

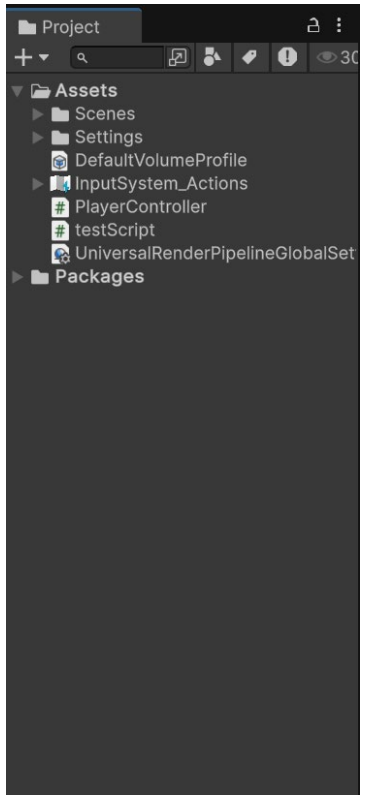
Projectが見にくい場合は
Projectタブの三つの点から
“One Column Layout”
を選択すると
１列になると思います
〔僕は１列で使っている〕



Defaultで使いたい方は
それで構いません！

1：プレイヤーの移動

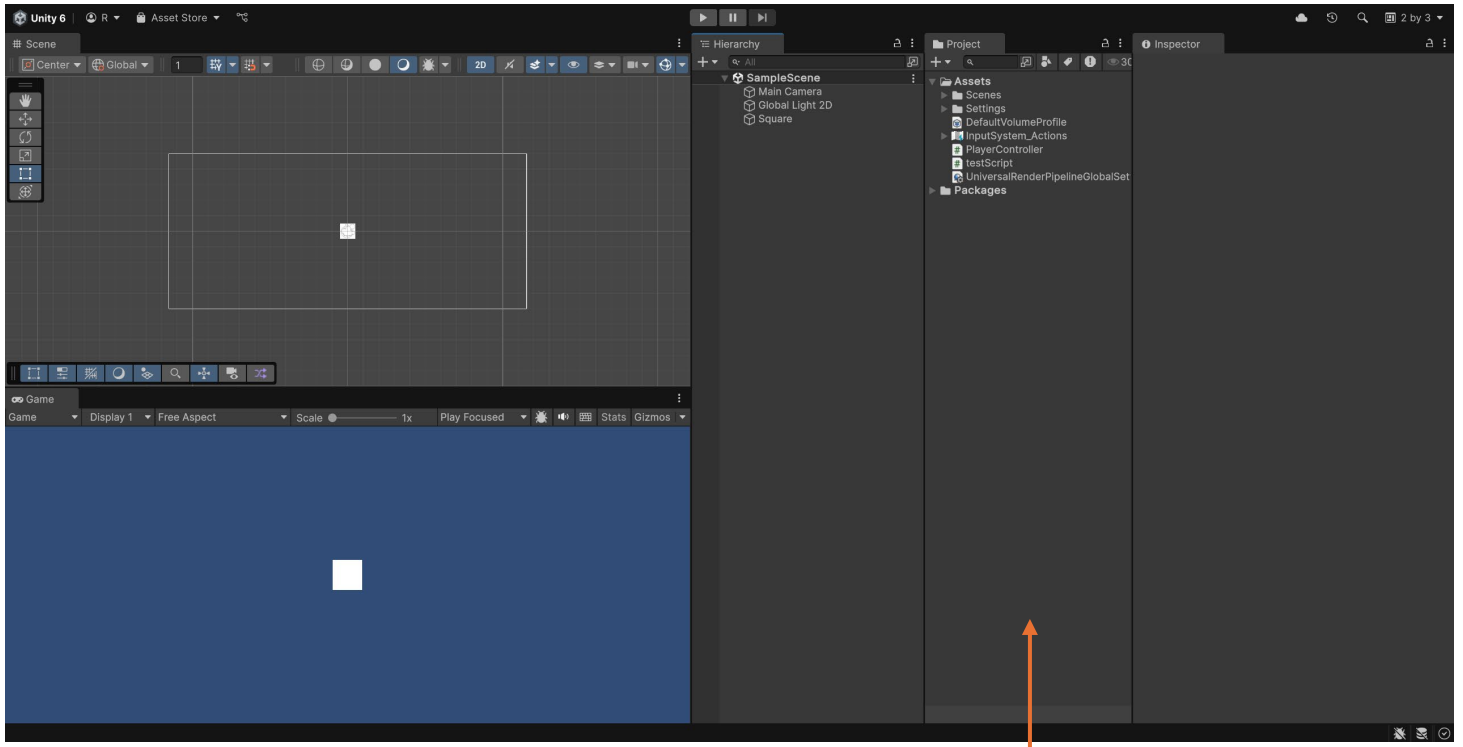
変えるとこんな感じ
〔Packagesは閉じた状態〕

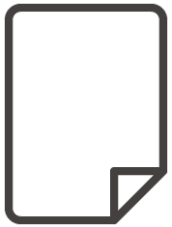


1：プレイヤーの移動

このページから素材を
ダウンロードする

ダウンロードしたものを
Projectに放り投げてみる



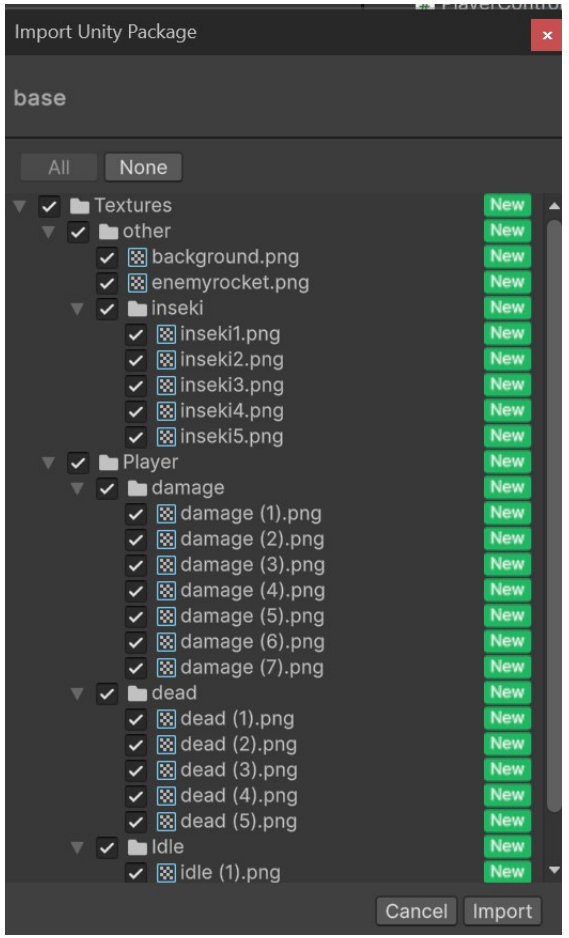


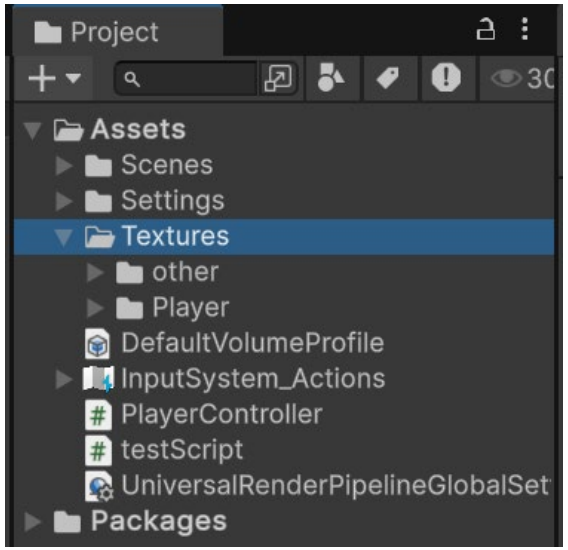
Base_ver2.unitypackage

1：プレイヤーの移動

Unityに素材を入れる
タブが出てくるので、
Importで入れる

入れたらProjectタブに
Texturesが出てくる





1：プレイヤーの移動

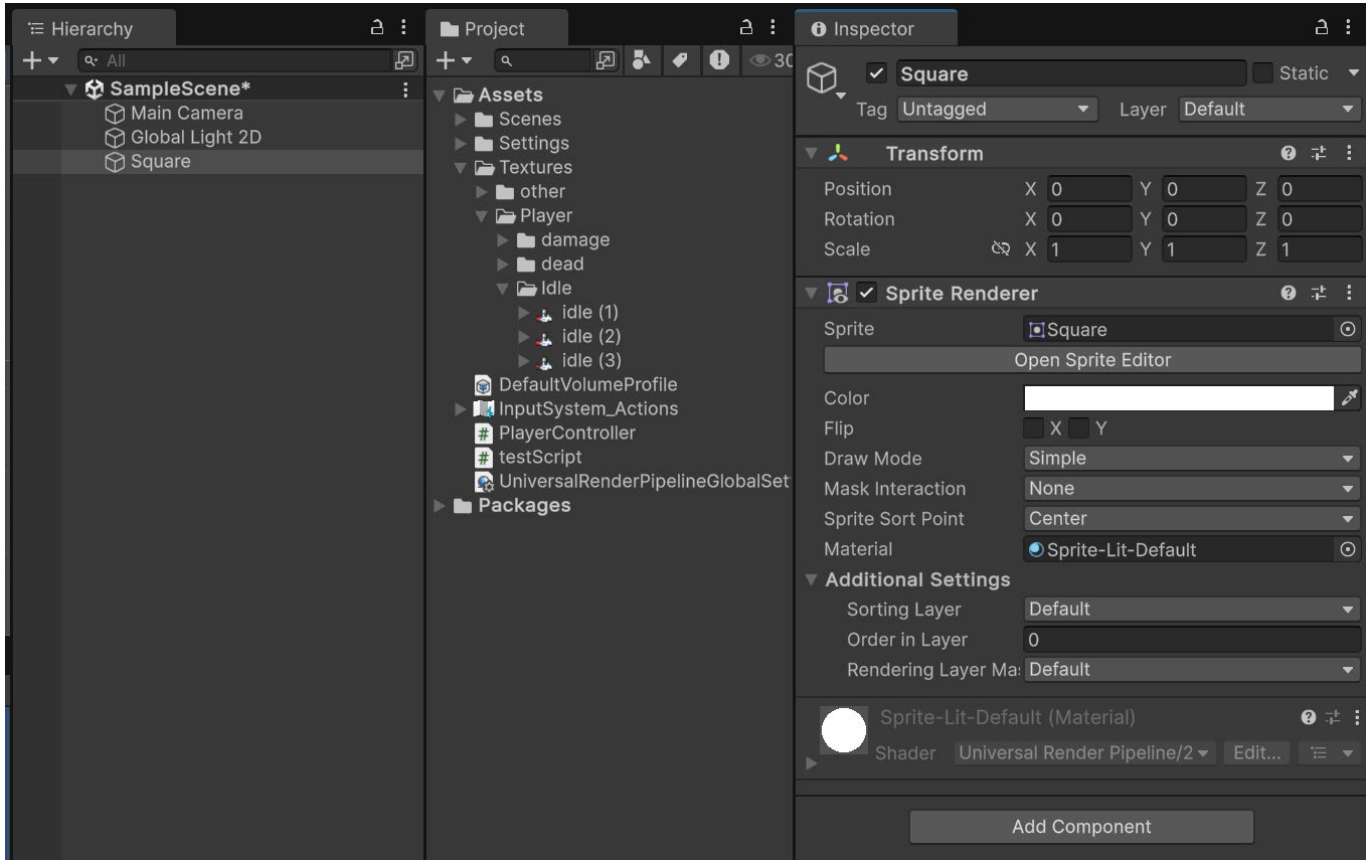
試しに画像を使用してみる

Squareを選ぶと
Inspecterに
”Sprite Renderer”がある

Q:Inspecterってなんだ？〔次ページに回答〕

1：プレイヤーの移動

”Sprite Renderer”の中にある“Sprite”に
Textures/Player/Idle/idle(1)を入れてみる



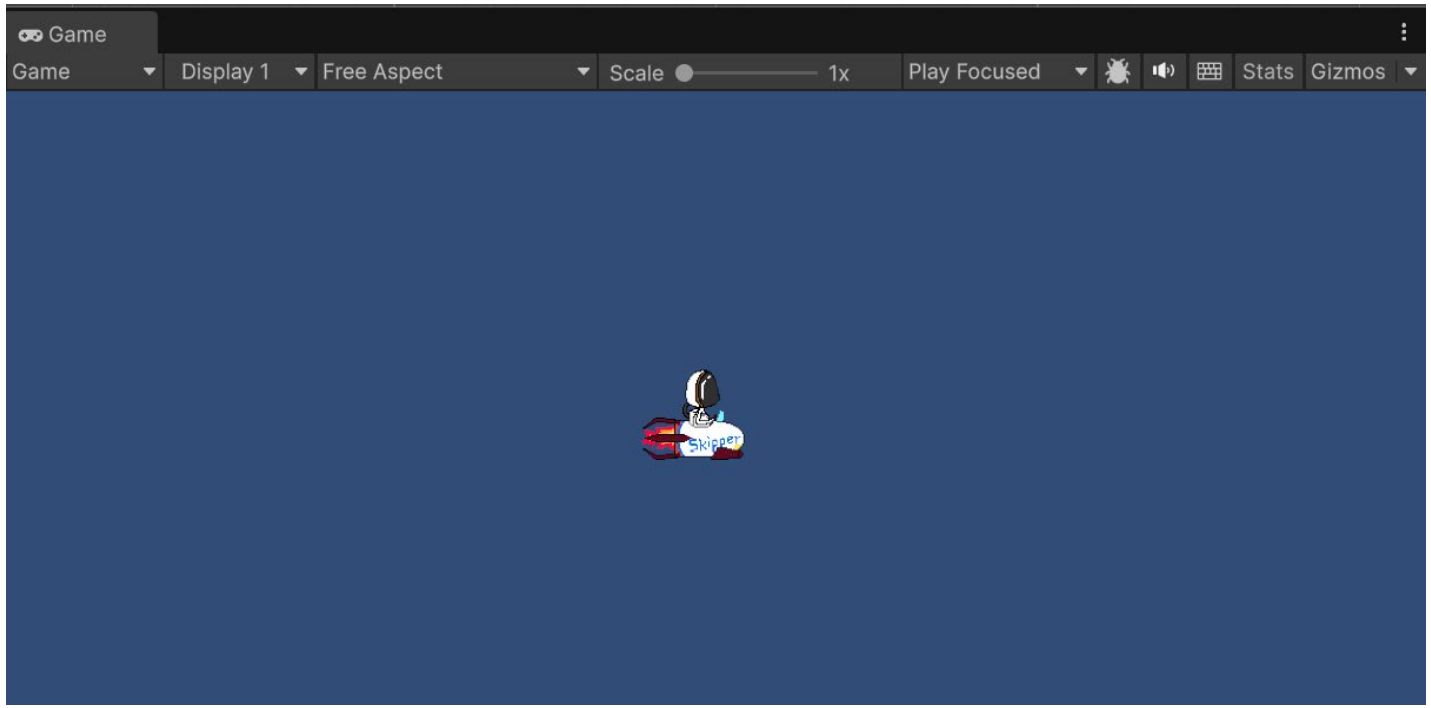
A:Hierarchyで選んだオブジェクトの詳細が見れるところ

1：プレイヤーの移動

四角だったものが
Idle[1]の画像に
置き換わっている

これをプレイヤーとして
作っていきます

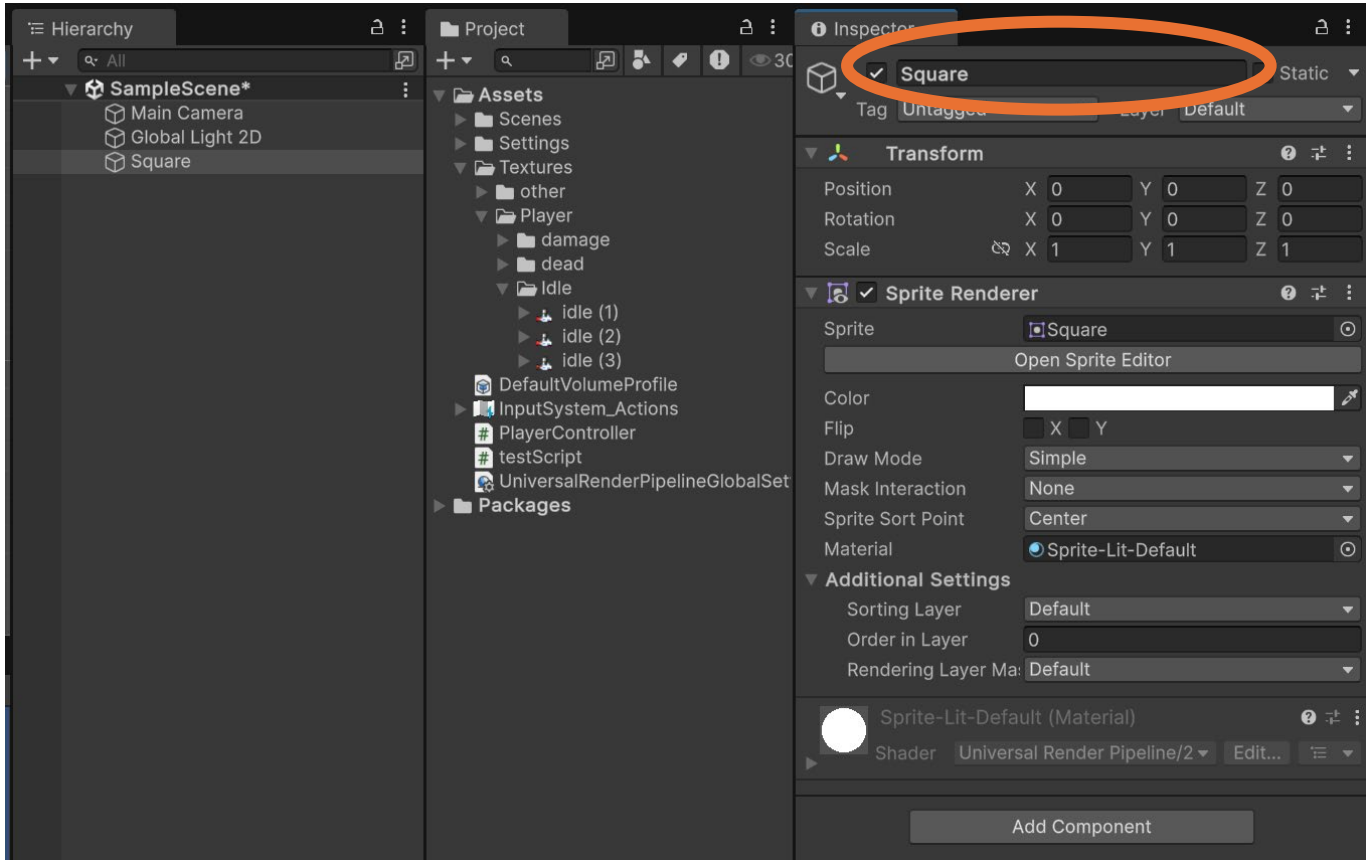
補足→p61



1：プレイヤーの移動

Squareの名前を
“Player”に置き換えておく

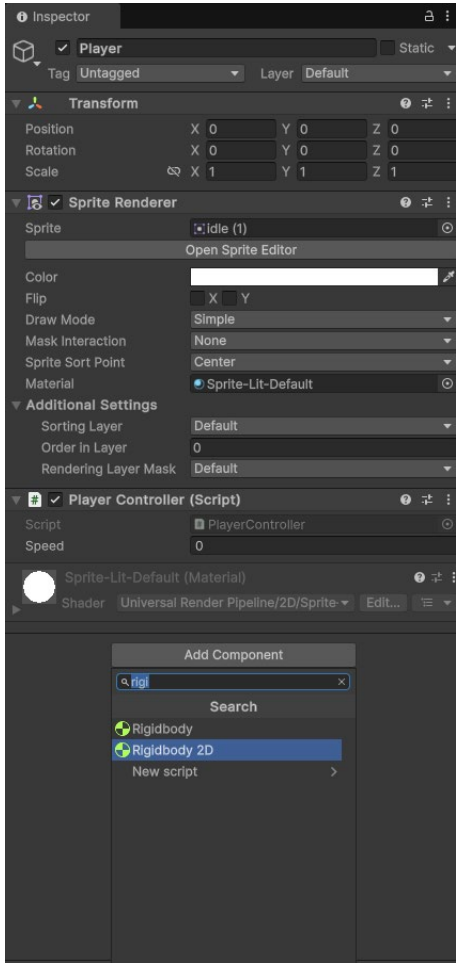
Squareを選んで
丸のところから変更できる



1：プレイヤーの移動

機能を入れてみる

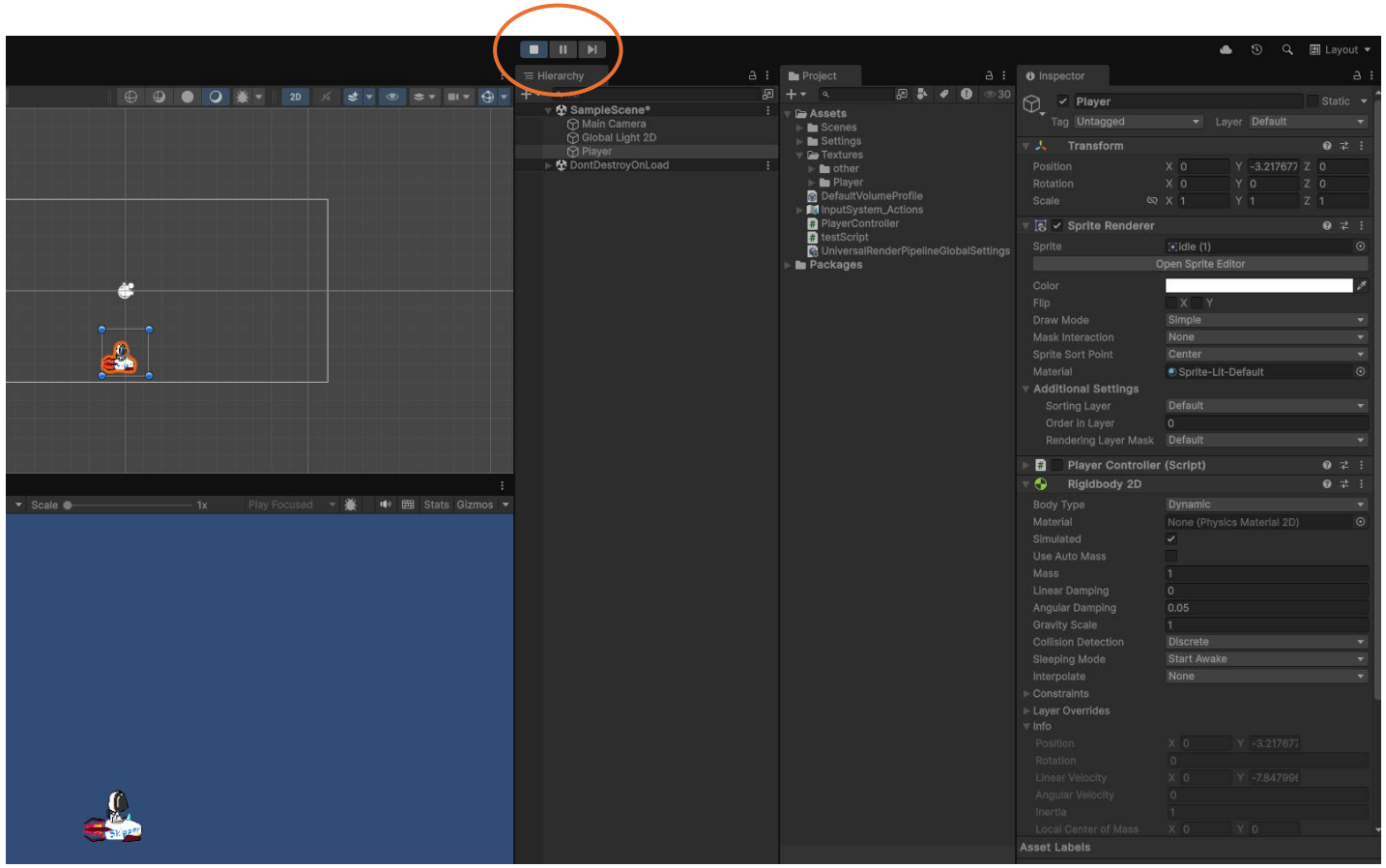
Playerを選び、一番下の
“Add Component”から
“Rigidbody 2D”を選択



1：プレイヤーの移動

入れた状態で丸から再生すると
Playerが下に落ちていく

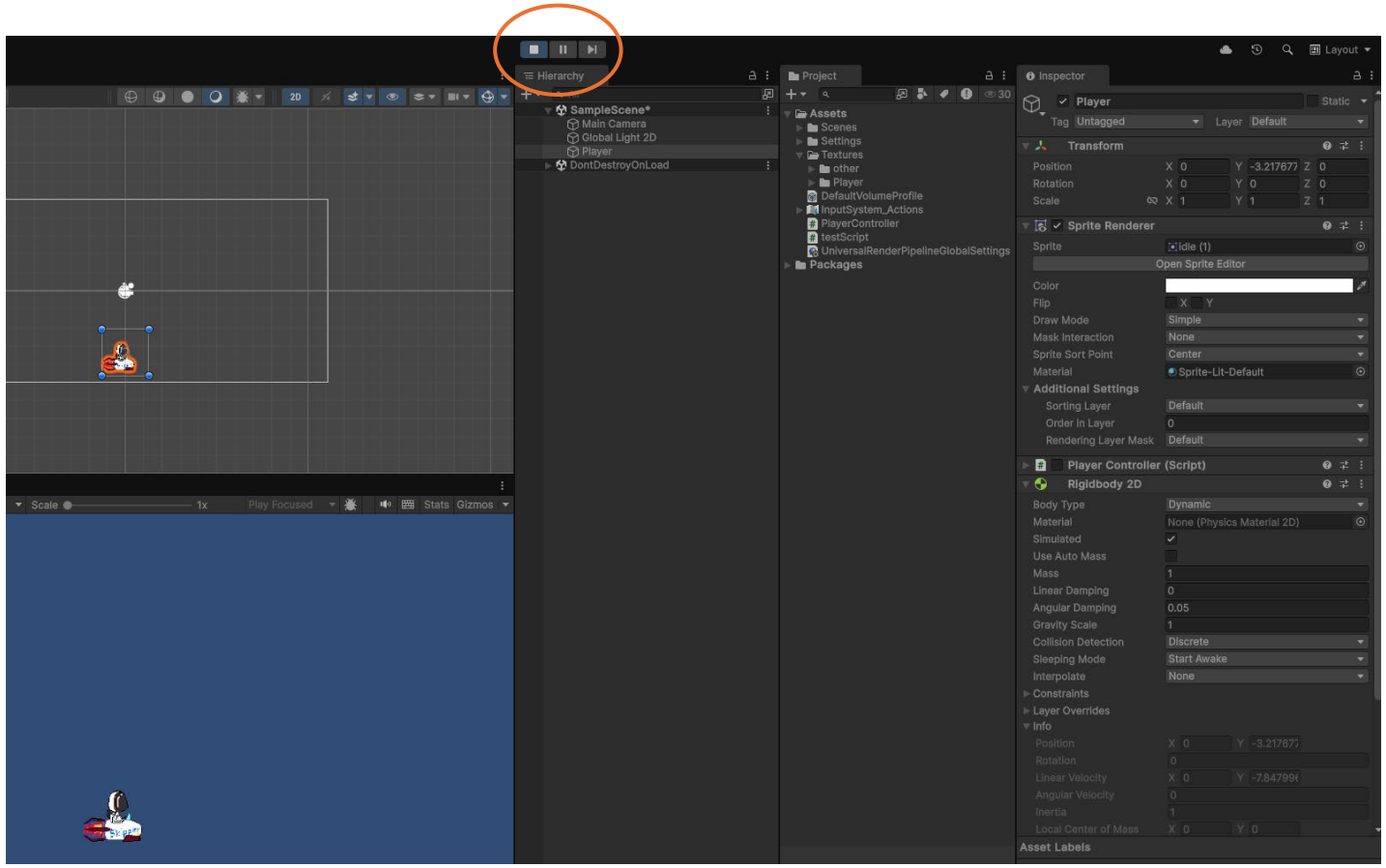
これは”Rigidbody 2D”を
Playerに追加した影響である



1：プレイヤーの移動

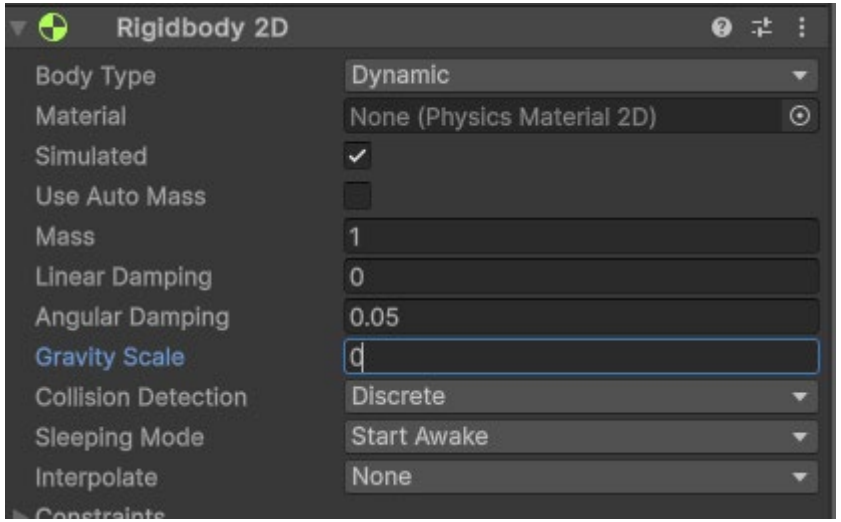
”Rigidbody 2D”は
総じていえば
『力を管理するもの』

Rigidbodyがないものに
ぶつけても物理的な反応はしない



1：プレイヤーの移動

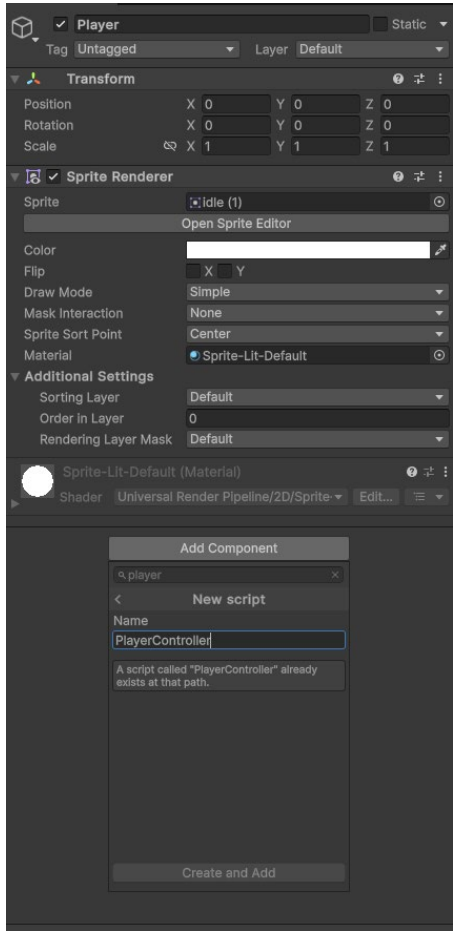
ただ、今回は重力は使用
しないため、
Rigidbody 2Dの
“Gravity Scale”を0に
変更しておく



1：プレイヤーの移動

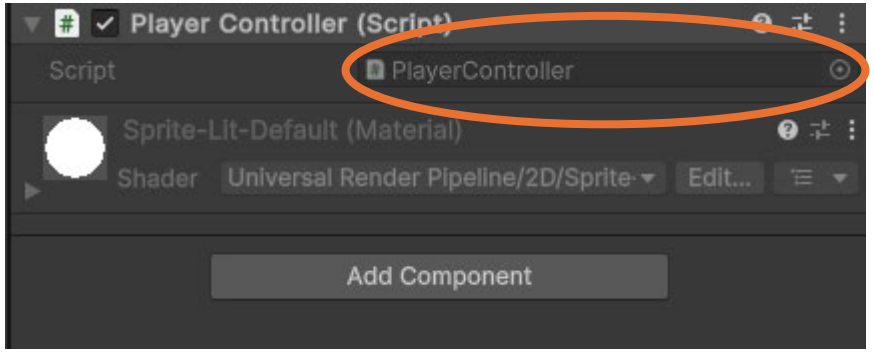
プレイヤーを操作するための
スクリプトを用意する

一番下の”Add Component”から
New Script → PlayerContorller
と入力して”Create and Add”



1：プレイヤーの移動

PlayerContollerができるので
Scriptをダブルクリックする



1：プレイヤーの移動

PlayerControllerの中身が見れる

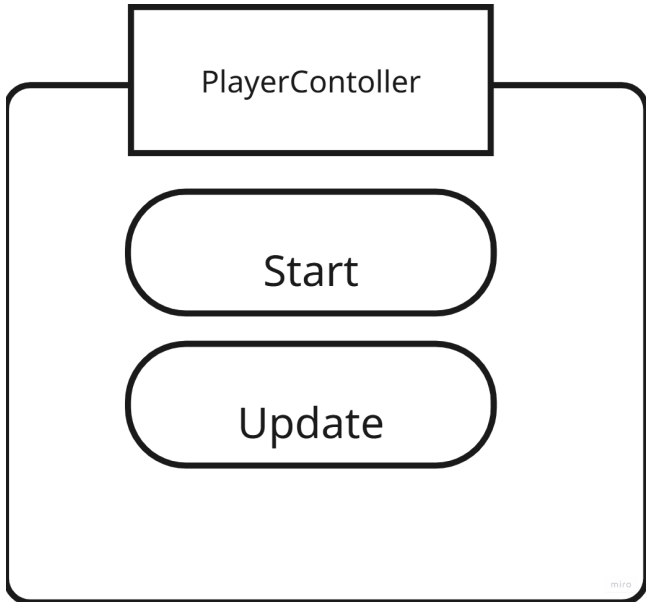
PlayerContollerの中には
作成時すでに

- Void Start()
  - 一番初めに実行されるところ
- Void Update()
  - 毎フレーム実行されるところ

という機能がある （メゾットといったりします）

```
using UnityEngine;

Unity スクリプト|0 個の参照
public class PlayerController : MonoBehaviour
{
    // Start is called once before the first execution of Update after the MonoBehaviour is created
    Unity メッセージ|0 個の参照
    void Start()
    {

    }

    // Update is called once per frame
    Unity メッセージ|0 個の参照
    void Update()
    {

    }
}
```

1：プレイヤーの移動




```using UnityEngine;

Unity スクリプト|0 個の参照
public class PlayerController : MonoBehaviour
{
    // Start is called once before the first execution of Update after the MonoBehaviour is created
    Unity メッセージ|0 個の参照
    void Start()
    {

    }

    // Update is called once per frame
    Unity メッセージ|0 個の参照
    void Update()
    {

    }
}
```

イメージは…こんな感じ…
PlayerControllerのように
一括りになっているものをクラスという

1：プレイヤーの移動

中身をこんなコードで書いてみる
解説はp25から

書き終わったら
Ctrl + S でセーブして
Unityにもどる

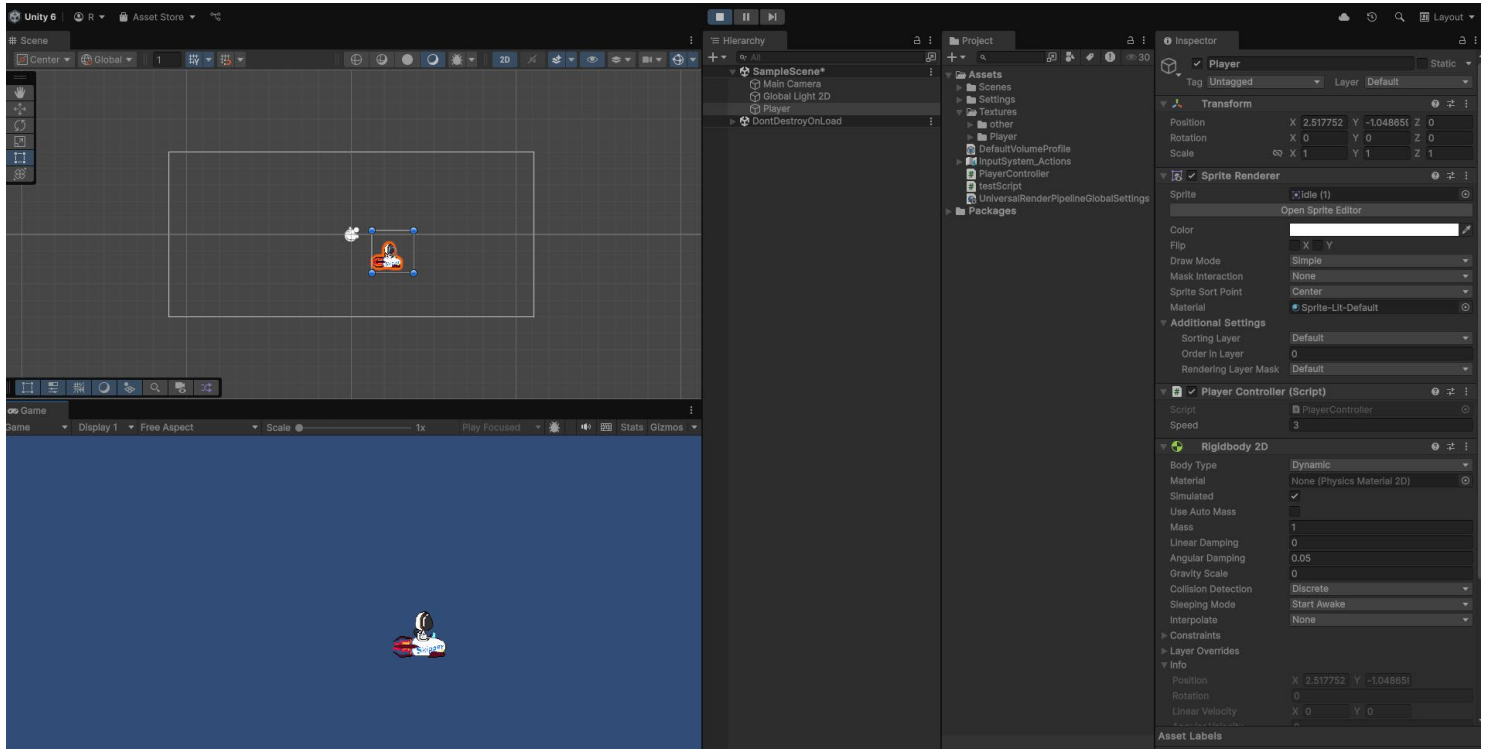
```
public class PlayerController : MonoBehaviour
{
    Rigidbody2D rb;
    public int speed;
    // Start is called once before the first execution of Update after the MonoBehaviour is created
    Unity メッセージ | 0 個の参照
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    // Update is called once per frame
    Unity メッセージ | 0 個の参照
    void Update()
    {
        float GetHorizontal = Input.GetAxis("Horizontal");
        float GetVertical = Input.GetAxis("Vertical");

        Vector3 Vec = new Vector3(GetHorizontal, GetVertical, 0);
        if (Vec.magnitude > 1)
        {
            Vec = Vec.normalized;
        }

        rb.linearVelocity = Vec * speed;
    }
}
```

1：プレイヤーの移動

Playerの
”PlayerContorller”
にspeedの項目があるので
好きな数を入れてから
再生する

WSAD/↑ ↓ →←で
Playerを動かせることが
確認できる

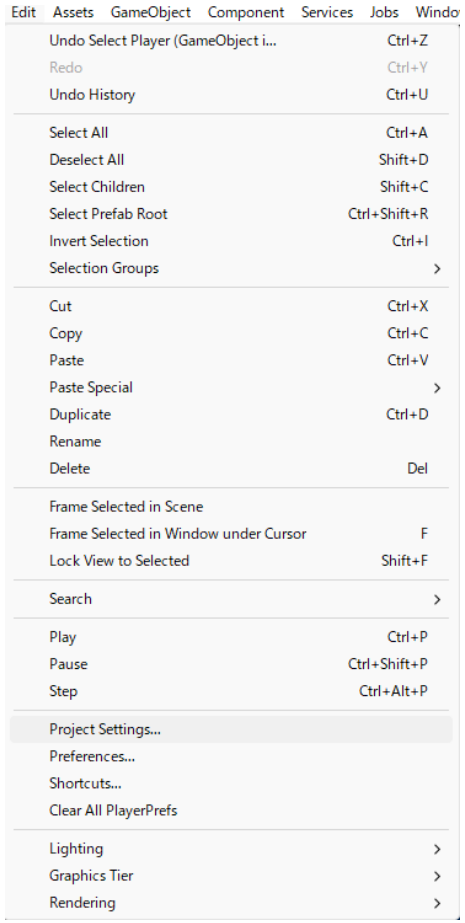
動かない場合、次のページに

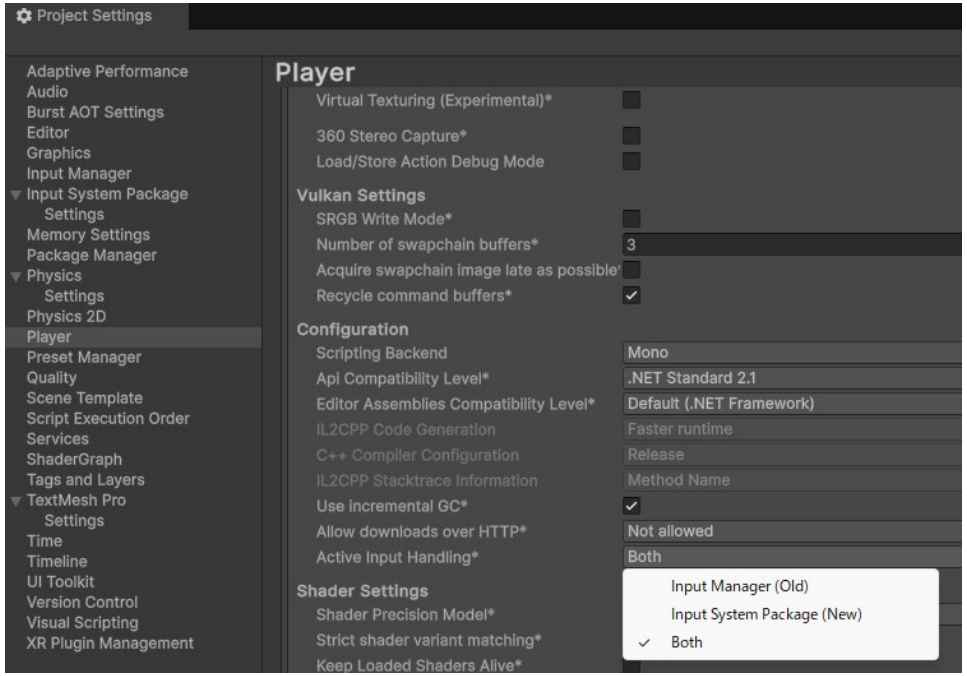


1：プレイヤーの移動

入力して動かない場合は
一番上のEditから
Project Settingsを選択

その後、
Playerを見てその中の
Other Settingから
“Active Input Handling”を
BothかInputManagerを選ぶ





1：プレイヤーの移動

確認ができたら
“再生を停止して”
コードに戻る

ここからコードの解説をする

```
public class PlayerController : MonoBehaviour
{
    Rigidbody2D rb;
    public int speed;
    // Start is called once before the first execution of Update after the MonoBehaviour is created
    Unity メッセージ | 0 個の参照
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    // Update is called once per frame
    Unity メッセージ | 0 個の参照
    void Update()
    {
        float GetHorizontal = Input.GetAxis("Horizontal");
        float GetVertical = Input.GetAxis("Vertical");

        Vector3 Vec = new Vector3(GetHorizontal, GetVertical, 0);
        if (Vec.magnitude > 1)
        {
            Vec = Vec.normalized;
        }

        rb.linearVelocity = Vec * speed;
    }
}
```

（もしlinearVelocityがエラーでているなら、velocityに置き換えてください
昔のUnityならこれで動きます）

1：プレイヤーの移動

ここではこれから使う変数、インスタンスを
宣言している

- 変数
  - 変更できる数
- インスタンス
  - 定義してあるクラスやUnityの機能をベースに
    変数のように扱うもの

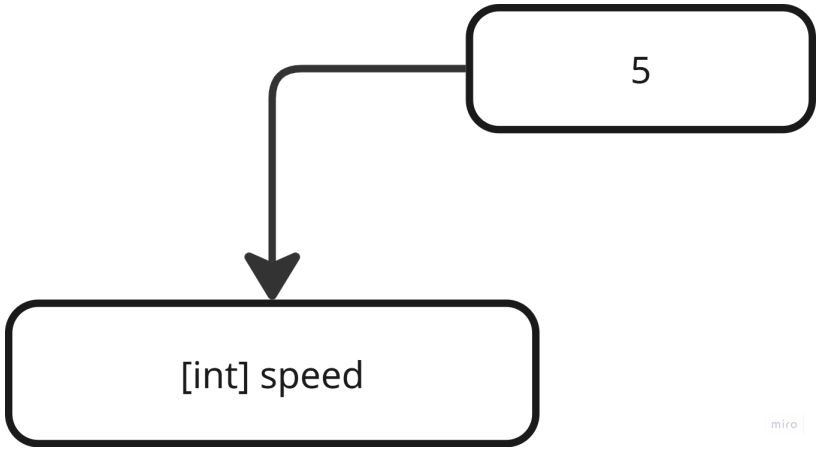
どちらも前に指定した”型”に合うものしか
入れれないものとなっている

```
public class PlayerController : MonoBehaviour
{
    Rigidbody2D rb;
    public int speed;
    // Start is called once before the first execution of
```

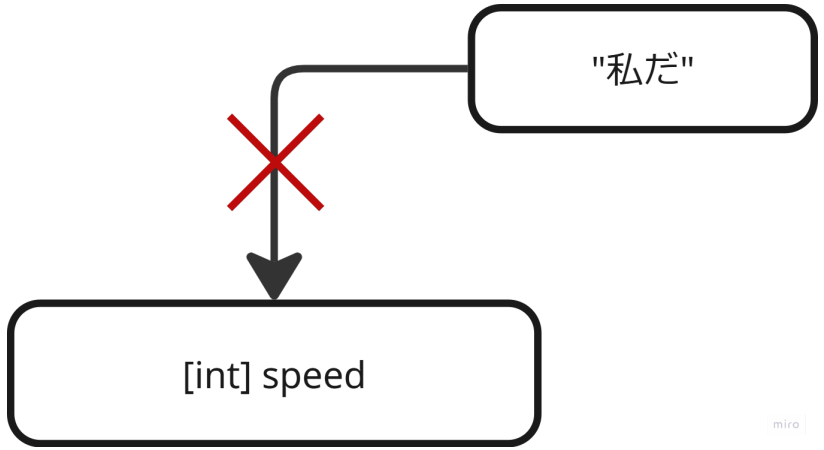
Rigidbody2D型のインスタンス”rb”と
Int〔整数〕型の変数”speed”

1：プレイヤーの移動

できる



できない



Int〔整数〕型で宣言した”speed”には
整数の数字〔例えば5〕は入れれるが、
“私だ”のような文字は入れることができない

ちなみに文字型はstring型という
〔正しくは１文字の列、ということで文字列型と呼ばれるが…〕

1：プレイヤーの移動

まぁ、色で違いがあるので
それで区別しても問題はない…

青色が変数で
エメラルド色？がインスタンス

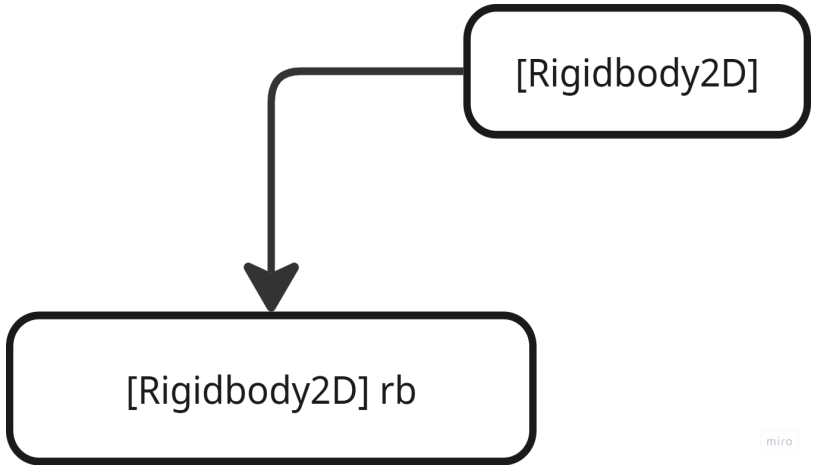
どちらにしても指定した型しか
入れられないのは共通である

```
public class PlayerController : MonoBehaviour
{
    Rigidbody2D rb;
    public int speed;
    // Start is called once before the first execution of
```

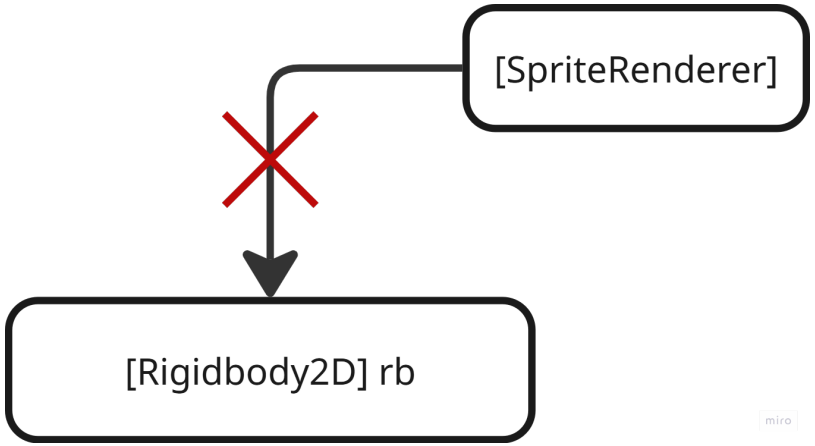
補足：p64

1：プレイヤーの移動

できる



できない



変数と同様に、インスタンスの場合も
Rigidbody2D型として宣言した”rb”には
Rigidbody2D型のモノは入れれるが、
それ以外の型のモノ〔例えばSpriteRenderer〕は
入れることができない

1：プレイヤーの移動

```
Unity メッセージ | 0 個の参照
void Start()
{
    rb = GetComponent<Rigidbody2D>();
}
```

Void Start() について見る

rbにGetComponentで取得してきたものを入れる、と
なっている

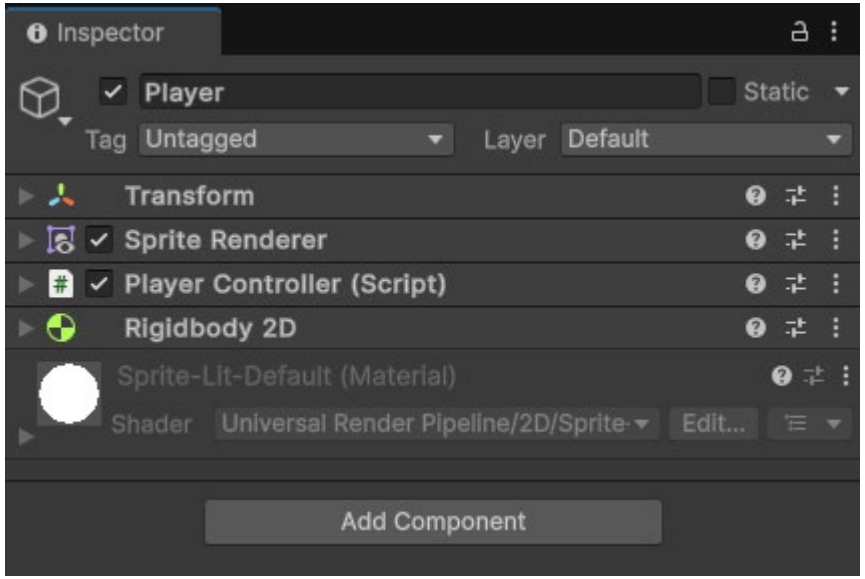
GetComponentは
コンポーネントを取得しますよ、というものである

じゃあコンポーネントってなんだ？

1：プレイヤーの移動

PlayerControllerをセーブしてから
Unityに戻る

Playerにいろんな機能を追加したはず
“Rigidbody 2D”だったり
“SpriteRenderer”だったり
“PlayerContoller”だったり
“Tranform”だったり…



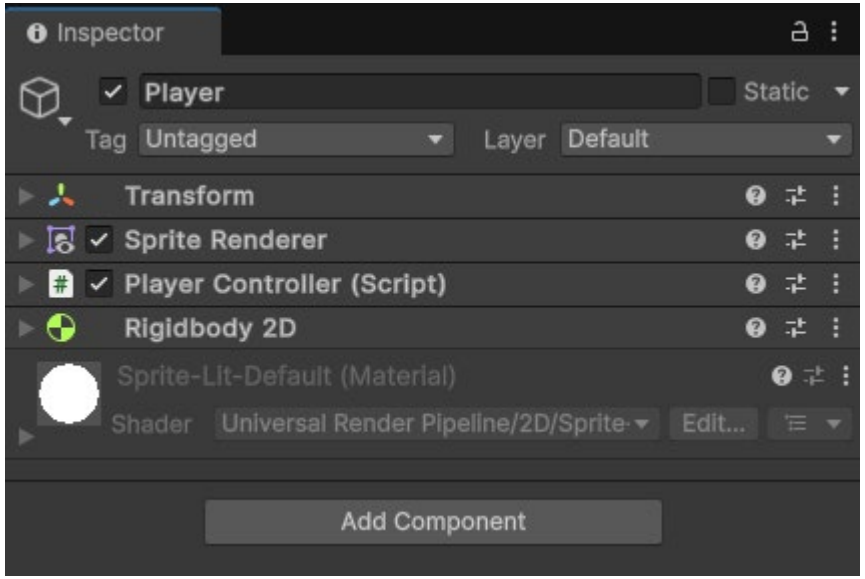
1：プレイヤーの移動

これらの機能がすべて

コンポーネント (Component)

と呼ばれている

追加する際に
Add Componentから行っているので
ここからコンポーネントを追加してる



1：プレイヤーの移動

コードに戻る

```
Unity メッセージ | 0 個の参照
void Start()
{
    rb = GetComponent<Rigidbody2D>();
}
```

コンポーネントの中でも、
今回は”Rigidbody 2D”が欲しいので
<Rigidbody2D>() と指定する

これによって
Playerが持っている
Rigidbody2Dを取得することができ、
取得したものを”rb”に入れている

1：プレイヤーの移動

Void Update()について見る

最初の2行では
Void Updateのみで使用できる
Float(小数)型の変数
GetHorizontalと
GetVerticalを宣言している

```
void Update()
{
    float GetHorizontal = Input.GetAxis("Horizontal");
    float GetVertical = Input.GetAxis("Vertical");
```

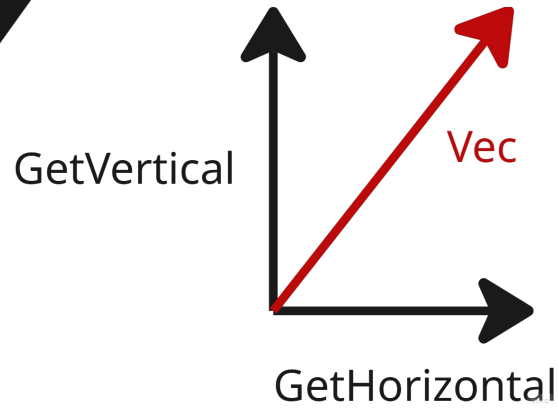
1：プレイヤーの移動

```
void Update()
{
    float GetHorizontal = Input.GetAxis("Horizontal");
    float GetVertical = Input.GetAxis("Vertical");
```

この2つはざっくり言えば
水平方向の入力〔A/D,←/→〕
垂直方向の入力〔W/S,↑/↓〕
を-1～1の値として
受け付けているものである

補足 p69

1：プレイヤーの移動

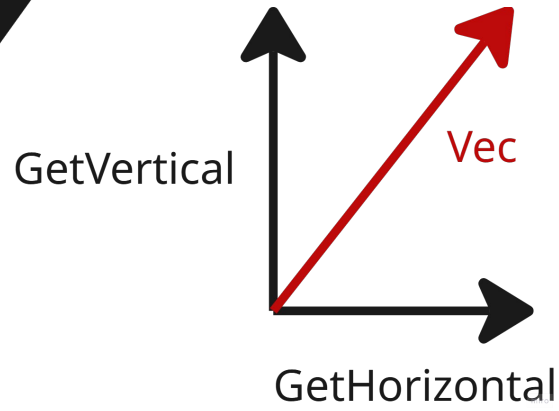


その次には宣言した値を使用して
3次元ベクトル型”Vec”
を用意している
〔z方向は0なので実質二次元〕

```
Vector3 Vec = new Vector3(GetHorizontal, GetVertical, 0);
if (Vec.magnitude > 1)
{
    Vec = Vec.normalized;
}
```

このベクトルを定義することで
プレイヤーの移動方向を定めている

１：プレイヤーの移動



もし用意した”Vec”の大きさが
１以上だと移動速度が速く感じるので

Vec=Vec.normalized

を実行して
”Vec”を単位ベクトル化している

```
Vector3 Vec = new Vector3(GetHorizontal, GetVertical, 0);
if (Vec.magnitude > 1)
{
    Vec = Vec.normalized;
}
```

1：プレイヤーの移動

最後に

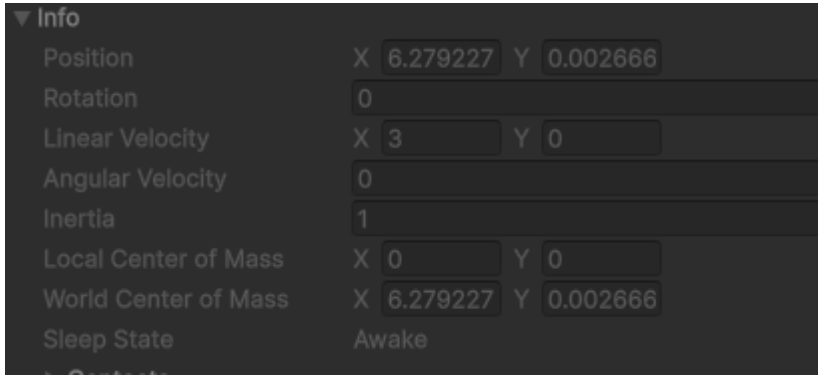
Rb.linearVelocity=Vec*speed

```
rb.linearVelocity = Vec * speed;
```

を実行すると
Rigidbody2D内のvelocityが
“Vec”に”speed”かけたベクトルが代入される
このvelocityがPlayerの動きとなる

1：プレイヤーの移動

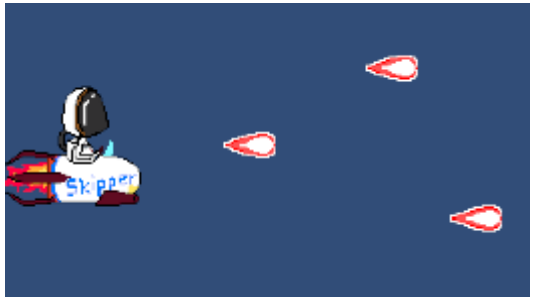
再生中に
Rigidbody2DのInfoを
見てみると
LinearVelocityがあるので
これで確認ができる



2：プレイヤーの攻撃

移動は実装できたので
次に攻撃の方に移ります

シューティングで攻撃する際、
プレイヤーから玉が出てくる感じだと
思うのですが、オブジェクトの種類で
はいくつ必要か？



2：プレイヤーの攻撃

攻撃するために必要な物として

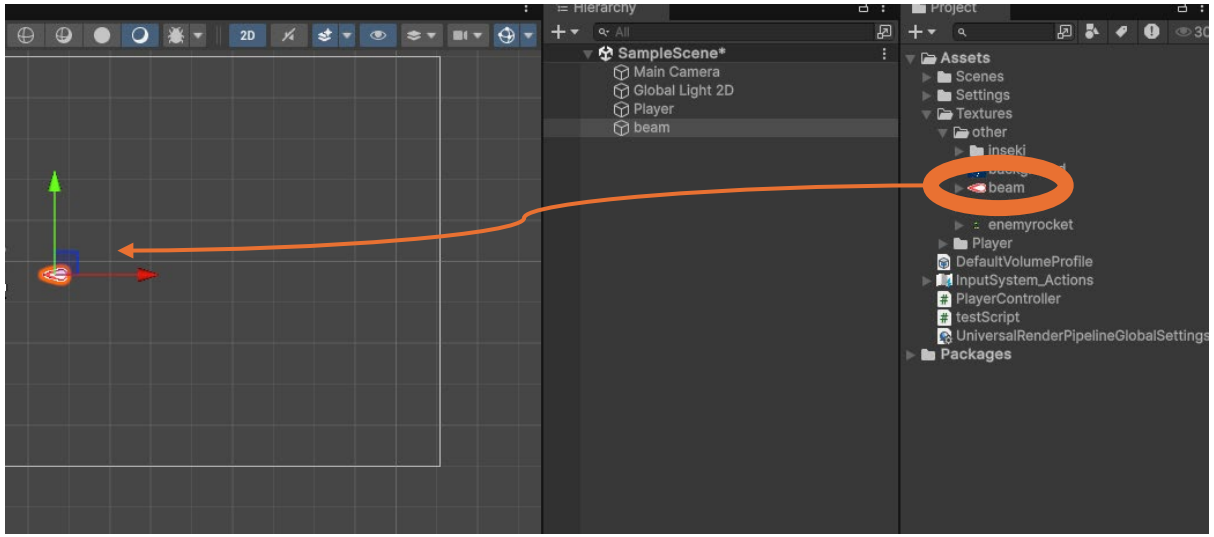
- プレイヤー本体
- 弾

が必要になる
（発射する位置は別として…）

2：プレイヤーの攻撃

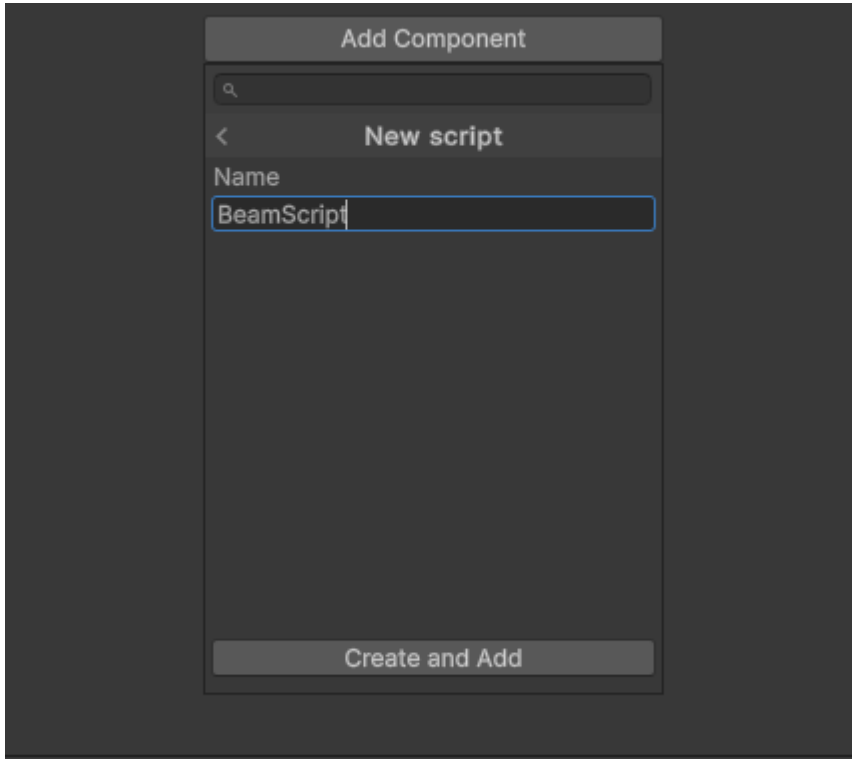
プレイヤーはできているので、
弾の本体を作っていきます

Projectタブにある
Textures/other/beamを
Sceneに投げると
Beamができる



2：プレイヤーの攻撃

Beamを選んで
“Add Component”から
New Scriptで
“BeamScript”を作成する



2：プレイヤーの攻撃

“BeamScript”を開いて
右のコードをうつしておく

```
public class BeamScript : MonoBehaviour
{
    public Vector3 MoveVector;
    // Start is called once before the first execution of Upda
    Unity メッセージ|0 個の参照
    void Start()
    {

    }

    // Update is called once per frame
    Unity メッセージ|0 個の参照
    void Update()
    {
        transform.position += MoveVector * Time.deltaTime;
    }
}
```

2：プレイヤーの攻撃

やっていることは
PlayerControllerより簡単

- 三次元ベクトル型
  “MoveVector”を作成
- 毎フレームごとに
  位置を１秒にMoveVectorぶん
  ずらす

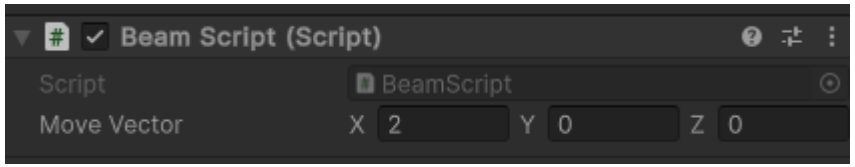
というだけ

補足 p72

```
public class BeamScript : MonoBehaviour
{
    public Vector3 MoveVector;
    // Start is called once before the first execution of Upda
    Unity メッセージ|0 個の参照
    void Start()
    {

    }

    // Update is called once per frame
    Unity メッセージ|0 個の参照
    void Update()
    {
        transform.position += MoveVector * Time.deltaTime;
    }
}
```

2：プレイヤーの攻撃

“BeamScript”をセーブして
Unityに戻り、
MoveVectorの値を設定して
再生すると弾が
設定したベクトルに
動くようになる

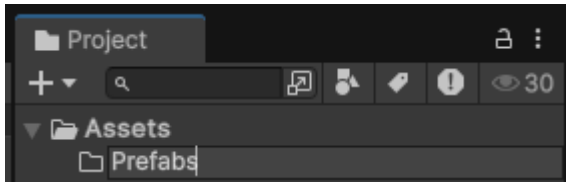


これで弾の本体は終わり

2：プレイヤーの攻撃

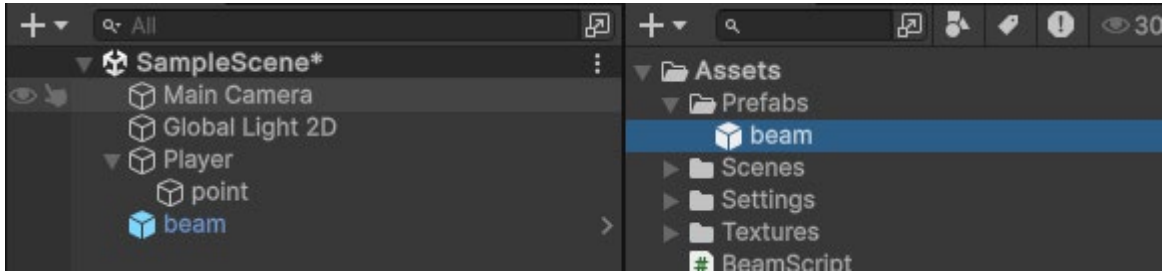
弾がプレイヤーから出ていくように
これから設定していく

その前にProjectタブに
フォルダを追加しておく
+のところから
Folderで新規フォルダが作成される
名前は”Prefabs”とおいておく



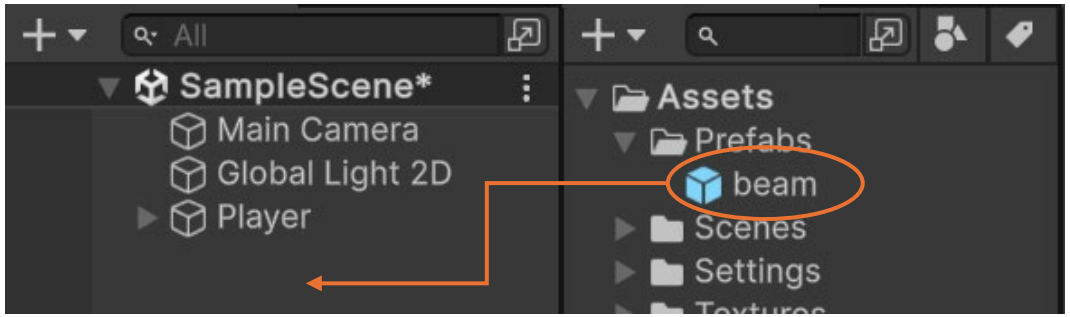
2：プレイヤーの攻撃

作ったフォルダに
Beamをドラック&ドロップすると
オブジェクトがなんか青くなる



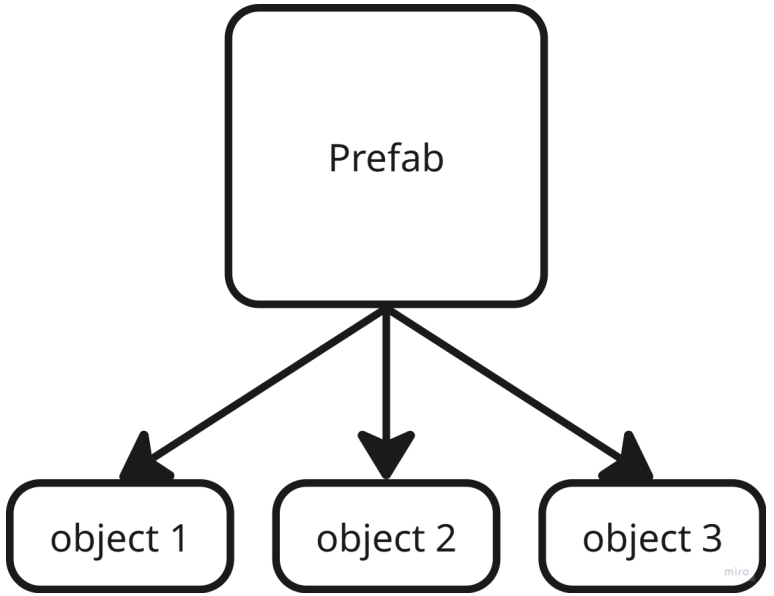
これはオブジェクトがデータとして
保存されたことを指しており、
青くなったオブジェクト
を”Prefab”[プレファブ]と呼ぶ

2：プレイヤーの攻撃



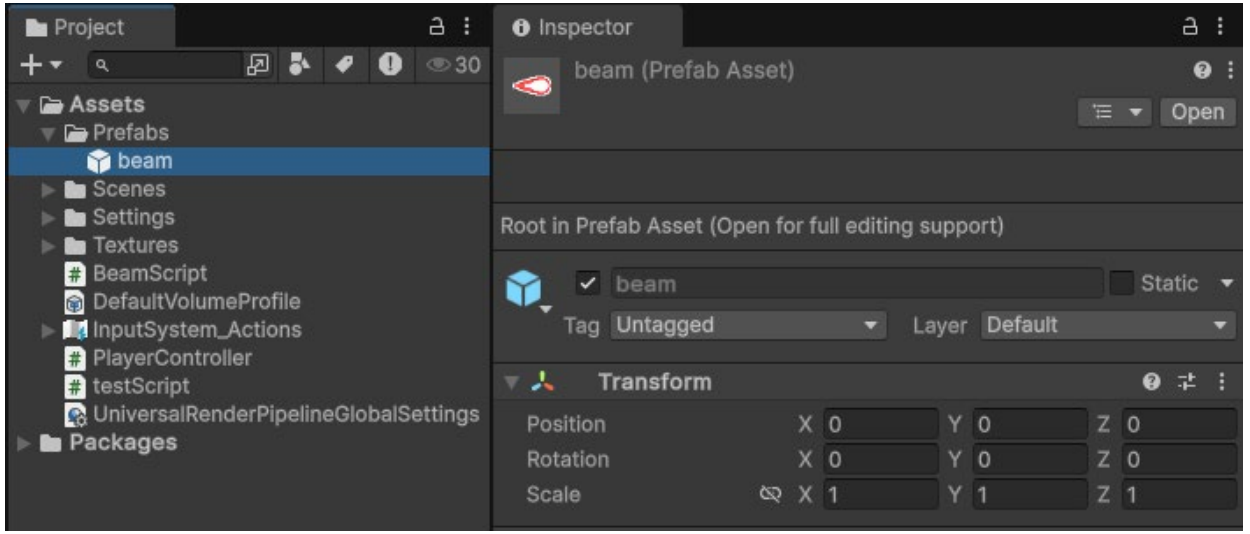
Prefabにすると
HierarchyにあるbeamをDeleteしても
Projectタブから復元が可能

さらにPrefabをベースに
同じオブジェクトを複製する
ことができる
（今回はこれをやる）



2：プレイヤーの攻撃

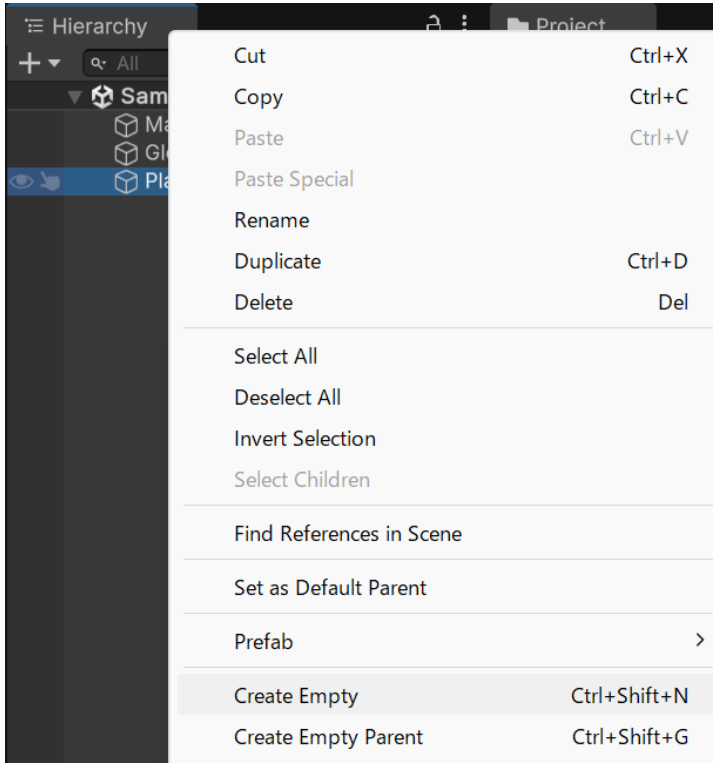
その前にPrefabの
位置を0にしないと
この後うまくいかないときが
あるので
ここでpositionを
全て0に設定しておく



2：プレイヤーの攻撃

弾を発射するための位置を作る

Playerを選択した状態で
Playerにカーソルを合わせて右クリック
そこから”Create Empty”を選択



2：プレイヤーの攻撃

選択すると
Playerの中に新しくオブジェクトが
生成される
名前は”point”にしておく

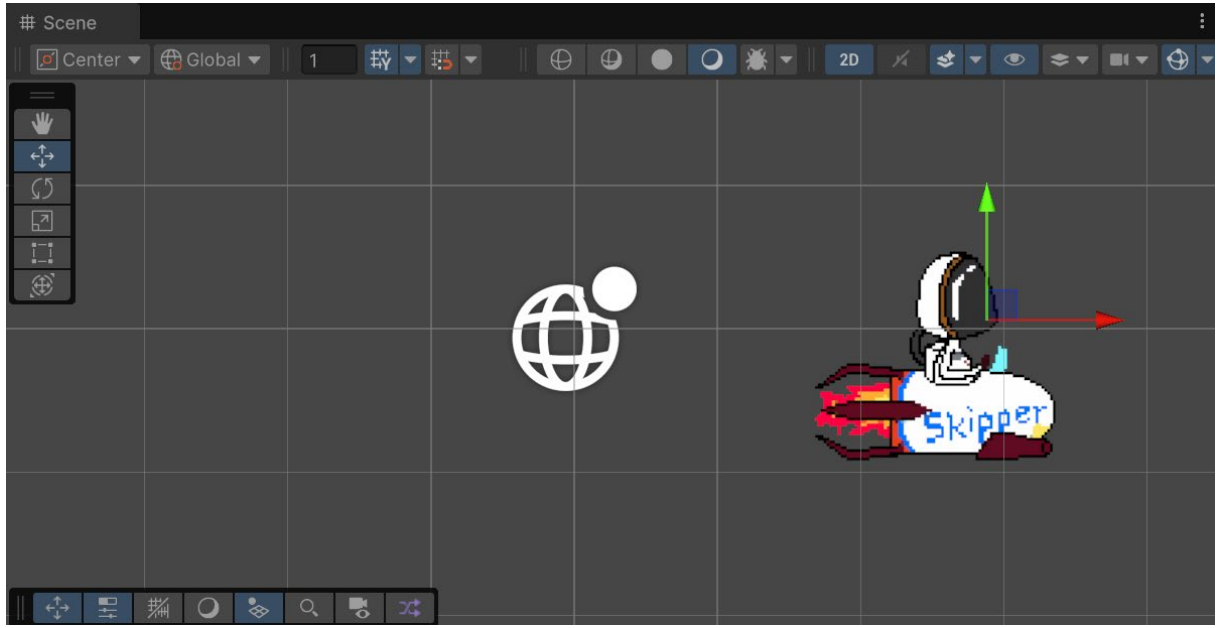
2：プレイヤーの攻撃

このように
オブジェクトの中にオブジェクトがある
状態のことを”親子関係“といい、
元のオブジェクトが親オブジェクト、
下のオブジェクトが子オブジェクトと
言ったりする

2：プレイヤーの攻撃

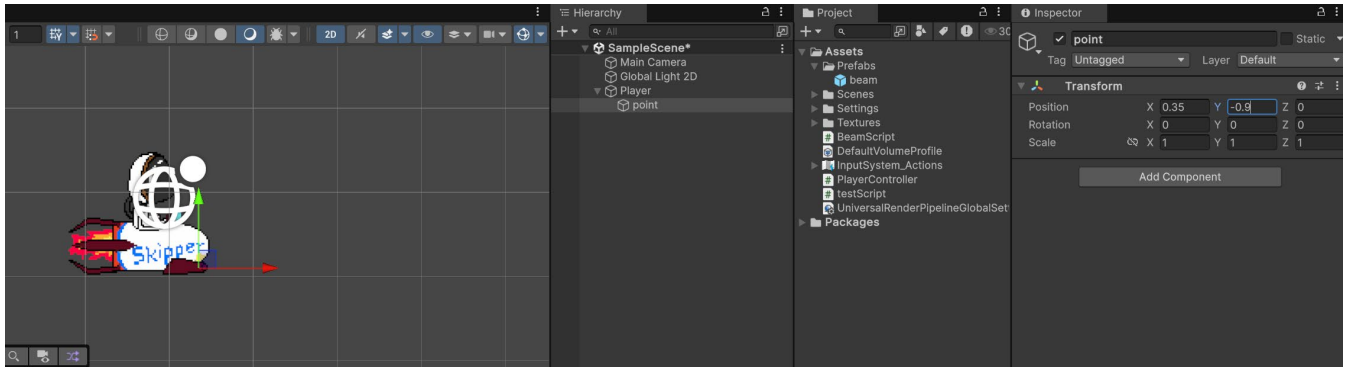
親子関係の利点としては
親オブジェクト[Player]が移動すると
子オブジェクトも一緒に動いてくれる
ところである

試しに再生して動かしてみると
Pointの位置がPlayerと同じなのが
確認できる



2：プレイヤーの攻撃

Pointの座標を
x:0.35
y:-0.9
に設定しておく

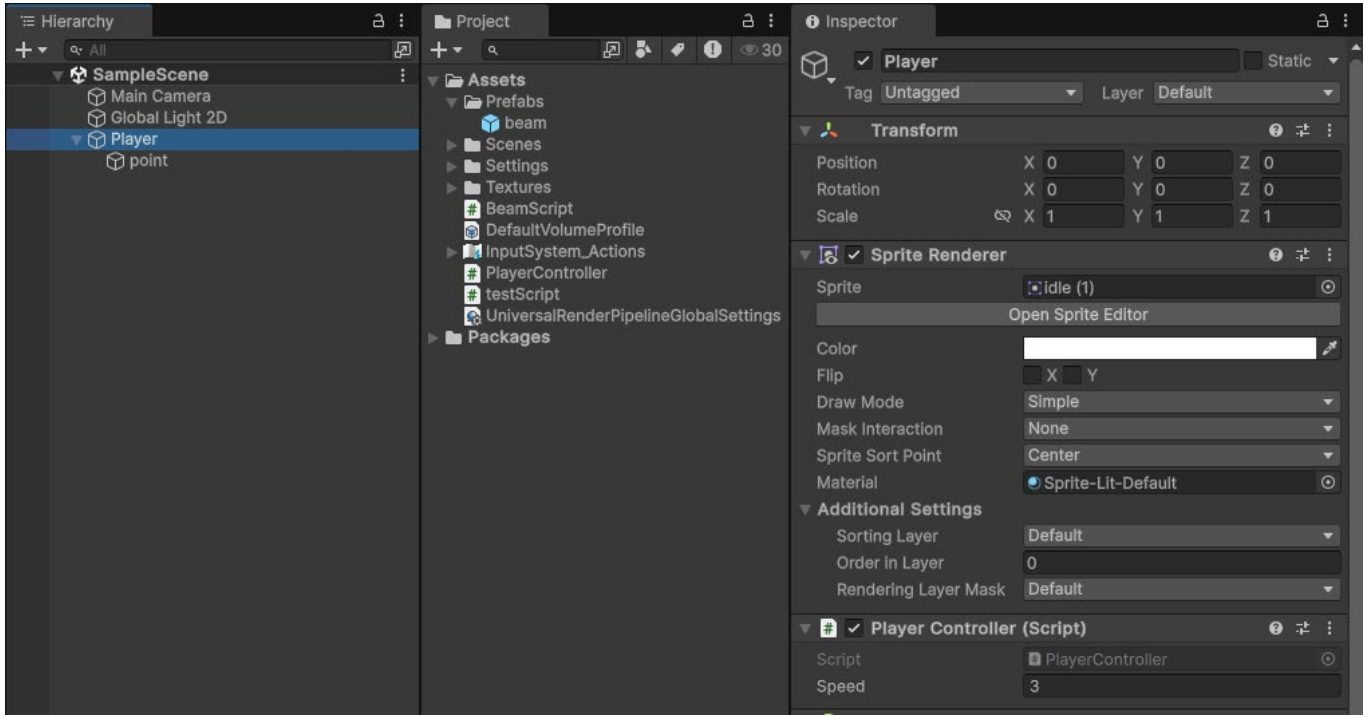


子オブジェクトの座標は
親オブジェクト本体からどれくらい離れて
いるかを示す”相対的な”座標となっている

2：プレイヤーの攻撃

Playerから弾を発射するための
設定をする

Playerを選び、
PlayerControllerを開く



2：プレイヤーの攻撃

PlayerControllerに
囲まれた部分を
追記する

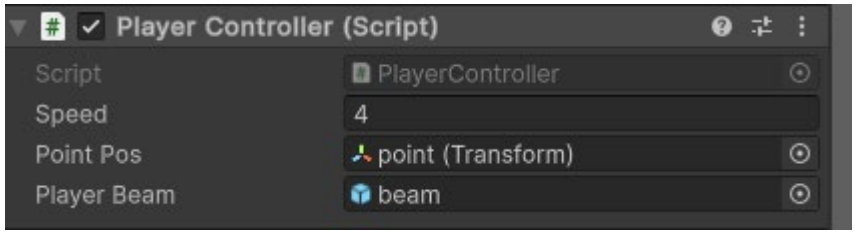
追記したらセーブをして
Unityに戻る

```
public class PlayerController : MonoBehaviour
{
    Rigidbody2D rb;
    public int speed;

    public Transform PointPos;
    public GameObject PlayerBeam;
    // Start is called once before the first execution of Update after the MonoBehaviour is
    Unity メッセージ | 0 個の参照
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    // Update is called once per frame
    Unity メッセージ | 0 個の参照
    void Update()
    {
        float GetHorizontal = Input.GetAxis("Horizontal");
        float GetVertical = Input.GetAxis("Vertical");

        Vector3 Vec = new Vector3(GetHorizontal, GetVertical, 0);
        if (Vec.magnitude > 1)
        {
            Vec = Vec.normalized;
        }

        rb.linearVelocity = Vec * speed;

        if (Input.GetKeyDown(KeyCode.Space))
        {
            Instantiate(PlayerBeam, PointPos.position,Quaternion.identity);
        }
    }
}
```

2：プレイヤーの攻撃

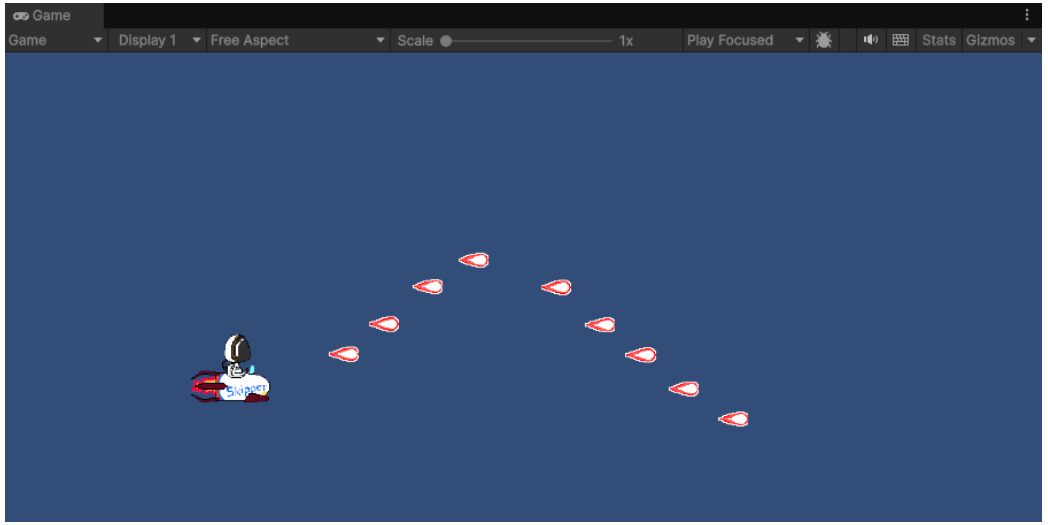
戻ると
PlayerControllerに
“PointPos”と
“PlayerBeam”が追加されている

“PointPos”には
Playerの子オブジェクトのpointを、
“PlayerBeam”には
PrefabのBeamを入れる



2：プレイヤーの攻撃

再生して
スペースキーを押すと
Playerの下の部分から
弾が出るようになります🙌



Playerが弾より速い場合は
- PlayerContollerのspeedを上げる
- PrefabのBeamのmoveVectorの値を上げる

と処理をするとよくなります！

2：プレイヤーの攻撃

ひとまず、これでプレイヤーの操作は
あらかた終わりました

次回は敵の製作と
Beamの当たり判定の話になります

がんばって続きを作るね…！！

Unity初心者講座

1 プレイヤーの操作

終わり

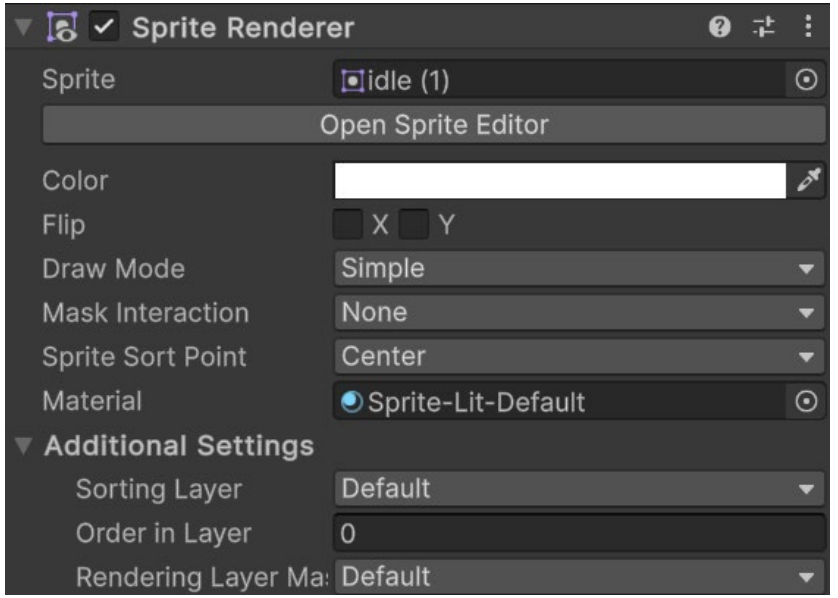
ここからは補足です

3：補足

Sprite Rendererについて

Sprite Rendererとは
Spriteを画面に表示するための
コンポーネントのこと

Spriteは画像のこと
画像をもとに色を乗算で入れたり
画像を反転することができる

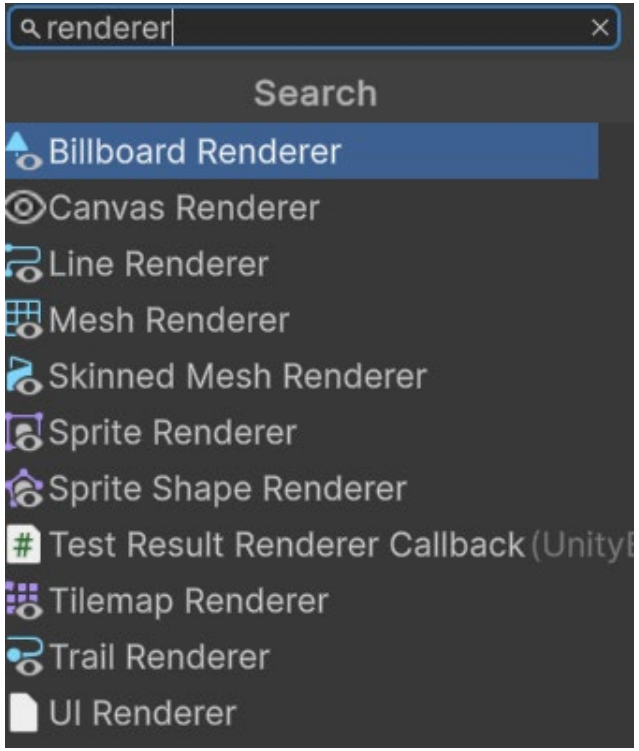


3：補足

Sprite Rendererについて

Sprite Rendererに限らず、
画面に何か表示するための物として
Renderer〔れんだらー〕と呼ばれている

Sprite Rendererは画像を、
Mesh Rendererは3DCGで作った
メッシュを表示するために使用される

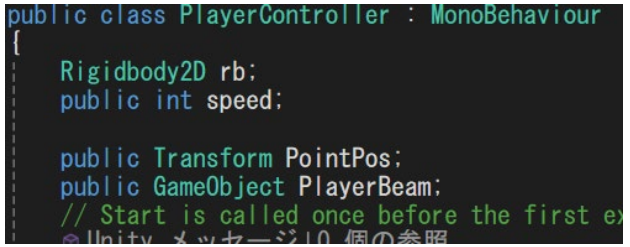


3：補足

publicについて

変数やインスタンスを宣言する際、
Publicを頭につけていたが
つけた場合とつけない場合がある

頭にpublicをつけてない場合はprivateと呼ぶが、
publicとprivateの違いを
知っておこう



3：補足

publicについて

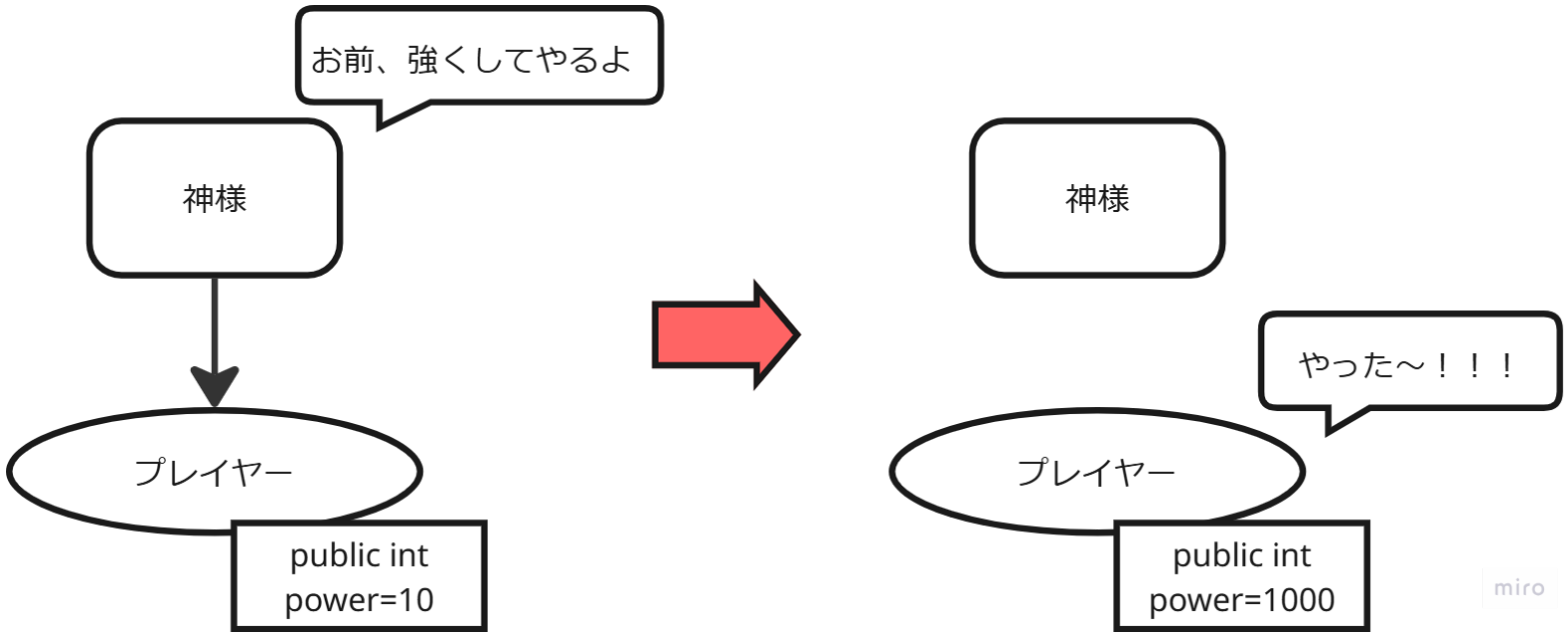
publicとprivateの大きな違いとしては
- 他のクラス、オブジェクトに干渉可能なのか？
  - Publicは干渉できる
  - Privateは干渉できない

がある

3：補足

publicについて

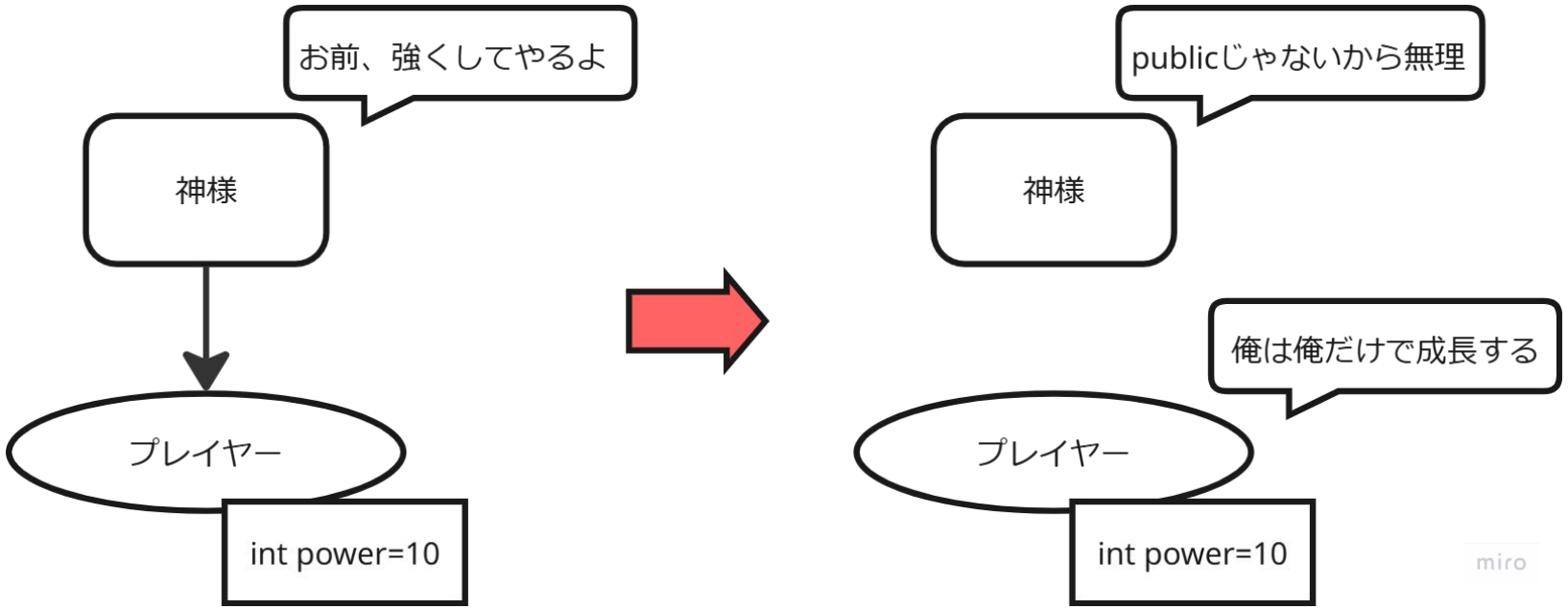
イメージはこう（Publicの場合）



3：補足

publicについて

イメージはこう（Privateの場合）



3：補足

publicについて

publicならUnity上で見れる

privateは干渉できない代わりに
使用できる変数、インスタンスを区別することができる

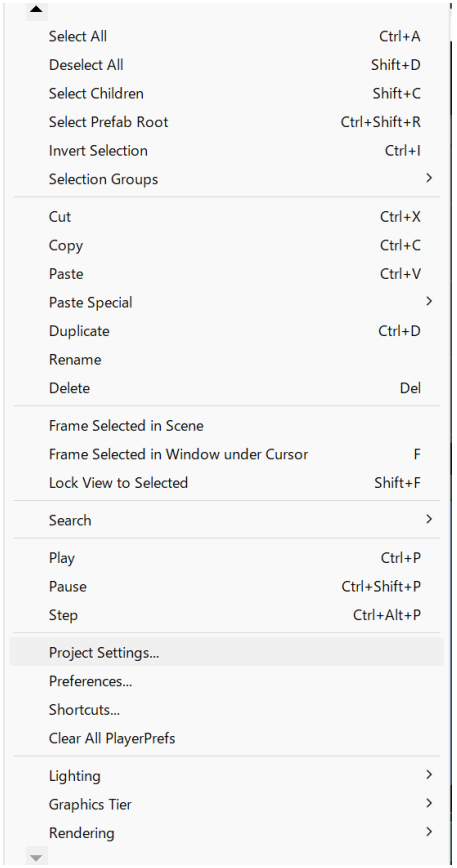
他、メリットはそれぞれあるので
使い分けることをお勧めします

3：補足

GetAxisについて

GetAxisで”Horizontal”など指定すると値が-1～1で取得できる、
と説明したがなぜできるのだろう？

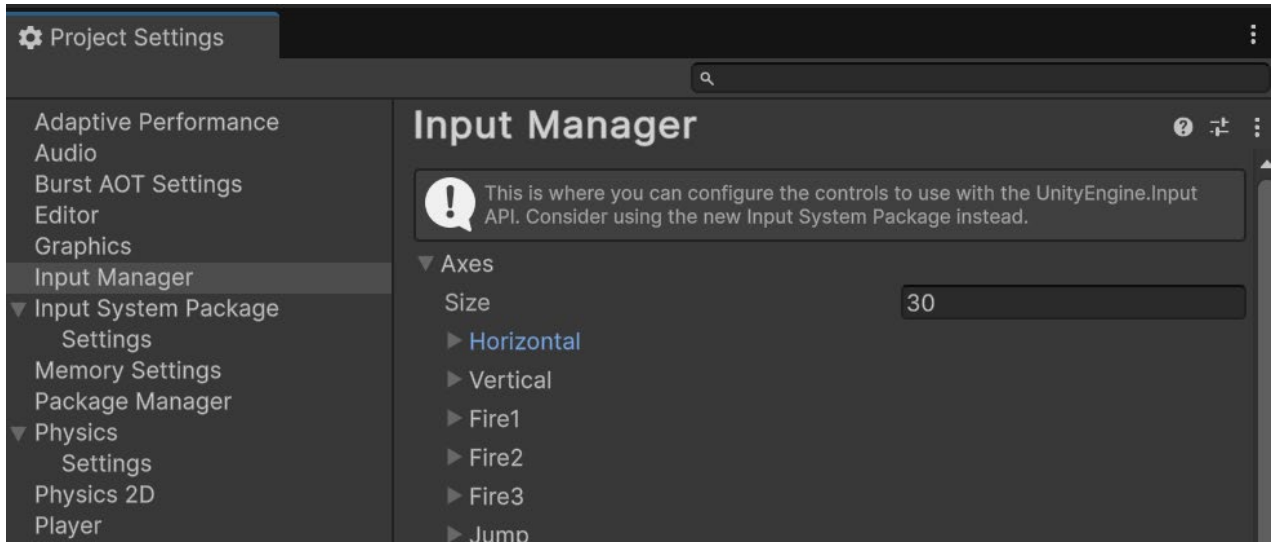
UnityのProject Settingsを見てみよう



3：補足

GetAxisについて

Project Settingsの中から
“Input Manager”を見て
“Axes”を開いてみると
いろんな項目が出てくるが、
その中に
“Horizontal”や”Vertical”が
あることが確認できる



3：補足

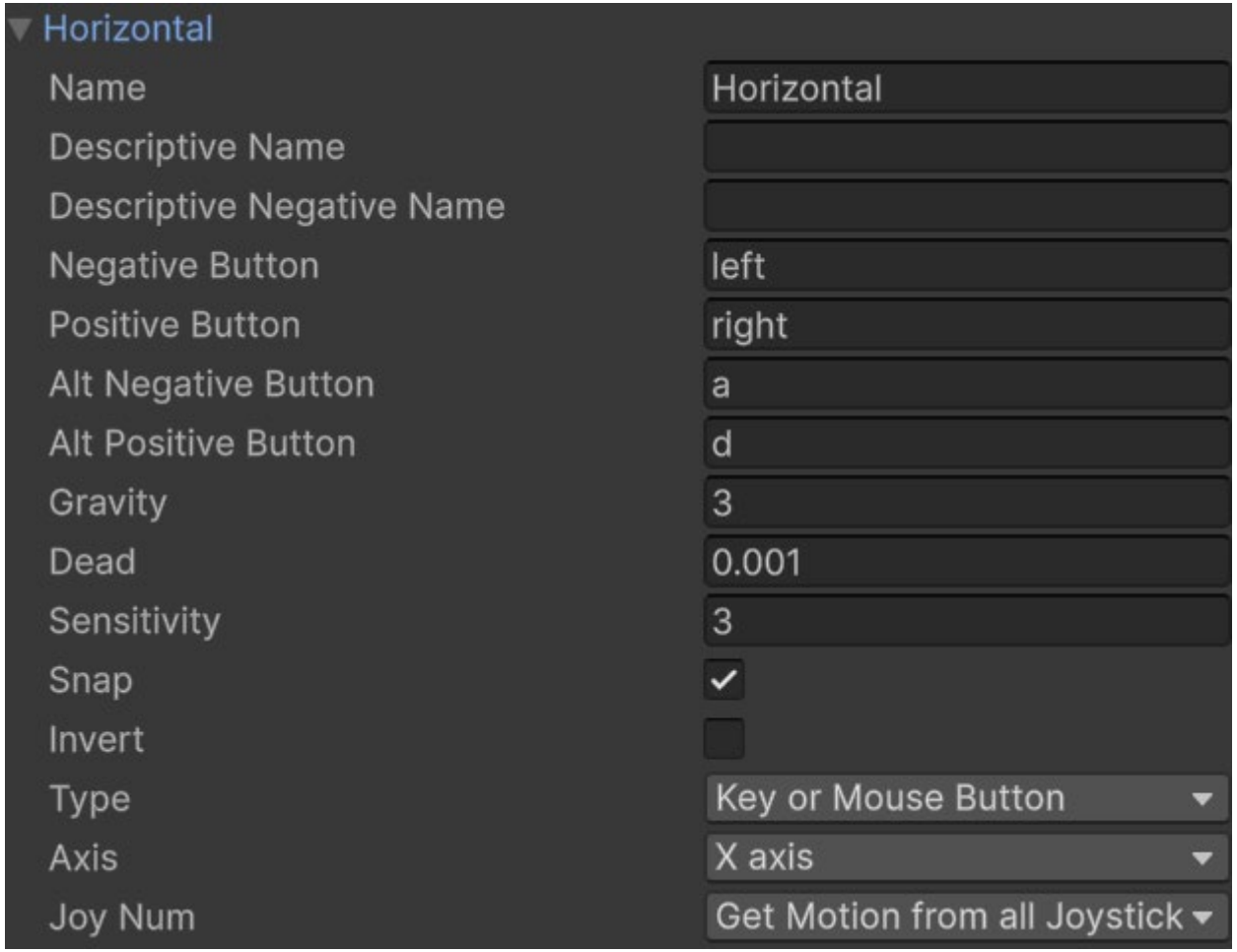
GetAxisについて

その中の“Horizontal”を見てみる
〔どれも中身は同じである〕

ざっくり説明すると
- 名前
- 対応するボタン、キー2組
- 重さ
- 無効
- 感度

の値などが変更できる
これで動きの強弱を調節することができる

詳しくはマニュアルから調節してみよう



3：補足

位置について

全てのオブジェクトには
“Transform”が備わっている

“Transform”には
・位置〔position〕
・回転〔rotation〕
・大きさ〔scale〕
がついていて、これで位置などを
処理している