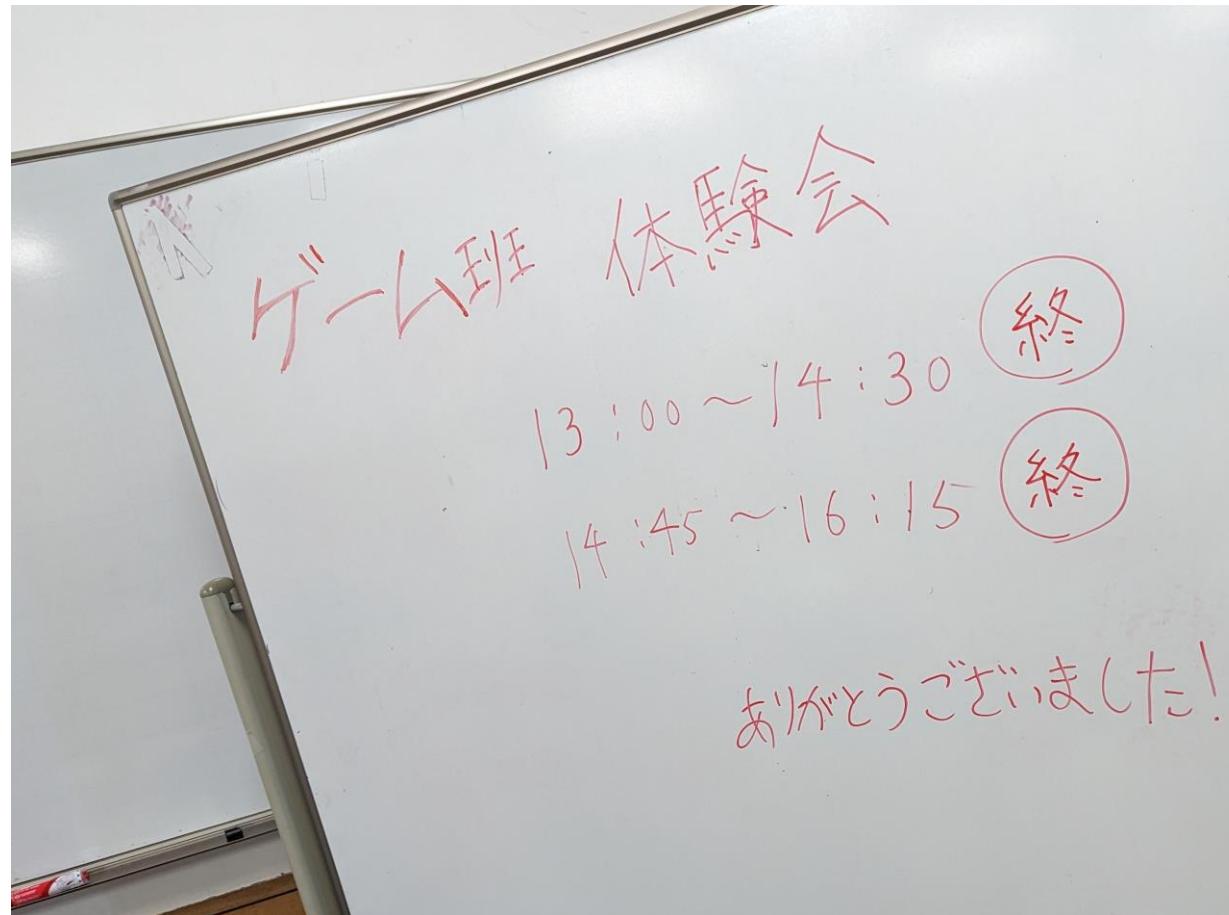


# ゲーム班体験会 資料

2023/05/16

2023/05/16に行ったゲーム班の体験会で  
行ったことについてまとめました



## ・ゲーム班の活動概要

ゲーム班では年2回  
ゲームの共同制作を行っている

夏休みと冬休みの  
長期休暇の間に制作を行い  
それぞれ金大祭・  
新入生歓迎期間に  
発表する

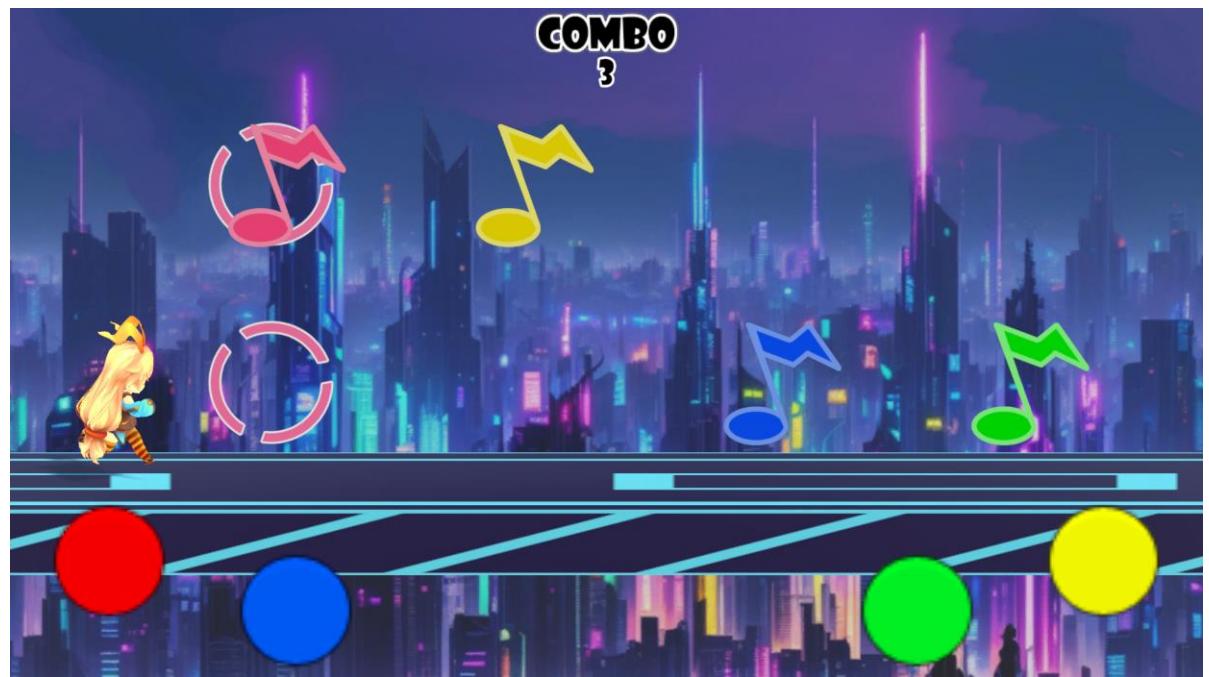


↑金大祭の展示の様子  
中央左側のポスターのゲーム「金大登山」では金大祭2日目にタイマー機能が追加されたことで、凄腕プレイヤー達はより速いタイムを求め熾烈な争いを繰り広げた  
なんでそうなった?????????

# 今年の新入生歓迎会のために作られたゲーム

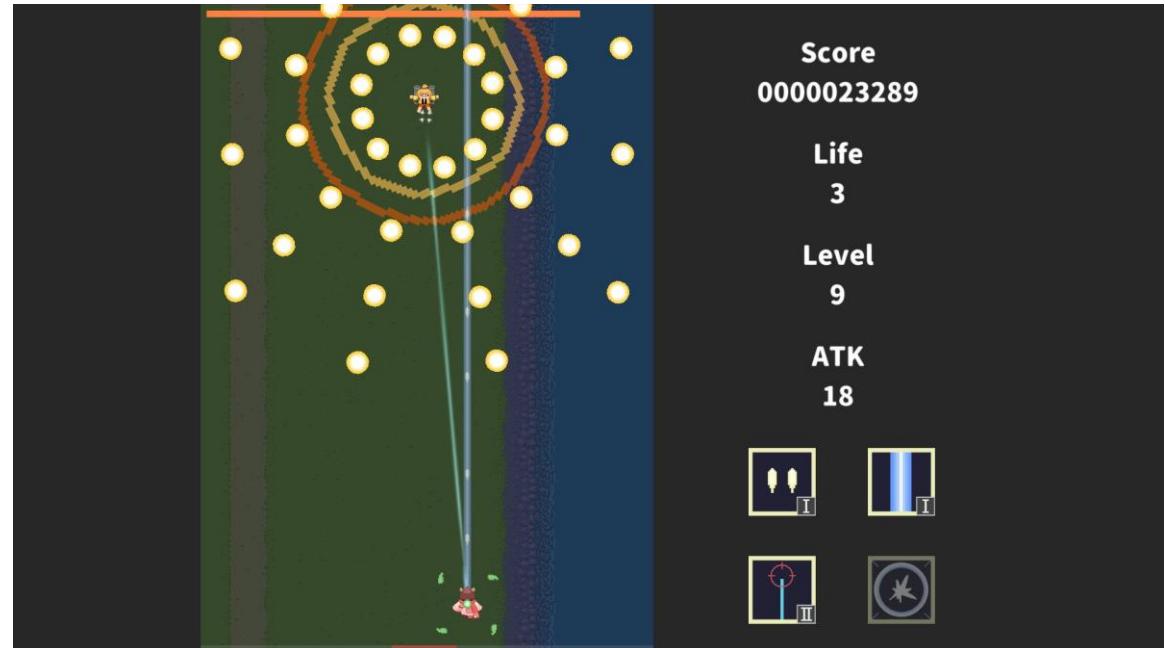


音ゲー : Cathm!!!(キャズム)

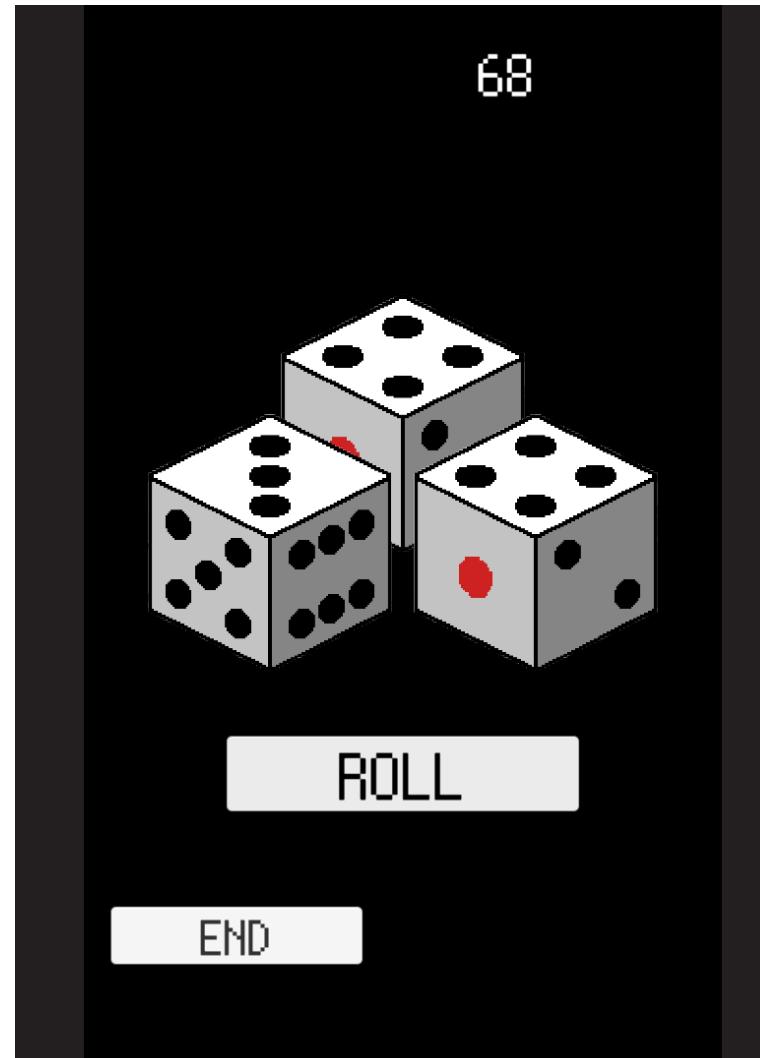


# 今年の新入生歓迎会のために作られたゲーム

## シューティングゲーム



# 今年の新入生歓迎会のために作られたゲーム



ダイスロール  
ゲーム:  
Finish before "1"

そんな感じで基本は1年に2回ほど集まって  
共同制作するって感じです

ただ流石に何もわからない状態でさあ  
共同制作しよう！ってなるのはアレな  
ので今年の次の学祭まではプログラミ  
ング教えるなりチュートリアルとして  
何かゲームを作ってもらうなりあると  
思います

まあまだ細かくは未定なので…



で！ ここから肝心のゲーム制作についての話です

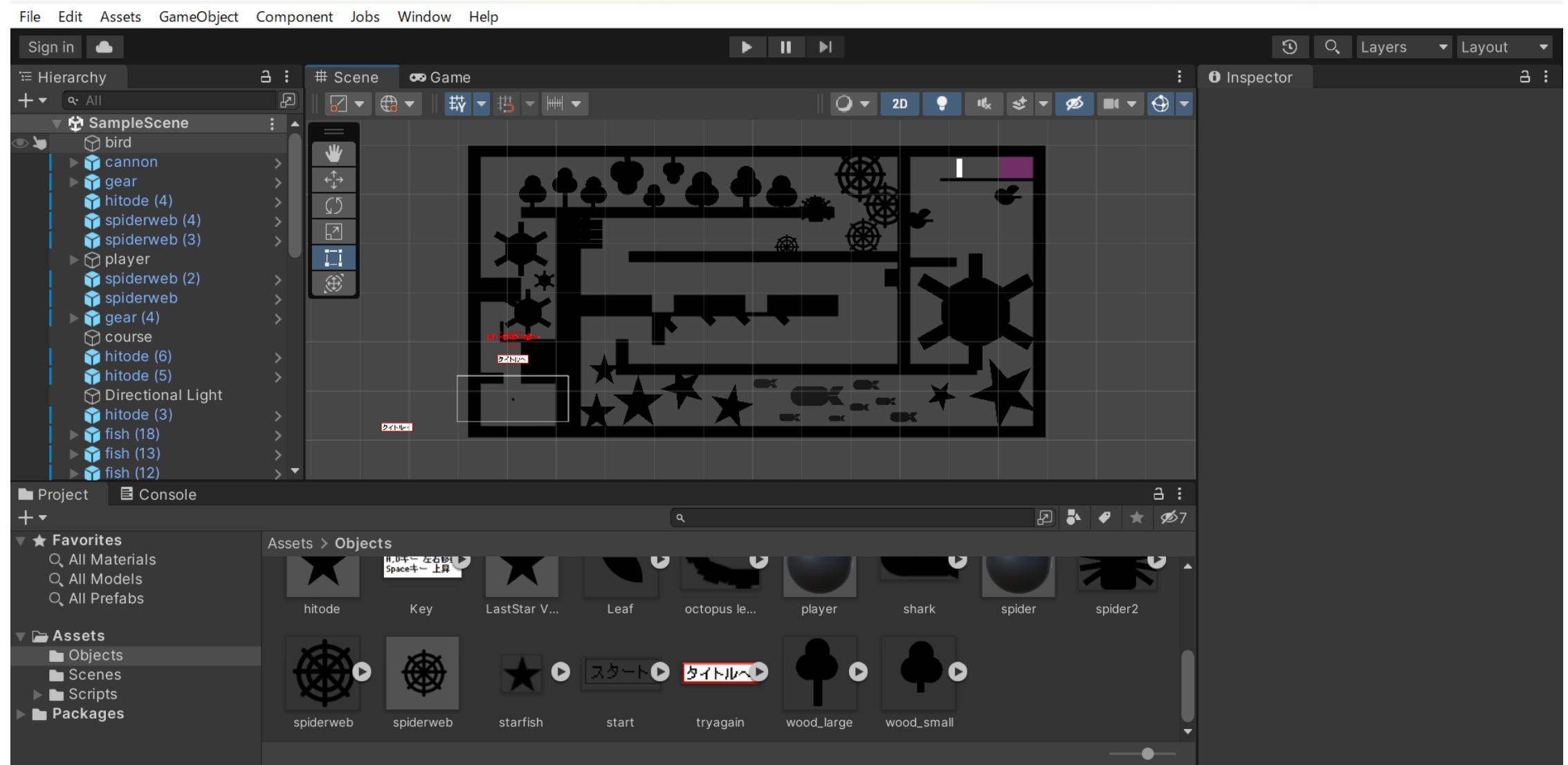
# ゲーム制作についての話

ゲーム班では基本的に”Unity”というゲームエンジン  
を用いて作業をします

結構普及しているゲームエンジンで現在  
色んなゲームがUnityを用いて作られています

なんか調べたら  
ポケモンGOとか  
ウマ娘とかもUnityで  
作られてるらしい  
すごい(小学生並みの感想)

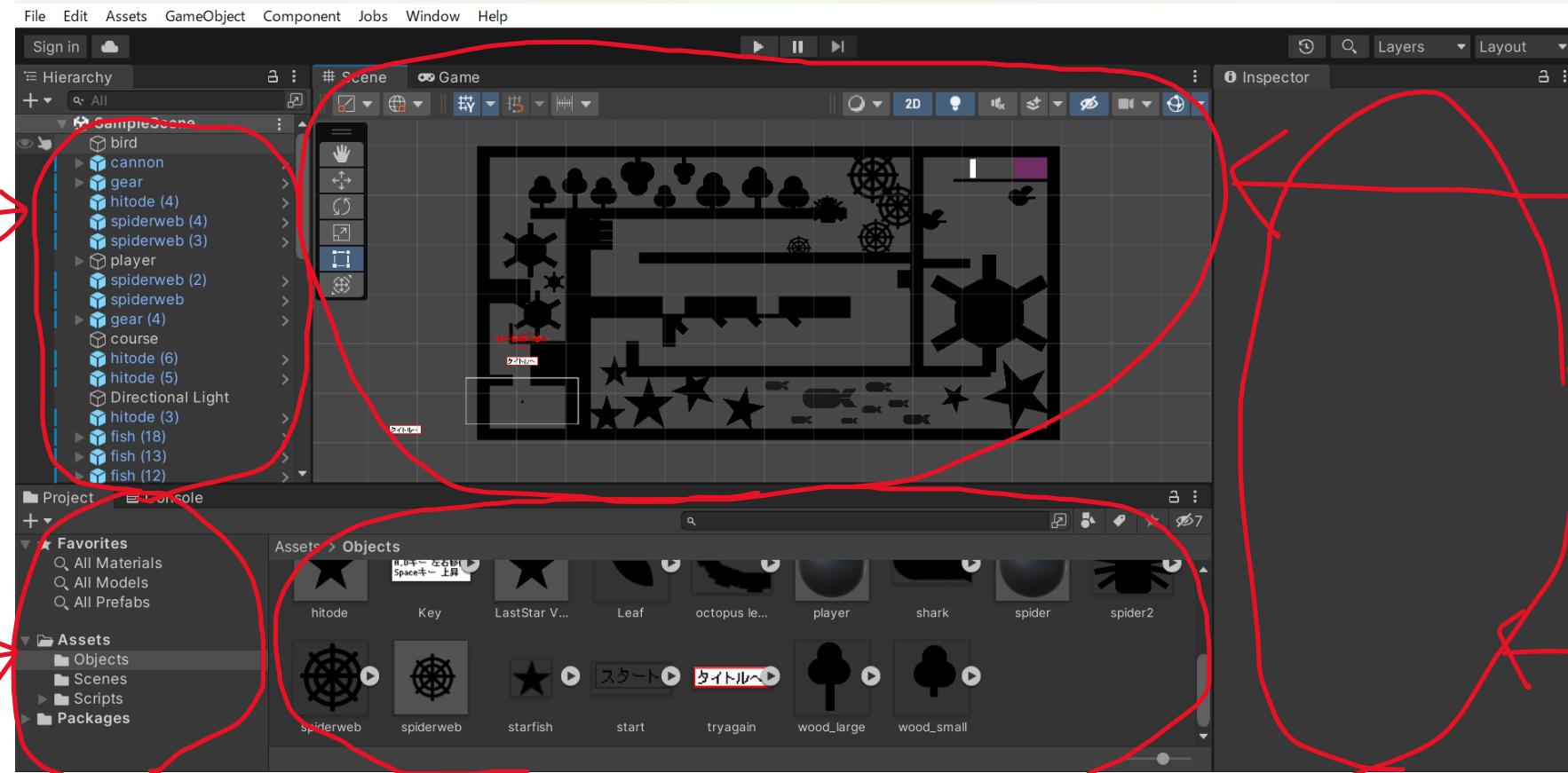
↓ Unityの画面（2年前に学祭で発表したゲーム）



# なんとなくの各ウィンドウの説明

ゲーに  
出でくる  
モノの一覧

素材フォルダ  
一覧



↑  
素材フォルダの中身

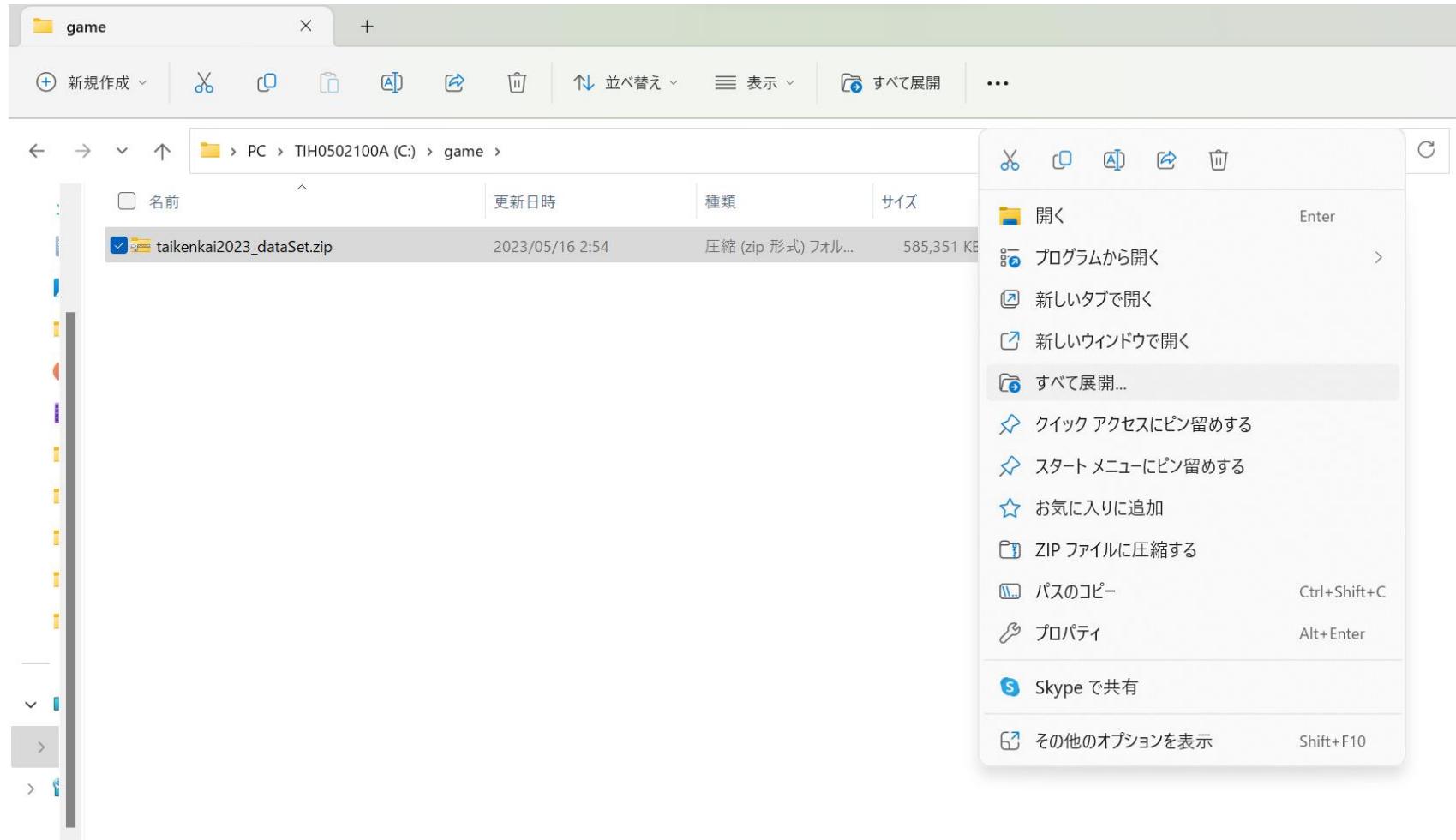
ゲーに  
出でる  
ところ

ゲーに  
出でる  
モノの  
説明

ということで実際に作ってみましょう！

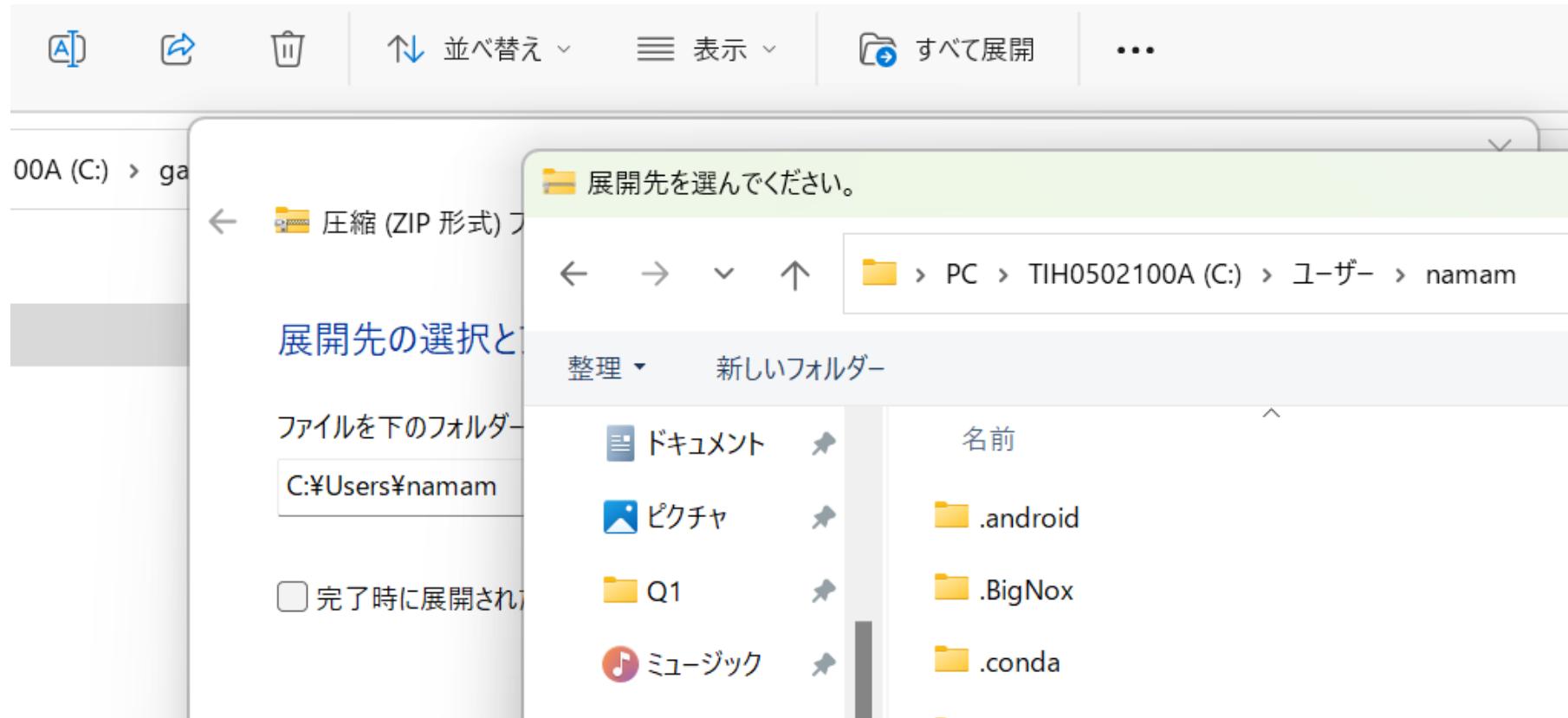
用意したデータ「unityCASruTutorial-main.zip」を解凍して、そのフォルダをPC→Cドライブ→ユーザー→(自分のユーザー名)となるところにおいて下さい

用意したデータ「taikenkai2023\_dataset.zip」を解凍して、そのフォルダをPC→Cドライブ→ユーザー→(自分のユーザー名)となるところにおいて下さい



フォルダを  
右クリック  
→すべて展開

用意したデータ「taikenkai2023\_dataset.zip」を解凍して、そのフォルダをPC→Cドライブ→ユーザー→(自分のユーザー名)となるところにおいて下さい



展開先をPC  
→Cドライブ  
→ユーザー  
→(自分のユーザー  
名)にして展開

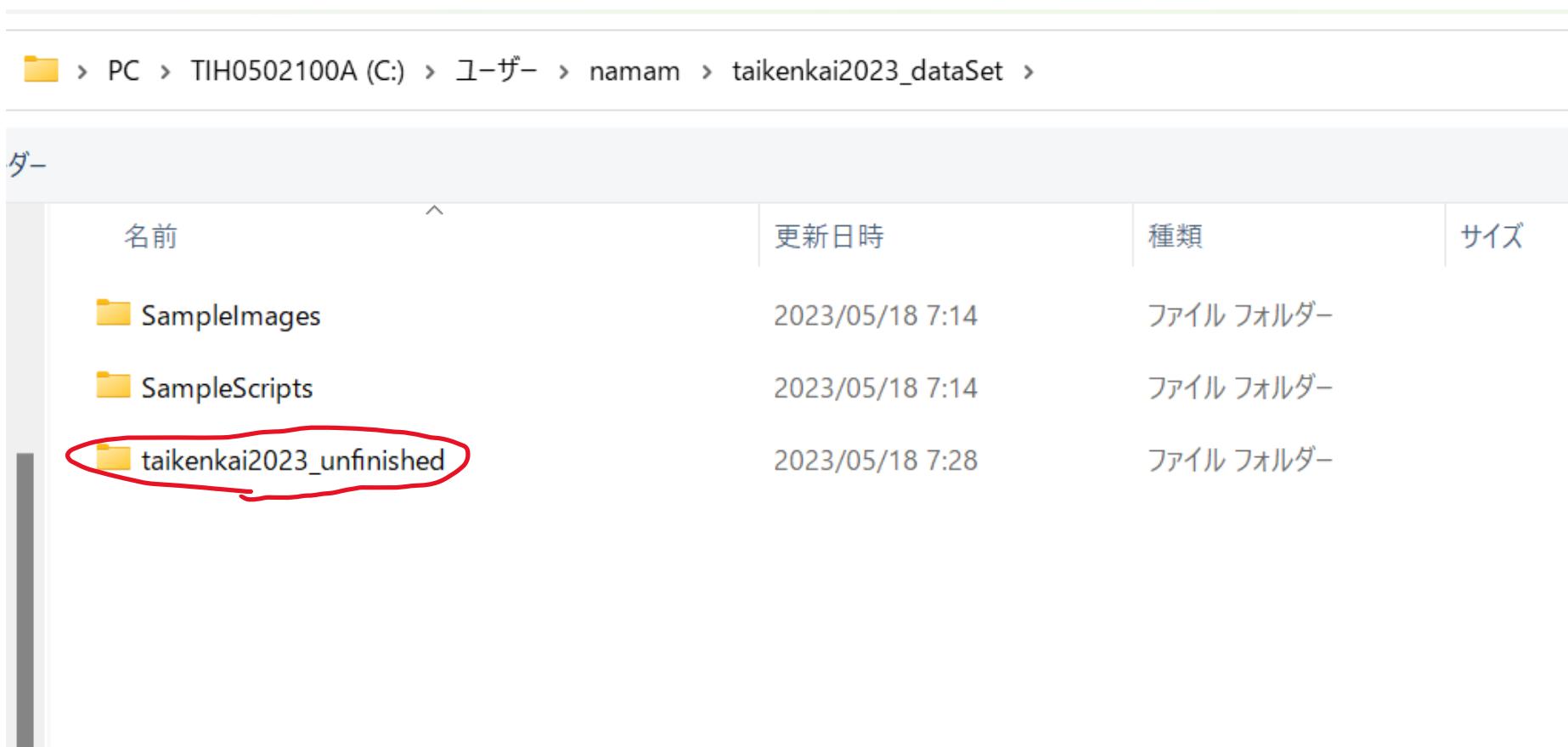
後は待ちましょう

途中エラー吐いたら  
「スキップ」押してください

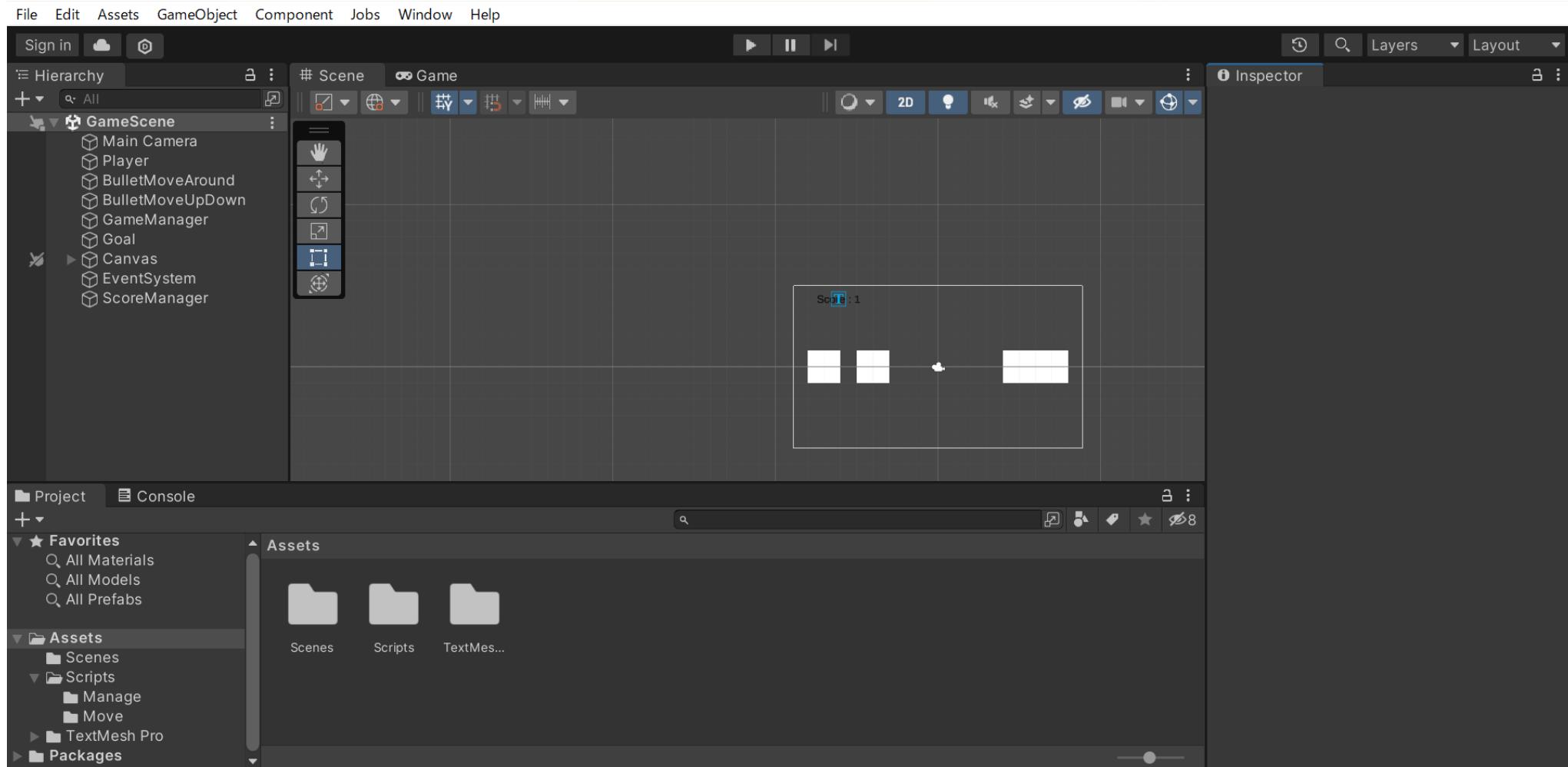
# Unity Hubを開いて「開く」をクリックします



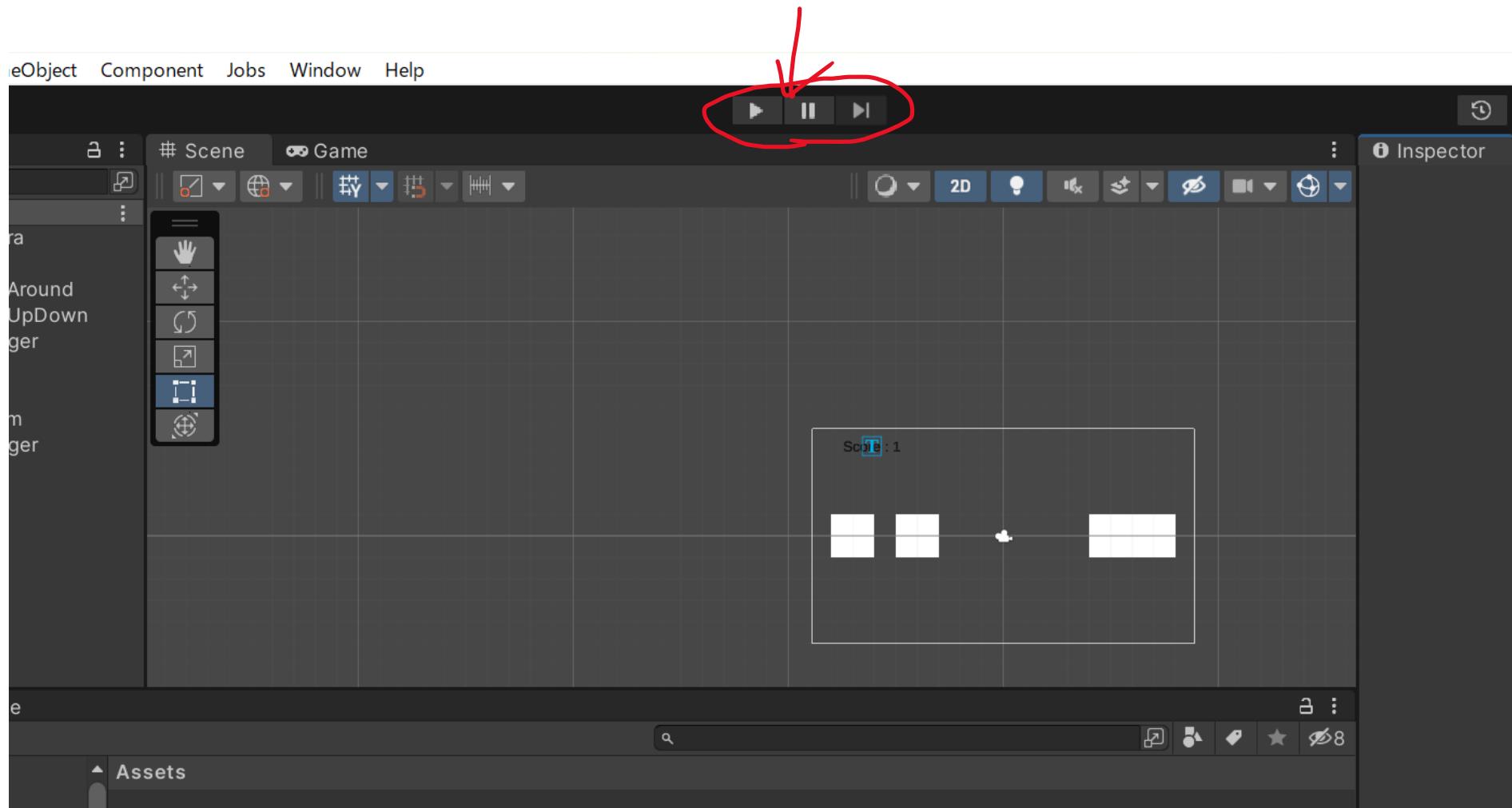
さっき開いたデータセットから  
「taikenkai2023\_unfinished」を選択して  
「開く」をクリック



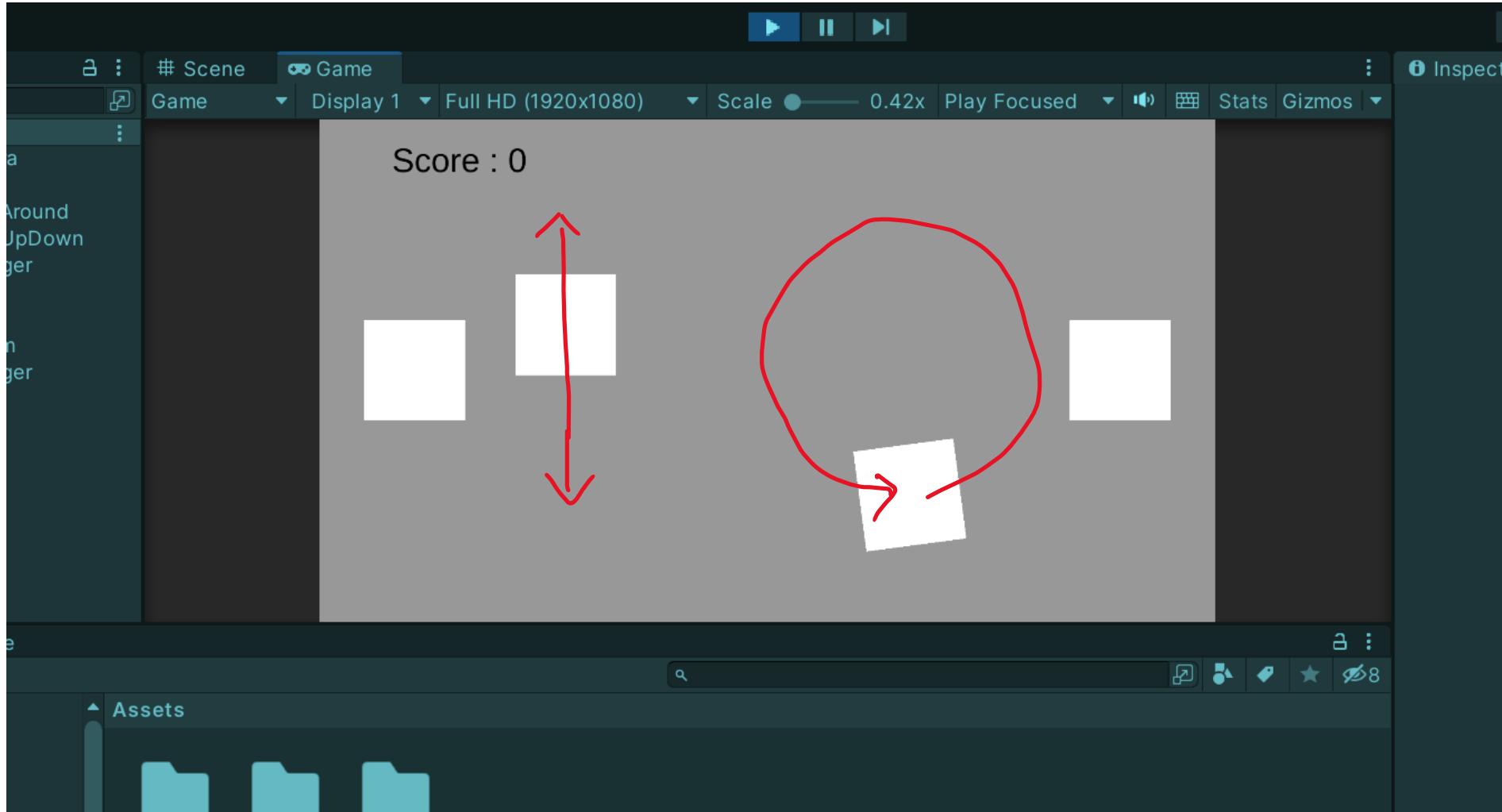
こんな画面が表示されると思います



ここで実際に作ったゲームを動かせます  
試しに再生ボタンをクリックして動かしてみてください

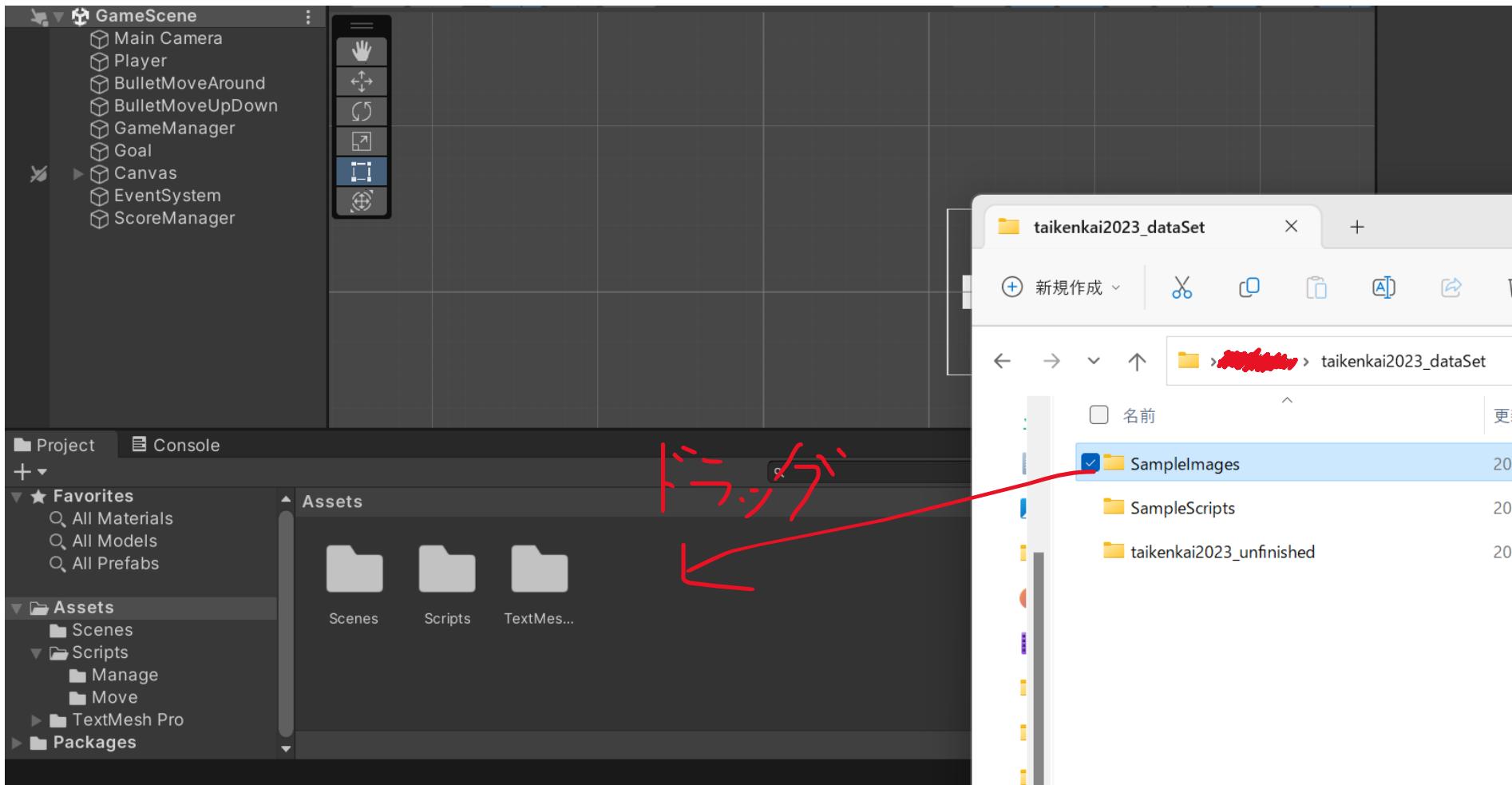


こういう風に動くと思います

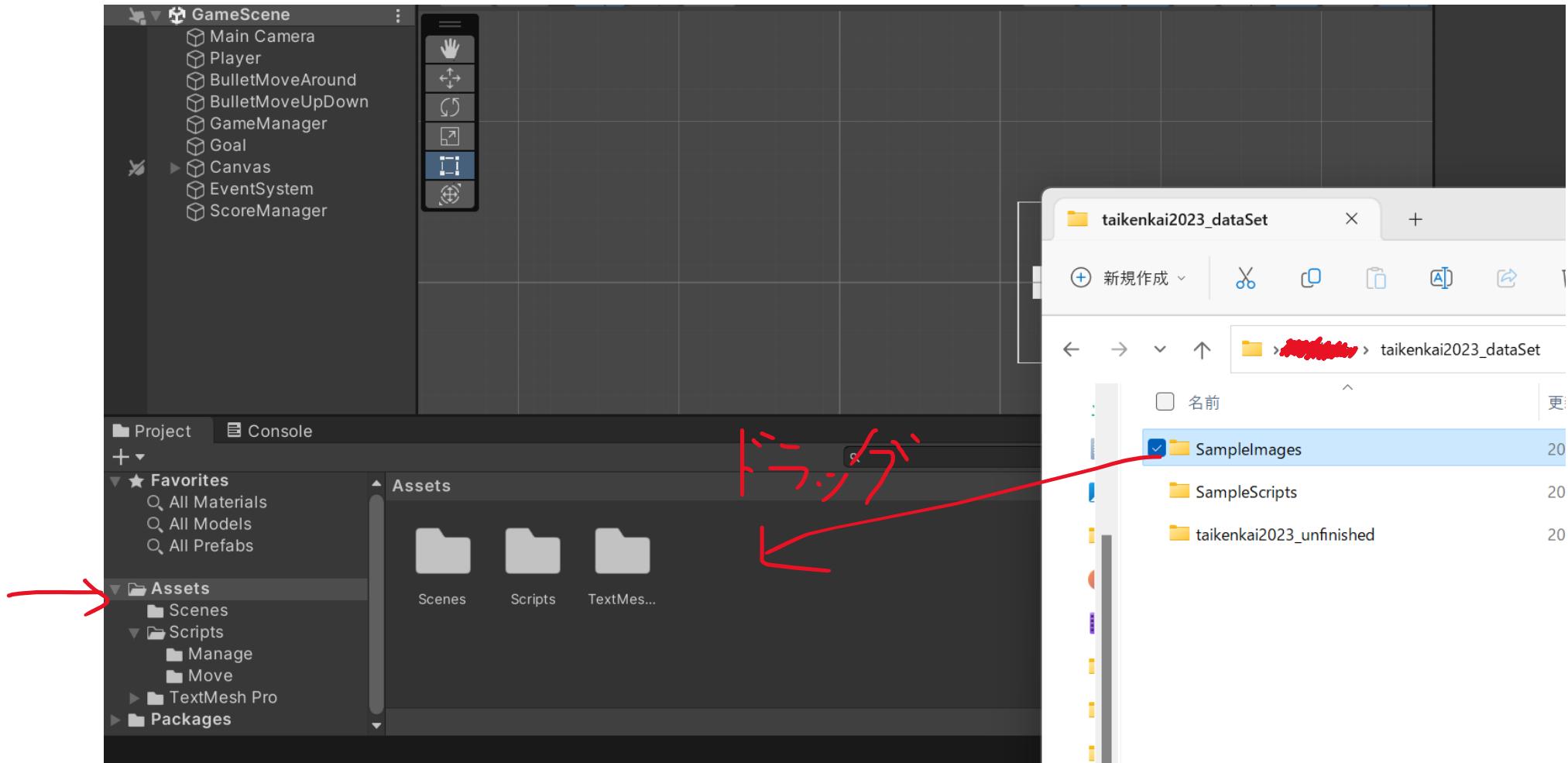


こんな風にゲームがある程度できているのですが  
今回はこれに手を加えて完成させてもらいます

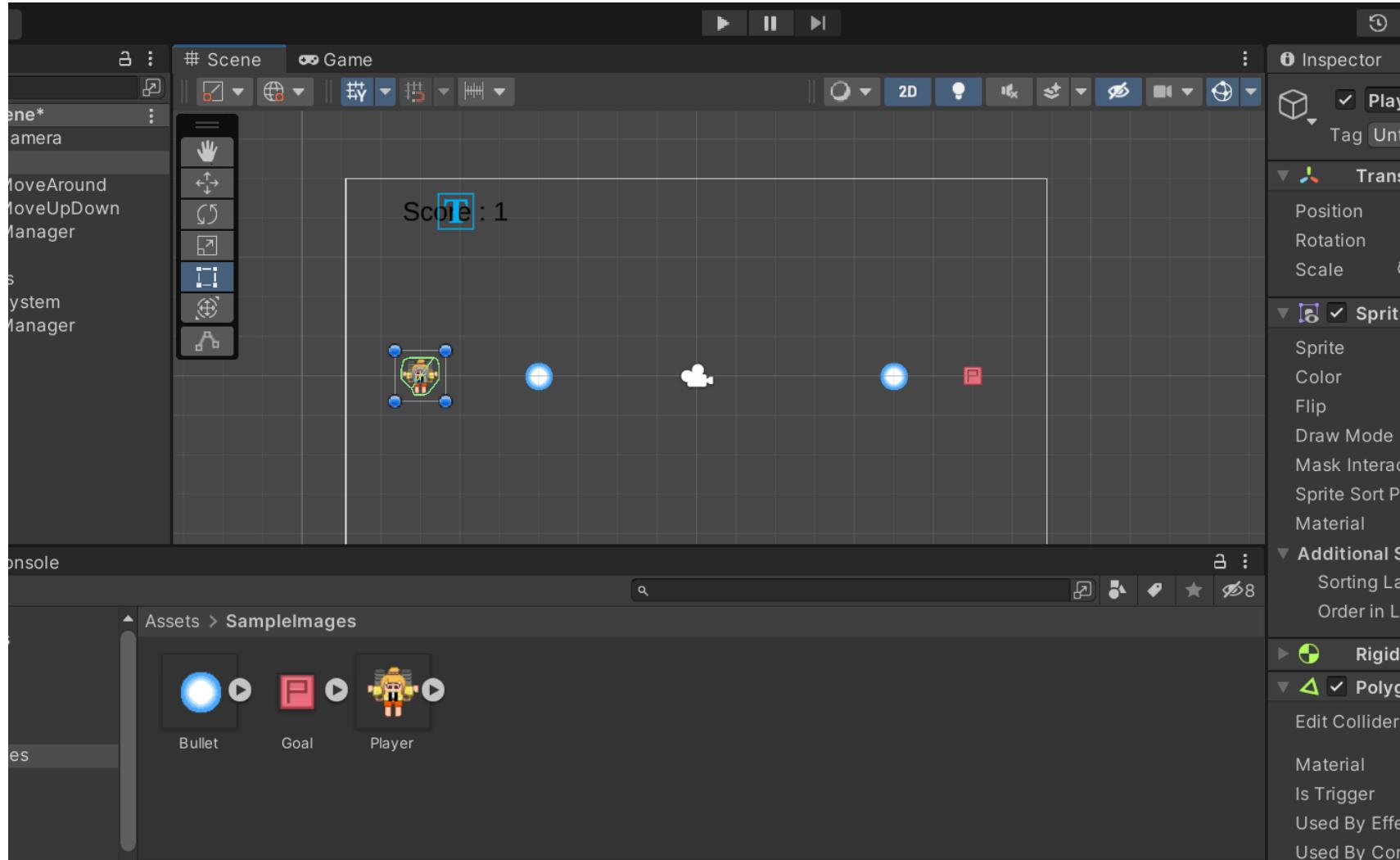
まずは画像を取り込んでゲームらしくします  
ダウンロードしたdatasetの  
中の”sampleImages”をUnityに取り込みます



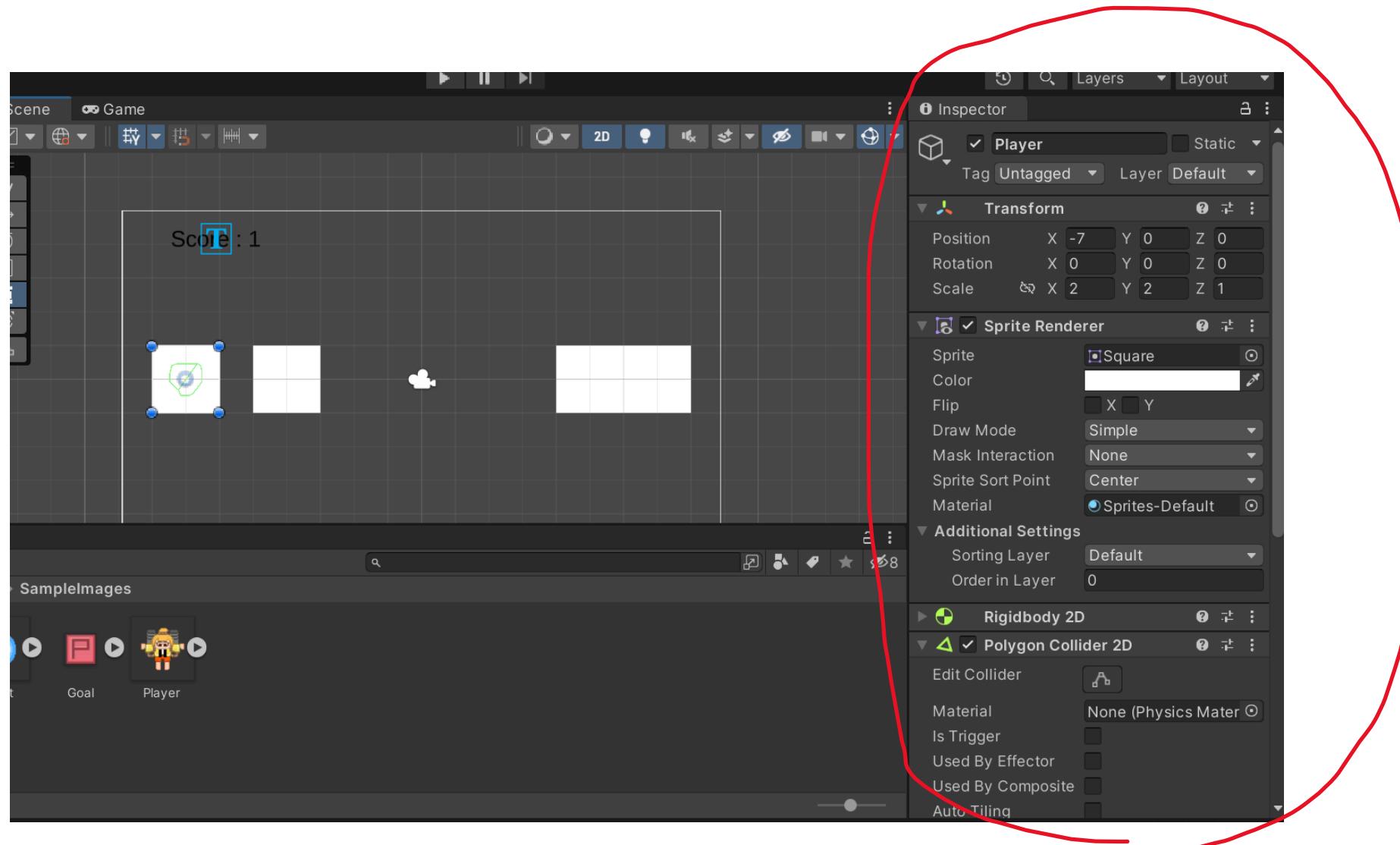
UnityのAssetsフォルダを開いてそこにドラッグします



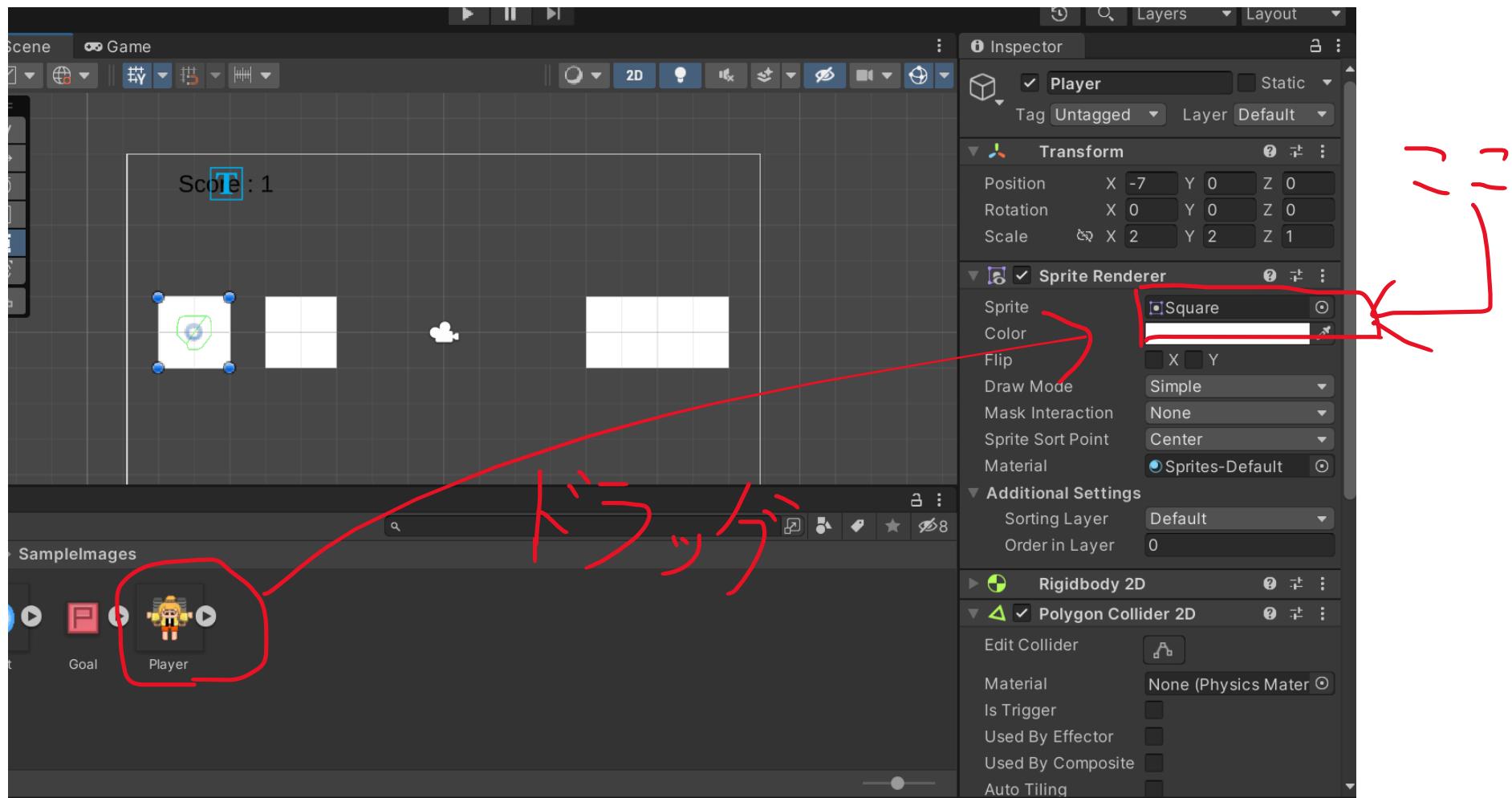
そうしたら、画像を上の画面の白い四角に反映させていきます  
完成形は↓のようになります



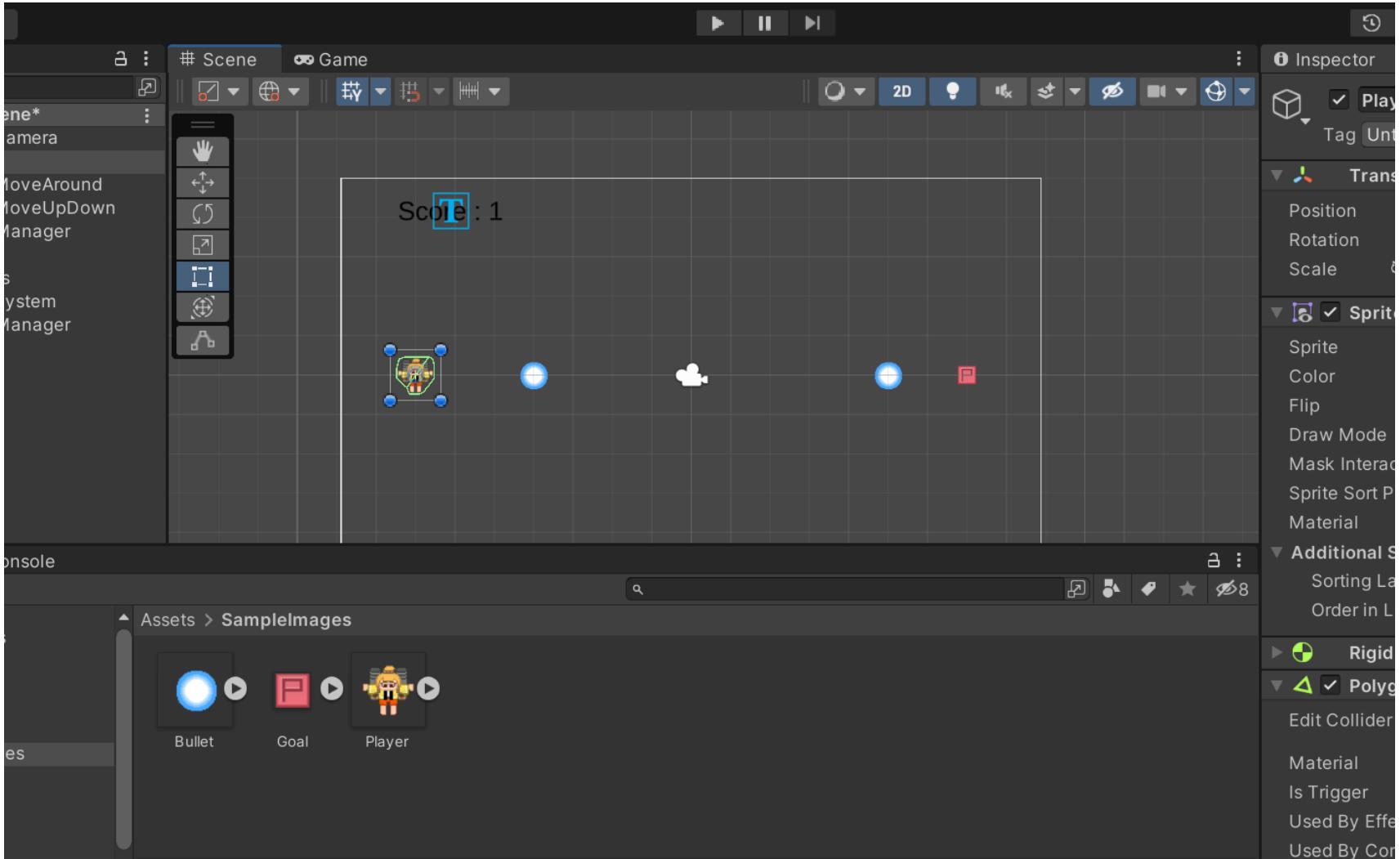
白い四角をクリックするとそれぞれの物体の情報が  
→のウインドウに出ると思います



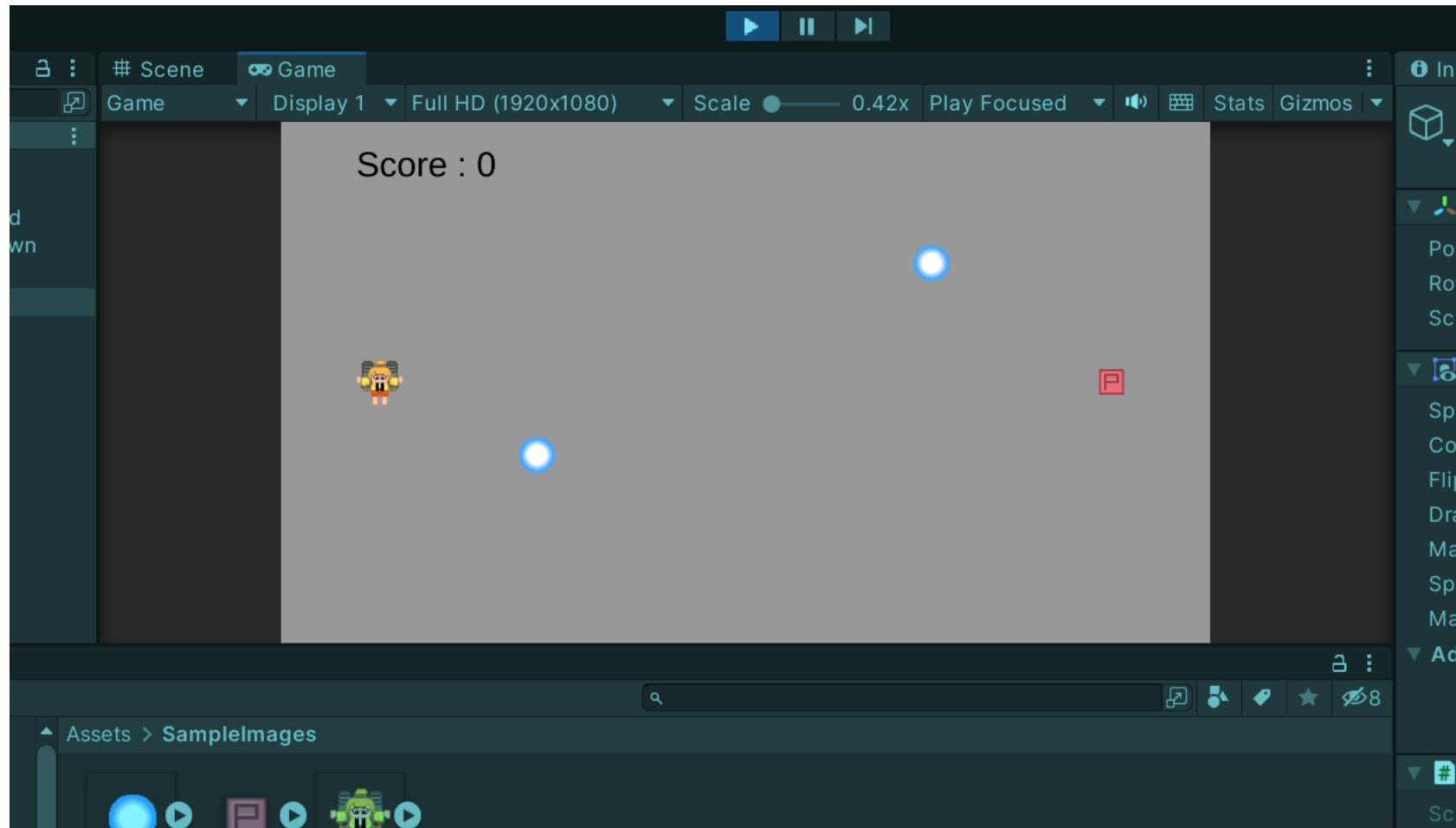
右側のウインドウの赤い四角で囲った部分が  
物体の画像の情報になるので  
反映させたい画像をそこにドラッグします



あとはそれを全部の四角についてやります



再生ボタンを押してみましょう  
ちょっとそれっぽくなりました



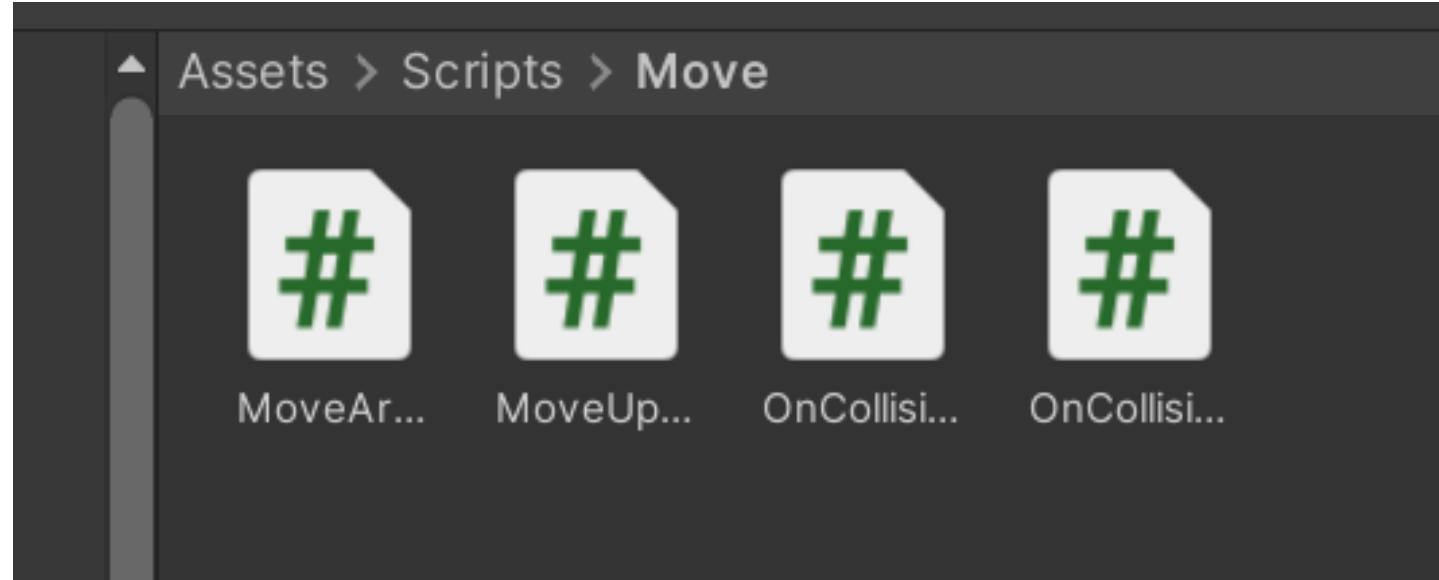
ただ肝心の自機がまだ動きません！！！

なのでスクリプトを書いて動かせるようにします

# スクリプトとは？

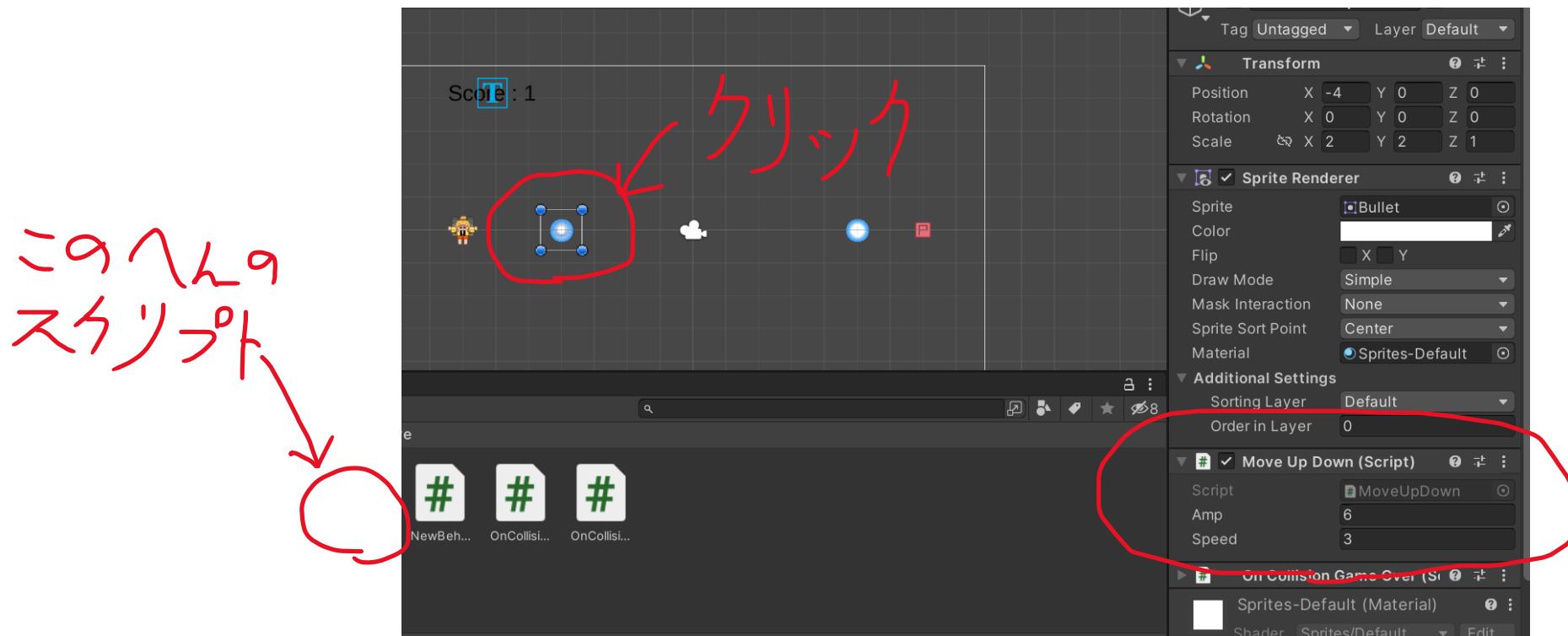
スクリプトとはプログラムが書かれたソースファイルを指します

UnityではC#というプログラミング言語を使ってコーディングをします

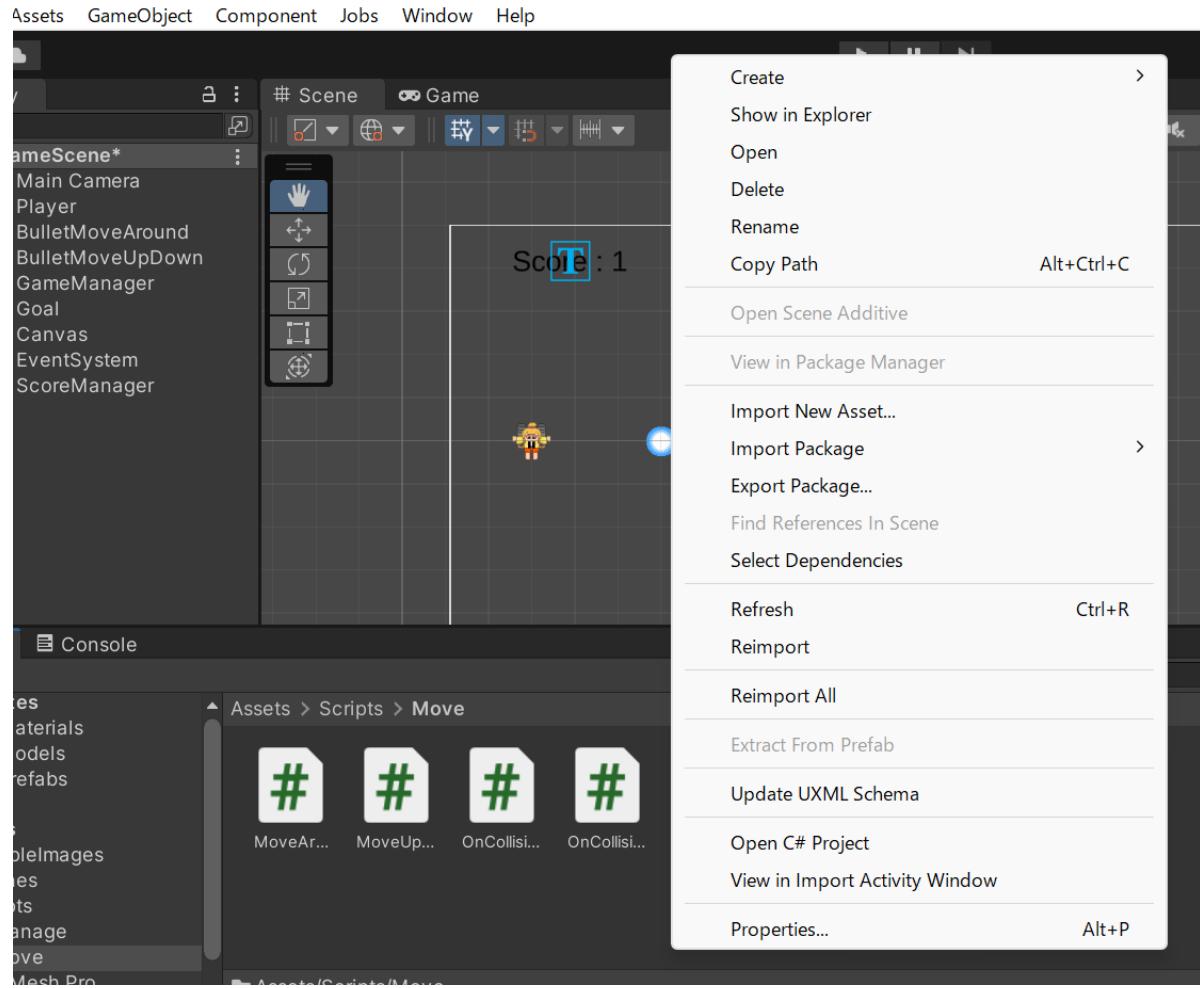


# スクリプトとは？

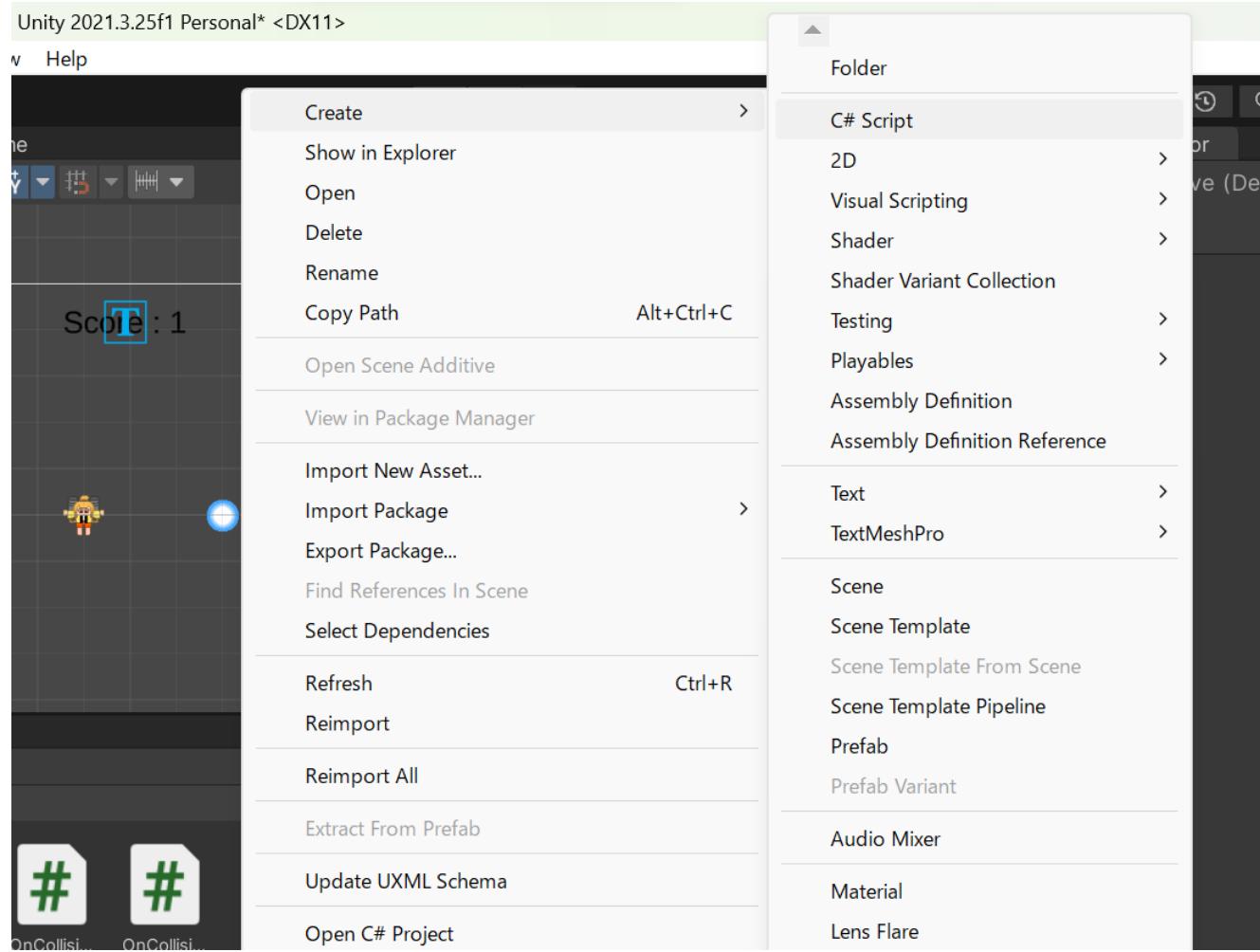
このスクリプト情報を物体に貼ることで動作させます  
例えば左側の弾をクリックしてみると「Move Up Down」というスクリプトが貼ってあるのが分かります  
これによって弾が上下に動いていたんですね



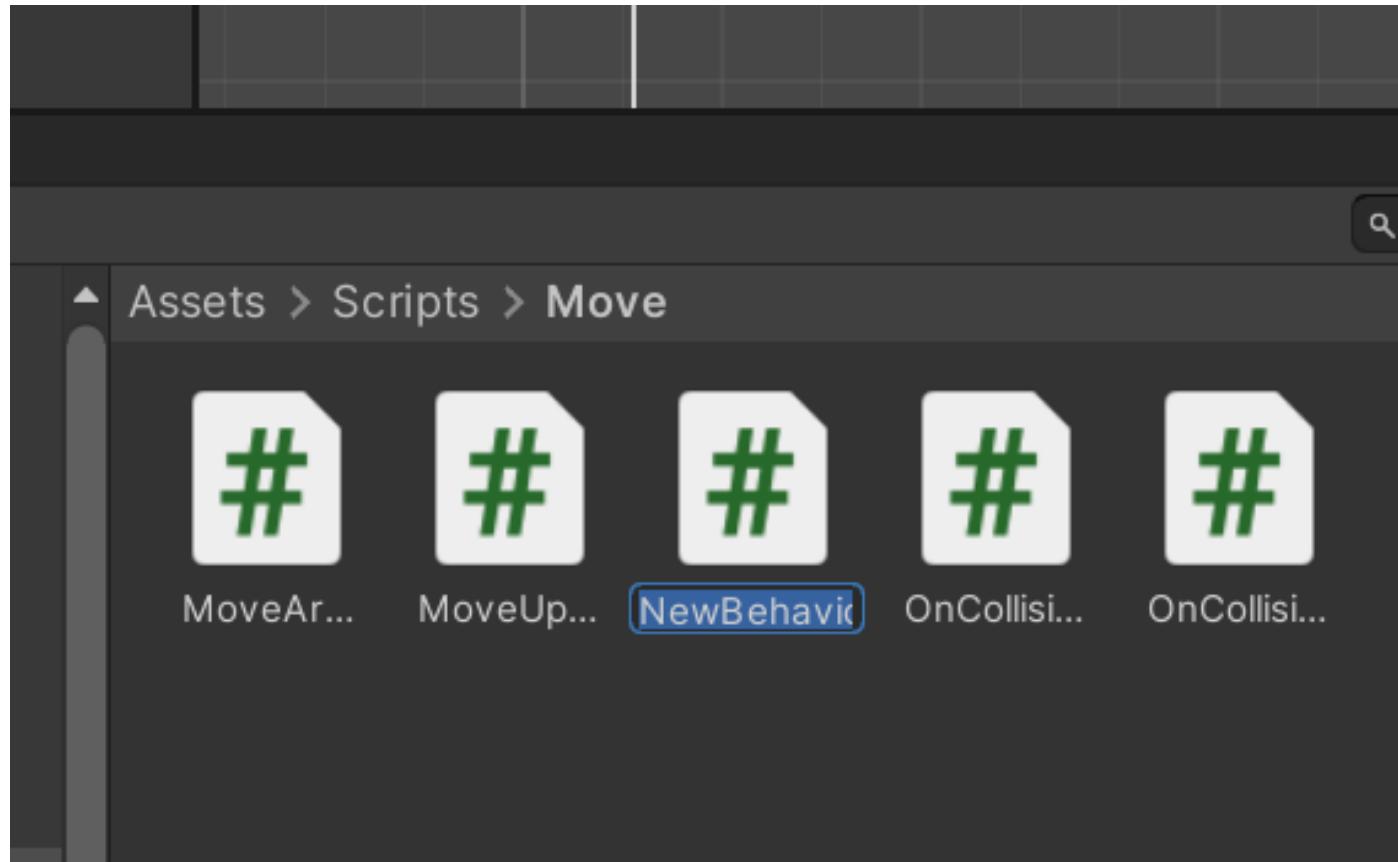
ではスクリプトを新規作成してみましょう  
Assets→Scripts→Moveのフォルダ内で右クリックします



そしたらCreate→C# Scriptを選択します



すると新しいスクリプトが作られます  
名前を入力するフォームが出ますが  
一旦そのままEnterを押します(後で変更します)



作ったスクリプトをダブルクリックすると  
何らかのテキストエディタが起動すると思います  
以下はVisual Studioのエディタでの画面になります

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

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

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

毎回このコードが書かれた状態から始まります

急にプログラミングっぽい画面が出てきて??????  
となっていると思いますが

今はこここの部分に注目してください(5行目です)

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

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

この行のClassと:(コロン)の間の

「NewBehaviourScript」

と書かれた部分が

スクリプトの名前になります

Unityの画面でも名前を

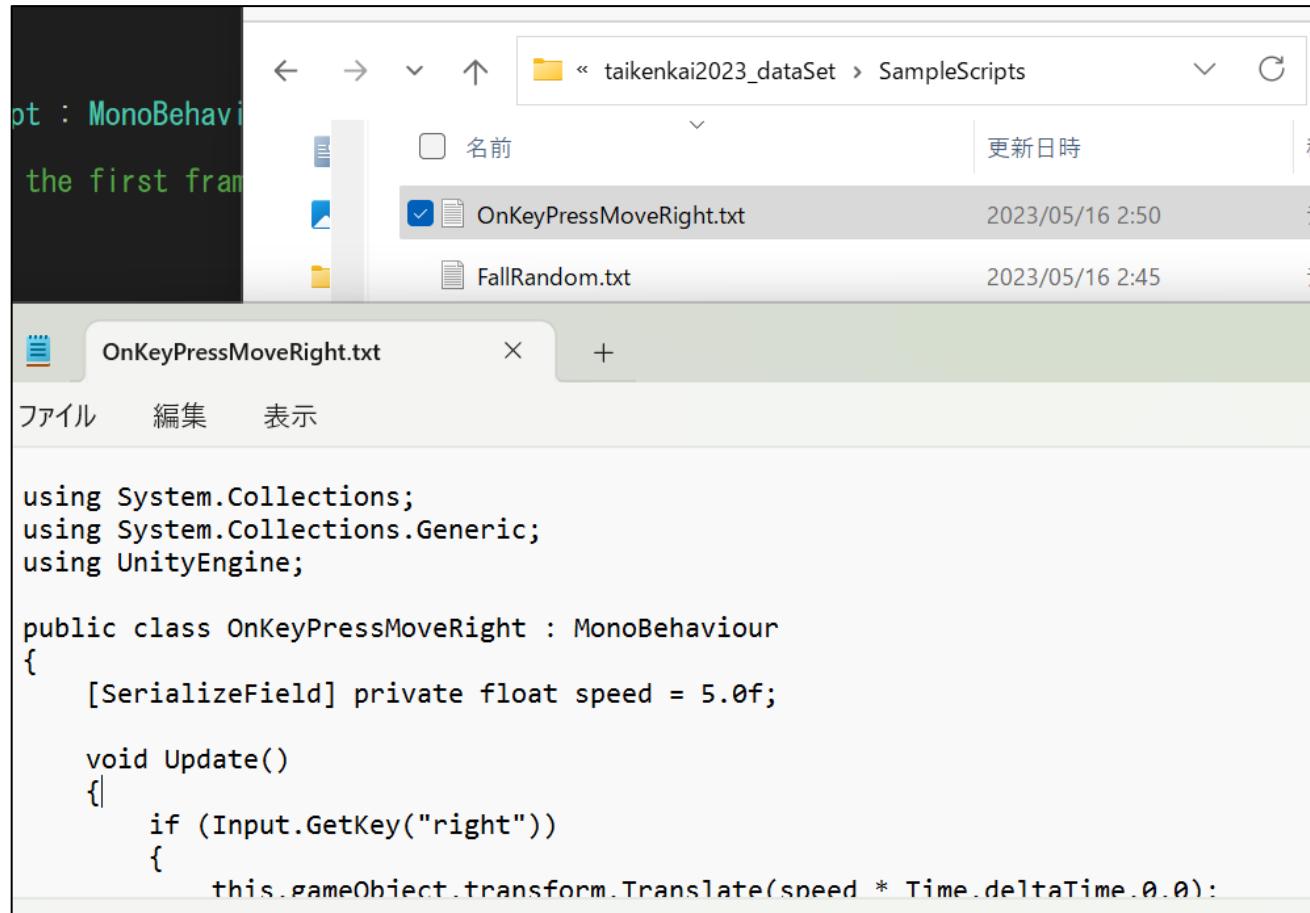
確認すると一致していると思います



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

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

ここでtaikenkai2023\_datasetの  
「OnKeyPressMoveRight.txt」を開いてください  
これが今回使うコードになります



The screenshot shows the Unity Text Editor window. The title bar says "OnKeyPressMoveRight.txt". The menu bar includes "ファイル" (File), "編集" (Edit), and "表示" (View). The main editor area contains the following C# code:

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

public class OnKeyPressMoveRight : MonoBehaviour
{
    [SerializeField] private float speed = 5.0f;

    void Update()
    {
        if (Input.GetKey("right"))
        {
            this.gameObject.transform.Translate(speed * Time.deltaTime, 0, 0);
        }
    }
}
```

In the background, the Unity Project window is visible, showing a folder structure under "taikenkai2023\_dataset > SampleScripts". There are two files listed: "OnKeyPressMoveRight.txt" (selected) and "FallRandom.txt". Both files were last modified on 2023/05/16 at 2:50 and 2023/05/16 at 2:45 respectively.

これをさっさと新規作成したスクリプトの中身を全部消して  
貼り付けてください

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class OnKeyPressMoveRight : MonoBehaviour
6  {
7      [SerializeField] private float speed = 5.0f;
8
9      void Update()
10     {
11         if (Input.GetKey("right"))
12         {
13             this.gameObject.transform.Translate(speed * Time.deltaTime, 0, 0);
14         }
15     }
16 }
17
```

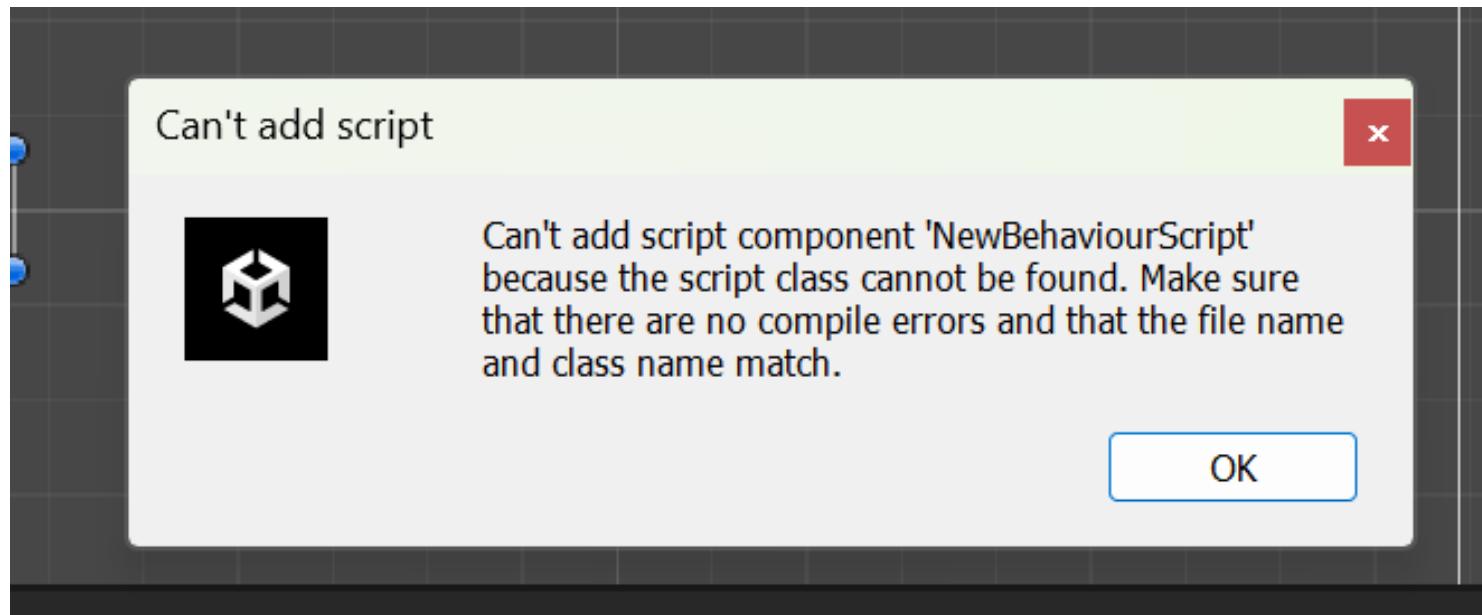
これでCtrl+sで上書き保存すれば  
「→キーを押すと物体が右に動く」スクリプトの完成です

…コピペだけでは味気ないですよね

今回は文法やらなんやらについて細かくは解説しませんが  
どこで何をしているかについて後のページでふんわり解説します

というわけで早速Playerに貼り付けて動かすぞ！！！！！

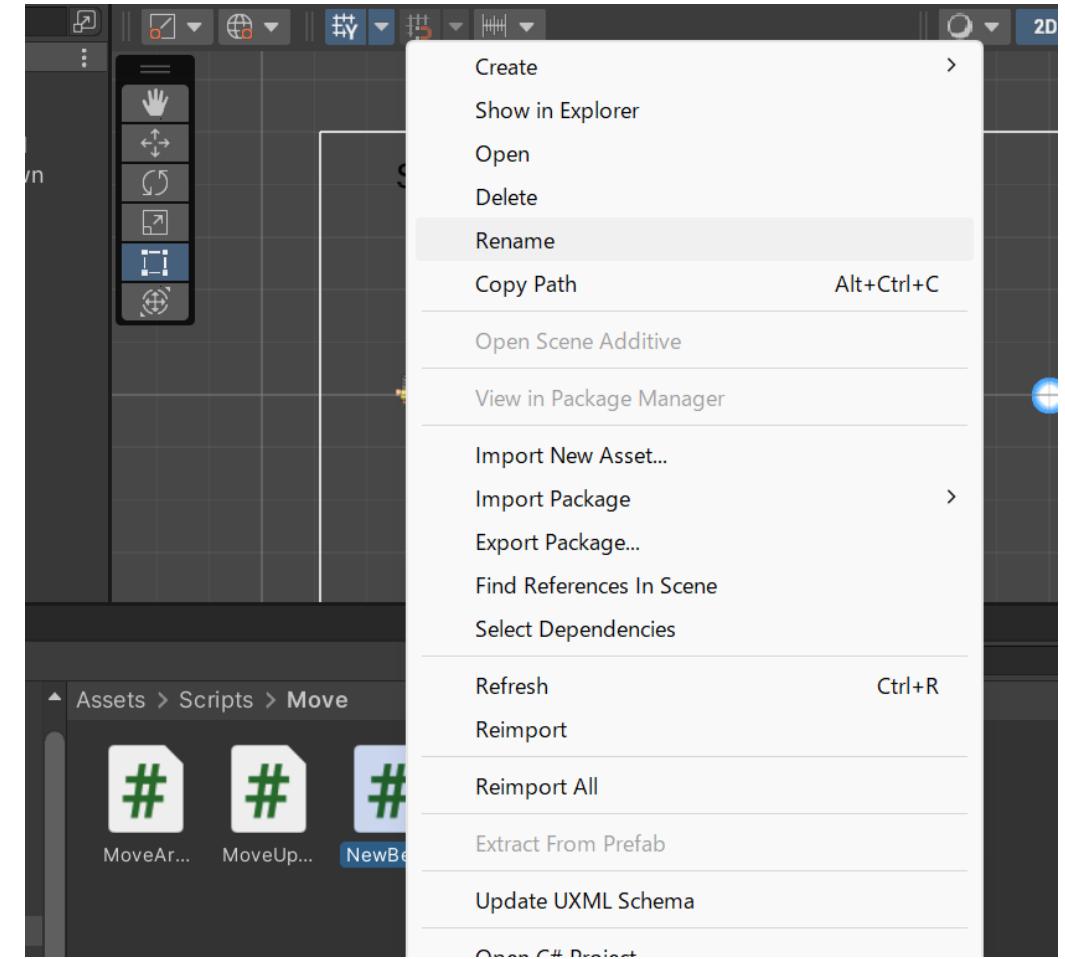
…と言いたいですがこのままでは動きません  
さっきのスクリプトの**名前**の部分が  
Unity側とエディタ側で異なるからです



↑ Unity君「なんかスクリプト見つからないんだけど名前とか間違えてね？」

# なので名前を揃えましょう

Unity側の名前を変えるときは  
スクリプトを右クリックして  
「Rename」を選択します



エディタ側ではここを変更します

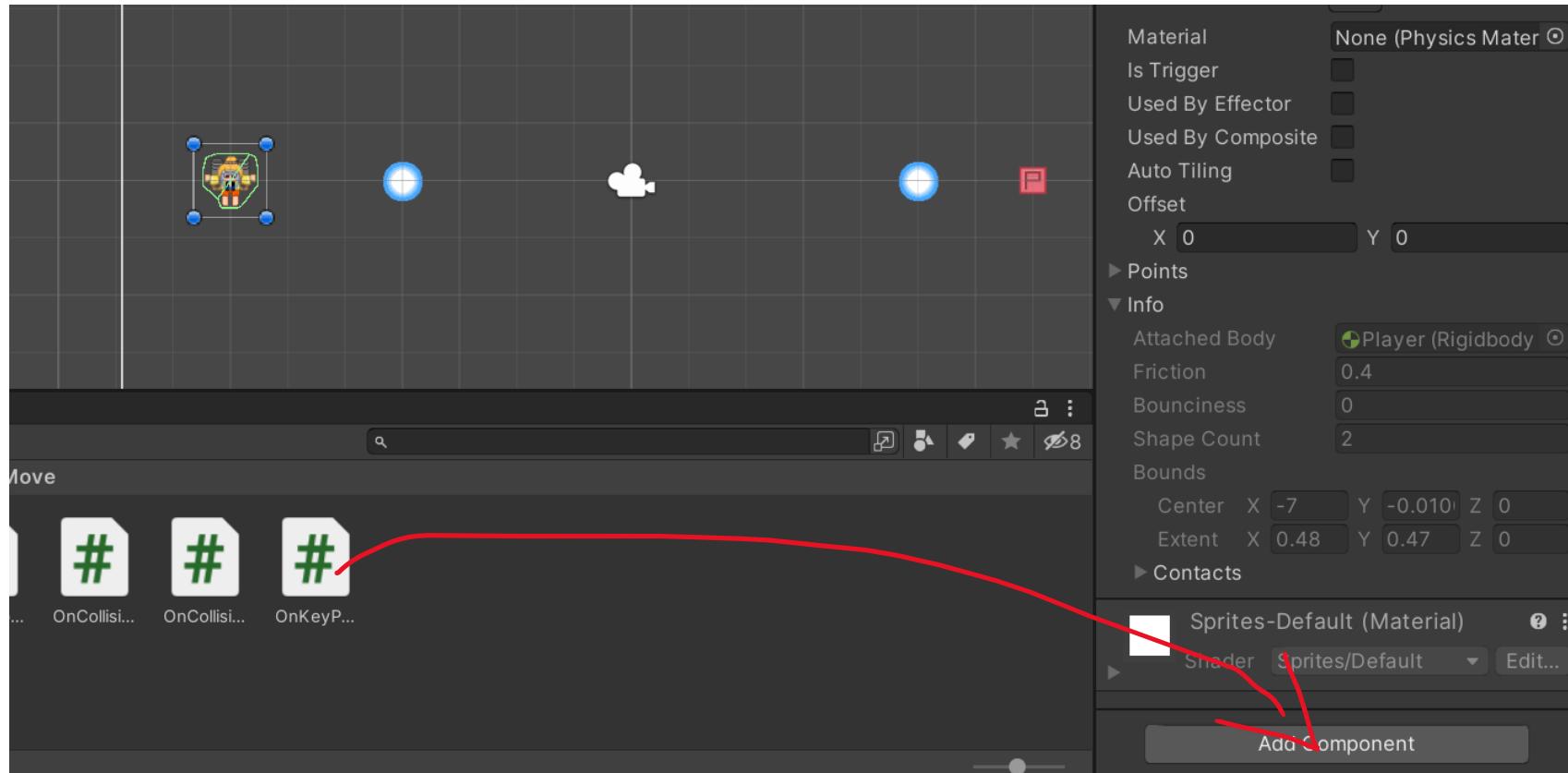
必ず「class」と「:」の間に  
半角スペースが入るようにしてください

あとこれはどちら側でもそうなのですが  
全角文字が入っているか最初の文字が数字だとダメです

```
using UnityEngine;
  
④ Unity スクリプト 10 個の参照
public class OnKeyPressMoveRight : MonoBehaviour
{
    [SerializeField] private float speed = 5.0f;
```

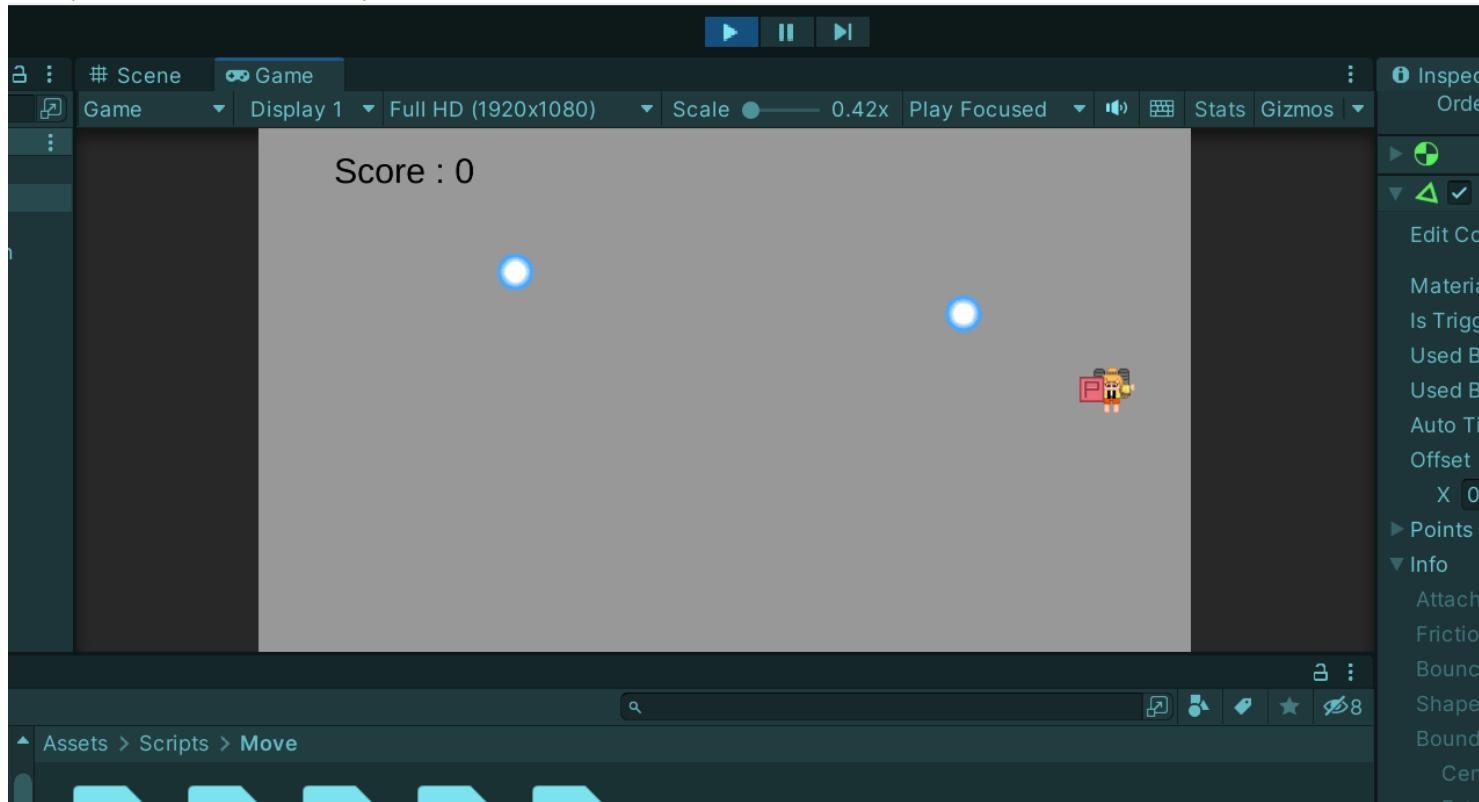
それではスクリプトを貼り付けます

Playerをクリックした後、→側のウインドウを下にスクロール  
そうすると出てくる「Add Component」の部分にスクロールを  
ドラッグします

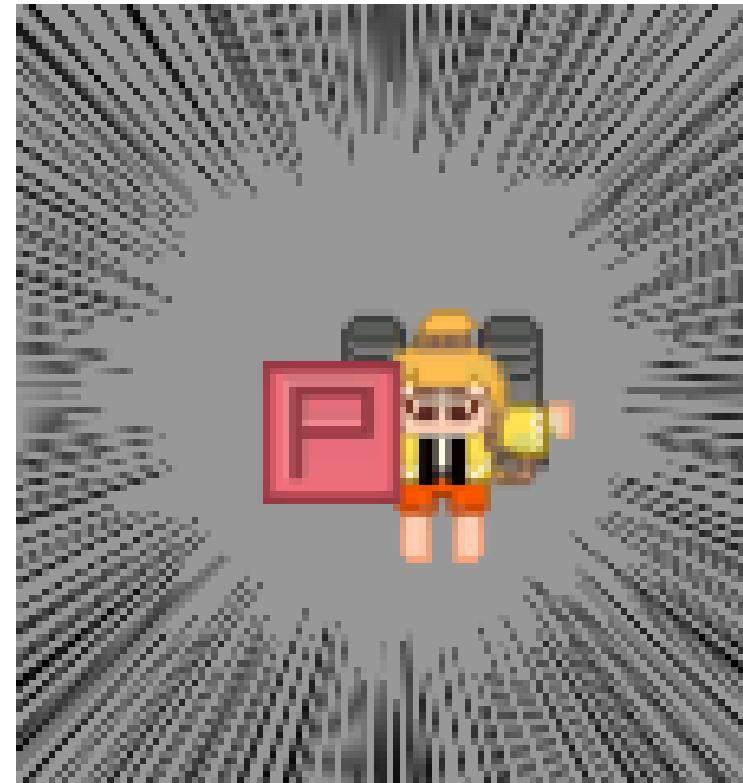


これでPlayerが動くはずです！！

これでゲームが完成します…ん？



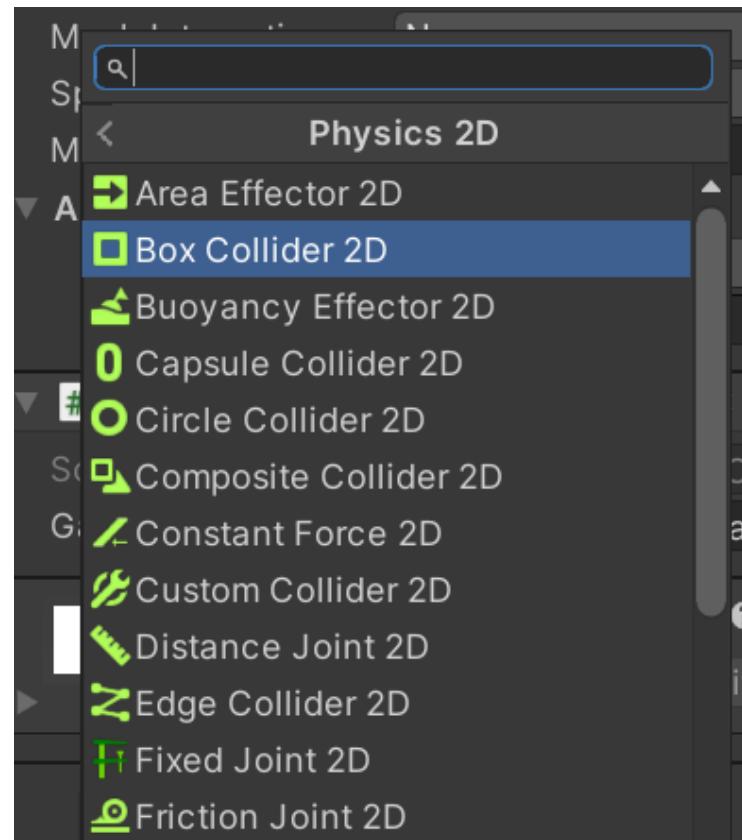
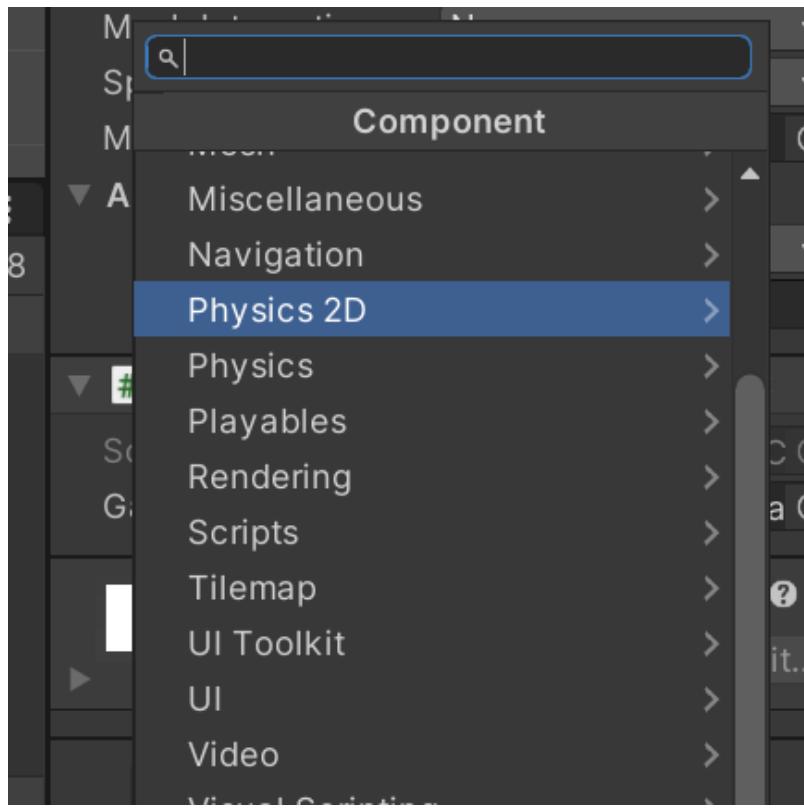
ゴールをすり抜けてる～～！！？？？



はい、大丈夫です想定内です。

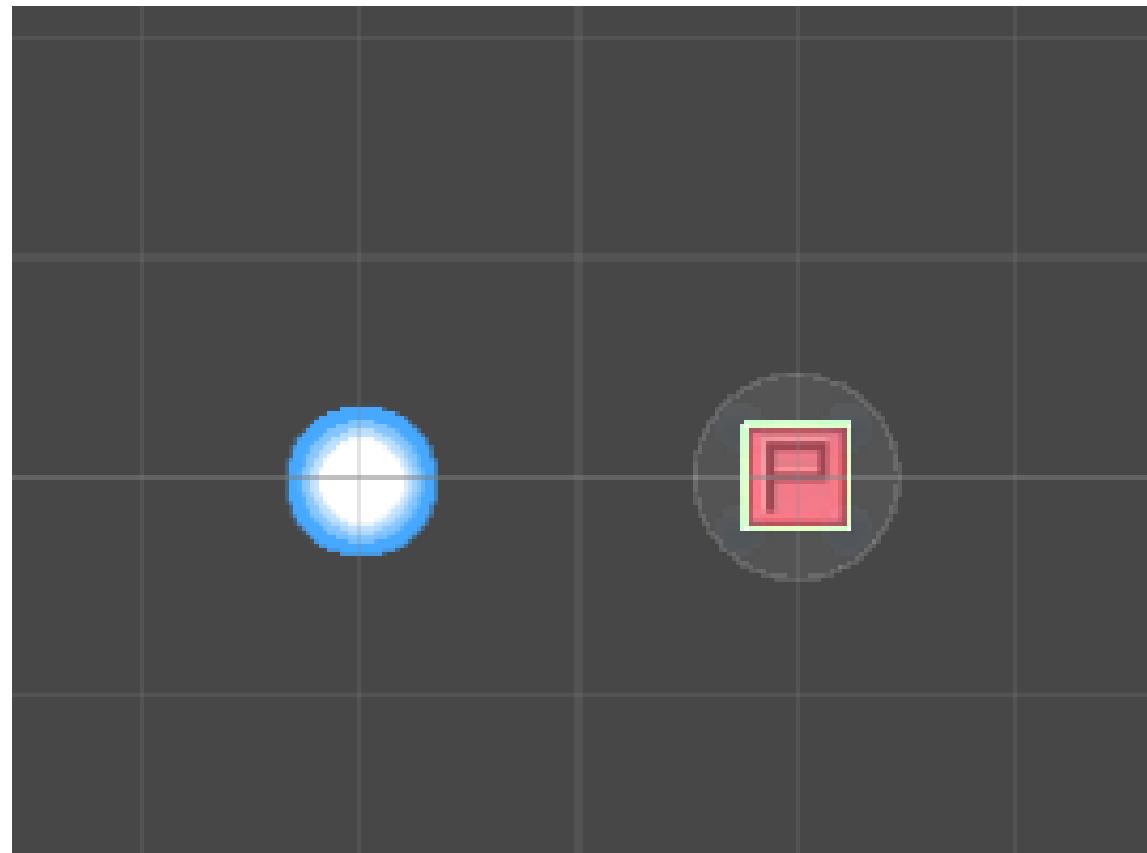
実は「プレイヤーに当たった時の処理」のスクリプトは作ってあるのですが「当たり判定」が設定されていません

ゴールをクリックして右のウィンドウ下側から  
「Add Component」→「Physics 2D」→「Box Collider 2D」  
と選択していきます

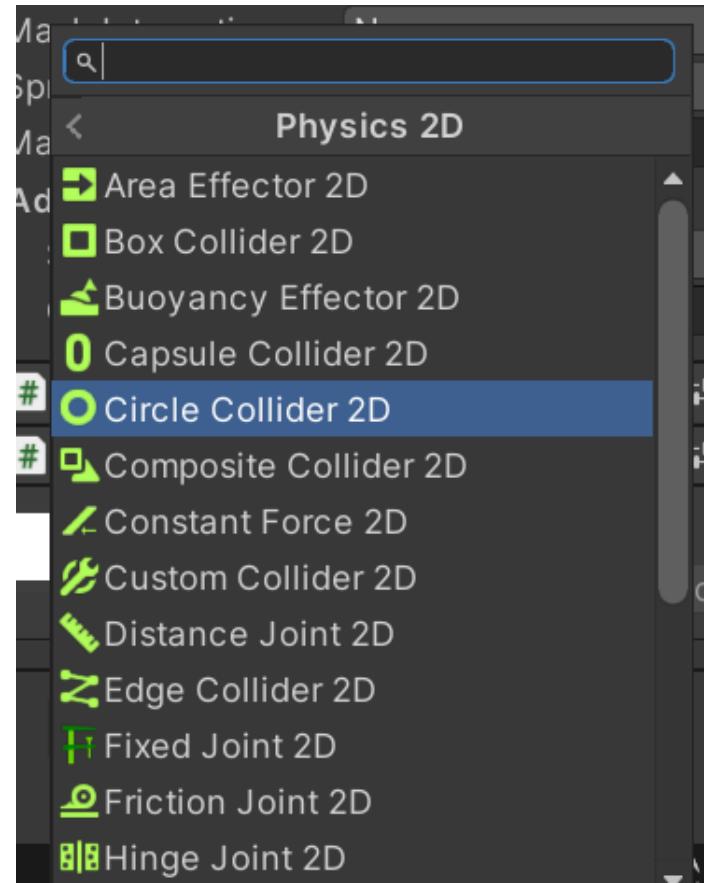
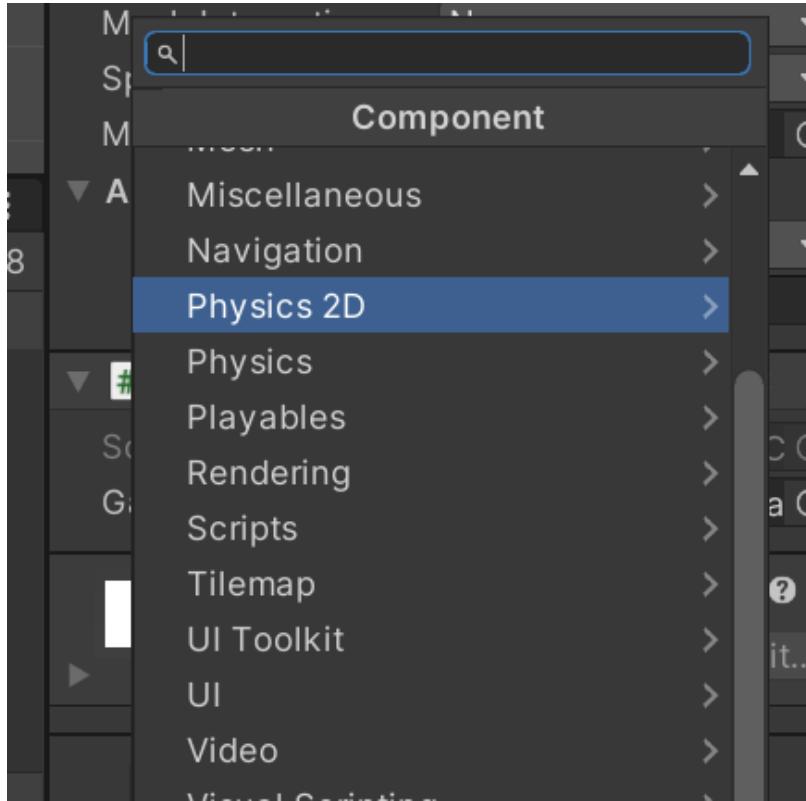


これでゴールに四角い当たり判定がつきました  
(緑色で囲まれた部分がそうです)

実は弾にも当たり判定がないのでつけていきます

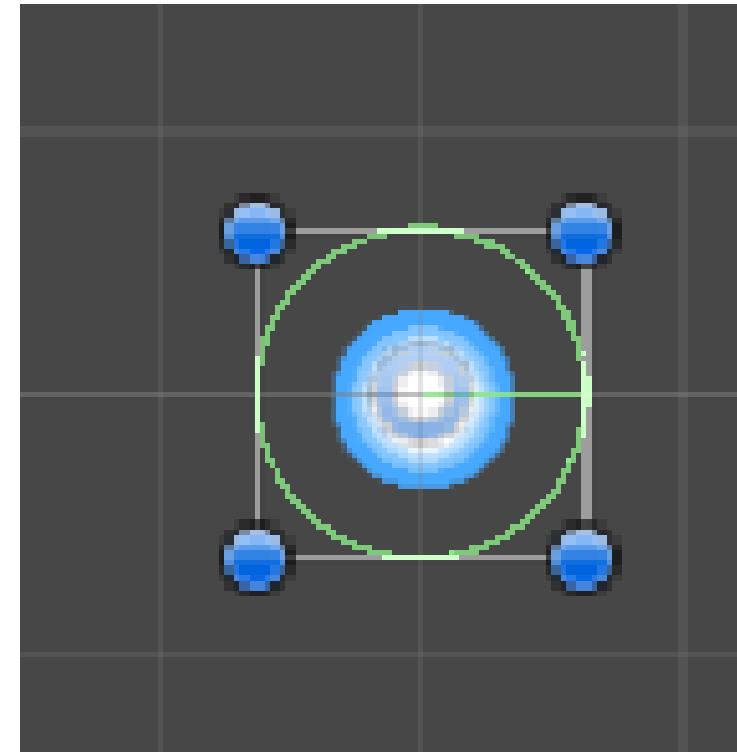


同様に弾をクリックして右のウインドウ下側から  
「Add Component」→「Physics 2D」→「Circle Collider 2D」  
と選択していきます

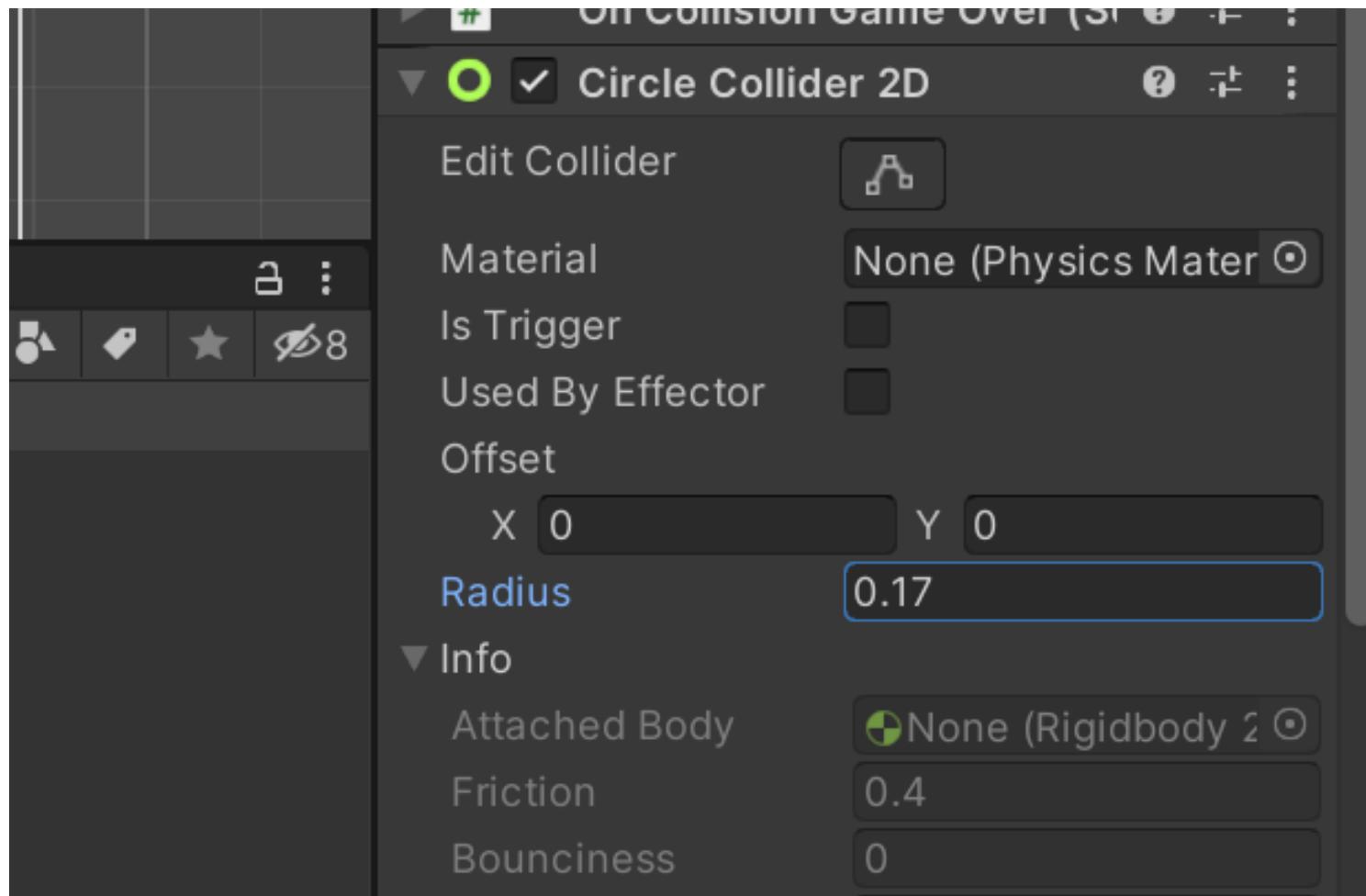


これで円状の当たり判定がつきました

ただこのままではかなり実際の円からずれてしまっています

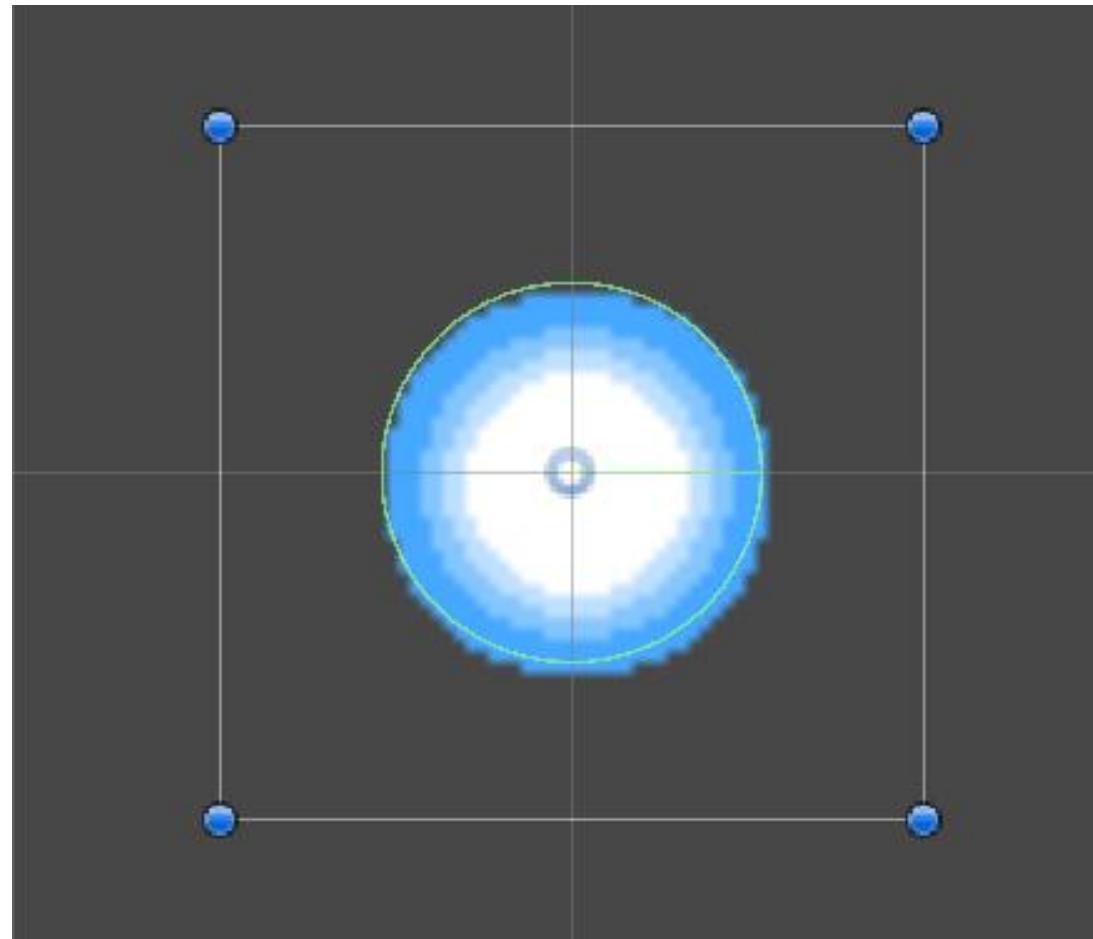


そこで、弾についた「Circle Collider 2D」の  
三角「▷」をクリックして  
出てきた「Radius」の欄に「0.17」と入力します

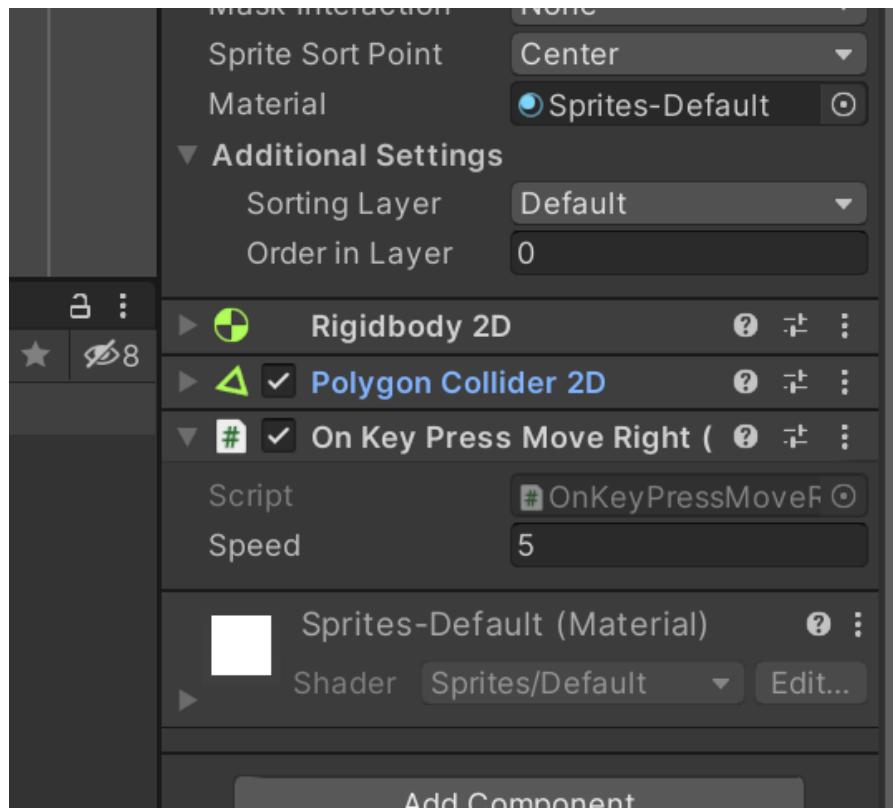


これで当たり判定がだいぶ合いました！

もう一個の弾の方にも当たり判定をつけましょう

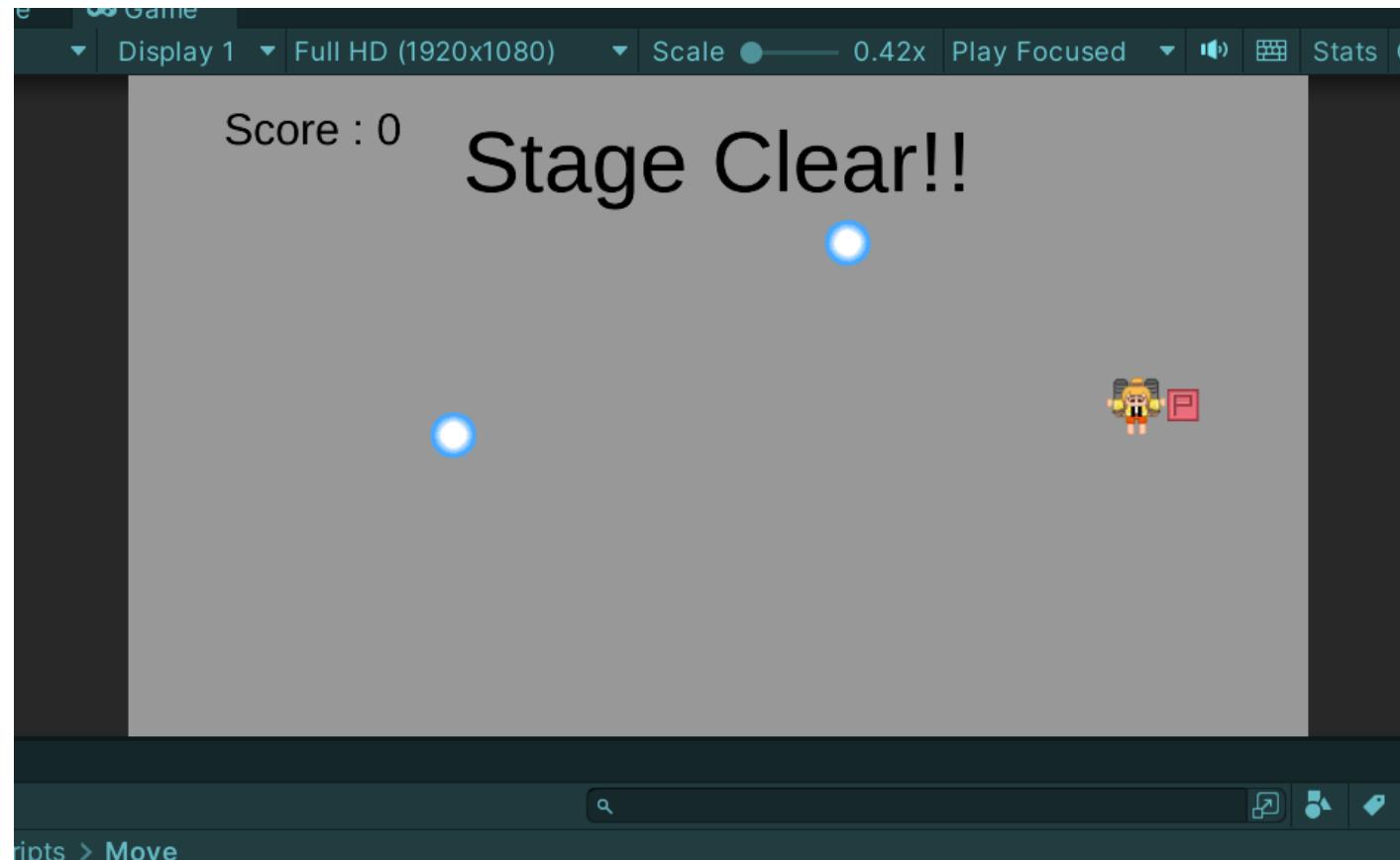


ちなみにPlayerには既に当たり判定がついています  
また、「Rigidbody 2D」というものがついていますが  
これがついてないと当たり判定が機能しません  
今回、説明は省きます



改めてこれでゲームが完成しました！

ちなみにGameOverかStageClearの時に  
Spaceを押すとリトライできます



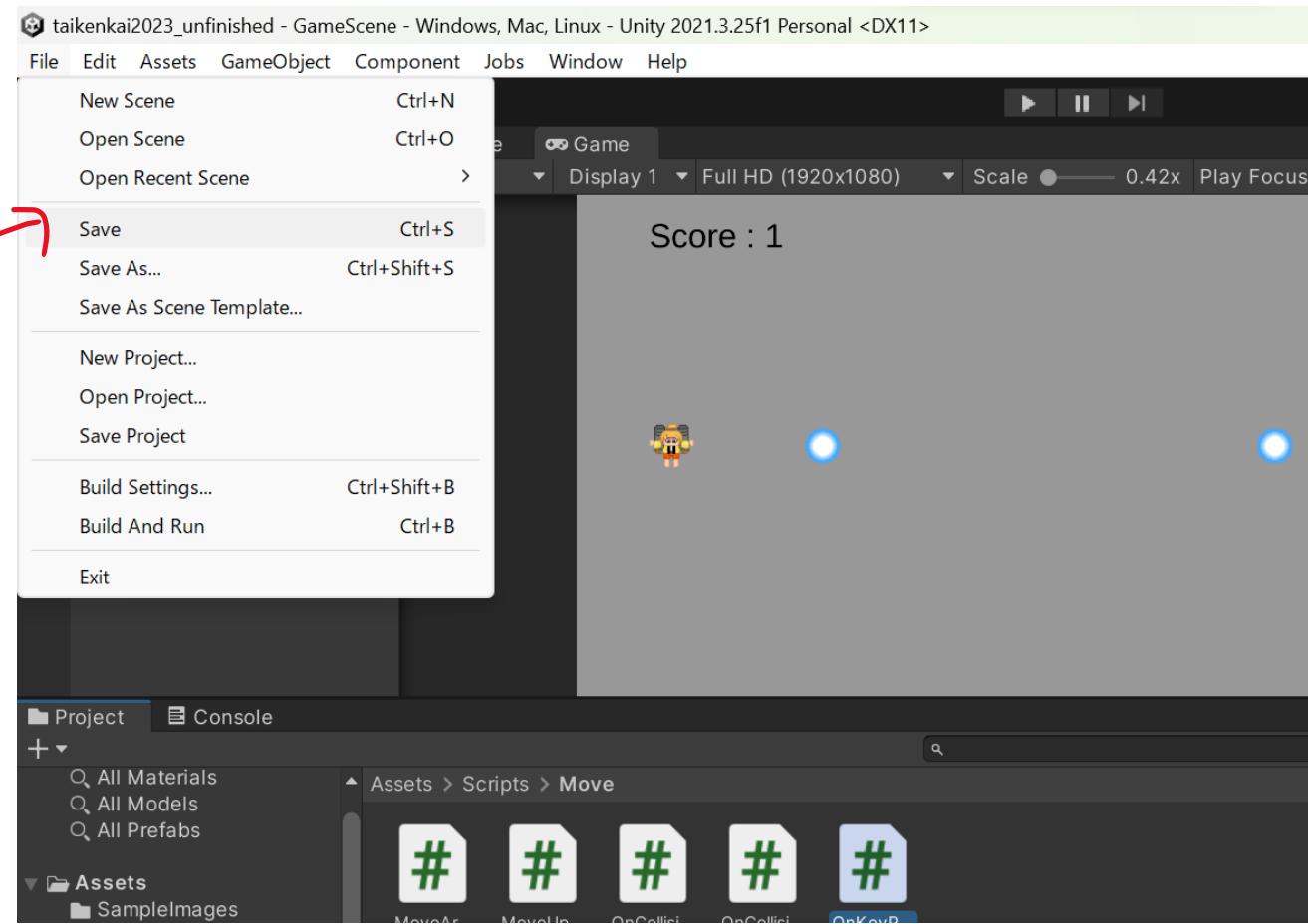
高得点を目指してみてね～～



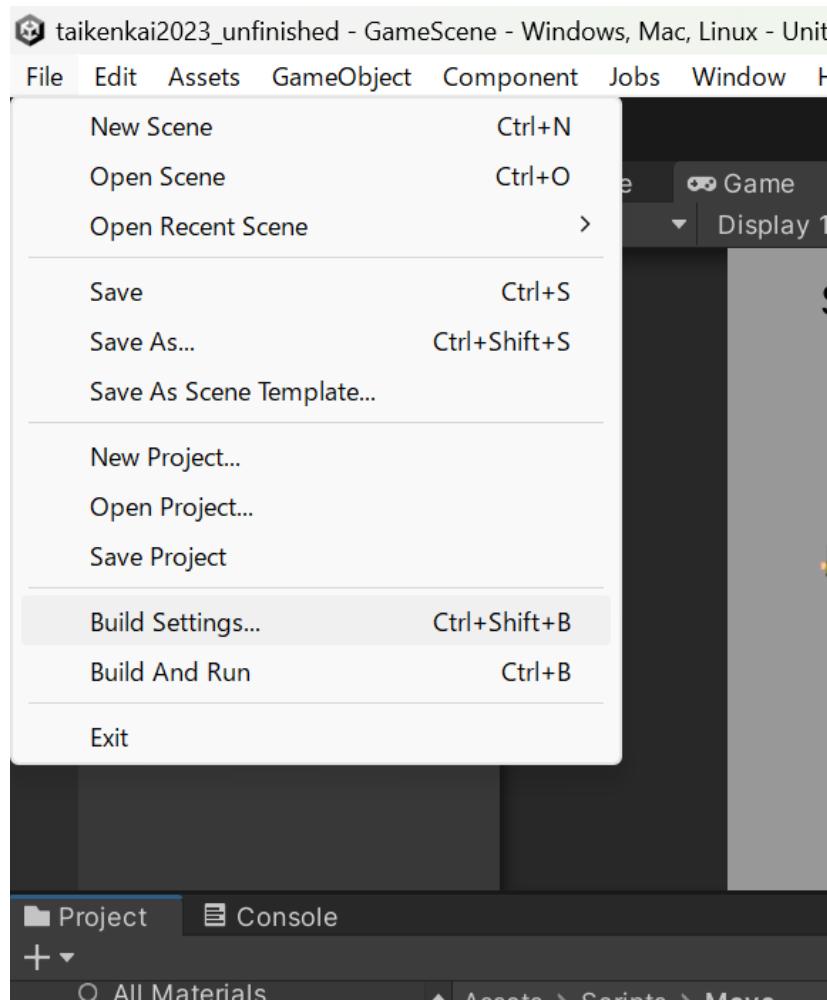
最後に、ゲームアプリとしてゲームを開けるように  
作ったゲームをビルドします

まずCtrl+Sで作業を保存しましょう

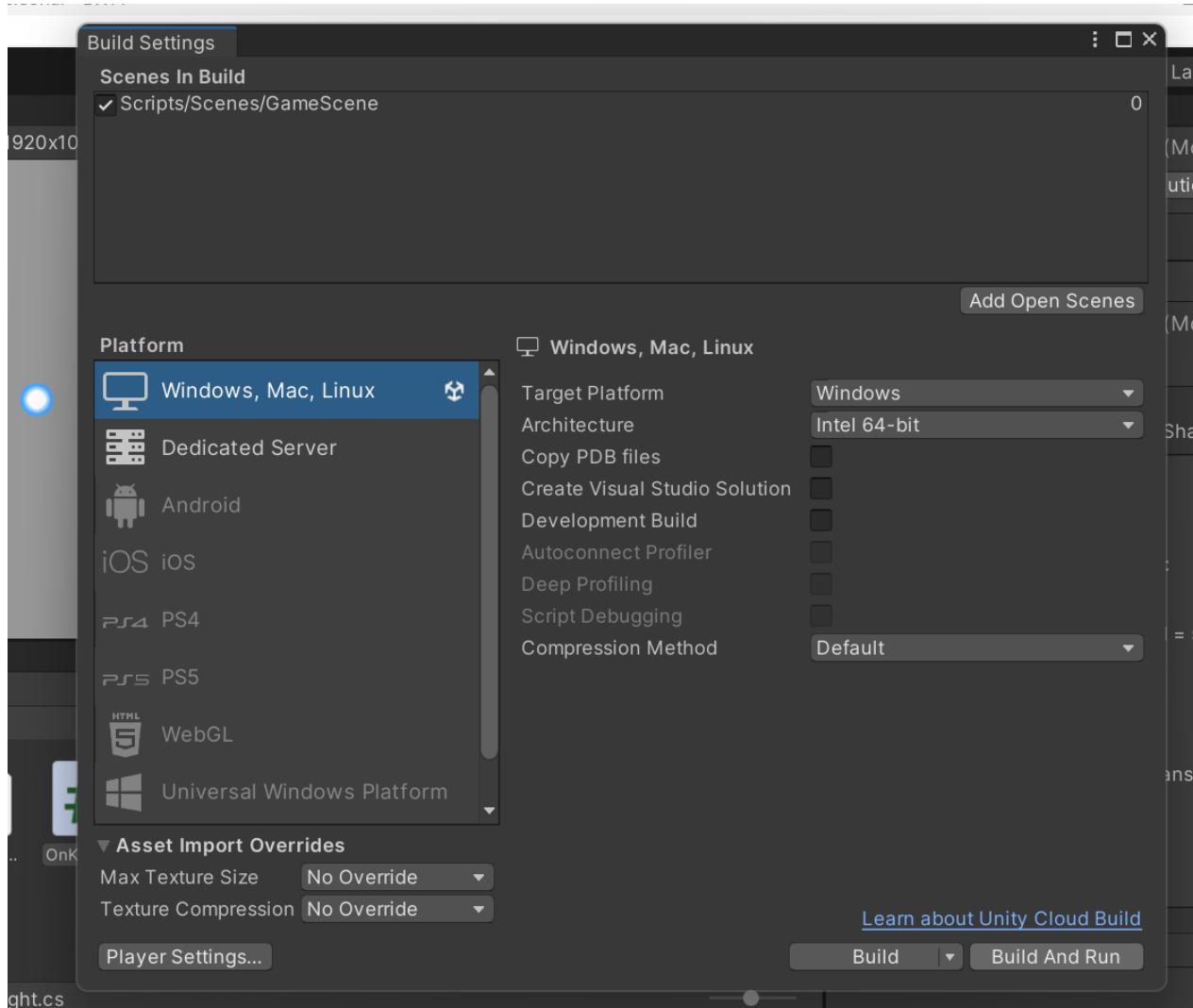
それからこれ



「File」 → 「Build Settings」 をクリックします

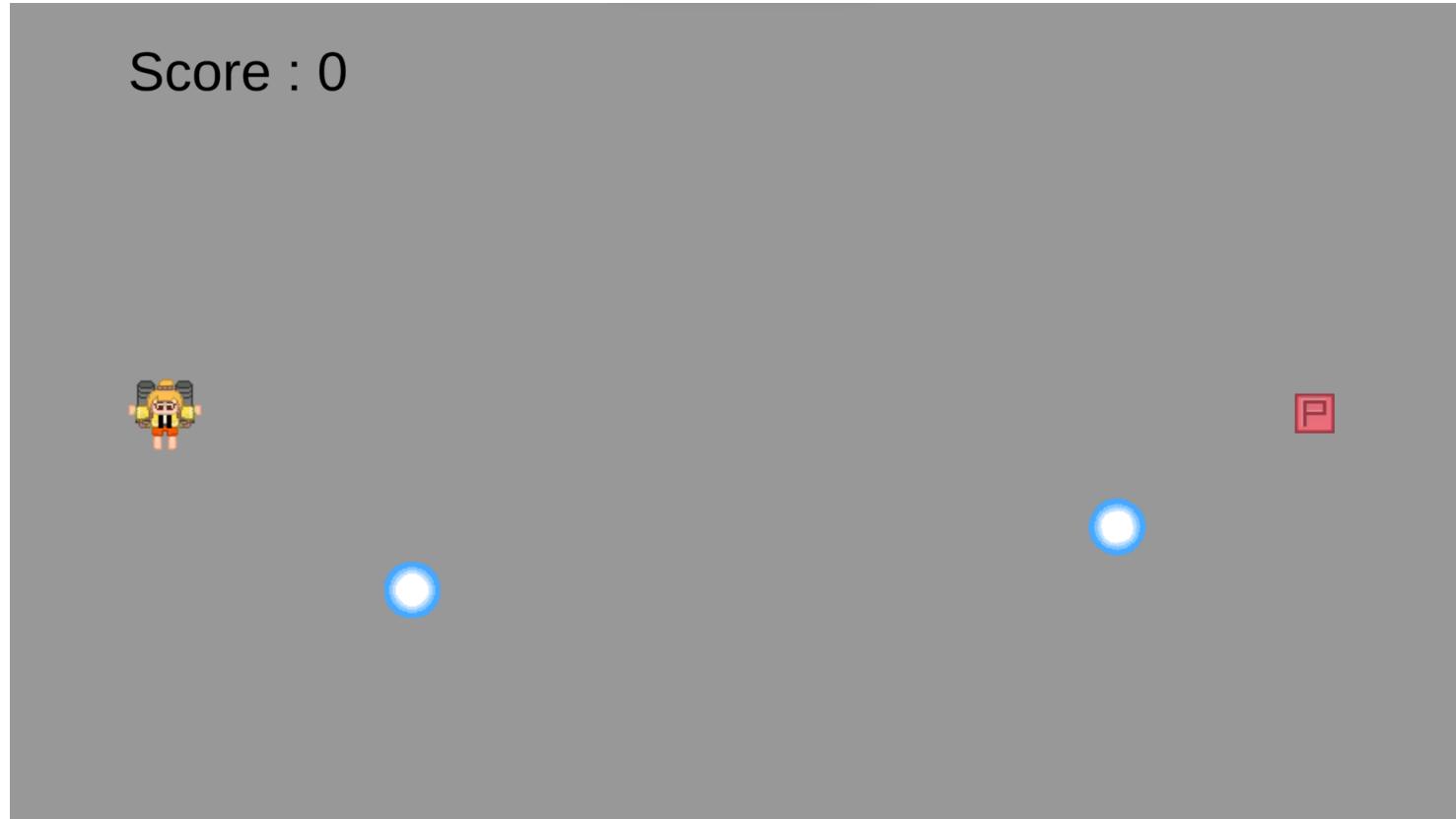


右下の「Build And Run」をクリックします



あとは適当な場所にフォルダを作成して選択します

しばらくするとゲーム画面が出てきてプレイできるはずです！

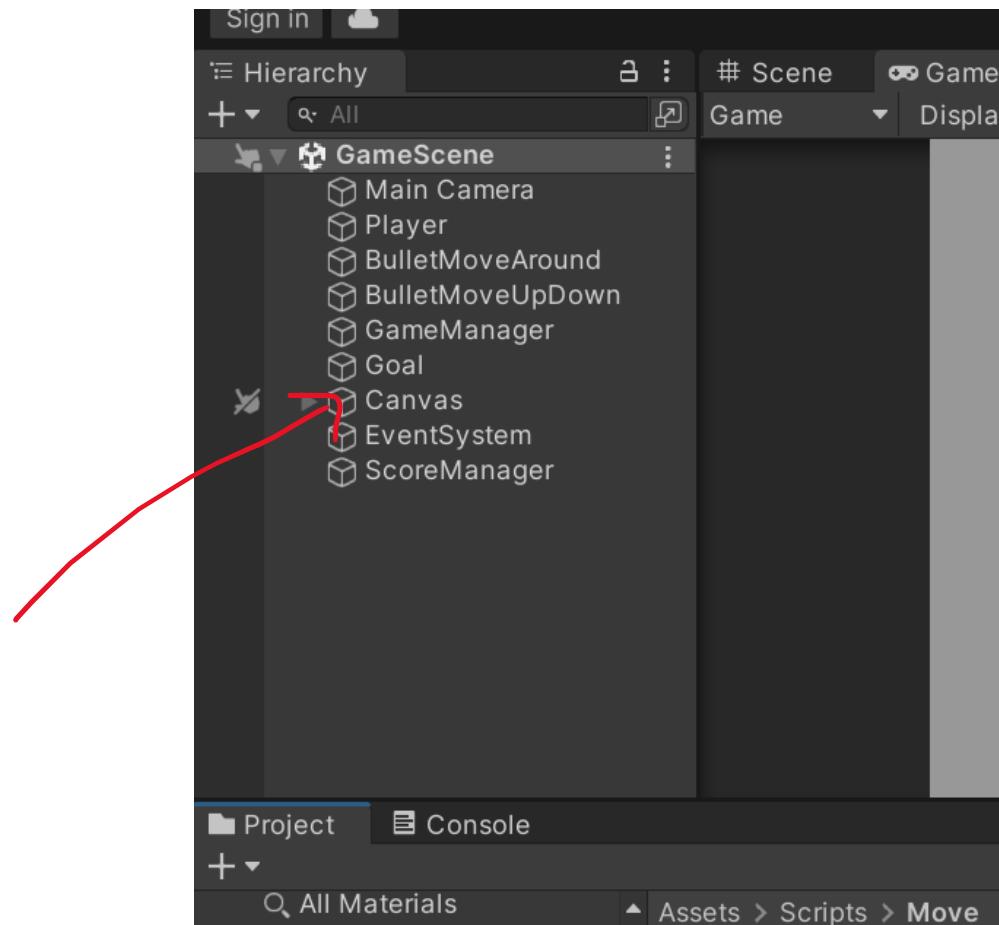


ゲームを実行するときはこの.exeとついたファイルを  
ダブルクリックすれば実行できます

taikenkai2023_Data	2023/05/21 20:55	ノンパルノナルマー
 taikenkai2023.exe	2023/05/21 20:33	アプリケーション 639 KB

もし、実際のゲーム画面で画面サイズが崩れる場合はこちらをお試しください

<https://www.create-forever.games/unity-resolution-game/>



これでゲーム体験会の内容は以上です！！！

最後まで見ててくれて  
ありがとうございました！！！

# 余談

右に動くプレイヤーのコードについて  
軽く解説していきます

興味ある方はどうぞ

## 1~3行目

ここは実際にプログラムとして動作する部分ではないです  
おまじないだと思ってもらって大丈夫です

```
1 |  using System.Collections;
2 |  using System.Collections.Generic;
3 |  using UnityEngine;
```

## 5~16行目

この6行目と16行目のカッコに囲まれた部分が  
実際に動作する部分です

内容を見ていきます

```
5  ◎Unity スクリプト 10 個の参照
6  public class OnKeyPressMoveRight : MonoBehaviour
7  {
8
9      ◎Unity メッセージ 10 個の参照
10     [SerializeField] private float speed = 5.0f;
11
12     void Update()
13     {
14         if (Input.GetKey("right"))
15         {
16             this.gameObject.transform.Translate(speed * Time.deltaTime, 0, 0);
17         }
18     }
19 }
```

7行目

何かごちゃごちゃしてますが注目すべきは  
「speed = 5.0f」  
の部分です

これで「speed」という変数は「5.0」という値を持ちました  
(fは文法上書いていますが無視してください)



```
[SerializeField] private float speed = 5.0f;
```

A screenshot of a code editor showing a single line of C# code. The code is annotated with color-coded highlights: the attribute [SerializeField] is in green, the keyword private is in blue, the type float is in blue, the variable name speed is in blue, and the value 5.0f is in green. The editor interface shows a vertical scrollbar on the right and some file navigation icons on the left.

9~15行目  
ここで動作の処理を行っています

まず注目すべきはUpdate()の部分と  
その後の{}の組み合わせです

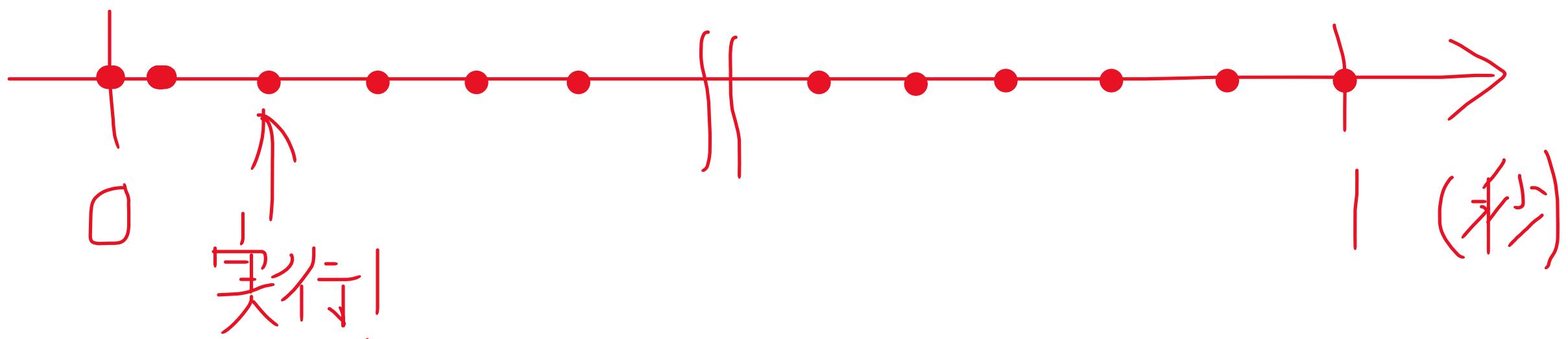
```
9 void Update()
10 {
11     if (Input.GetKey("right"))
12     {
13         this.gameObject.transform.Translate(speed * Time.deltaTime, 0, 0);
14     }
15 }
```

9~15行目

このUpdate()後の{}で囲まれた部分は

1秒間に何度も実行されます

(実行環境によりますが基本1秒間に60回くらいとかです)



9~15行目

で、Update()の中身なのですがすぐ下に  
if (Input.GetKeyDown("right"))と{}があります  
とあります

これは何となく文章から読み取れるかとは思います  
「→キーが押されている時」に行われる処理を{}内に  
書いています

```
9     void Update()
10    {
11        if (Input.GetKeyDown("right"))
12        {
13            this.gameObject.transform.Translate(speed * Time.deltaTime, 0, 0);
14        }
15    }
```

9~15行目

そしてそのif (Input.GetKey("right"))の{}の中身です

長いですが何をしているかというと  
「この 物体の 位置情報を 移動させる (x方向,y方向,z方向)」  
といった感じです

()内のy方向,z方向に当たる部分が0なので動くのはx方向だけです

```
this.gameObject.transform.Translate(speed * Time.deltaTime, 0, 0);
```

9~15行目

あとはx方向の部分に入っているやつです  
これは「speed」と「Time.deltaTime」との  
掛け算になっています

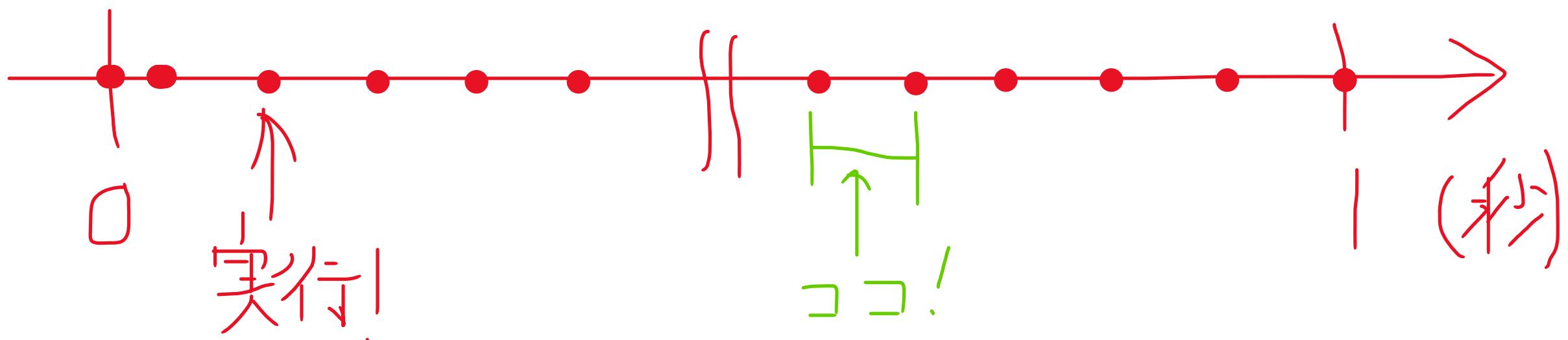
…Time.deltaTimeとはなんでしょうか…?

```
e (speed * Time.deltaTime,
```

9~15行目

実はTime.deltaTimeはさっきの図のココ！に当たります

1秒間に何回も繰り返されるところの  
短い間隔の時間になっています



9~15行目

なので「speed」と「Time.deltaTime」を掛け合わせた値は  
最初に設定した「5.0」よりも小さくなるわけですね

こうすることでUpdate()とif (Input.GetKey("right"))によって  
「→キーを押すと1秒間に何度も右にちょっと動く」

すなわち

「→キーを押すと右に動く」  
動きになっているわけです

```
this.gameObject.transform.Translate(speed * Time.deltaTime, 0, 0);
```

9~15行目

余談ですが今回のゲームは  
x座標が-9.0~9.0のところで作っています

なので移動スピードが5.0というのは1秒にxが5.0だけ  
右に動くということになります

Time.deltaTimeを取ると大体60倍くらいの速度になるので  
Playerが右に吹き飛びます

コード解説は以上です

ちょっと長くなってしまった…