UnityにおけるC#_1

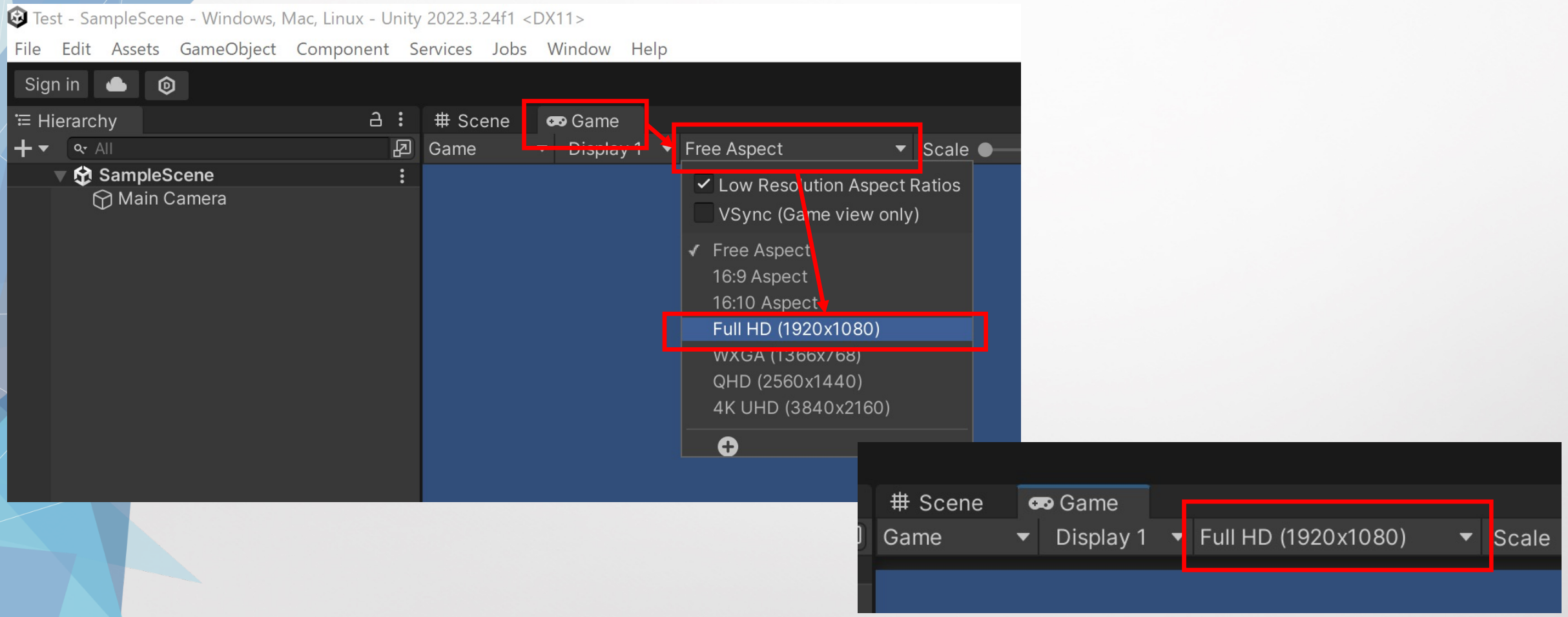
目次

１．準備
２．キーボードの入力の受け方
３．キャラクターの動かし方 その1
４．キャラクターの動かし方 その2

１．準備

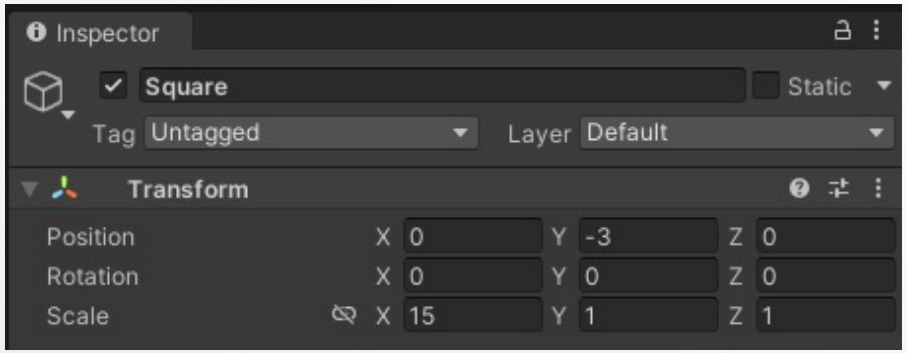
新規で2Dプロジェクトを作成、開いた人は、以下の操作をお願いします。

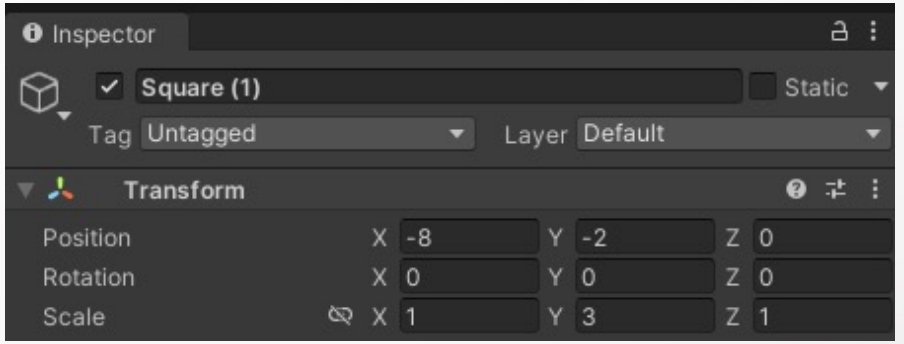
(画面の解像度の設定)

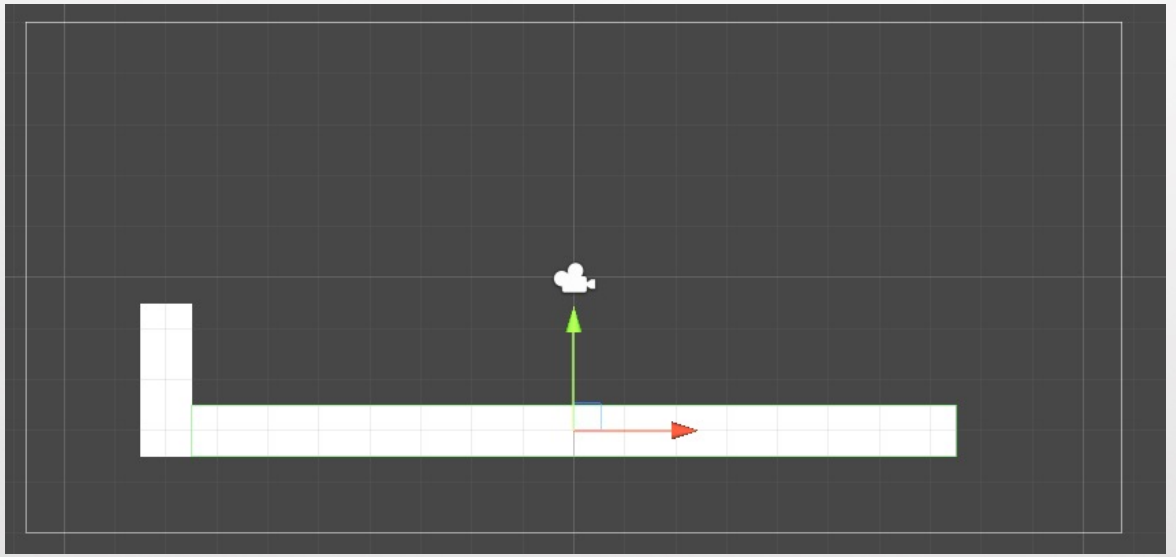


１．準備

ヒエラルキーから2D Object⇒Sprites⇒Squareを2つ作って、以下のように設定をお願いします。

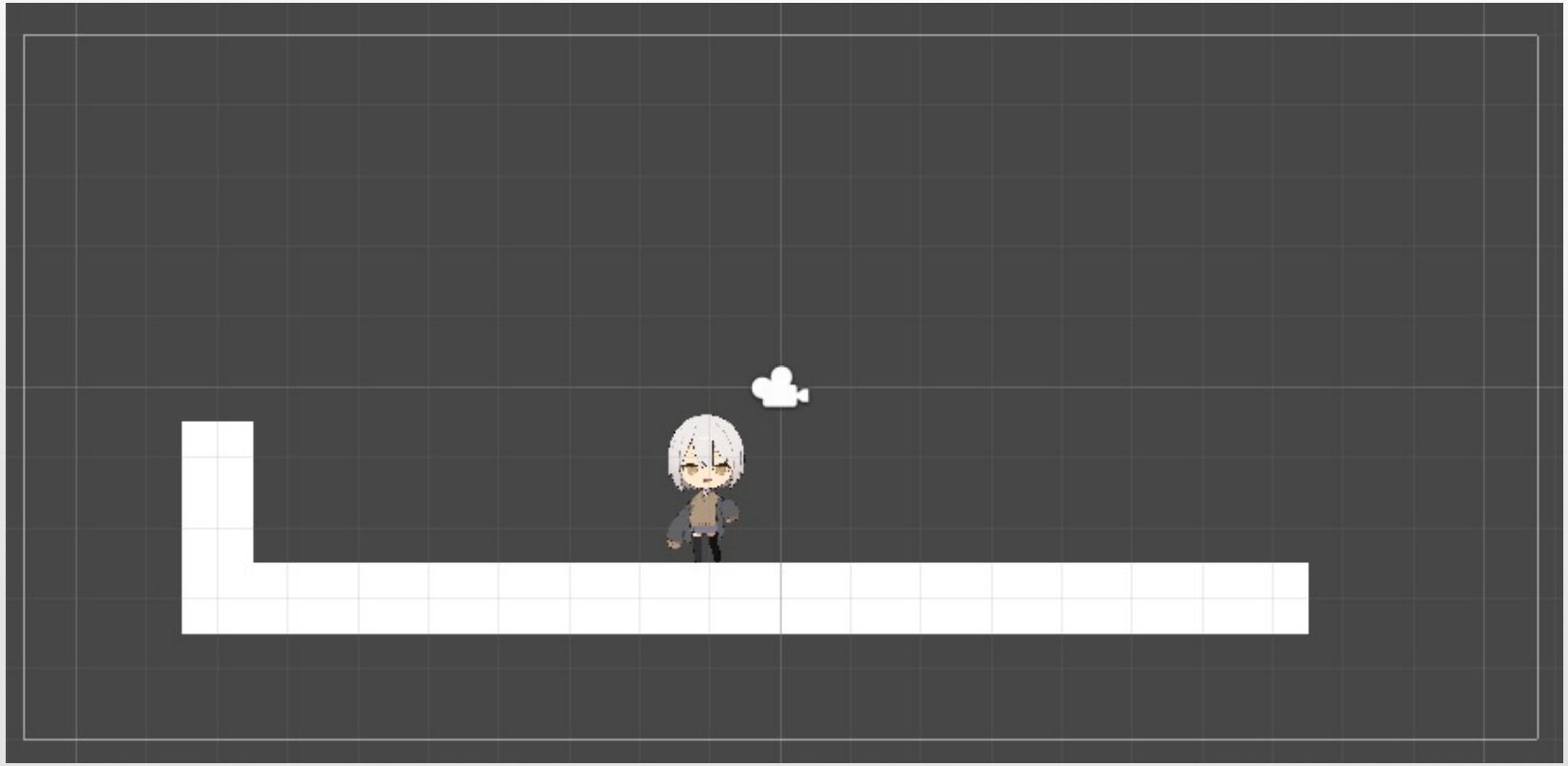






１．準備

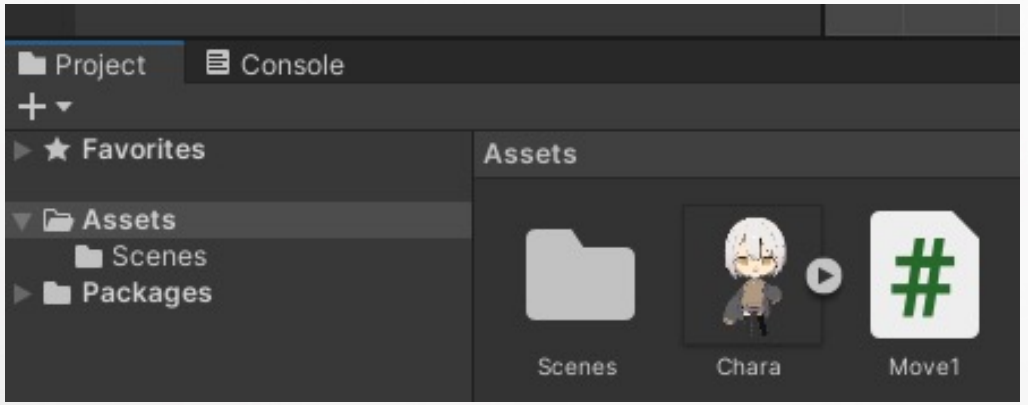
次に、シーンにCharaの画像をドラッグしてみよう！大きいので、Scaleを
X/Y/Zともに0.2に設定してみてください。
設定出来たら、位置を調整して地面に立たせてみてください。



２．キーボードの入力の受け方

キーボードの入力の受けるプログラムを作成していきます。

Projectで右クリック⇒Create⇒C# Scriptを選択し、名前を「Move1」としてください。

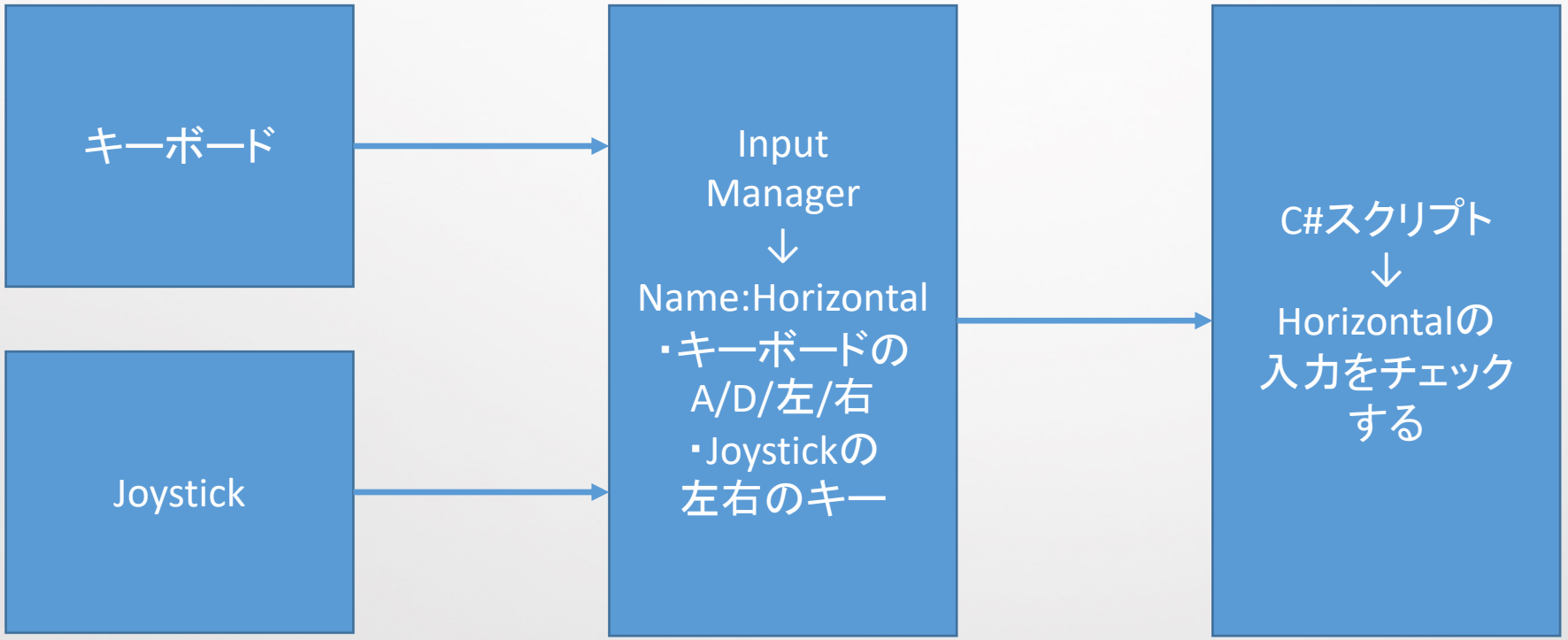


出来たらダブルクリックを押し、以下のプログラムを記載して保存してください。

```
16      void Update()
17      {
18          float dx = Input.GetAxis("Horizontal");
19          Debug.Log(dx);
20      }
```

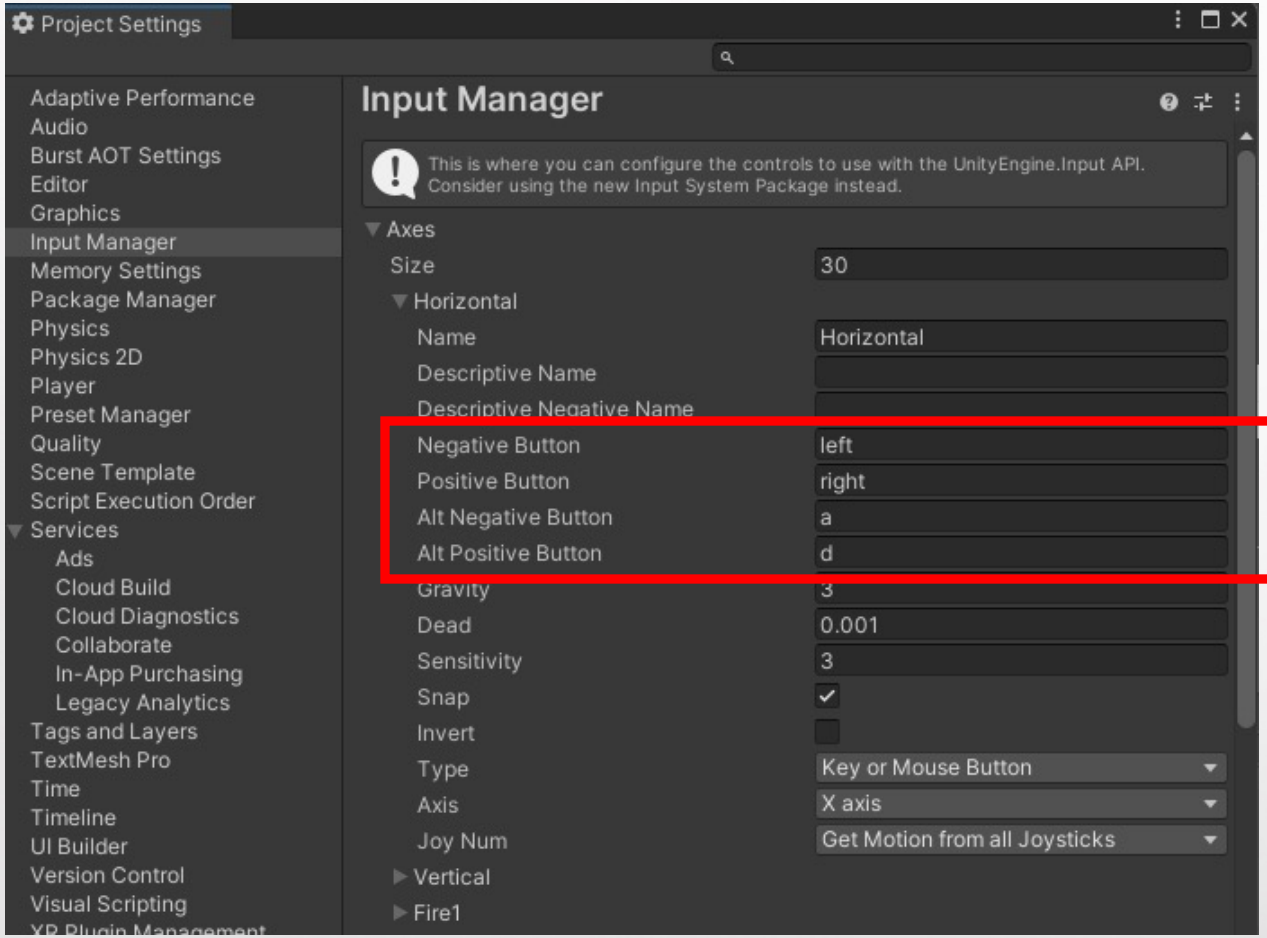
２．キーボードの入力の受け方

Input.GetAxisは、UnityのInputManagerという機能を使用しているため、InputManagerについて紹介します。



2．キーボードの入力の受け方

Edit⇒Project Settingから、InputManagerを選択。
右側の画面のHorizontalを選んでみてください。



Negativeはマイナス
Positiveはプラス

２．キーボードの入力の受け方

＜Input.GetKeyとInput.GetAxisの違い＞

Unityでは入力を受けとる方法として、Input.GetKeyという関数も
存在します。
こちらはキーボードの入力を個別に受け取る場合になるので、今回みたいな
WASDにも上下左右の矢印にも対応しなければならないときに、例えば

if(Wが押されたか || 上矢印が押されたか)

というように複数の条件を見ないといけないので、あまり使い勝手は良く無いです(その分、シンプルで分かりやすいですが。。。)

出来るだけ、Input.GetAxisを使うようにしてもらえれば良いかと思います。

３．キャラクターの動かし方 その1

まずは、座標移動という考えでキャラクターを動かしてみよう！

これは、座標(X座標/Y座標)をプログラムで直接指定して動かす方法。
簡単に移動させることが出来るが、瞬間移動での移動となるため物理的な
挙動とはならないので注意。

まずは、プログラムを作成していこう！先ほどのMove1.csを次のページのように
修正してください。
出来たら保存してUnityを起動し、エラーが無いことを確認してください。

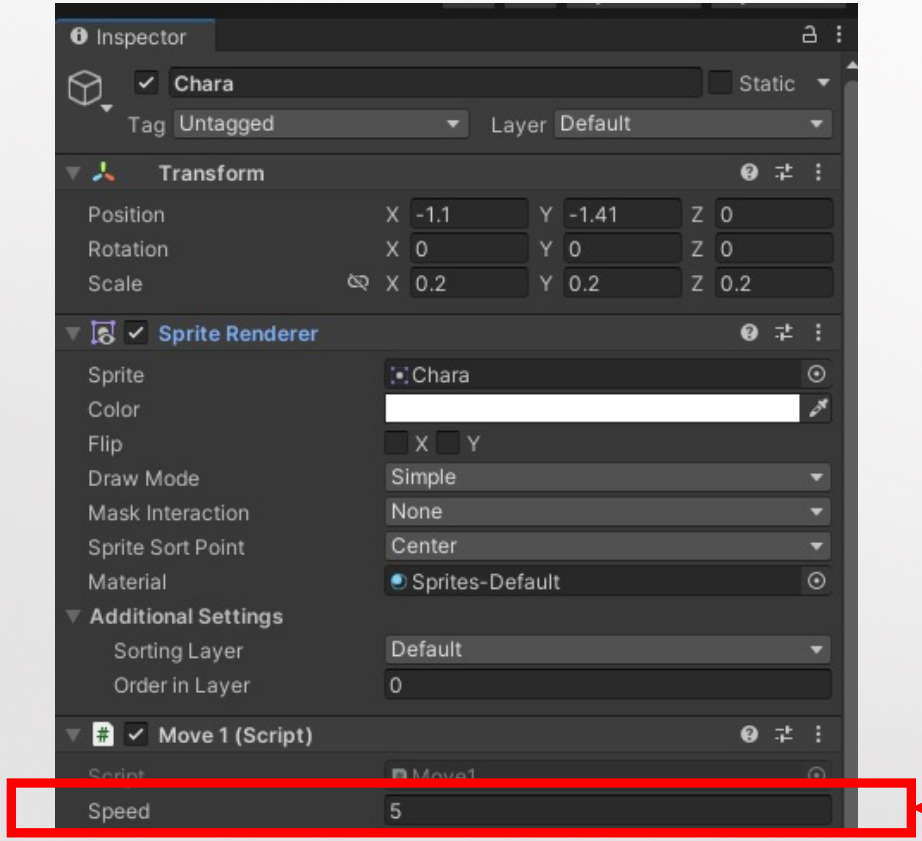
3．キャラクターの動かし方 その1

```
5  ^o  public class Move1 : MonoBehaviour
6      {
7          public float speed;  "5"
8
9          // Start is called before the first frame update
           イベント関数
10         void Start()
11         {
12
13         }
14
15         // Update is called once per frame
           イベント関数
16         void Update()
17         {
18             float dx = Input.GetAxis("Horizontal") * Time.deltaTime * speed;
19             Debug.Log(dx);
20
21             transform.Translate(dx,0,0);
22         }
23     }
```

3．キャラクターの動かし方 その1

出来たら、Unityの画面のMove1にSpeedという項目があるので、5程度の値を入力してください。

このSpeedはプログラムで書いたSpeedという値とリンクしています。




```
5  public class Move1 : MonoBehaviour
6  {
7      public float speed;  "5"
8
```

３．キャラクターの動かし方 その1

出来たら実際に動かしてみてください！

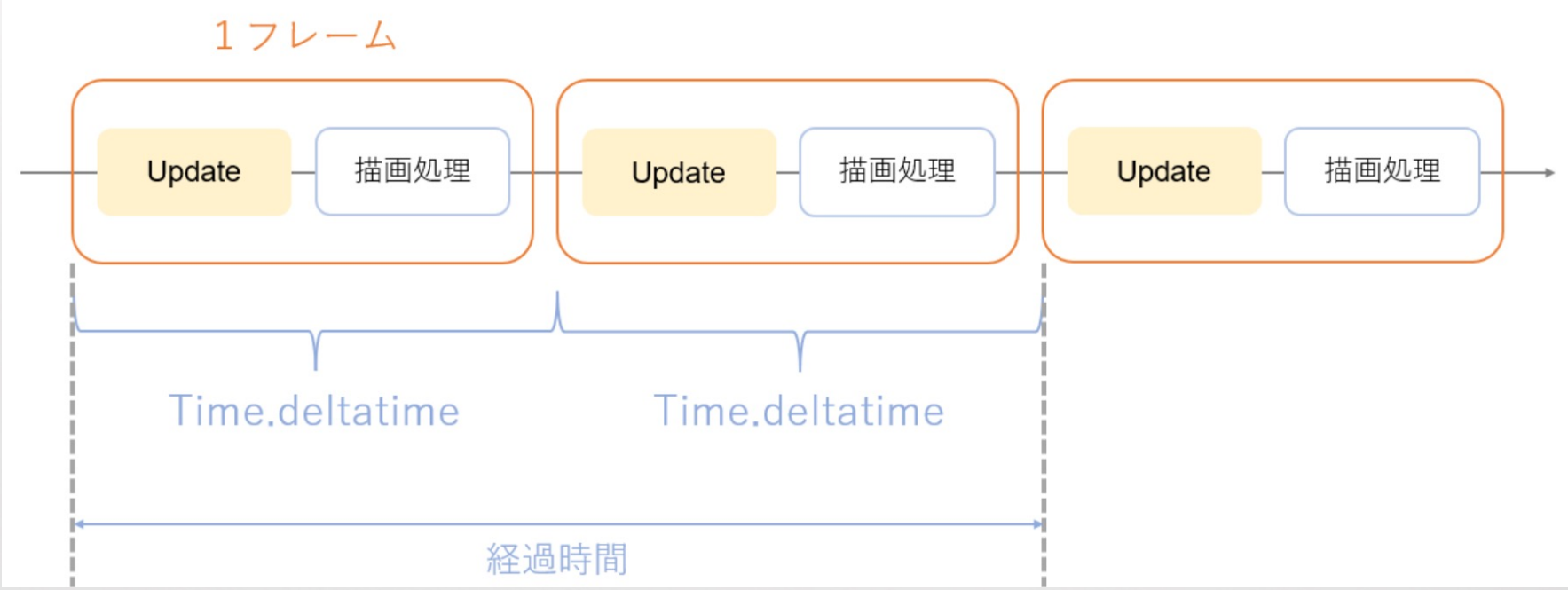
物理的な挙動は行わず、一定速度での移動になるかと思います。

３．キャラクターの動かし方 その1

＜プログラム解説＞

Time.deltaTime

⇨前回のUpdateの時間から、今回のUpdateが実行されるまでの時間を格納しています。つまり、1フレーム当たりの時間が格納されています！



３．キャラクターの動かし方 その1

1フレーム当たりの時間が格納されているということは、移動量に乗算することで1フレームに動くべき距離を算出することが出来るということです！！

＜移動量が5の場合の例＞

1秒間に50回Updateが実行される

⇨ 1 / 50 ＝ 0.02sec(これがTime.deltaTimeになる)

1回のUpdateで動かす量

⇨ 5 * 0.02 ＝ 0.1(1フレームあたり、0.1ずつ動かす)

50回のUpdateで動く量

⇨ 0.1 * 50 ＝ 5(50回のupdateで5動くことになる)

３．キャラクターの動かし方 その1

transform.Translate(X軸, Y軸, Z軸);

今の位置から、相対的に動かす距離を指定する。

例えば、

transform.Translate(10, 0, 0);

と指定した場合は今の位置からX軸の方向に+10する、という意味になる。

今回の場合だと、Input.GetAxisには何も押さないと0なので、

transform.Translate(0, 0, 0);

となり、動かさないというプログラムとなる。入力があればX軸が(1 * speed)、もしくは(-1 * speed)となるため今の位置から動くという動作になる。

３．キャラクターの動かし方 その1

<練習問題>

左右には動けているはずなので、今度はInput.GetAxisを使用して

上下に動けるようにしてください！

移動量は左右と同じくspeedの変数を使用してください！

３．キャラクターの動かし方 その1

<解答例>

上下は、InputManagerを確認するとVerticalという名前で登録されているので、以下のようになります！

```
void Update()
{
    float dx = Input.GetAxis("Horizontal") * Time.deltaTime * speed;
    float dy = Input.GetAxis("Vertical") * Time.deltaTime * speed;
    transform.Translate(dx,dy,0);
}
```

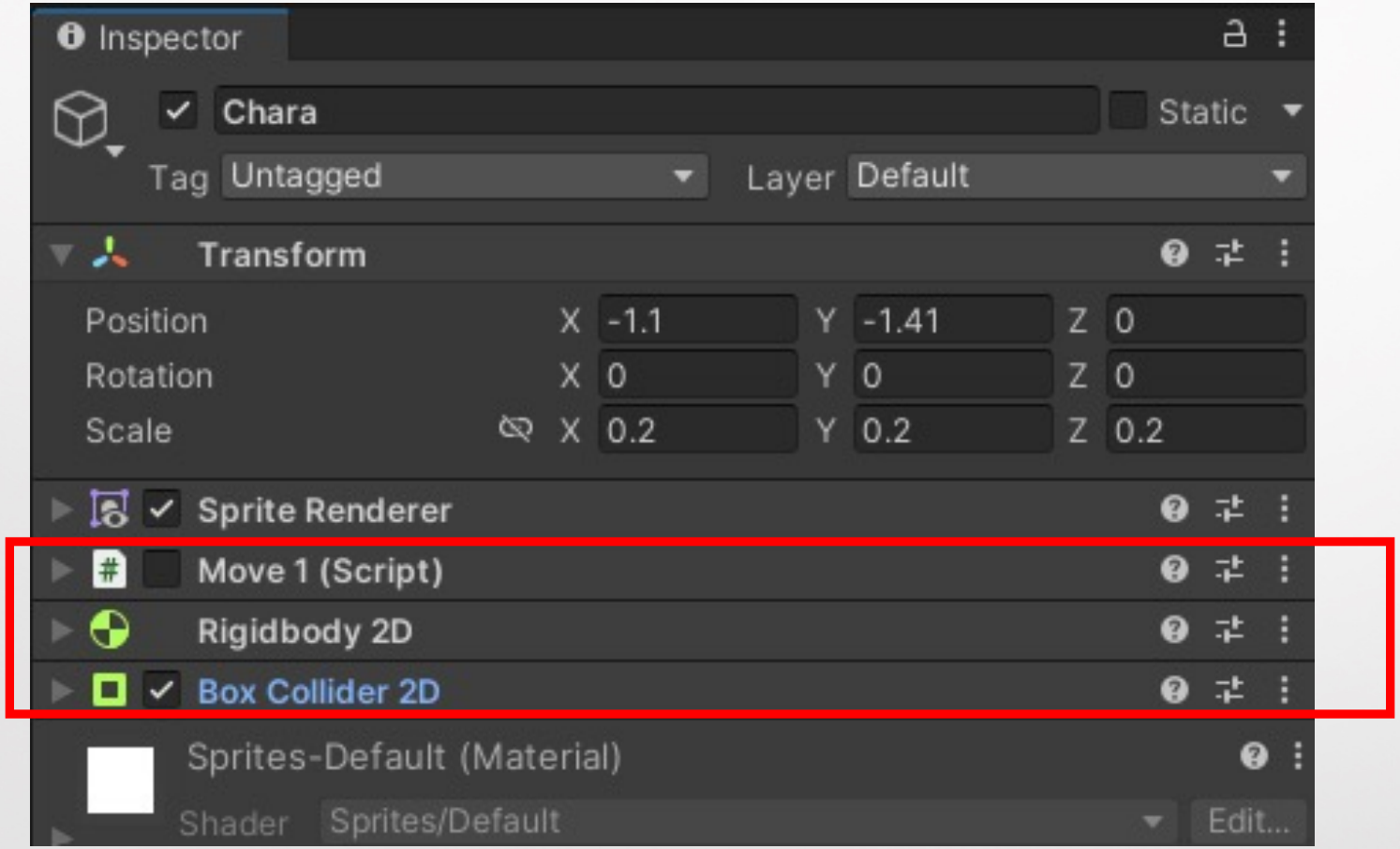
４．キャラクターの動かし方 その2

これでは面白くないので、次に物理演算を用いたキャラクターの移動を
やってみましょう！

UnityにはRigidbodyという物理演算があるので、そのコンポーネント(機能)
をプログラムで操作して動かすということをやります。
物理演算に関わらず、Unityの標準の機能をプログラムで使う場合は必須の
やり方となるので絶対覚えておいてください！！

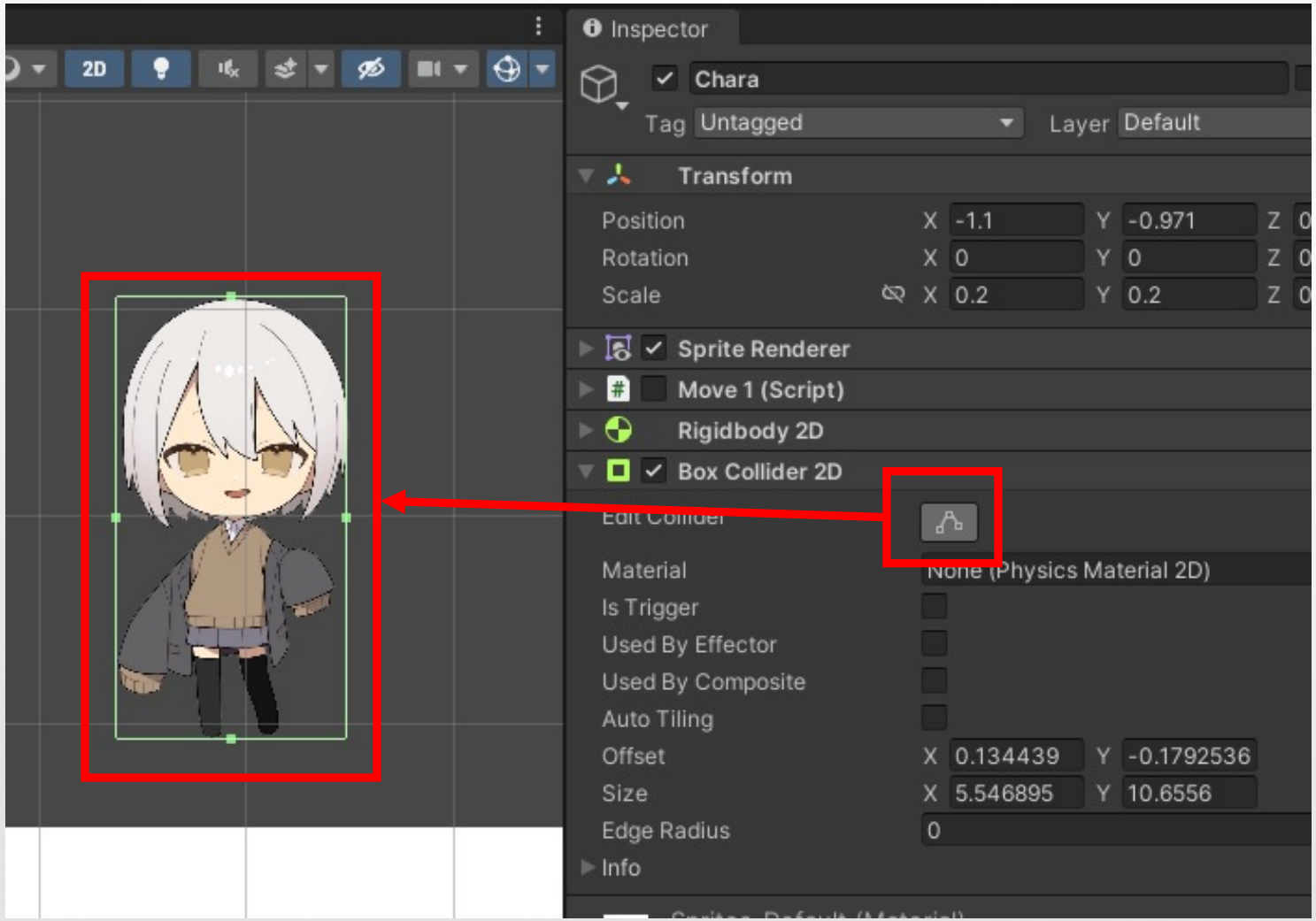
４．キャラクターの動かし方 その2

キャラクターを少し中に浮かせ、Move1のチェックを外し、Rigidbody 2DとBoxCollider 2Dを付けてください。



４．キャラクターの動かし方 その2

また、緑のBoxCollider 2Dの当たり判定を画像に合うように調整してください。



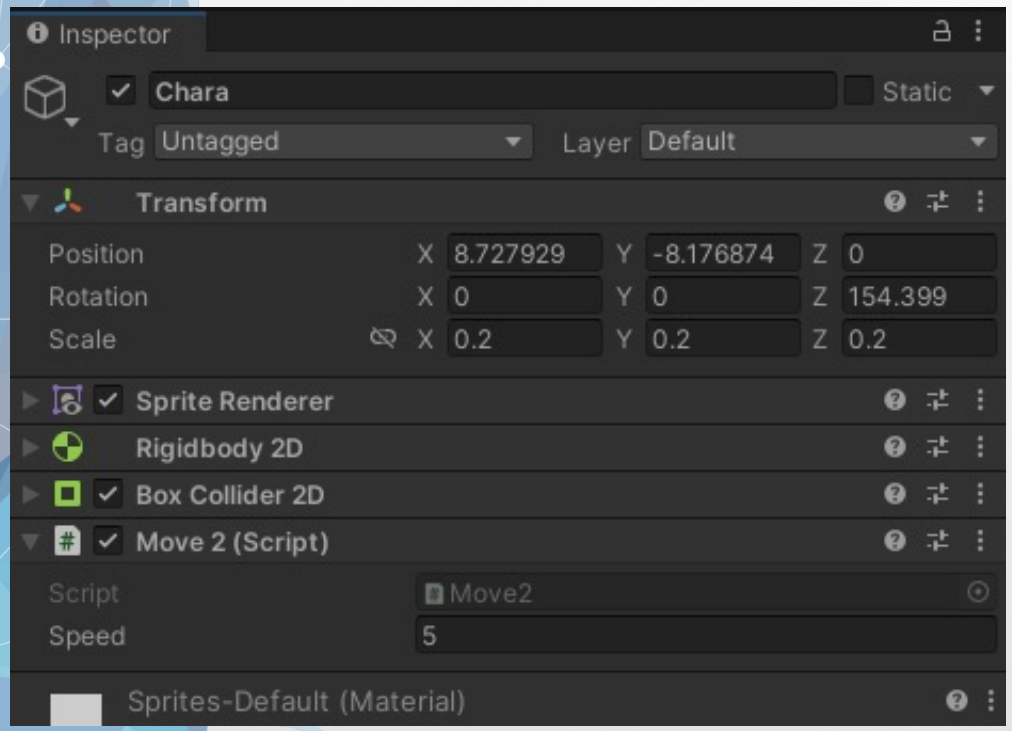
４．キャラクターの動かし方 その2

出来たらプログラムを作成していこう！Move2.csを新しく作成してください。

```
5  public class Move2 : MonoBehaviour
6  {
7      public float speed;  "5"
8
9      private Rigidbody2D rb2;
10
11     // Start is called before the first frame update
       イベント関数
12     void Start()
13     {
14         rb2 = GetComponent<Rigidbody2D>();
15     }
16
17     // Update is called once per frame
       イベント関数
18     void Update()
19     {
20         float dx = Input.GetAxis("Horizontal");
21
22         rb2.velocity = new Vector2(dx * speed, rb2.velocity.y);
23     }
24 }
```

４．キャラクターの動かし方 その2

プログラムが完成したら、Unityに戻ってMove2のコンポーネントを付けてください。Speedは先ほどと同じで5程度で大丈夫です。
壁にぶつかったり、地面が無い場合は落ちたりしましたでしょうか？



４．キャラクターの動かし方 その2

先ほど話をしたUnityの物理演算を使う、ということをプログラム上で実施しているのがこの行です！

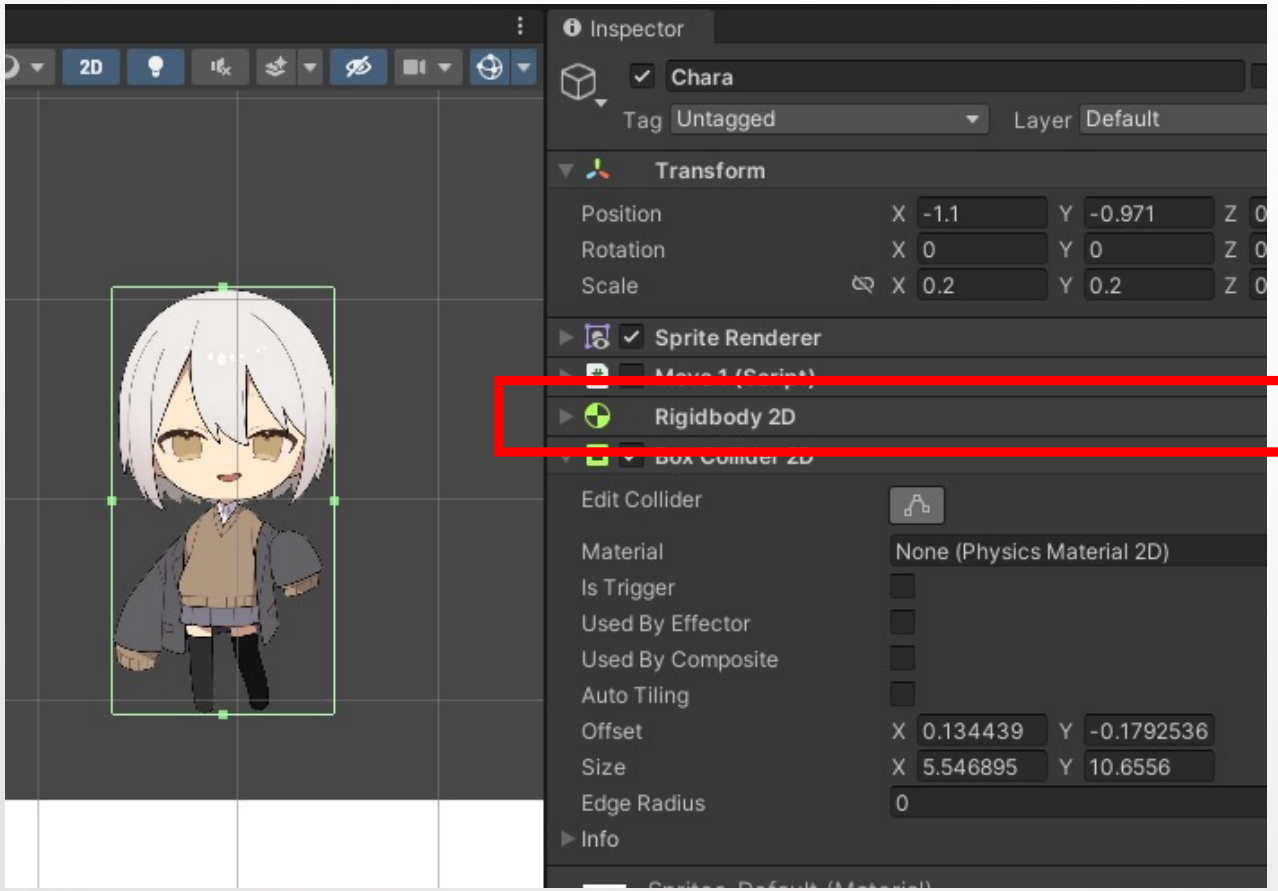
```
private Rigidbody2D rb2;
```

```
rb2 = GetComponent<Rigidbody2D>();
```

```
rb2.velocity = new Vector2(dx * speed, rb2.velocity.y);
```

Startの中の右辺のGetComponent<Rigidbody2D>();とは、物理演算の機能をプログラムで使わせてね、という内容になっています。
(左辺はRigidbody 2Dの情報を受け取る変数(正確にはクラス)です)

４．キャラクターの動かし方 その2



このコンポーネントのこと

この物理演算の機能の中には、指定した方向に動かすというプログラムが用意されているので、それを使って移動させています。

４．キャラクターの動かし方 その2

Rigidbody2Dにどのような機能があるのかはUnityの公式ページで展開されているので、一度確認してみてください。

リンク

今回使っているのは、velocityという機能(日本語訳)

Rigidbody2D .velocity

パブリックVector2 **速度**;

説明

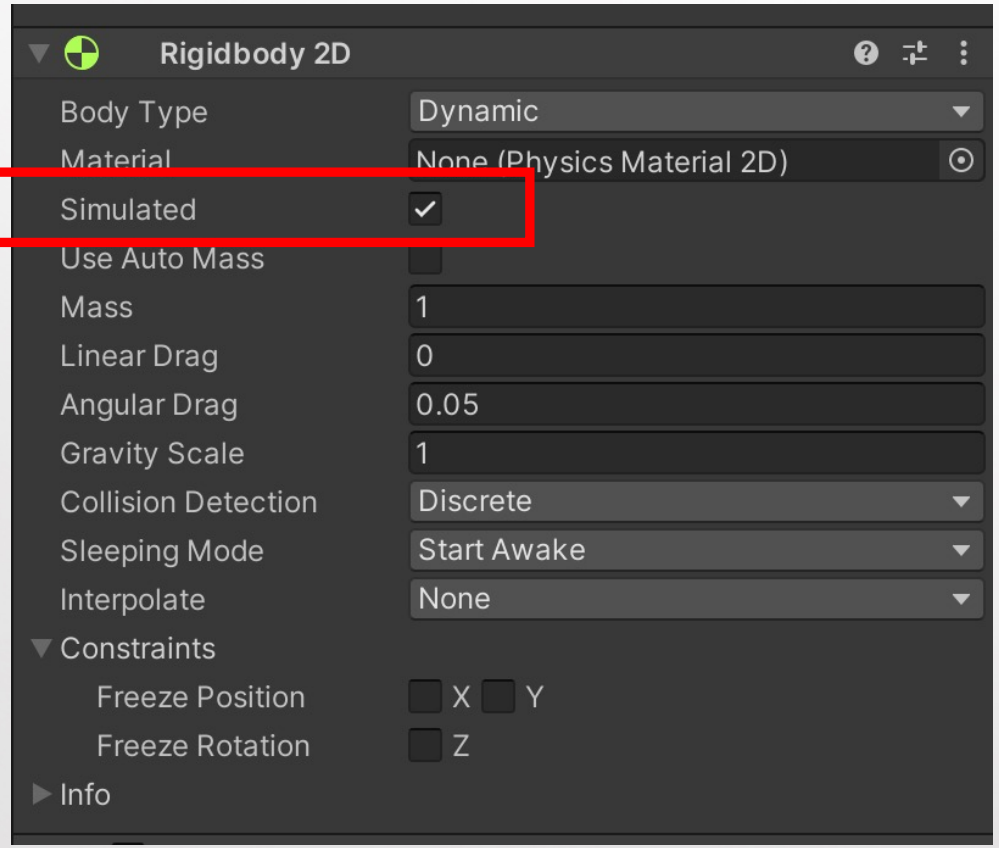
リジッドボディの線速度 (単位/秒)。

速度は、X 方向と Y 方向の成分を持つベクトルとして指定されます (2D 物理学には Z 方向はありません)。通常、値は直接設定されるのではなく、*force*を使用して設定されます。インスペクターでドラッグを無効にして、速度の緩やかな減衰を停止します。

４．キャラクターの動かし方 その2

＜練習問題＞

スクリプトを新しく作成し、スクリプトの中からRigidbody2Dのsimulated
をOFFにするプログラムを作成してみてください！



４．キャラクターの

＜解答例＞

スクリプトリファレンスを確認すると、simulatedはbool型で指定出来るので、falseを指定すればOKです。

```
public class Test : MonoBehaviour
{
    private Rigidbody2D rb2;

    // Start is called before the first frame update
    void Start()
    {
        rb2 = GetComponent<Rigidbody2D>();
        rb2.simulated = false;
    }
```