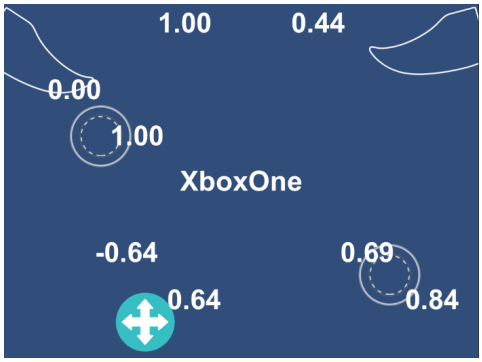
Xbox Controller : Xbox コントローラー



ゲーム制作で頻繁に用いられるゲームコントローラーについて、Unity で情報を取得してゲームに活かす方法を紹介します。

この概念が判ると、操縦席を想定したハンドルやトリガーなどのタイプのコントローラーでも Unity で扱ってゲーム作りが可能になります。

また、今回はコントローラーを1台で利用する手法に限定し、Unity のインプット・マネージャーの理解を中心に説明しています。

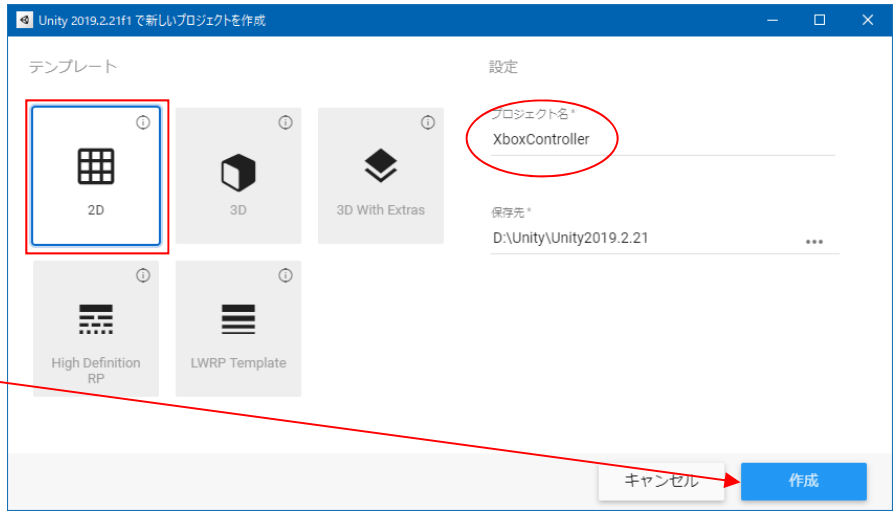
2台以上を接続して対戦プレイする場合は、また異なるアプローチが推奨されることに留意して下さい。

【STEP1】 プロジェクトの準備

- Unity を起動し、新規プロジェクト XboxController を作成します。

後でも修正できるのですが、テンプレートに2Dを選択します。

ボタン作成を押下します。



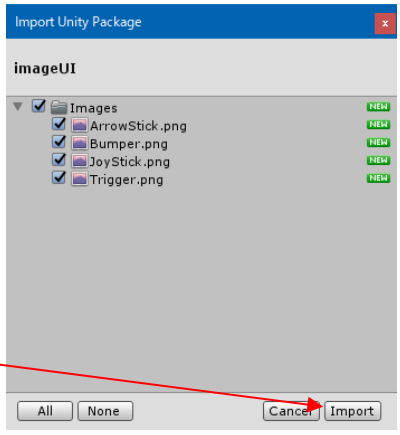
- Game 画面のサイズを Standalone (1024x768)に設定します。

【STEP2】 外部素材を取得

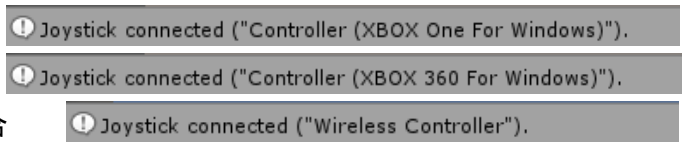
配布された素材を読み込みます。

- メニューAssets から Import Package > Custom Package と進み、配布した imageUI.unitypackage を指定します。

内容物が表示されたら Import を押下します。

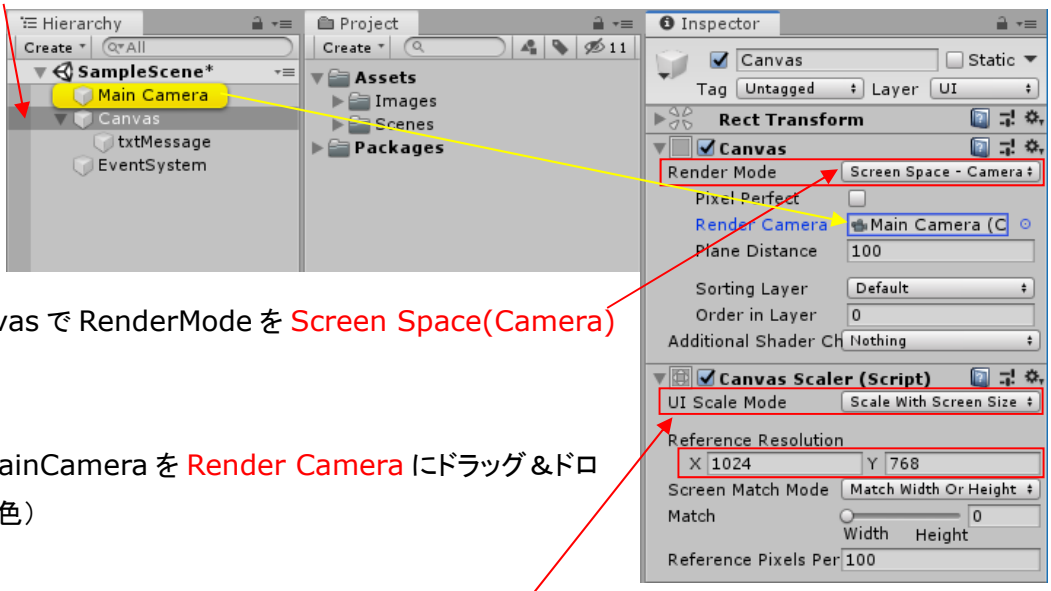


- このタイミングで、ゲームコントローラーを USB 接続します。
  Xbox One Controller For Windows の場合
  Xbox360 Controller For Windows の場合
  Playstation4 DualShock Controllerfor の場合



【STEP3】 検出したボタン押下の表示

- ヒエラルキー欄の Create から UI > Text を選択し、txtMessage と命名します。
- 同時に自動的に制作された Canvas について、先に設定を行います。
  ヒエラルキー欄の Canvas を選択します。

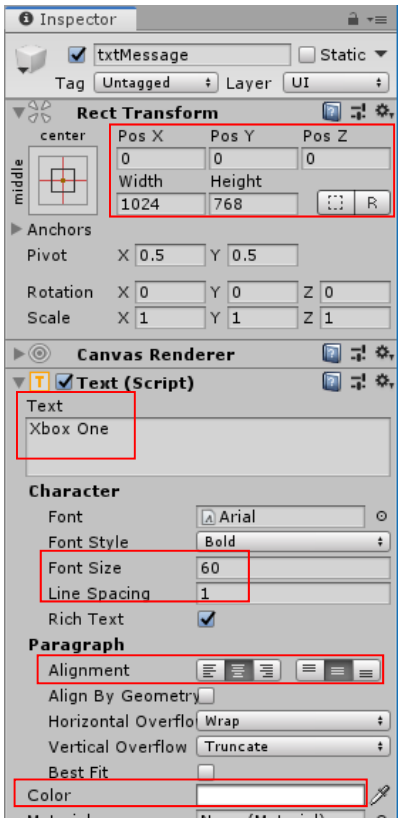


- インスペクタの Canvas で RenderMode を Screen Space(Camera) に設定します。

- ヒエラルキー欄の MainCamera を Render Camera にドラッグ＆ドロップします。（矢印黄色）

- インスペクタの Canvas Scaler で UI Scale Mode を Scale With Screen Size に設定します。
  さらに Reference Resolution を X:1024 / Y:768 に設定します。

- ヒエラルキー欄の txtMessage を選択し、インスペクタでパラメータを設定します。

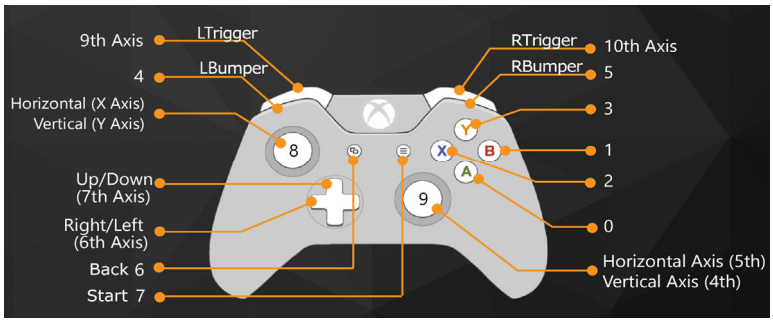


【STEP4】 インプット・マネージャーの設定

ゲームコントローラーのAボタンを押したら、画面中央に「A」と表示させます。

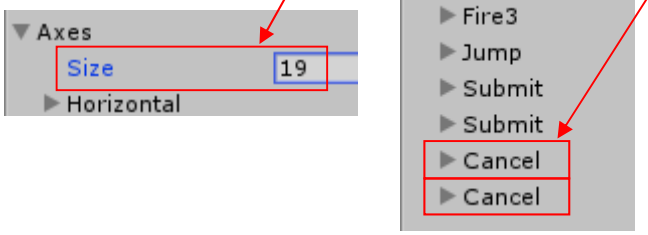
今回の場合、ゲームコントローラーのAボタン押下では、機器から0番の信号が出ます。

（機器メーカーによって割り当ては異なります。図を参照）

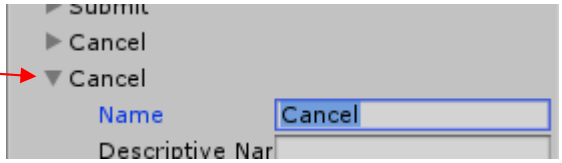


- メニューEdit > Project Setting を選択します。
  Input に切り替えて▼Axes をクリックして開きます。
  規定値でのインプット名の設定が18個、表示されます。

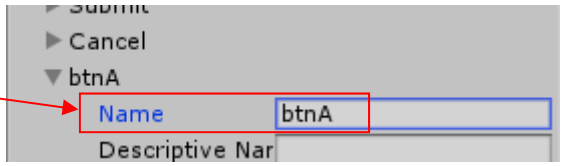
- Size（サイズ）を修正して 19 とすると、最後の項目が複製されます。



- 最終行（19個目）の▼Cancel をクリックし、項目を開きます。

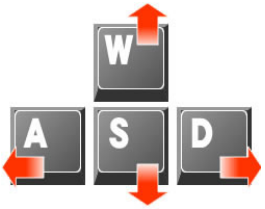


- Name（インプット名）を btnA と命名します。

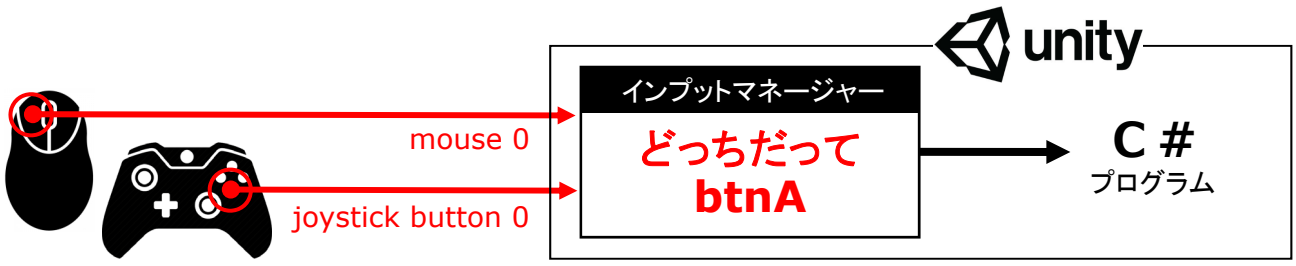


ゲームコントローラーが無い時や、ゲームコントローラーに不具合があった時に、キーボードとマウスでもゲーム制作やプレイが続けられることを考えるべきです。

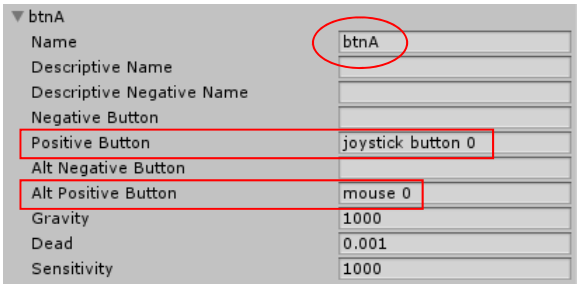
Aボタンですので、キーボードのAキーで代替できるかを考えると、デフォルト（最初からの設定）でW・A・S・Dという一群に属しており、ジョイスティックの代替キーとして既に使われてしまっています。



一般的にAボタンはマウスのLMBで代替させることが多く、ここでもその手法を採用します。



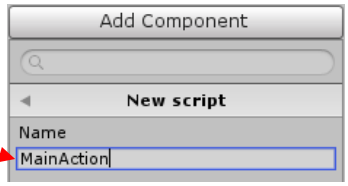
- インプット名 btnA のパラメータを設定します。
  - Positive Button : joystick button 0
  - Alt Positive Button : mouse 0



【STEP5】 ボタン系の受け取り処理

受け取ったインプット名：btnA を受信するプログラムを記述します。

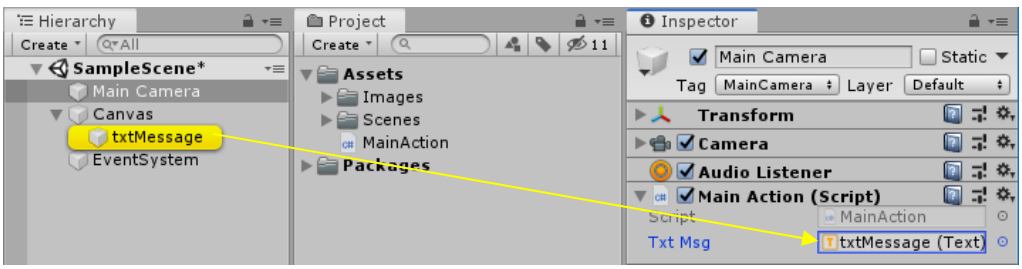
- ヒエラルキー欄の MainCamera を選択し、Add Component から最下段の New Script を選択します。名称を MainAction とします。



- スクリプト MainAction を次のように編集します。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI; //uGUIを用いるのに必要

public class MainAction : MonoBehaviour {

    public Text txtMsg;
    string EscText; //初期メッセージの退避エリア

    void Start() {
        //初期メッセージの退避
        EscText = txtMsg.text;
    }

    void Update() {
        if (Input.GetButton("btnA")) {
            txtMsg.text = "A";
        } else {
            txtMsg.text = EscText;
        }
    }
}
```

- ヒエラルキー欄の MainCamera を選択すると、インスペクタのスクリプト MainAction に項目 TxtMsg が登場しているので、txtMessage をマウスでドラッグ &ドロップします。



● プレイボタンを押下します。 

ゲームコントローラーのAボタンを押している間、画面に「A」と表示されるか確認します。

またマウスのLMBで代替機能が果たせているか？も確認します。

【学習項目：GetButton】

インプット名で情報取得する際に用いるのが GetButton です。Update 処理の中で用いることで、3つのタイミングでボタンの動きを検知することが出来ます。

①押した瞬間を1度だけ検出するのが GetButtonDown

③離した瞬間を1度だけ検出するのが GetButtonUp

②押している間、秒間 100 回検出が GetButton

【STEP6】 Bボタン押下を検出する

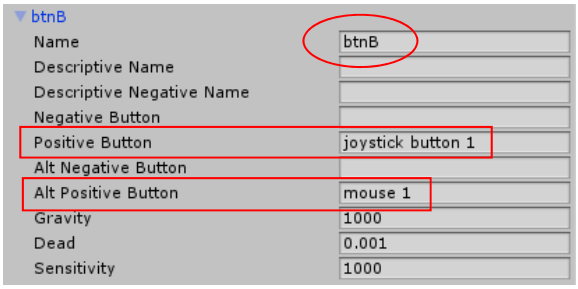
B ボタン押下は機器から1の信号が出ています。プログラムからはこれをインプット名 btnB で受信できるようにします。また、マウスの RMB に代替機能を持たせるのが一般的ですので採用します。

● インプット・マネージャーを開き、Size を 20 にします。

● インプット名 btnA が2つになっているので、増えた方（下の方）の名称を btnB に変更します。

btnB のパラメータを設定します。

- Positive Button : joystick button 1
- Alt Positive Button : mouse 1



● スクリプト MainAction を次のように編集します。

実際のゲームでは、Bボタンを押しながらLボタンを押す。。。などという操作もありそうです。

その場合は else if ではなく、普通に if 文を用いて両方を押しているか？のチェックをすることになります。

```
//～前略～

    void Update() {
        if (Input.GetButton("btnA")) {
            txtMsg.text = "A";
        } else if (Input.GetButton("btnB")) {
            txtMsg.text = "B";
        } else {
            txtMsg.text = EscText;
        }
    }
}
```

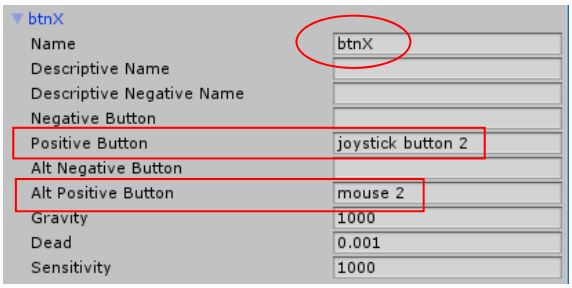
- プレイボタンを押下します。 
  ゲームコントローラーのBボタンを押している間、画面に「B」と表示されるか確認します。
  またマウスの RMB で代替機能が果たせているか？も確認します。

【STEP7】 Xボタン押下を検出する

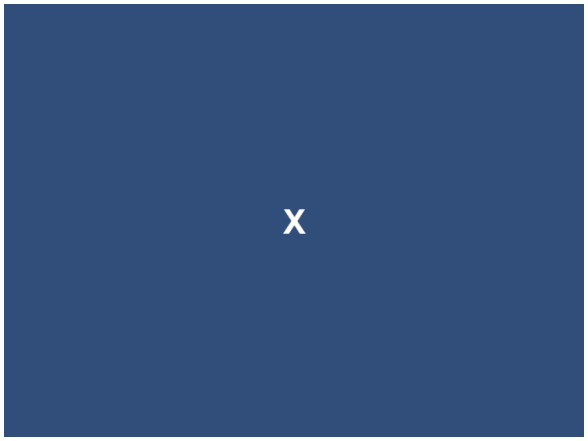
Xボタン押下は機器から2の信号が出ています。プログラムからはこれをインプット名 btnX で受信できるようにします。また、マウスの MMB に代替機能を持たせるのが一般的ですので採用します。

- インプット・マネージャーで、Size を 21 にします。

- インプット名 btnB が2つになっているので、増えた方（下の方）の名称を btnX に変更します。
  btnX のパラメータを設定します。
  - Positive Button : joystick button 2
  - Alt Positive Button : mouse 2



- **【命題①】**
  XボタンとMMBの押下で画面に文字「X」が表示されるように、スクリプト MainAction を編集してください。
- プレイボタンを押下します。 
  ゲームコントローラーのXボタンを押している間、画面に「X」と表示されるか確認します。
  またマウスの MMB で代替機能が果たせているか？も確認します。

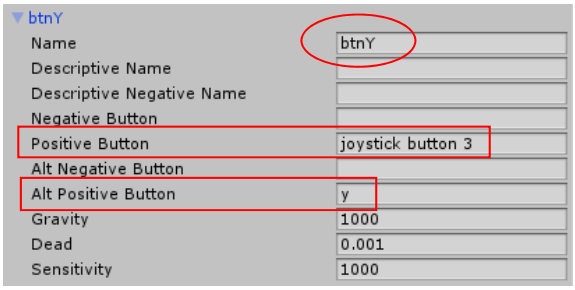


【STEP8】 Yボタン押下を検出する

Yボタン押下は機器から3の信号が出ています。プログラムからはこれをインプット名 btnY で受信できるようにします。マウスのボタン3個を使い切ってしまったので、キーボードの Y ボタン「y」を借りることにします。

- インプット・マネージャーで、Size を 22 にします。

- インプット名 btnX が2つになっているので、増えた方（下の方）の名称を btnY に変更します。
  btnY のパラメータを設定します。
  - Positive Button : joystick button 3
  - Alt Positive Button : y

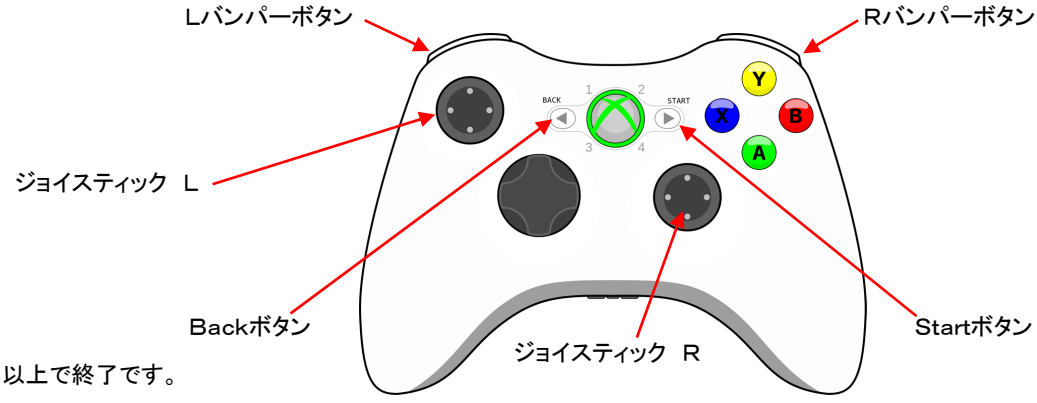


- **【命題②】**
  Y ボタンとキーボードのYボタン押下で、画面に文字「Y」が表示されるように、スクリプト MainAction を編集してください。
- プレイボタンを押下します。

ゲームコントローラーのYボタンを押している間、画面に「Y」と表示されるか確認します。
またキーボードのYボタンで代替機能が果たせているか？も確認します。

- **【命題③】**
  以下の6個のボタン番号について、指定する **インプット名** で受信して、画面に **文字** を表示して下さい。実はジョイスティックは、そのまま真っ直ぐに押すことが出来ます。

ボタン	インプット名	コントローラーでの番号	キーボードでの代替キー	画面に表示する文字
Lバンパーボタン	**btnL**	joystick button 4	l（小文字のエル）	**L**
Rバンパーボタン	**btnR**	joystick button 5	r	**R**
Back ボタン	**btnBack**	joystick button 6	k	**Back**
Start ボタン	**btnStart**	joystick button 7	t	**Start**
ジョイスティックL	**btnJoyL**	joystick button 8	left shift	**JoyL**
ジョイスティックR	**btnJoyR**	joystick button 9	right shift	**JoyR**



ボタン系は以上で終了です。

【STEP9】 左ジョイスティックの操作を取得

次は軸を回転させる入力操作で、「ジョイスティック」と「トリガー（引き金）」「カーソルパッド」を用います。

これら3つのパーツを動作すると出て来る回転値をインプット名で取得することになります。

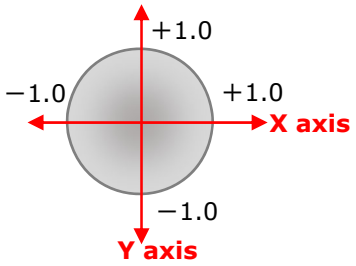
デフォルト（規定値）のインプット・マネージャーには、ジョイスティックが1つだけ定義されています。とは言え、左右への回転と、前後への回転に分解して受信するので、インプット名は2個で定義されています。

- 左右への情報はインプット名 Horizontal で受信し、代替主キーは「←、→」で副キーが「aキー、dキー」です。
- 前後への情報はインプット名 Vertical で受信し、代替主キーは「↑、↓」で副キーが「wキー、sキー」です。

Xbox コントローラーの場合は、左ジョイスティックが相当しています。

左右への操作は、X axis という信号が出ていて、

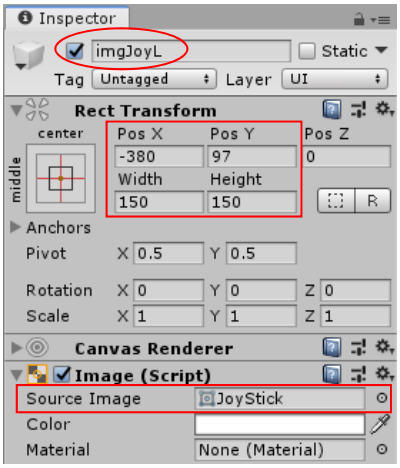
前後への操作は、Y axis という信号が出ています。



画面に左スティックの情報を表示させてみます。

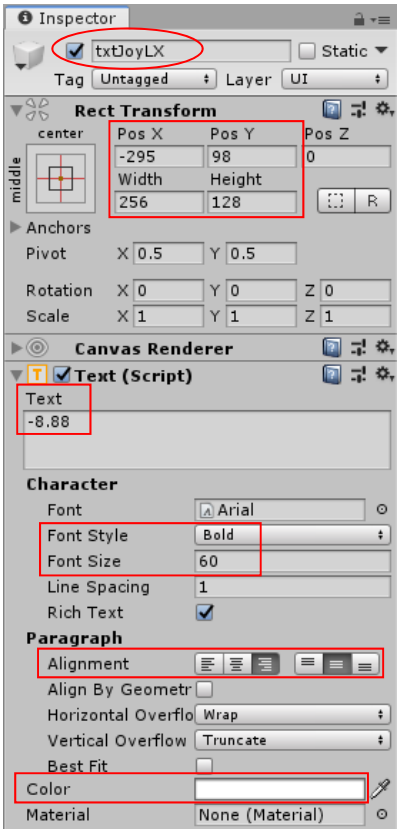
- ヒエラルキー欄の Create から UI ＞ Image を選択し、imgJoyL と命名します。

  インスペクタでパラメータを設定します。



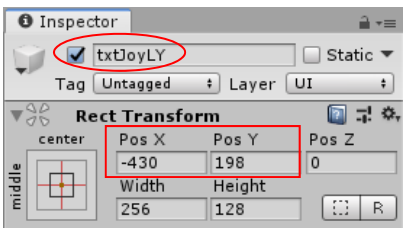
- ヒエラルキー欄の Create から UI ＞ Text を選択し、txtJoyLX と命名します。

  インスペクタでパラメータを設定します。



- この txtJoyLX を選択して複製（Ctrl + D）し、名称を txtJoyLY とします。

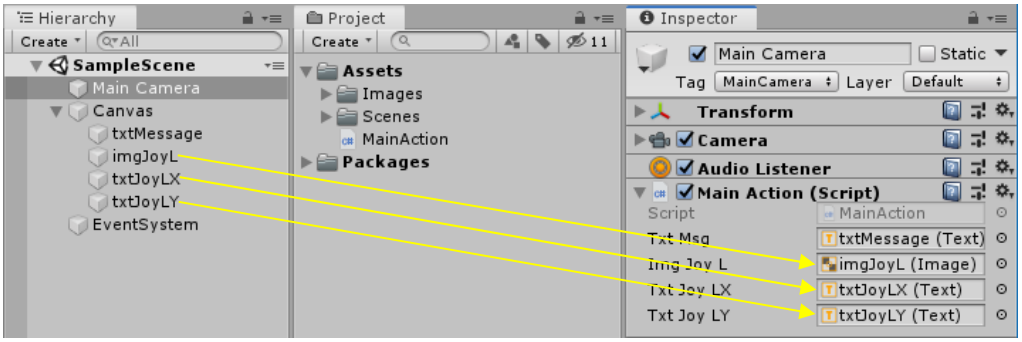
  インスペクタでパラメータを設定します。



● プログラム MainAction を以下のように修正します。

```
//～前略～
public class MainAction : MonoBehaviour {

    public Text txtMsg;
    string EscText; //初期メッセージの退避エリア
    public Image imgJoyL;
    public Text txtJoyLX;
    public Text txtJoyLY;
    Vector2 PosJoyL; //左スティックの初期位置

    void Start() {
        //初期メッセージの退避
        EscText = txtMsg.text;
        //左スティックの初期位置を退避
        PosJoyL = new Vector2(
            imgJoyL.rectTransform.position.x,
            imgJoyL.rectTransform.position.y);
    }

    void Update() {
        //左スティック
        float LX = Input.GetAxis("Horizontal");
        float LY = Input.GetAxis("Vertical");
        txtJoyLX.text = LX.ToString("f2");
        txtJoyLY.text = LY.ToString("f2");
        imgJoyL.rectTransform.position = PosJoyL + new Vector2(LX,LY);

//～後略～
```

● ヒエラルキー欄の MainCamera を選択し、インスペクタに登場した項目を設定します。



● プレイボタンを押下します。

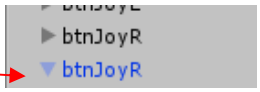
左スティックと方向キーで入力が管理できていることが判ります。

【STEP10】 右ジョイスティックの操作を取得

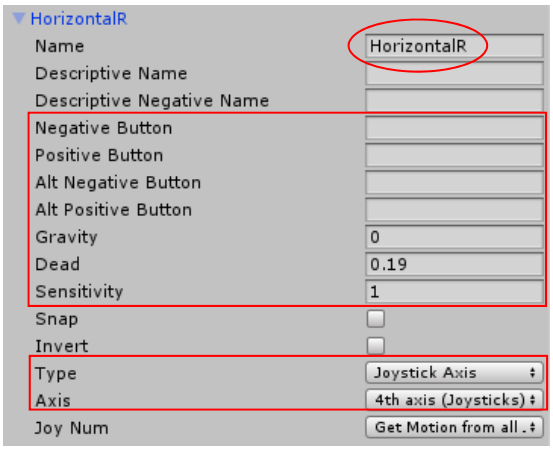
右ジョイスティックは、既存の左と違って、インプット名すら定義されていません。左右と上下のインプット名を2つ作ります。

- インプット・マネージャーで軸の数 Size を 29 にします。

最下段の btnJoyR が複製されます。

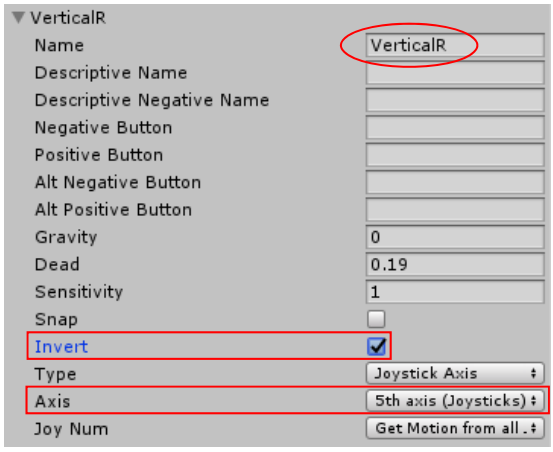


- 名称を HorizontalR とし、パラメータを設定します。

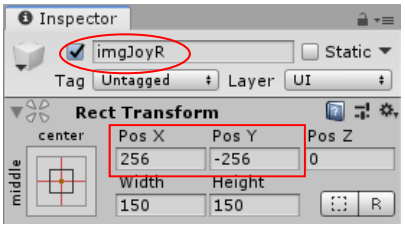


- 軸の数 Size を 30 にします。

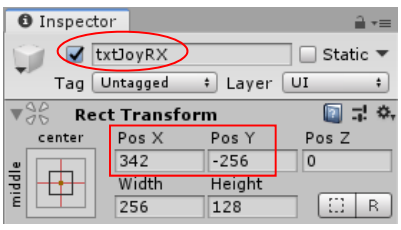
- 増えた方の名称を VerticalR とし、パラメータを設定します。
  同じ部品が逆向きに使われており、Invert：逆を設定します。



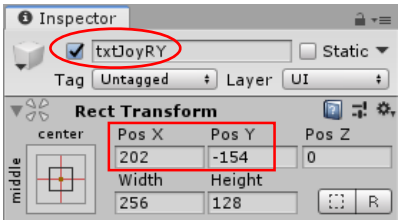
- ヒエラルキー欄の imgJoyL を複製（Ctrl ＋ D）し、imgJoyR と命名します。インスペクタでパラメータを設定します。



- ヒエラルキー欄の txtJoyLX を複製（Ctrl ＋ D）し、txtJoyRX と命名します。インスペクタでパラメータを設定します。



● ヒエラルキー欄の txtJoyLY を複製(Ctrl ＋ D)し、txtJoyRY と命名します。インスペクタでパラメータを設定します。

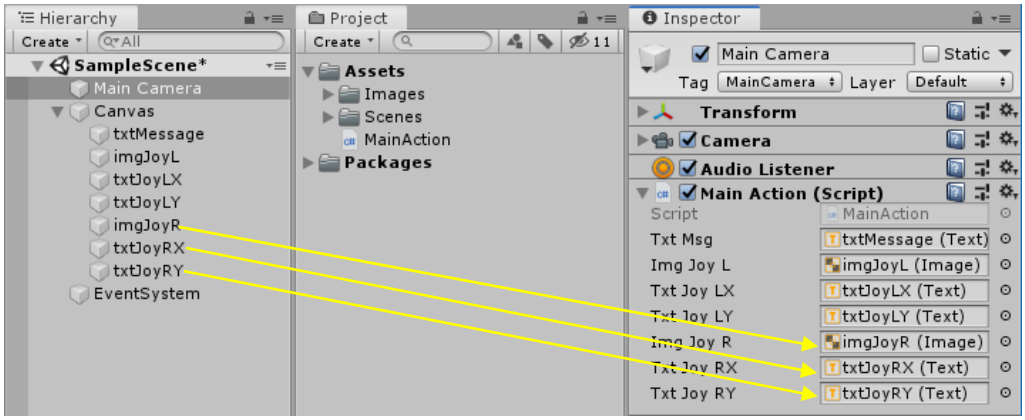


こんな感じになります。

● プログラム MainAction を以下のように修正します。

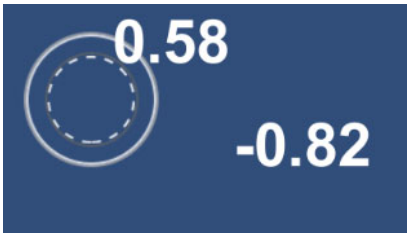
```
//～前略～

    public Image imgJoyR;
    public Text txtJoyRX;
    public Text txtJoyRY;
    Vector2 PosJoyR; //右スティックの初期位置

    void Start() {
        //初期メッセージの退避
        EscText = txtMsg.text;

        //右スティックの初期位置を退避
        PosJoyR = new Vector2(
            imgJoyR.rectTransform.position.x,
            imgJoyR.rectTransform.position.y);

        //左スティックの初期位置を退避
        PosJoyL = new Vector2(
            imgJoyL.rectTransform.position.x,
            imgJoyL.rectTransform.position.y);
    }

    void Update() {
        //右スティック
        float RX = Input.GetAxis("HorizontalR");
        float RY = Input.GetAxis("VerticalR");
        txtJoyRX.text = RX.ToString("f2");
        txtJoyRY.text = RY.ToString("f2");
        imgJoyR.rectTransform.position = PosJoyR + new Vector2(RX, RY);

//～後略～
```

● ヒエラルキー欄の MainCamera を選択し、インスペクタに登場した項目を設定します。



● プレイボタンを押下します。

右スティックで入力が管理できていることが判ります。

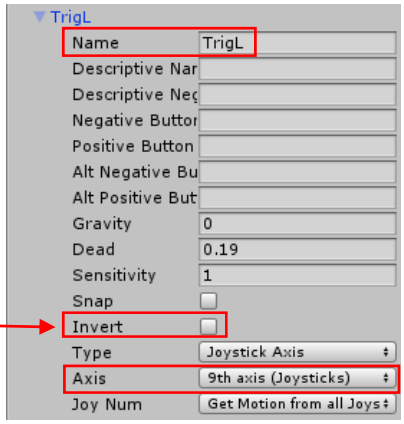


【STEP10】 左右のトリガー操作を取得

● インプット・マネージャーで軸の数 Size を31にします。

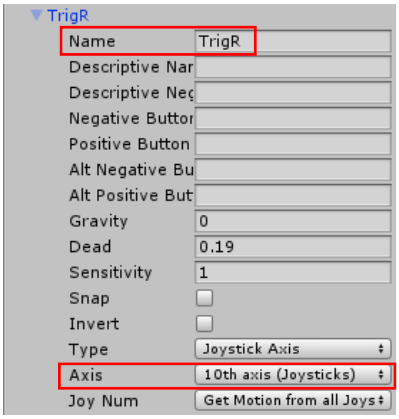
● 名称を TrigL とし、パラメータを設定します。

Invert を外すのを忘れないように。



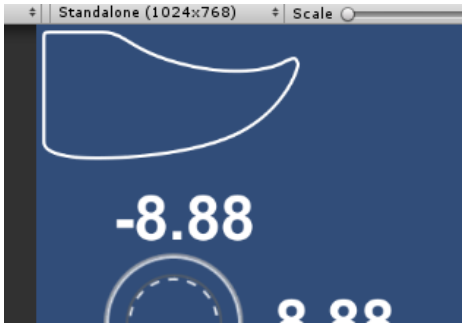
● 軸の数 Size を32にします。

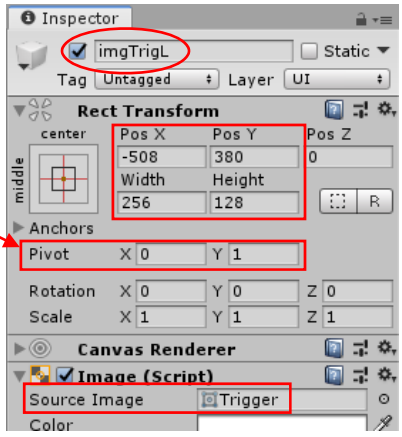
● 名称を TrigR にし、パラメータを設定します。



ヒエラルキー欄の imgJoyL を複製（Ctrl ＋ D）し、imgTrigL と命名します。インスペクタでパラメータを設定します。

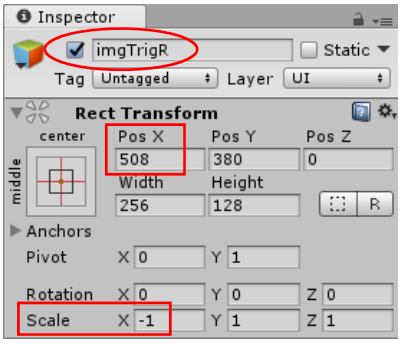
先に Pivot を設定します。



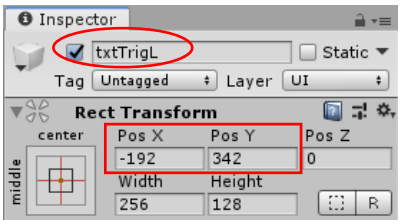


● この imgTrigL を複製（Ctrl ＋ D）し、imgTrigR と命名します。

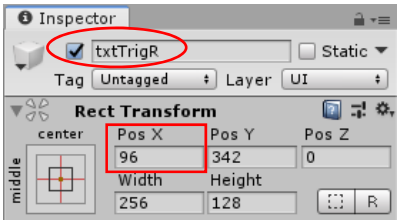
インスペクタでパラメータを設定します。



● ヒエラルキー欄の txtJoyLX を選択して複製（Ctrl ＋ D）し、名称を txtTrigL とします。
インスペクタでパラメータを設定します。



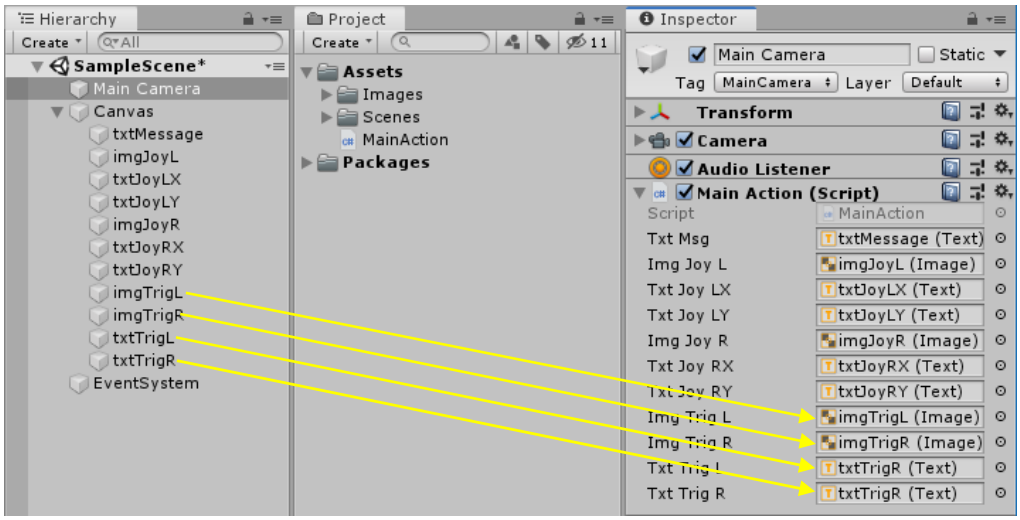
● この txtTrigL を選択して複製（Ctrl ＋ D）し、名称を txtTrigR とします。
インスペクタでパラメータを設定します。



● プログラム MainAction を以下のように修正します。

```
//～前略～
    public Image imgTrigL;
    public Image imgTrigR;
    public Text txtTrigL;
    public Text txtTrigR;

    void Update() {
        // 左右トリガー
        txtTrigL.text = Input.GetAxis("TrigL").ToString("f2");
        txtTrigR.text = Input.GetAxis("TrigR").ToString("f2");
        imgTrigL.rectTransform.rotation = new Quaternion(0, 0, Input.GetAxis("TrigL") * -0.3f, 1);
        imgTrigR.rectTransform.rotation = new Quaternion(0, 0, Input.GetAxis("TrigR") * 0.3f, 1);

//～後略～
```

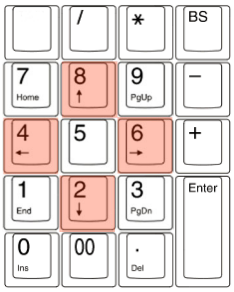
● ヒエラルキー欄の MainCamera を選択し、インスペクタに登場した項目を設定します。



● プレイボタンを押下します。 

トリガーを引いて、その様子が取得できていることを確認します。

【STEP11】 方向パッドの操作を取得

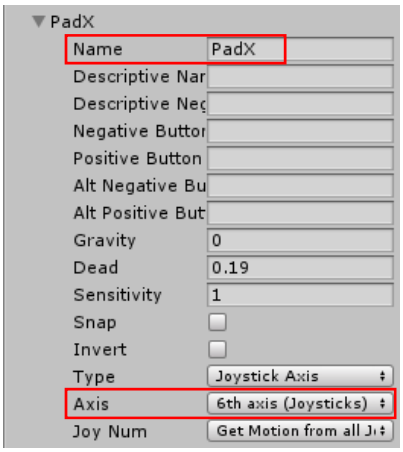


方向パッドはキーボードのテンキー・パッドでも代用されるように作るのが一般的です。(簡素なノートパソコンなどの場合はテンキー・パッドが付属していないこともあります。)

左右と上下の2方向に対応するインプット名を2個作れば終わりのハズが、パッドで2個、テンキー・パッドで2個の計4個を作ることになります。

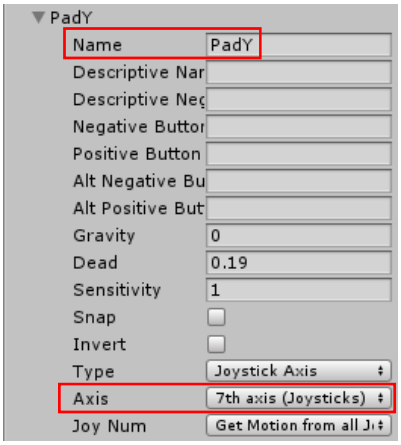
● インプット・マネージャーで軸の数 Size を 33 にします。

● 名称を PadX とし、インスペクタでパラメータを設定します。

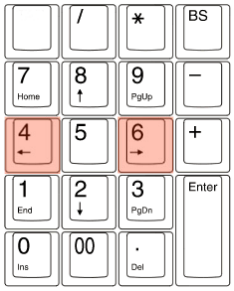


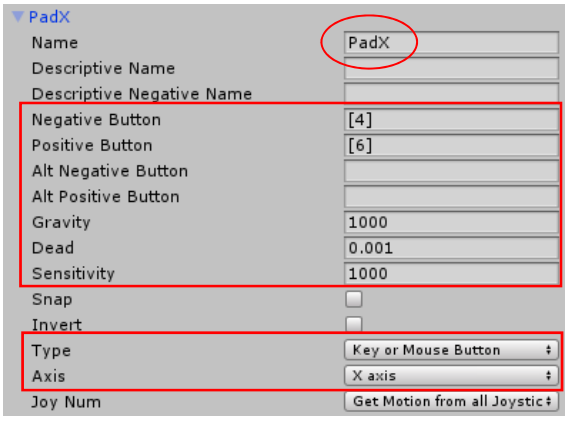
● インプット・マネージャーで軸の数 Size を 34 にします。

● 名称を PadY とし、インスペクタでパラメータを設定します。



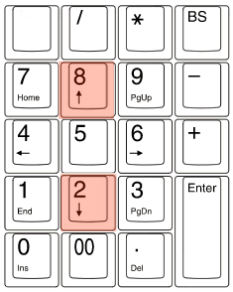
- インプット・マネージャーで軸の数 Size を 35 にします。
- テンキー・パッド向けです。再び、名称を PadX とし、インスペクタでパラメータを設定します。

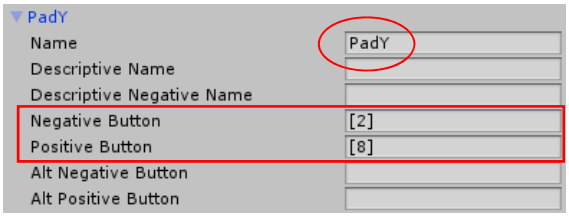




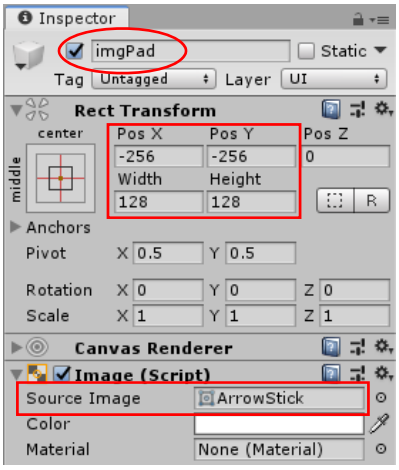
- インプット・マネージャーで軸の数 Size を 36 にします。（これで終わりです。）

- テンキー・パッド向けです。再び、名称を PadY とし、インスペクタでパラメータを設定します。

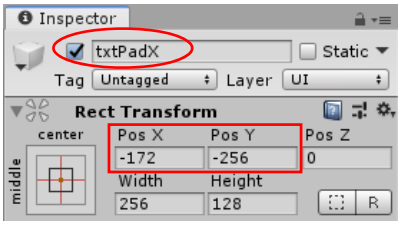




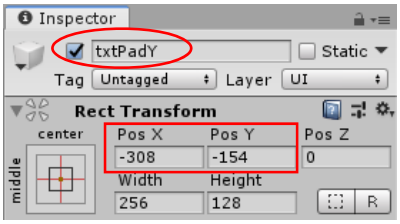
- ヒエラルキー欄の imgJoyL を複製（Ctrl + D）し、imgPad と命名します。
インスペクタでパラメータを設定します。



- ヒエラルキー欄の txtJoyRX を選択して複製（Ctrl + D）し、名称を txtPadX とします。
インスペクタでパラメータを設定します。



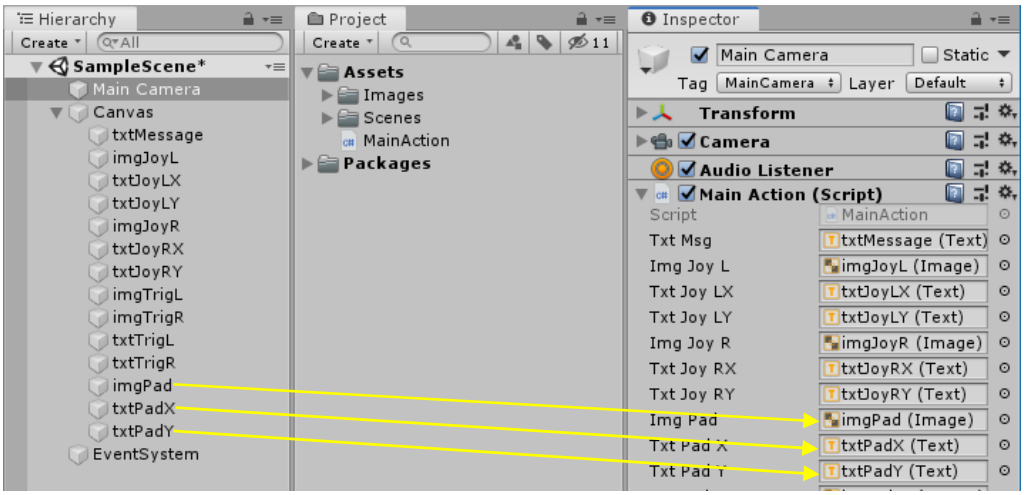
- この txtPadX を選択して複製（Ctrl + D）し、名称を txtPadY とします。
インスペクタでパラメータを設定します。



● プログラム MainAction を以下のように修正します。

```
//～前略～
    public Image imgPad;
    public Text txtPadX;
    public Text txtPadY;
    Vector2 PosPad; //方向パッドの初期位置

    void Start() {
        //初期メッセージの退避
        EscText = txtMsg.text;
        //方向パッドの初期位置を退避
        PosPad = new Vector2(
            imgPad.rectTransform.position.x,
            imgPad.rectTransform.position.y);
//～中略～

    void Update() {
        // 方向パッド
        float PX = Input.GetAxis("PadX");
        float PY = Input.GetAxis("PadY");
        txtPadX.text = PX.ToString("f2");
        txtPadY.text = PY.ToString("f2");
        imgPad.rectTransform.position = PosPad + new Vector2(PX, PY);
//～後略～
```

● ヒエラルキー欄の MainCamera を選択し、インスペクタに登場した項目を設定します。

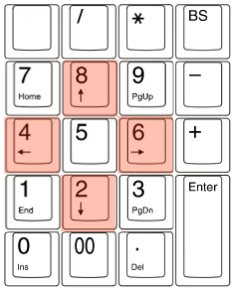


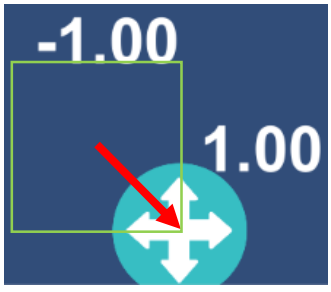
● プレイボタンを押下します。 

ゲームコントローラーの方向パッドの情報が取得できていることを確認します。斜め45度の方向に押した時に、不思議な値が出ていることも確認します。（注:テンキー・パッドの NumLock がオンになっていますか？）

ここで興味深いのは、キーボードのテンキー・パッドを操作した時と挙動が異なる事です。つまり、異なる値が取得されているので、プログラム次第では不具合の発生個所になる可能性を意識して扱う事です。

キーボードでの値を小さく修正するプログラムなどで応じるのが妥当な考え方でしょう。





【STEP12】 バイブレーションを発生させる

コントローラーの振動（バイブレーション）機能を実現する手法を幾つか検討した結果、実装が最も簡素だったので、今回、紹介する手順で挑みます。極端に言うと、今まで作って来た「インプットの手間」を最初から全て構築してある別の設定構成（アセット）があったのです。それを用いるものの、バイブレーション機能だけを使います。

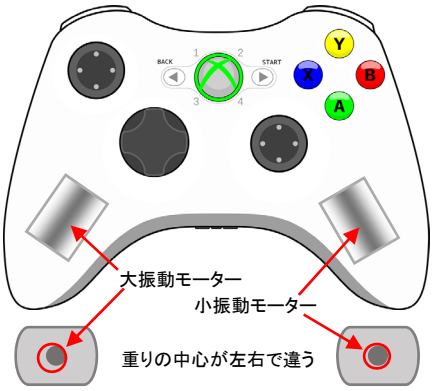
また、世界に向けて公開しているだけあって、4台までの Controller を接続しても、個々に認識できるなど、我々がここまでに作って来たものと比して、はるかに高機能です。（但し、理解や応用に時間と勉強量が必要ですヨ。）

バイブレーションの仕組み

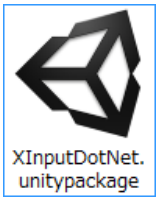
左右に2個のモーターが入っていて、モーターの軸の中心から少しずれた部位に重りが付いています。左右のモーターでは、その少しの差に違いがあり、左は大きく、右は小さく設定されています。

その結果、左は大きく振動し、右は小さく振動します。

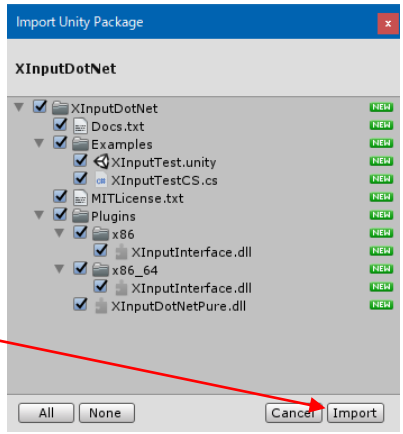
多くは両方に同じ値（1～0）を与えて、一気に同じ動作をさせるのですが、心臓の鼓動など複雑な印象を表現する際は、左右に別々の振動値を与えて操作すると言われています。



- 配布された素材を読み込みます。メニューAssets から Import Package > Custom Package と進み、配布した XInputDotNet.unitypackage を指定します。



内容物が表示されたら Import を押下します。



- スクリプト MainAction を以下のように修正します。

```
using XInputDotNetPure; // XInputDotNetの利用

public class myScript : MonoBehaviour {

// ～中略～

    float Elapsed = 0.0f;

    void Update () {
        Elapsed += Time.deltaTime;
        if (Elapsed < 0.3f) {
            //バイブレーション機能1
            GamePad.SetVibration((PlayerIndex) 0, 1, 1);
        } else {
            //バイブレーション機能2
            GamePad.SetVibration((PlayerIndex) 0,
                Input.GetAxis("TrigL"),
                Input.GetAxis("TrigR"));
        }
// ～後略～
```

まずは、起動時に1回だけ振動させます。
次に、本来の使い方ではないのですが、与える値によって振動の状況が変わることも理解して欲しかったので、無理やりにトリガーの引き具合に応じて振動量が変化するようにしました。

- プレイボタンを押下します。

起動時に1回だけ振動します。
以降は、トリガーの引き代（ひきしろ）に応じて、振動量が変化することを確認します。

以上