

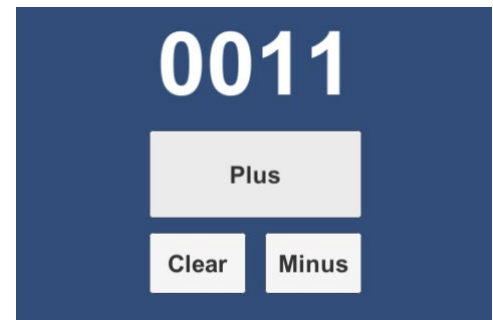


## カウンター (Counter)

プログラム制作に取り組む際に、まず、最初に作るのがハローワールドというプログラムではないか？と言われています。プログラムを実行したら、文字 Hello,World!が表示するだけの内容です。現に、多数のプログラミング修得書籍の冒頭でこれを紹介しています。

さらに踏み込むと、次は今回のカウンターが適しているのでは？と考えています。交通量調査などで用いられる、手押し式の計数機です。

1ずつ増えることとリセットしかできず、4桁までしか数えられないシンプル機能で構成されており、プログラミングの入門には最適な素材と考えています。

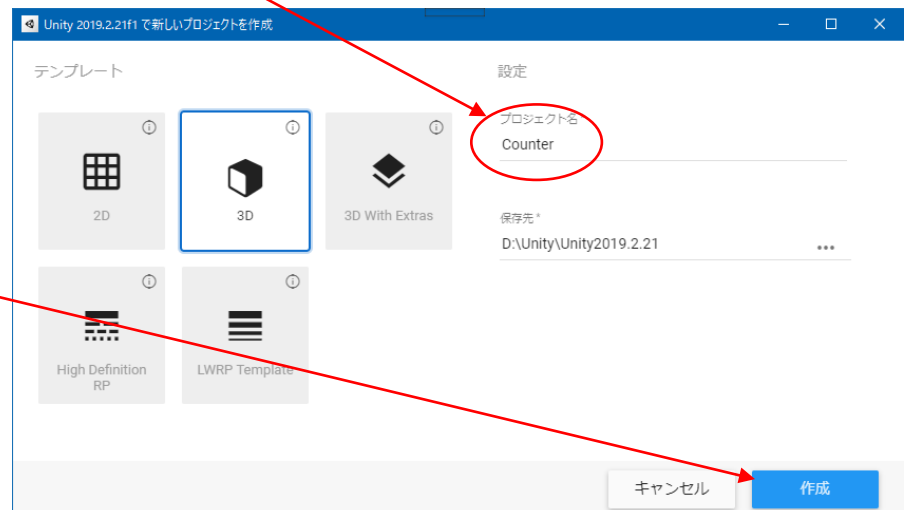


### 制作環境の整備

#### 【STEP1】プロジェクトの準備

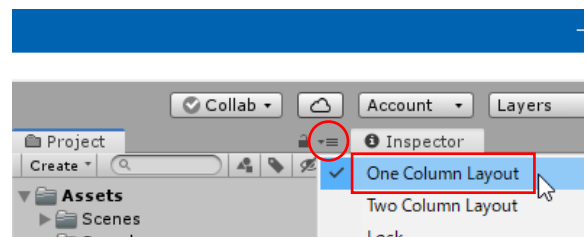
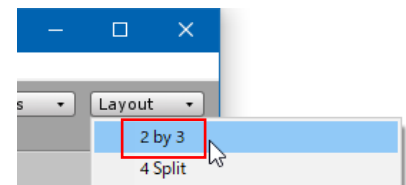
- Unity を起動し、プロジェクト Counter を作成します。

ボタン作成を押下します。

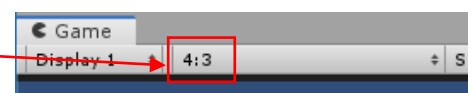


Unity が起動したら、画面のパネルレイアウトを確認します。

- 画面右上の Layout から 2 by 3 を選びます。
- プロジェクト欄のオプションから One Column Layout を選びます。



- Gameパネルのサイズを 4:3 にします。



- ヒエラルキー欄の **Main Camera** を選択します。  
インスペクタでパラメータを修正します。



## 【STEP2】 外部素材の読み込み

- 今回配布した**フォルダ Sounds** を Windows のエクスプローラーで表示し、Unity のプロジェクト欄にドラッグ & ドロップします。

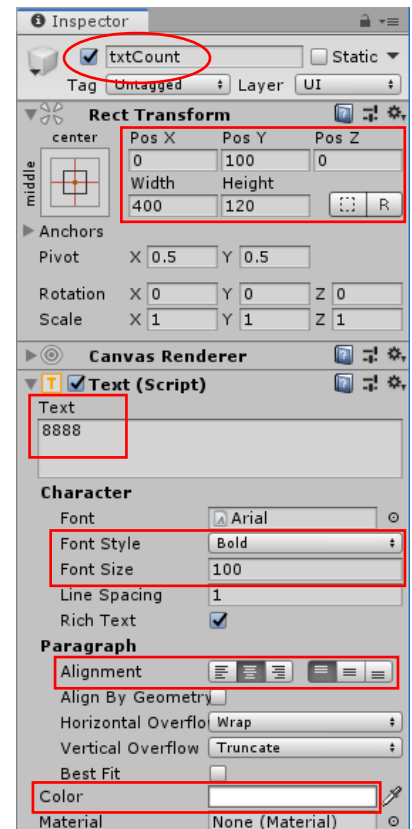
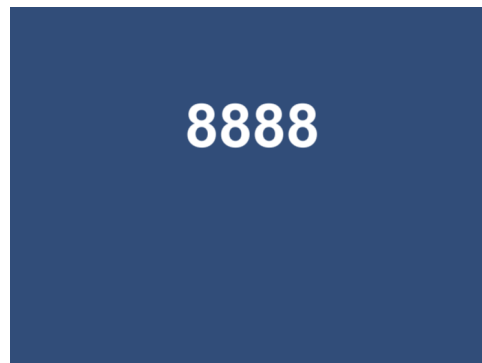
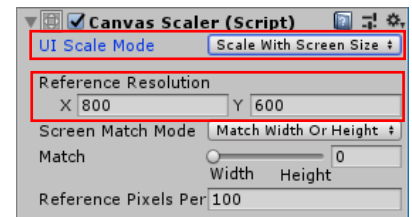
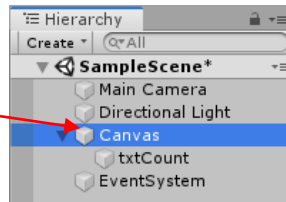
この様にプロジェクト内に取り込まれます。



## ユーザーインターフェースの構築

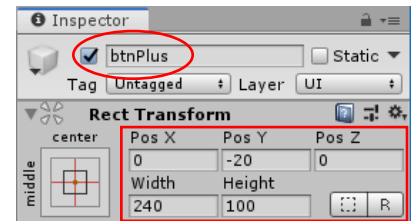
### 【STEP3】 文字の準備

- ヒエラルキー欄の Create から UI > Text を選択します。名称を **txtCount** と命名します。
- 先に、同時にヒエラルキー欄に作られた **Canvas** を選択します。  
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 **txtCount** を選択し、インスペクタでパラメータを設定します。

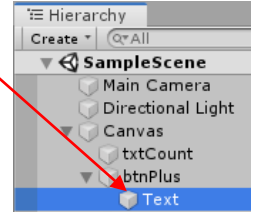


#### 【STEP4】 ボタンの準備

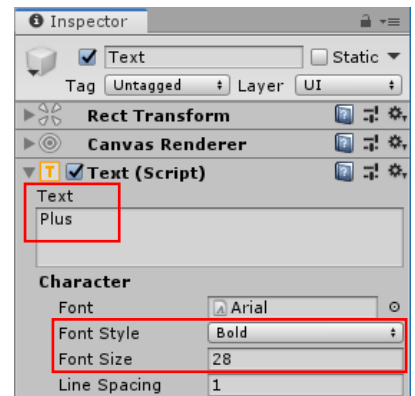
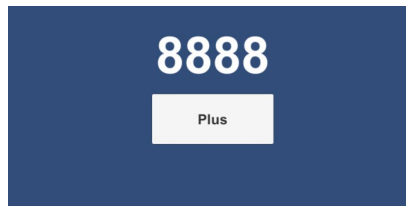
- ヒエラルキー欄の Create から UI > **Button** を選択し、名称を **btnPlus** とします。  
インスペクタでパラメータを設定します。



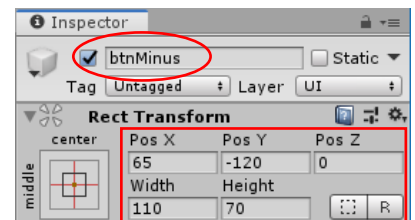
- この **btnPlus** の▼ボタンを押下して、配下構造を表示します。**Text** を選択します。



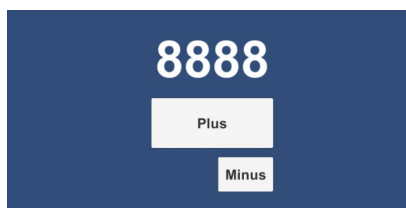
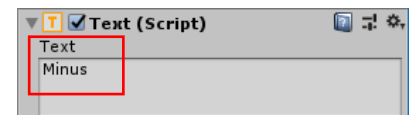
インスペクタでパラメータを設定します。



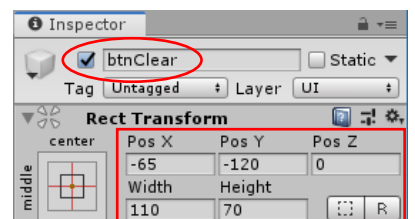
- この **btnPlus** を選択している状態で複製 (Ctrl + D) を行い、名称を **btnMinus** とします。  
インスペクタでパラメータを設定します。



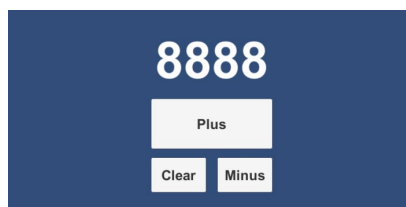
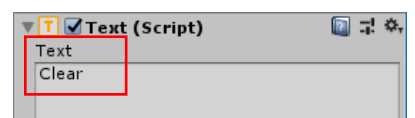
- 同様に、配下の **Text** のパラメータを設定します。



- この **btnMinus** を選択している状態で複製 (Ctrl + D) を行い、名称を **btnClear** とします。  
インスペクタでパラメータを設定します。



- 同様に、配下の **Text** のパラメータを設定します。



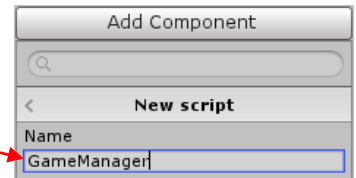
## プログラミングを行う

### 【STEP5】プログラムの準備

プログラムをどのゲームオブジェクトに取り付けるのか？は非常に重要な概念です。ゲーム全体を取りまとめるマネージャーを新規に作ってプログラムを持たせることもあります。

しかし、今回はどのオブジェクトに取り付けても、影響がないことから、新たにオブジェクトを作らずに、既存のゲームオブジェクト(ここではメインカメラ)に持たせることとして運営します。

- ヒエラルキー欄の **Main Camera** を選択し、インスペクタの Add Component から **New script** を選択し、名称を **GameManager** とします。



- スクリプト **GameManager** を次のように編集します。

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

public class GameManager : MonoBehaviour {

    void Start() {
        Debug.Log("Hello,Unity!");
    }

    void Update() {

    }

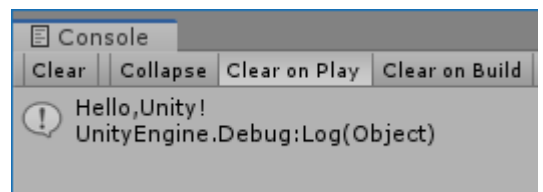
}
```

- プレイボタンを押下します。



画面左下の1行に、指定したメッセージが表示されているのが視認できます。

クリックするとコンソールというウィンドウが開きます。



### 【学習項目:Start 処理】

Start 関数や Start メソッドと呼ぶこともあり、プログラムが実行されたら最初に1度だけ行われる処理群を記述する欄となります。多くは様々な初期化や準備作業を事前に行っておく場所として捉えられています。

### 【学習項目:Debug.Log】

C言語を履修すると、もっと簡単な print 命令も出て来るのですが、Unity の C#でコンソールに情報を表示する場合は、この Debug.Log を用います。

## 【STEP6】 カウンターのプログラム

カウンターの処理機能は数を加算・減算し、クリアするだけですが、プログラムで考えると、もう少し手間が増えます。その数を数える領域を定義したり、計算結果を画面に表示したりする処理が必要です。

- スクリプト **GameManager** を次のように編集します。

数を数える領域を `int`(整数)型で定義しています。名称は `Cnt` です。

こういうデータの入れ物を変数と呼びます。

右辺を左辺に代入する記号が「`=`」となります。代入演算子といい、慣れ親しんだ等号記号:イコールの意味ではありません。

整数型のデータを文字列型の画面項目には直接送ることが出来ません。データ型を文字列型に `ToString()` で変換してから送ります。

加算では、右辺で計算した結果を左辺に代入しています。

3つのボタン処理は `public` になっています。画面のボタンから見えるように公開している訳です。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI; //uGUIを利用する際に必要

public class GameManager : MonoBehaviour {

    public Text txtCount; //画面のGUI文字
    int Cnt; //カウント領域

    void Start() {
        Debug.Log("Hello,Unity!");
        Cnt = 0; //カウント領域をゼロクリア
        txtCount.text = Cnt.ToString(); //画面に転記する
    }

    //クリアボタンを押した処理
    public void PushClear() {
        Cnt = 0; //カウント領域をゼロクリア
        txtCount.text = Cnt.ToString(); //画面に転記する
    }

    //プラスボタンを押した処理
    public void PushPlus() {
        Cnt = Cnt + 1; //カウント領域を1加算
        txtCount.text = Cnt.ToString(); //画面に転記する
    }

    //マイナスボタンを押した処理
    public void PushMinus() {

        //【命題】
    }

    //～後略～
}
```

### 【学習項目:演算記号】

加算方法について取り上げると、以下の3通りは同じことをしています。

- `Cnt = Cnt + 1;` : 上記のパターン。右辺を演算した結果を左辺に代入する。
- `Cnt += 1;` : 代入と演算を両方行う記号で、複合代入演算子という。
- `Cnt ++;` : 1ずつ増える処理を簡素化したもの。インクリメントという。

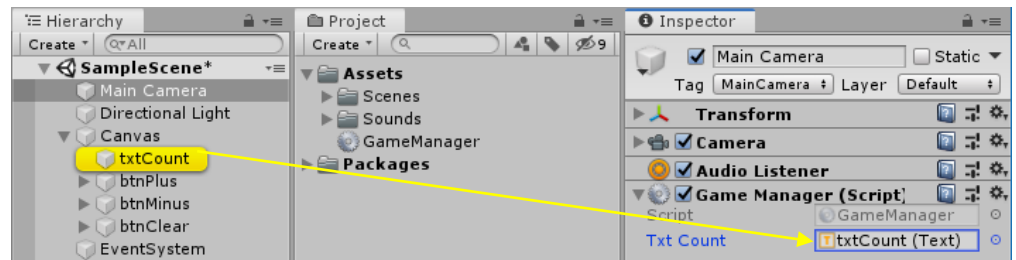
### 【命題】

マイナスボタンを押下した時の処理が抜けています。加算処理を参考にして記述してください。

## 画面のUI部品とプログラムの接続

### 【STEP4】 UI部品との接続①

- ヒエラルキー欄の **Main Camera** を選択し、インスペクタに登場した項目にヒエラルキー欄の **txtCount** をドラッグ&ドロップします。

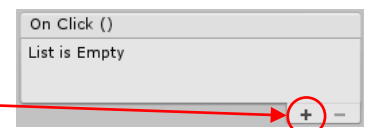


#### 【学習項目:パブリック】

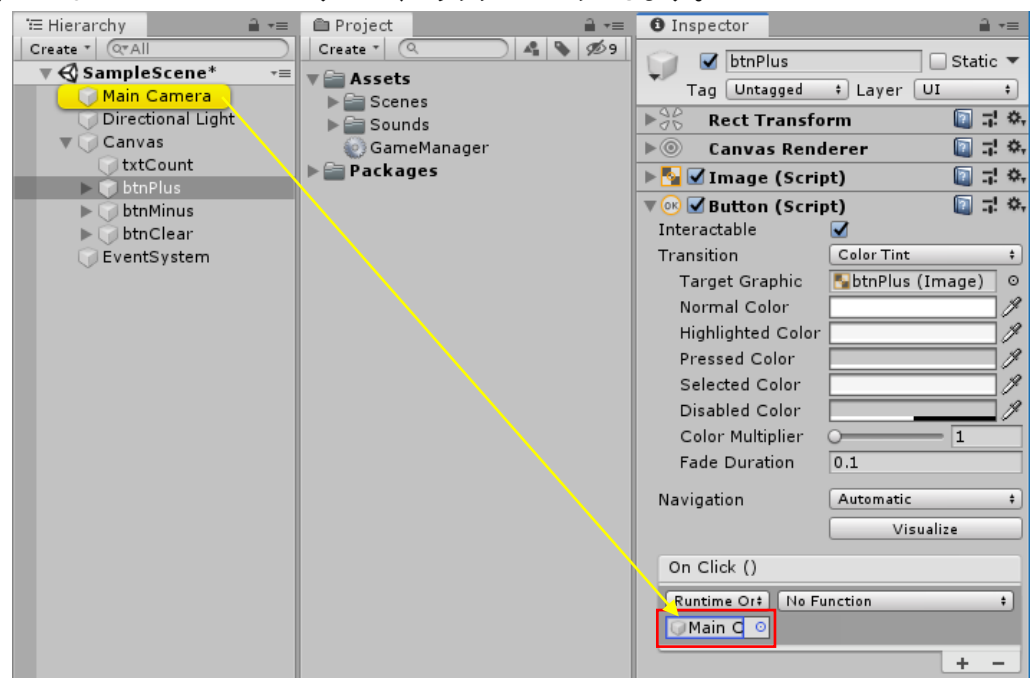
本格的に説明すると非常に複雑な概念です。ただ、今回の捉え方としては、public で定義した変数はインスペクタに登場することで、我々がデータを直接与えることが出来るのだ、と把握しておいて良いでしょう。

### 【STEP5】 UI部品との接続②

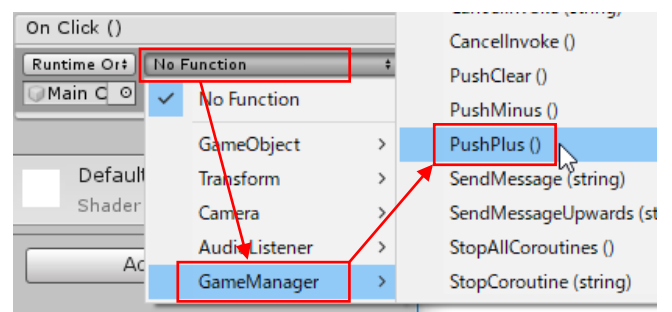
- ヒエラルキー欄の btnPlus を選択し、インスペクタの On Click()にある**プラスボタン(+)**を押下します。



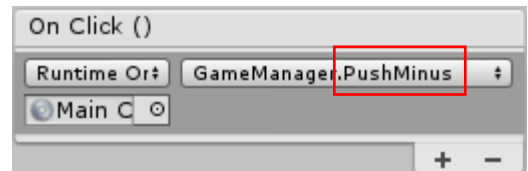
- 動作させたい処理を持つのは **Main Camera** ですので、ドラッグ&ドロップします。



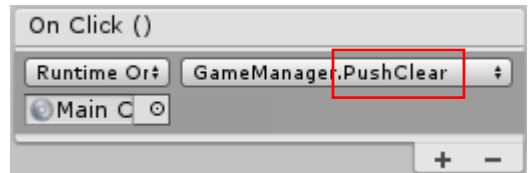
- 動作させたい処理が No Function になっているので、GameManager > **PushPlus()**を指定します。



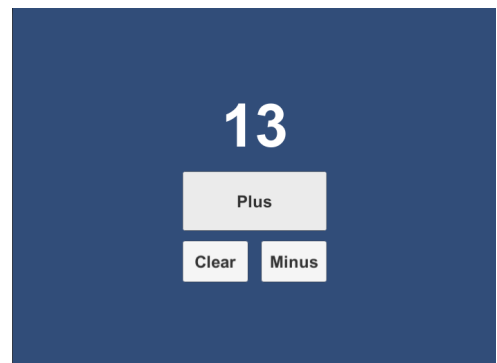
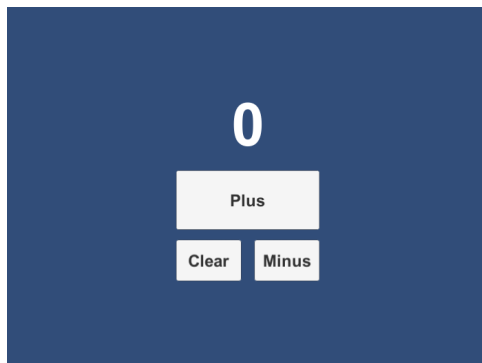
- 同様の作業を **btnMinus** にも行います。  
動作させる処理は **PushMinus()**となります。



- 同様の作業を **btnClear** にも行います。  
動作させる処理は **PushClear()**となります。



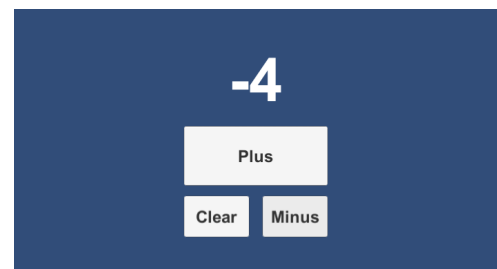
- プレイボタンを押下します。   
実際にプラスボタン、マイナスボタン、クリアボタンを押下して、カウンターを運用して下さい。



## 表示の改善

### 【STEP6】 表示の改善(マイナス対応)

クリアボタン押下後にマイナスボタンを押下すると、マイナス表記になってしまいます。これは防ぐべきです。




- スクリプト **GameManager** を次のように編集します。

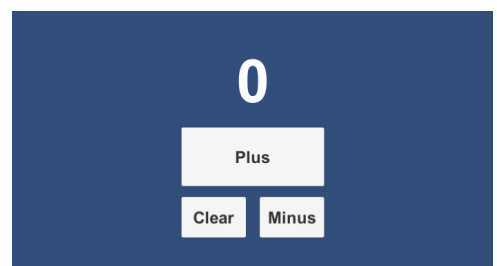
条件分岐を行う if 文を用いています。  
小括弧内の条件式を満たす(正しい)場合、直後の中括弧内の処理群(複数可)を実行します。

```
//～前略～

//マイナスボタンを押した処理
public void PushMinus() {
    Cnt = Cnt - 1; //カウント領域を1減算
    if (Cnt < 0) {
        Cnt = 0;
    }
    txtCount.text = Cnt.ToString(); //画面に転記する
}

//～後略～
```

- プレイボタンを押下します。   
ゼロ以下のマイナス表記にならないことを確認します。



## 【STEP7】 表示の改善(4桁固定対応)

実際のカウンターのように、4桁を固定で表示します。

- スクリプト **GameManager** を次のように編集します。

トータルで4桁になるように、不足する場合は左に詰め物(パッド)を入れる指示となります。  
詰め物は文字 0 としています。

```
//～前略～


void Start() {
    Debug.Log("Hello,Unity!");
    Cnt = 0; //カウント領域をゼロクリア
    txtCount.text = Cnt.ToString().PadLeft(4,'0'); //画面に転記する
}

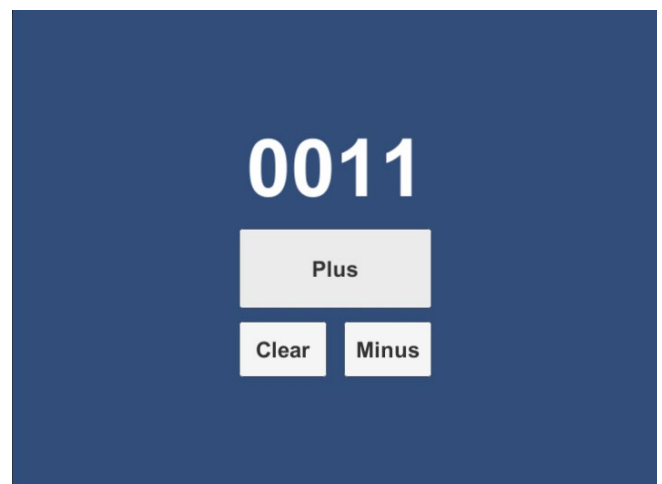
//クリアボタンを押した処理
public void PushClear() {
    Cnt = 0; //カウント領域をゼロクリア
    txtCount.text = Cnt.ToString().PadLeft(4,'0'); //画面に転記する
}

//プラスボタンを押した処理
public void PushPlus() {
    Cnt = Cnt + 1; //カウント領域を1加算
    txtCount.text = Cnt.ToString().PadLeft( 4, '0' ); //画面に転記する
}

//マイナスボタンを押した処理
public void PushMinus() {
    Cnt = Cnt - 1; //カウント領域を1減算
    if (Cnt < 0) {
        Cnt = 0;
    }
    txtCount.text = Cnt.ToString().PadLeft( 4, '0' ); //画面に転記する
}

//～後略～
```

- プレイボタンを押下します。 4桁で表示することが出来ます。





## 【STEP8】記述の効率化①

前述の作業は、同じ修正を複数個所に行いました。これは4カ所だったので、まだ作業も軽微でしたが、非常に数多くの個所であった場合、手間もそうですが、修正ミスなどが懸念されます。

こういう場合、処理は1か所に1度だけ記述しておき、実行したいタイミングでその処理を行えば、非常に効率的でシンプルになり、修正ミスも減ります。

- スクリプト **GameManager** を次のように編集します。

```
//～前略～

void Start() {
    Debug.Log("Hello,Unity!");
    Cnt = 0; //カウント領域をゼロクリア
    DisplayCount(); //カウント表示
}

//クリアボタンを押した処理
public void PushClear() {
    Cnt = 0; //カウント領域をゼロクリア
    DisplayCount(); //カウント表示
}

//プラスボタンを押した処理
public void PushPlus() {
    Cnt = Cnt + 1; //カウント領域を1加算
    DisplayCount(); //カウント表示
}

//マイナスボタンを押した処理
public void PushMinus() {
    Cnt = Cnt - 1; //カウント領域を1減算
    if (Cnt < 0) {
        Cnt = 0;
    }
    DisplayCount(); //カウント表示
}

//カウント表示
void DisplayCount() {
    txtCount.text = Cnt.ToString().PadLeft( 4, '0' ); //画面に転記する
}

//～後略～
```

- プレイボタンを押下します。



修正によって機能が失われていないか？を確認します。

結果は同じですが、効率化した記述で同じ処理が実現できています。

## 【STEP9】 記述の効率化②

開始時の処理に着眼します。2行の処理はクリアボタンを押下した時の処理と同じであることが判ります。

2行なので影響が低いようにも感じますが、この処理が大量にあった場合を想定すると、ここでも2度手間な記述は避けることが望ましいでしょう。


- スクリプト **GameManager** を次のように編集します。

```
//～前略～

void Start() {
    Debug.Log("Hello,Unity!");
    Cnt = 0; //カウント領域をゼロクリア
    DisplayCount(); //カウント表示
}

//クリアボタンを押した処理
public void PushClear() {
    Cnt = 0; //カウント領域をゼロクリア
    DisplayCount(); //カウント表示
}

//～後略～
```

- プレイボタンを押下します。 

修正によって機能が失われていないか？を確認します。  
結果は同じですが、効率化した記述で同じ処理が実現できています。

```
//～前略～

void Start() {
    Debug.Log("Hello,Unity!");
    PushClear(); //クリア処理
}

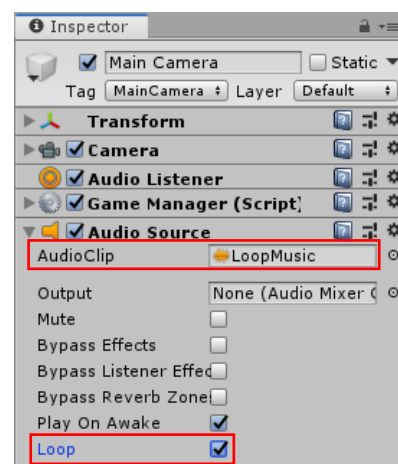
//クリアボタンを押した処理
public void PushClear() {
    Cnt = 0; //カウント領域をゼロクリア
    DisplayCount(); //カウント表示
}


//～後略～
```

## サウンド機能の実装

### 【STEP10】 サウンド鳴動①（BGM編）

- ヒエラルキー欄の **Main Camera** を選択します。  
インスペクタの Add Component から Audio > **AudioSource** を選択します。  
インスペクタでパラメータを設定します。



- プレイボタンを押下します。 

BGMがループ再生で鳴動し続けることを確認します。  
もちろん、ボタン押下で停止・再生、音量をコントロールすることも可能です。今回は割愛しています。

## 【STEP11】 サウンド鳴動②（タイミング編）

加算と減算のタイミングでサウンドを鳴動します。

```
//～前略～

public AudioClip SE_Plus; //加算サウンド
public AudioClip SE_Minus; //減算サウンド
AudioSource MyAudio; //自身の音源

void Start() {
    MyAudio = GetComponent<AudioSource>(); //自身の音源を取得
    Debug.Log("Hello,Unity!");
    PushClear(); //クリア処理
}

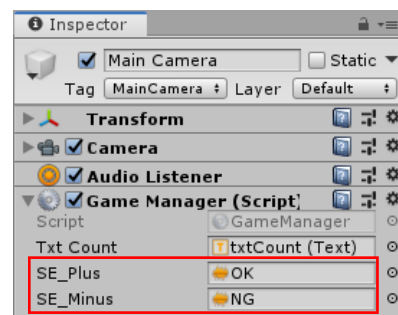
//クリアボタンを押した処理
public void PushClear() {
    Cnt = 0; //カウント領域をゼロクリア
    DisplayCount(); //カウント表示
}

//プラスボタンを押した処理
public void PushPlus() {
    Cnt = Cnt + 1; //カウント領域を1加算
    DisplayCount(); //カウント表示
    MyAudio.PlayOneShot( SE_Plus ); //加算サウンド鳴動
}

//マイナスボタンを押した処理
public void PushMinus() {
    Cnt = Cnt - 1; //カウント領域を1減算
    if (Cnt < 0) {
        Cnt = 0;
    }
    DisplayCount(); //カウント表示
    MyAudio.PlayOneShot( SE_Minus ); //減算サウンド鳴動
}

//～後略～
```

- ヒエラルキー欄の **Main Camera** を選択し、インスペクタに登場した項目に音声ファイルを指定します。



- プレイボタンを押下します。  プラスボタンとマイナスボタンの押下で、サウンドが鳴動することを確認します。

以上