

# Unity講座 ～ブロック崩し～

3年 海苔 威

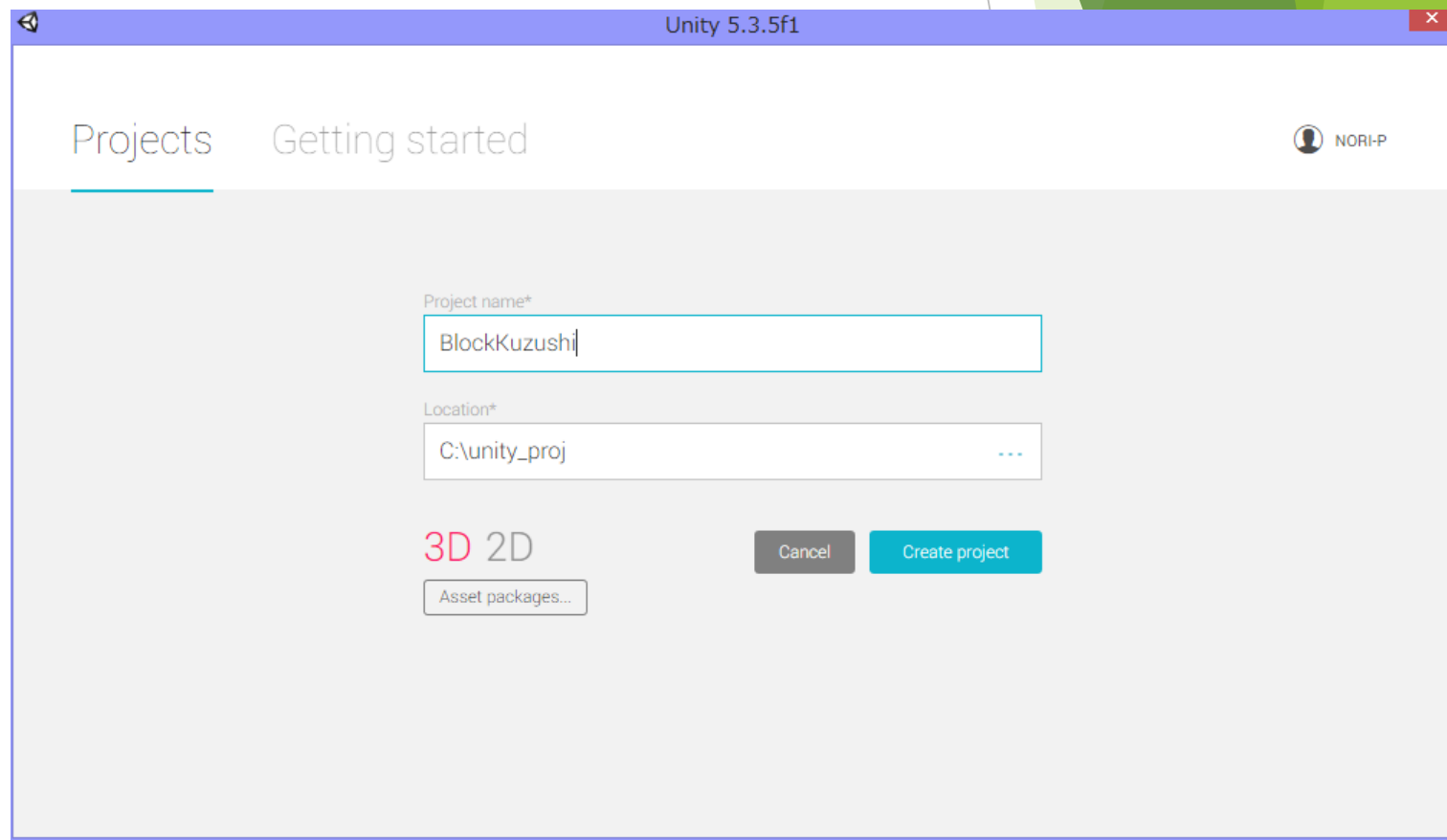
# 1. 初めに

# ざっくり説明

- ▶ Unityではゲーム全体構成する1つ1つのシーンを作っていきます
- ▶ 例えば、タイトル、キャラ選択、バトル、結果etc... がそれぞれ1つのシーン
- ▶ ボタンごとの動作の設定やシーンの遷移とかをプログラミングしていく
- ▶ まずは「タイトル画面」を作ろう

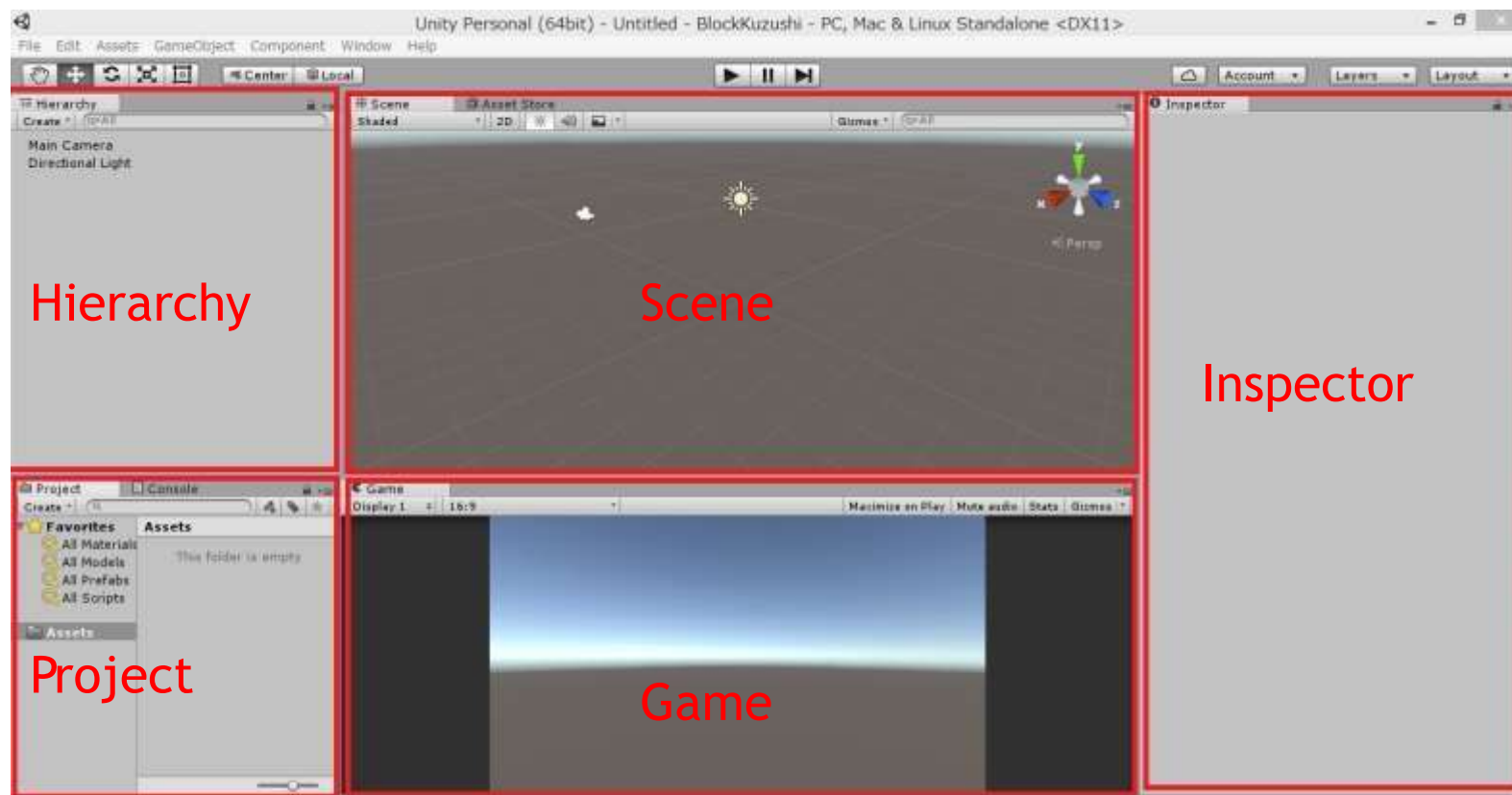
# プロジェクト作成

- ▶ Unityを起動して、真ん中のNew Projectか右上のNewをクリック
- ▶ 右の画面が出るので
  - ①Project name
  - ②Location
  - ③「3D」か「2D」をそれぞれ決める
- ①、②は自由。③は「3D」で！
- ▶ Create Projectをクリック



# 画面の見方

- ▶ 画面はこんな感じ

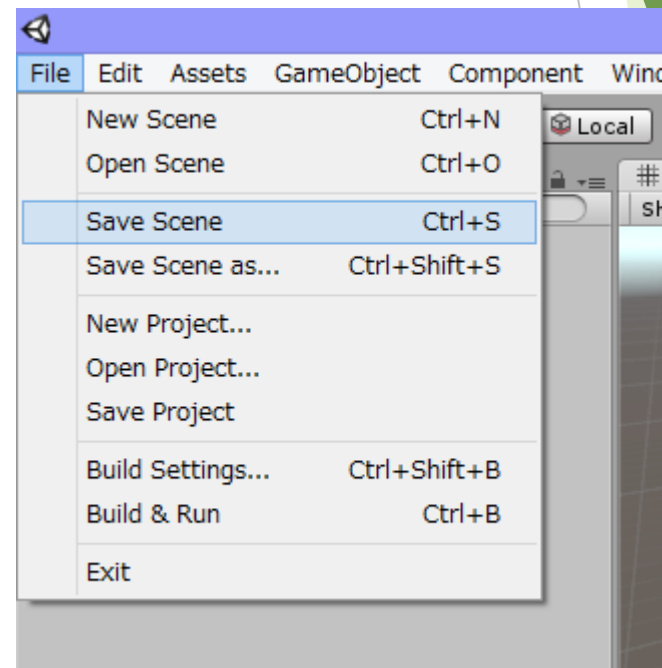


- ▶ Hierarchy  
Sceneビューにあるゲームオブジェクトを表示
- ▶ Project  
保存先にある「Assets」フォルダの中身を表示
- ▶ Scene  
ゲームオブジェクトを配置する画面
- ▶ Game  
実際にゲームを実行した時の画面
- ▶ Inspector  
ゲームオブジェクトのステータスを表示

## 2. タイトル画面の作成

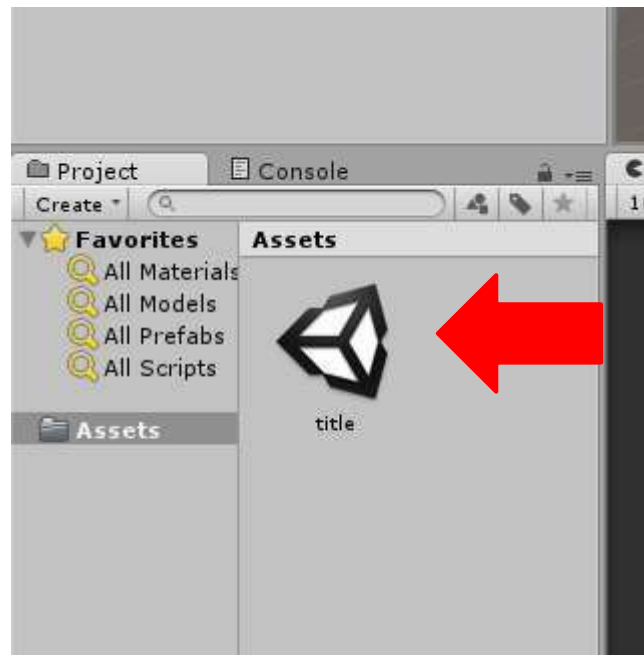
## 2-1. シーンの保存

- ▶ まずは「タイトル」シーンとして保存しよう
- ▶ 「File」 → 「Save Scene」で保存（ctrl + Sで良いよ）
- ▶ 名前は「title」で



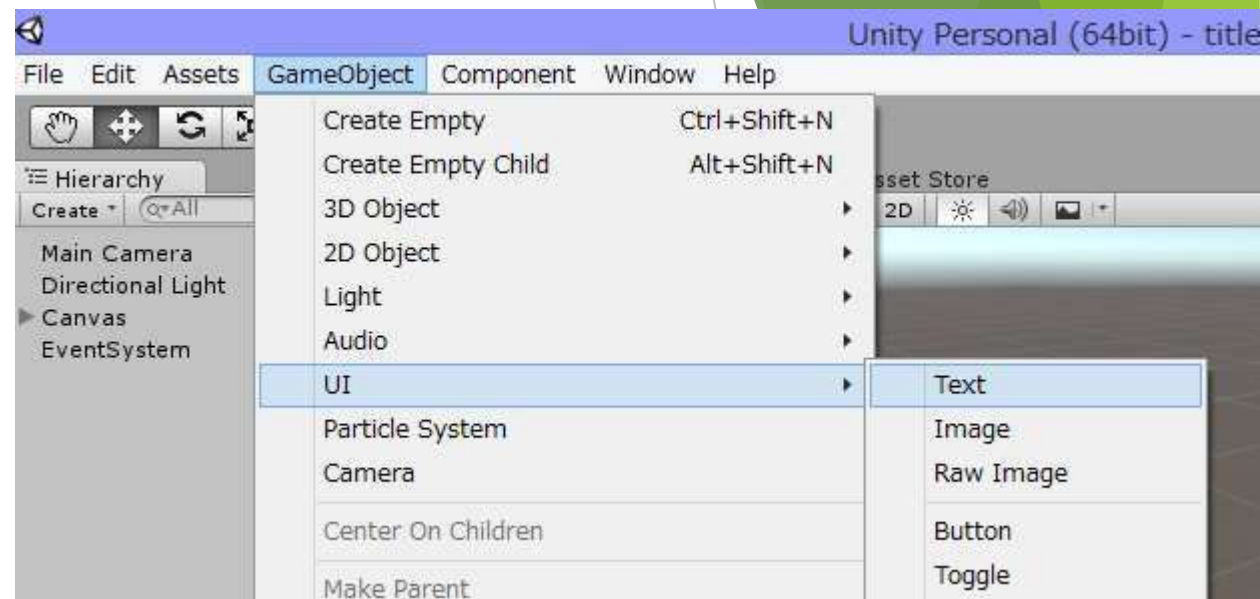


- ▶ Projectの「Assets」をみると「title」が追加されているはず
- ▶ こんな感じでシーンを作ると「Assets」に保存され表示されます

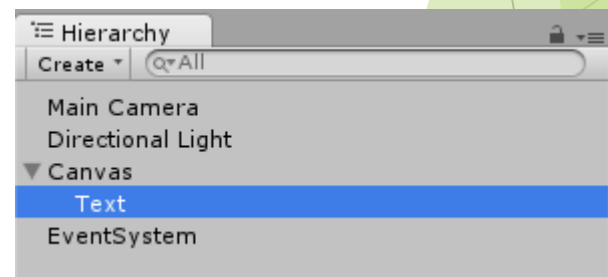


## 2-2. タイトルシーンの作成

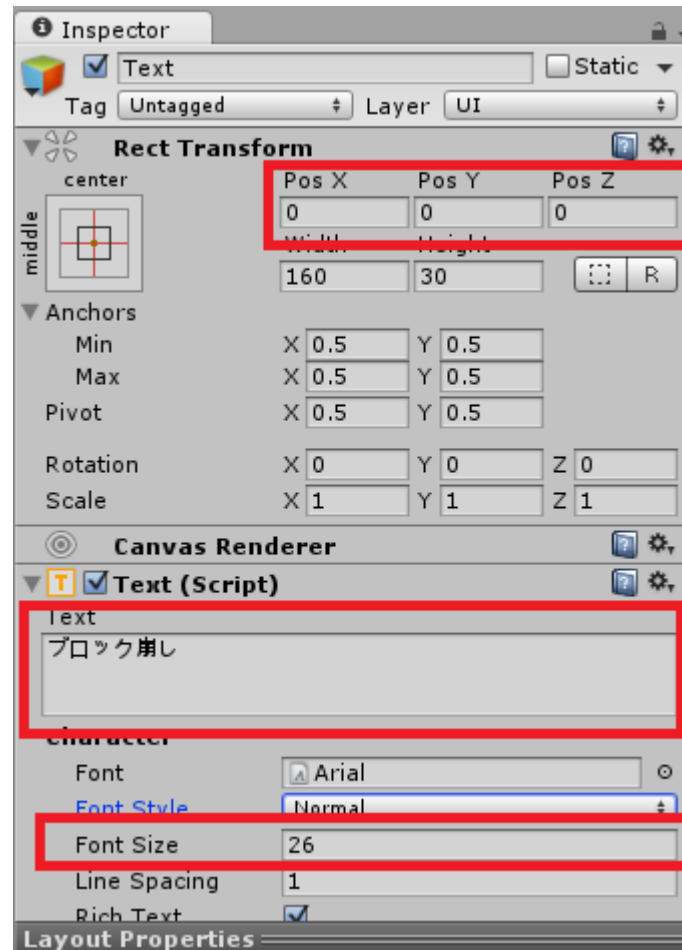
- ▶ 上のメニュー画面から「Game Object」 → 「UI」  
→ 「Text」を選択



- ▶ Hierarchyの「Canvas」内にある「Text」をクリック
- ▶ Inspectorを見る



- ▶ 右みたいな画面になるはず
- ▶ 赤枠の値に変えてください
  
- ▶ Pos X, Yはそれぞれ文字を表示する位置の左上の座標を示します
- ▶ 文字列は2次元で扱うのでZは無関係
  
- ▶ Textは表示する文字列
  
- ▶ Font Sizeは文字の大きさ



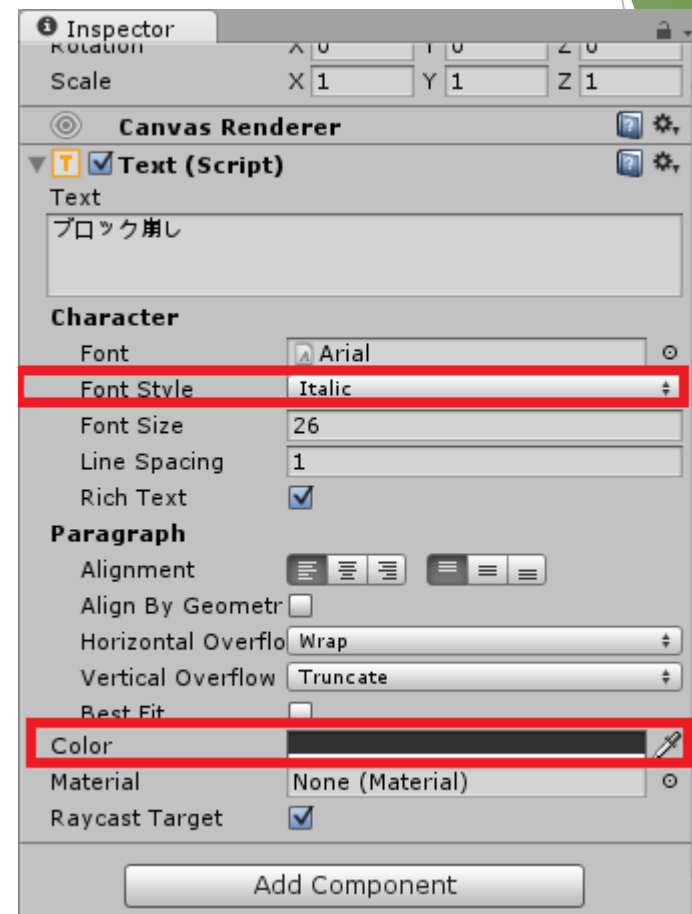
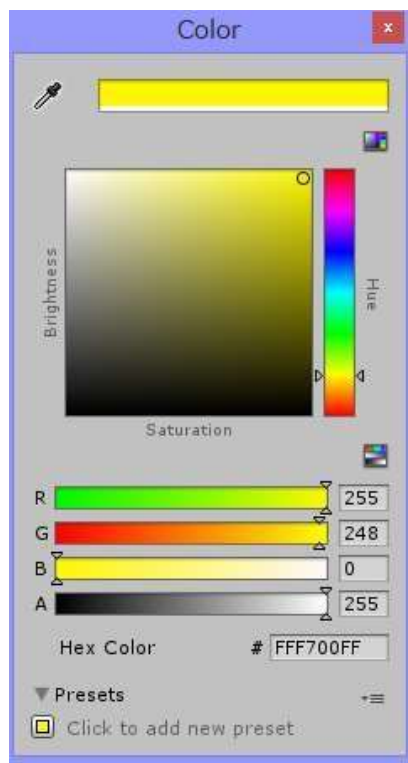
- ▶ 前ページの設定を行ったらGameビューを見てください
- ▶ こんな感じのはず



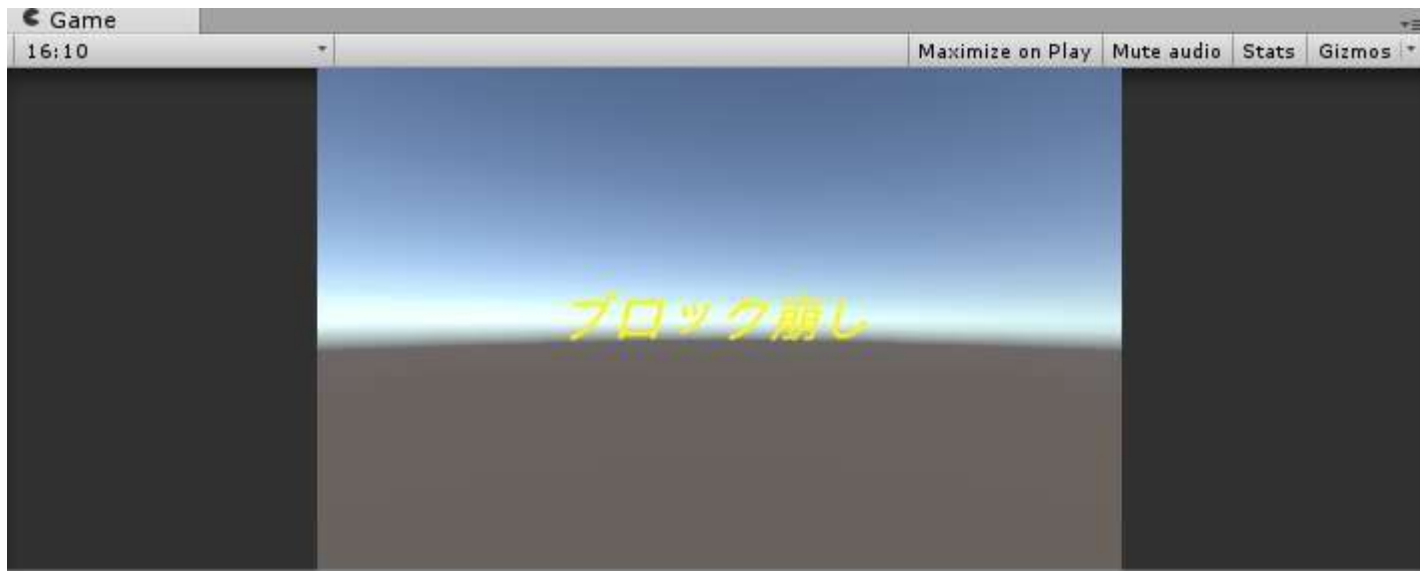
- ▶ 続けていじっていきます

- ▶ 同じくInspectorの下の方をみてください
- ▶ 赤枠をいじる
- ▶ Font Styleはイタリックとボールドが選べる  
お好きにどうぞ

- ▶ Colorを選ぶと→  
こんなのが出る  
ので各自自由に  
色を選ぶ



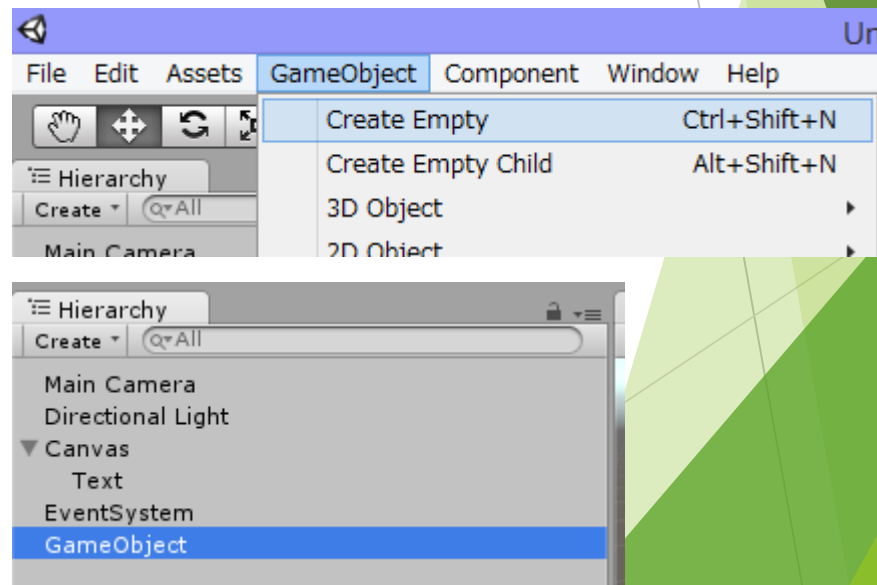
- ▶ 「Italic」で「黄色」を選んだのでGameビューはこんな感じ



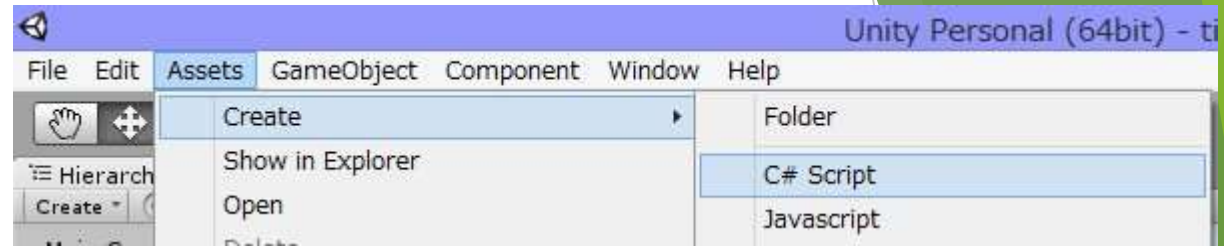
- ▶ それぞれ設定したようになっていけばおけ
- ▶ 次に進みます

## 2-3. スタートボタンの作成

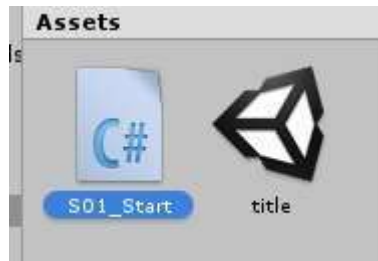
- ▶ ゲーム画面に進むためのスタートボタンを作ります
- ▶ プログラムでボタンを作成します
- ▶ メニューの「Game Object」 → 「Create Empty」
- ▶ Hierarchyに追加されているのを確認



- ▶ ではプログラムを書いていきます
- ▶ メニューの「Assets」 → 「Create」  
→ 「C# Script」



- ▶ Assets内のC#ファイルの名前を変える（選択してF2を押すと楽）  
名前は「S01\_Start」

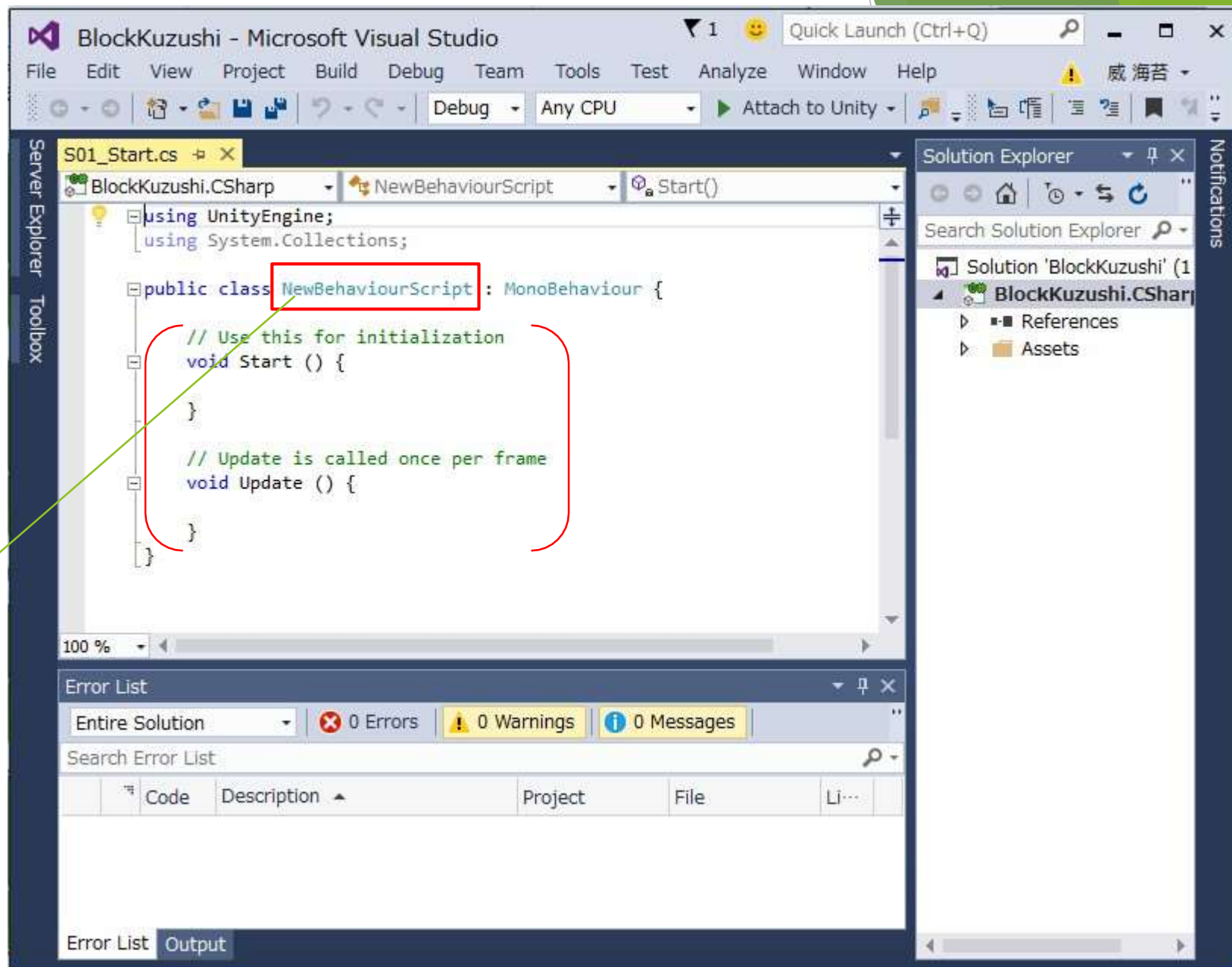


- ▶ 出来たらこのファイルをダブルクリック
- ▶ たぶんVisual Studio2015が開く



- ▶ こんな (→) 画面
- ▶ 既にいろいろ書いてある！
- ▶ なんだこれ！
- ▶ というわけでまず赤い()の中身を消します

ここが「S01\_Start」じゃない人は「S01\_Start」に変更



▶ 書き直し！

！

```
hello.cs | S01_Start.cs | X
BlockKuzushi.CSharp | S01_Start | OnGUI()
using UnityEngine;
using System.Collections;

public class S01_Start : MonoBehaviour {
    void OnGUI()
    {
        GUI.Button(new Rect(20, 20, 100, 20), "スタートボタン");
    }
}
```

▶ 今回使うのはこれだけ

意味は

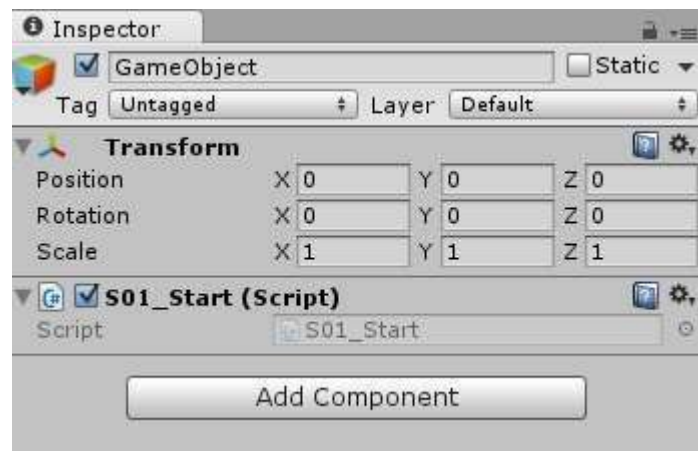
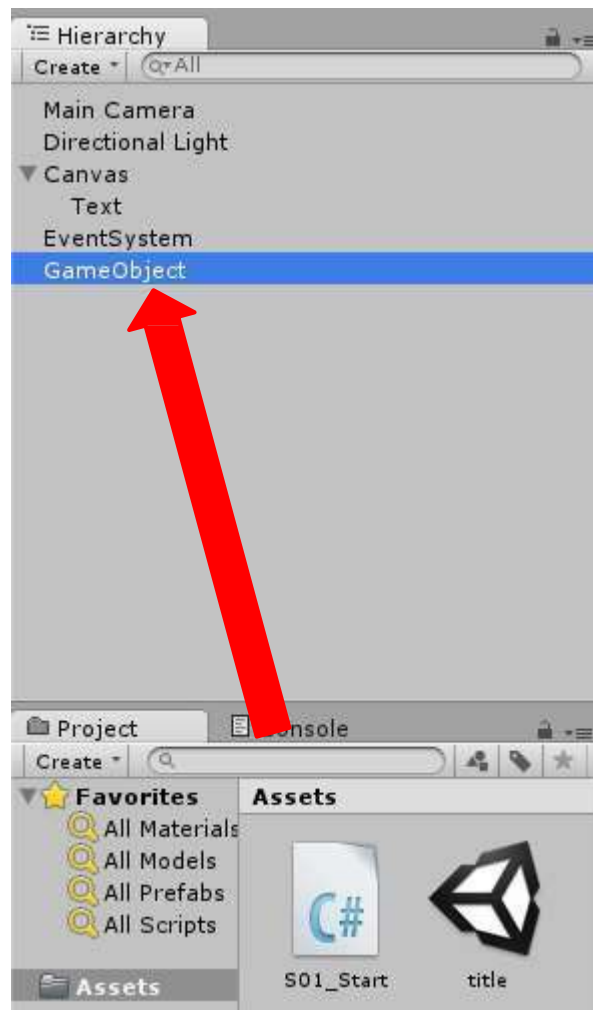
GUI.Button → ボタン作るよ

new Rect(20, 20, 100, 20) → 左上x, y座標(20, 20)から幅100高さ20の四角を作るよ

“スタートボタン” → 作った四角の中にこの文字を書くよ

▶ 書いたら保存

- ▶ Unity画面に戻り、作った「S01\_Start」を「Game Object」にドラッグ&ドロップ
- ▶ Inspectorにスクリプトが追加されるはず
- ▶ 「Game Object」が「S01\_Start」を持つことで、記述したプログラムがシーン上で動くようになります



- ▶ 上にある再Thボタンみたいなやつをクリック



- ▶ こうなっているとき、ゲーム画面が実際のゲームと同様の動きをします
- ▶ というわけでGameビューを確認



- ▶ ボタンが表示されていれば勝ち
- ▶ 押せるけど、押したときの処理を何も書いていないので何も起きません

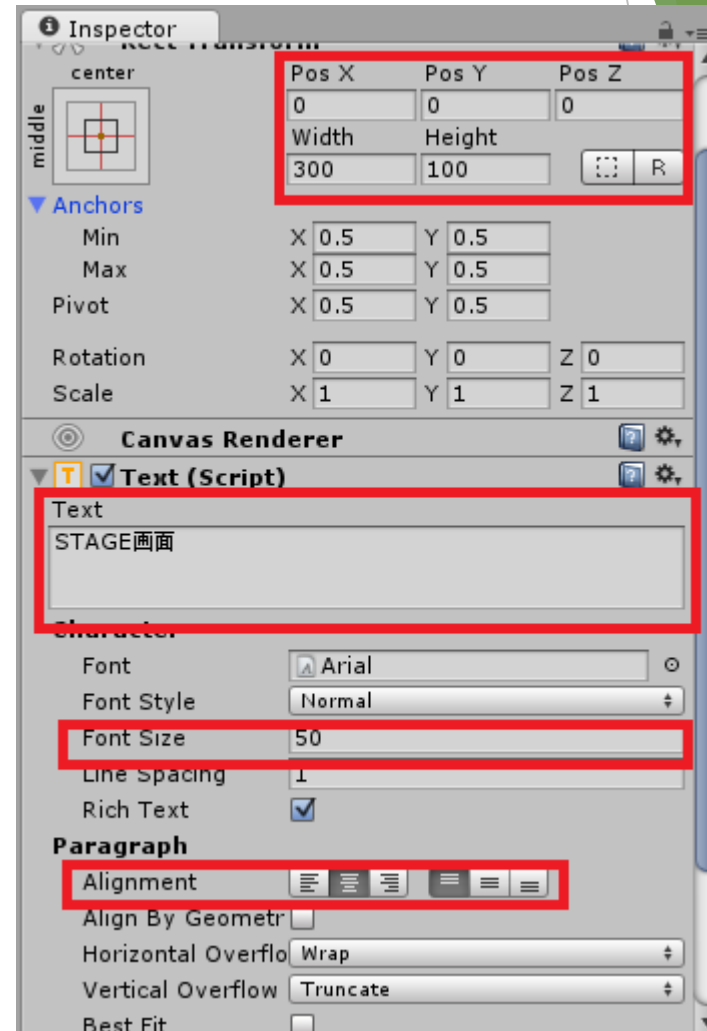
# 3. シーン遷移

## 3-1. ステージ画面（仮）の作成

- ▶ 先ほどまでのtitleシーンを上書き保存してください  
(再Thボタンをもう1回押すとゲーム実行画面から戻ります)
- ▶ メニューから「File」→「New Scene」で新規シーンを作成
- ▶ 作成したらまず保存。名前は「stage」
- ▶ さっきまでのオブジェクトは?????  
という人はAssetsのtitleをダブルクリック  
titleシーンが呼べます



- ▶ Titleと同じようにメニューから「Game Object」 → 「UI」 → 「Text」
- ▶ Inspectorを見て、赤枠と同じに
- ▶ 色とスタイルは自由
- ▶ Gameビューに「STAGE画面」と出ていればおけ
- ▶ 大丈夫な人はシーンを保存



## 3-2. シーン遷移の作り方

- ▶ Assetsからtitleを開く
- ▶ シーン遷移のための準備は2つ
  - ①ゲームビルドにシーンを入れる
  - ②遷移のためのプログラムを書く



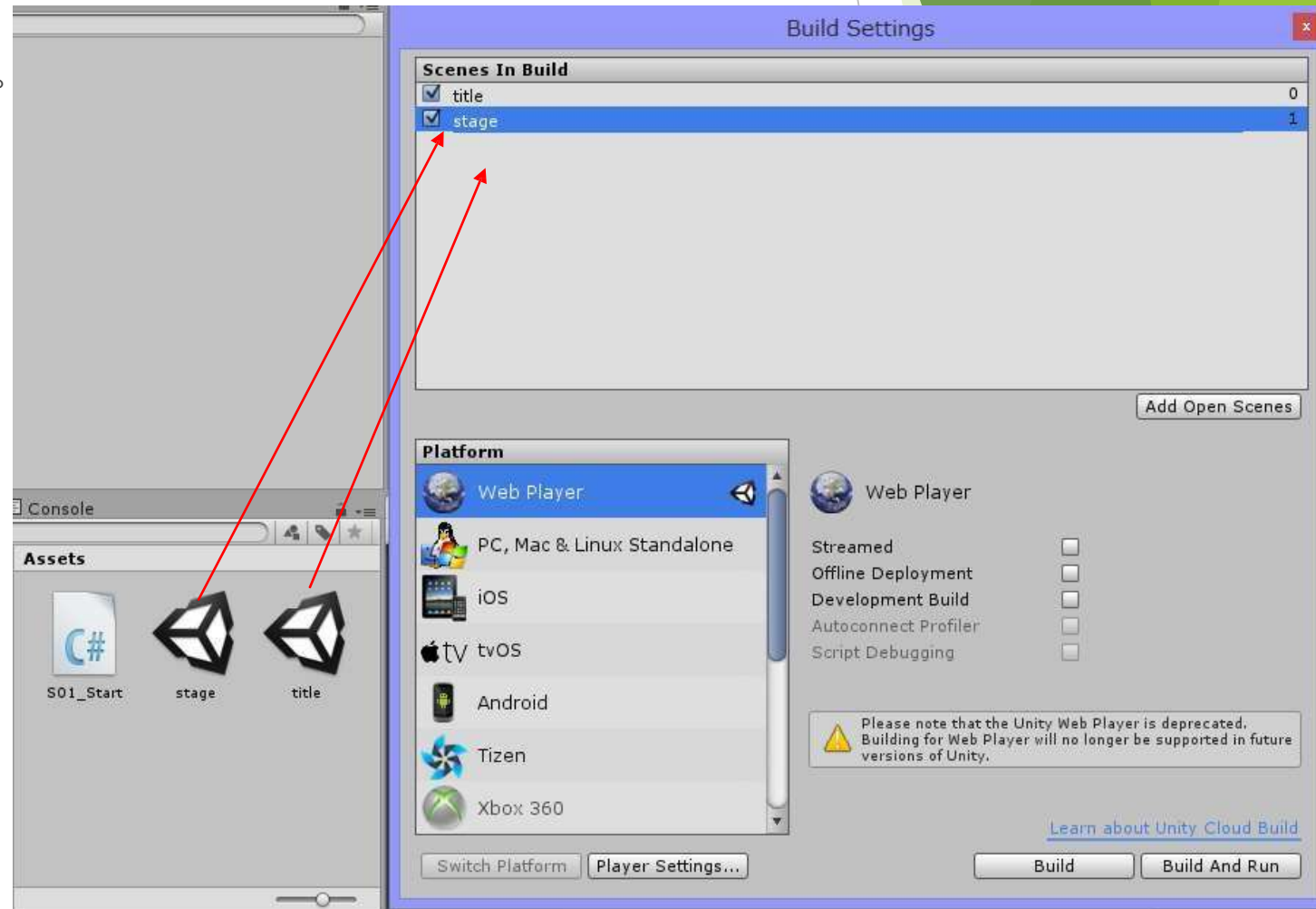
- ▶ まず①
- ▶ 「File」 → 「Build & Settings」 を選択すると↓の画面がでる

- ▶ Assetsにある「title」「stage」を「Scene In Build」にドラッグ&ドロップ

- ▶ titleが上になるように
- ▶ 順番は動かさず

- ▶ Build Settingsはそのまま閉じて

- ▶ ①終了



- ▶ 次は②
- ▶ 「S01\_Start」を開く
- ▶ 以下の部分を変更

```
using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;

public class S01_Start : MonoBehaviour {

    void OnGUI()
    {
        if (GUI.Button(new Rect(Screen.width - 120, Screen.height - 40, 100, 20), "スタートボタン"))
        {
            SceneManager.LoadScene("stage");
        }
    }
}
```

- ▶ 上書き保存

- ▶ ②終了

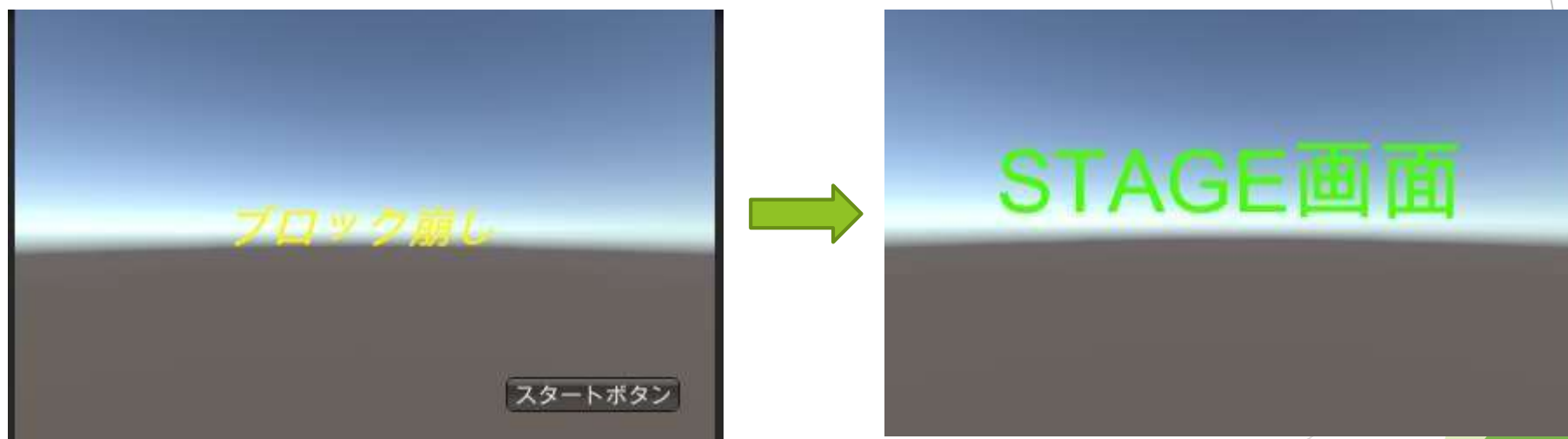
▶ 意味

UnityEngine... → シーン遷移の命令を使えるようにする

SceneManager... → 「stage」というシーンに遷移する

if文の中でGUIボタンを作ってます。ボタンは押された状態がtrueとなるので、これで作成したボタンが押されたときに実行！ということになります。

▶ 再Thボタンで実行

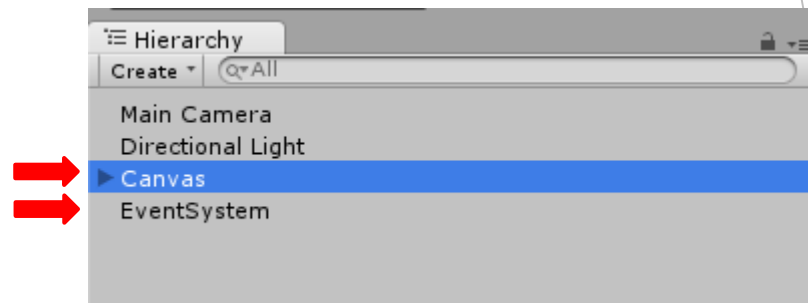


▶ 遷移しましたか？

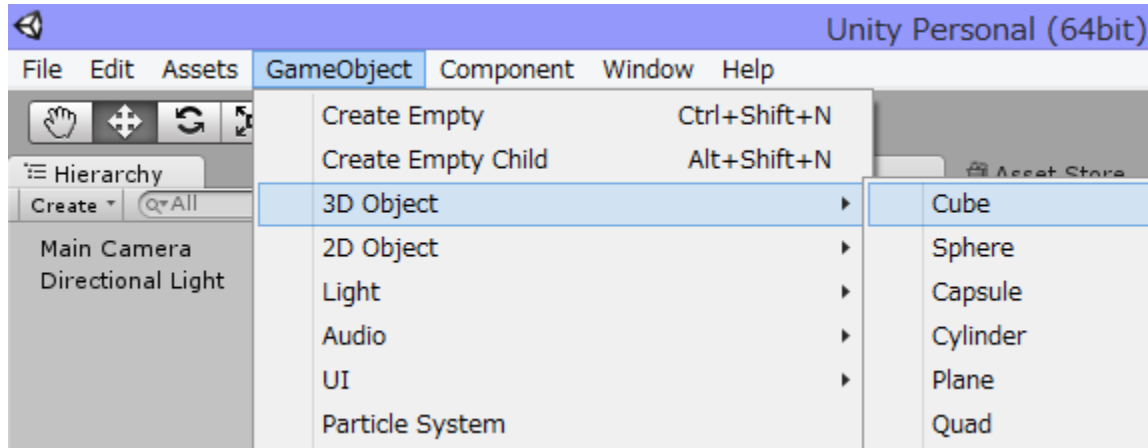
## 4. ステージ画面の作成

## 4-1. 壁の作成

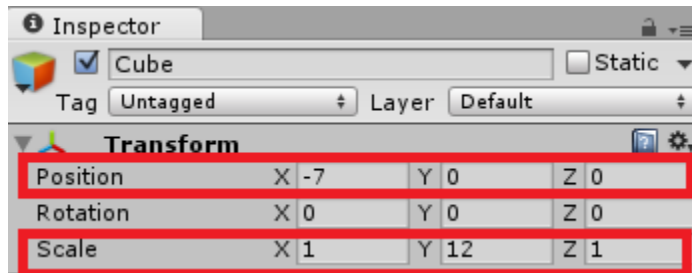
- ▶ Assetsのstageをダブルクリックで開く
- ▶ 作ったテキストはもういらないので削除→
- ▶ 右クリックでdelete  
選択した状態でdelキー、backspaceキーでも
- ▶ ブロック崩しに必要な壁とボールを作ります



- ▶ メニューの「Game Object」 → 「3D Object」 → 「Cube」をTh成

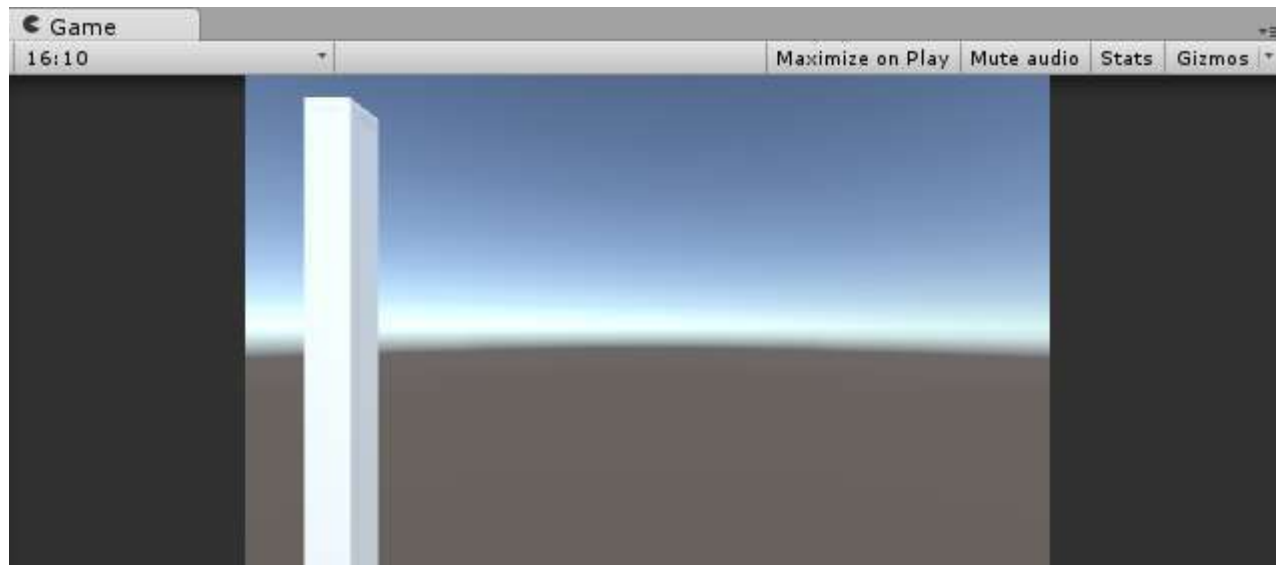


- ▶ 作ったCubeのInspectorを開く

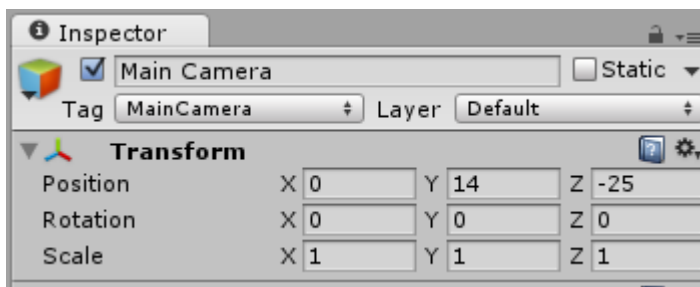


- ▶ 赤枠を変更
- ▶ Cubeの名前を「Wall\_left」に変更

▶ Gameビューがこんな感じになるはず



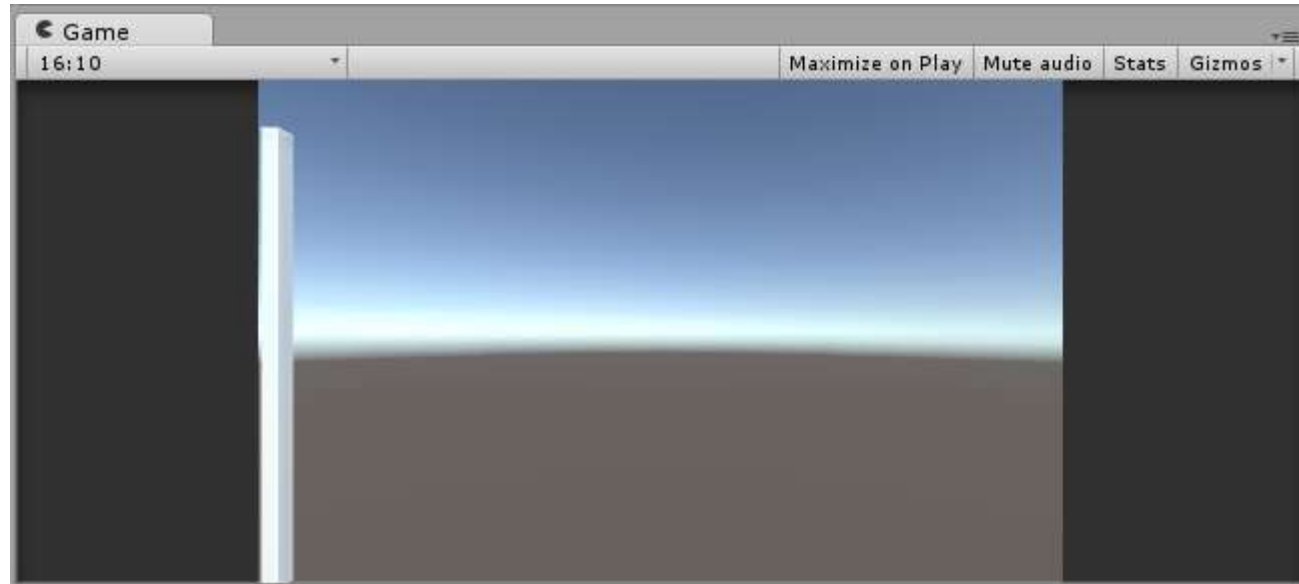
- ▶ このサイズで壁を作ると小さいのでスケールを大きくします
- ▶ まずカメラの位置を移動
- ▶ Hierarchyの「Main Camera」のInspectorを開き、x, y, zの位置だけ変更



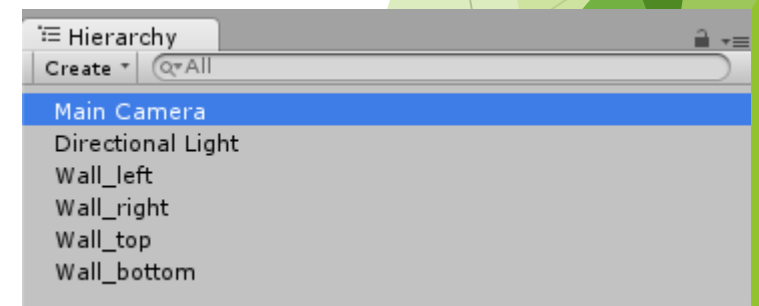
- ▶ 次に「Wall\_left」のPosition(x, y, z) = (-22, 12.5, 0)

Scale(x, y, z) = (1, 26, 1)

に変更

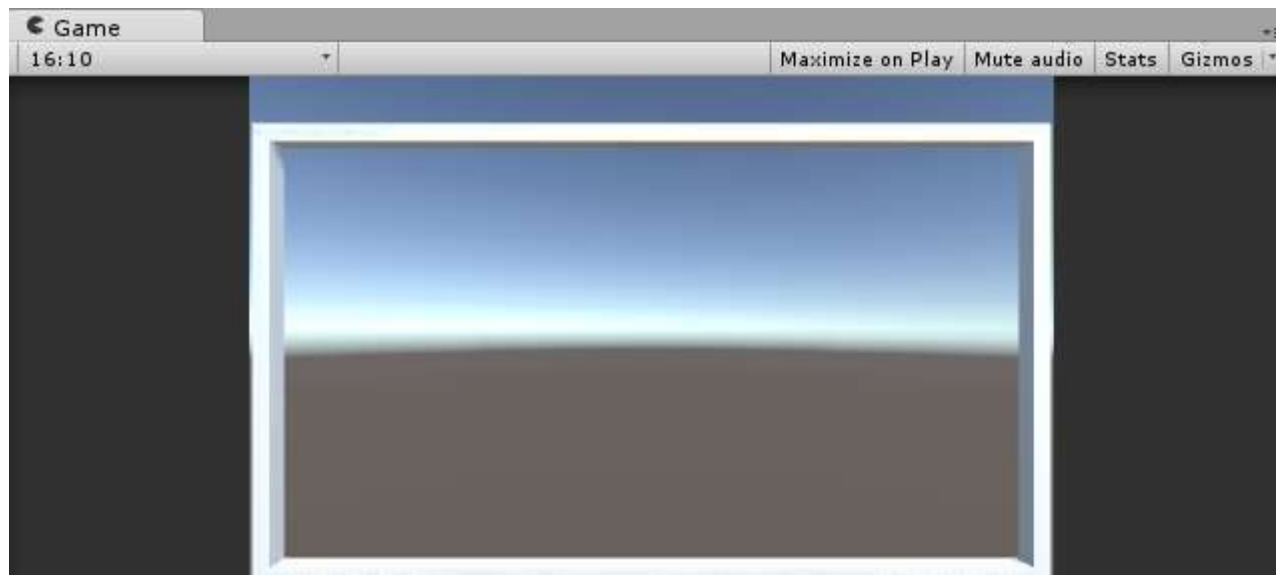


- ▶ 見た感じ大きな壁になってそう
- ▶ 残りの上、右、下も壁を作ります
- ▶ 「Wall\_left」をコピーしてHierarchyに「Wall\_top」「Wall\_right」「Wall\_bottom」をそれぞれ作ってください



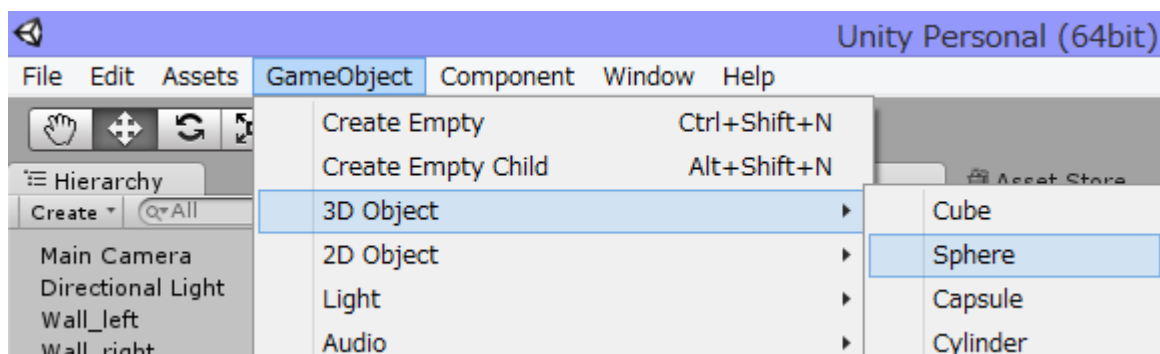


- ▶ Wall\_leftと同様に3つも数値を変えます
- ▶ 「Wall\_top」のPosition(x, y, z) = (0, 25, 0)  
Scale(x, y, z) = (44, 1, 1)
- ▶ 「Wall\_right」のPosition(x, y, z) = (22, 12.5, 0)  
Scale(x, y, z) = (1, 26, 1)
- ▶ 「Wall\_bottom」のPosition(x, y, z) = (0, 0, 0)  
Scale(x, y, z) = (44, 1, 1)
  
- ▶ ここまでできたらGameビューを見てみるとこんな感じ
- ▶ 壁が出来ました
- ▶ 次はボール



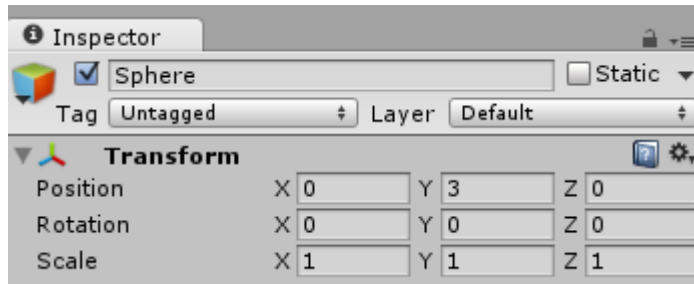
## 4-2. ボールの作成

- ▶ 次はボール。最初の壁と同じように「Game Object」 → 「3D Object」 → 「Sphere」

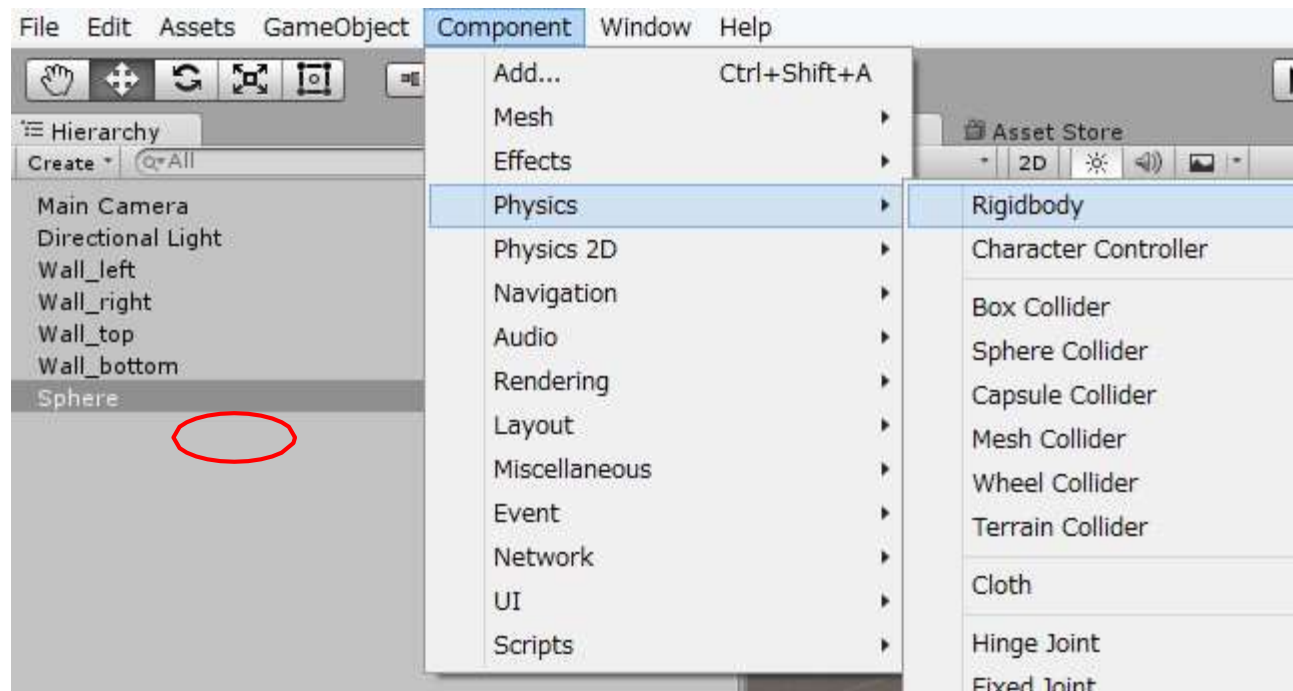


- ▶ Sphereの名前を「Ball」に変更

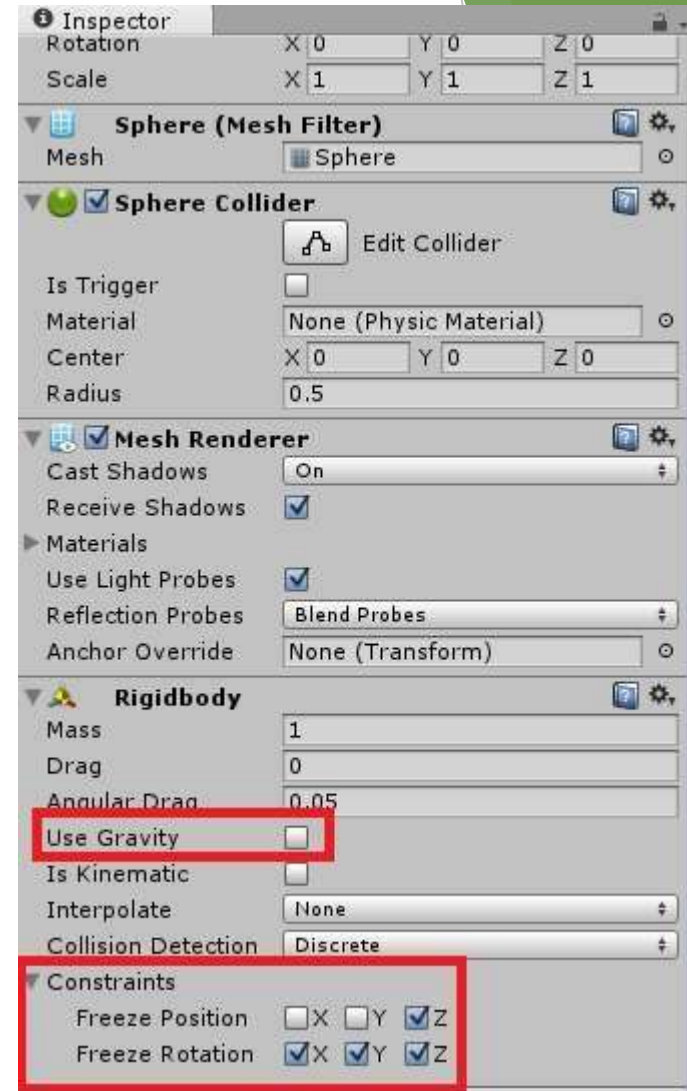
- ▶ BallのInspectorを開き、赤枠を変更



- ▶ これだと(0, 3, 0)にあるだけの球なので動くようにしよう
- ▶ Hierarchyで「Ball」を選択した状態でメニュー「Component」→「Physics」  
→「Rigidbody」を追加



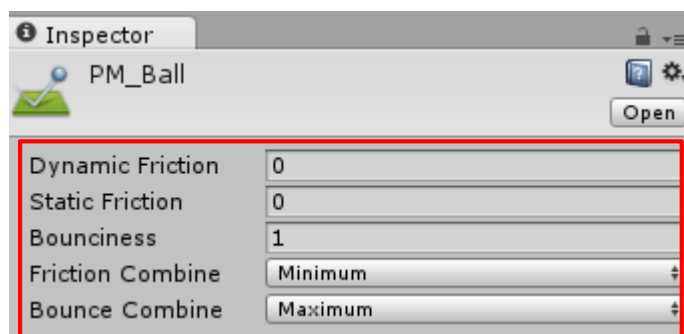
- ▶ BallのInspectorを開くとRigidbodyが追加されている
- ▶ 赤枠を変更
- ▶ Constraintsは名前の左の矢印を押すと開きます
- ▶ Use Gravityのチェックを外すと重力の影響を受けない
- ▶ Freeze PositionでZ軸には移動しない
- ▶ Freeze Rotationでそれぞれの軸で回転しない
- ▶ といった設定になります
- ▶ まだ足りないのでどんどん追加します



▶ 今度はメニューのAssetsから「Create」→「Physic Material」

▶ 名前を「PM\_Ball」に変更

▶ Inspectorを開いて赤枠を変更



▶ 上から

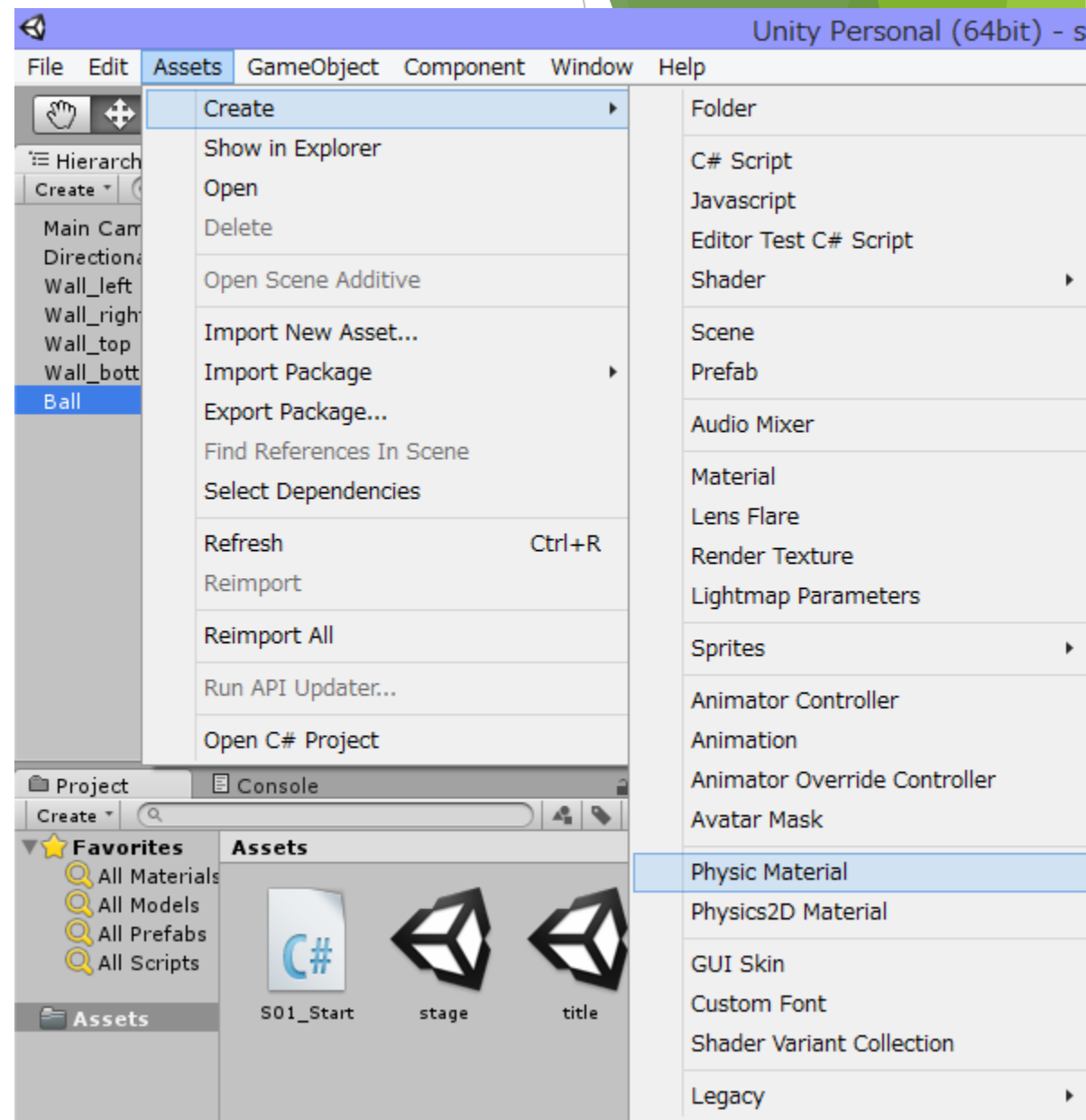
動摩擦

静止摩擦

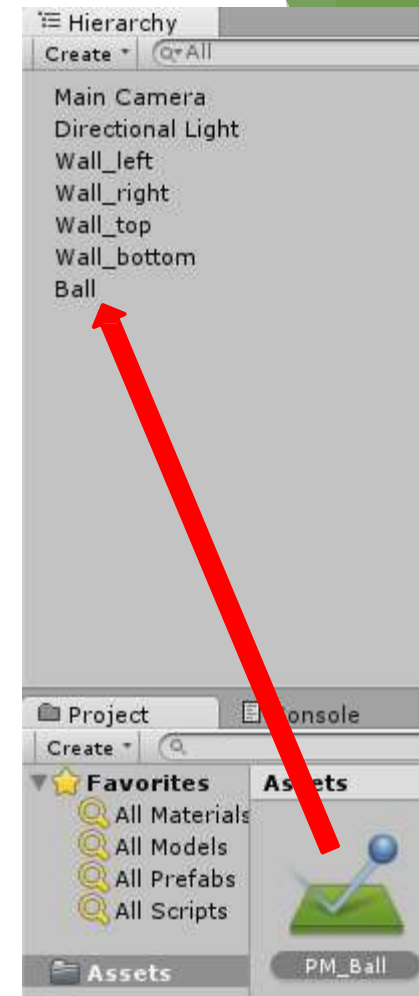
弾性力？

衝突したとき摩擦係数は小さい方を選ぶ

衝突したとき弾性力？は大きい方を選ぶ



- ▶ できたら「PM\_Ball」を「Ball」にドラッグ&ドロップ
- ▶ ここまで（特殊な）物理法則を与えてきました
- ▶あとは動きの情報を与えるだけ
  
- ▶ ここからプログラム
- ▶ メニュー、「Assets」→「Create」→「C# Script」
  
- ▶ 名前を「S02\_Ball」に変更
  
- ▶ 上の「PM\_Ball」と同様に「S02\_Ball」も「Ball」にドラッグ&ドロップしときましょう



- ▶ Assetsの「S02\_Ball」をダブルクリックで開き、プログラムを書いていきます

```
S02_Ball.cs* × S01_Start.cs
BlockKuzushi.CSharp S02_Ball
using UnityEngine;
using System.Collections;

public class S02_Ball : MonoBehaviour {

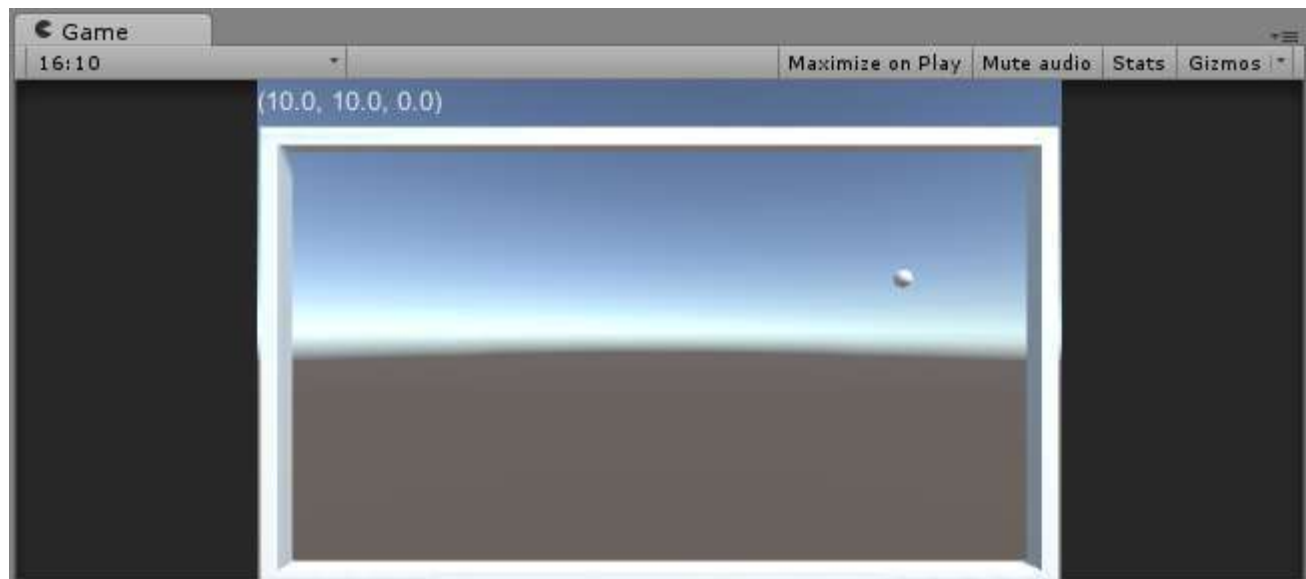
    void Start () {
        transform.GetComponent<Rigidbody>().velocity = new Vector3(10.0f, 10.0f, 0.0f);
    }

    void OnGUI()
    {
        GUI.Label(new Rect(0,0,200,20), ""+transform.GetComponent<Rigidbody>().velocity);
    }

}
```

- ▶ Start()は最初に関数でここでxとy方向に初速を10ずつ持たせる
- ▶ 設定した物理法則では、物体の衝突時にエネルギーの損失無しに跳ね返るため 壁にぶつかったとき速度がそのまま反転する
- ▶ OnGUIで画面左上に現在の速度を表示する

▶ できているとこんな感じ

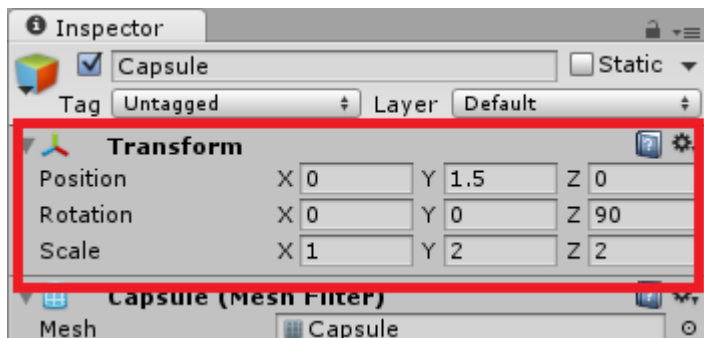


- ▶ 球が動き出して左上に速度が表示される
- ▶ ボールが壁にぶつくと跳ね返る
- ▶ 次はプレイヤーが動かすバーを作ろう



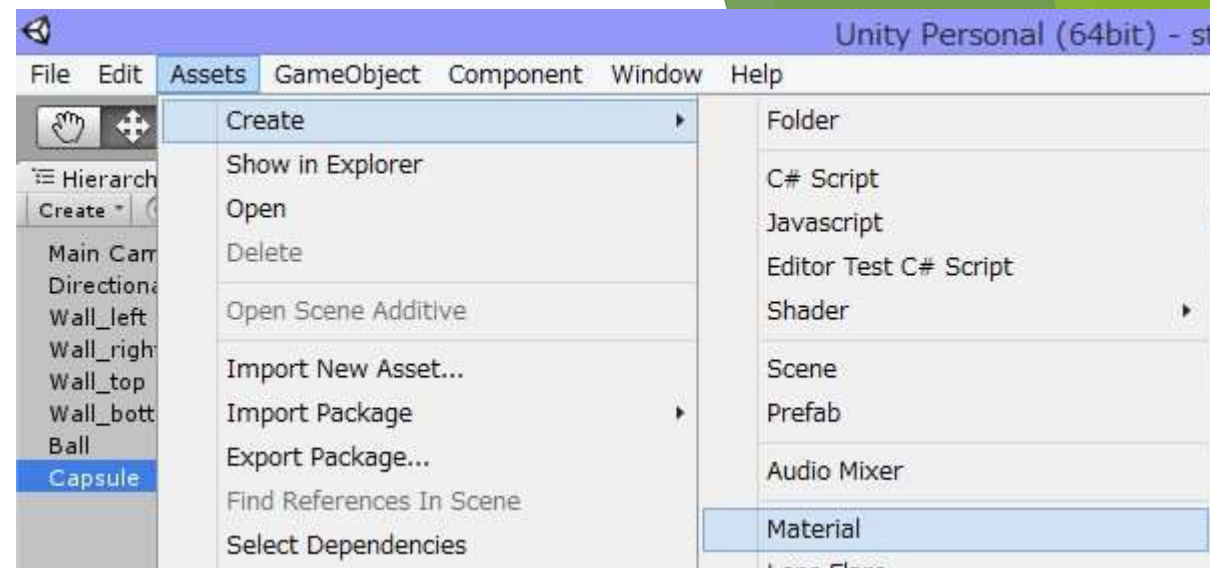
## 4-3. バーの作成

- ▶ 左右のカーソルキーで動くバーを作る
- ▶ 「Game Object」 → 「3D Object」 → 「Capsule」
- ▶ 名前を「Bar」に変更
- ▶ 赤枠を変更

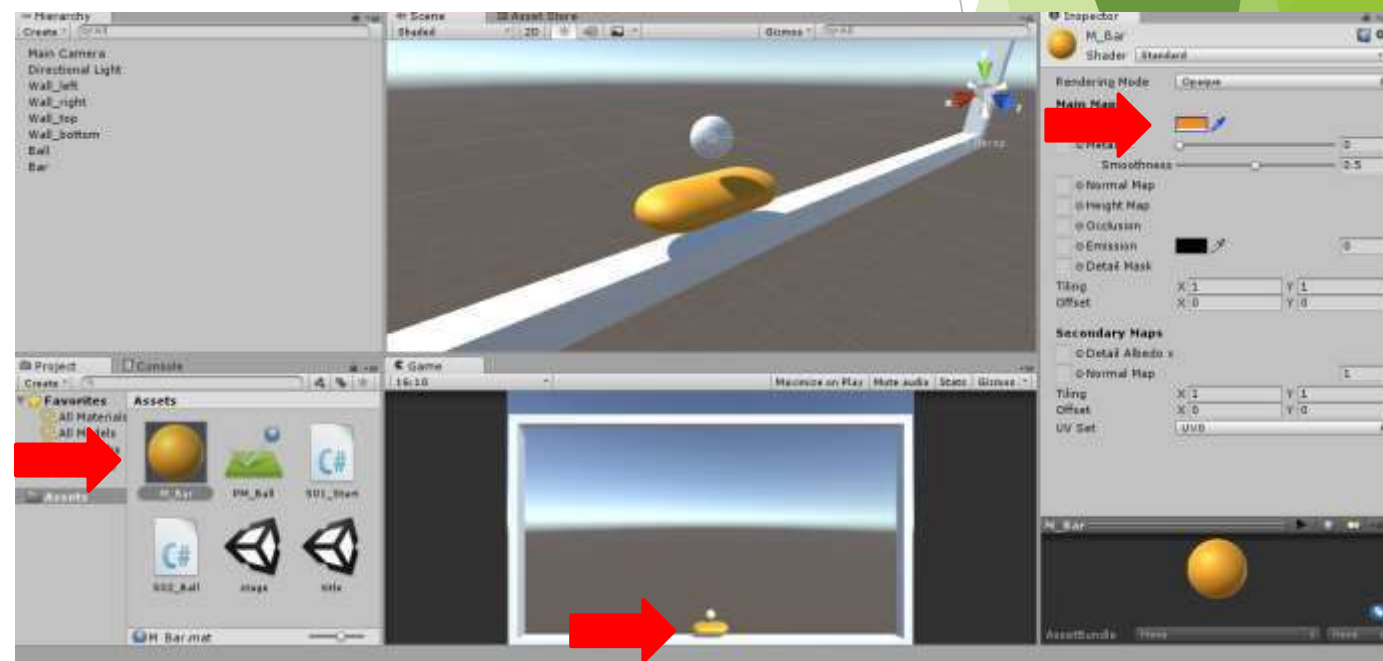


- ▶ Gameビューのボールの真下にバーが出来る

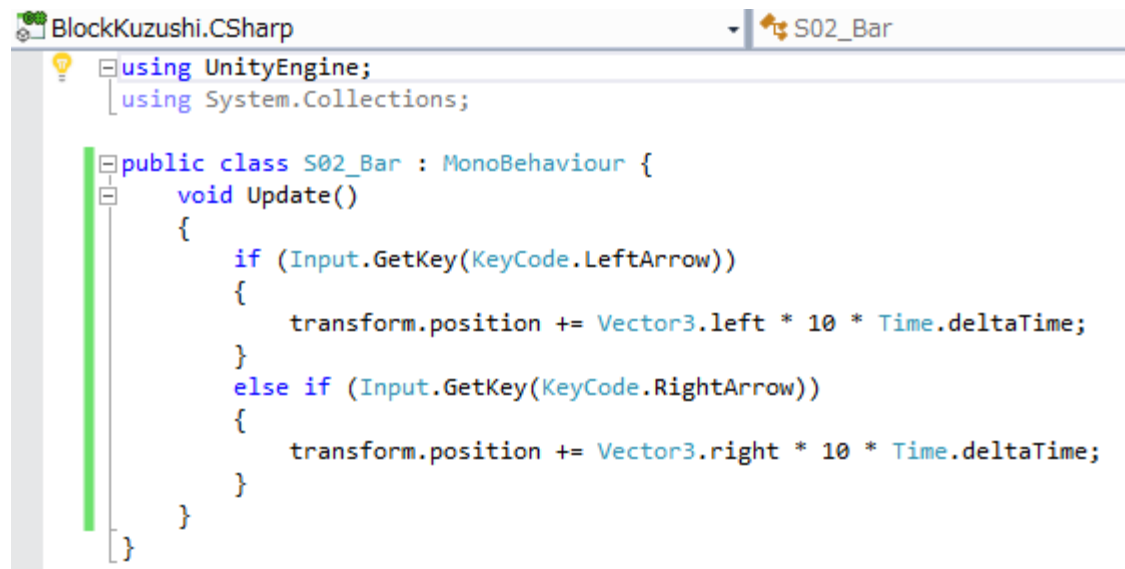
- ▶ バーに色を付けよう
- ▶ 「Assets」 → 「Create」 → 「Material」
- ▶ Assetsに出来るので名前を「M\_Bar」に変更
- ▶ 「M\_Bar」をHierarchyの「Bar」にドラッグ&ドロップ



- ▶ AssetsのM\_BarのInspectorを開く
- ▶ 好きな色に変える
- ▶ Gameビューのバーをの色を確認



- ▶ バーを動かすプログラムを書きます
- ▶ 「Assets」 → 「Create」 → 「C# Script」
- ▶ 名前を「S02\_Bar」に変更
- ▶ Hierarchyの「Bar」にドラッグ&ドロップ
- ▶ 「S02\_Bar」をダブルクリックで開く
- ▶ ↓を書く



```
BlockKuzushi.CSharp S02_Bar
using UnityEngine;
using System.Collections;

public class S02_Bar : MonoBehaviour {
    void Update()
    {
        if (Input.GetKey(KeyCode.LeftArrow))
        {
            transform.position += Vector3.left * 10 * Time.deltaTime;
        }
        else if (Input.GetKey(KeyCode.RightArrow))
        {
            transform.position += Vector3.right * 10 * Time.deltaTime;
        }
    }
}
```

- ▶ 書いたら保存してUnity画面に戻る

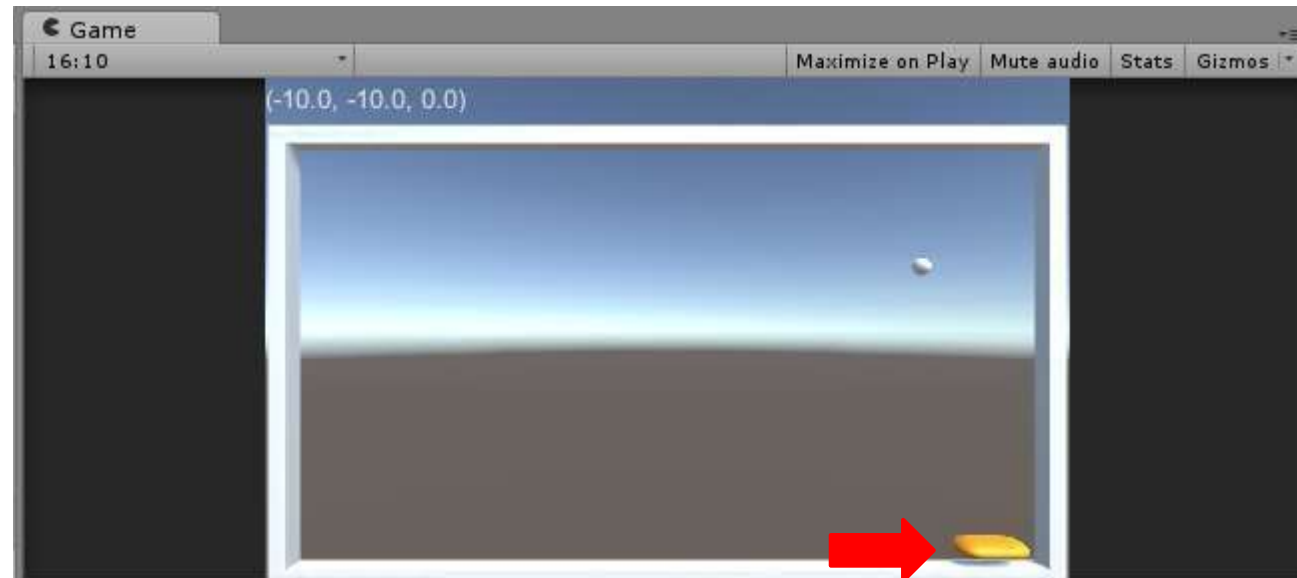
- ▶ 意味
- ▶ `Input.GetKey(KeyCode...)` → `KeyCode...`が押されたら`true`
- ▶ `transform.position` → Inspectorのtransformのpositionを示す
- ▶ `Vector3.left/right` →  $(-1.0f, 0, 0)$  /  $(1.0f, 0, 0)$ をそれぞれ示す
- ▶ `Time.deltaTime` → 秒間の速度になる（これを書かないと一瞬押しただけでものすごい進みます。一度試すといいかも）
  
- ▶ ゲームを実行する
- ▶ バーは動きますか？
  
- ▶ 動くけど、実はおかしいところがある
  - ①壁まで動かすとすり抜ける
  - ②よく見るとボールがftった時、バーじゃないとこで跳ね返る

- ▶ ①を直します
- ▶ 「S02\_Bar」を開く
- ▶ コードを修正
- ▶ If文を2つ追加
- ▶ それぞれバーのxが
  - ・ -18.5より大きい
  - ・ 18.5より小さい 場合にのみ動けるように

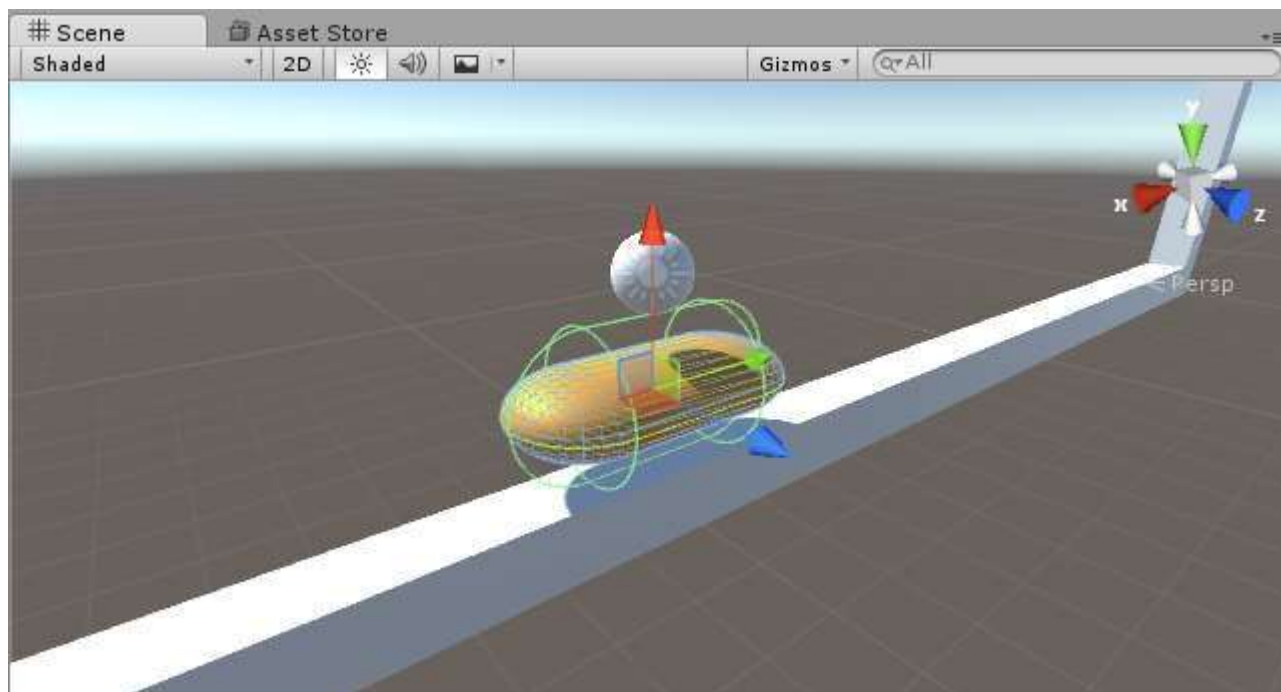
```
BlockKuzushi.CSharp S02_Bar
using UnityEngine;
using System.Collections;

public class S02_Bar : MonoBehaviour {
    void Update()
    {
        if (Input.GetKey(KeyCode.LeftArrow))
        {
            if (transform.position.x > -18.5f)
            {
                transform.position += Vector3.left * 10 * Time.deltaTime;
            }
        }
        else if (Input.GetKey(KeyCode.RightArrow))
        {
            if (transform.position.x < 18.5f)
            {
                transform.position += Vector3.right * 10 * Time.deltaTime;
            }
        }
    }
}
```

- ▶ ゲームを実行
- ▶ これで壁をすり抜けられない！

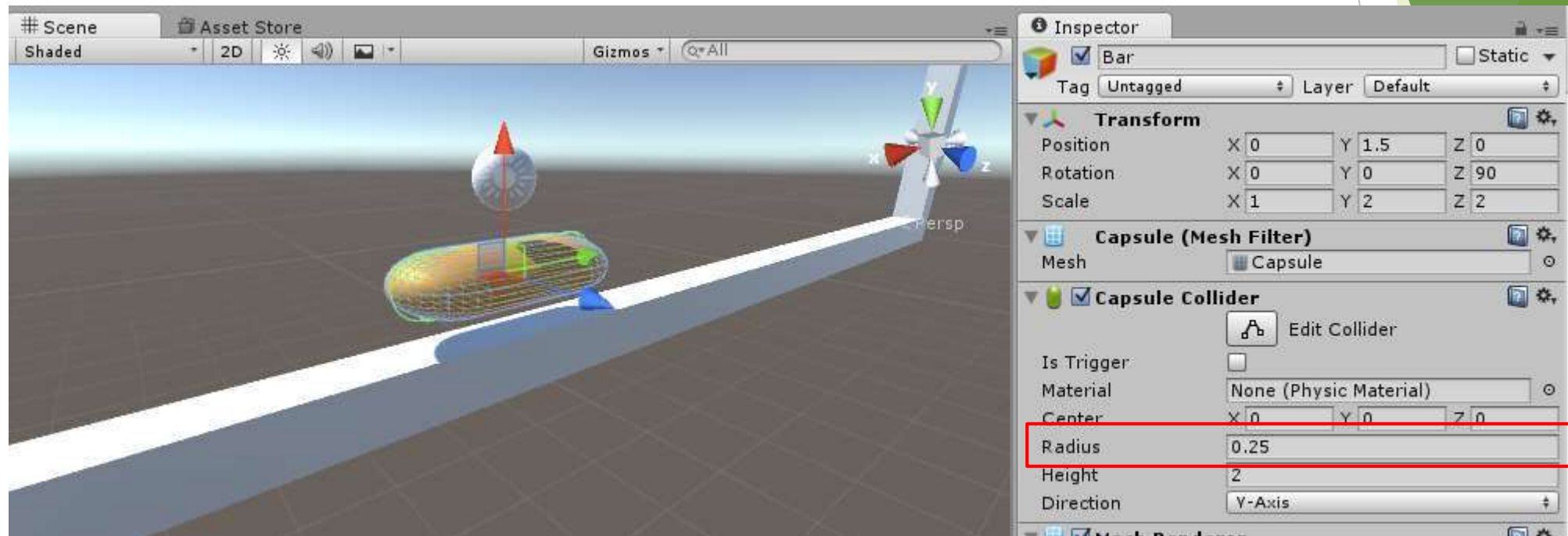


- ▶ ②を直します
- ▶ Rigidbodyを持つ「Ball」は「Bar」との接触面を判定する
- ▶ HierarchyのBarをダブルクリック
- ▶ するとSceneビューにBarがアップされて見える
- ▶ この緑線が接触面



- ▶ 緑線が実際のバーより大きくなっているのが分かる
- ▶ じゃあこの緑を変えよう

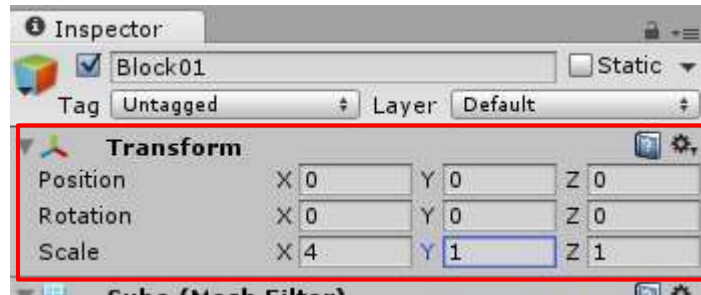
- ▶ BarのInspectorを開く
- ▶ 「Capsule Collider」の「Radius」を0.25に
- ▶ これで緑線が良い感じになる



- ▶ できたらゲーム実行
- ▶ これで問題解決

## 4-4. ブロックの作成

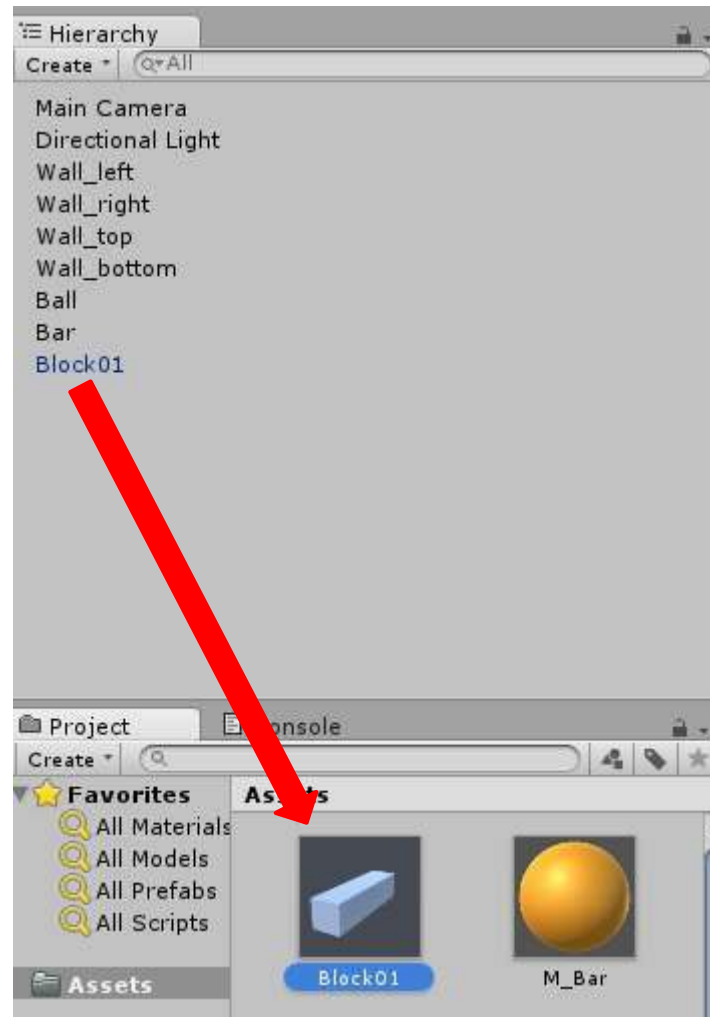
- ▶ ブロックを作っていきます
- ▶ 「Game Object」 → 「3D Object」 → 「Cube」
- ▶ 名前を「Block01」に変更
- ▶ まずはTransformの設定を変更



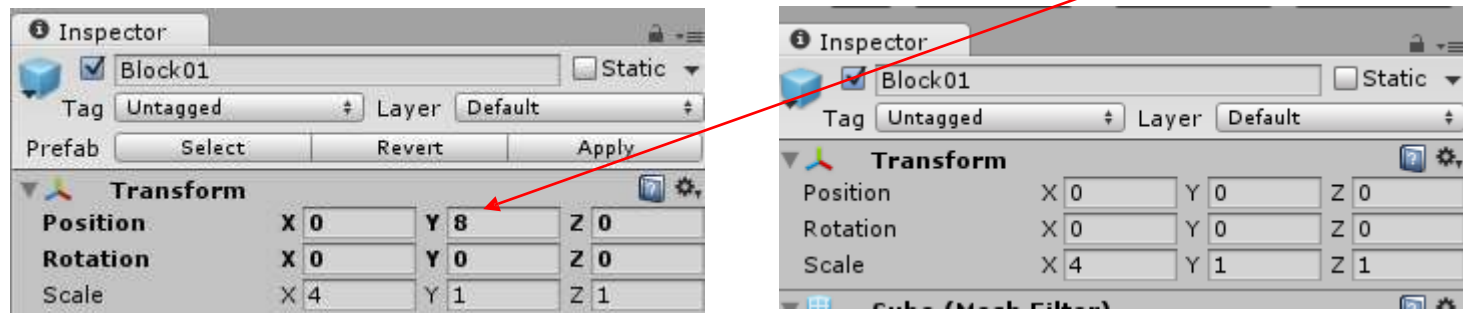
- ▶ 見えない場所にあるけど気にしない



- ▶ ここからちょっと複雑
- ▶ 今までと逆にHierarchyにある「Block01」をAssetsにドラッグ&ドロップ
- ▶ Assetsに「Block01」ができました
- ▶ これを「プレハブ化」と言います



- ▶ 左がHierarchyにある元のBlock01、右がAssetsのBlock01
- ▶ Transformの中が太字になっているものがある 見えるようにyを8に

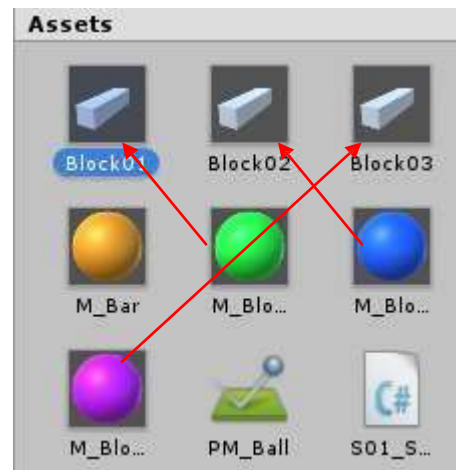
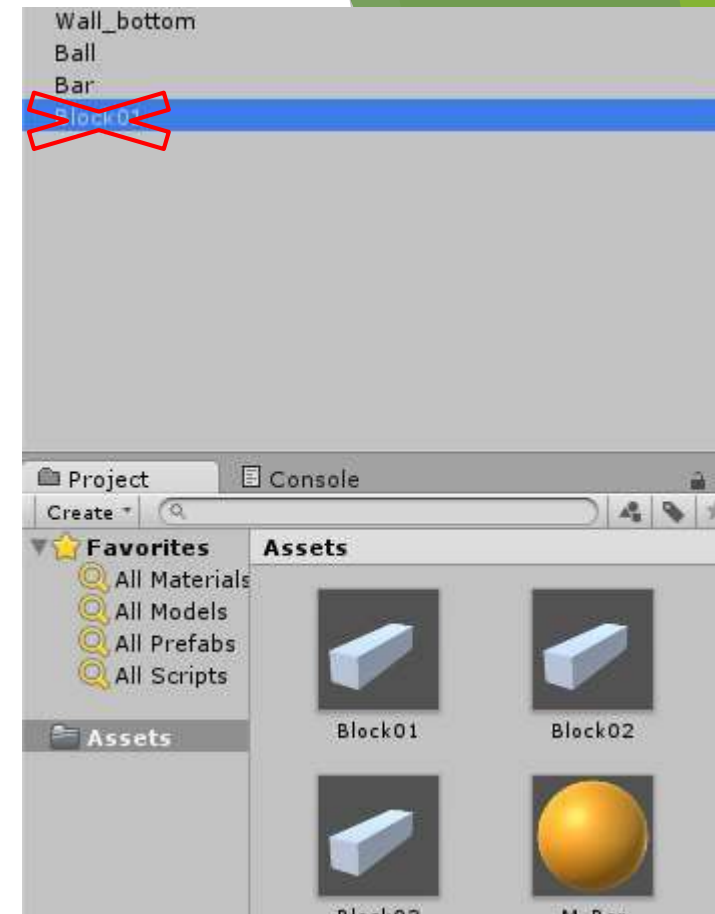


- ▶ 太字はオリジナルが持つ特有のデータです
- ▶ つまりScaleだけ特有ではない
- ▶ AssetsのBlock01のScaleを変更するとHierarchyのBlock01も変わる！  
(特有のデータであるposition, rotationは変わりません)
- ▶ プレハブは同じようなものを大量に作る際に便利

- ▶ ブロックが1種類だとつまらないので3種類作ります
- ▶ Hierarchyの「Block01」を2回Assetsにドラッグ&ドロップ
- ▶ Block01のプレハブを合計3つ用意
- ▶ それぞれ名前は「Block01」「Block02」「Block03」
- ▶ ブロック作成はプレハブ化したものを使うので  
HierarchyのBlock01は消去

- ▶ それぞれブロックに色を付けたいので
- ▶ 「Assets」→「Create」→「Material」×3
- ▶ 名前は「M\_Block01」「M\_Block02」「M\_Block03」
- ▶ 3つともInspectorを開いて色を設定（自由）

- ▶ M\_Block01はBlock01に
- ▶ M\_Block02はBlock02に
- ▶ M\_Block03はBlock03に
- ▶ それぞれドラッグ&ドロップ



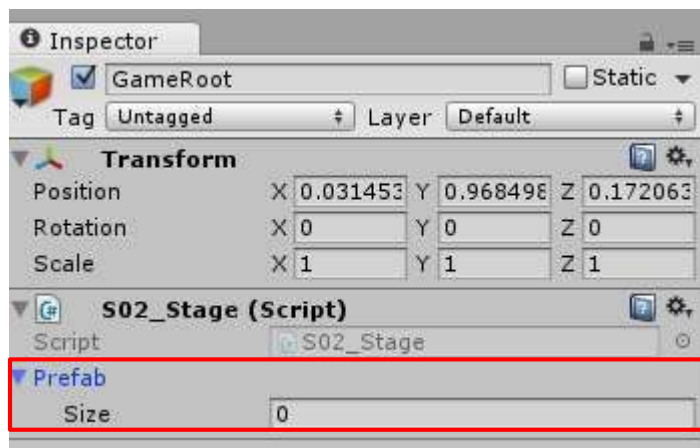
- ▶ ブロックをTh成するためのプログラムを書きます
- ▶ 「Assets」 → 「Create」 → 「C# Script」
- ▶ 名前は「S02\_Stgae」
  
- ▶ これをゲーム上で動かすためにオブジェクトに持たせる
- ▶ どれに持たせよう？  
→常にゲーム実行中に存在するオブジェクト
- ▶ ライトやカメラオブジェクトでいいけど、今回はそれ用のオブジェクトを作る
  
- ▶ 「Game Object」 → 「Create Empty」
- ▶ 名前は「GameRoot」
  
- ▶ Assetsの「S02\_Stage」をHierarchyの「GameRoot」にドラッグ&ドロップ

- ▶ まず「S02\_Stage」をいじる

```
C# Miscellaneous Files
using UnityEngine;
using System.Collections;

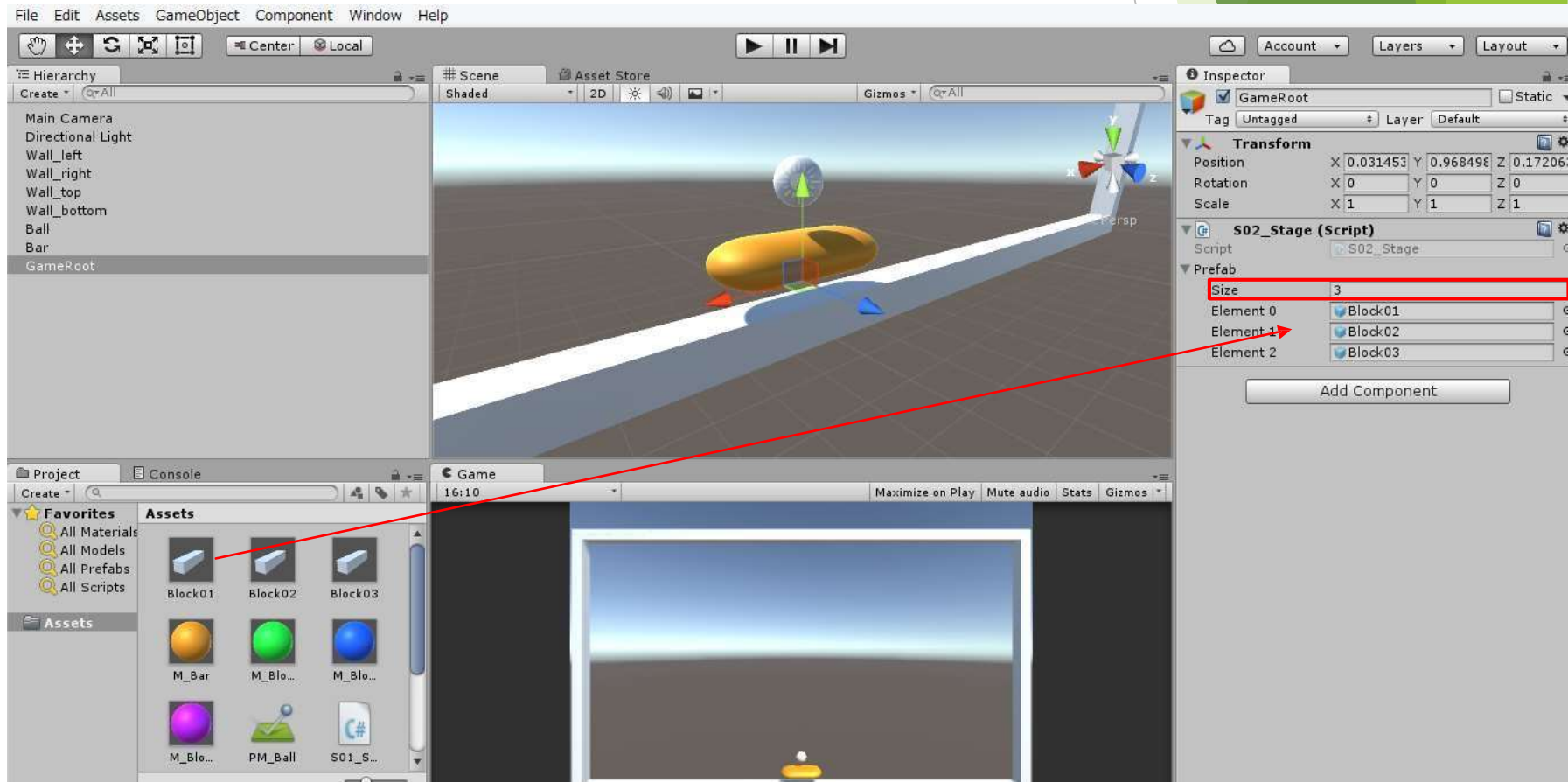
public class S02_Stage : MonoBehaviour {
    public GameObject[] prefab;
}
```

- ▶ これでprefabという変数を複数個用意してる
- ▶ 一度保存してUnityの画面に戻る
- ▶ HierarchyのGameRootのInspectorを開く



- ▶ S02\_Stageに「prefab」という項目が追加されてる

- ▶ Sizeを3に変更
- ▶ するとElement0, 1, 2がでてくるのでAssetsから
- ▶ Element0にBlock01
- ▶ Element1にBlock02
- ▶ Element2にBlock03
- ▶ をドラッグ&ドロップ



- ▶ これで使うブロックのオブジェクトを設定できた
- ▶ プログラムを書いていく

```
S02_Stage.cs  S02_Bar.cs  S02_Ball.cs  S01_Start.cs
Miscellaneous Files  S02_Stage

using UnityEngine;
using System.Collections;

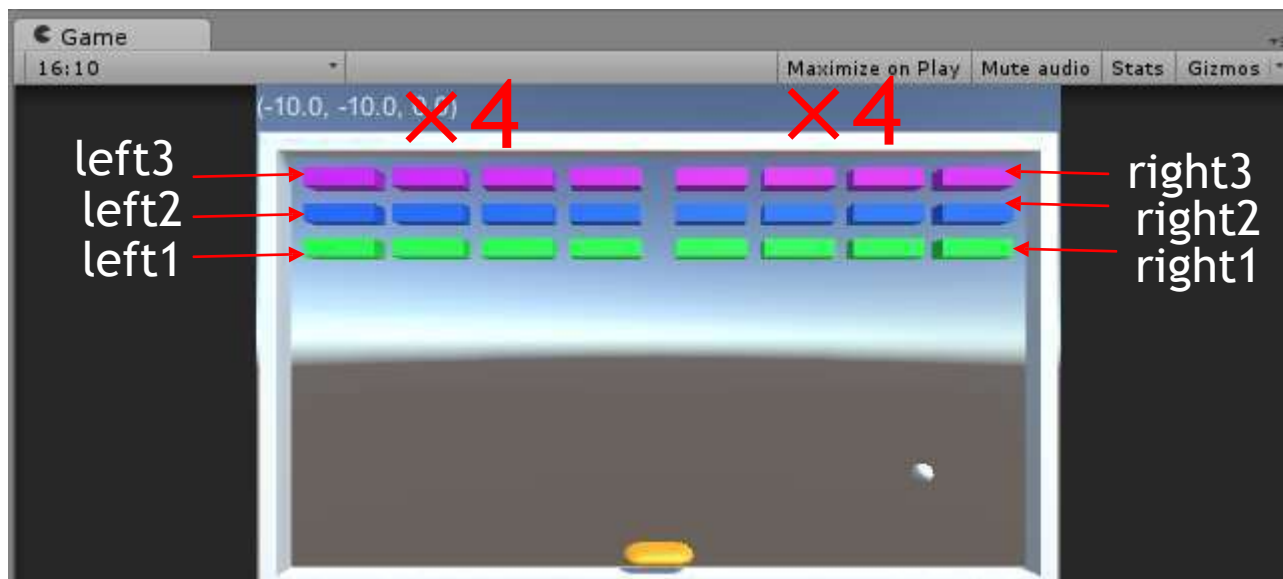
public class S02_Stage : MonoBehaviour {
    public GameObject[] prefab;

    void Start()
    {
        for(int i = 0; i < 4; i++)
        {
            GameObject block_right1 = GameObject.Instantiate(prefab[0]) as GameObject;
            GameObject block_right2 = GameObject.Instantiate(prefab[1]) as GameObject;
            GameObject block_right3 = GameObject.Instantiate(prefab[2]) as GameObject;
            GameObject block_left1 = GameObject.Instantiate(prefab[0]) as GameObject;
            GameObject block_left2 = GameObject.Instantiate(prefab[1]) as GameObject;
            GameObject block_left3 = GameObject.Instantiate(prefab[2]) as GameObject;

            block_right1.transform.position = new Vector3(3.0f + (5 * i), 19.0f, 0.0f);
            block_right2.transform.position = new Vector3(3.0f + (5 * i), 21.0f, 0.0f);
            block_right3.transform.position = new Vector3(3.0f + (5 * i), 23.0f, 0.0f);
            block_left1.transform.position = new Vector3(-3.0f - (5 * i), 19.0f, 0.0f);
            block_left2.transform.position = new Vector3(-3.0f - (5 * i), 21.0f, 0.0f);
            block_left3.transform.position = new Vector3(-3.0f - (5 * i), 23.0f, 0.0f);
        }
    }
}
```

- ▶ 頑張ってください

- ▶ 上6行で配置するゲームオブジェクトを定義
- ▶ 下6行で配置する場所を指定
- ▶ ゲーム実行



- ▶ こんな感じでブロックがTh成される
- ▶ 衝突判定はあるけど壊れない
- ...



## 4-5. ブロックの破壊

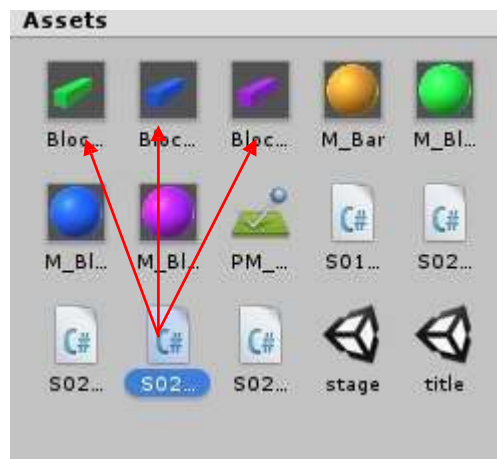
- ▶ 「Assets」 → 「Create」 → 「C# Script」
- ▶ 名前は「S02\_Block」
- ▶ →のプログラムを書く

```
using UnityEngine;
using System.Collections;

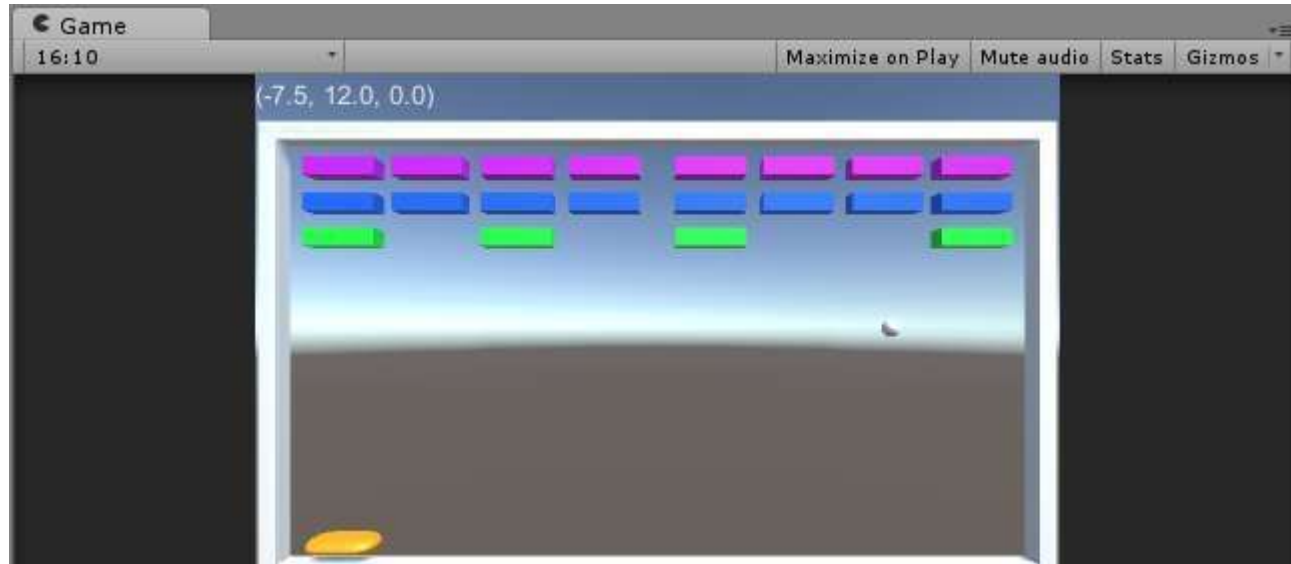
public class S02_Block : MonoBehaviour {

    void OnCollisionEnter(Collision other)
    {
        GameObject.Destroy(this.gameObject);
    }
}
```

- ▶ オブジェクトに衝突したとき、このオブジェクトを破壊するという意味です
- ▶ 書けたらAssetsにある「Block01」「Block02」「Block03」にドラッグ&ドロップ



## ▶ ゲーム実行



- ▶ こんな感じでボールにぶつかったブロックが消える
- ▶ なんとなくそれっぽくなってきた

▶ 今回はここまで  
で

▶ 次回は

- ・ スコア処理
- ・ ブロックの体力設定
- ・ 効果音追加
- ・ ゲームクリア処理
- ・ etc

▶ という感じでいこうかと思います

▶ 分からないことがあったら聞いてください

▶ メールで連絡くれても大丈夫です

お疲れさまでした