

Unity講座①

2018/6/18(月)

Unityとは…？

以下Wikipediaから(教授ブチギレ～)

Unityとは、統合開発環境を内蔵し、複数のプラットフォームに対応する
ゲームエンジンである。

iOS, Android, PlayStation 3, PlayStation 4, PS Vita, Xbox 360, Xbox One, Wii U
そしてVR/AR向けの開発に対応している

要するに...

- ゲーム制作における難しい部分を既にやっておいてくれて
ユーザーは最低限の技術でゲーム制作を始められる！！(いいのかこれで?)

どうやってゲームを作っていくのか

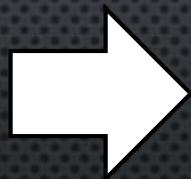


オブジェクト
(3Dモデル)

+



移動スクリプト
(プログラム)



オブジェクトが動く(ように見える)

つまり！

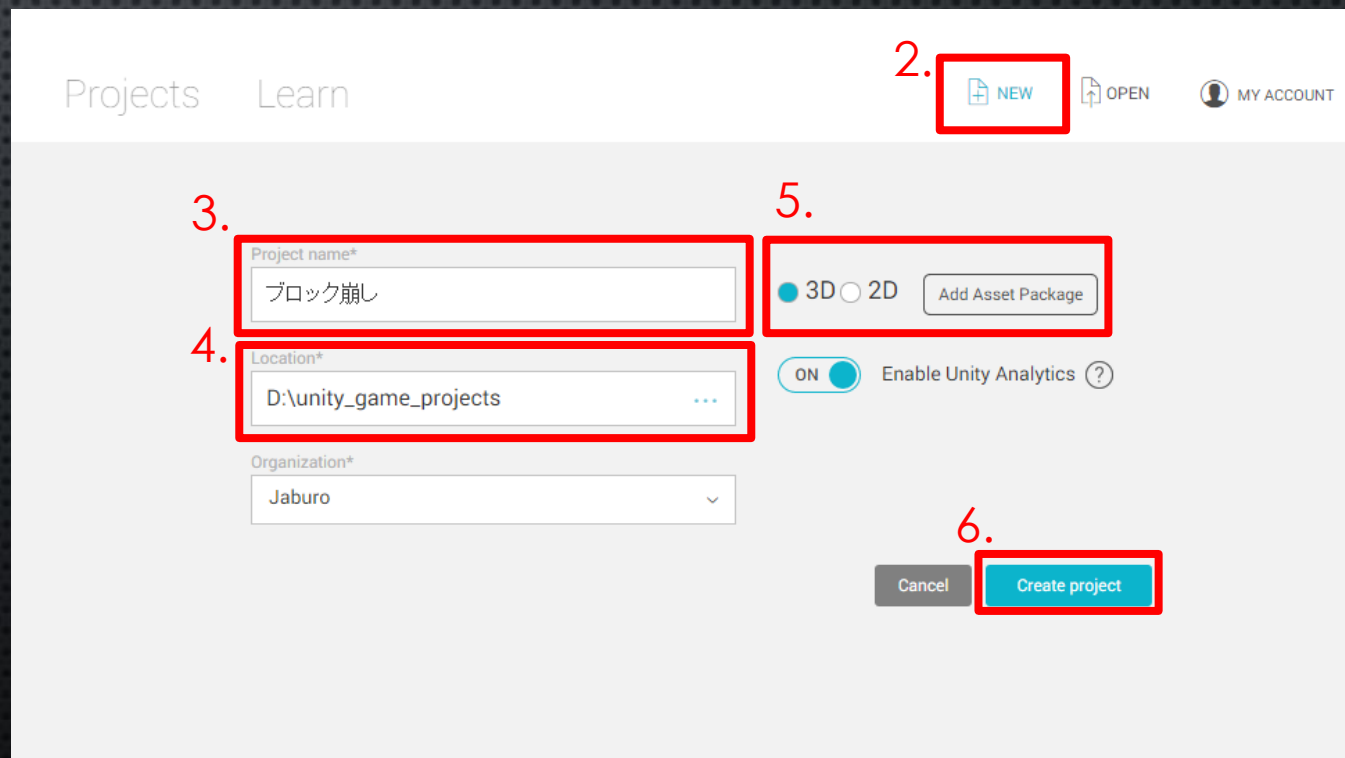
- ゲームというのは現実のように「歩いているから進んでいる」ではなく
「キャラクターの歩きモーション」と「オブジェクトを動かすプログラム」を
組み合わせることで「そういう風に錯覚させている」のである！

※あくまで自分が作ってきた時に行きついた考え

プロジェクト作成

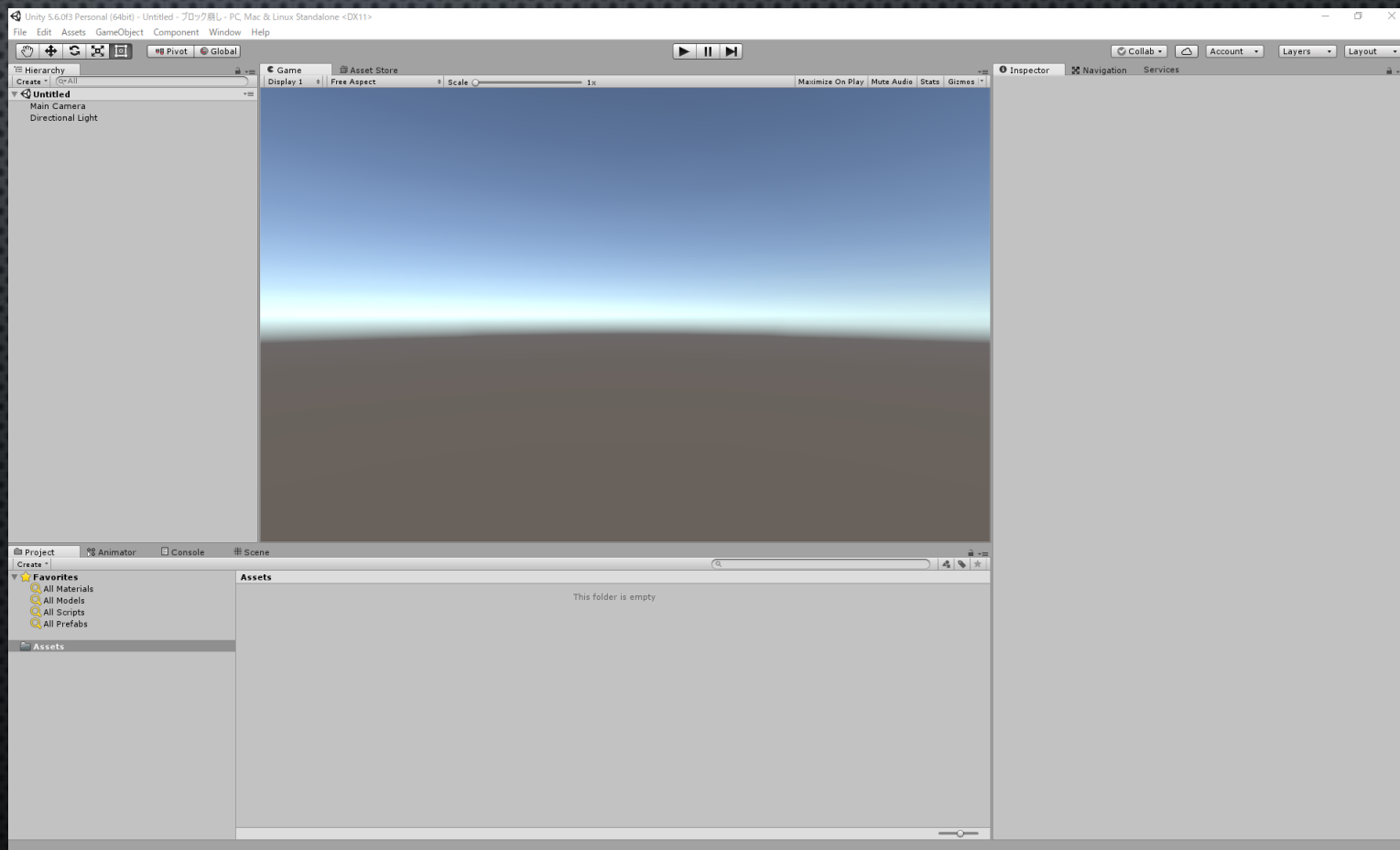
まずプロジェクトから作らないと何も始まらない！

1. Unityを起動
2. 右上にある「NEW」を押す
3. プロジェクト名を決める
(今回はブロック崩し)
4. 保存先は各自におまかせ
5. 「3D」にチェックが入っているか
6. 上記ができたなら右下
「create project」をクリック

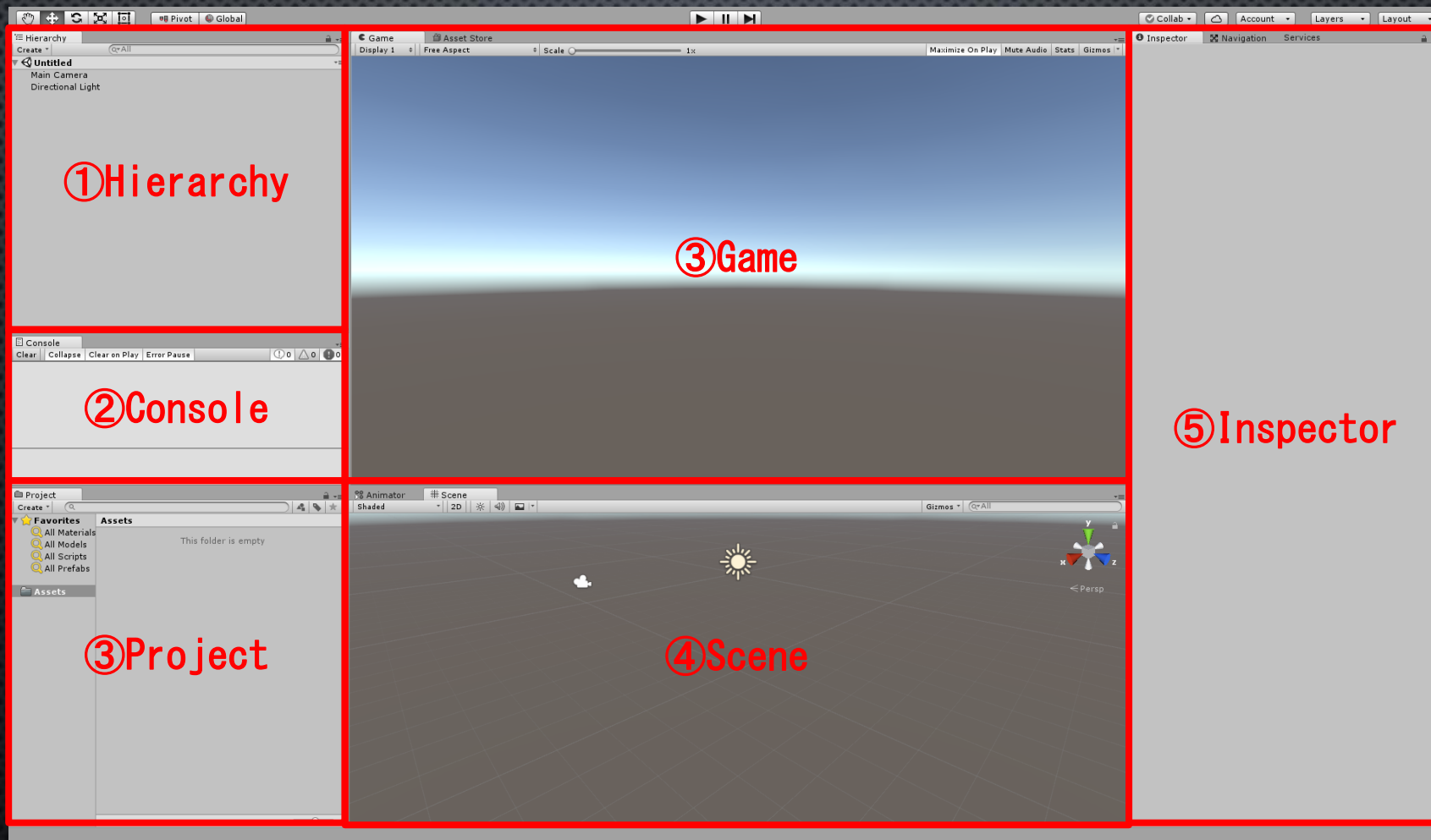


Unityの画面

プロジェクトを作り終わるとこんな画面が出てくる
英語ばかりでわけわからね



画面の見方



Hierarchy

そのシーン内にあるオブジェクトを表記

※シーンに関しては後で説明

初期状態だと「Main Camera」(カメラ)と
「Directional Light」(太陽?)が
入っている

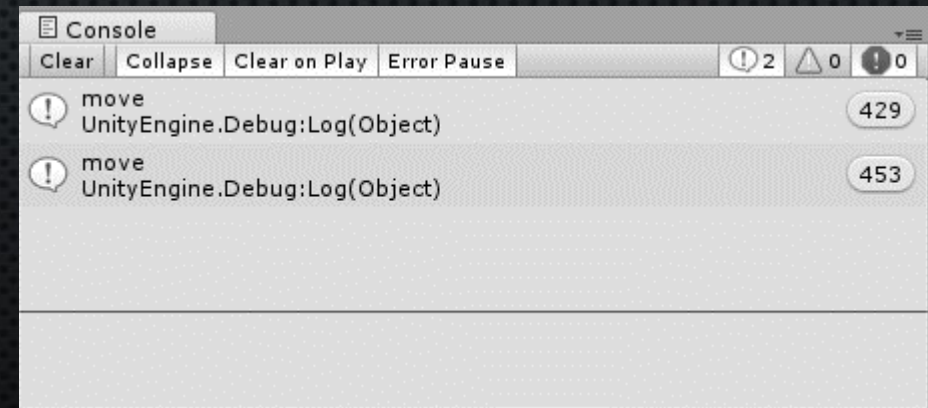


Console

エラー、警告など Unity によって生成されたメッセージを表示する。
Debug.Log, Debug.LogWarning や Debug.LogError 関数を使用して、
コンソールでメッセージを表示することもできます。

ゲームには表示されない(開発者のみ見れる)。

プログラムが思ったように動かない時はこれを活用してプログラムの動きを追おう
(超重要)



Project

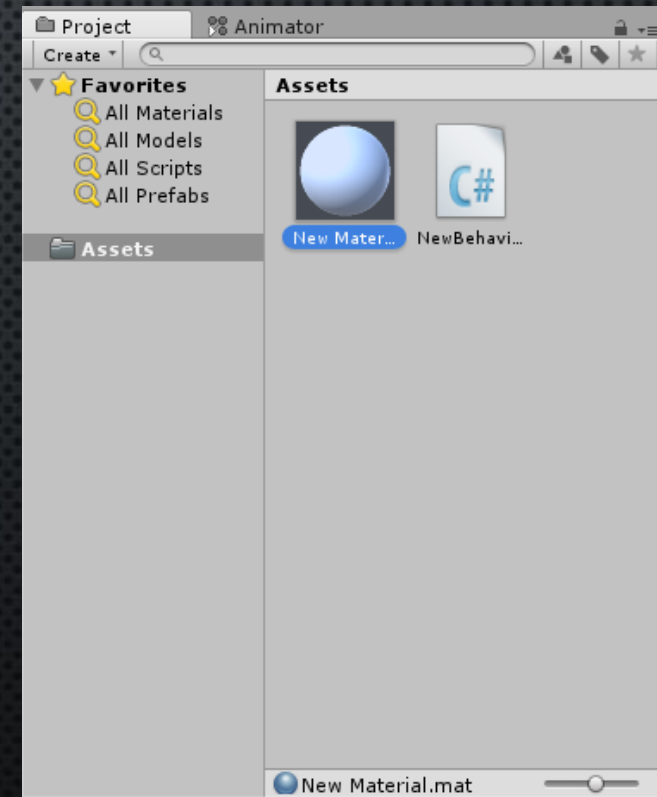
プロジェクトの保存先にある「Assets」フォルダの中身を表示する画面

スクリプト(プログラム)データはここに保存される

他にも様々なデータを保存する場所

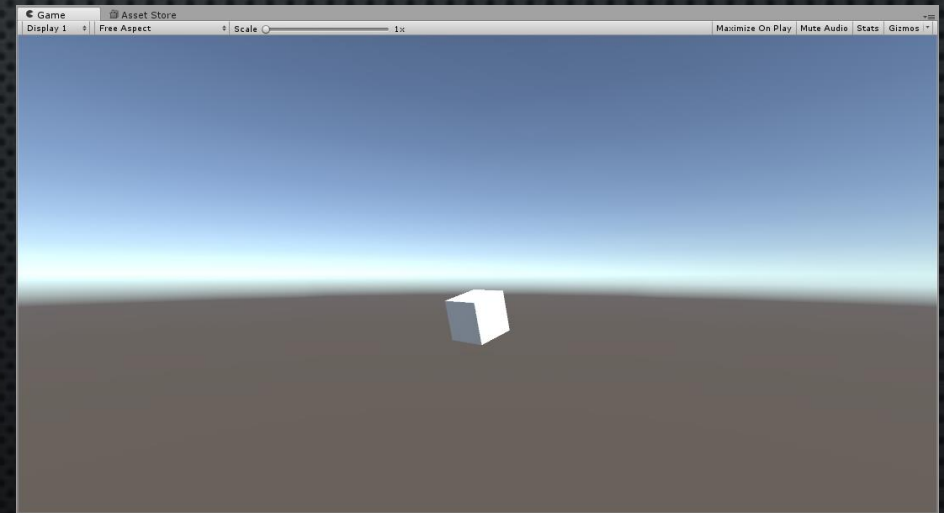
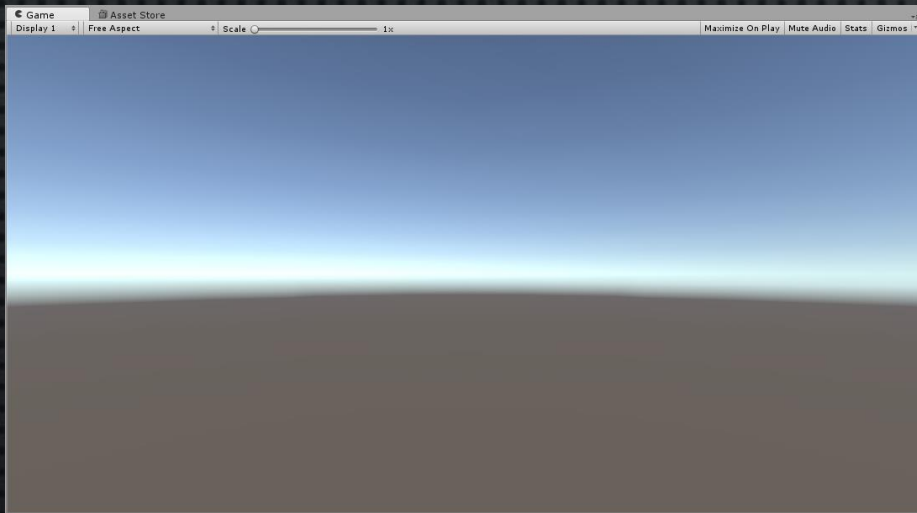
右の画像だと「New Material」(マテリアル)と
「New Behaviour」(スクリプト)が入ってる

データフォルダって考えるのが楽かな？



Game

現在作っているゲームを実際にプレイした時に映る画面
初期状態(左下)だと何もないので空しか映らないが
オブジェクトを追加するとこんな感じ(右下)



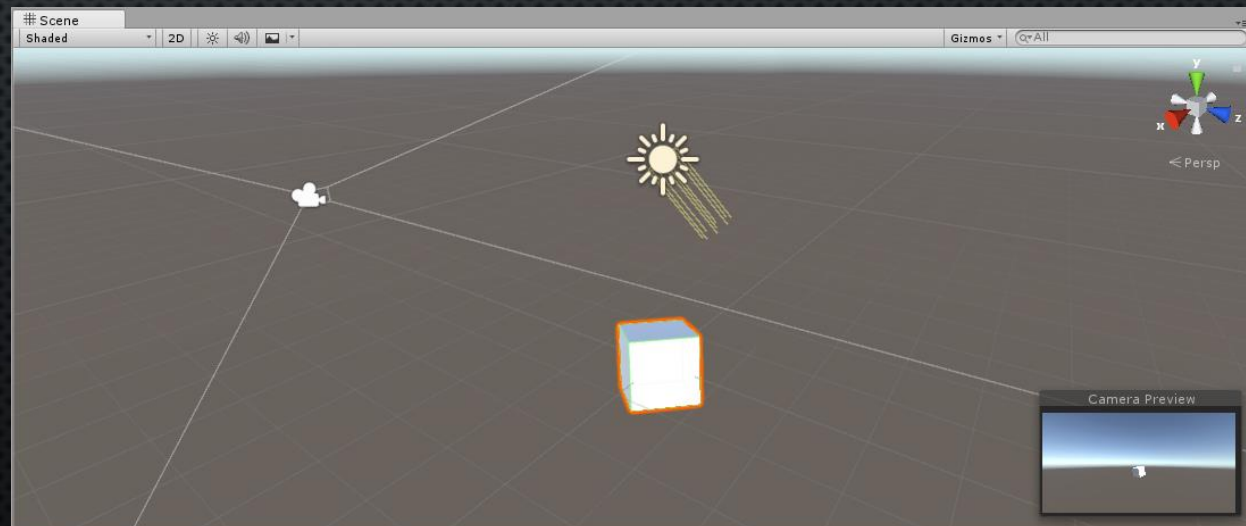
Scene

ゲームオブジェクトを配置する画面
ワールド内を自由に見渡せる

Gameとの違い：

パワポで例えると

作ってるときの画面が「Scene」で
スライドショーにしてる時が「Game」



Inspector

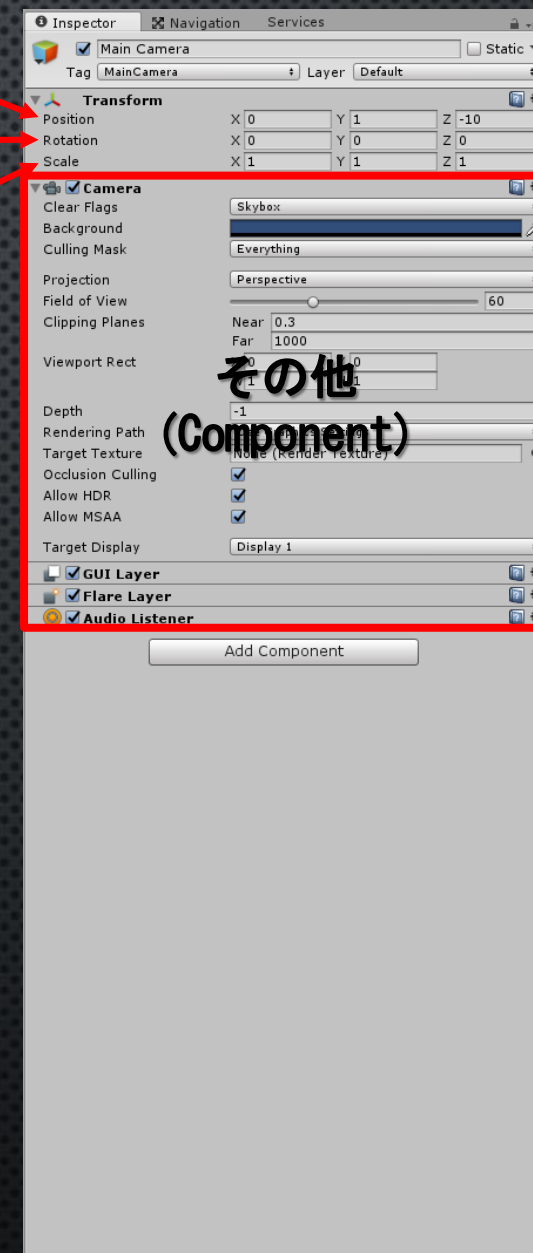
選択したゲームオブジェクトのステータスを表示する画面

オブジェクトの「座標」、「向き」、「大きさ」はここから設定する

座標

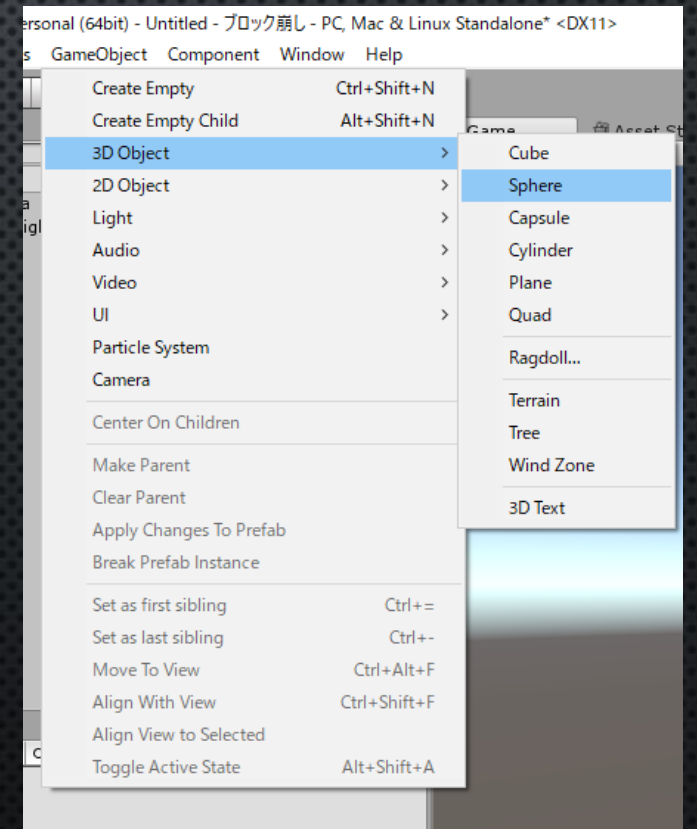
向き

大きさ



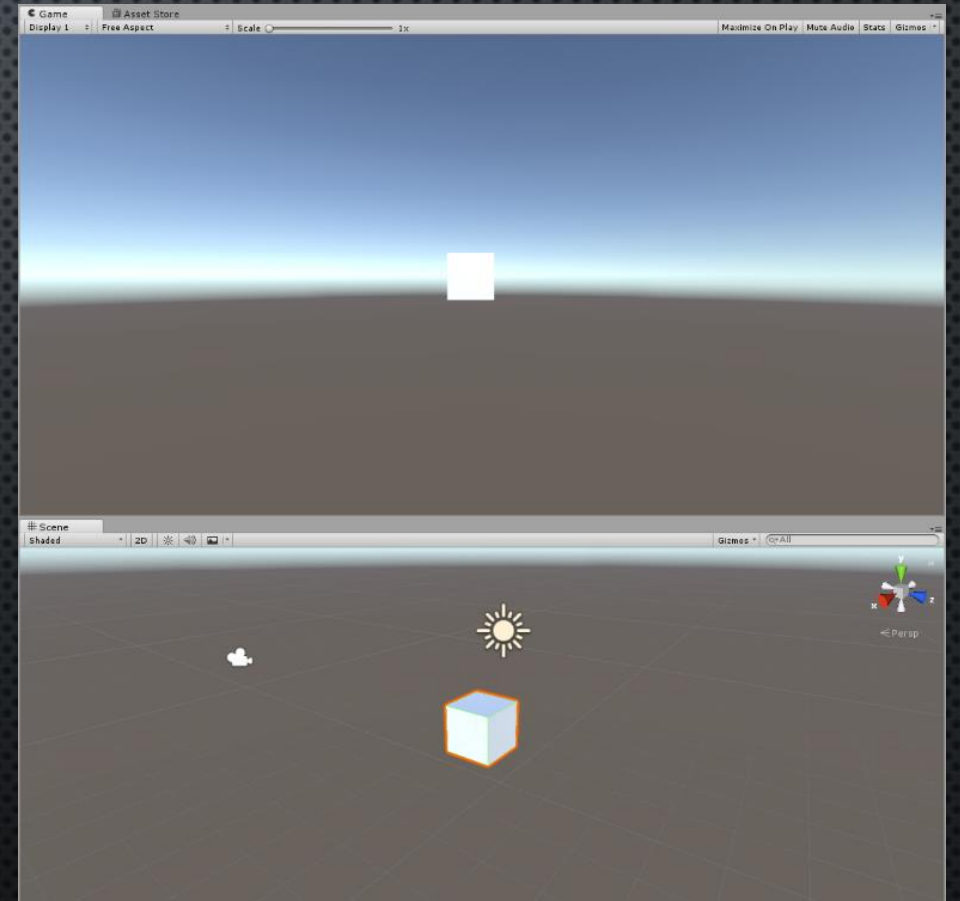
オブジェクト生成①

- 実際にオブジェクトを作ってみよう
上のメニューバーから
GameObject>3D Object>Cube
を選択



オブジェクト生成②

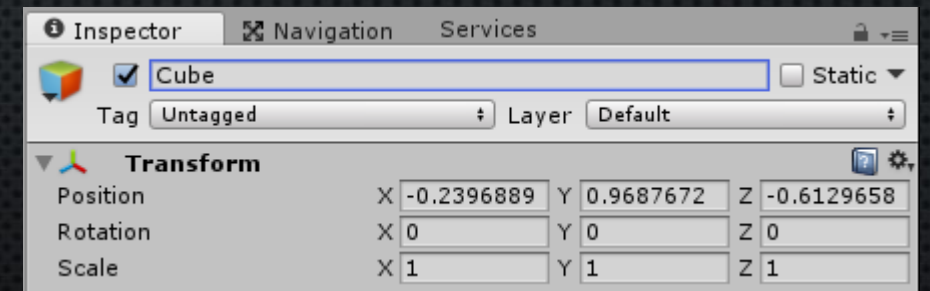
なんか出た！！



オブジェクトを動かす

オブジェクトの動かし方(初期位置の決め方)は何種類がある

1. 左上のアイコンをクリックして操作
2. Inspectorから座標入力
3. Inspectorから(part2)



オブジェクトを動かす

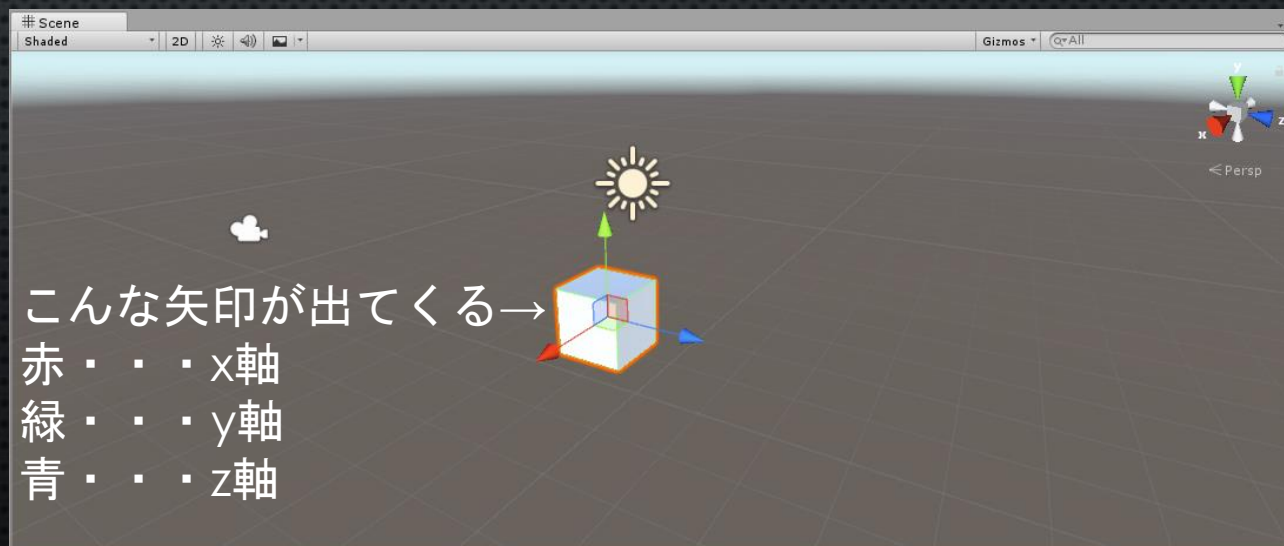


Unityの左上のメニューバー下にこんなものがあるはず



・・・このアイコンをクリックすると選択したオブジェクトを動かせるようになる
(Wキーでショートカット)

矢印をドラッグして
動かせる
(選ばなかったら軸に関係なく自由に
動かせる)



その他機能



ついでにその他のアイコンの機能も紹介(ショートカットキー)



．．． 現在見ている視点を動かす(Qキー)



．．． 選択したオブジェクトを回転させる(Eキー)



．．． 選択したオブジェクトの大きさを変える(Rキー)

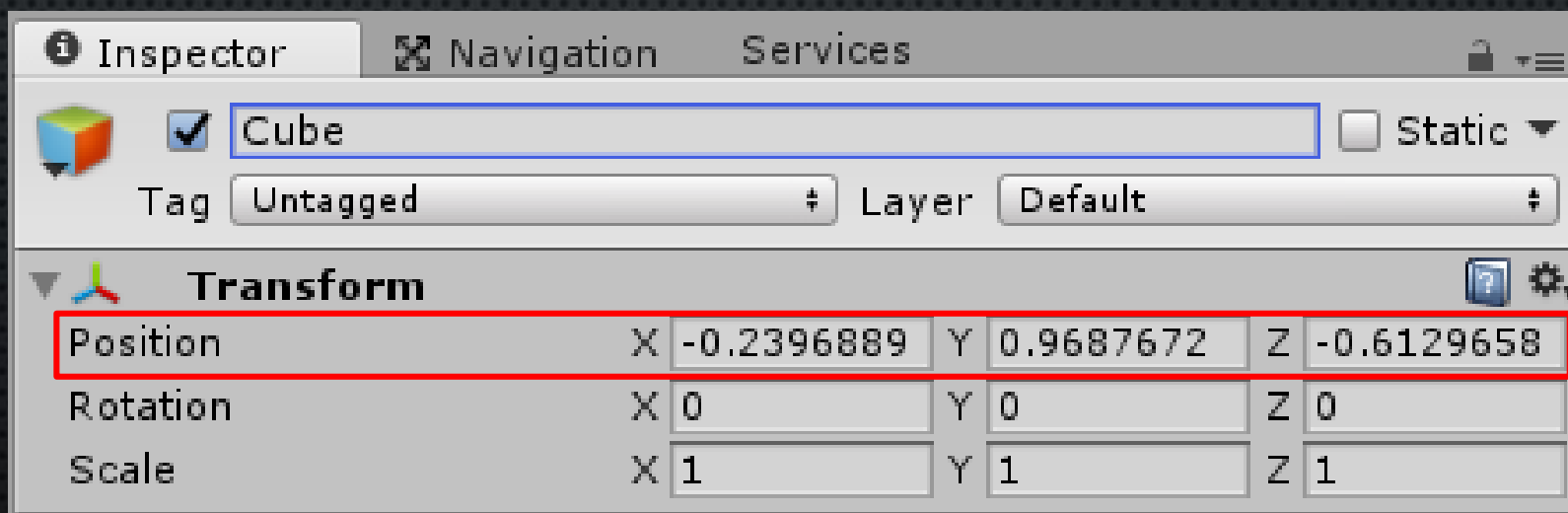


．．． 上と同じで大きさを変えるが少しやり方が違う(Tキー)

オブジェクトを動かす

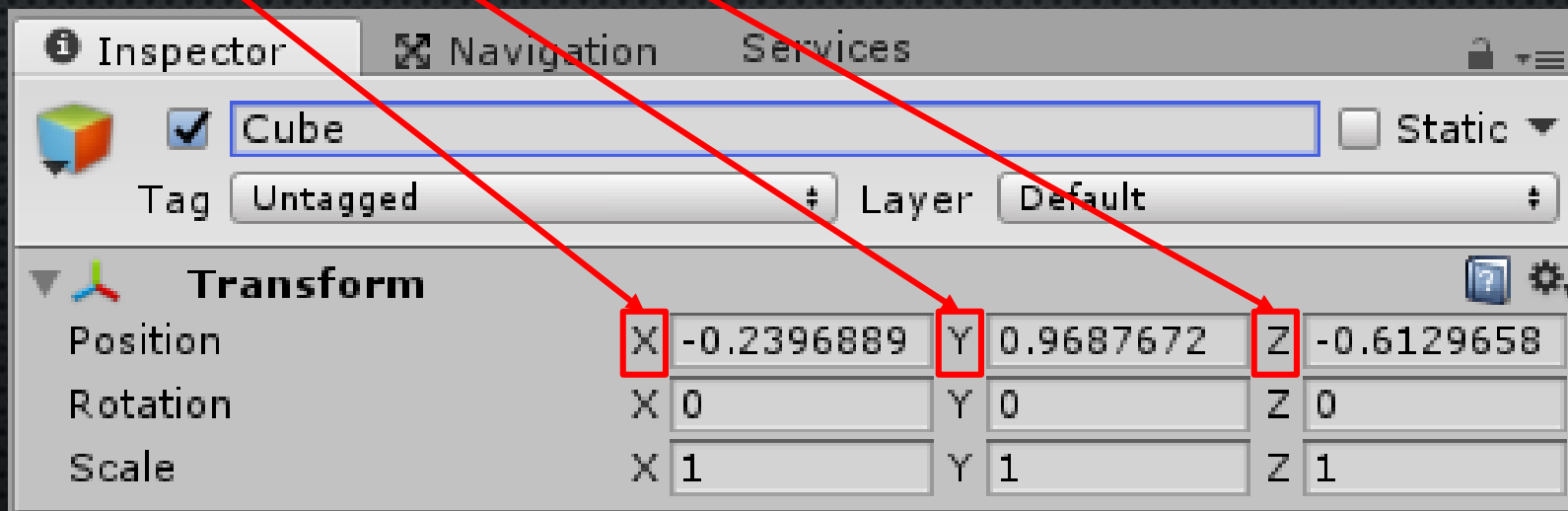
Inspectorから座標入力

座標を入力してその位置に移動させる
細かい調整や、計算して設置するときはこちら



オブジェクトを動かす(おまけ)

実は「X」、「Y」、「Z」の文字をクリックすると  みたいに操作できる



ゲームの実行



ゲームの実行はUnityの画面真ん中上にある再生ボタンを押すとゲームが開始されGameタブでプレイすることができる



・・・ゲームが開始される



・・・ゲームを一時停止



・・・ゲームを一時停止し、1フレームごとに進ませる

重要

デバッグ時

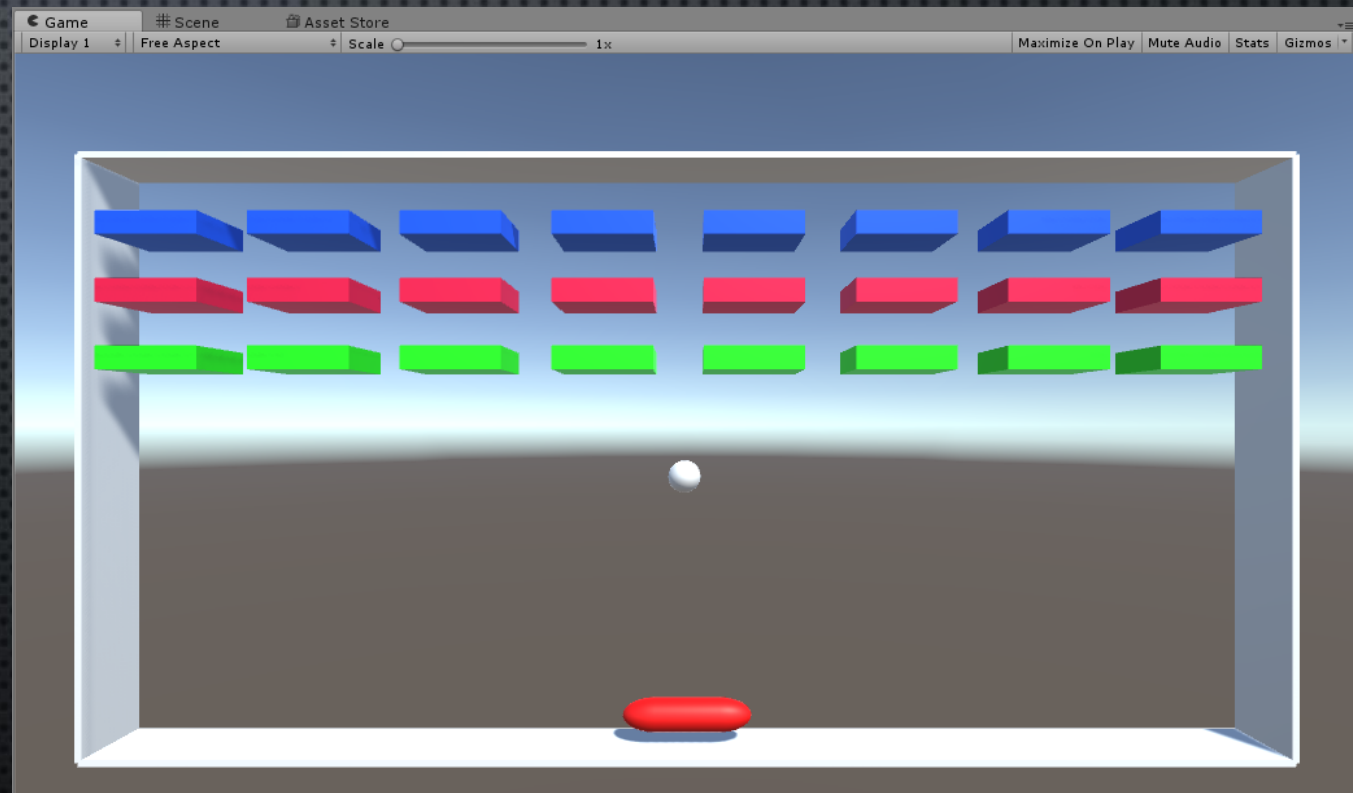
今回のテーマ

テーマ

初心者向け講座ということで今回は

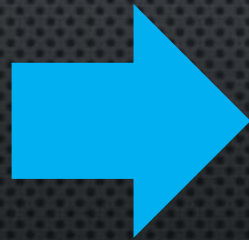
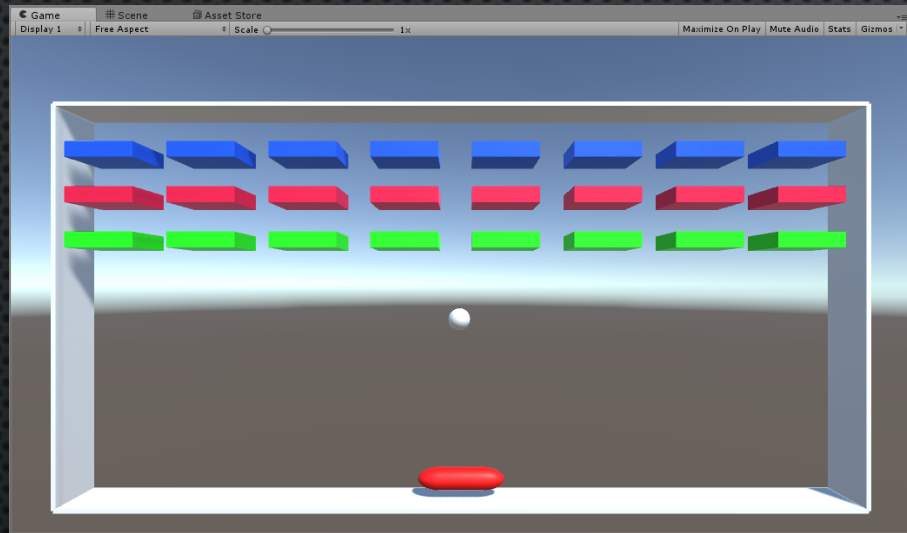
ブロック崩し

を作っていきます



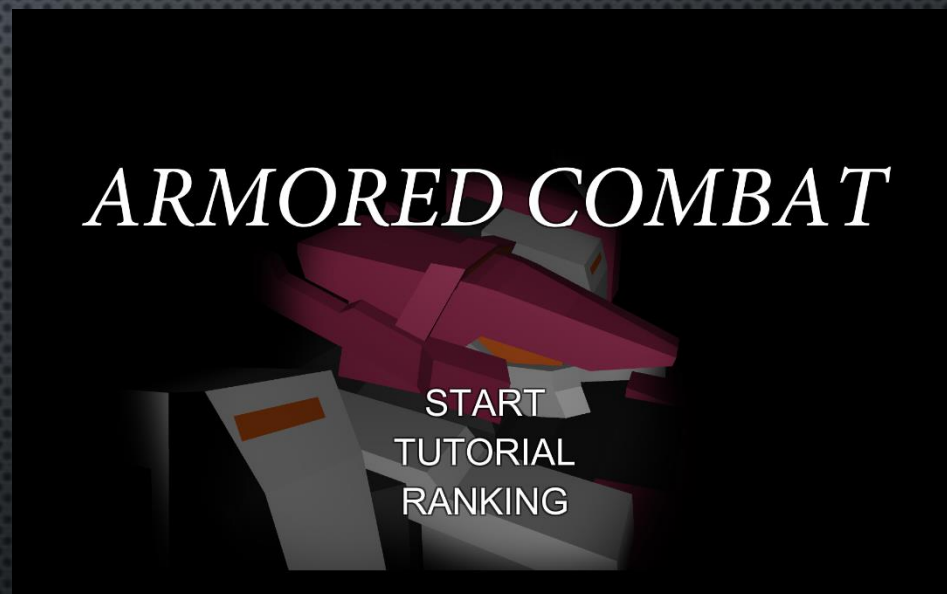
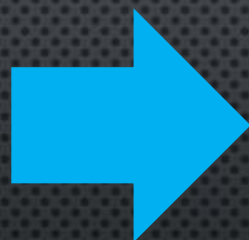
「は？アーマードコアみたいなゲームじゃねーのかよ」と思ったあなた！

ブロック崩しにはUnityの基礎がたくさん詰まっています！



ブロック崩しを改造すると「こがねこのエアライド」になるんです！

さらに！！



こがねこのエアライドを改造すると「ARMORED COMBAT」になるんです！

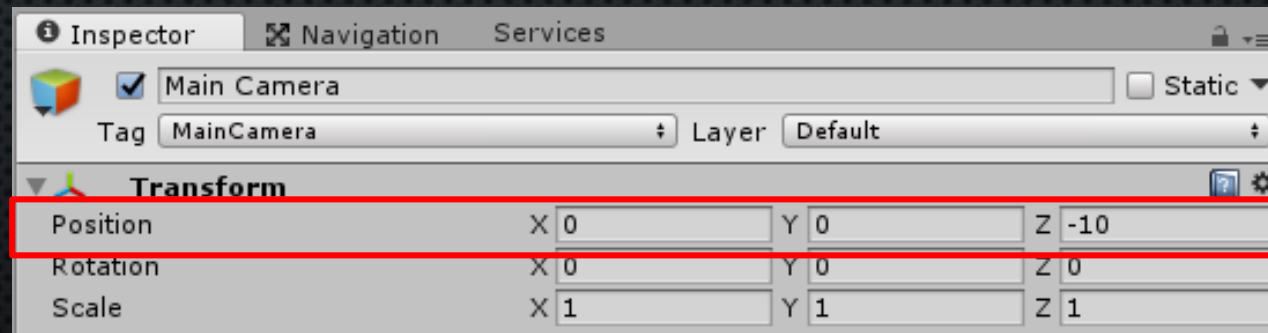
ステージ作成

の前に...

カメラの位置を設定しよう

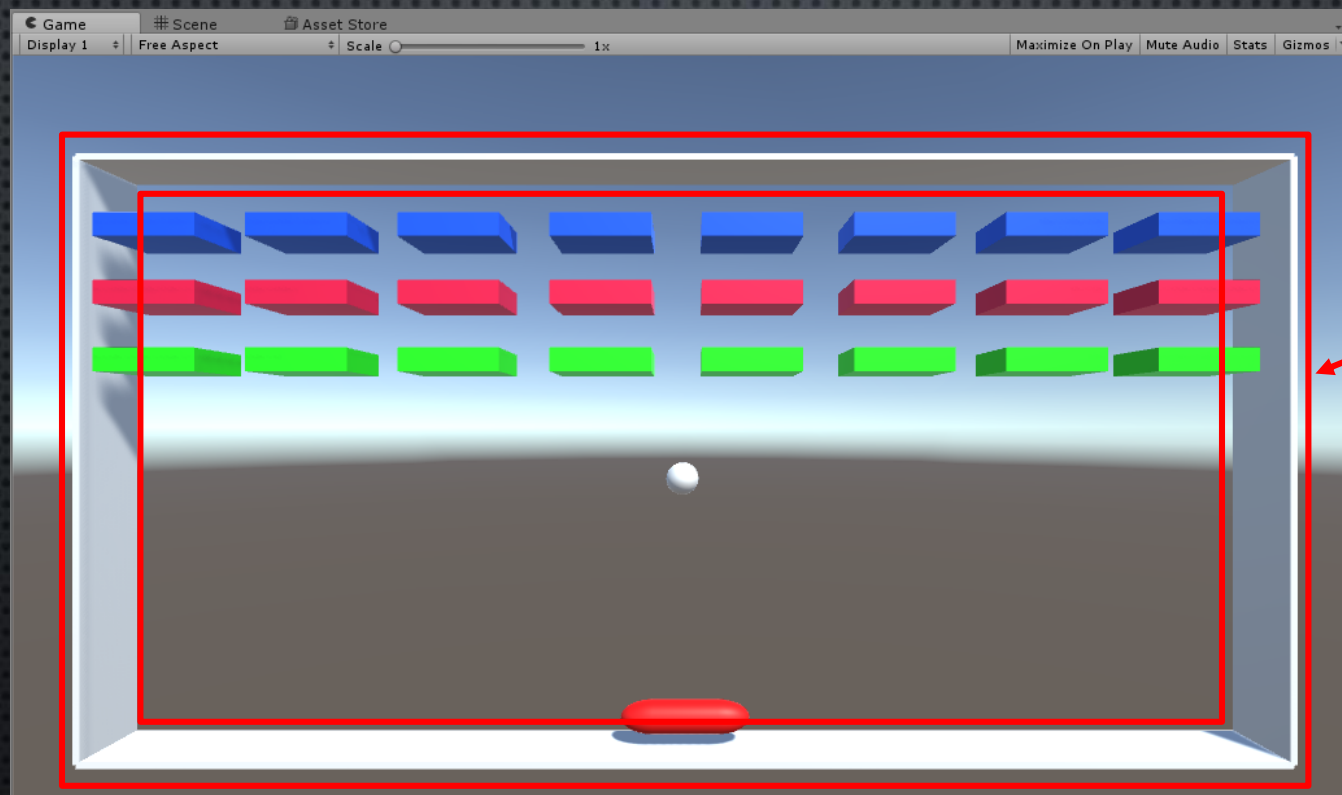
これが基準になっていくので後から変えると全部変えることになる

Hierarchyの「Main Camera」をクリック(選択)し、Inspectorを下のようにする

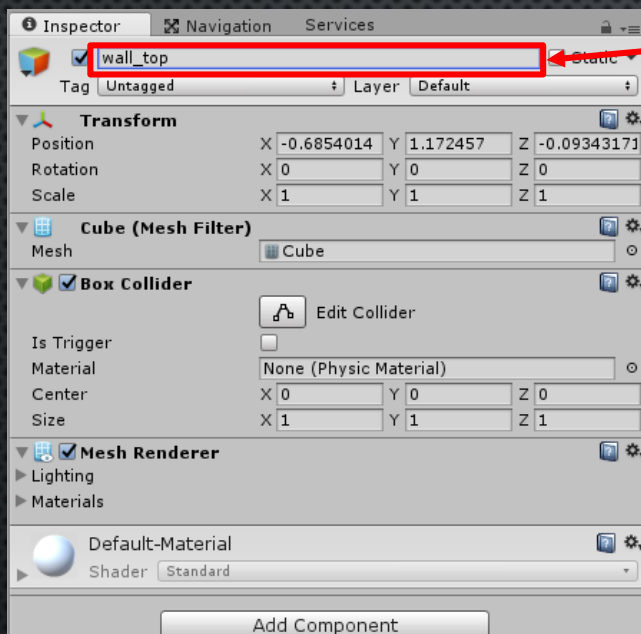


変更点

壁の作成

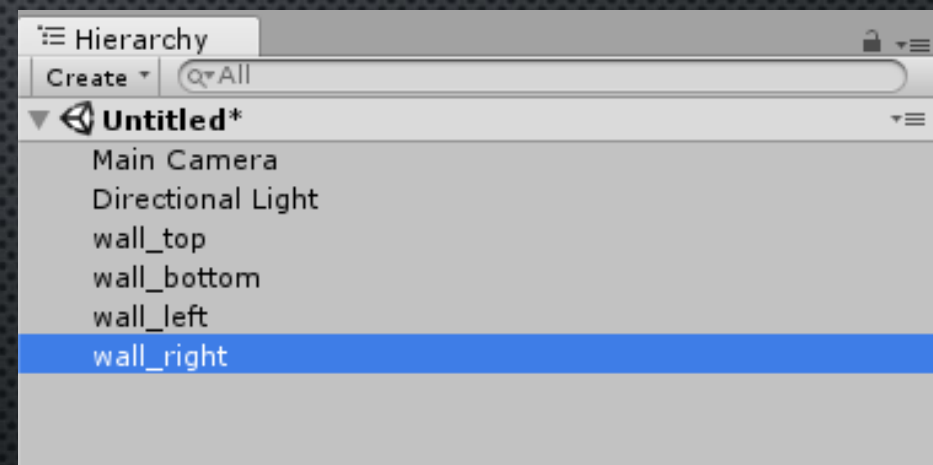


先ほどと同じように
GameObject>3D Object>Cube
でCubeオブジェクトを4つ作る
それぞれ名前を
「wall_top」, 「wall_bottom」, 「wall_left」, 「wall_right」
にする



ここで名前を変更できる

または
オブジェクトを選択して
「F2キー」でも行ける

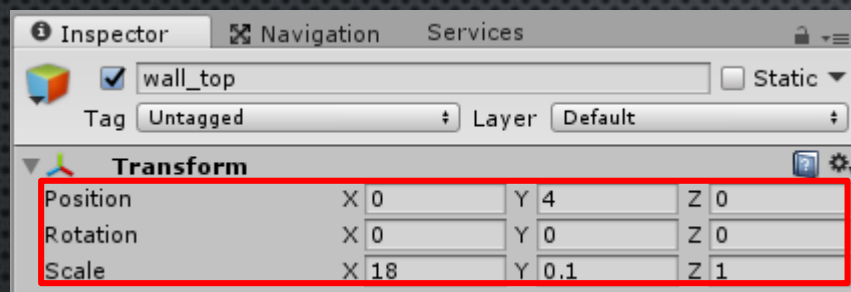


制作を楽にするワンポイント :

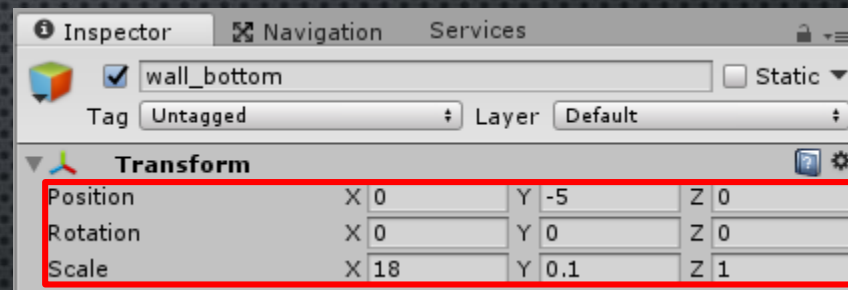
オブジェクトを選択して
「Ctrl+c」でコピー
「Ctrl+v」で貼り付け

各オブジェクトの座標、大きさを設定する

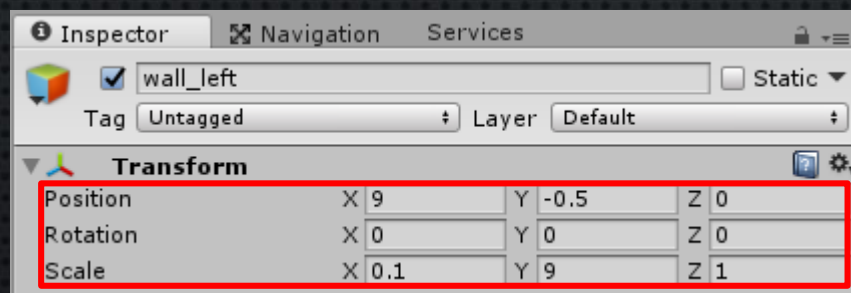
wall_top



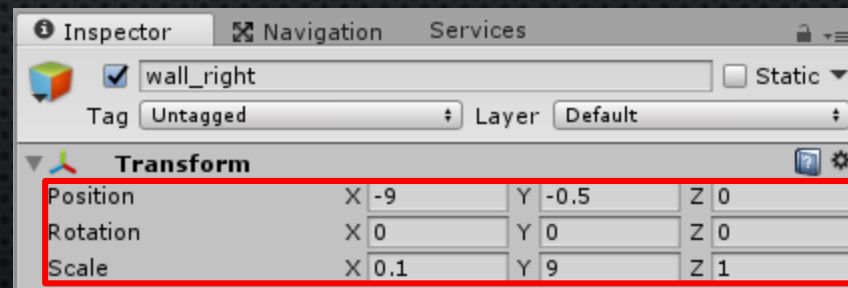
wall_bottom



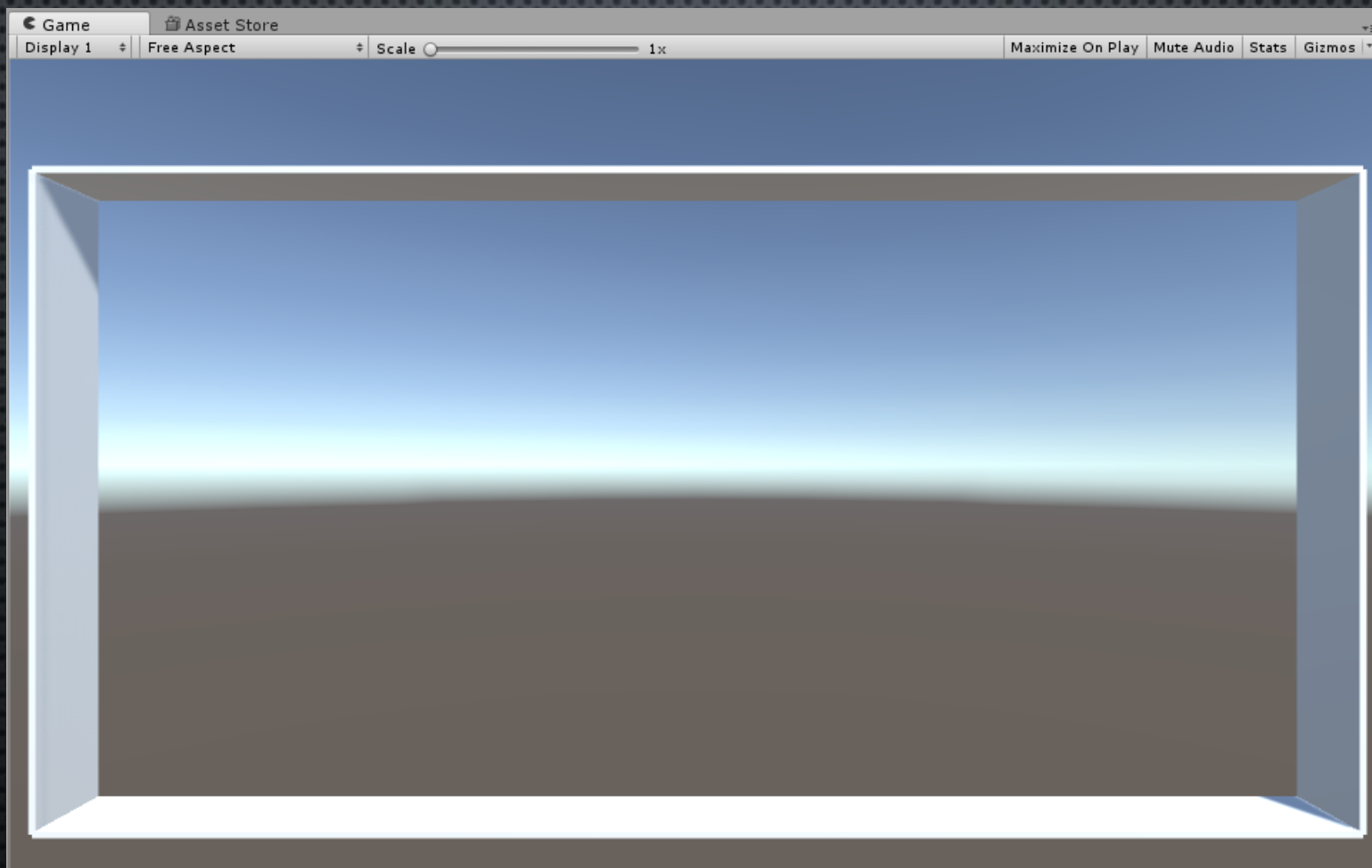
wall_left



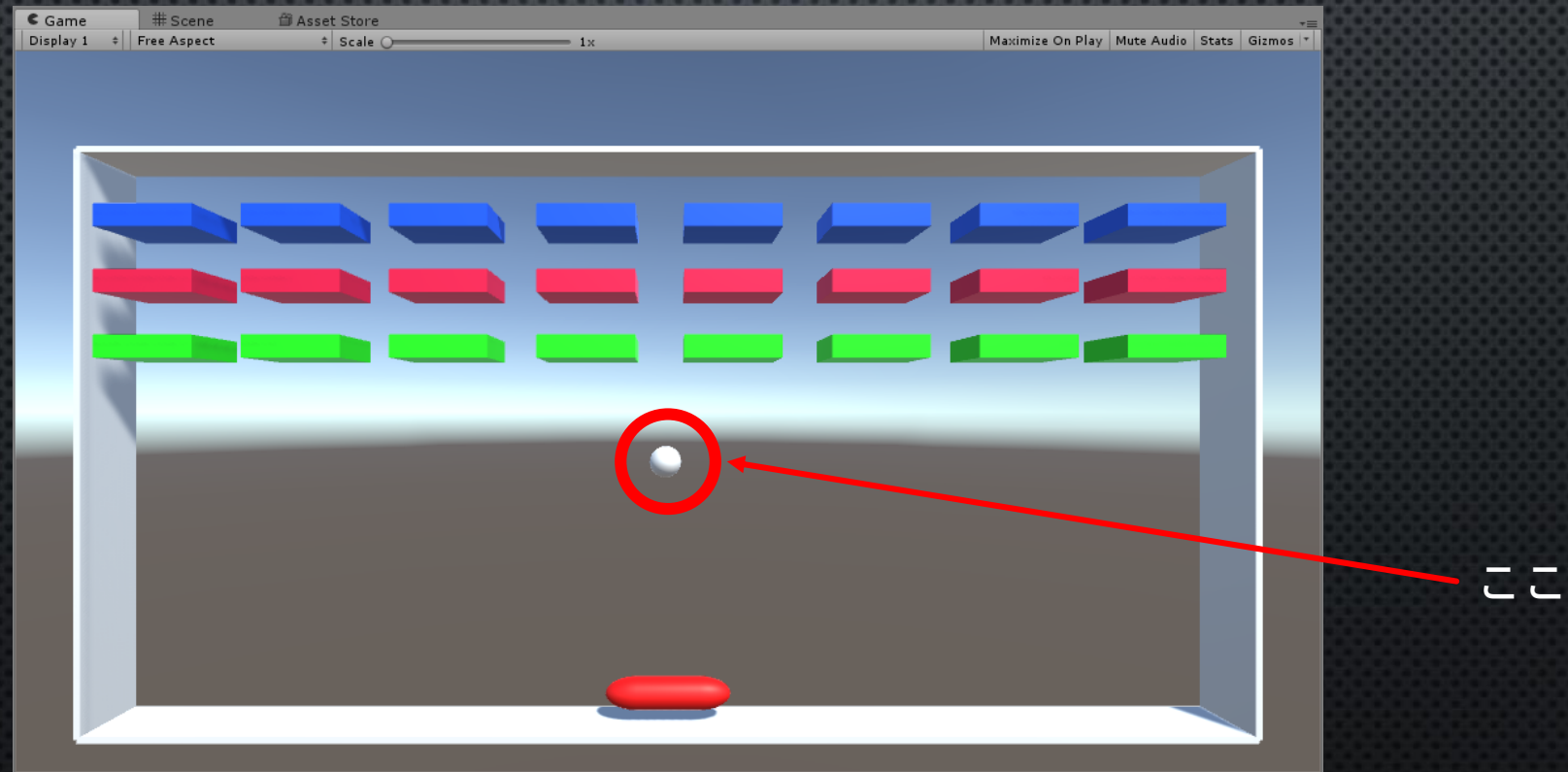
wall_right



こんな感じの壁ができれば完成



ボールの作成

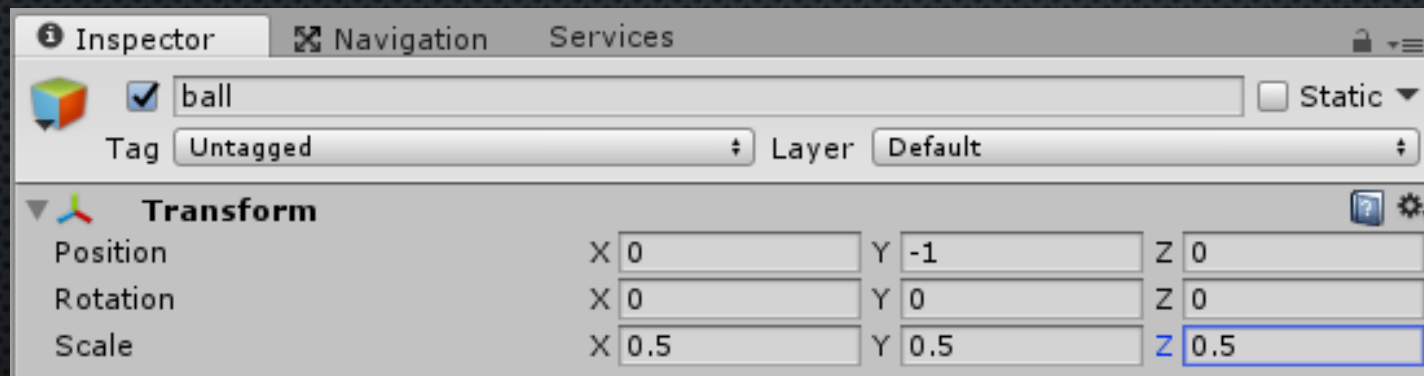


球体は

GameObject>3D Object>Sphere

で出てくる

名前は「ball」、座標と大きさは下のようにする

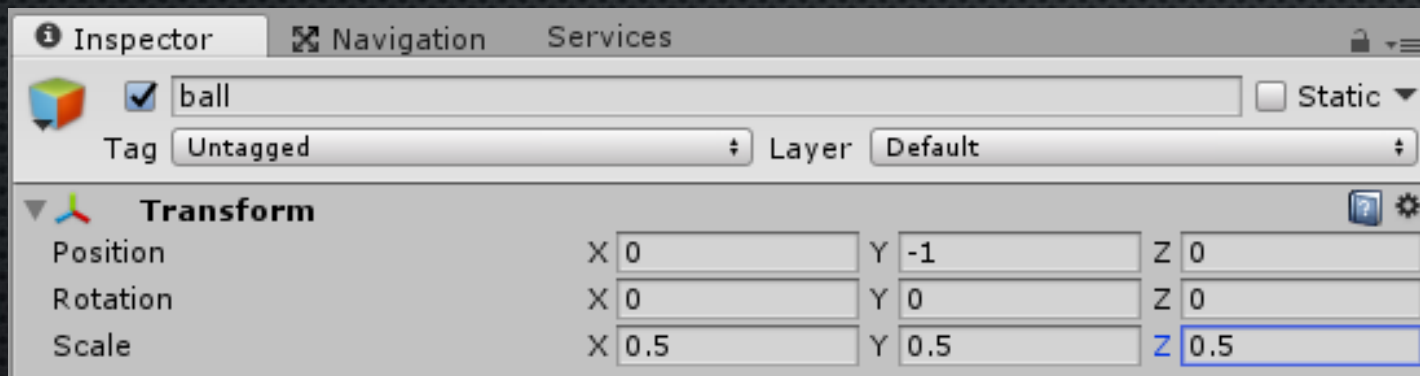


球体は

GameObject>3D Object>Sphere

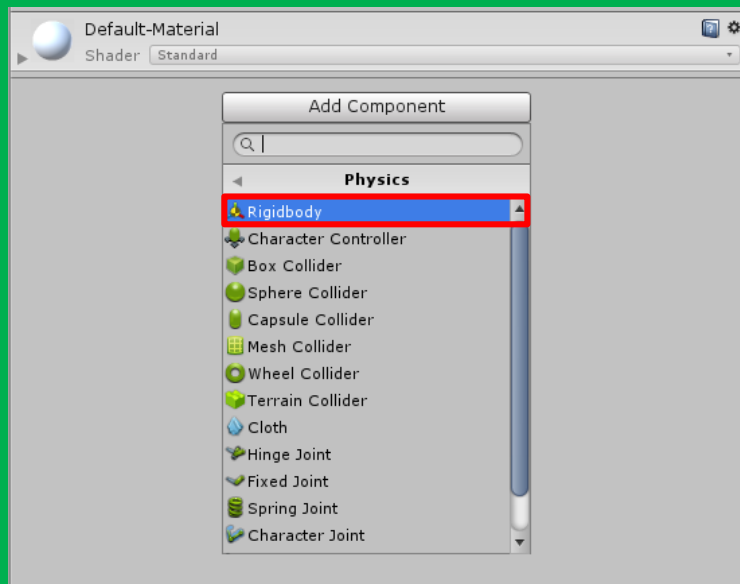
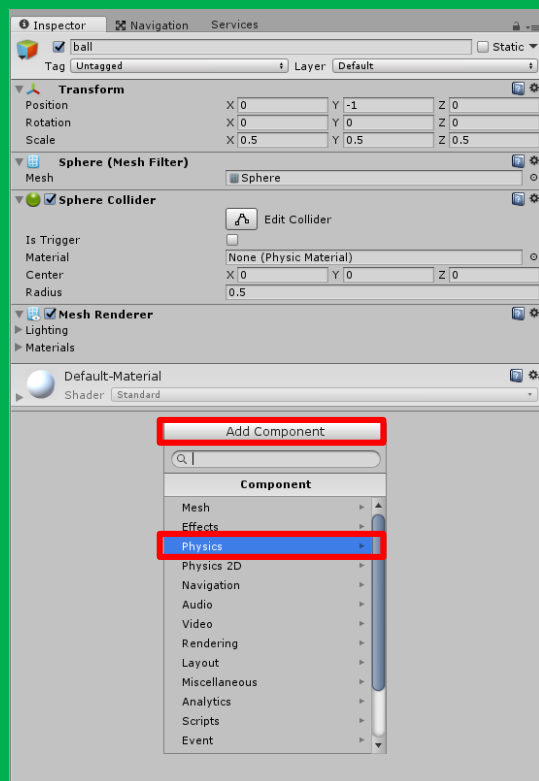
で出てくる

名前は「ball」、座標と大きさは下のようにする



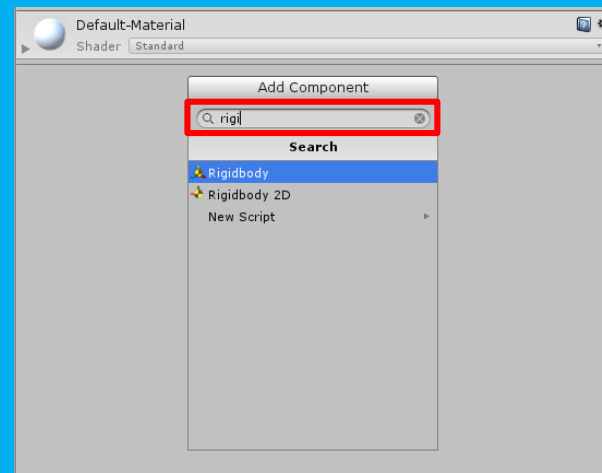
この状態だと宙に浮くだけで動かないので
物理法則を与えます(すごい(小並感))

ballを選択してInspector内の「Add Component」を
クリック>Physics>Rigidbodyを選択

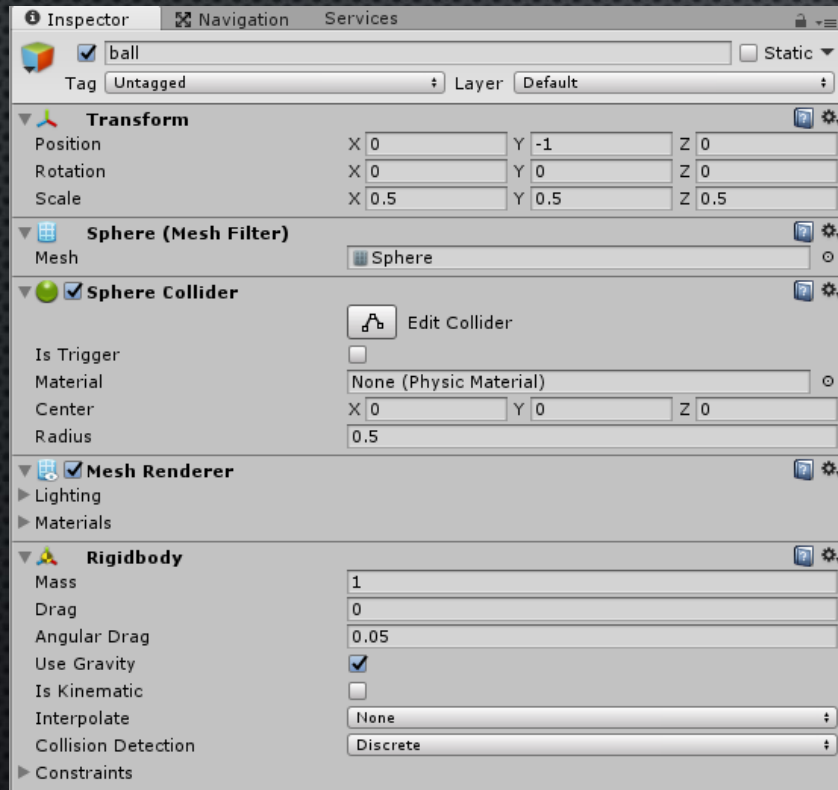


ちなみに…

上のQから検索することもできる
(名前知ってるならこっちのが楽)

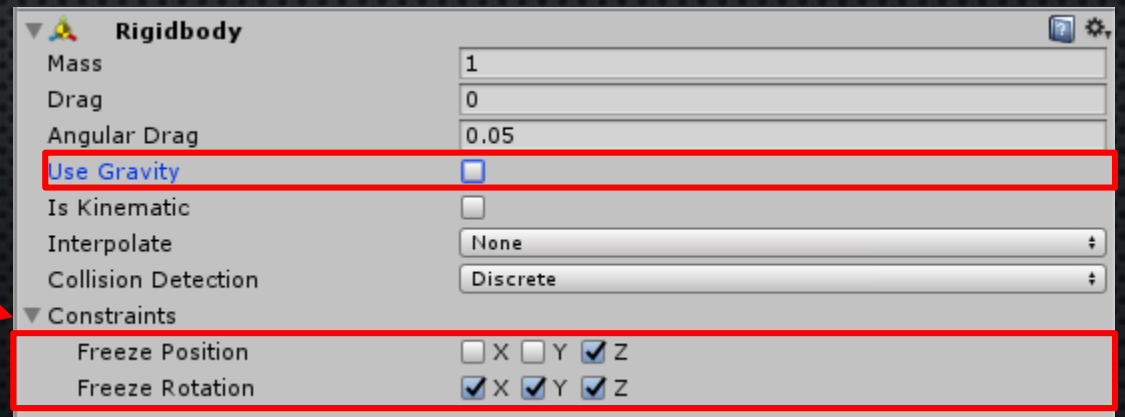


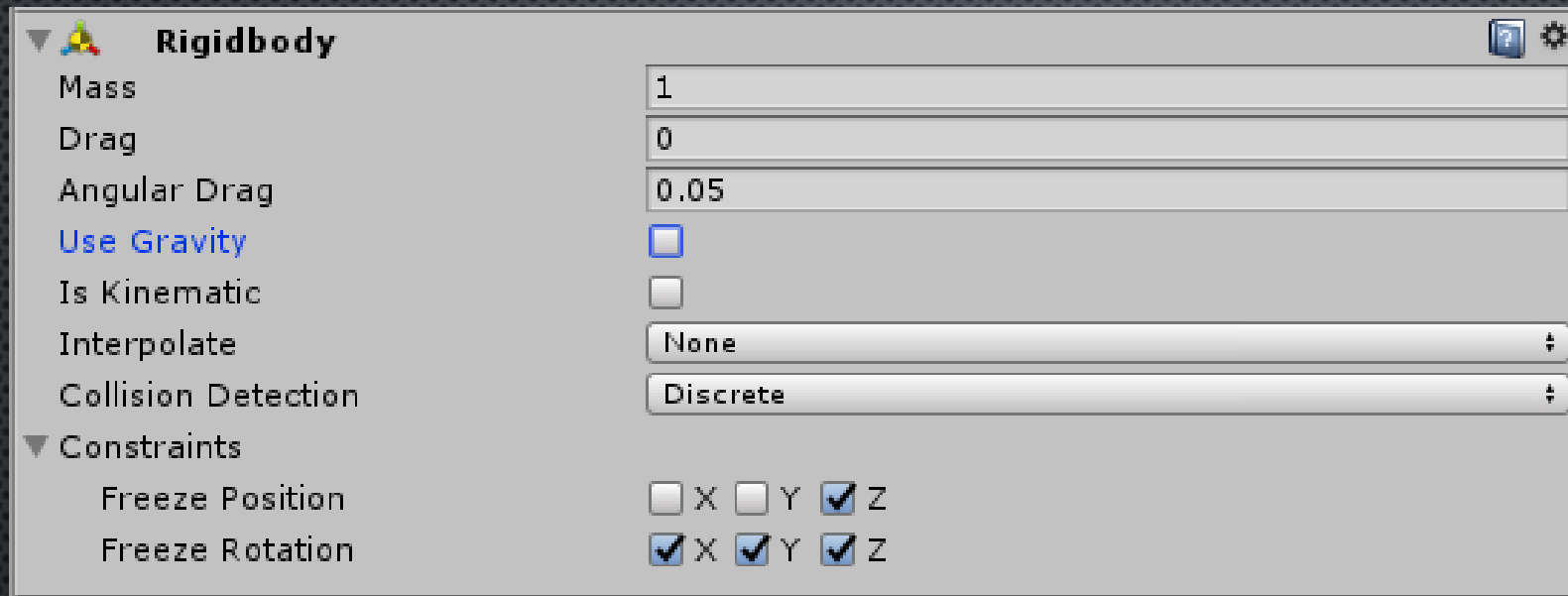
するとInspector内に「Rigidbody」が追加されます



「Use Gravity」のチェックを外し
「Freeze Position」の「Z」
「Freeze Rotation」の「X」「Y」「Z」
にチェックを入れる

クリック





Mass . . . 質量

Drag . . . 空気抵抗

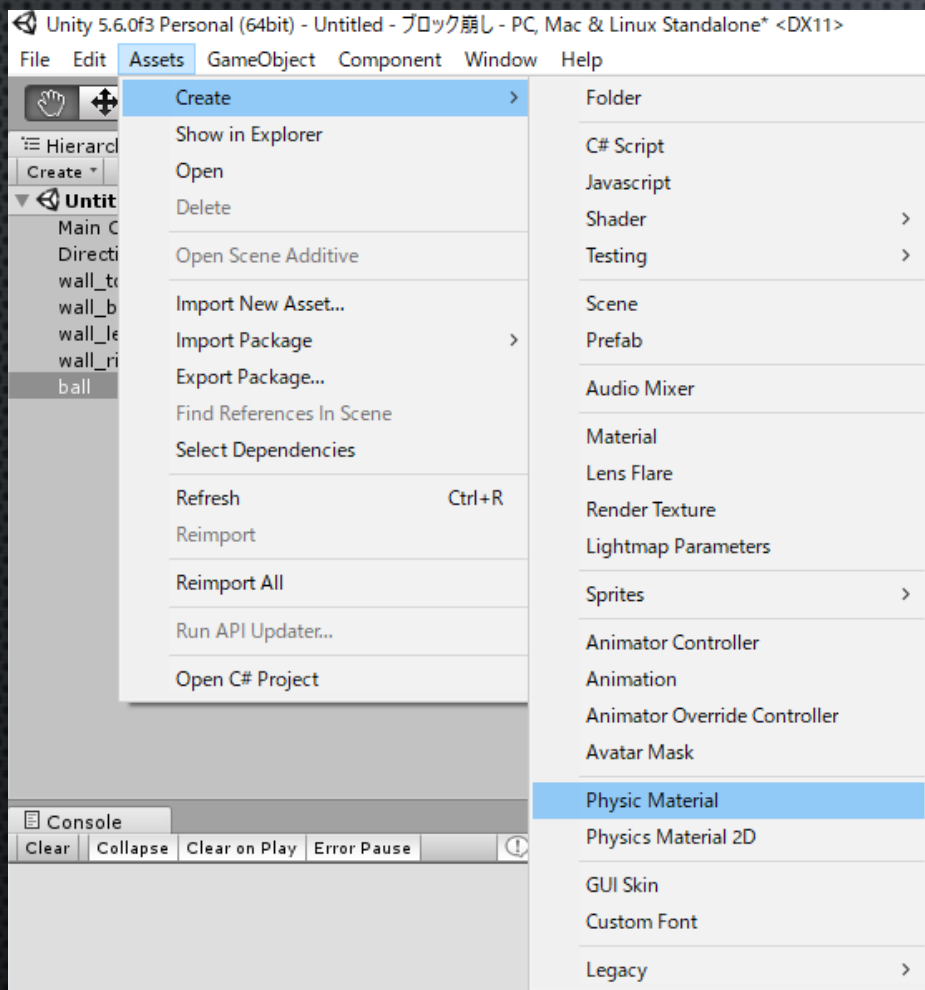
Use Gravity . . . 重力の有無

Constraints . . . それぞれposition, rotationを固定するかしないか

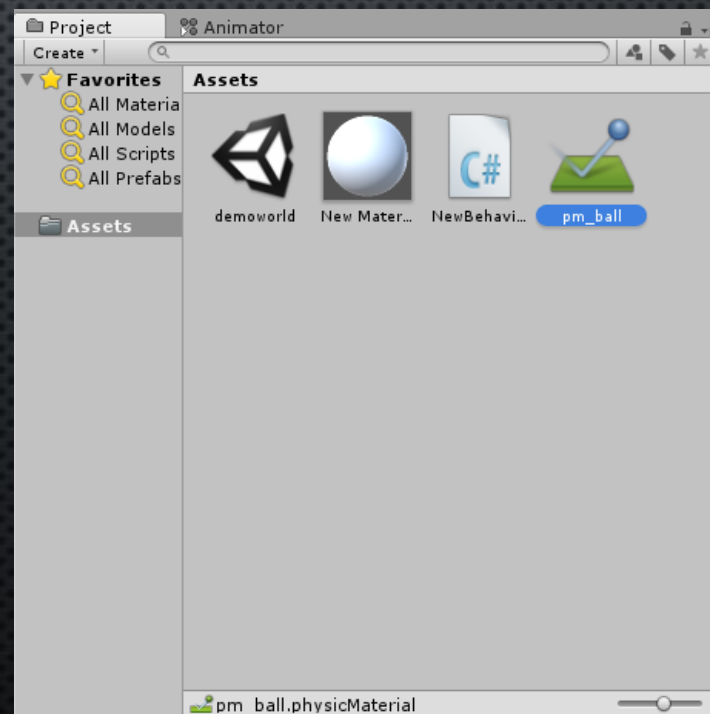
などなどいろいろあるが詳しくは下記のサイトに説明が載ってる

<https://docs.unity3d.com/ja/current/Manual/class-Rigidbody.html>

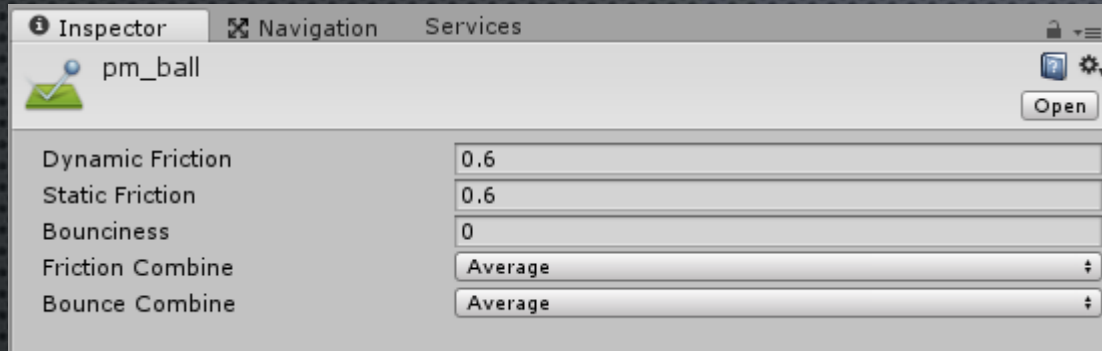
今度はボールに「摩擦力」や「反発力」といった物理演算を追加します(そんなこともできるのかっ!!)!



左上のメニューバーから
Assets>Create>Physics Material
名前を「pm_ball」に

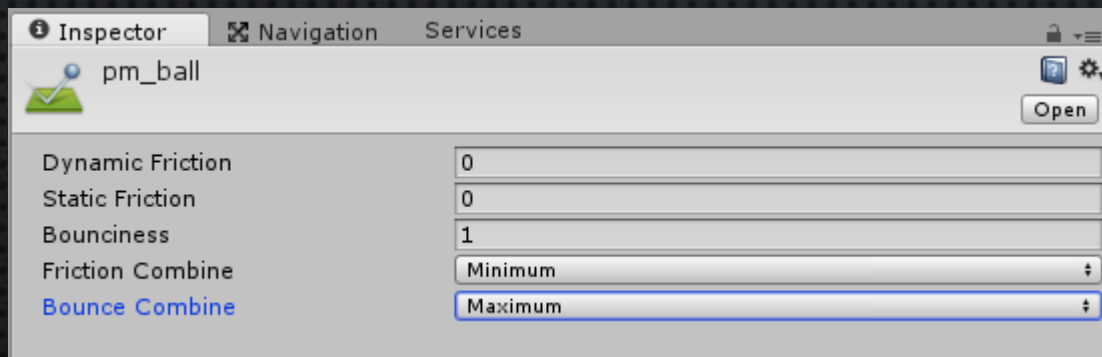


pm_ballを選択するとInspectorにこんなのが表示される

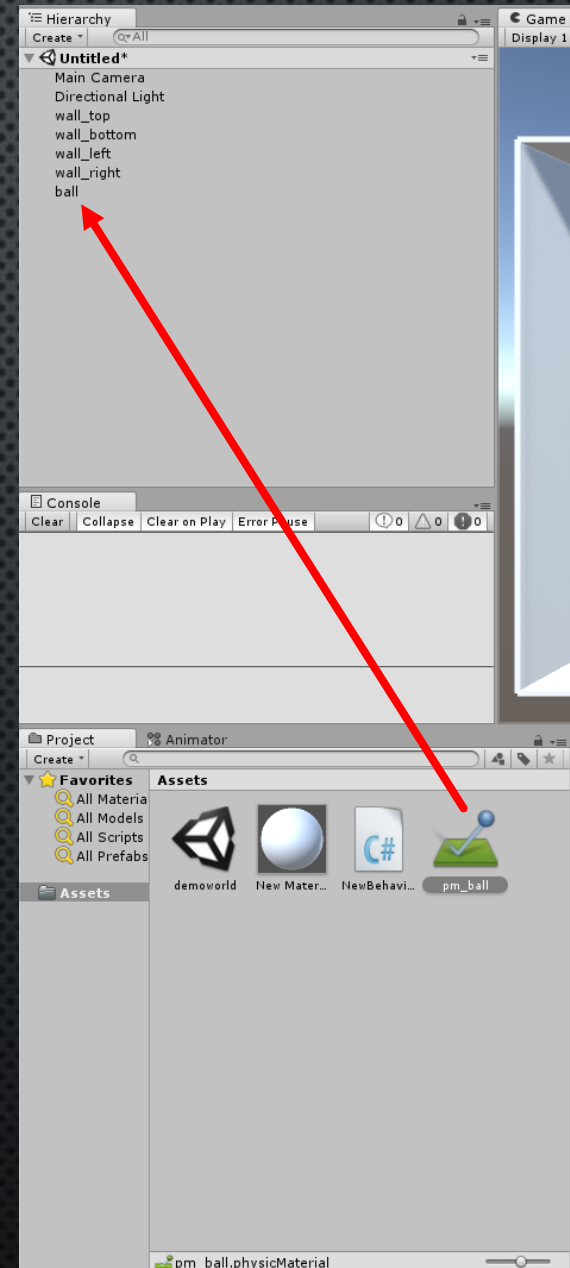


- Dynamic Friction → 動摩擦力
- Static Friction → 静止摩擦力
- Bounciness → 弾性力
- Friction Combine → 衝突時の物体間の摩擦力
- Bounce Combine → 衝突時の物体間の弾性力

こんな感じに値を変更



Assets内の「pm_ball」をHierarchy内の「ball」に
ドラッグ&ドロップ

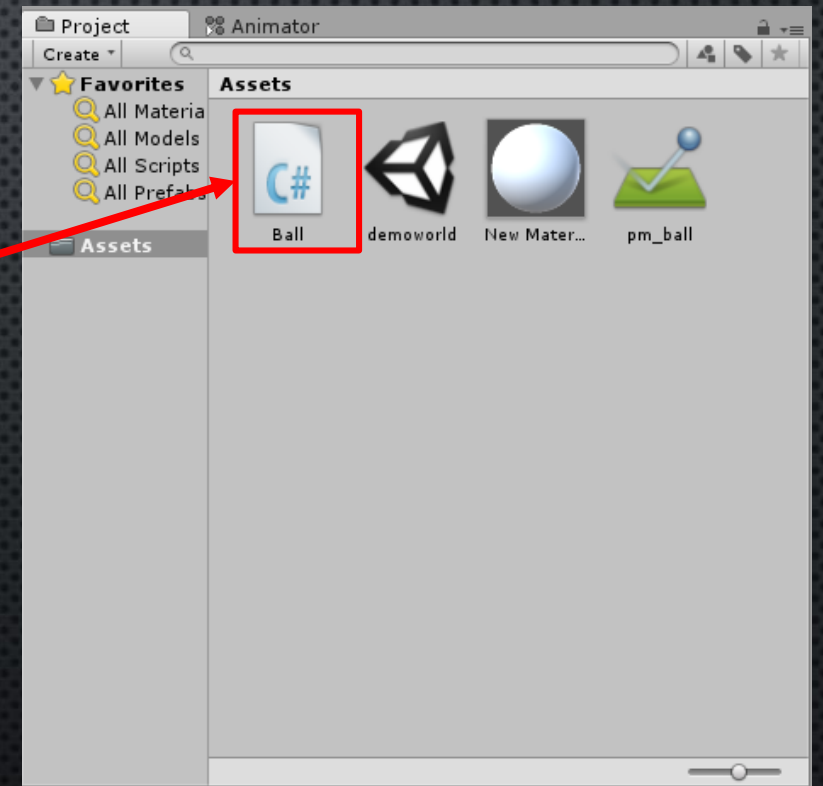


いよいよ玉を動かすプログラムを書いてみよう

メニューバーから
Assets>Create>C# Script
を選択

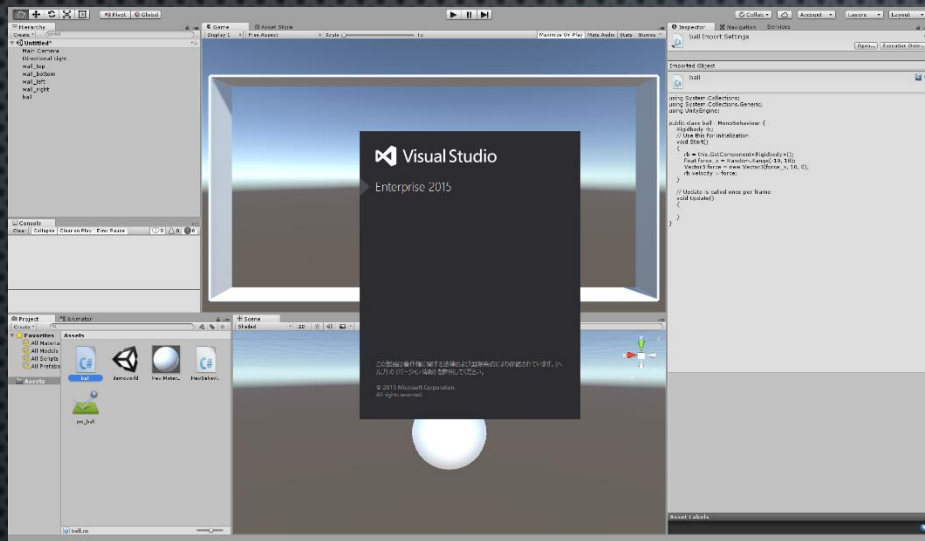
または
Project内で
右クリック>Create>C# Script

名前を「Ball」にする



※先ほどのpm_ballのようにBallにドロップ&ドロップ

ballスクリプトをクリックすると
Visual Studioが立ち上がる(はず)



こんな感じにコードを追加

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

public class ball : MonoBehaviour {
    Rigidbody rb;
    // Use this for initialization
    void Start()
    {
        rb = this.GetComponent<Rigidbody>();
        float force_x = Random.Range(-10, 10);
        Vector3 force = new Vector3(force_x, 10, 0);
        rb.velocity = force;
    }

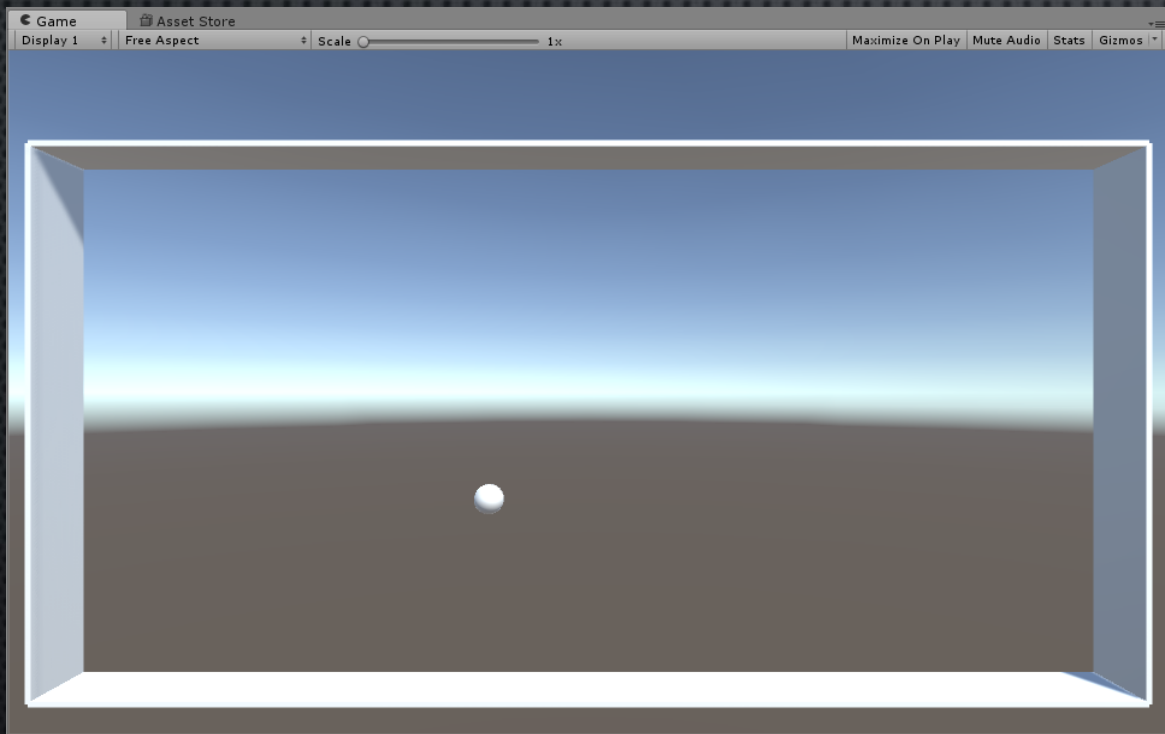
    // Update is called once per frame
    void Update()
    {

    }
}
```

追加

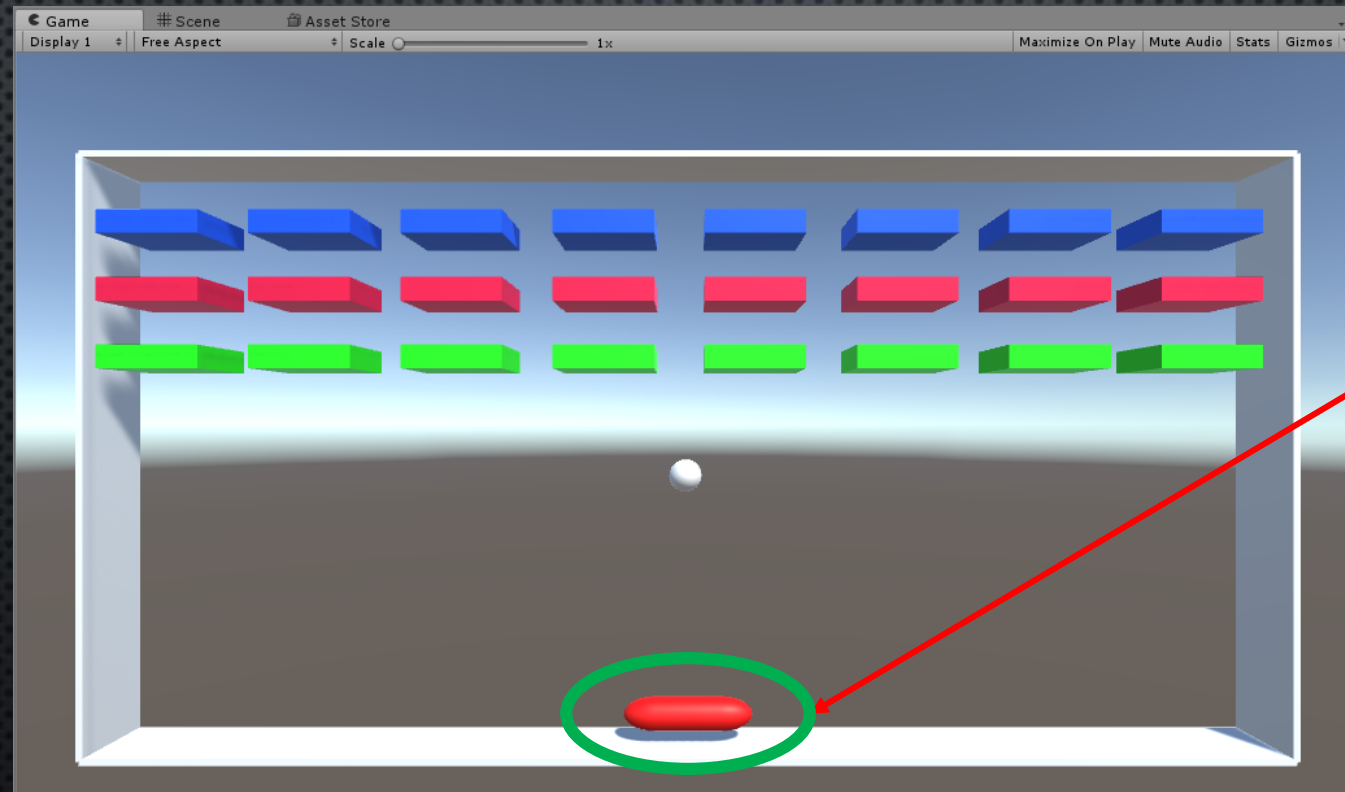
実際に動かしてみよう！

先ほど教えた再生ボタンを押してゲームを開始してみよう！



ボールが勝手に動き出して
壁に当たって跳ね返っている

バーの作成



Capsule



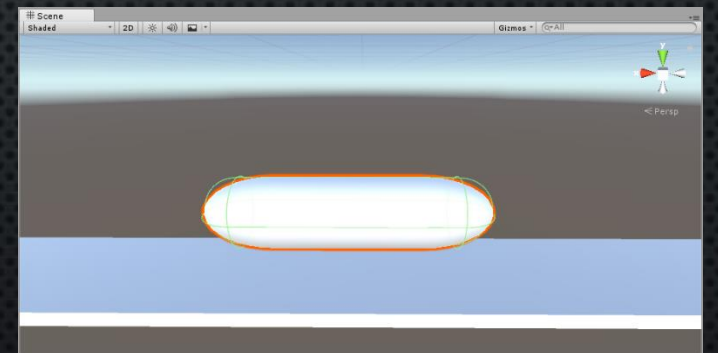
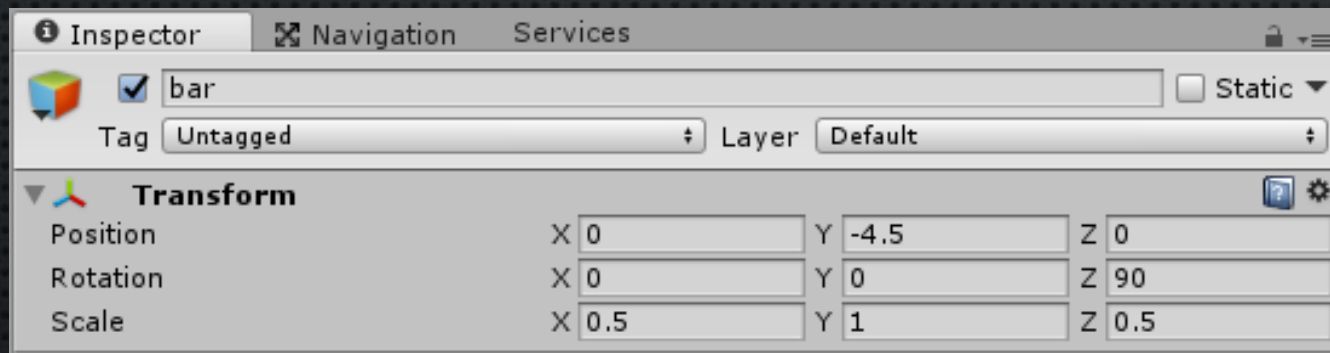
バーはカプセルタイプのオブジェクトを変形させて使う

カプセルは

GameObject>3D Object>Capsule

で出てくる

名前は「bar」、座標と大きさは下のようにする



バーを動かすプログラム

ボールの時と同じように
C# Scriptを作る
名前を「Bar」にする

同じくBarスクリプトをbarオブジェクトにドロップ&ドロップする

コードを追加

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

public class Ball : MonoBehaviour {

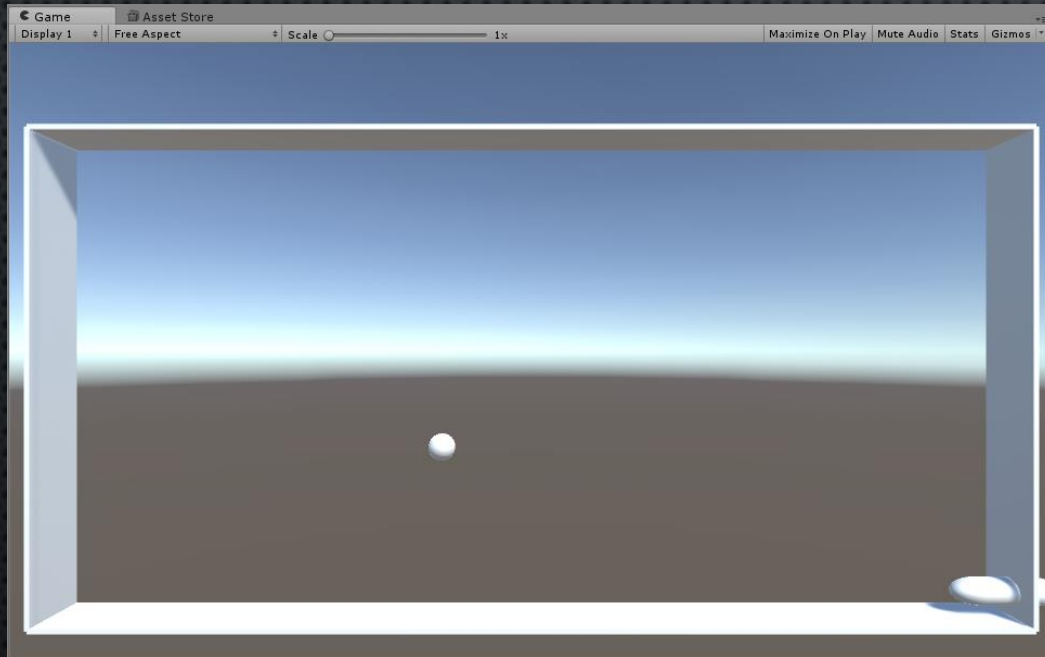
}

// Use this for initialization
void Start()
{

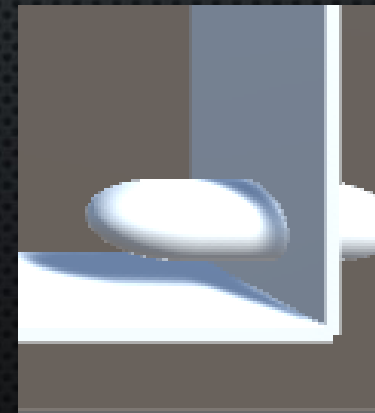
}

// Update is called once per frame
void Update()
{
    if (Input.GetKey(KeyCode.D) || Input.GetKey(KeyCode.RightArrow))
    {
        transform.position += new Vector3(10, 0, 0) * Time.deltaTime;
    }
    else if (Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.LeftArrow))
    {
        transform.position += new Vector3(-10, 0, 0) * Time.deltaTime;
    }
}
}
```

ちゃんと動作するか実際にプレイして確認
バーがちゃんと移動してればOK！



このままだと壁を貫通してしまうのでまずい



壁を貫通しないための関数 (limit_area) を追加①

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

public class Bar : MonoBehaviour {
    float[] limit_area = new float[2];
    float bar_width;
    void limit_area_over()
    {
        Vector3 pos = transform.position;
        if (this.transform.position.x - bar_width < limit_area[0])
        {
            pos.x = limit_area[0] + bar_width;
            transform.position = pos;
        }
        else if (this.transform.position.x + bar_width > limit_area[1])
        {
            pos.x = limit_area[1] - bar_width;
            transform.position = pos;
        }
    }
}
```

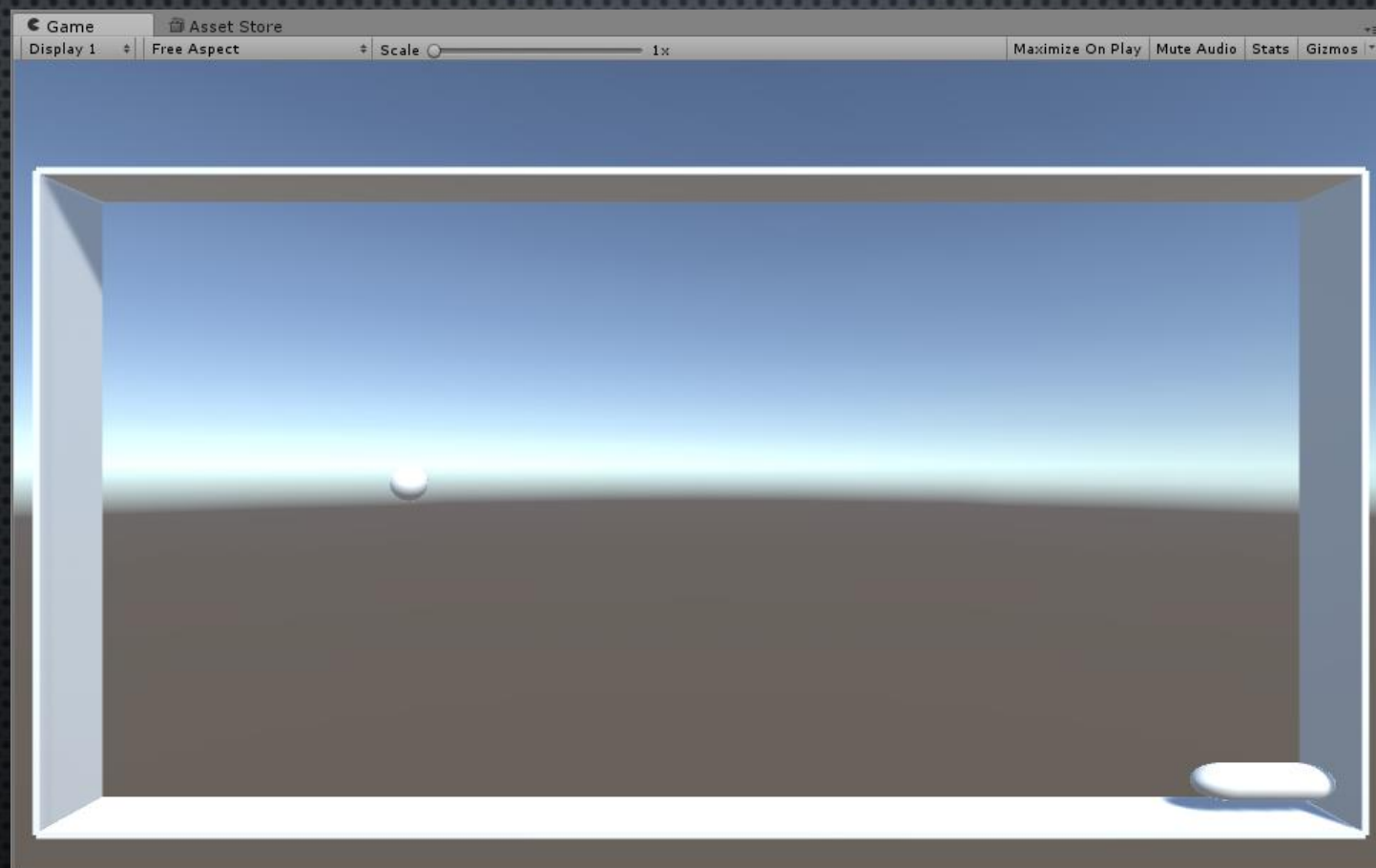
壁を貫通しないための関数 (limit_area) を追加②

```
// Use this for initialization
void Start()
{
    limit_area[0] = GameObject.Find("wall_left").transform.position.x;
    limit_area[1] = GameObject.Find("wall_right").transform.position.x;

    bar_width = this.gameObject.transform.localScale.y;
}

// Update is called once per frame
void Update()
{
    if (Input.GetKey(KeyCode.D) || Input.GetKey(KeyCode.RightArrow))
    {
        transform.position += new Vector3(10, 0, 0) * Time.deltaTime;
    }
    else if (Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.LeftArrow))
    {
        transform.position += new Vector3(-10, 0, 0) * Time.deltaTime;
    }
    limit_area_over();
}
}
```

これで壁を貫通する問題は解決！



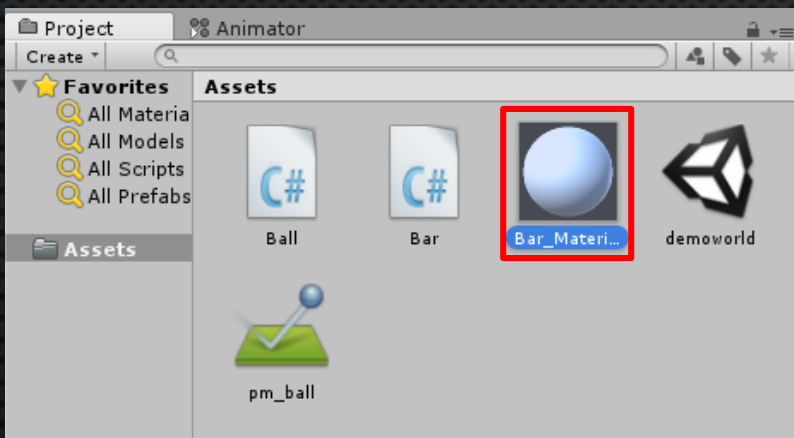
Barと壁の色が同じで見づらいので

次はバーの色を変えていきます

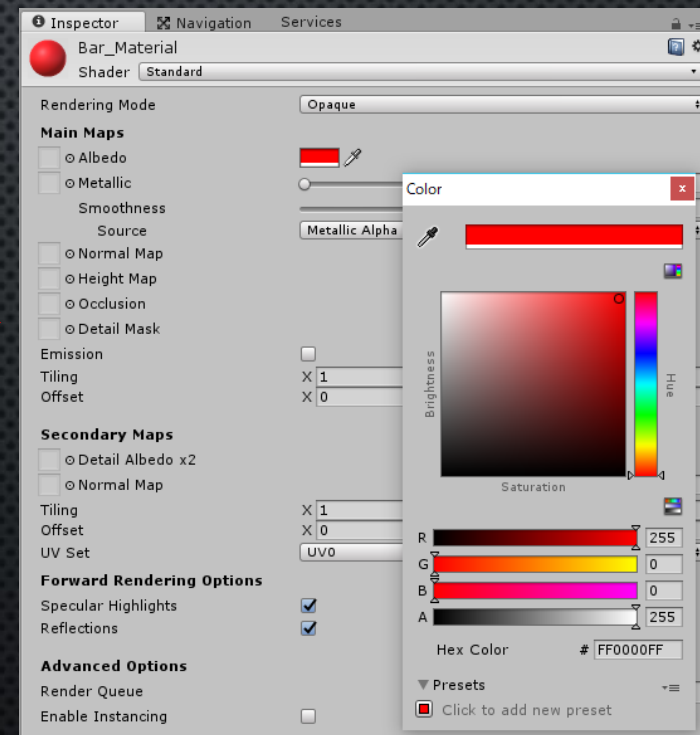
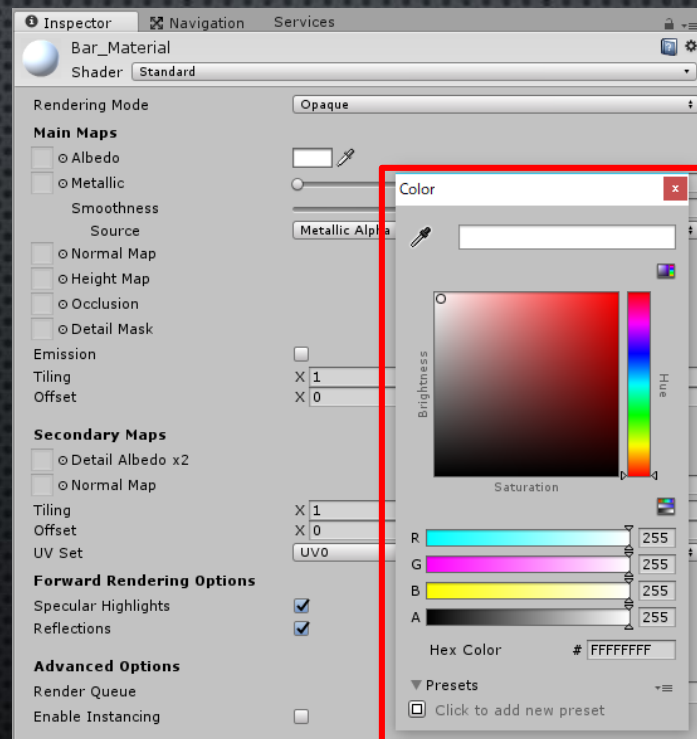
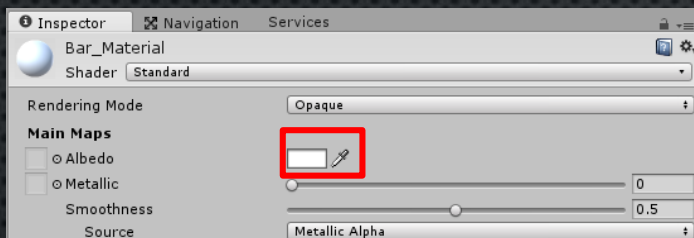
Physic MaterialやC#Scriptを作った時と同じように

Assets>Create>Material（右クリックも同様）

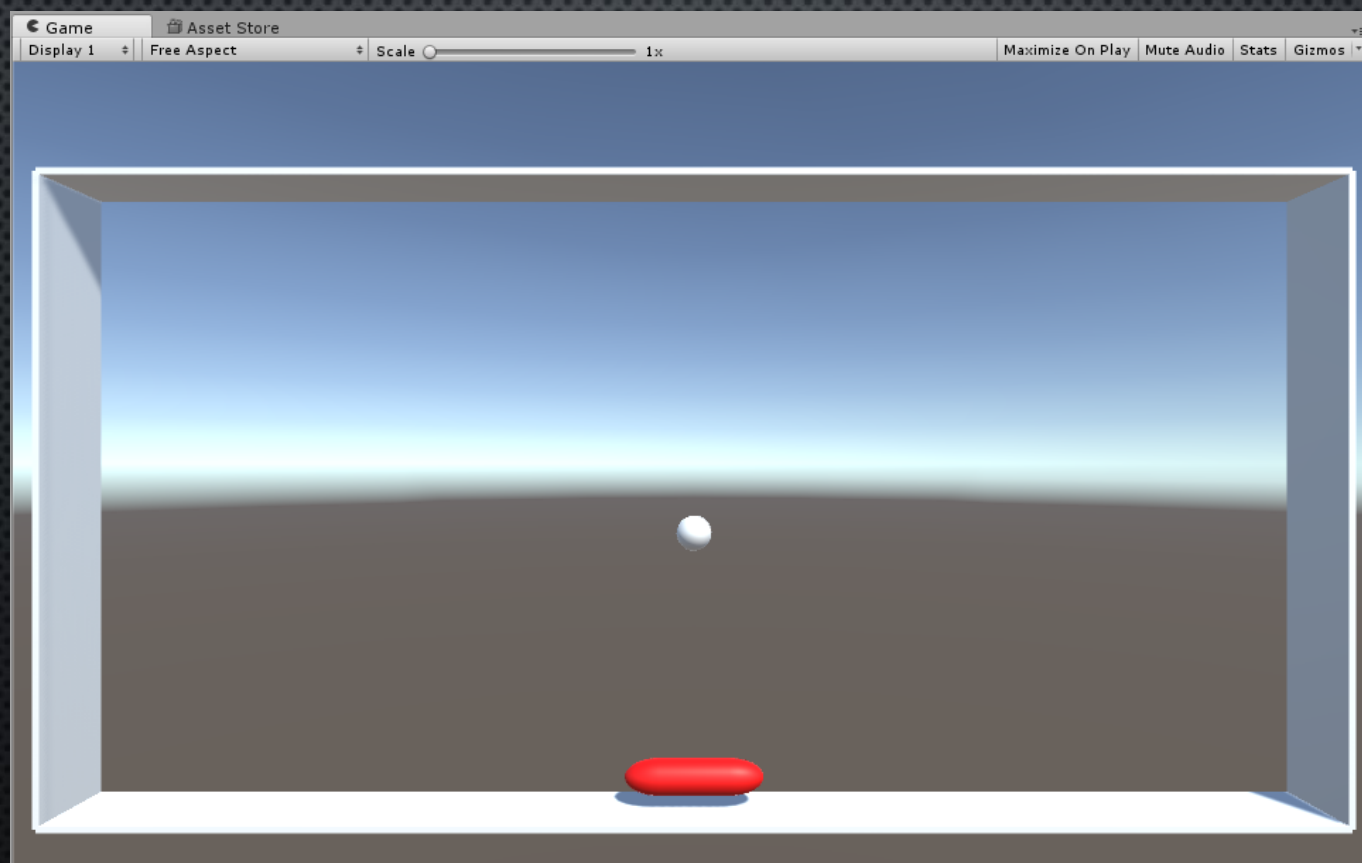
名前を「Bar_Material」にしてbarオブジェクトにドラッグ&ドロップ



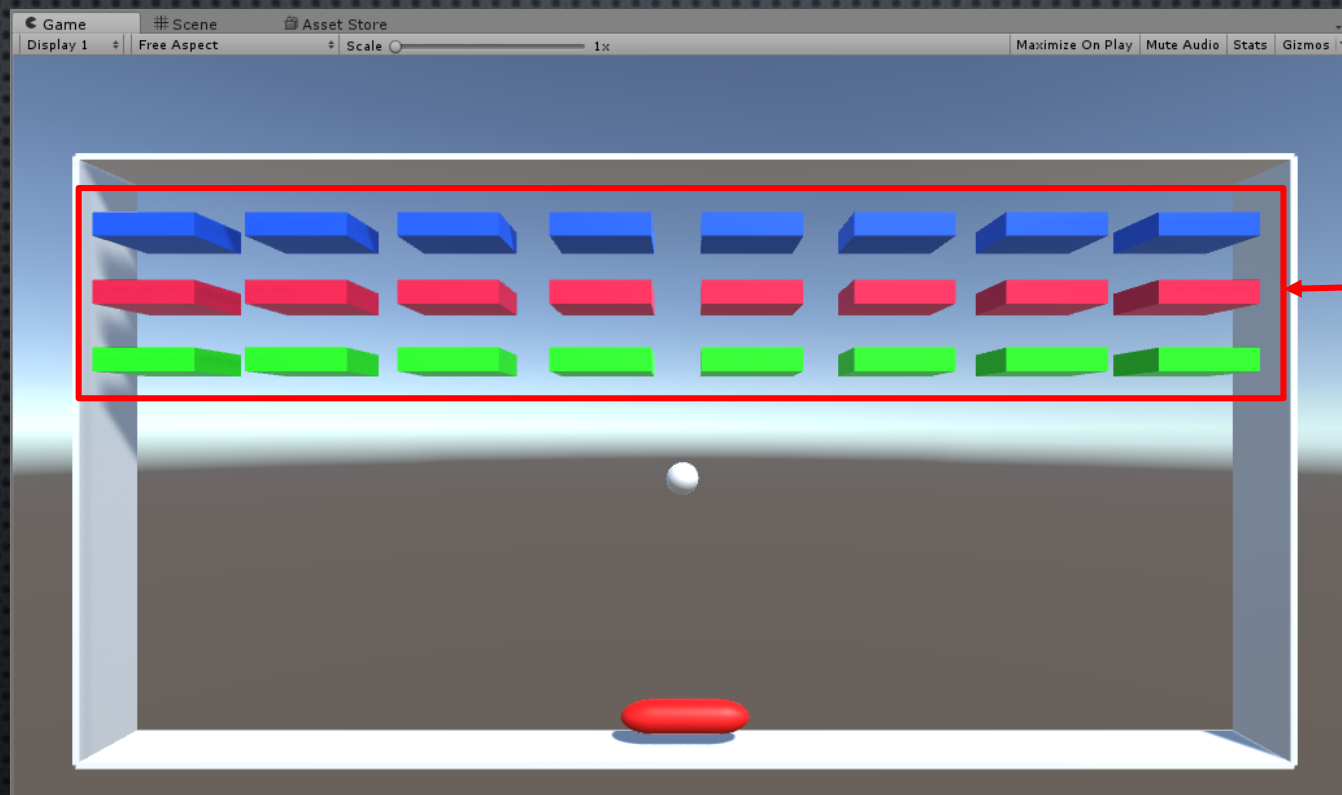
1. Bar_Materialを選択し、Inspector内のMainMapsの下にある「Albedo」を選択
2. 皆さんのお好みの色に設定してください



色が変わった！



ブロックの作成

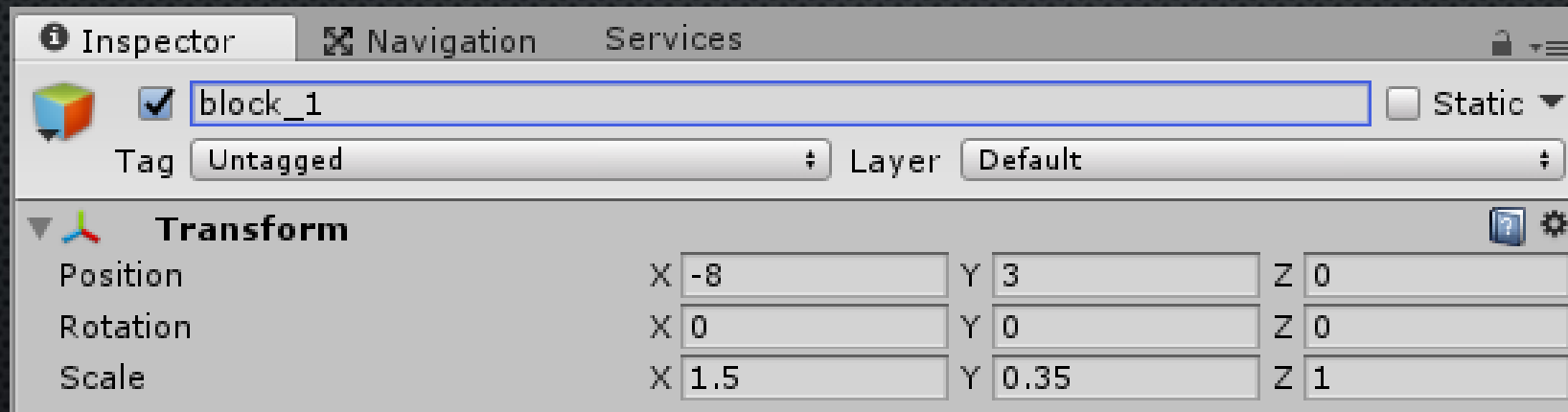


11 11

Cubeオブジェクトを作り、座標と大きさを下のようにする

名前は「block_1」にする

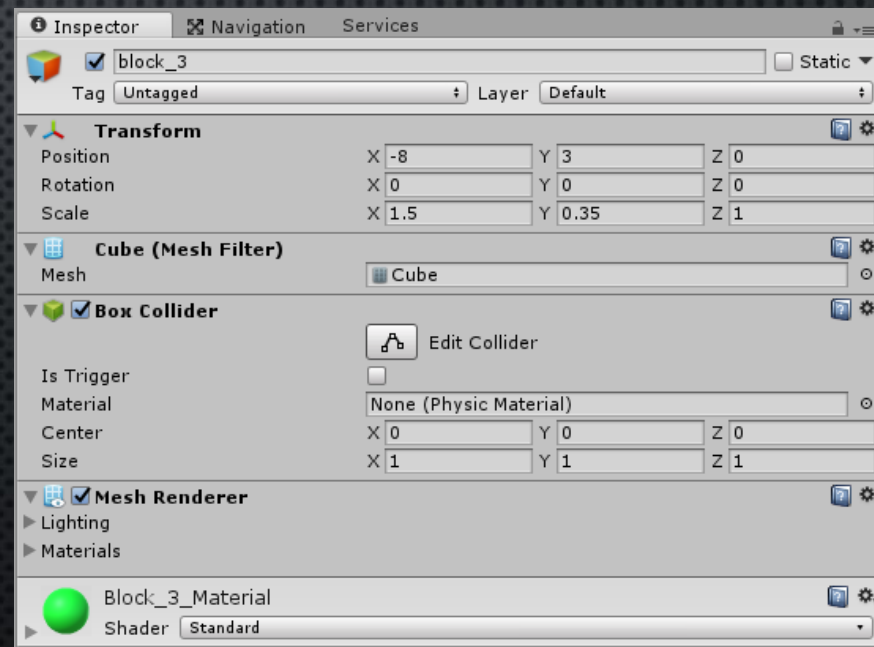
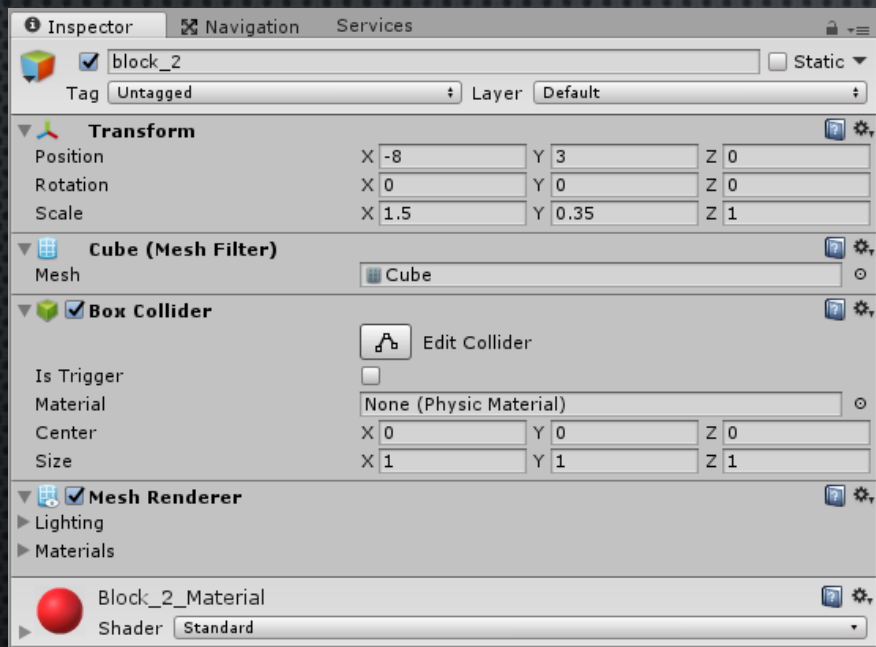
色は先ほどバーとやったようにMaterialを作り、好きな色を付けてください



今の工程をblock_1とは別で

「block_2」, 「block_3」という名前で新しく作ってください

(Materialも「Block_2_Material」「Block_3_Material」という名前で好きな色で作ってください)

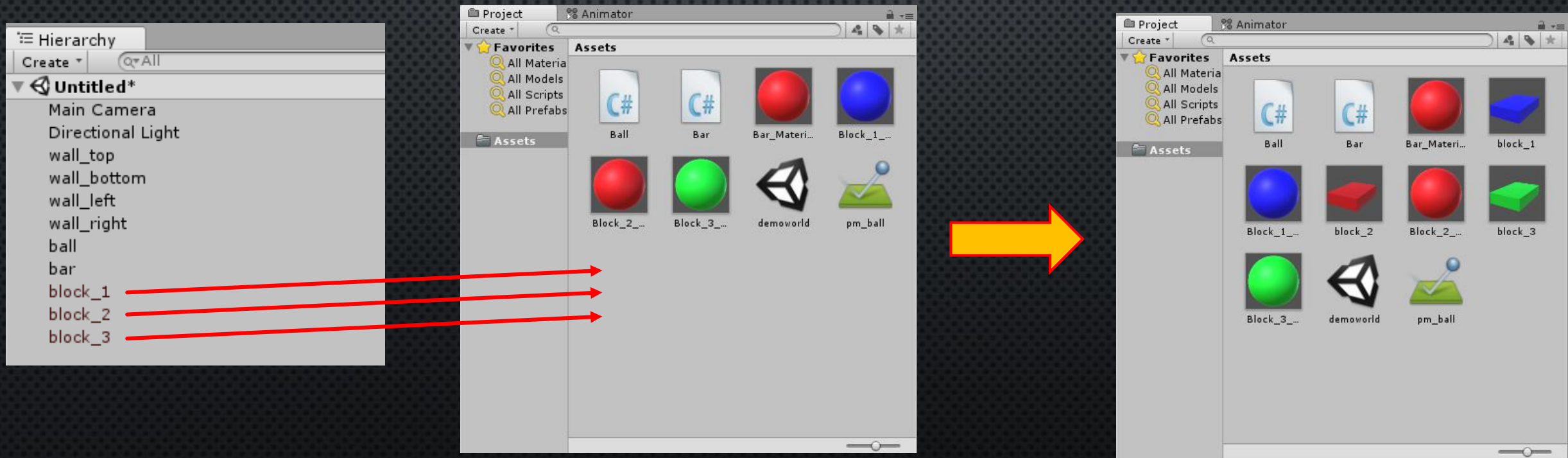


Prefab化

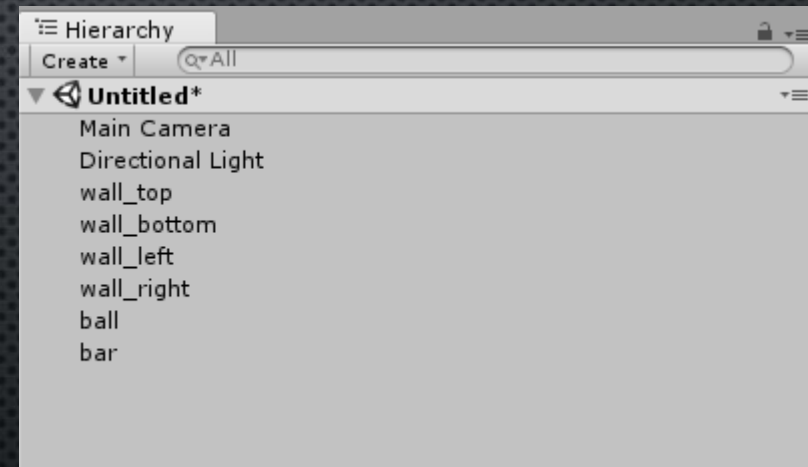
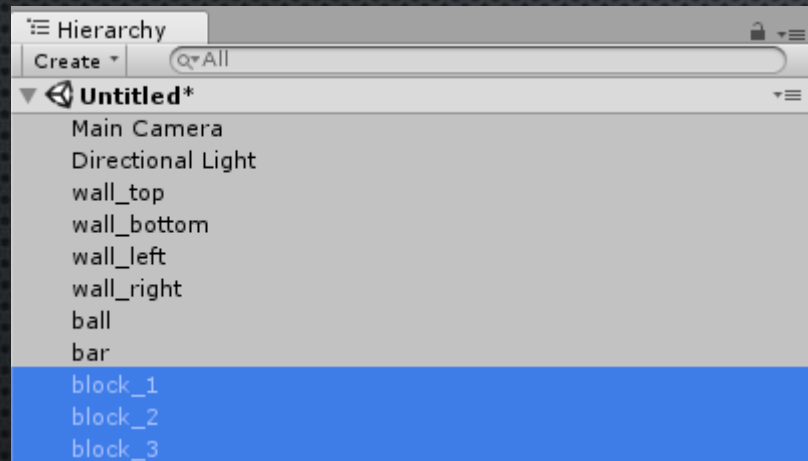
Hierarchyにあるblock_1,block_2,block_3をProject内にドラッグ&ドロップ
これをPrefab化(プレハブ化)という

同じオブジェクトを大量に生成したり、使う時は便利なので積極的に使っていきましょう

(これを使いこなせるかで作業効率がかなり変わります)



Prefab化した後は元になったオブジェクトは破棄しましょう



プログラムからオブジェクトを生成

手動でブロックを配置するのは面倒なのでプログラムからオブジェクトを生成→設置しましょう

「Create_Blocks」という名前のC#Scriptを作ってください

コードを追加①

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

public class Create_Blocks : MonoBehaviour {
    public GameObject[] prefabs;
    GameObject block;
    float block_width;
    float block_height;
    int create_count_x = 8;
    int create_count_y = 3;
    float[] create_range_x = { -9, 9 };
    float distance_x;
    float[] create_range_y = { 3, 0 };
    float distance_y;
    // Use this for initialization
```

コードを追加②

```
// Use this for initialization
void Start () {
    block_width = prefabs[0].transform.localScale.x;
    block_height = prefabs[0].transform.localScale.y;
    distance_x = (Mathf.Abs(create_range_x[0] - create_range_x[1]) - create_count_x * block_width) / create_count_x;
    distance_y = (Mathf.Abs(create_range_y[0] - create_range_y[1]) - create_count_y * block_height) / create_count_y;

    int prefabs_number = 0;
    Vector3 pos = prefabs[prefabs_number].transform.position;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 8; j++) {
            block = GameObject.Instantiate(prefabs[prefabs_number], pos, Quaternion.identity);
            pos.x += block_width + distance_x;
        }
        pos.x = prefabs[prefabs_number].transform.position.x;
        pos.y -= block_height + distance_y;
        prefabs_number += 1;
    }
}

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

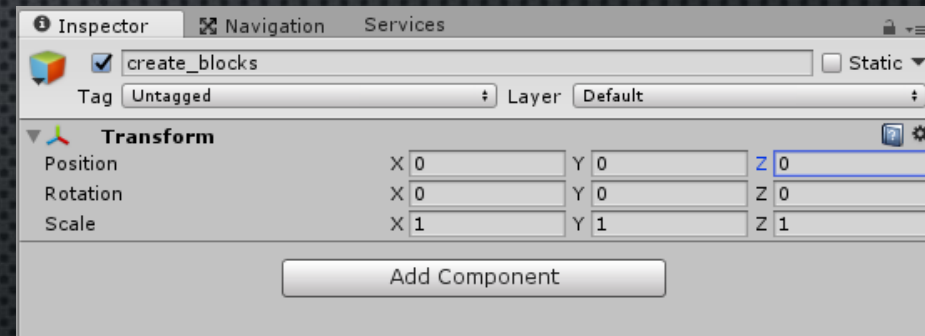
Create_Blockを持たせるオブジェクトを生成

Create_Blockはどのオブジェクトに持たせても問題はないが「誰でも良かった」みたいな犯罪者じみている発想は危険なのでしっかり専用のオブジェクトを用意したあげましょう

Cubeなどと同じようにメニューバーから
GameObject>Create Empty
で「空のGameObject」を生成
名前を「create_blocks」にする

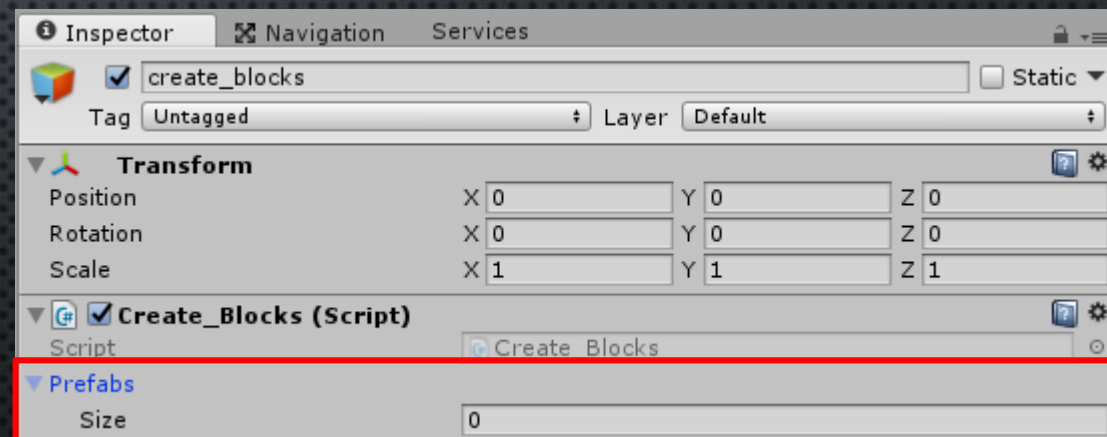
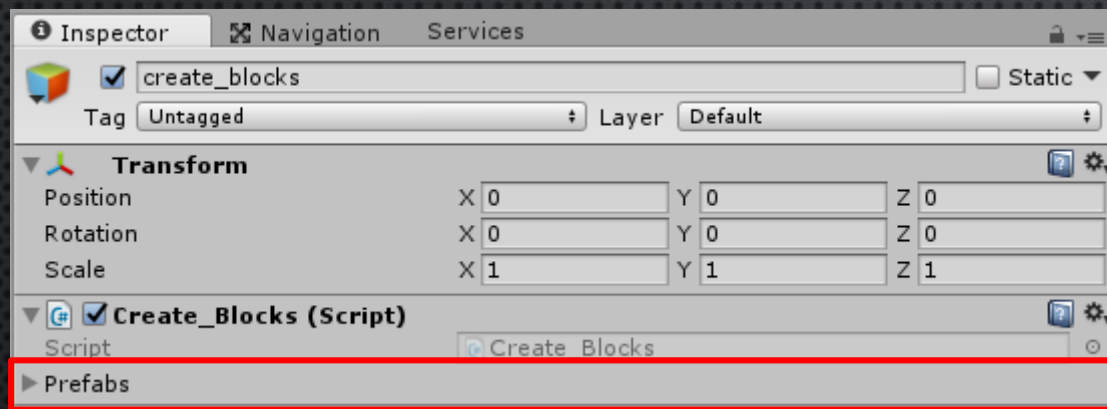
Create Emptyとは...

CubeやSphereのような3Dモデルがなく、初期状態では「Transform」(座標)しかもっていないオブジェクトの生成やユーザーには見えない、裏でやってるようなプログラムなどをここに付けるといいかも



先ほど作ったCreate_Blocksスクリプトを
Create_blocksオブジェクトにドラッグ&ドロップする



Prefabsをクリックすると...
下にSizeっていうものが出てくる
ここに先ほど作ったblock達を入れていくのだが



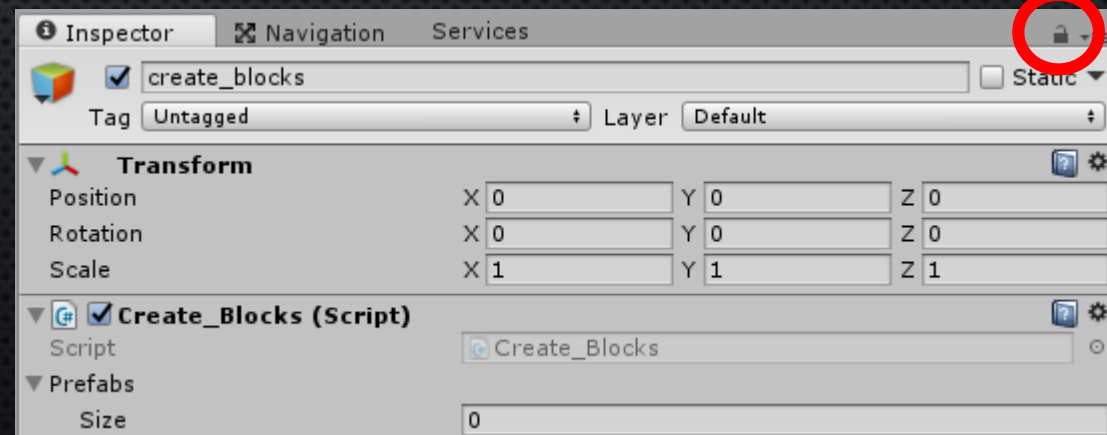
ここでちょっとした小技を...

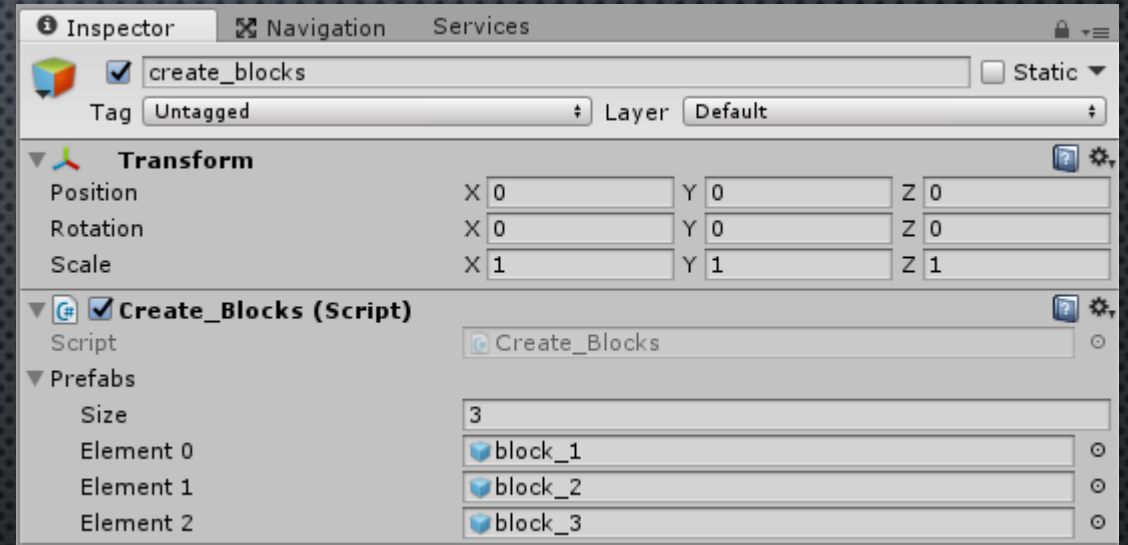
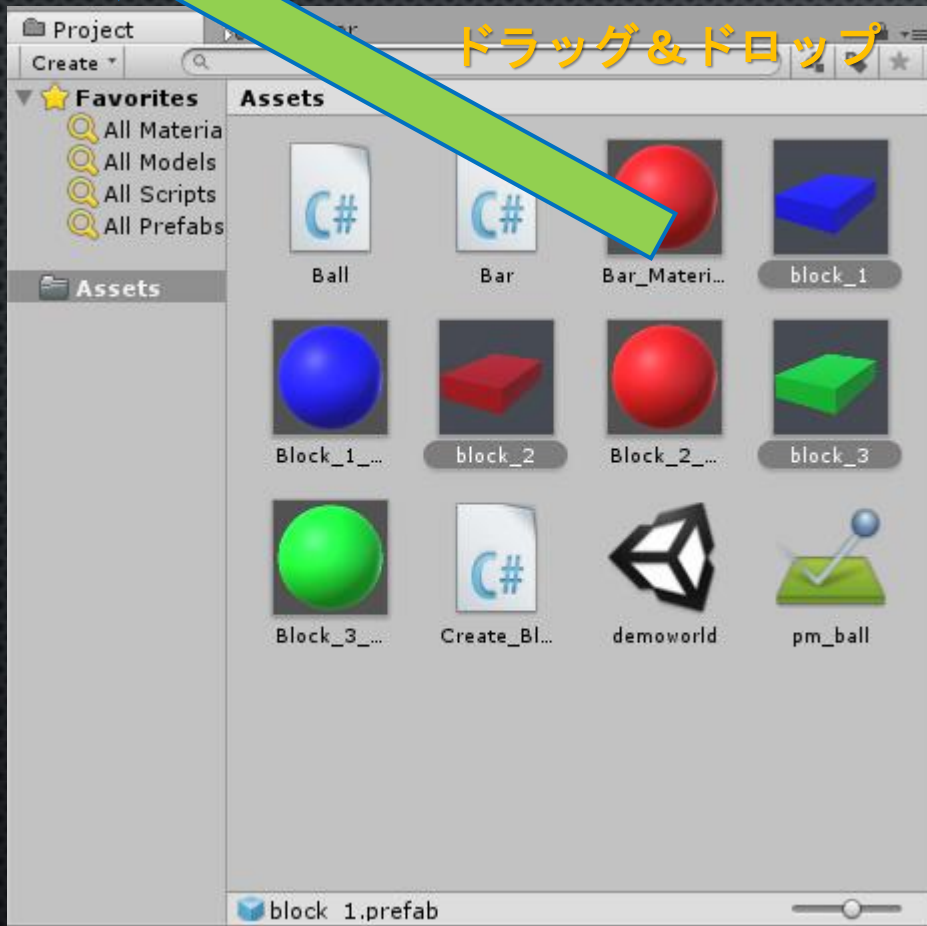
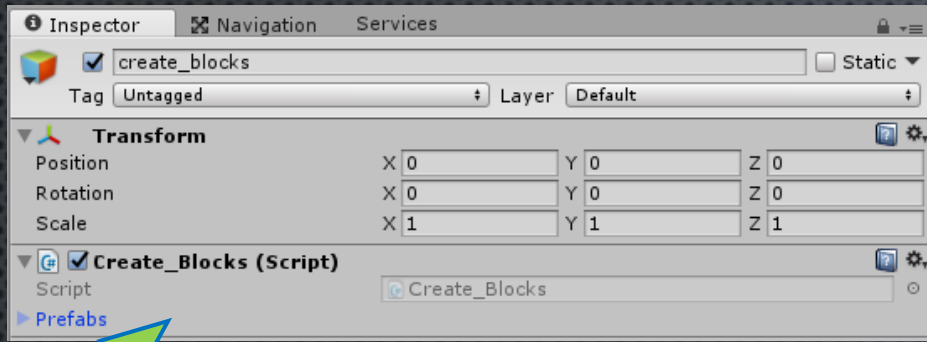
通常、ここで入れるオブジェクトの数をSizeに入力して入れるオブジェクトを一つ一つドラッグ&ドロップするのだが数が多いときはかなりめんどくさい！

ので一度に大量のオブジェクトをリストに入れる方法を紹介

1. まずcreate_blocksオブジェクトを選択し、Inspectorの右上にある「」マークをクリックして「」にする
2. 入れるオブジェクトを複数選択する (Ctrlを押しながらか選択していただく)
3. そのままprefabsにドラッグ&ドロップする

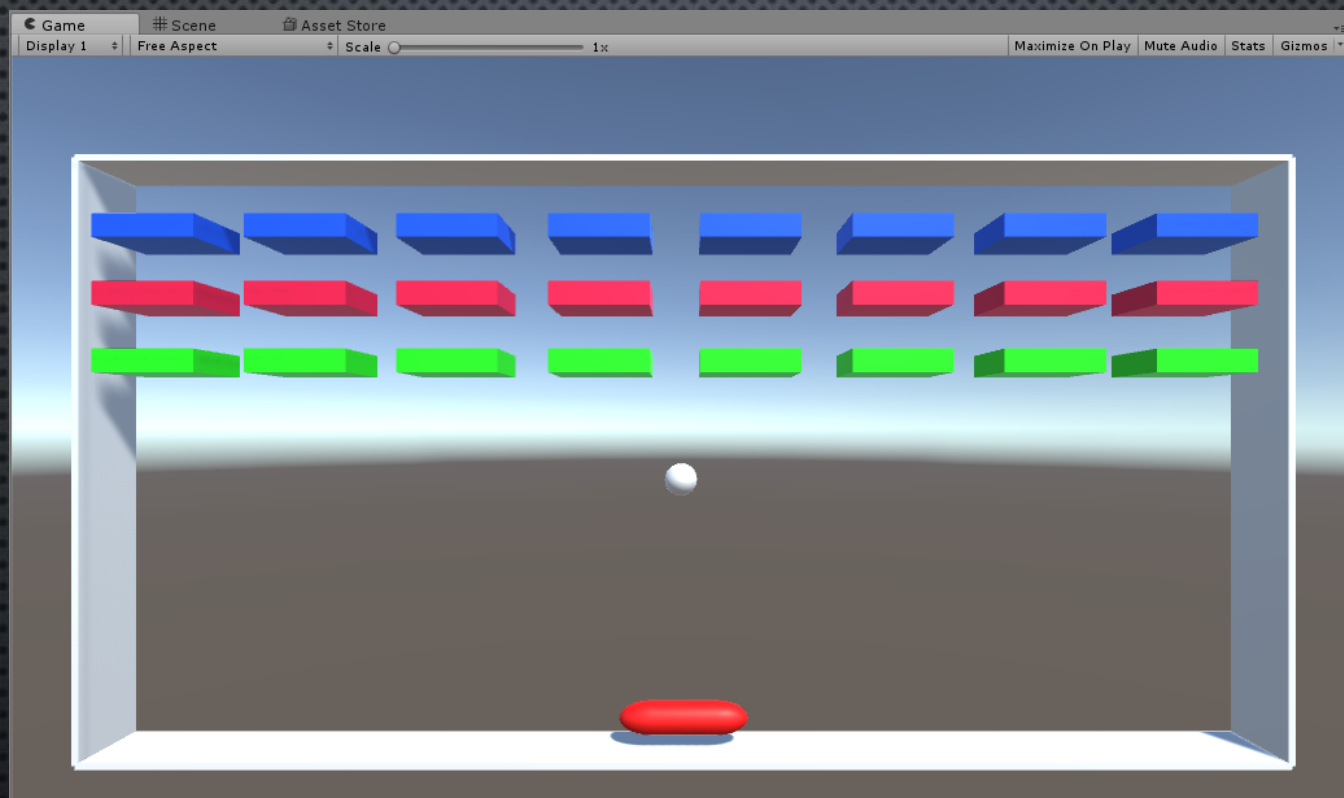
ここ





一発で全部入った！！
便利！！
(最後に🔒を外すのを忘れずに)

生成するオブジェクトも入れられたので早速確認



や っ た ぜ

このままではballがblockに
当たっても破壊されないクソゲーなので
「ballに当たったら破壊される」プログラムを実装する

「Block」という名前のC#Scriptを作り blocks_1, 2, 3にドラッグ&ドロップする

コードを追加

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

public class Block : MonoBehaviour {

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

    // Use this for initialization
    void Start () {

    }

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

    }
}
```

プログラム上エラーは吐かれないが
関数名を「OnCollisionEnter()」にしないと
ちゃんと衝突時に動作してくれないので注意!

今回はここまで

すいません、ここまでしか資料が間に合いませんでした...
全ては大学と前日にガンダム動物園で遊んだのが原因です
心よりお詫びします