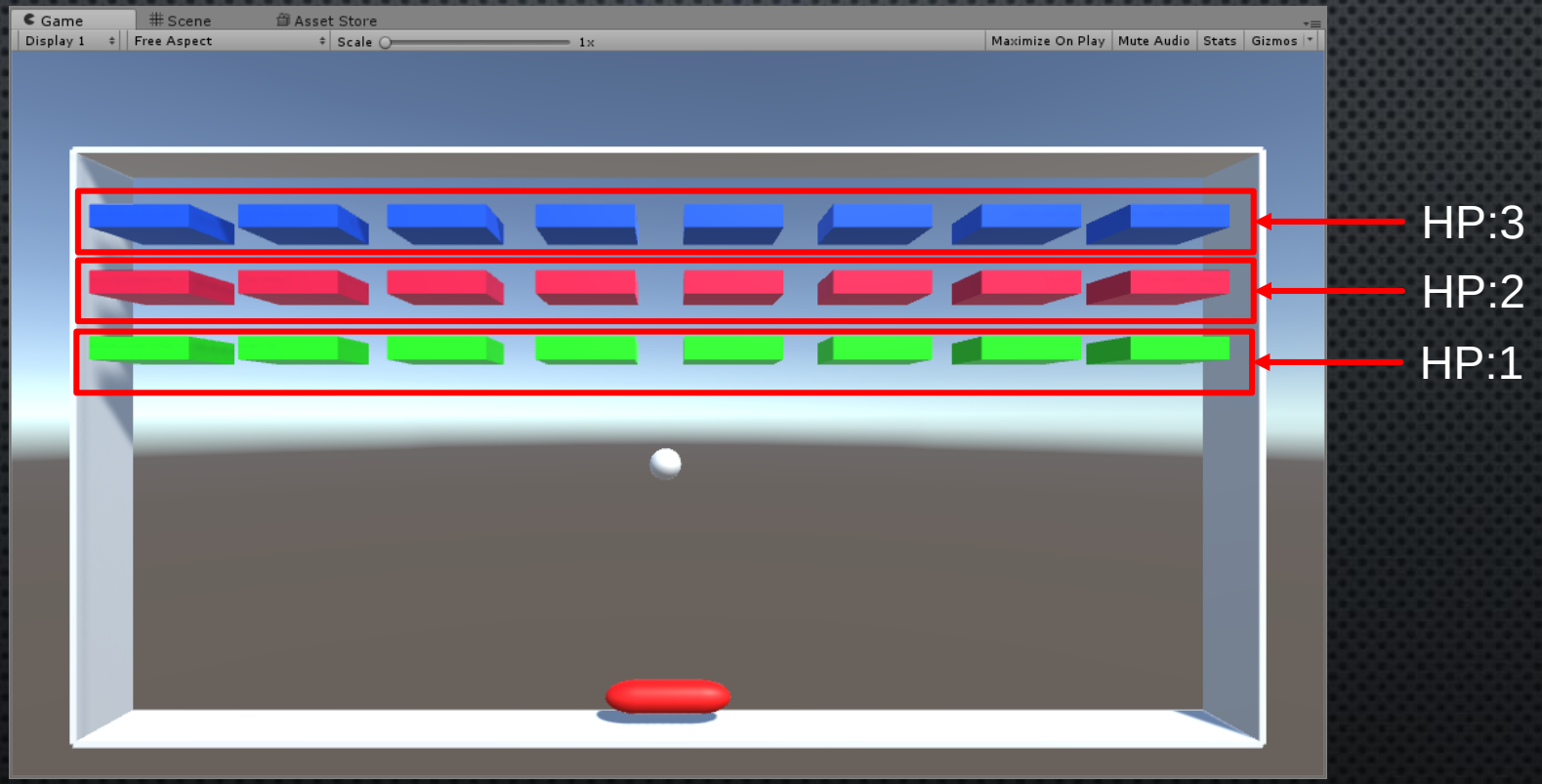
Unity講座②

2018/6/25(月)

ブロックにHPをつける

こんな感じにHPを分ける

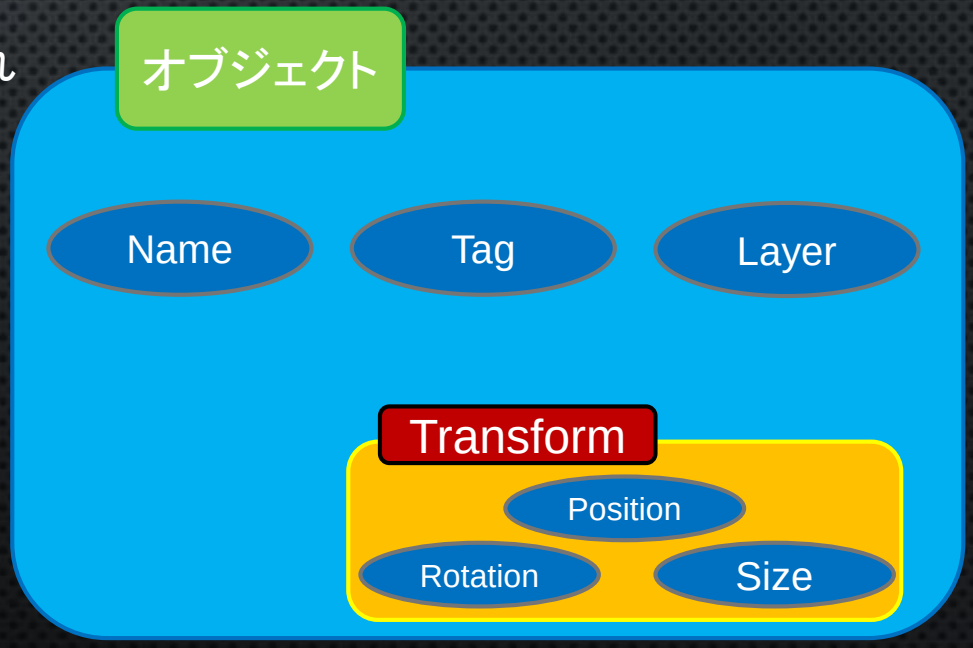


タグ付け

どのブロックがどのタイプかを判別するために
Tag付けを行う。

どのオブジェクトか見分けるには
Name(名前),Tag(タグ)、Layer(レイヤー)
の3種類から判断できる。

Name:オブジェクトの名前、固有のオブジェクトを見分けるにはこれ
Tag:わかりやすく言うとこの「ジャンル」みたいなもの
　　一つのタグに対して複数のオブジェクトが該当
Layer:Tagと似てるができることが違う(今回は割愛)

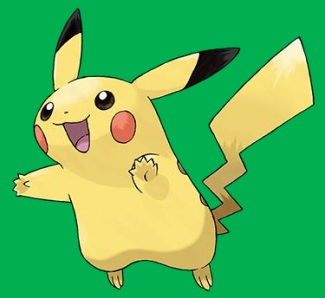


ポケモンで例えると・・・

Tag:ねずみポケモン

Name:サンド

Name:コラッタ



Name:ピカチュウ

Name:サンドパン

Name:ラッタ

Name:ライチュウ

Tag:アンテナポケモン

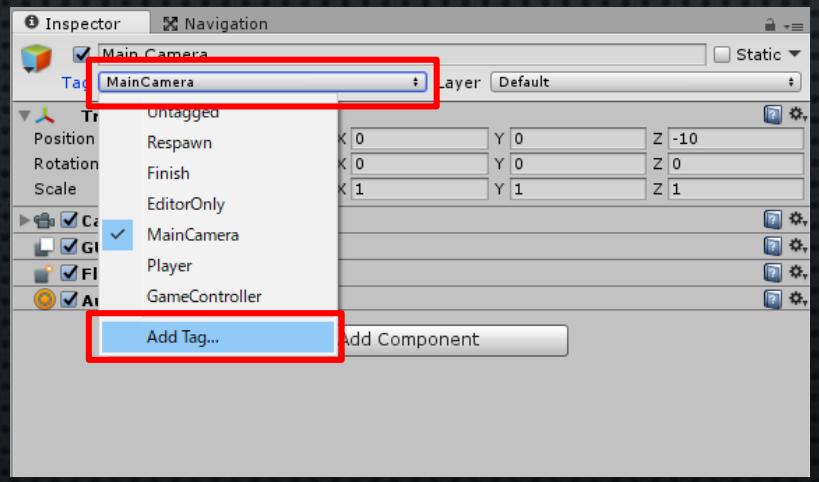
Name:デデカス

タグ作成

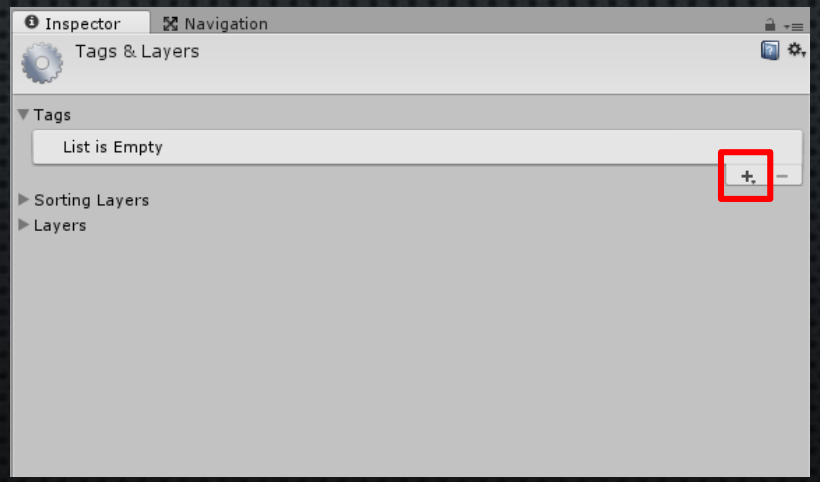
タグの作成
1. 適当にオブジェクトを選ぶ
2. Inspector内のTagをクリック
3. Add Tag…をクリック
4. 「List is Empty」の右下にある「+」をクリック
5. 「block1」と入力して「Save」をクリック

1～5を「block2」「block3」というタグ名で再度行う

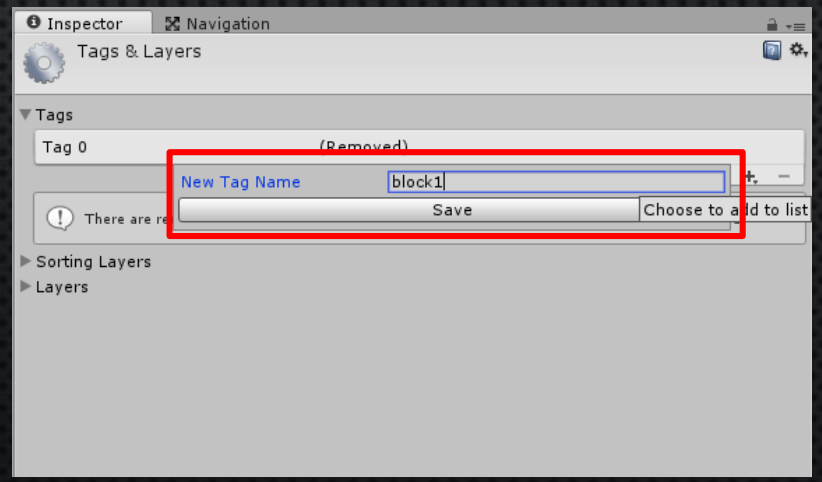
1. 2. 3.



4.



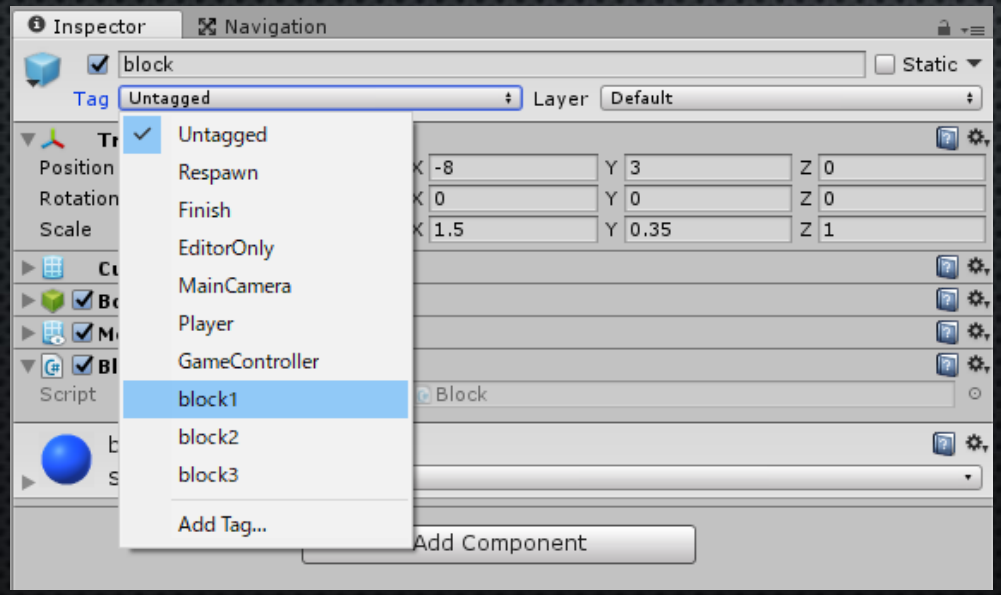
5.



タグ設定

タグの設定
1. Project内のblockオブジェクトを選ぶ
2. Inspector内のTagをクリック
3. 「block1」をクリック

同じように「block2」,「block3」をのタグを「block2」「block3」に設定する



プログラムにHPを追加

Blockスクリプトを開いて中身を変更

- Int型変数 hpを追加
- OnCollisionEnter関数の中身を変更
- Start関数内の中身を変更

this.gameObject.tag
これでこのスクリプトをアタッチ(持っている)オブジェクトのtagの情報を得られる

```
public class Block : MonoBehaviour {
    private int hp;//←追加
    void OnCollisionEnter(Collision other)
    {
        //↓変更
        hp -= 1;
        if (hp == 0) {
            Destroy(this.gameObject);
        }
    }

    // Use this for initialization
    void Start () {
        //↓変更
        if (this.gameObject.tag == "block1") {
            hp = 3;
        }
        else if (this.gameObject.tag == "block2")
        {
            hp = 2;
        }
        else if (this.gameObject.tag == "block3")
        {
            hp = 1;
        }
}

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

}
}
```

これで起動してみると…

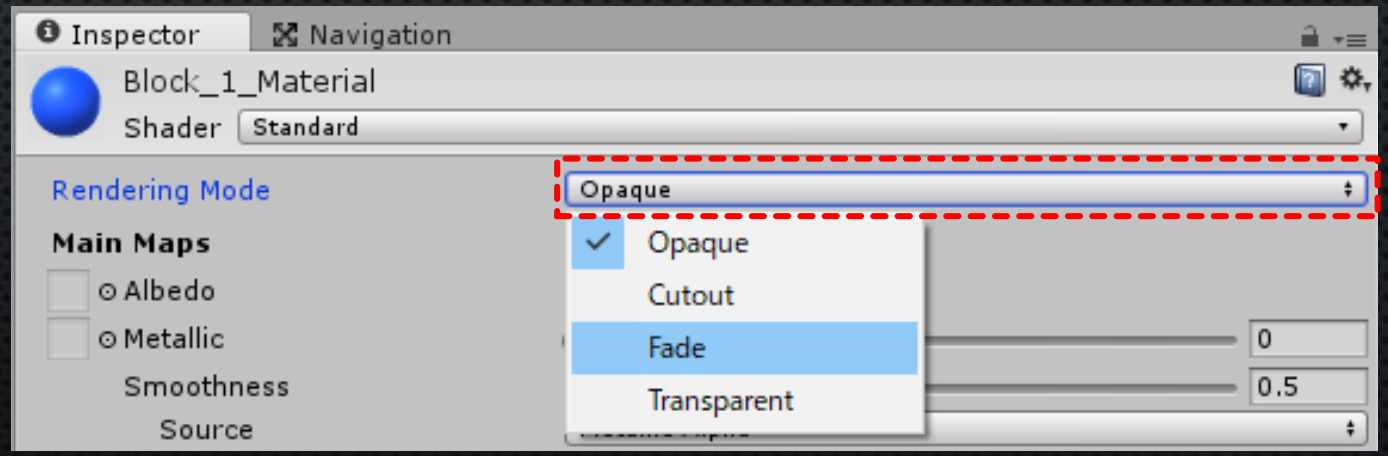
一段目のブロック(緑)は一回当たると壊れ、
二段目のブロック(赤)は二回、三段目のブロック(青)は三回当たると壊れる

しかしこのままでは二、三段目のブロックがあと何回で壊れるかわからない!!

HPに応じてブロックの透明度を変更

1. Block_1_Materialを選択
2. Inspector内の上にあるRendering Modeの「Opaque」をクリック
3.「Fade」を選択

これで透明度をいじれるようになった



HPに応じてブロックの色を変更

またBlockスクリプトを開いて中身を変更

- OnCollsionEnter関数内の中身を変更

```
public class Block : MonoBehaviour {
    private int hp;
    void OnCollisionEnter(Collision other)
    {
        hp -= 1;
        if (hp == 0)
        {
            Destroy(this.gameObject);
        }
        //追加
        else {
            GetComponent<Renderer>().material.color *= new Color(1.0f, 1.0f, 1.0f, 0.5f);
        }
    }

    // Use this for initialization
    void Start () {
        if (this.gameObject.tag == "block1") {
            hp = 3;
        }
        else if (this.gameObject.tag == "block2")
        {
            hp = 2;
        }
        else if (this.gameObject.tag == "block3")
        {
            hp = 1;
        }
}
```

`GetComponent<Renderer>()`
このスクリプトをアタッチしているオブジェクトの
Rendererを取得

`Renderer.material.color`
Rendererの持っているマテリアル(material)の色を取得

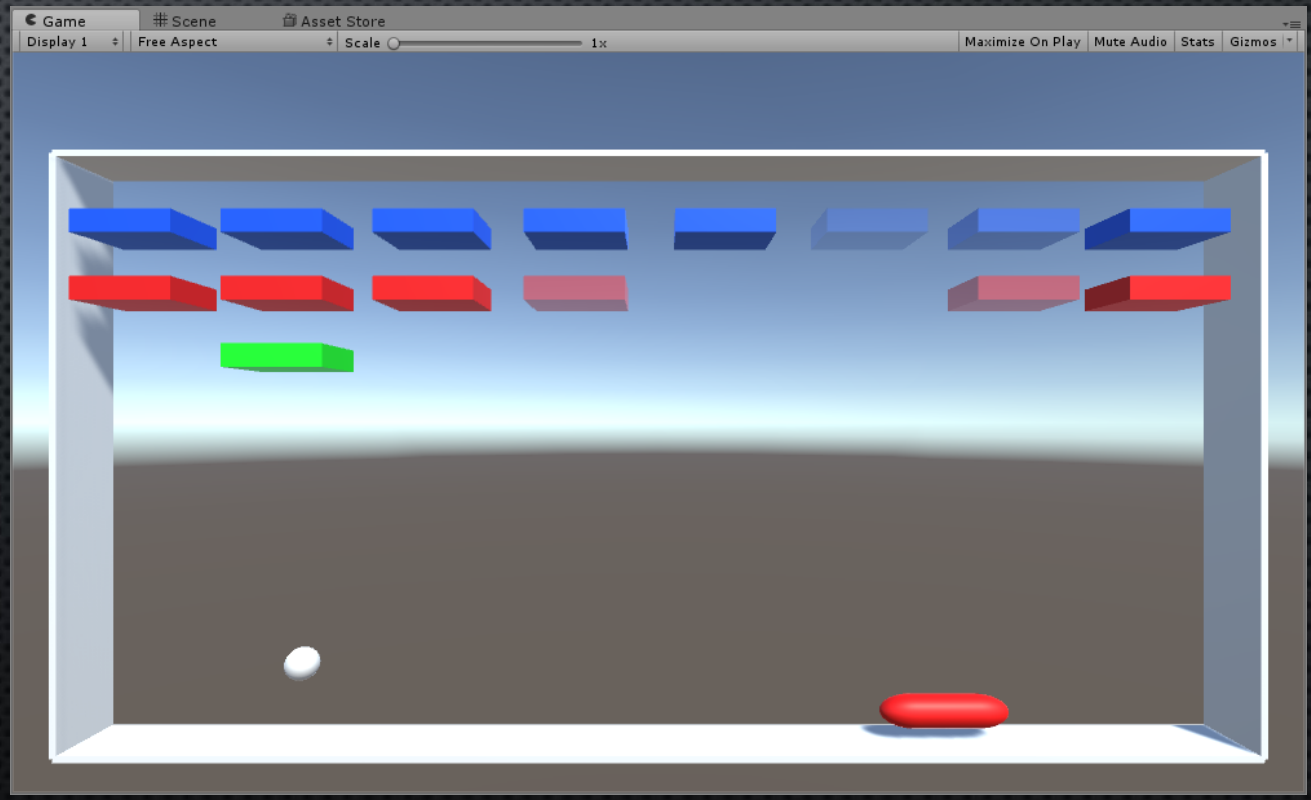
訳すと…

```
GetComponent<Renderer>().material.color *= new Color(1.0f, 1.0f, 1.0f, 0.5f);
```

「このオブジェクトのRendererを取得し、そのRendererのマテリアルの色に右の値をかけた色に更新する」

これで起動してみると…

HPに応じてブロックが薄くなっていく！



クリア処理

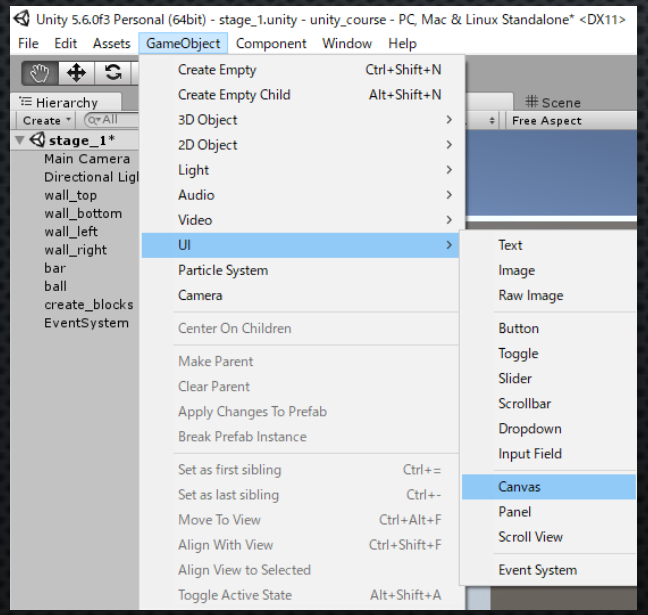
ゲームクリアを判定する(準備)

1. C#スクリプト Stage_Manageを作る
2. CreateEmpty stage_manageを作る
3. Stage_Manageスクリプトをstage_manageオブジェクトにアタッチ(ドラッグ＆ドロップ)する

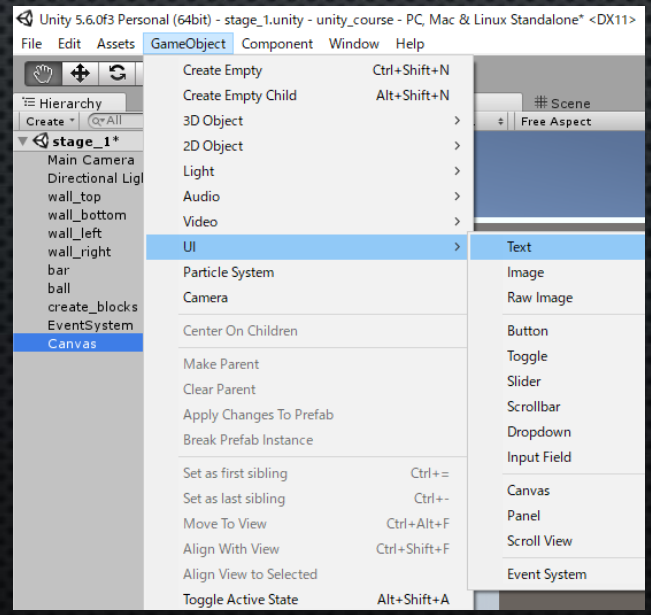
ゲームクリアを判定する(UI作成)

1. GameObject>UI>Canvasを選択
2. GameObject>UI>Textを選択
3. Textオブジェクトの名前をclear_text
4. clear_textを選びInspector内を図のようにする

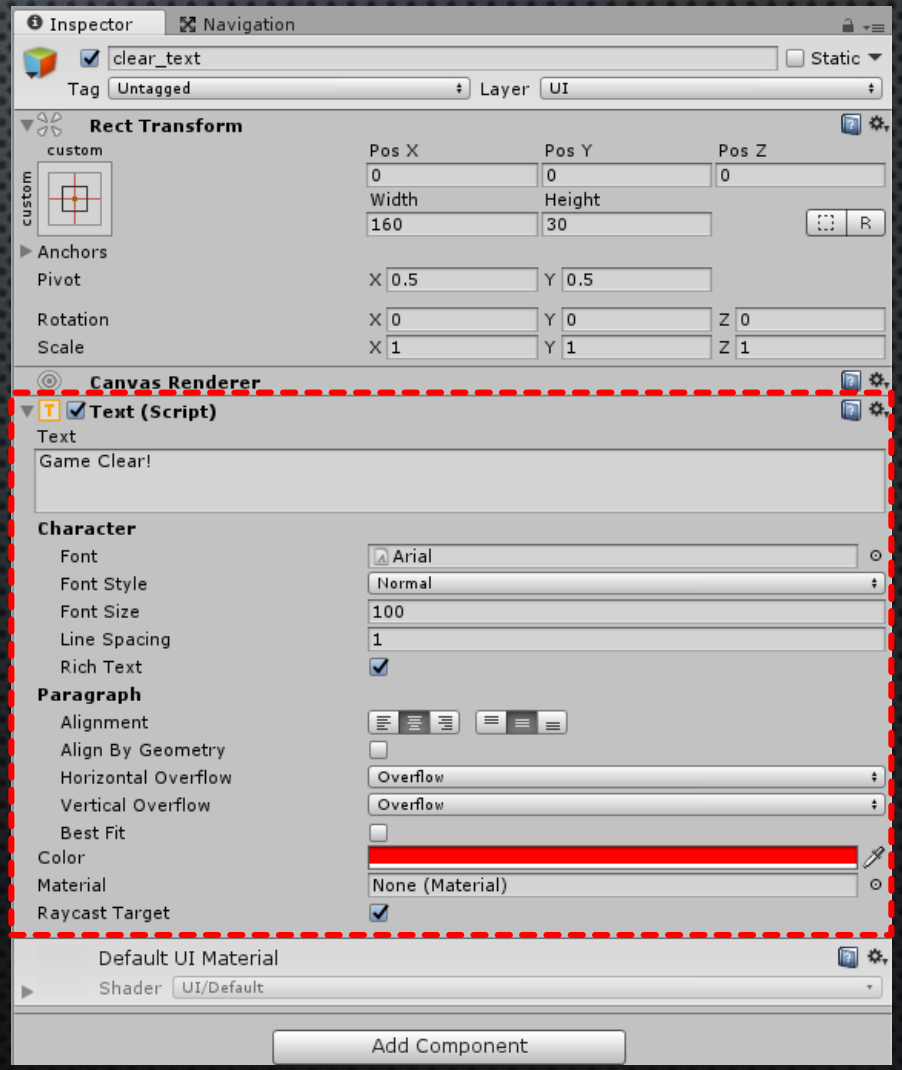
1.



2.



4.



ゲームクリアを判定する(プログラム)

Stage_Manageスクリプトを開いて以下のようにする

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

public class Stage_Manage : MonoBehaviour {
    public bool game_clear;
    public Text clear_text;

    public bool check_clear() {
        if (GameObject.FindGameObjectWithTag("block1") == null
            && GameObject.FindGameObjectWithTag("block2") == null
            && GameObject.FindGameObjectWithTag("block2") == null) {
            game_clear = true;
            clear_text.enabled = true;
        }
        return game_clear;
    }



// Use this for initialization
void Start () {
        game_clear = false;
        clear_text.enabled = false;
    }

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

}
}
```

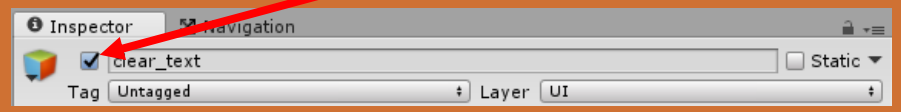
忘れがちなので注意!!

プログラム解説

GameObject.FindGameObjectWithTag()
()内に文字列を入れてその文字列と同じタグの
オブジェクトを「一つのみ」取得、無い場合はnullを返す
複数の取得をしたい場合は
GameObject.FindGameObjectsWithTag ()

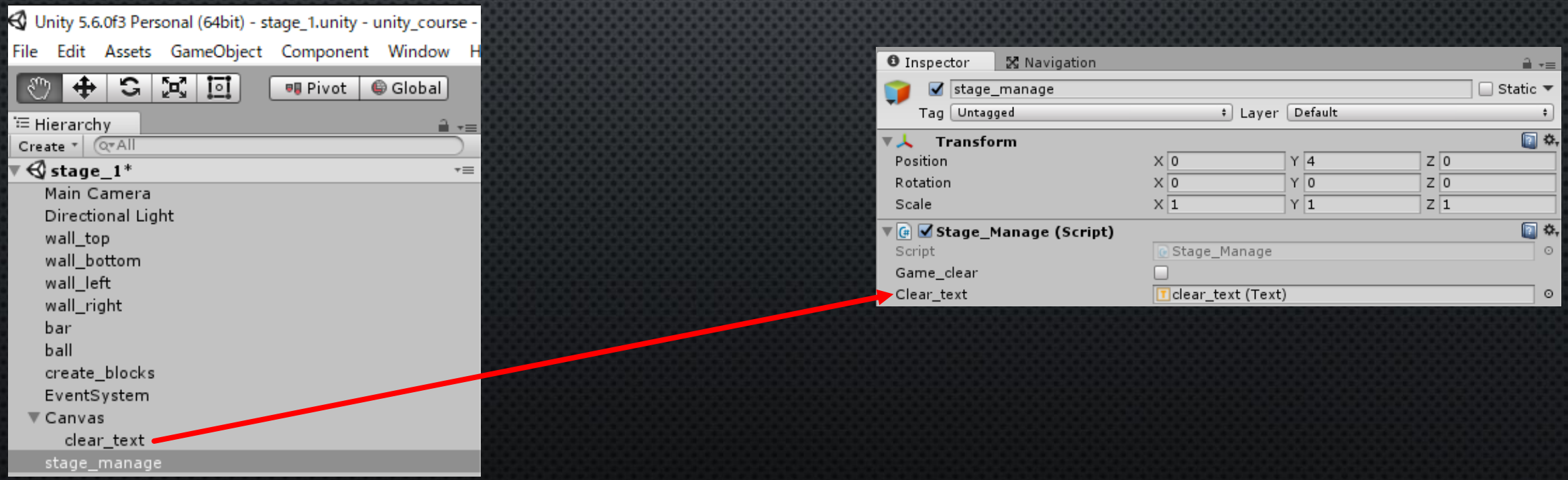
～(GameObject).enabled
指定したオブジェクトをシーン内で
有効化(true)、無効化(false)を設定
(オブジェクトの色を透明化させるとはまた違う)

ここをいじると
同じ事ができる




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

public class Stage_Manage : MonoBehaviour {
    public bool game_clear;
    public Text clear_text;

    public bool check_clear() {
        if (GameObject.FindGameObjectWithTag("block1") == null
            && GameObject.FindGameObjectWithTag("block2") == null
            && GameObject.FindGameObjectWithTag("block2") == null) {
            game_clear = true;
            clear_text.enabled = true;
        }
        return game_clear;
    }



// Use this for initialization
void Start () {
        game_clear = false;
        clear_text.enabled = false;
    }

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

}
}
```

clear_textをstage_manageのStage_Manageスクリプト内の
Text型変数「clear_text」にドラッグ＆ドロップ



ボールからクリア判定を行う

Ballスクリプトを開いてコードを以下のように変更

GameObject.Find()
()内に入れた文字列と同じ名前のオブジェクトをこのシーンないから検索し取得する。
ない場合はnullを返す

GetComponent<Stage_Manage>()
<>内のスクリプトを取得する
使い方としては
「どのオブジェクトか」指定し、そのオブジェクトから「どのスクリプトを取得するか」決める

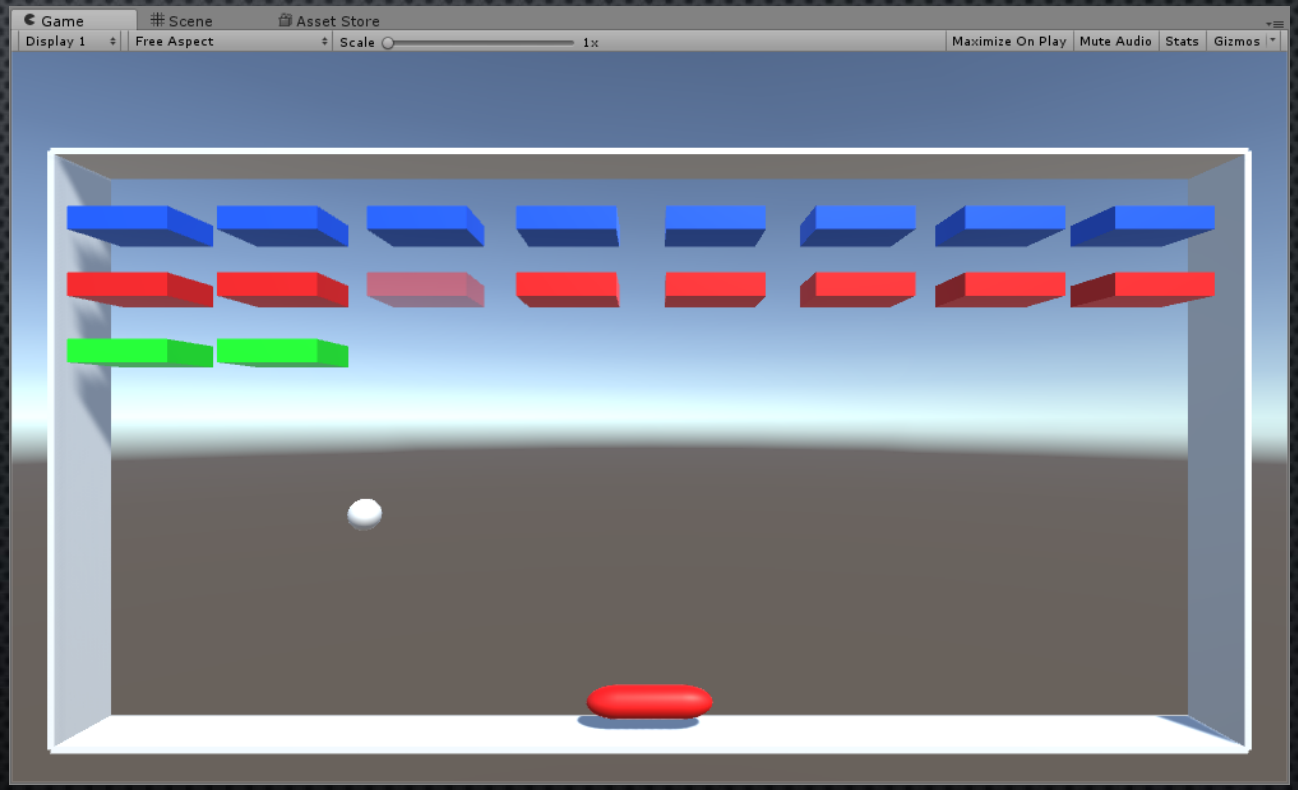
```
public class Ball : MonoBehaviour {
    Rigidbody rb;
    private Stage_Manage s_manage;

    void OnCollisionEnter(Collision other) {
        if (s_manage.check_clear()) {
            //玉の動きを止める
            rb.velocity = new Vector3(0, 0, 0);
        }
    }

// 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;
        s_manage = GameObject.Find("stage_manage").GetComponent<Stage_Manage>();
}
```

受け取り先 / どのオブジェクトか / どのスクリプトを取得するか

```
s_manage = GameObject.Find("stage_manage").GetComponent<Stage_Manage>();
```

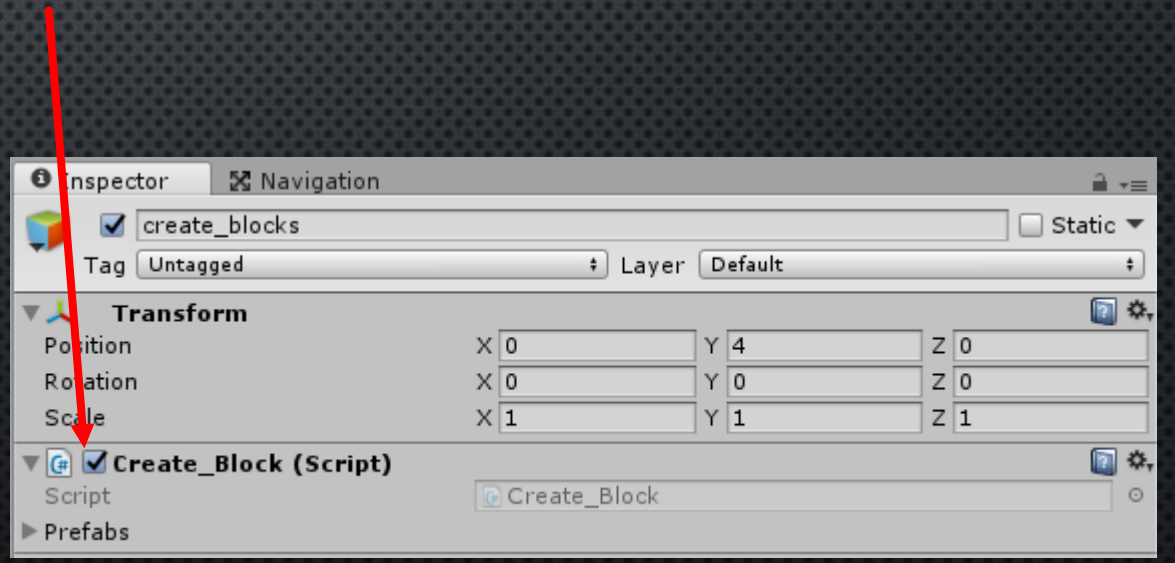
早速起動させて確認したいが…

ブロックが多すぎてめんどくさいわ!！



こういうときは…

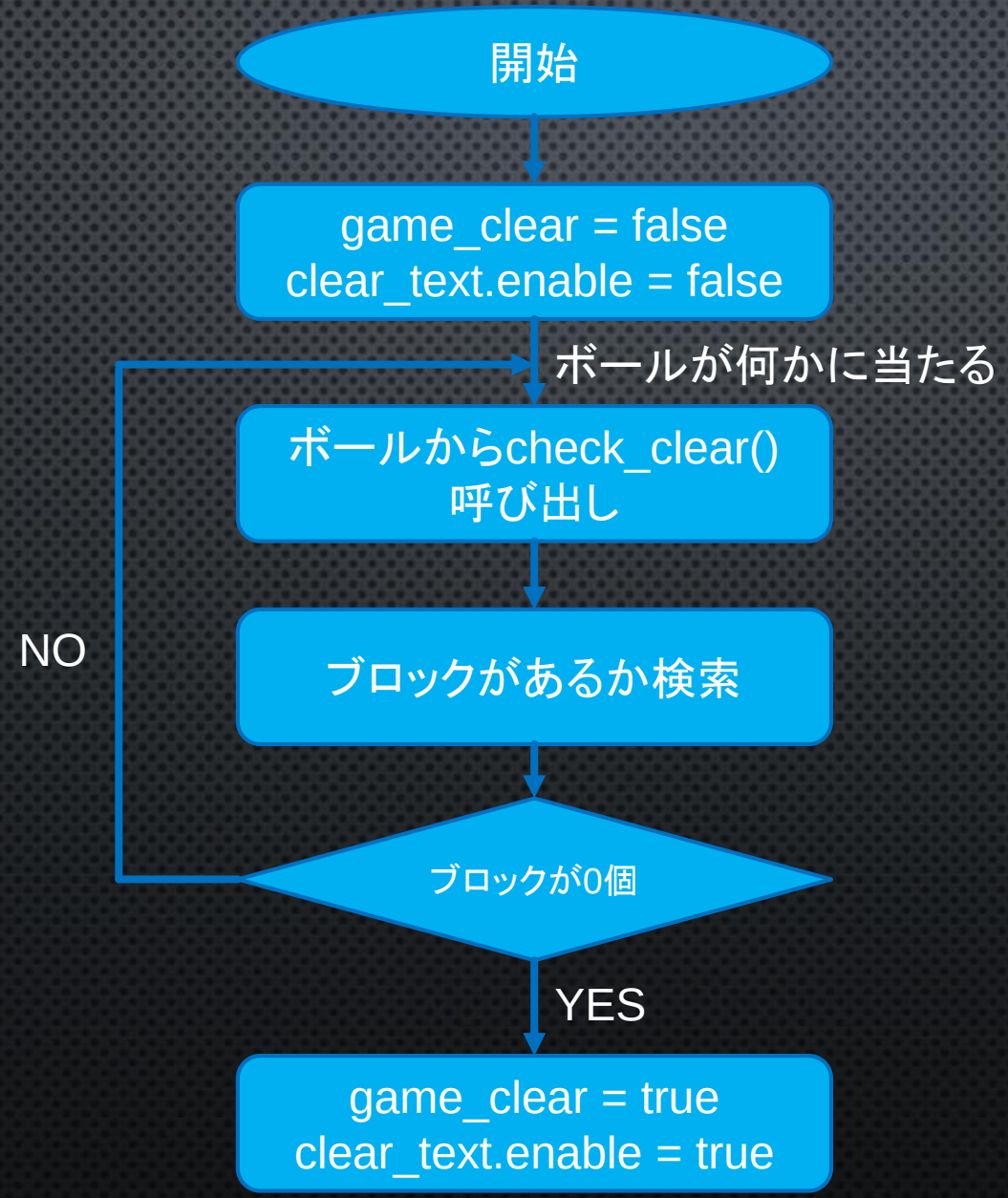
create_blocksオブジェクトのCreate_Blockのチェックを外す(enableをfalse)にする



そうすると…

ブロックが生成されなくなりボールが何かに当たった瞬間クリアになる

クリア判定(フローチャート)



タイトル画面作成

タイトル画面

今現在メインのステージ画面だけでゲームとしての起承転結ができていない
タイトル画面→(ステージ選択画面→)→ステージ画面→リザルト画面→タイトル画面→…

タイトル画面

ステージ選択画面

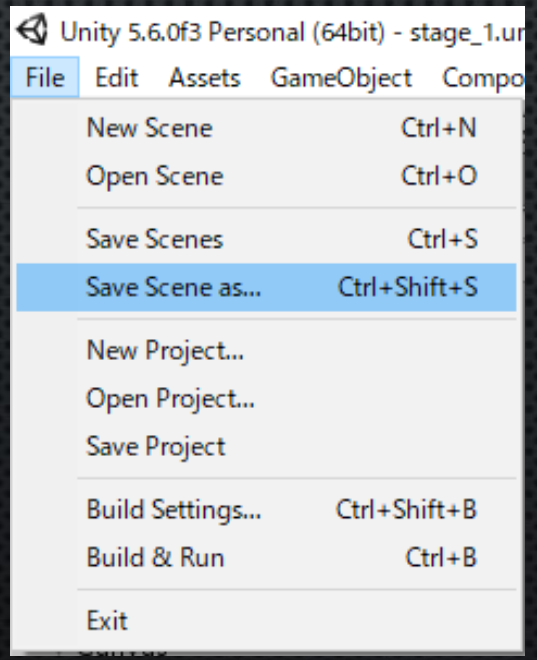
ステージ画面

リザルト画面

ステージシーンの保存

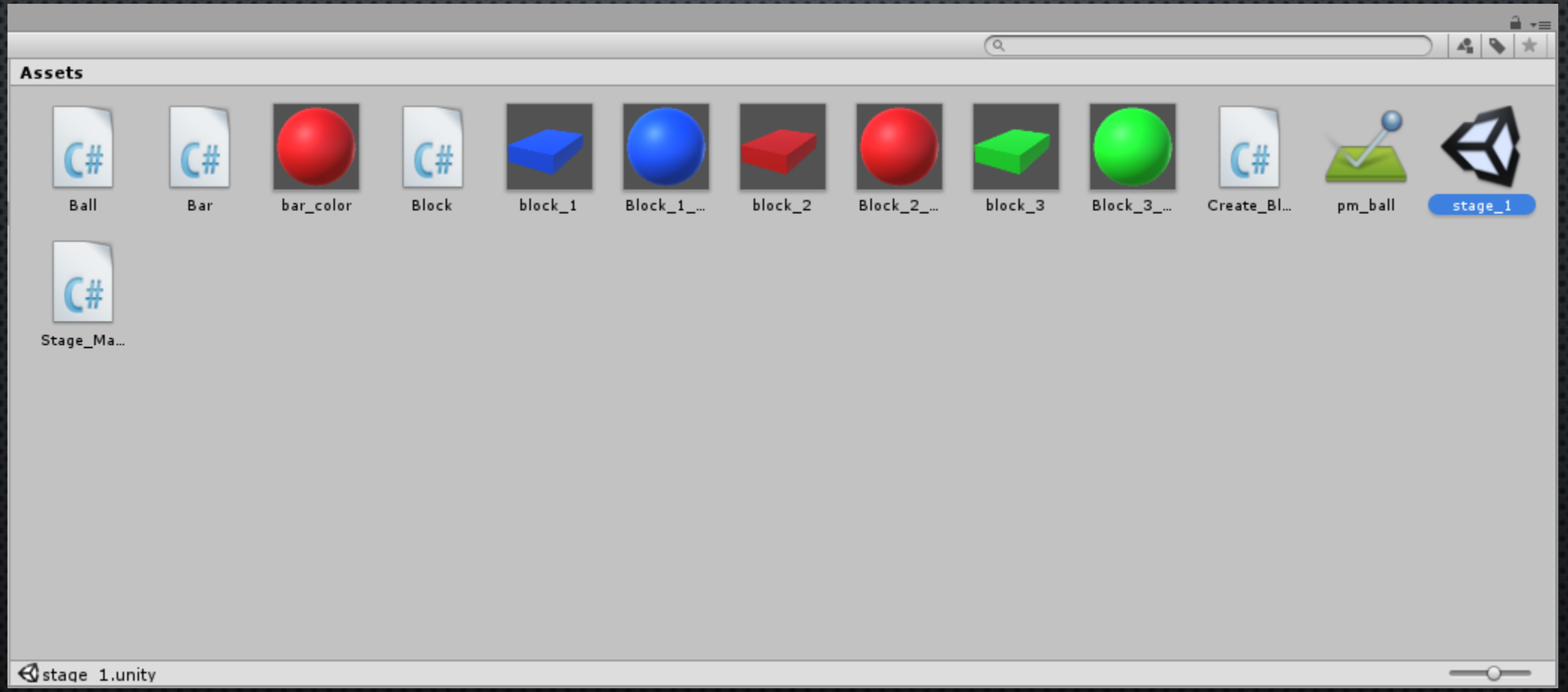
最新版Unityだと自動的にシーンを作っておいてくれてるらしいが
改めて自分で作る

左上のメニューバーから
File>Save Scene as…をクリック
ファイル名を「stage_1」にして保存



保存完了！

Project内に「stage_1」という名前のUnityロゴと同じアイコンがあったらOK!

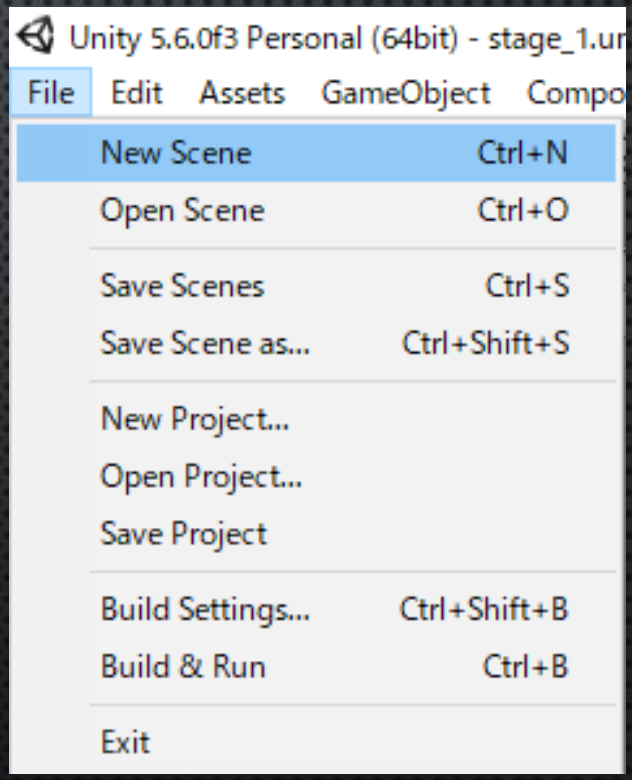


タイトルシーンの作成

タイトル画面を作る

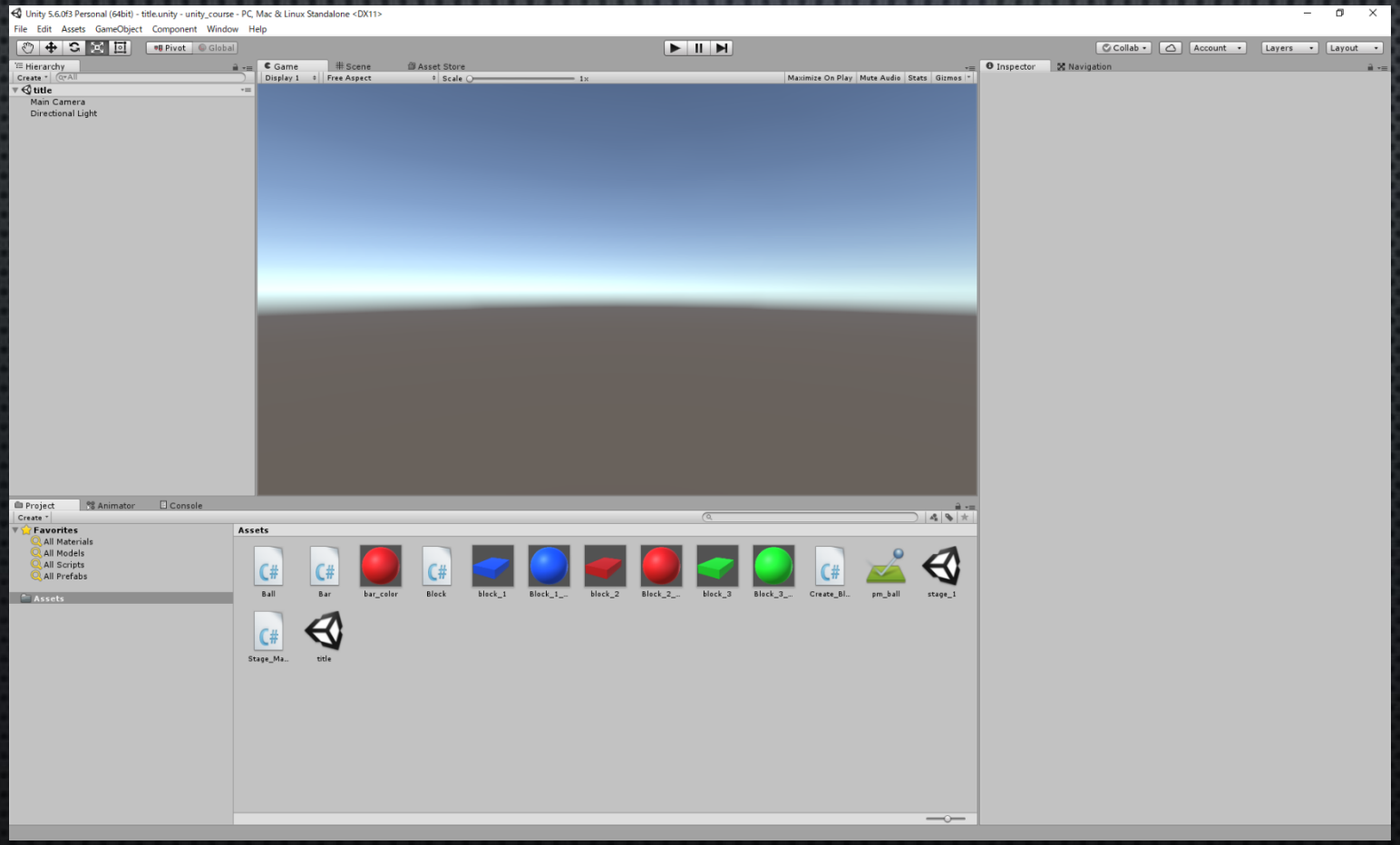
左上のメニューバーから
File>New Sceneをクリック

その後、先ほど同じ手順でシーンを保存し
ファイル名を「title」にする



初期と同じ状態になった!！

これまでのデータは全てstage_1に残っているので落ち着いて

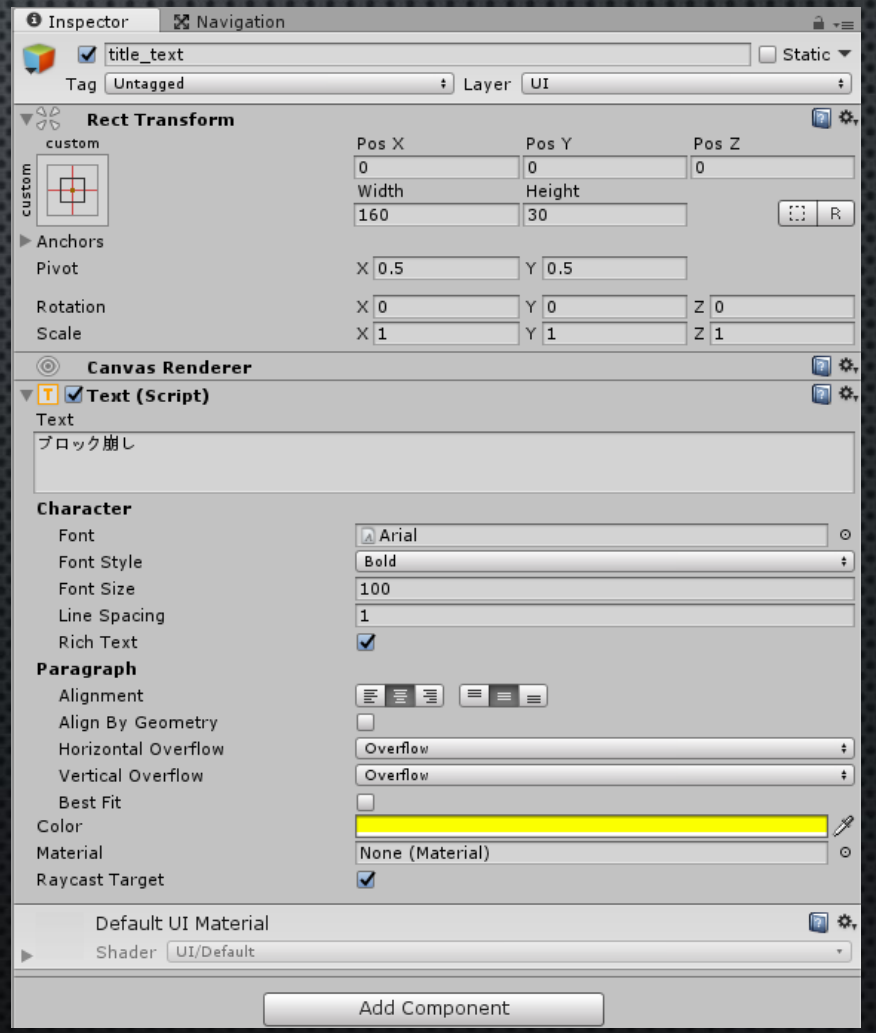


タイトル画面作成

準備も終わったのでいよいよ作成に移る

1. GameObject>UI>Textをクリック
2. 名前を「title_text」に
3. title_textの中身を右図のようにする

Canvasがなかった場合
Canvasを自動で作ってくれる

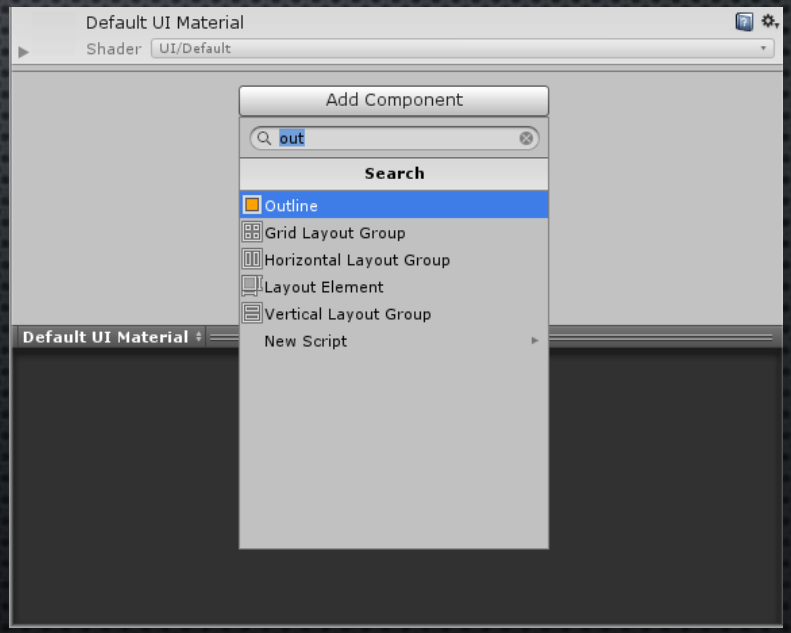


こんな感じになる


Game
Scene
Asset Store
Display 1
Free Aspect
Scale
1x
Maximize On Play
Mute Audio
Stats
Gizmos
ブロック崩し

見づらいので文字に縁をつける

1. title_textを選択
2. Add Componetをクリック
3. outと入力すると「outline」が
   検索結果の一番上に出てくる
4. Outlineを選択



見やすくなった

before

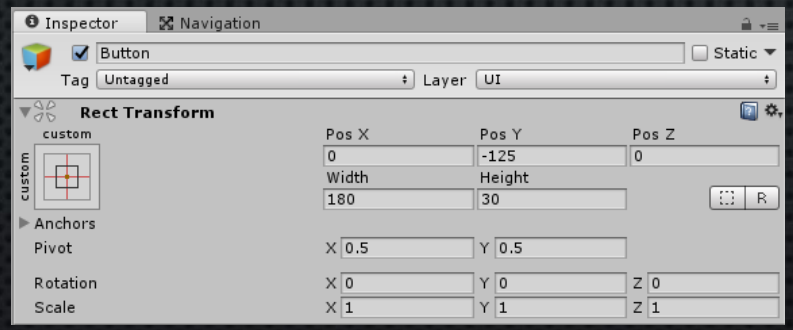
after

ボタン作成

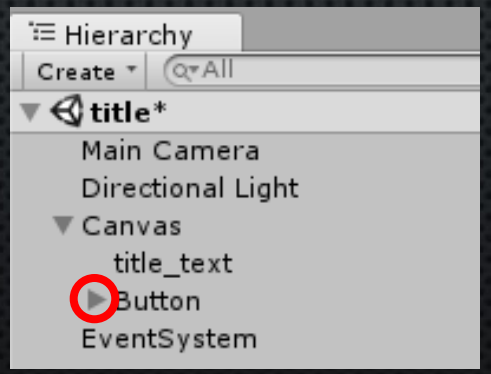
クリックされたらstage_1に移動するようなボタンを作る

1. GameObject>UI>Buttonをクリック
2. 名前を「start_button」に
3. start_buttonの位置や大きさは下図のように
4. Hierarchy内のstart_buttonの左にある「▶」をクリック
5. すると下にTextが現れるので名前をbutton_textにし右図のようにする

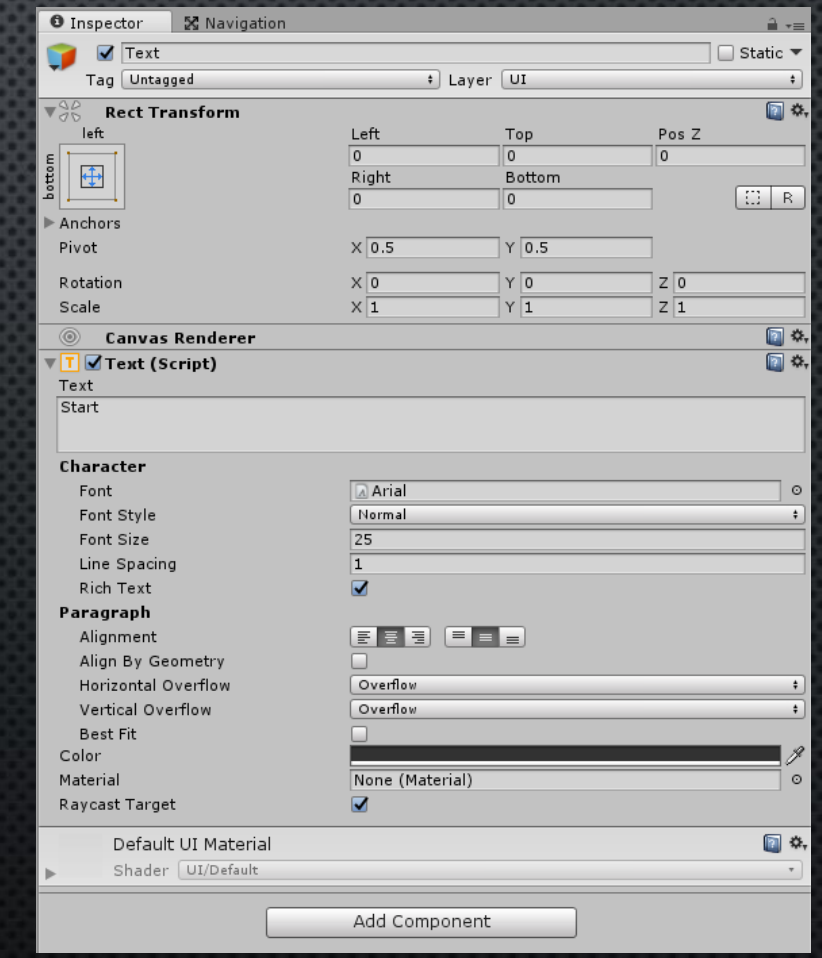
3.



4.



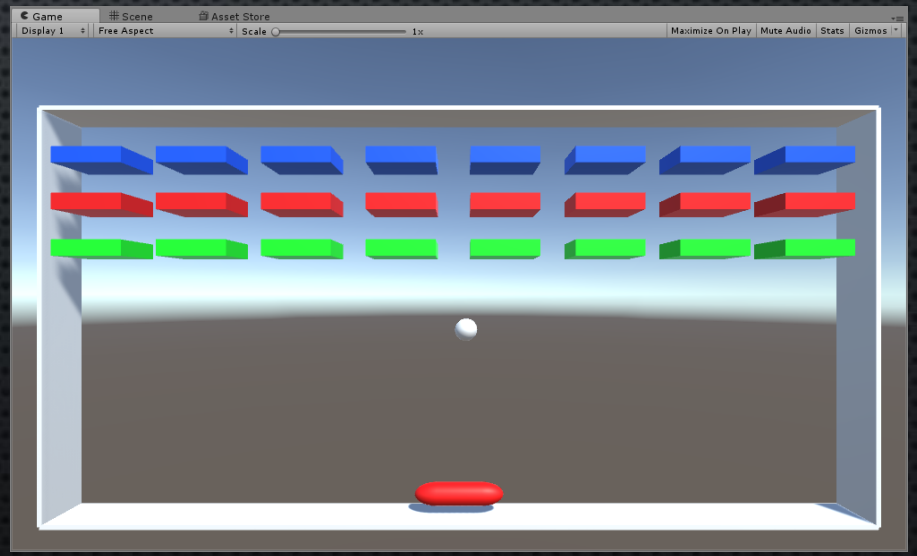
5.



シーン切り替え

シーン切り替え

titleシーンのStartボタンが押されたらstage_1シーンに遷移するようにする



シーン切り替え(準備)

1. C#スクリプト Start_Buttonを作る
2. CreateEmpty title_manageを作る
3. Start_Buttonスクリプトをtitle_manageオブジェクトに
   アタッチ(ドラッグ &ドロップ)する

シーン切り替え(プログラム)

Start_Buttonスクリプトを開いて以下のようにする

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

public class Start_Button : MonoBehaviour {

    public void title_to_stage() {
        SceneManager.LoadScene("stage_1");
    }

// Use this for initialization
void Start () {

}

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

}
}
```

忘れがちなので注意!!

プログラム解説

SceneManager.LoadScene();
()内に文字列を入れてProject内にあるその文字列と同じ名前のシーンにを読み込み移動する
※using UnityEngine.SceneManagement;
をしておかないとできないので忘れずにしておこう

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

public class Start_Button : MonoBehaviour {

    public void title_to_stage() {
        SceneManager.LoadScene("stage_1");
    }

// Use this for initialization
void Start () {

}

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

}
}
```

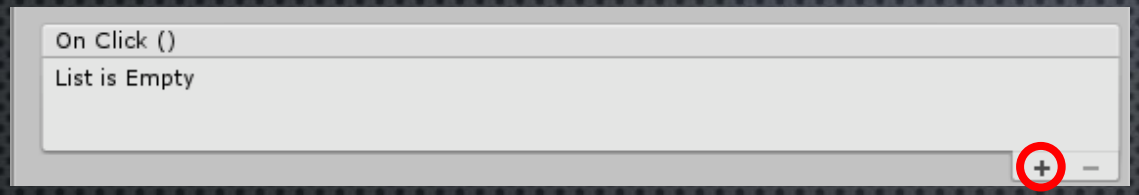
シーン切り替え(ボタン同期)

ボタンが押されたら先ほどのtitle_manage内にある
Start_Buttonの「title_to_stage」関数が呼び出されるようにする

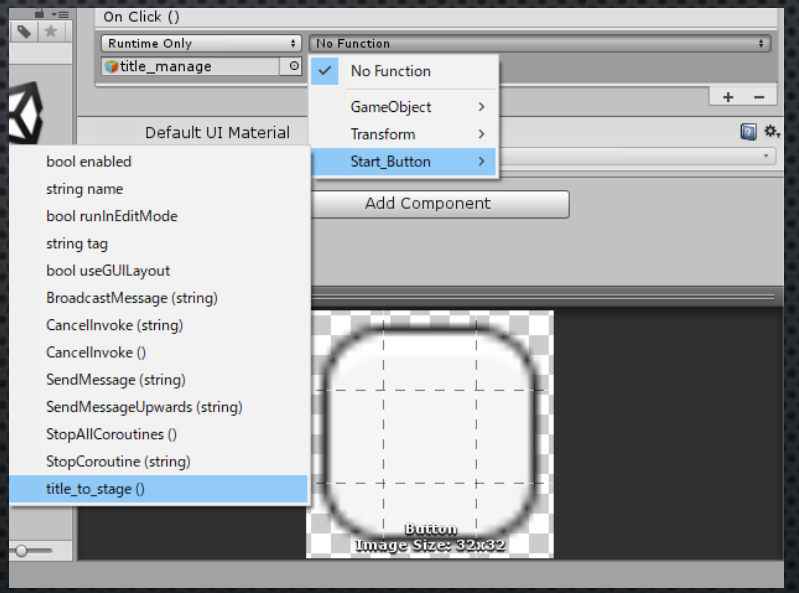
1. start_buttonを選択する
2. Inspector下にあるOn Click()内の「＋」を押す
3. どのオブジェクトか聞かれるのでtitle_manageをドラッグ &ドロップ
4. 「No Function」をクリック>Start_Button>title_to_stageeをクリック

シーン切り替え(ボタン同期)

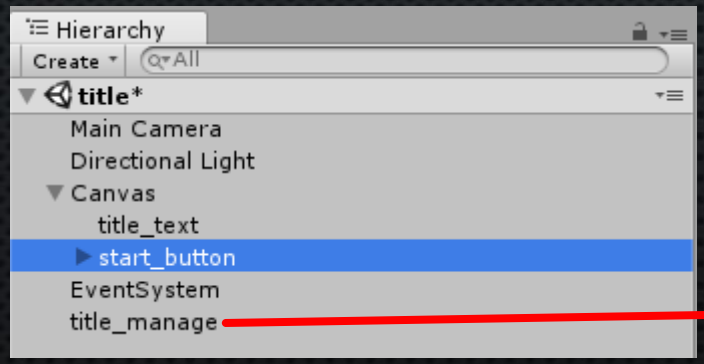
2.

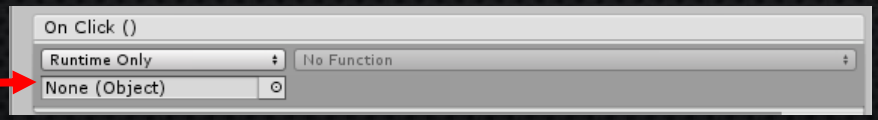


4.



3.



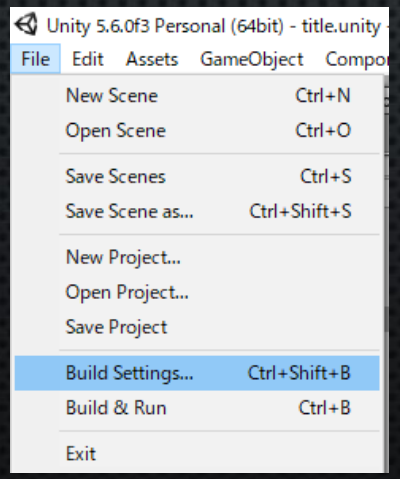


シーン切り替え(シーンを追加する)

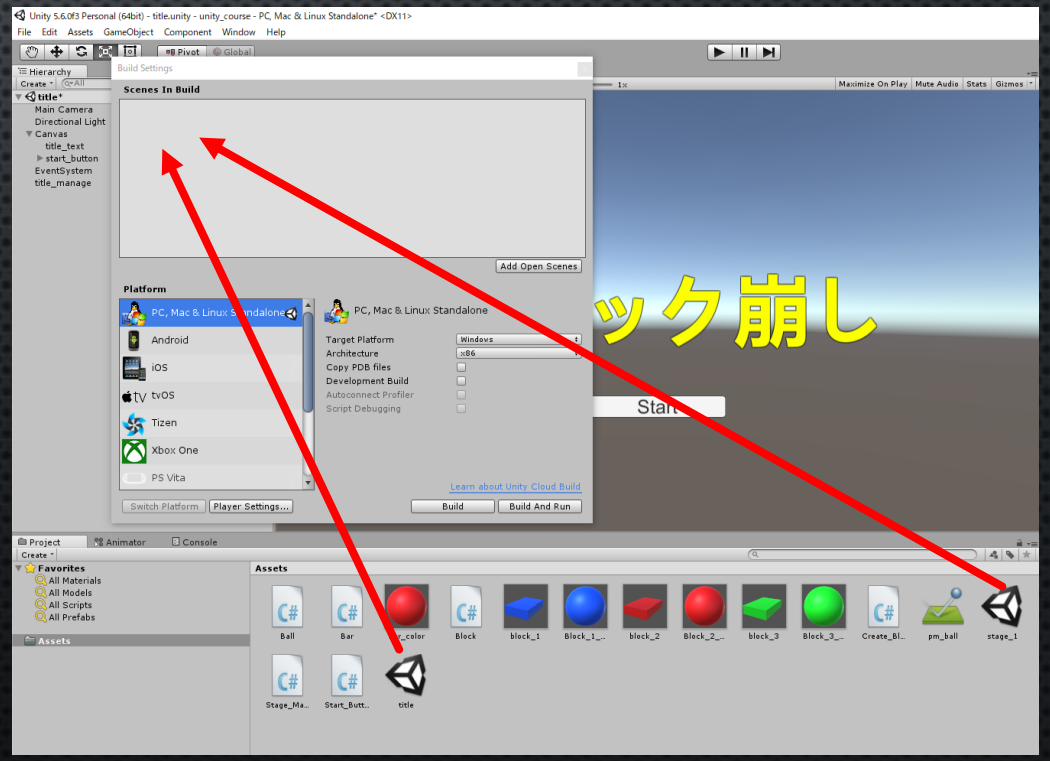
最後に二つのシーンをゲームに追加する
(この時点ではまだゲーム内には二つのシーンは入っていない)

1. 左上のメニューバーから
   File>Build Settingを選択
2. 出てきた画面内のScenes Buildに
   二つのシーンをドラッグ＆ドロップ

1.

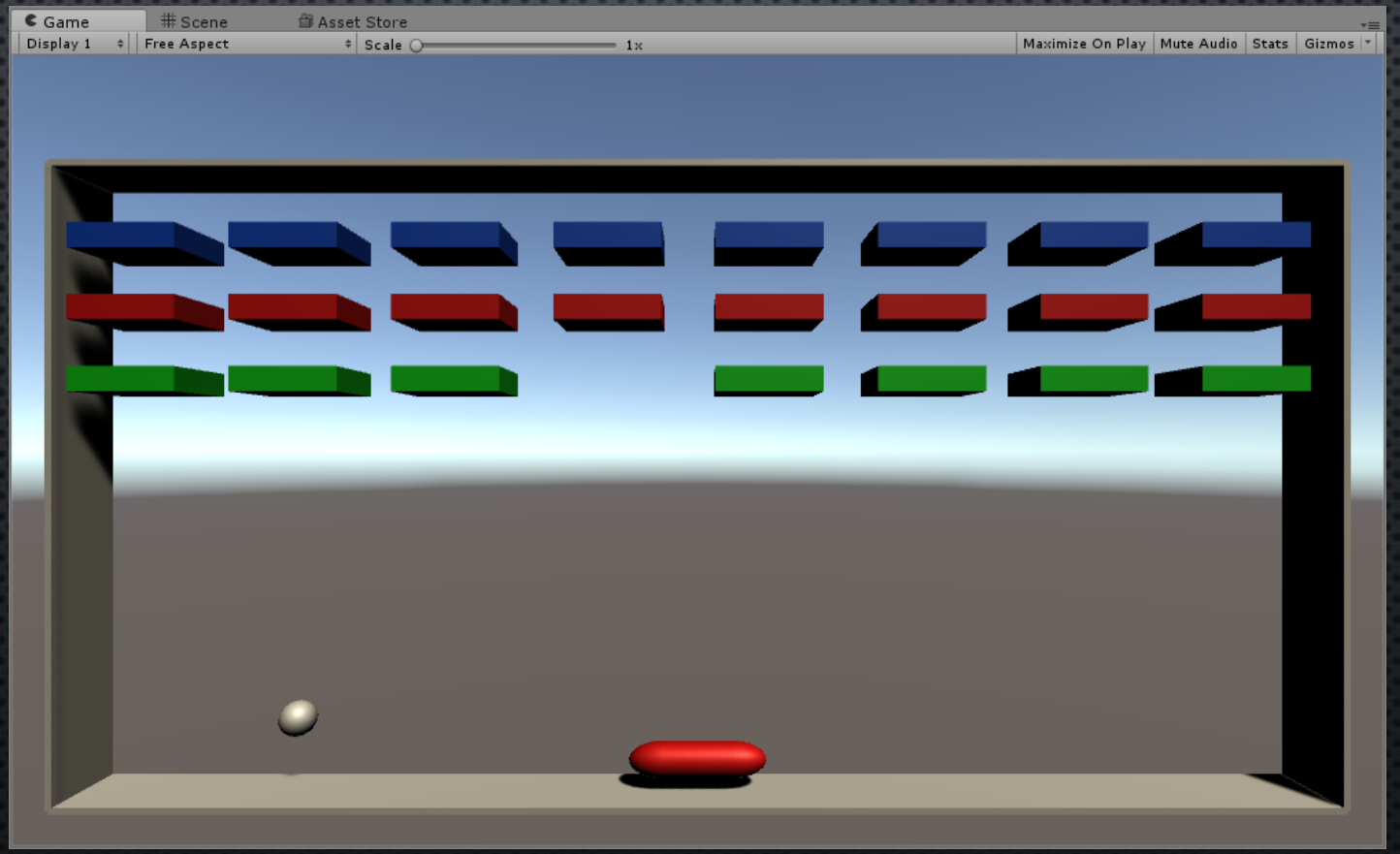


2.



実際に押してみると…

シーンは切り替わったけどなんか暗い！



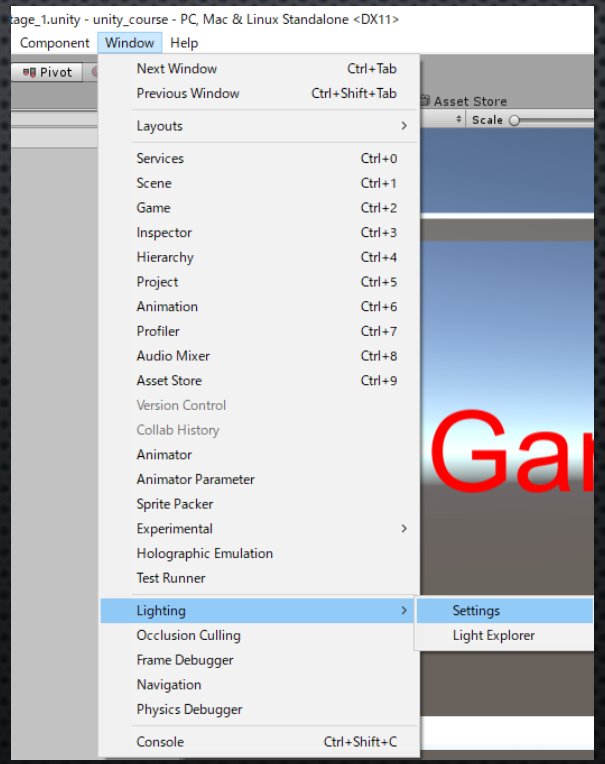
シーン切り替え(Light設定)

シーンの切り替えを行うと移動した先のシーンが暗くなってしまう問題

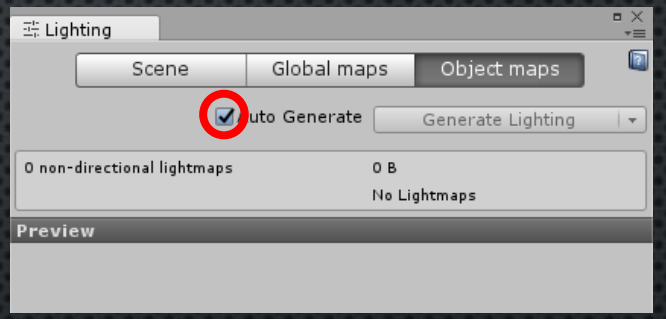
～解決法～

1. シーン(stage_1)を開く
2. メニューバーから
   Windows>Lighting>Settings
3. Object mapsをクリックし、
   「Auto Generate」のチェックを外す
4. Generate Lightingをクリック
5. 読み込みが終わったら完了

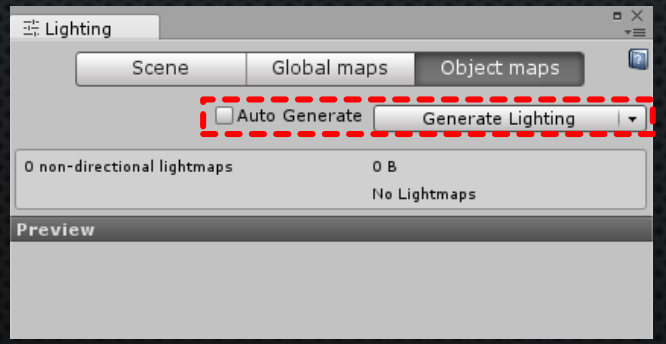
2.



3.

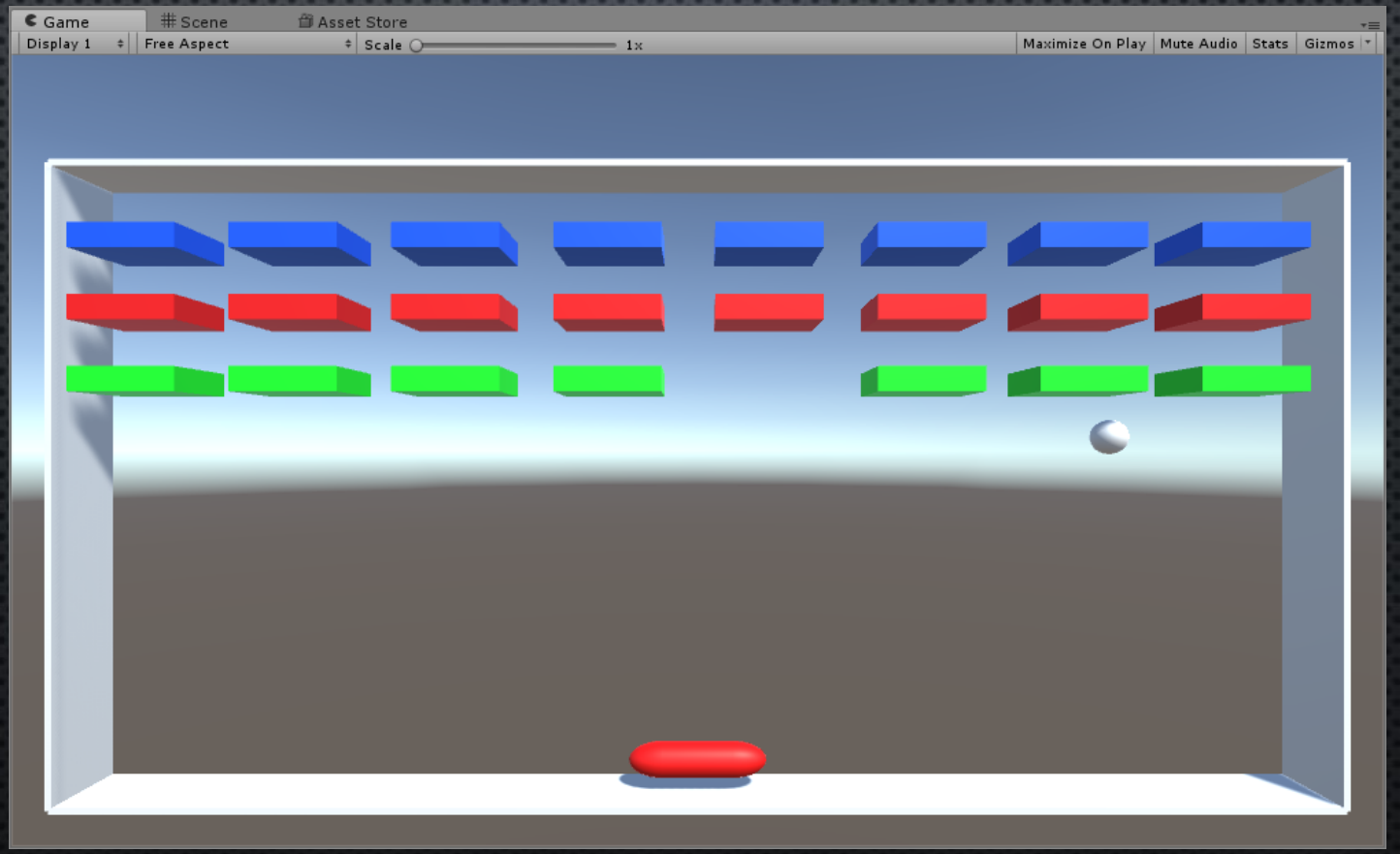


4.



元の明るさに戻った!!

Unity初心者あるあるなのでよく覚えときましょう



シーン切り替え(stage_1からtitleへ)

今度は
ゲームをクリアしてクリックしたらタイトル画面に戻るようにする

1. Stage_Manageを開く
2. コードを追加(右図)

注意!!

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

public class Stage_Manage : MonoBehaviour {

～省略～

// Update is called once per frame
void Update () {
        if (game_clear) {
            if (Input.GetMouseButtonDown(0)) {
                SceneManager.LoadScene("title");
            }
        }
}
}
```

`Input.GetMouseButtonDown(0)`
()内に「0」(左クリック)、「1」(右クリック)を入れて
該当するクリックが行われたらtrueを返し、
そうでない場合はfalseを返す

タイトル画面に戻れた

これで無限に遊べますね！

左クリック

バグの修正

少し触ってみればわかるが…

このゲーム、致命的なバグがいくつかある

- 玉が加速しすぎると壁を抜ける
- 玉が横にしか動かず、縦に進まない

壁抜けバグの修正

そもそもなぜ壁抜けをするのか…

Unityでは0.02秒間おきにオブジェクト同士が衝突してないか判定している

つまり…

衝突判定後、0.02秒以内ならオブジェクトは壁をすり抜けられる

壁抜け！！

発射

壁

更新

0.02秒

物理判定処理:判定無し

壁

更新

0.02秒

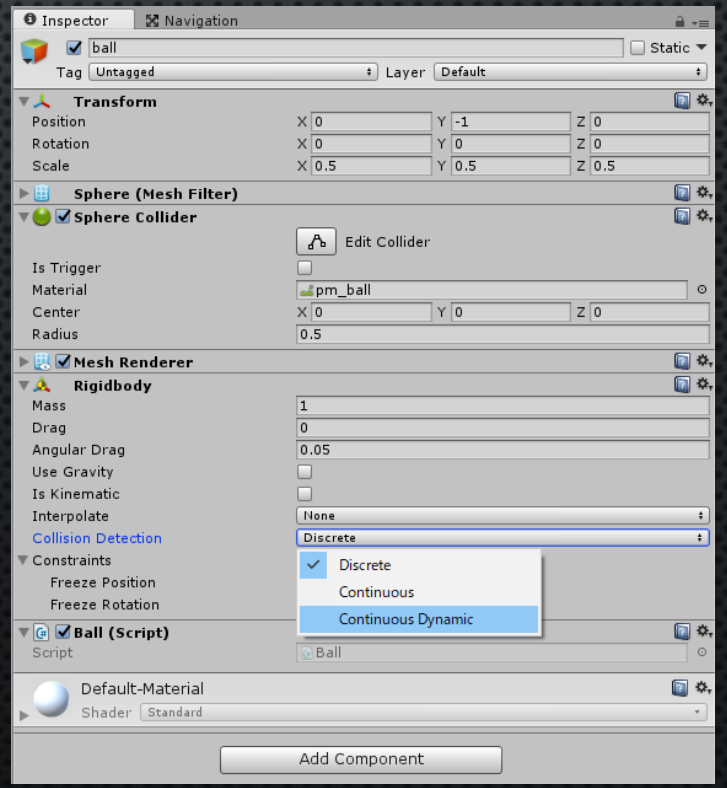
物理判定処理:判定無し

壁

解決策

1. ballオブジェクトを選択
2. Inspector内のRigidBodyの
   Collision Detectionの「Discrete」を
   「Continuous Dynamic」に変更

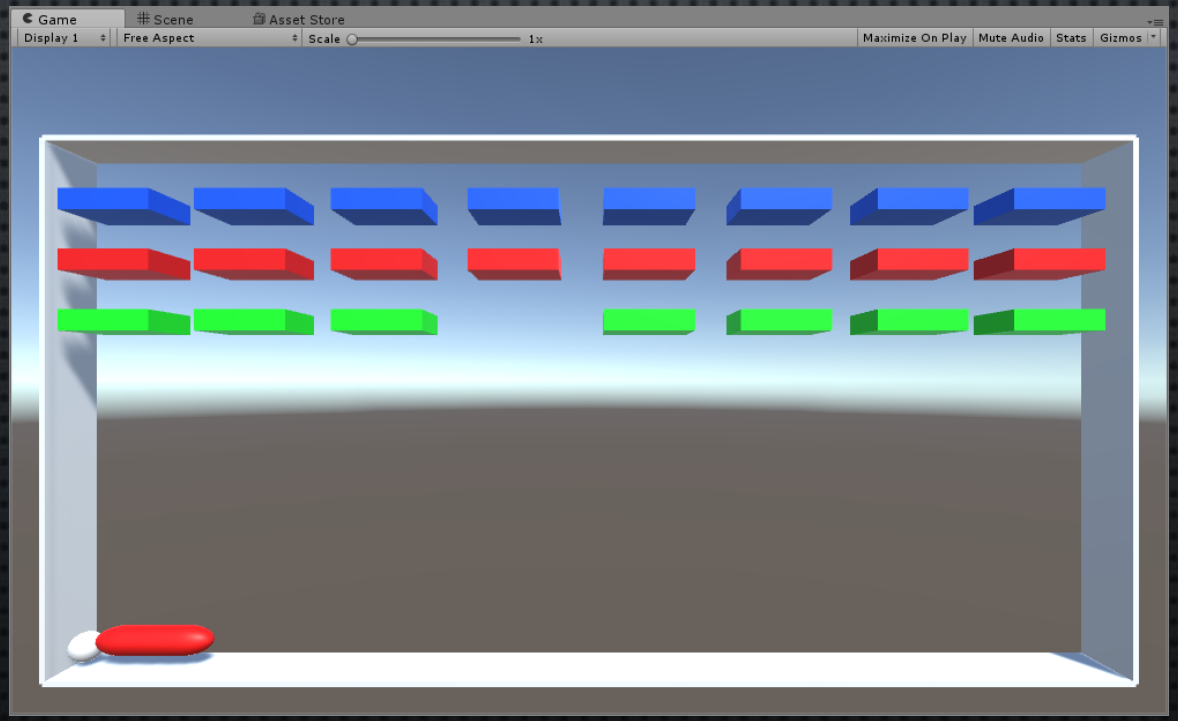
解説は省かせてもらうが(自分も自信がない)
高速で動くオブジェクトにはこの設定をしとけば壁抜け問題を
解決できると思います



玉進まないバグの修正

当たり所が悪いとボールの進まなくなる

こんな感じで壁端でバーを使ってボールを捕まえると勢いが死んで動かなくなる



解決策

1. Ballスクリプトを開く
2. コードを右図のように変更

```
public class Ball : MonoBehaviour {
    Rigidbody rb;
    private Stage_Manage s_manage;
    float n_proceed_count;

    void OnCollisionEnter(Collision other) {
        Debug.Log("test");
        if (s_manage.check_clear()) {
            //玉の動きを止める
            rb.velocity = new Vector3(0, 0, 0);
        }
    }

// 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;
        s_manage = GameObject.Find("stage_manage").GetComponent<Stage_Manage>();
}

// Update is called once per frame
void Update () {
        if (Mathf.Abs(rb.velocity.y) < 0.01)
        {
            n_proceed_count += 1 * Time.deltaTime;
        }
        else {
            n_proceed_count = 0;
        }

        if (n_proceed_count > 3) {
            rb.velocity = new Vector3(rb.velocity.x, 5, 0);
            n_proceed_count = 0;
        }
}
}
```

プログラム解説

Mathf.Abs()
()内に入れた数値の絶対値を返す

RigidBody.velocity
指定したRigidbody の速度ベクトル
velocityの後に「.」を入れてx,y,zの
各速度ベクトルも個別に取得できる
(例: velocity.x　←x軸の速度ベクトル)

Time.deltaTime
おおざっぱに説明すると、この値をかけると
「1フレームおきではなく1秒おき」になる
仕組みとしては起動してるゲームが60fpsの場合
かけてる値に×$\frac{1}{60}$している
n_proceed_count += 2 * Time.deltaTime;
訳:n_proceed_countに2が一秒間で
加算されるように2をフレーム分(60fps)
分割した値を毎フレーム加算している

```
public class Ball : MonoBehaviour {
    Rigidbody rb;
    private Stage_Manage s_manage;
    float n_proceed_count;

    void OnCollisionEnter(Collision other) {
        Debug.Log( “test” );
        if (s_manage.check_clear()) {
            //玉の動きを止める
            rb.velocity = new Vector3(0, 0, 0);
        }
    }

// 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;
        s_manage = GameObject.Find( “stage_manage” ).GetComponent<Stage_Manage>();
}

// Update is called once per frame
Void Update () {
        if (Mathf.Abs(rb.velocity.y) < 0.01)
        {
            n_proceed_count += 1 * Time.deltaTime;
        }
        else {
            n_proceed_count = 0;
        }

        if (n_proceed_count > 3) {
            rb.velocity = new Vector3(rb.velocity.x, 5, 0);
            n_proceed_count = 0;
        }
}
}
```

実行ファイル作成

今現在では

Unity内でしか遊べず、ゲームとしてはまだ「開発モード」の状態である

なので、Unityを持ってない人でも遊べるようにexeファイル化させる

exeファイル化

1. メニューバーから
   File>Build Settingを選択

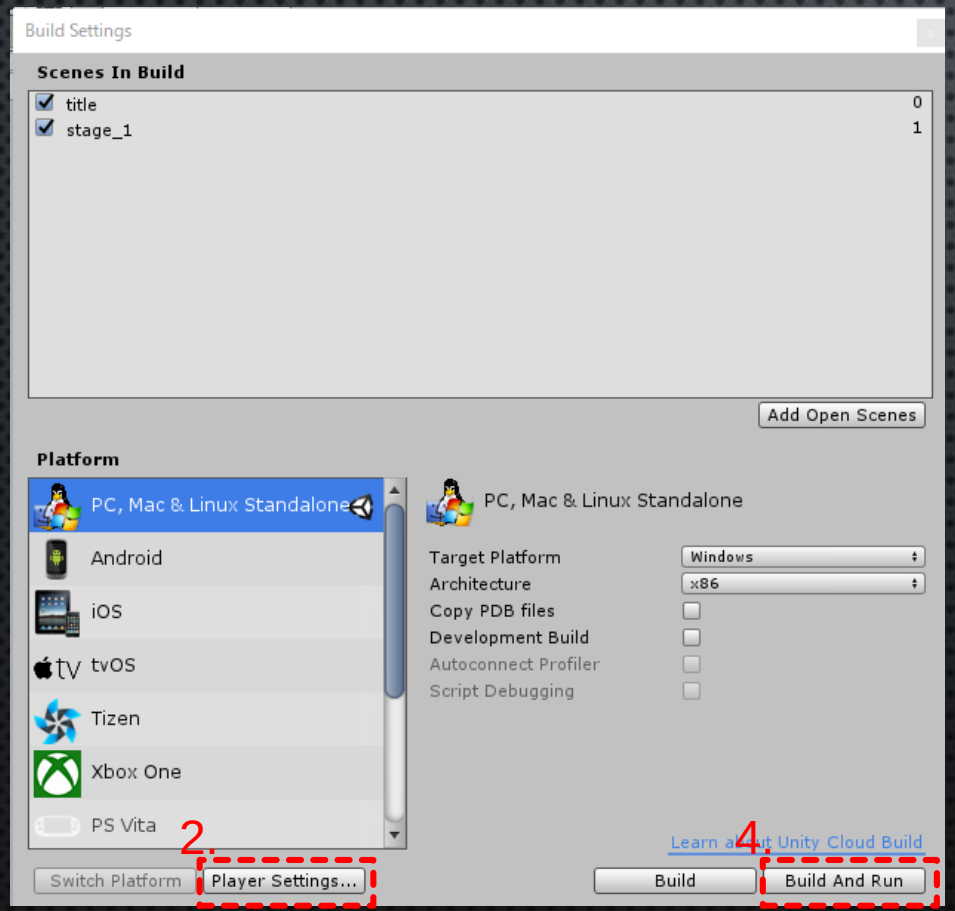
2. Player settingsをクリック

3. Inspector内の
   「Supported Aspect Ratios」の
   「16:9」以外のチェックを外す

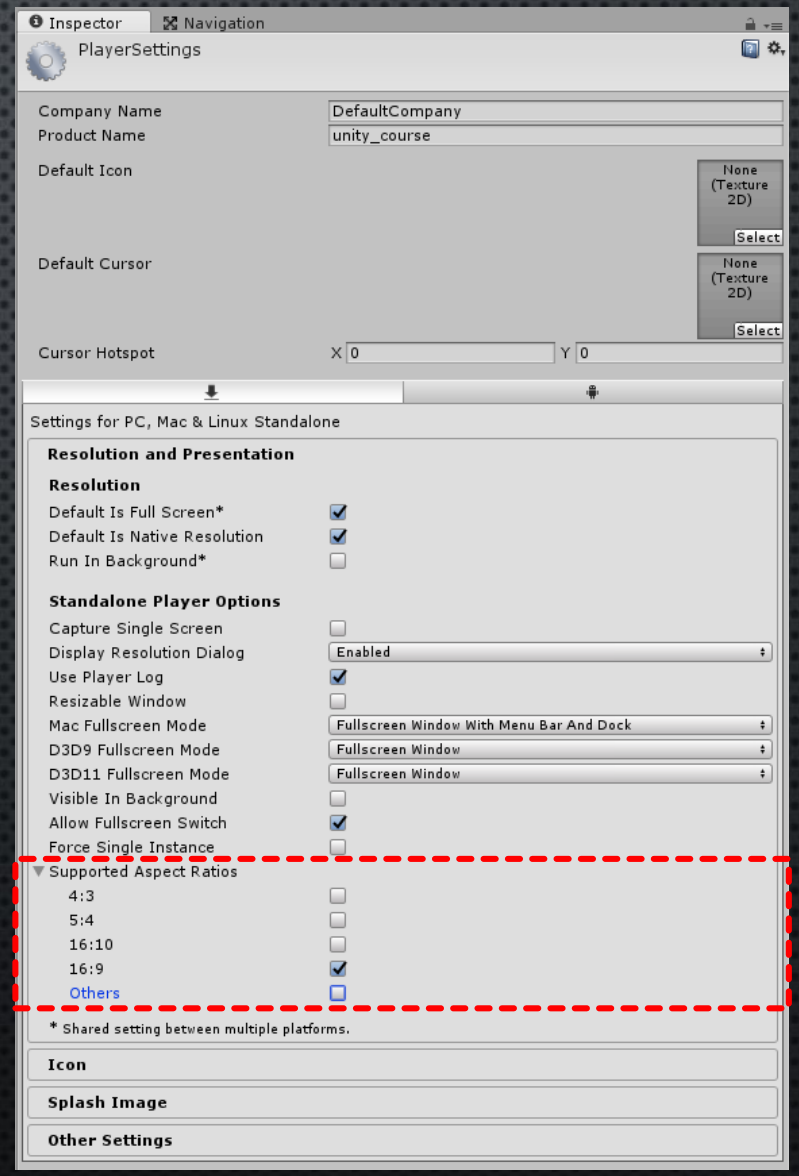
4. 「Build And　Run」をクリック

5. ファイル名はお好みで(ブロック崩しでいいかと)

exeファイル化

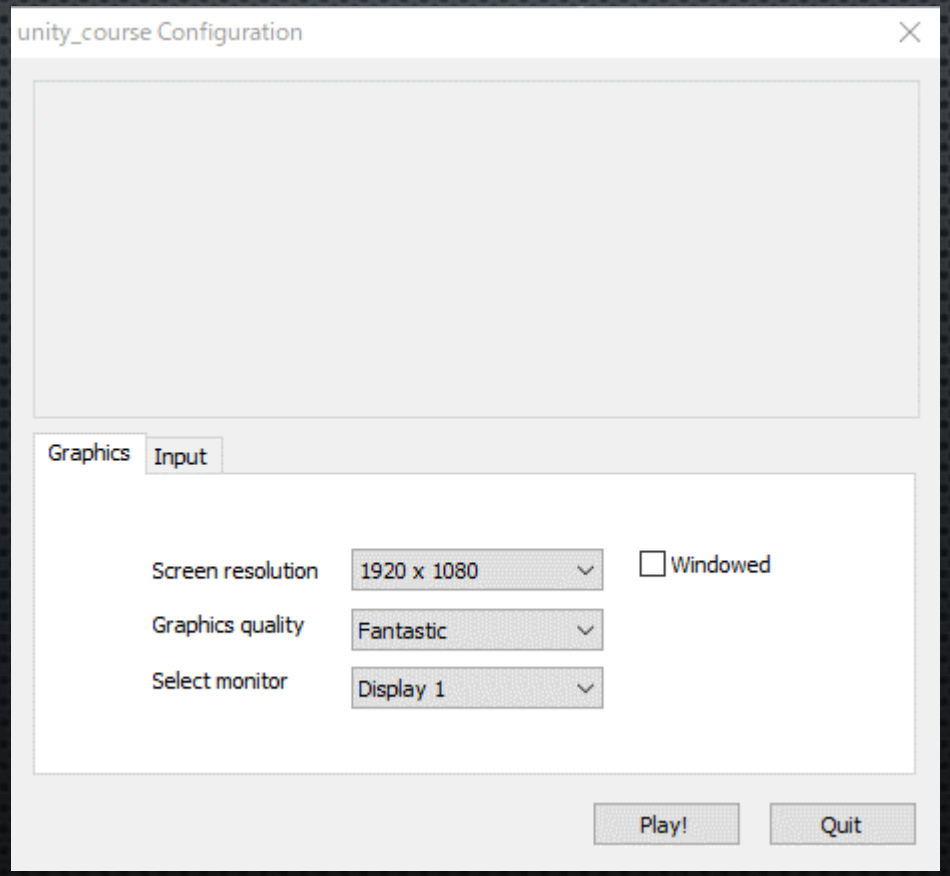

Build Settings
Scenes In Build
title 0
stage_1 1
Add Open Scenes
Platform
PC, Mac & Linux Standalone
Android
iOS
tvOS
Tizen
Xbox One
PS Vita
PC, Mac & Linux Standalone
Target Platform Windows
Architecture x86
Copy PDB files
Development Build
Autoconnect Profiler
Script Debugging
Learn about Unity Cloud Build
Switch Platform
Player Settings...
Build
Build And Run
2
4

3.


Inspector
Navigation
PlayerSettings
Company Name DefaultCompany
Product Name unity_course
Default Icon
None (Texture 2D)
Select
Default Cursor
None (Texture 2D)
Select
Cursor Hotspot X 0 Y 0
Settings for PC, Mac & Linux Standalone
Resolution and Presentation
Resolution
Default Is Full Screen*
Default Is Native Resolution
Run In Background*
Standalone Player Options
Capture Single Screen
Display Resolution Dialog Enabled
Use Player Log
Resizable Window
Mac Fullscreen Mode Fullscreen Window With Menu Bar And Dock
D3D9 Fullscreen Mode Fullscreen Window
D3D11 Fullscreen Mode Fullscreen Window
Visible In Background
Allow Fullscreen Switch
Force Single Instance
Supported Aspect Ratios
4:3
5:4
16:10
16:9
Others
* Shared setting between multiple platforms.
Icon
Splash Image
Other Settings

Playを押して実際に遊んでみよう！

Unity内で遊んだ時と同じように動けばOK!!



今回はここまで

今回はTOEICが原因です

次回はアイテム等ゲーム要素の追加