Unity 講習資料 1

0．はじめに

Unity 部門の講習を始めます。これから皆さんには、Unity というソフトを使ってゲームを作ってもらいます。目標は、文化祭で自作のゲームを一本公開することです。さて、この講習資料についてですが、実際に Unity を動かしながら操作やコマンド、用語などを覚えてもらうことを目的に書いています。が、まあ完璧に覚えるというのは無理なので（自分も覚えてないです)、この資料を片手にゲームを作ってもらえればと思います。また、今はまだわからなくてもいい、本筋からそれる所は「飛ばしてください」と書いてあります。そこは飛ばして、あとで戻ってきたり家で読んだりしてください。

※資料は Unity がインストールされていることを前提としています。

1．Unity をはじめる

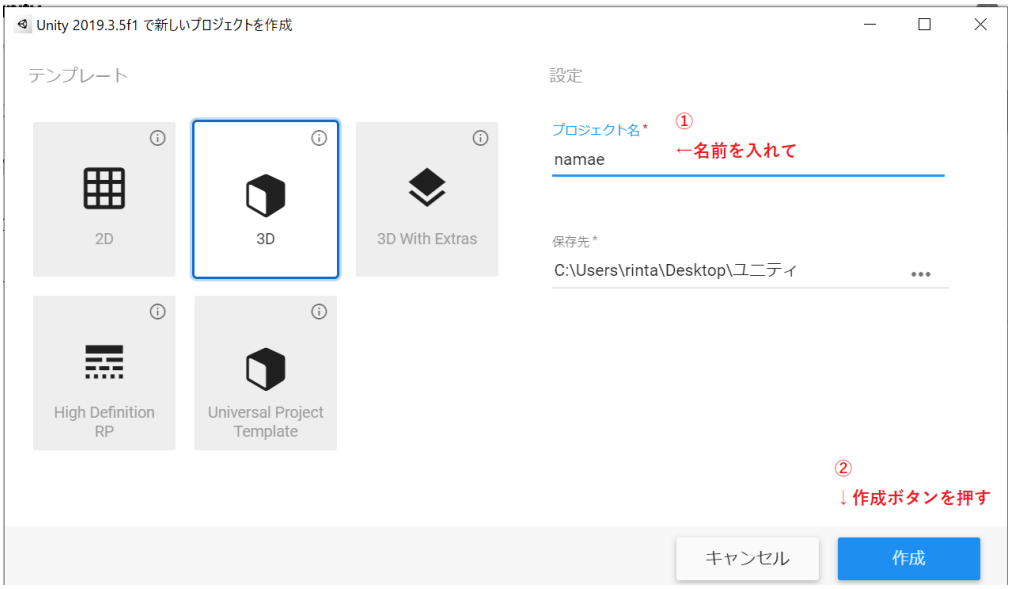
Unity は、世界中のゲームクリエイターが使っているゲーム開発ツールです。「これを使えば簡単にいろいろなゲームを作れる」と思ってくれて構いません。無料で使用できます。さっそく始めてみましょう。

① デスクトップ上の「Unity Hub」のアイコンをダブルクリックして開いてください。するとこんな画面が出ると思います。

Unity Hub は Unity 本体の管理をするソフトです。ここから Unity をインストールしたり、プロジェクトを作ったりできます。

プロジェクト：作るゲームそのもののデータをまとめたもの。

② 新規作成ボタンを押してプロジェクト名に好きな名前を入れて（わかりやすいほうがいいです)、作成ボタンをクリックしてください。しばらく待つと Unity 本体が起動します。



③　上の画面の説明をします。飛ばしてください。

（0）　そもそもこの画面は

Unity Hub で新しくプロジェクトを作るとき、最初に設定ができる画面です。

（1）　テンプレート

２Ｄゲームか３Ｄゲームか、あるいはその設定はどうかを選べます。

２Ｄ：２Ｄゲームに適した設定のプロジェクトを作ります。

３Ｄ：３Ｄゲームに適した設定のプロジェクトを作ります。

３DWithExtras：PostProcessing なる機能を使える３Ｄのプロジェクトを作ります。使うとグラフィックがきれいにできるようです。

HighDefinitionRP：とてもきれいなグラフィックの３Ｄのプロジェクトを作ります。HDRP とか ShaderGraph とかいう機能が使えるようです。

UniversalProjectTemplate：いろんなところで使える３Ｄのプロジェクトを作ります。設定が簡単に変更できるらしいです。

このうち下の三つは、パソコンに負荷がかかりすぎる上、そこまでハイレベルなものを使う必要がないのでやめたほうがいいです。

ということで、２Ｄか３Ｄを選びましょう。

（2）　プロジェクト名

プロジェクトの名前を付けられます。今後も何度か書くつもりですが、名前はわかりやすいものを付けたほうがいいです。「aaa」とかじゃ後になって絶対に「なんだこれ」ってなります。

（3） 保存先

プロジェクトは当然 PC のどこかにデータの形で保存されるのですが、その保存先を決められます。「…」ボタンから保存したい先のフォルダを選びます。部の PC を使っている間はいじらないでください。

（4） プロジェクトの作成

右下の作成ボタンを押すと、いままでした設定に従ってプロジェクトが作られます。そしてしばらく待つと（ここではプロジェクトの構造の作成やプロジェクトの呼び出しが行われています）Unity 本体が新しいプロジェクトを開いて起動します。

飛ばしてください、の部分終わりです。

2．シーンとオブジェクトとコンポーネント

Unity を使う上で重要な概念、シーンとオブジェクト、コンポーネントについて説明します。この先何度も使う用語なので、覚えてください。

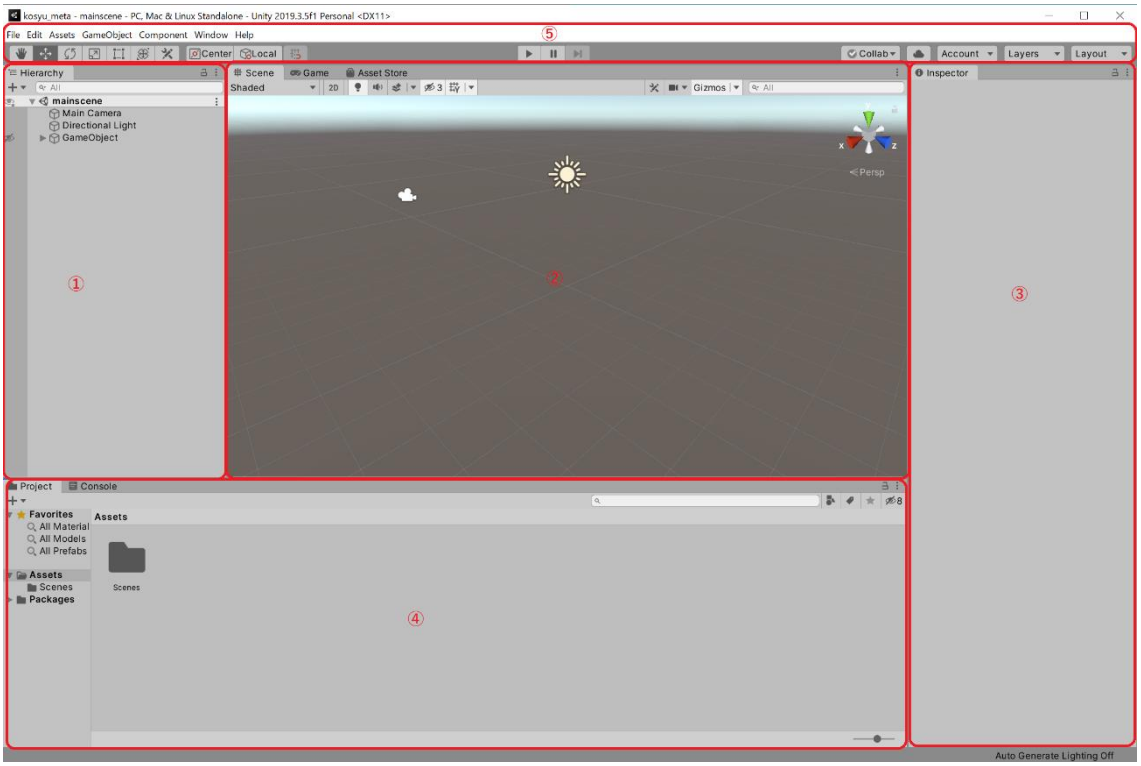
シーン：Unity が作り出した仮想空間のことです。日本語にすれば「舞台」「場面」。このシーンにオブジェクトを配置することで、ゲームが成り立ちます。また、シーンはプロジェクト内で複数制作でき、それぞれを転換することでゲームを作っていきます（タイトル画面、ホーム画面、プレイ画面などを別々のシーンで作り、転換していきます)。

オブジェクト：「もの」のことです。しかし、それ自体では何も為さず（目には見えませんし、形もありません)、「コンポーネント」という情報、機能がつくことで、はじめて役目をもちます。コンポーネントのくっつく場所、というのがより正確です。

コンポーネント：オブジェクトにつく情報や機能のことです。オブジェクトはそれだけでは目には見えないし、動いたりしないし、何かにぶつかることもありません。しかし、そのオブジェクトにコンポーネントを付けると、目に見えるようになり、動き出し、何かにぶつかるようになります。コンポーネントは Unity に初めから設定されているものの他、自分で書いたプログラムもコンポーネントとして扱われます。

3．画面の説明

Unity の画面の説明をします。



① ヒエラルキー

シーンにあるオブジェクトを表示します。

② シーン・ゲームビュー

シーンビューとゲームビューの二つを見れます。

シーンビューではシーンの状況を確認できます。

ゲームビューでは、レンダリング（リアルに表示すること）後のシーンを確認できます。指定のカメラから見た景色が映されます。

シーンビュー、ゲームビューの切り替えは左上のタブでできますが、実行および実行停止時に自動で切り替わるので、必要はないでしょう。

③ インスペクター

選択しているオブジェクトについているコンポーネントを確認、編集できます。

④ プロジェクト・コンソールビュー

プロジェクトビューとコンソールビューの二つを見れます。

プロジェクトビューでは外部ファイルなどの操作ができます。プロジェクトのフォルダ内の Asset フォルダと同期しています。

コンソールビューではバグの表示やデバッグログの表示を行います。

左上のタブで変更できます。

⑤ ツールバー

いろんなことができます。

4．オブジェクトの作成

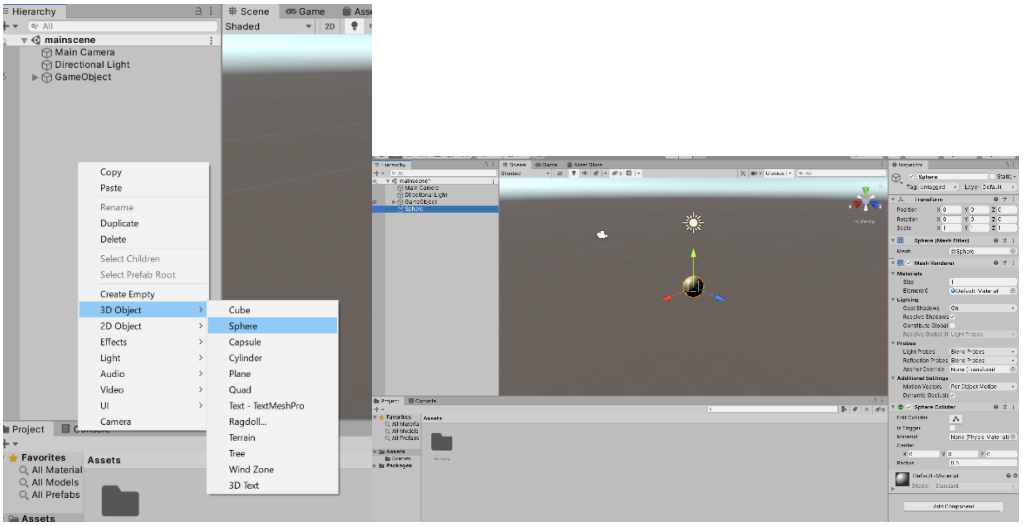
⓪　シーンには作成された時点で既にいくつかのオブジェクトが存在します。簡単にその説明をします。

Main Camera：カメラです。ここから見える景色が最終的なゲームの視点になります。

Directional Light：光源です。Unity は光の当たり方なども計算してくれます。ないと画面が暗くなります。

オブジェクトを作ってみましょう。

①　ヒエラルキー上のどこかを右クリック or 左上の create をクリックし、3D Object>Sphere を選択します。するとシーンに球が作られます。インスペクターにも作った球のコンポーネントが表示されます。



このようにすると、オブジェクトを作れます。

②　作れるものの説明をします。飛ばしてください。

CreateEmpty：実体のないオブジェクトを作ります。ゲームの進行管理などに使います。

Cube：立方体を作ります。

Sphere：球を作ります。

Capsule：カプセル（昔の錠剤みたいな形）を作ります。

Cylinder：円柱を作ります。

Plate：厚さ 0 の直方体を作ります。床や壁に使用します。

Quad：厚さ 0 の目に見えない直方体を作ります。画像を張り付けて使用します。

Text,Regdoll,Terrain,Tree,WindZone,3DText：難易度が高かったりアセットストアの使用を前提としているのでやりません。気になった人は自分で調べて挑戦し

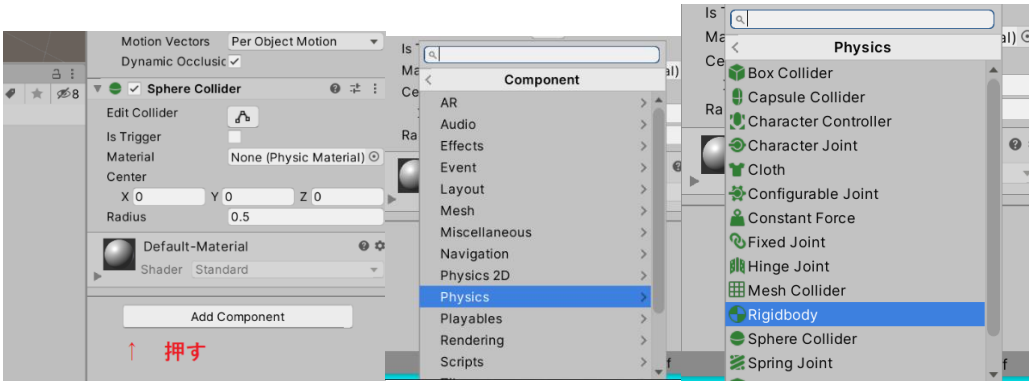
てみてください。
２D Object：２D のオブジェクトを作ります。そのうちやります。
Effect：エフェクトを作ったり制御できます。扱う予定はありませんがおもしろいのでやってみることをおすすめします。
Light：光源を作れます。
Audio：音声にかかわるものを作ったりできます。
UI：UI（入力や表示のこと）に関するものを作れます。（ボタンなど）
Camera：カメラ（視点）を作ります。
飛ばしてください、の部分終わりです。

5．コンポーネントを付ける

⓪　オブジェクトは作った時点ですでにいくつかのコンポーネントがついています。簡単にその説明をします。
Transform：位置の情報。あとでやります。
Sphere(Mesh Filter),Mesh Renderer：どちらもオブジェクトの描画に関するものです。
Sphere Collider：当たり判定です。
Default-Material：オブジェクトの材質や色の情報です。

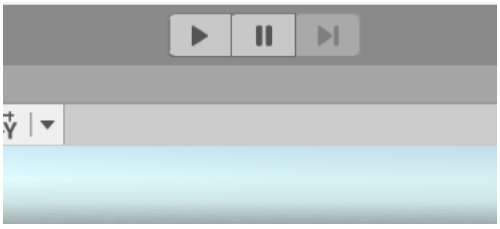
球に物理演算をさせるコンポーネントをつけて、重力をかけてみましょう。

①　球を選択した状態で（シーンの球をクリックすると選択できます）インスペクターの一番下にある Add Component ボタンを押し、Physics＞Rigidbody を選択します。これで球に Rigidbody コンポーネントが追加されました。

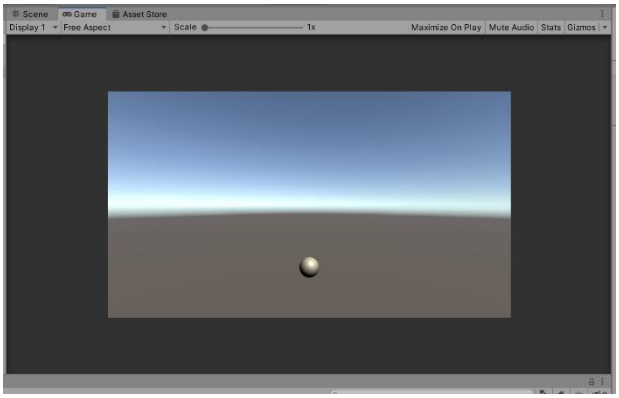


Rigidbody 以外の様々なコンポーネントも Add Component ボタンから追加できます。
Rigidbody：これがついたオブジェクトに物理演算をさせるということを意味するコンポーネント。ここでいう物理演算とは、力を受けた時の挙動のほか、重力を含みます。

② シーン・ゲームビューの上の再生ボタン（▶）をクリックしてください。



すると視点が変わり球が落ちるのが見えたと思います。

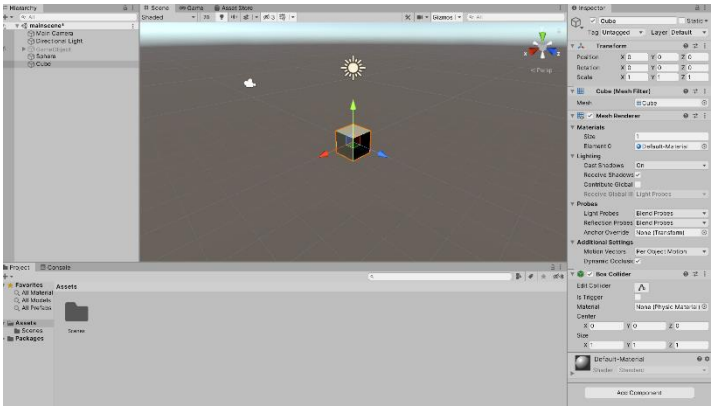


③ もう一度再生ボタンをクリックし、もとの画面に戻りましょう。球ももとの位置に戻ります。
再生ボタン：押すとゲームが実行されます。つまり、入力を受け付けたり、物理演算を開始したりします。

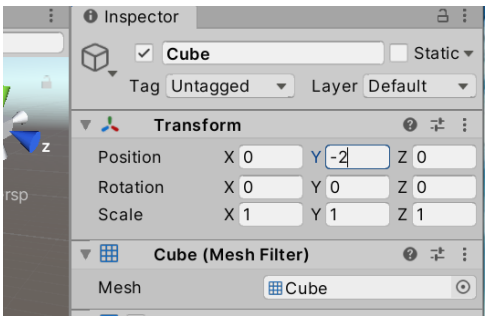
6．足場を作る

球が落ちないように足場を作ります。

① ヒエラルキー上で右クリックし、3D Object > Cube を選択します。すると球があった位置に立方体が出現します。重なっているだけで、球は消えていません。



② インスペクターの上側の Transform の Position の Y を、０から－２にします。欄をクリックしてキーボードで入力してください。



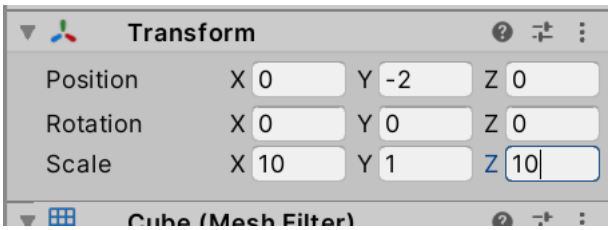
すると立方体の位置が変わりました。コンポーネントの数値もインスペクターから編集することができます。

Transform：オブジェクトの位置や大きさの情報を表すコンポーネント。

Position：Transform コンポーネントのもつ情報の一つで、位置の情報。座標空間上での座標を表す。

座標空間：座標平面（数学でやります）の立体版。ある一点を基準とし、そこから前、横、上方向にそれぞれどれだけ離れているかを表す。

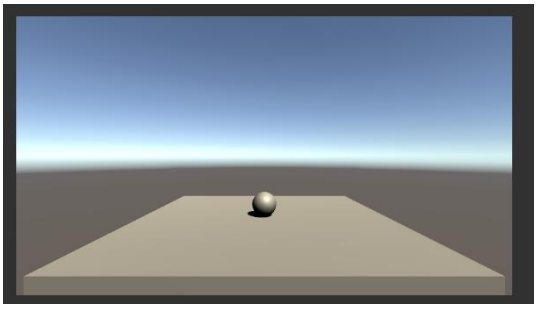
③ このままだと狭いので大きくします。Transform の Scale の X,Z をどちらも１０にしてください。



立方体が縦横に大きくなりました。

Scale：Transform のもつ情報の一つで、大きさの情報。

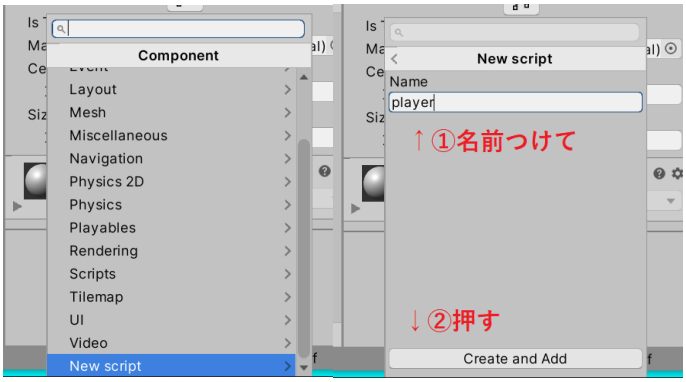
④ 再生ボタンを押して再生してみましょう。球が足場に落ちて止まったのがわかります。



7．動かせるようにする

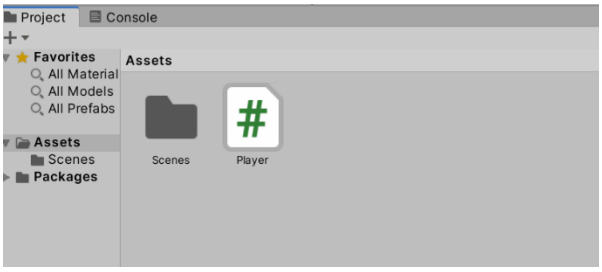
球を自分で動かせるようにします。

① シーン上の球をクリックし、インスペクターから Add Component > New Script を選択し、名前を Player とつけ、create and add を押します。

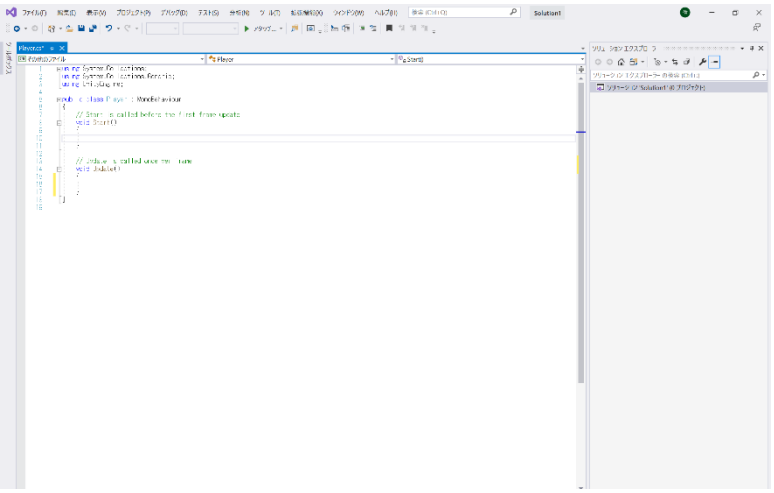


スクリプト：プログラムのこと。よくわからない英語を書いてなんかが動く、という一般的なイメージのプログラムのこと。Unity ではスクリプトもコンポーネントのひとつとして扱われます。

② プロジェクトビュー（画面下）にできた Player と書かれたものをダブルクリックします。

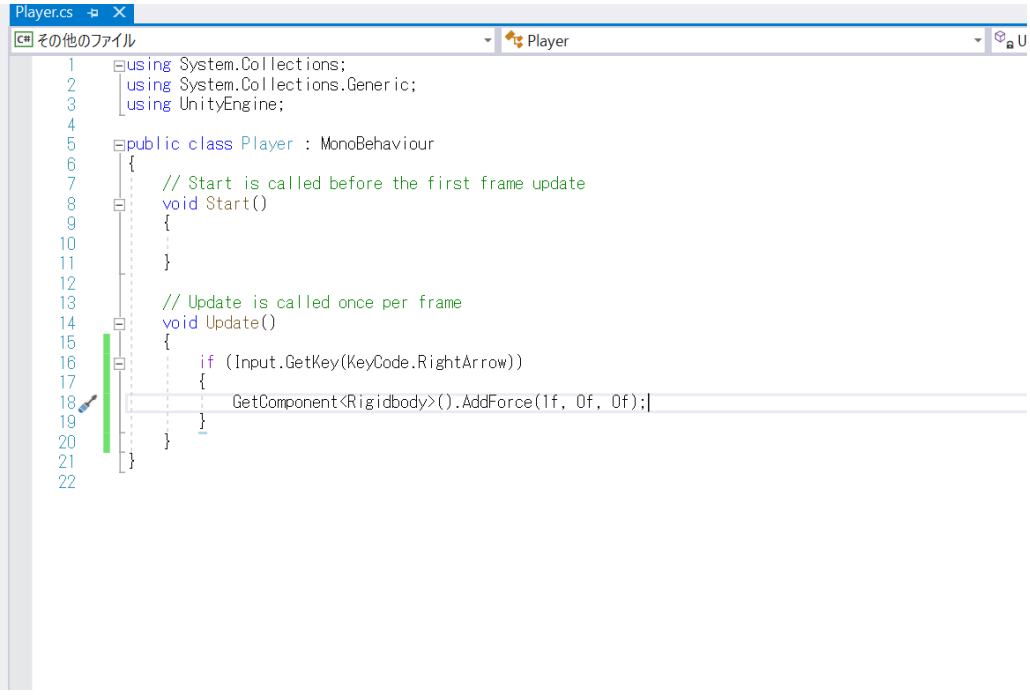


しばらく待つと「Visual Studio」もしくは「Visual Studio Code」という別のソフトが起動します。

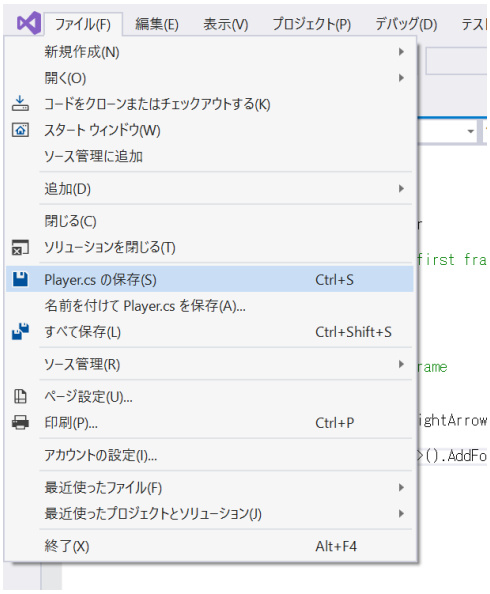


「Visual Studio」：スクリプトを書くソフト。スクリプトはしょせん文字なので、メモ帳でも書けるのですが、これにはプログラムを書くうえで便利な機能がついています。Visual Studio Code も同じです。

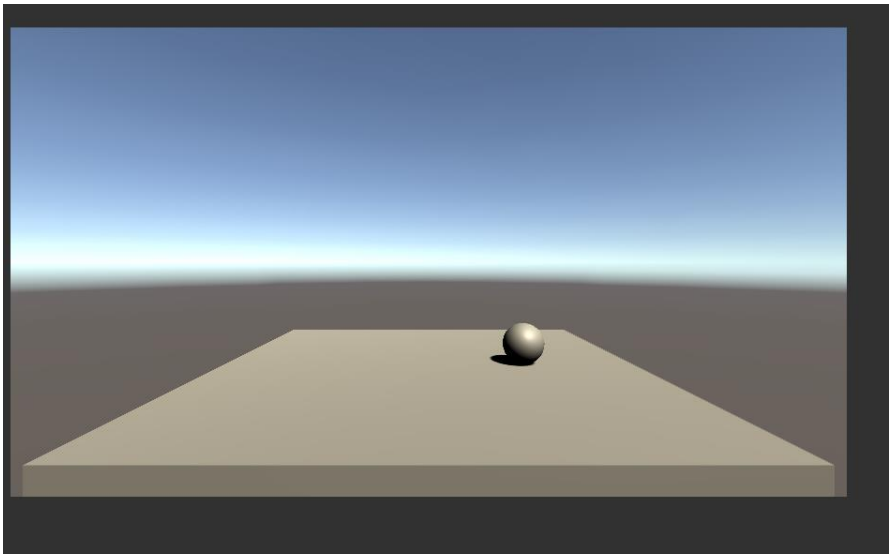
③ 以下のように打ち込んでみましょう。大文字と小文字の区別は正確に。



④ 右上にあるファイル（File）＞Player.cs の保存(save,セーブ)　を選択し、保存します。Ctrl キー＋S キーでも保存できます。保存しないと変更が反映されないので注意。



⑤ シーンを再生して、→キーを押してみましょう。ボールが右側に動いていきました。



8．７の解説

元から書いてあるやつ　こっちは流し読みでいいです。

１～３行目：Unity のコマンドを使用する、という定義。いじらないでください。

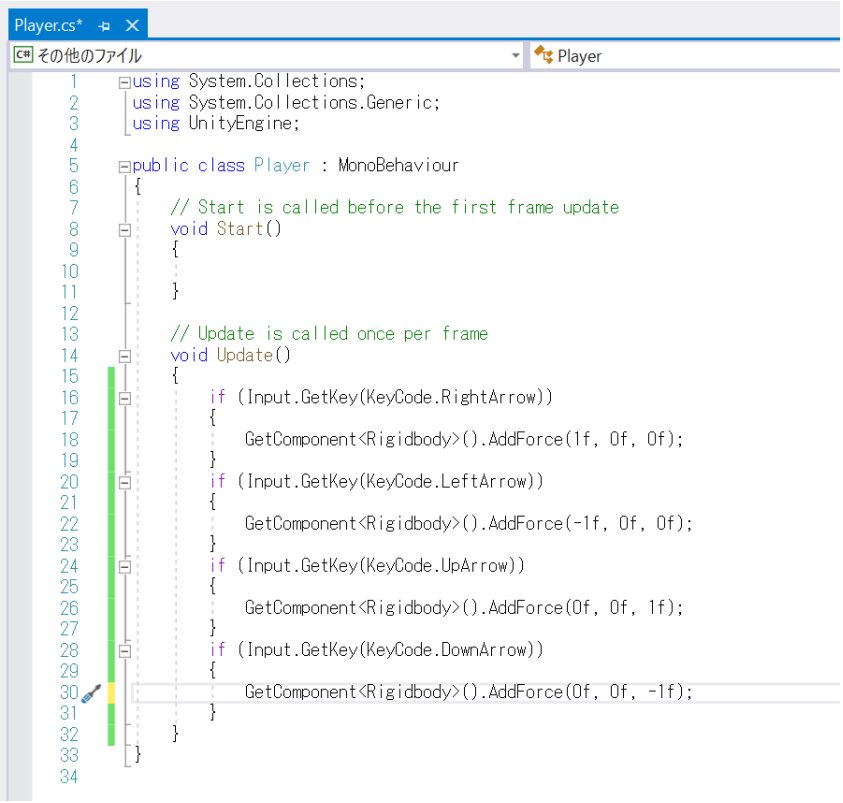
５行目：Player というクラス（わかんなくていいです）の宣言。この中に書いたことがシーンに反映されると思ってください。いじらないでください。

８～１１行目：このスクリプトが呼び出された最初の一回だけカッコ内の処理を実行する、という意味。

１４行目：毎フレームカッコ内の処理を実行する、という意味。
フレーム：ゲームの処理時間の単位。初期設定では１フレーム＝0.0166…秒であり、0.0166…秒に一回処理が行われます。
７、１３行目：コメントアウトされた説明。「//」と打った後の文はプログラムに無視されます。これをコメントアウトといいます。これでプログラムの説明を行ったり、一時的に問題のあるプログラムを処理させなくしたりします。

書き足したやつ　こっちはちゃんと読んでください。
if(~){…}：もし～なら…する、の意味。一番よくつかうプログラムで、基本中の基本。～には真偽を判断できる文が書かれる。
Input.GetKey(KeyCode.~)：～キーが押されているかいないかを返すプログラム。この場合は RightArrow、つまり右矢印キーが押されているかどうか。
GetComponent<Rigidbody>().AddForce(x,y,z);：球に力をかけるプログラム。x,y,z はそれぞれ左右、上下、前後にどれだけ力をかけるかを表します。今回は 1f,0f,0f なので、右方向にのみ 1 だけ力がかかります。f についてはあとで説明します。また、文の最後にある「;」（セミコロン）は、命令の最後に必ずつけなければいけない記号です。
以上より、このプログラムは「もし→キーが押されているなら、球に右方向に力をかける」という意味になります。

９．もっと動かせるようにする
右以外にも動かせるようにします。
①　以下のように加筆して保存してください。コピー＆ペーストを使うと楽です。



② 再生ボタンを押すと、押したキーの方向に球が動きます。

１０． シーンの保存

① このあたりでシーンを保存しましょう。保存はこまめに行わないと、PC が落ちたときに今までの作業が水の泡となってしまいます。今回は初めての保存なので、ファイルに名前を付けてもらいます。右上の File→Save As を選択し、mainscene と名付けて保存してください。

② 上書き保存の方法は、プログラムと同じように File→Save もしくは Ctrl＋S キーです。これからは保存の指示はしませんが、適宜保存をしてください。

１１． 変数

プログラムには「変数」という概念があります。その解説をします。

変数とは、データを入れる箱です。その中には数値だったり文字だったり、Unity ではオブジェクトなんかも入れられます。しかし、中に入れられるデータの種類は決まっており、数値を入れる箱に文字を入れることはできません。この中に入れられるデータの種類は「型」と呼ばれます。

変数にできることは主に①宣言、②代入、③参照の３つです。

① 宣言

まず初めに、「こんなデータが入れられるこんな名前の箱をつくります！」ということ。変数は宣言しないと使えません。

具体的には

int A ;

のように書きます。これは、int 型（整数を入れられる型）の変数 A を作る、という意味です。

② 代入

宣言した変数のなかにデータを入れ、書き換えること。

具体的には

A = 2;

のように書きます。ここでの「＝」は数学的な＝とは違い、左のデータを右の変数に代入する、という意味です。そのため、

A = 2;

A = 3;

のように書いても問題ありません。（このプログラムは、A の中身を 2 にした後、3 にする、という意味です。そのため、現在の A の値は 3 です。2 というデータは消えます）

また、宣言と代入は同時に行えます。

int A = 3;

と書けば初めから 3 が入った変数 A がつくれます。

③ 参照

変数は中身のデータと同じようにふるまいます。

例えば、

int A = 2;

int B;

B = A + 3;

と書いたとき、B の値は 5 になります。

１２．　変数を使う

変数を使ってみましょう。

① プログラムを以下のように書き換え、保存します。（背景が変わったのは気にしないでください）

```
Player.cs
Assets > Player.cs > Player > Update()
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

0 references
public class Player : MonoBehaviour
{
    5 references
    public float Speed;
    // Start is called before the first frame update
    0 references
    void Start()
    {
        Speed = 0.5f;
    }

    // Update is called once per frame
    0 references
    void Update()
    {
        if (Input.GetKey(KeyCode.RightArrow))
        {
            GetComponent<Rigidbody>().AddForce(Speed, 0f, 0f);
        }
        if (Input.GetKey(KeyCode.LeftArrow))
        {
            GetComponent<Rigidbody>().AddForce(-Speed, 0f, 0f);
        }
        if (Input.GetKey(KeyCode.UpArrow))
        {
            GetComponent<Rigidbody>().AddForce(0f, 0f, Speed);
        }
        if (Input.GetKey(KeyCode.DownArrow))
        {
            GetComponent<Rigidbody>().AddForce(0f, 0f, -Speed);
        }

    }
}
```

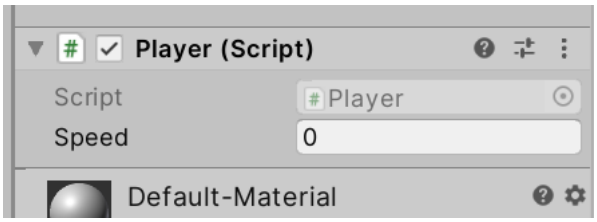
float：小数を扱う型。この型に代入する数値の最後には「f」がつきます。
public：宣言の前に public とつけると、このスクリプトの外（別のスクリプトやインスペクターなど）からの操作を受け付ける変数ができます。逆に、操作を受け付けない変数は private と書きます。省略すると private になります。

② 再生ボタンを押すと、先ほどより遅いスピードで球が動きます。
これは与える力が 1 から Speed（ここでは 0.5）になったからです。

③ Speed = 0.5f;
を
Speed = 10.0f;
に書き換えてみましょう。
すると、動く速度がとても速くなります。
変数を使うメリットの一つは、このように書き換えが 1 回で済むところにあります。

④ Speed =10.0f;
を削除してみましょう。
ここで球のコンポーネントを見ると、Player の下に Speed という欄ができていま

す。これを１００に書き換えてみましょう。



球の動きがとても速くなりました。このように、public 変数は Unity エディター上からも編集できます。

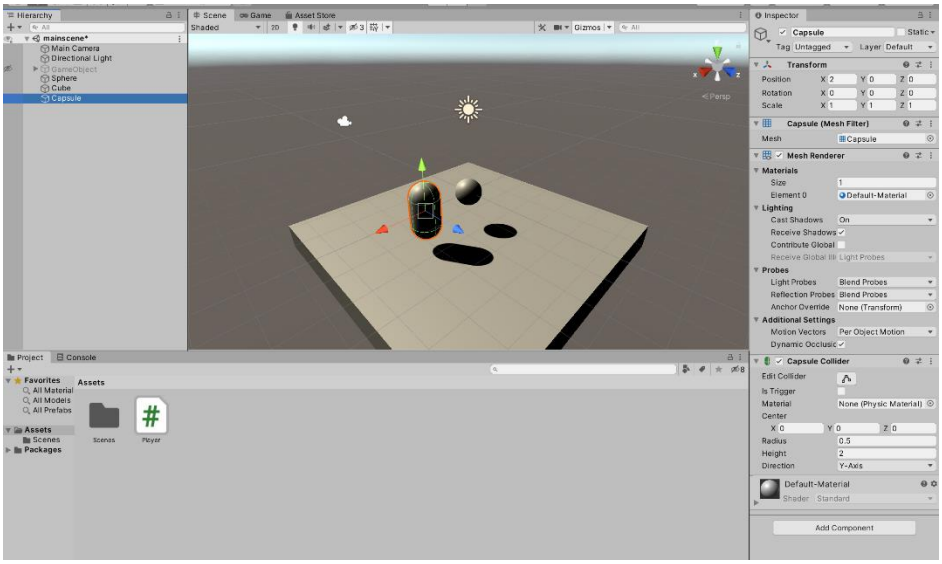
⑤ Speed は 5 くらいに戻してください。

１３．　アイテムを作る

このゲームはアイテムを集めるゲームにしようと思います。そのため、アイテムを作ります。

① 3D Object の Capsule を作ります。

② Capsule の x 座標を 2 にします。



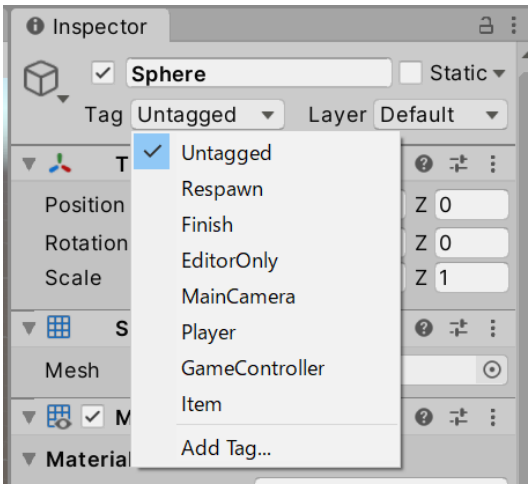
この時点では Capsule は空中に浮かぶ障害物です。

③ 触ったら消えるようにします。Player スクリプトを以下のように加筆して保存してください。

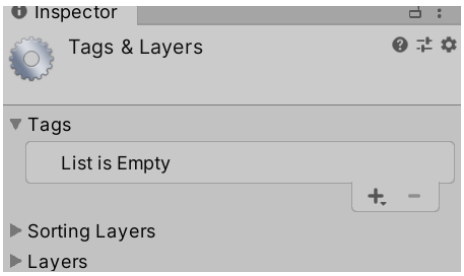
```
        if (Input.GetKey(KeyCode.DownArrow))
        {
            GetComponent<Rigidbody>().AddForce(0f, 0f, -Speed);
        }

    }

    0 references
    void OnCollisionEnter(Collision Hit)
    {
        GameObject obj = Hit.gameObject;
        if(obj.CompareTag("Item")){
            Destroy(obj);
        }
    }
}
```

36 行目：このオブジェクトが何かと衝突したときにカッコ内の処理を実行し、またその衝突判定（Collition）を Hit と名付ける、という意味。
38 行目：GameObject 型の変数 obj を宣言し、Hit の（当たった）オブジェクトを代入する、という意味。Hit.gameObject の「.」は「複数の情報をもつもののうち、これ」という感じの意味です。今後も出てきます。
GameObject 型：オブジェクトを入れる型。
39 行目：obj（当たったオブジェクト）のタグが Item ならカッコ内の処理を実行。タグについてはあとで解説します。
40 行目：obj を消滅させる、という意味。
Destroy(オブジェクト)：オブジェクトをゲーム内から消すプログラム。

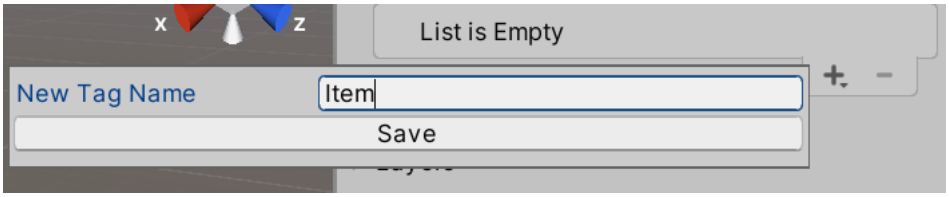
④ このままだと動かないので、Unity でタグに関する編集をします。
タグ：コンポーネントの一種で、オブジェクトの識別によく使われます。インスペクターの一番上から編集できます。



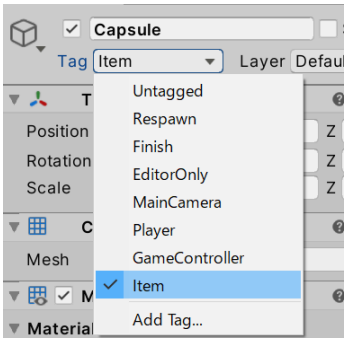
最初は Item というタグはないので、Add Tag を押して追加します。



＋を押して、Item と入力して save ボタンを押します。



Item タグが追加されたので、Capsule のタグを Item に変更します。クリックでできます。

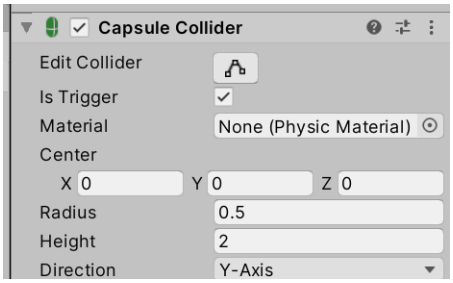


⑤ 実行すると球が衝突したときにアイテムが消滅します。

１４．　アイテムの挙動を変更する

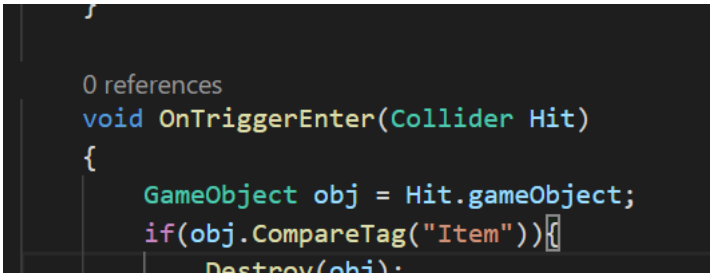
このままだとアイテムに衝突したときに一瞬止まってしまいます。別にこれでもいいのですが、プレイのスムーズさを考えて止まらないようにします。

① Capsule の Capsule Collider にある Is Trigger にチェックをいれます。



こうすると、球が衝突せずにすり抜けるようになります。実行すると、アイテムが消えずに球をすり抜けているのがわかります。

② プレイヤースクリプトを以下のように書き換え、保存してください。



OnTriggerEnter は OnCollisionEnter の Is Trigger 版です。Is Trigger なオブジェクトに触れるとカッコ内の処理が実行されます。また、()の中身が Collision から Collider に変更されていますが、Collision が衝突の情報を表すのに対し、Collider はコライダーという当たり判定のコンポーネントを表します。(この辺は複雑なのでいつか改めてやります)

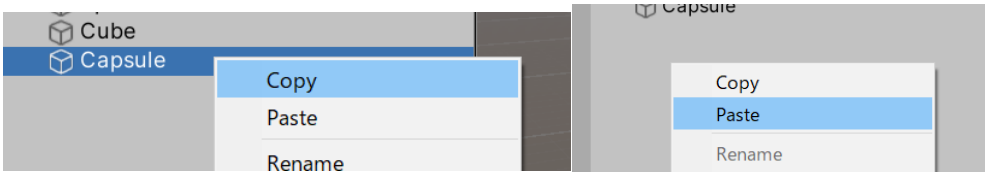
③ 実行すると、衝突時に速度を落とさずに消えるのが分かります。

１５．　オブジェクトのコピペと削除と命名

アイテムが一個だとつまらないので、増やします。しかし、これと同じものを一から作るのは面倒なので、便利な機能を使います。

① コピペをします。

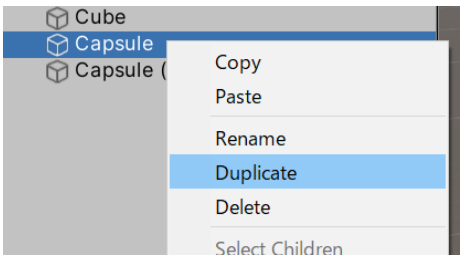
ヒエラルキーから Capsule を右クリックし、copy を選択し、ヒエラルキー上で右クリックし、paste を選択します。



すると Capsule がもう一つできます。Position の X を 2 にするとわかりやすいです。

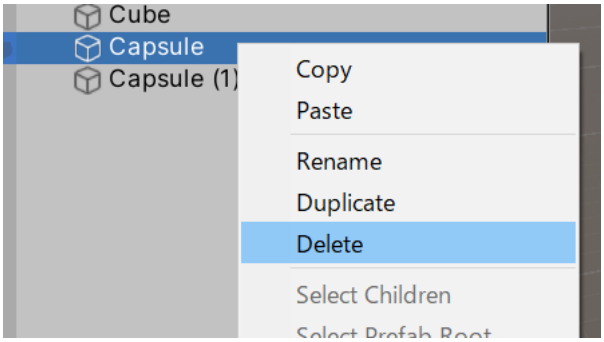
② 別のコピペの方法を教えます。

ヒエラルキーから Capsule を右クリックし、Duplicate を選択します。これだけでもコピペができます。

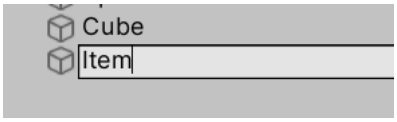


③ これでもいいのですが、もっと便利な方法があるので、そちらの方法で進めま

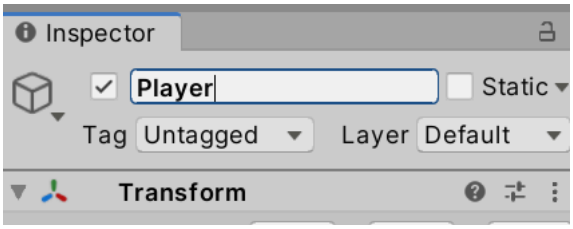
す。せっかく作ったオブジェクトですが、削除してください。ヒエラルキーから Capsule(1)を右クリックし、Delete を選択します。これでオブジェクトが削除されます。Capsule(2)も削除してください。Capsule は残しておいてください。



④ さっきから Capsule と言っていますが分かりづらいので、名前を変えます。Capsule を右クリックし、Remene を選択して Item と名付けてください。



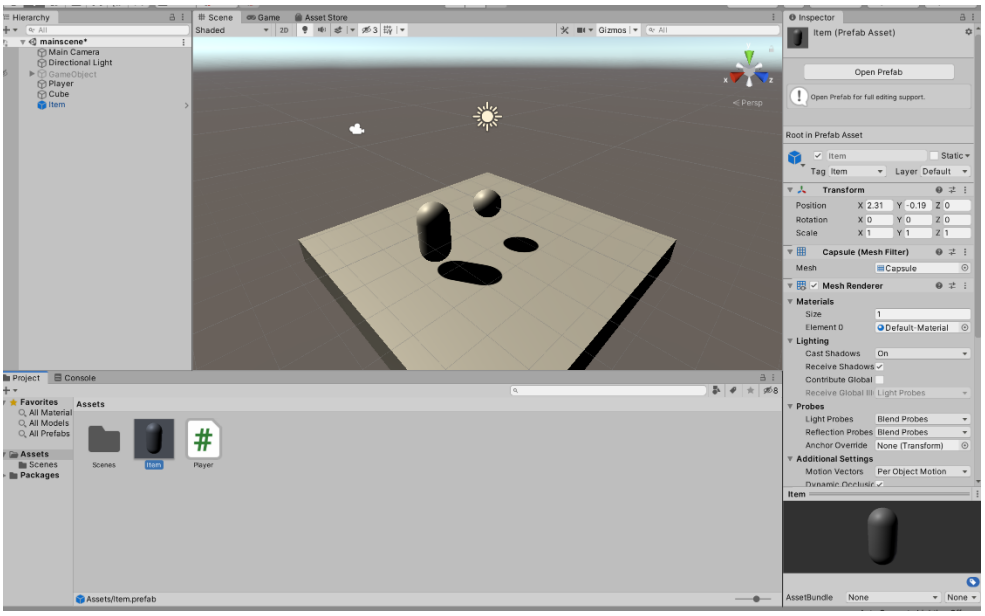
ついでに球の名前も Player に変えましょう。別の方法を教えます。
球を選択した状態で、インスペクターの一番上の Sphere と書かれた部分を選択し、Player に書き換えましょう。



１６．　プレハブとインスタンス化

Unity にはプレハブという機能があります。これは、オブジェクトの設計図のようなもので、これを使ってスクリプトからオブジェクトを作ることができます。プレハブを使ってスクリプトからオブジェクトを作ることをインスタンス化といいます。

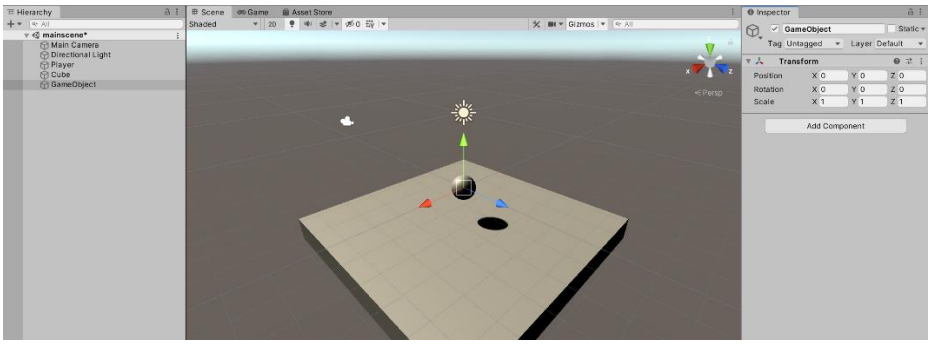
① まずはプレハブを作ります。ヒエラルキーの Item をプロジェクトビューにドラッグアンドドロップします。



すると、プロジェクトビューになんかができます。これがプレハブです。
このプレハブは、もとにしたオブジェクト（ここでは Item）と同じデータを持っています。

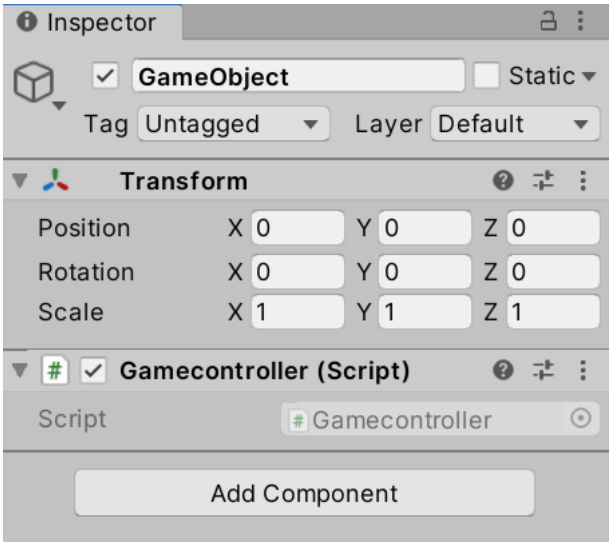
② もとにした Item は削除しましょう。

③ インスタンス化をつかさどるオブジェクトを作ります。ヒエラルキーを右クリックし、Create empty を選択します。すると GameObject ができます。



画面には何もできていないように見えますが、これは GameObject が実体を持っていないからです。

④ Add Component>New Script から GameController という名前でスクリプトを追加します。



⑤ 以下のように GameController を編集してください。

```
Gamecontroller.cs
Assets > Gamecontroller.cs > Gamecontroller > Update()
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
    0 references
5   public class Gamecontroller : MonoBehaviour
6   {
        1 reference
7       public GameObject Item;
        1 reference
8       public Vector3 ItemPosition;
9       // Start is called before the first frame update
        0 references
10      void Start()
11      {
12          Instantiate(Item,ItemPosition,Quaternion.Euler(0,0,0));
13      }
14
15      // Update is called once per frame
        0 references
16      void Update()
17      {
18
19      }
20  }
21
```

８行目：Vector3 型の変数 ItemPosition を宣言します。

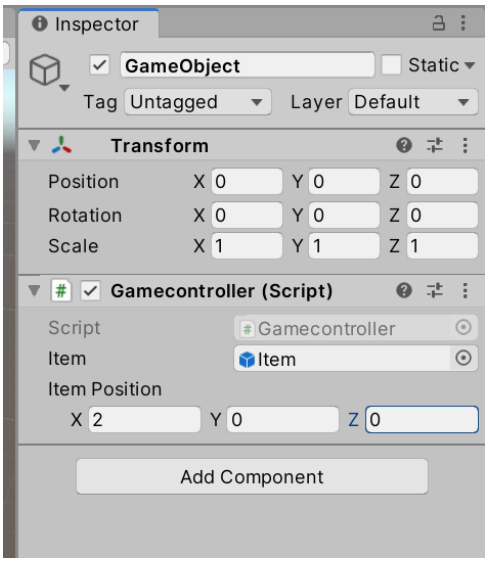
Vector3：数値を三つ持つ変数の型。位置情報などにつかいます。

１２行目：ItemPosition に Item を生成します。

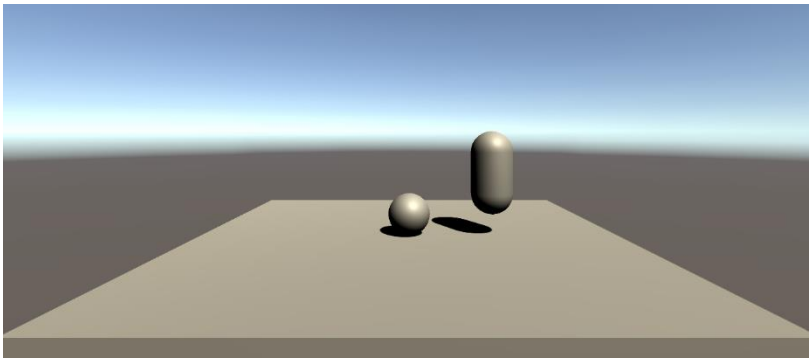
Instantiate(オブジェクト,位置,角度)：オブジェクトを生成する命令。() の中身は「,」（カンマ）で区切られていて、一つ目は作るオブジェクト、二つ目はオブジェ

クトを作る位置、三つ目は作るオブジェクトの向きです。
Quarternion.Euler(0,0,0)については今はいいです。角度がついていないことを表します。

⑥ このスクリプトを動かす準備をします。GameObject を選択し、ヒエラルキーから Item にプレハブをドラッグアンドドロップします。また、ItemPosition の X の値を 2 にします。



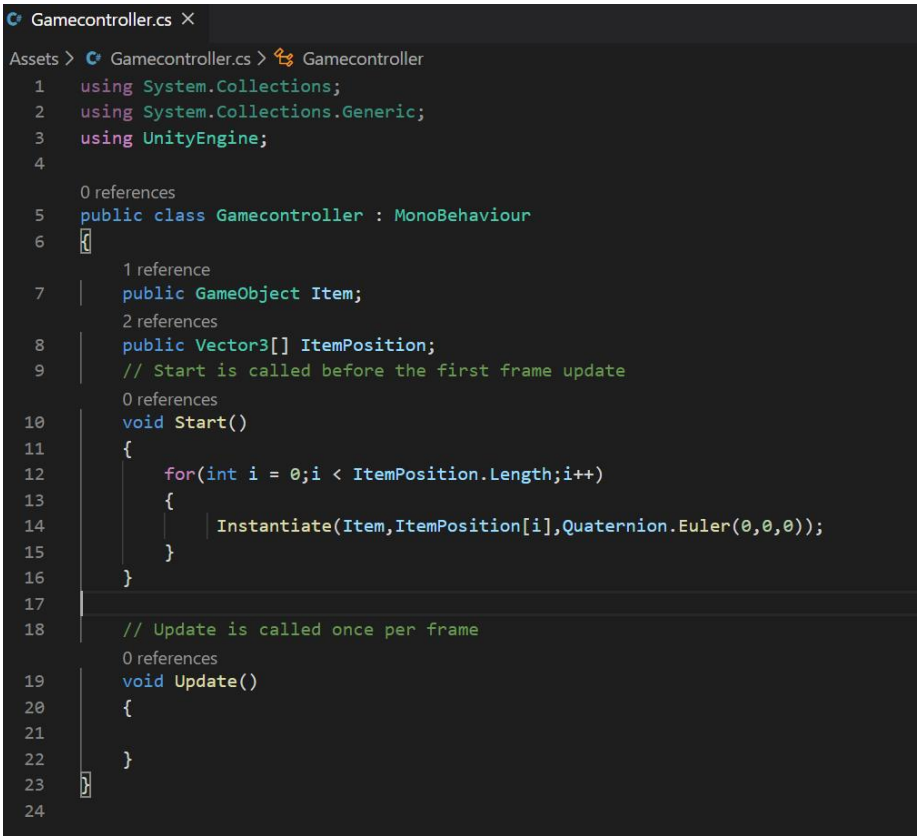
⑦ 実行すると、Item が作られたのが分かります。当たるとちゃんと消滅します。



１７．　配列と for 文

作る Item の数を増やします。

① 以下のように GameController スクリプトを編集してください。



② ここで新しいことを教えます。

一つ目は「配列」です。配列とは、同じ型の変数が連なったものです。

宣言の際は int[] A;のように、型の後ろに[]をつけます。

このように宣言された配列は、A[0],A[1]のように「変数名[数]」の形で一つの変数として扱うことができます。数は 0 から順番に割り当てられます。例えば

int[] Score;

Score[0] = 50;

Score[1] = 70;

float Average = (Score[0] + Score[1]) / 2;

とすると、変数 Average には 60 が入ります。

また、配列は「いくつの変数を連ねるか」という値を持ちます。これを配列の長さといいます。配列は長さを指定しないと使えません（そのため上の Score の例は、厳密にいうと正しくありません)。今回は public にしたので、Unity エディター上から長さや数値を指定します。長さは「配列名.Length」で取得できます。

二つ目は for 文です。この文は

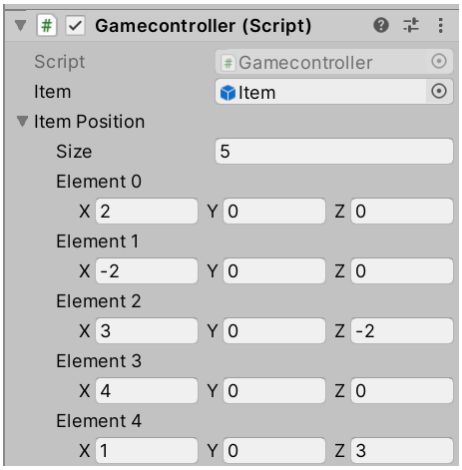
for(int i = 0;i < ～;i++){

　中身

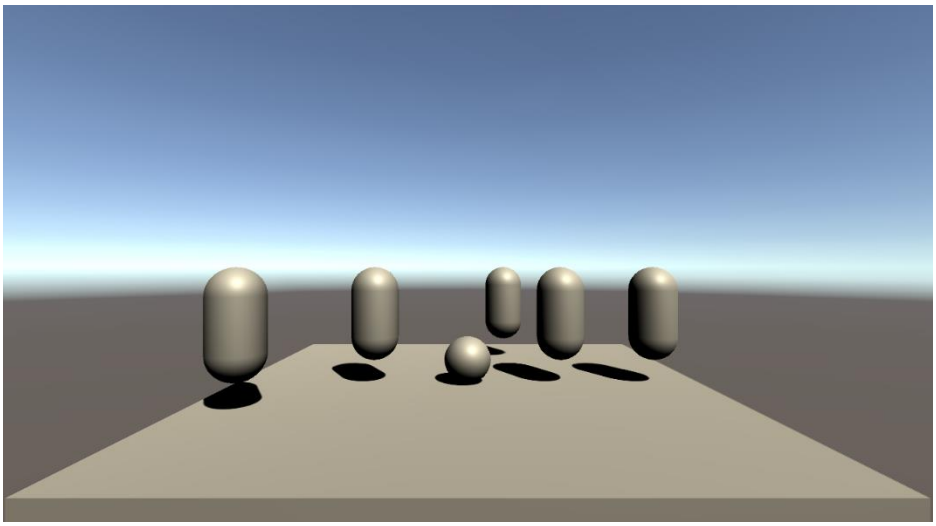
}
の形で使われ、中身の処理を～回繰り返す、という意味です。
具体的にいうと
変数 i を宣言し（int i = 0;）、もし i の値が～未満なら（i < ~;）（～は数値）、中身の処理を実行し、i に 1 を足し（i++）
この時点でもし i の値が～未満なら中身の処理を実行し i に 1 を足し……
という意味です。
また、{}内では変数 i を使うことができます。i は一回目の処理では 0、二回目の処理では 1、三回目の処理では 3、と、 1 ずつ増えていきます。

以上を踏まえると、このプログラムの意味は、ItemPosition の長さ回、ItemPosition[0],ItemPosition[1],ItemPosition[2]……の位置に Item を配置する、という意味になります。

③ このままでは動かないので、Unity から編集をかけます。GameObject の Gamecontroller の ItemPosition の左の三角形を押して、size を 5 に変更してください。これが配列の長さです。すると Element0 とかが出てくると思うので、以下のように入力してください。



④ 実行すると Item が 5 つそれぞれ指定した位置にできます。



１８．　乱数

ステージが毎回変わるようにします。

① GameController を以下のように加筆してください。

```
void Start()
{
    for(int i = 0;i < ItemPosition.Length;i++)
    {
        ItemPosition[i] = new Vector3(Random.Range(-5,5),0,Random.Range(-5,5));
        Instantiate(Item,ItemPosition[i],Quaternion.Euler(0,0,0));
    }
}
```

new：Vector3 型の変数に値を代入するには、position = new Vector3(x,y,z);のように、new を付けなければいけません。

Random.Range(-5,5)：－５から５までの値をランダムに出力してくれます。

② 実行すると、そのたびごとに Item の位置が変わるようになりました。

１９．　UI

ゲームらしくするため、アイテムがあと何個あるかを表示するようにしましょう。

① GameController を以下のように加筆してください。

```
void Update()
{

}

2 references
public int Count;
0 references
void OnGUI()
{
    Count = GameObject.FindGameObjectsWithTag("Item").Length;
    GUI.TextField(new Rect(10,10,300,80),"残り" + Count.ToString() + "個");
}
```

void OnGUI(){}：内部で GUI に関する命令を扱える、という意味。GUI とは UI の一種で、まあ文字やボタンと考えてください。詳しいことは調べてください。

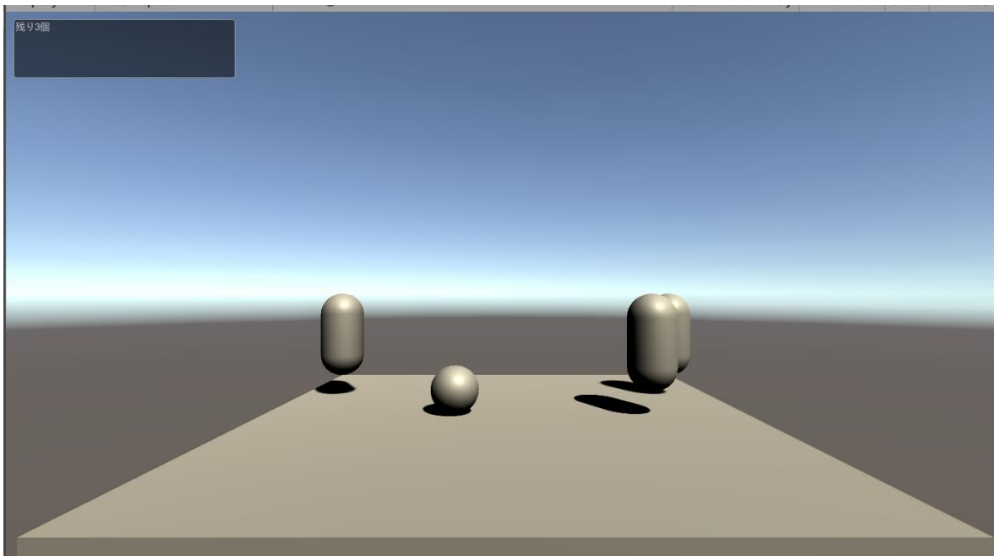
GameObject.FindGameObjectsWithTag(“Item”)：Item というタグが付いているオブジェクトを配列で取得する命令。この文は Item というタグのついたオブジェクトがすべて入った配列として扱えます。.Length は配列の長さを表すので、変数 Count には Item というタグがついたオブジェクトの数が入ります。

GUI.TextField(new Rect(x,y,width,height),”text”);：画面に文字を表示する命令。Rect は長方形という意味で、点(x,y)を起点に横 width、縦 height の長方形範囲を表します。その範囲に、text 部分に書かれたことが書かれます。今回はテキストが+でつながれていますが、これでも ok です。

Count.ToString();：int 型変数 Count を string 型に変換する、という意味。変数 Count は、このままでは文字として扱えないので、こんなことをする必要があります。

以上より、この文は「画面左上に Item の残っている数を「残り～個」と表示する」という意味です。

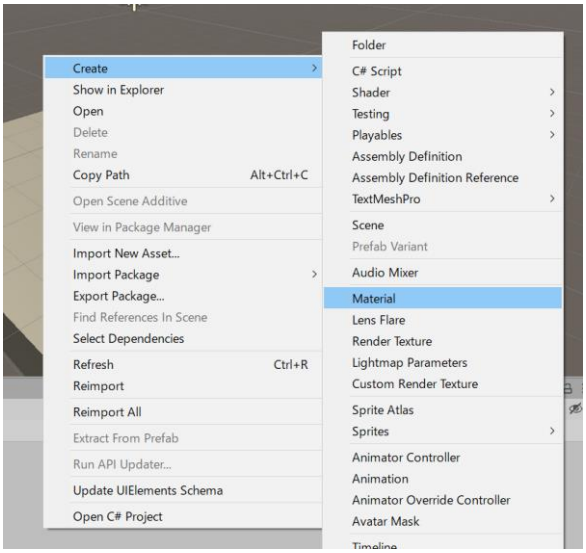
② 実行すると、左上に Item の残った数が表示されます。Item をとると数が 1 減ります。



２０．　マテリアル

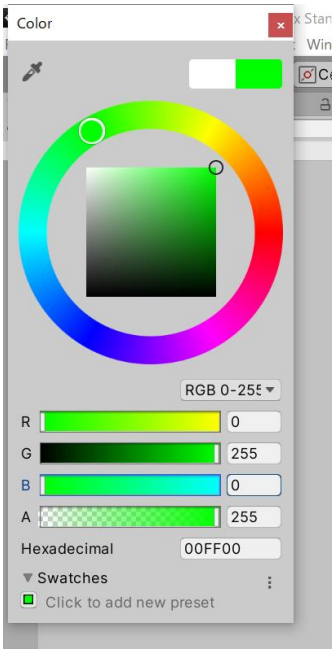
見た目がぱっとしないので、オブジェクトの色を変更します。

① プロジェクトビュー上で右クリックし、Create>Material を選択します。

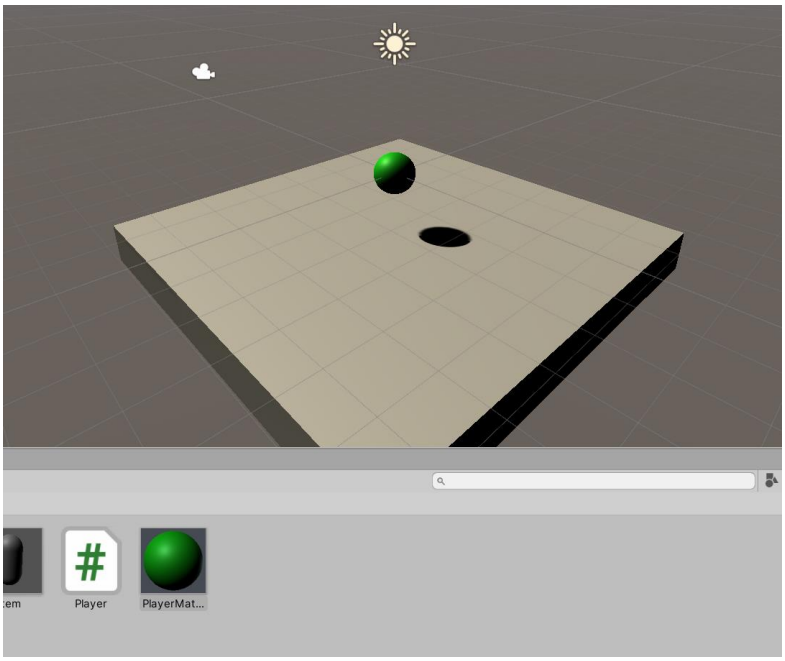


作成時に名前の入力ができるようになるので PlayerMaterial と名付けましょう。Material はオブジェクトの描画にかかわるコンポーネントで、色や質感を変更できます。

② インスペクターの Main Maps から Albedo の右の四角を選択します。すると色を選べる画面が現れるので、好きな色に変更してみましょう。

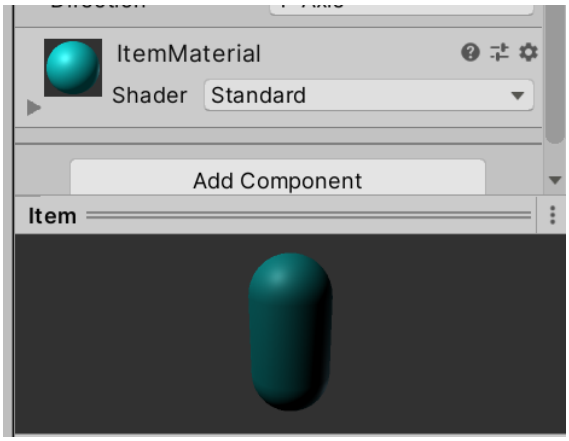


③ プロジェクトビューの PlayerMaterial をシーンの Player にドラッグアンドドロップします。すると Player の色が変わります。これは Player に PlayerMaterial が適応されたからです。



④ 同様にして ItemMaterial を作成し、色を変更しましょう。

⑤ Item のプレハブを選択し、インスペクターの一番下の Add Component の横に ItemMaterial をドラッグアンドドロップしましょう。すると DefaultMaterial が ItemMaterial に置き換わります。



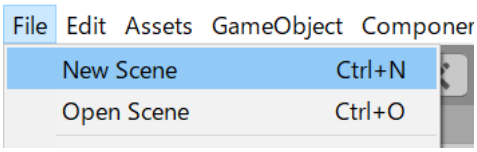
⑥ お好みで足場の色も変えましょう。

２１． シーン作成

ゲームの始まりと終わりを作りましょう。最初に言ったっきりだったシーン機能を使います。

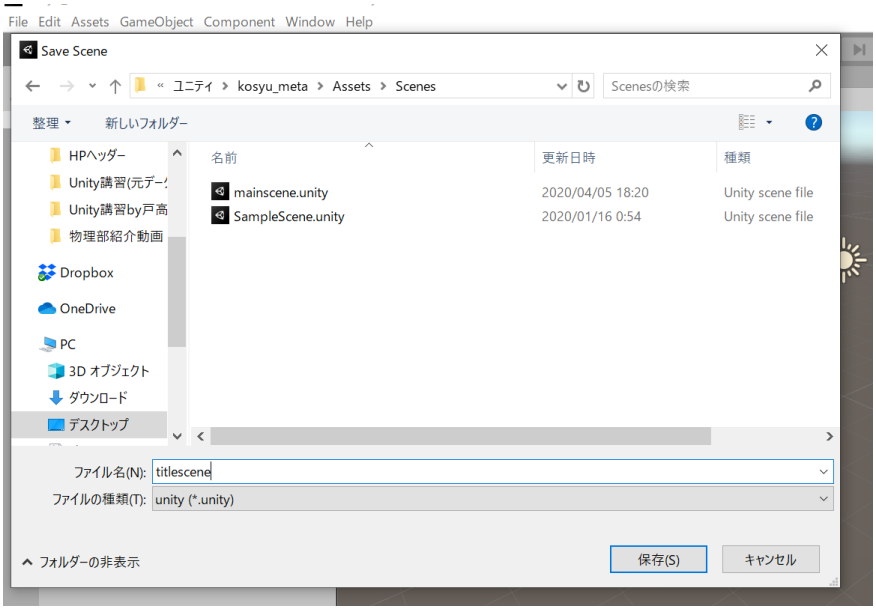
① 今のシーンを保存してください。

② 画面左上の File>New Scene を押してください。

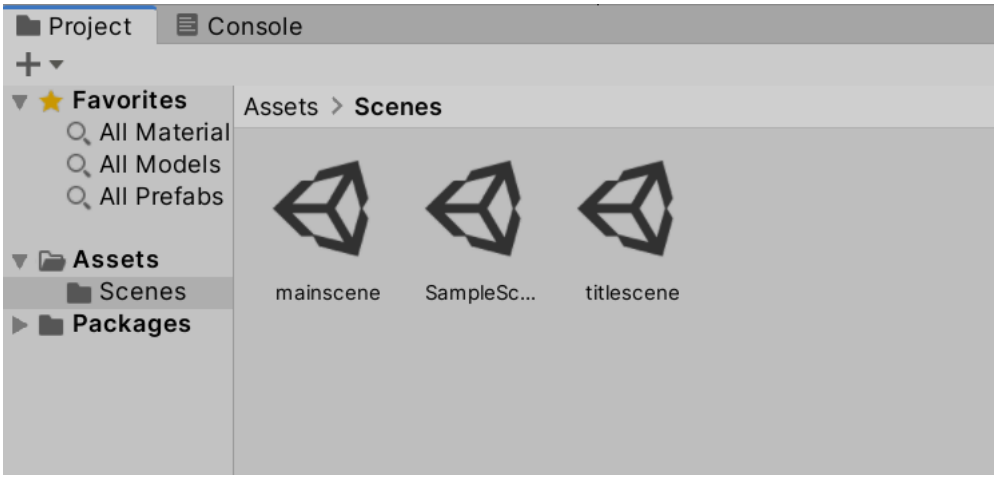


するとシーンからオブジェクトがなくなりました。これはデータが消えたわけではなく、新しいシーンになったからです。

③ File>Save As をクリックし、保存先に Scenes を選択します。ファイル名を titlescene にして保存ボタンを押します。

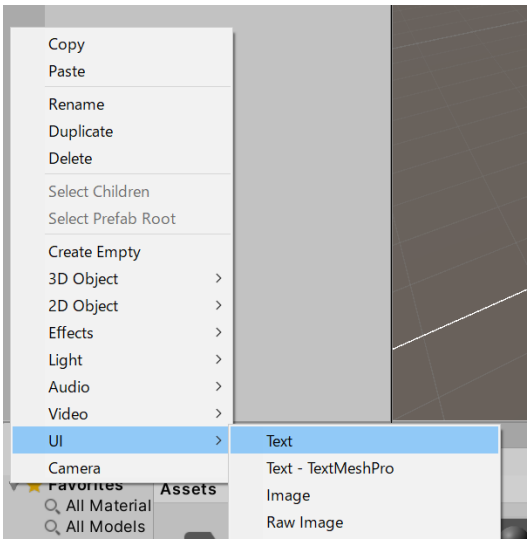


④ プロジェクトビューの Scenes を選択すると mainscene,titlescene,SampleScene の三つがあるのがわかります。mainscene を選択すると、先ほどまで編集していたシーンへ戻ることができます。

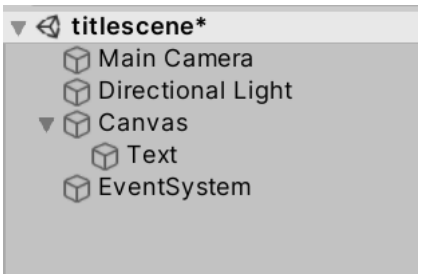


⑤ このゲームのタイトルを表示させます。OnGUI でもいいのですが、別の方法を使

います。
ヒエラルキーで右クリックして、UI>Text を選択します。

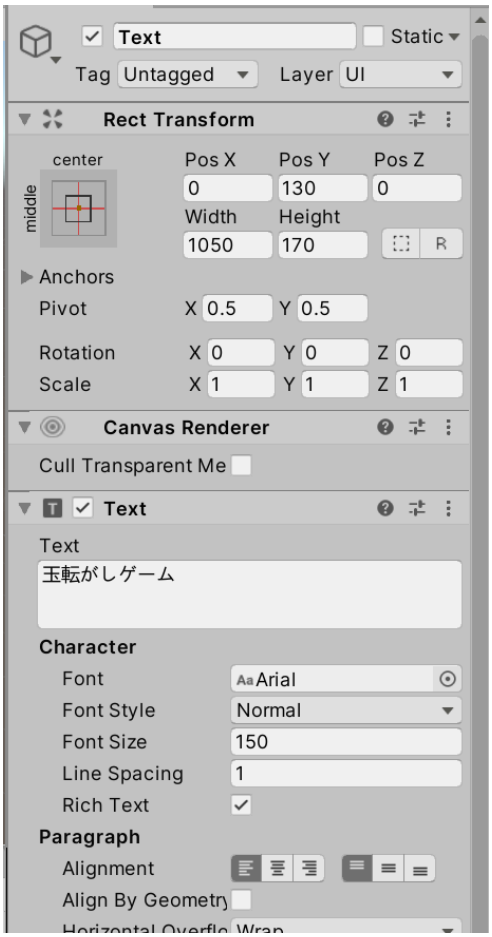


すると、Canvas、Text、EventSystem が作られます。

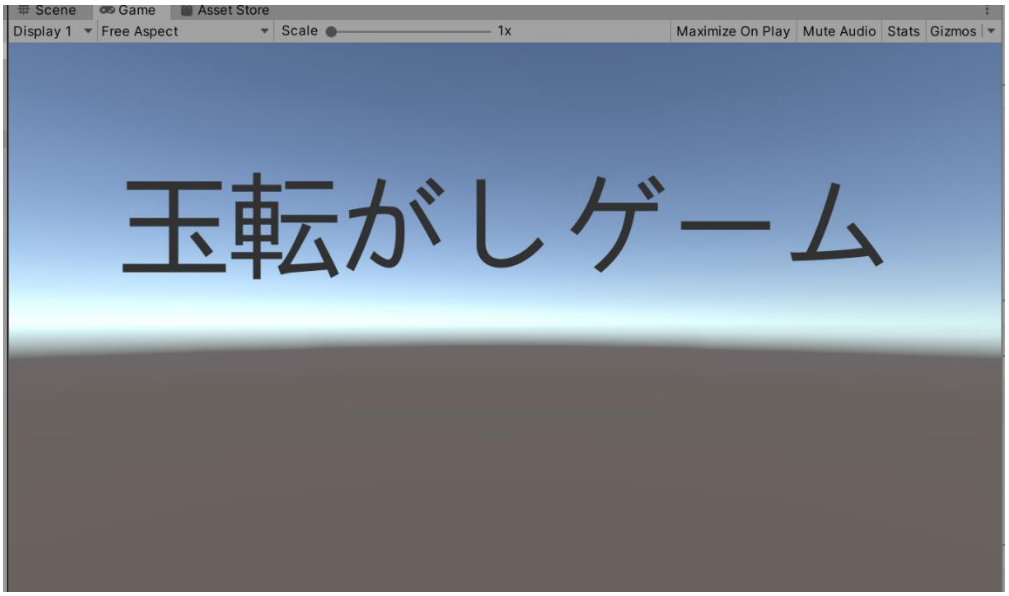


Text が文字で、Canvas はそれを映し出すスクリーンだと思ってください。また、Text は Canvas の子オブジェクトです。子オブジェクトは、親であるオブジェクトから様々な影響を受けるようになります（座標など）。詳しいことは後でやります。
EventSystem はボタンなどの入力を感知するために必要なオブジェクトです。いじる必要はありません。

⑥ Text のコンポーネントの Rect Transform,Text,Font Size を以下のように変えてください。PosX,Y,Z は中心の座標、Width,Height は大きさ、Text は書かれる内容、Font Size は字の大きさを表します。



⑦ 実行すると「玉転がしゲーム」と書かれているのが分かります。

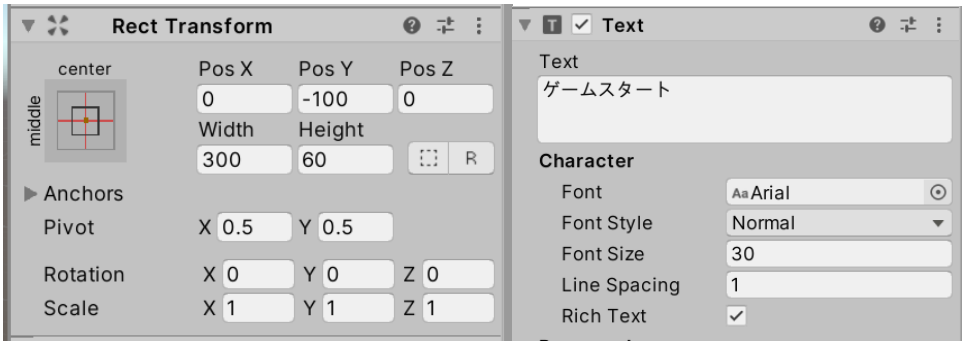


２２．　シーン遷移

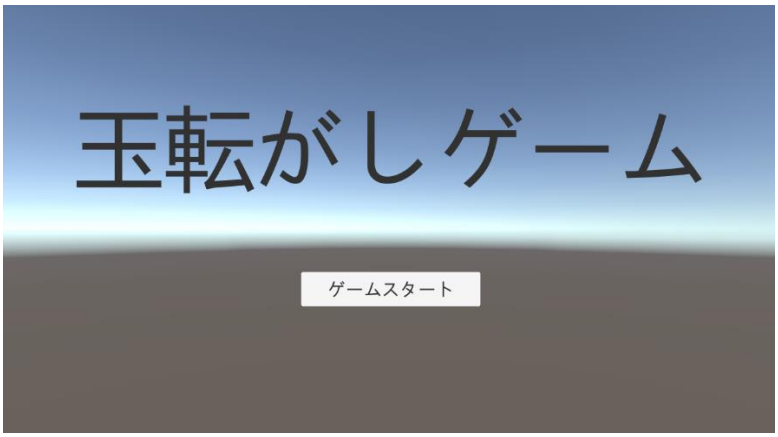
実行中にシーンを変更できるようにします。

① UI>Button からボタンオブジェクトを作ります。

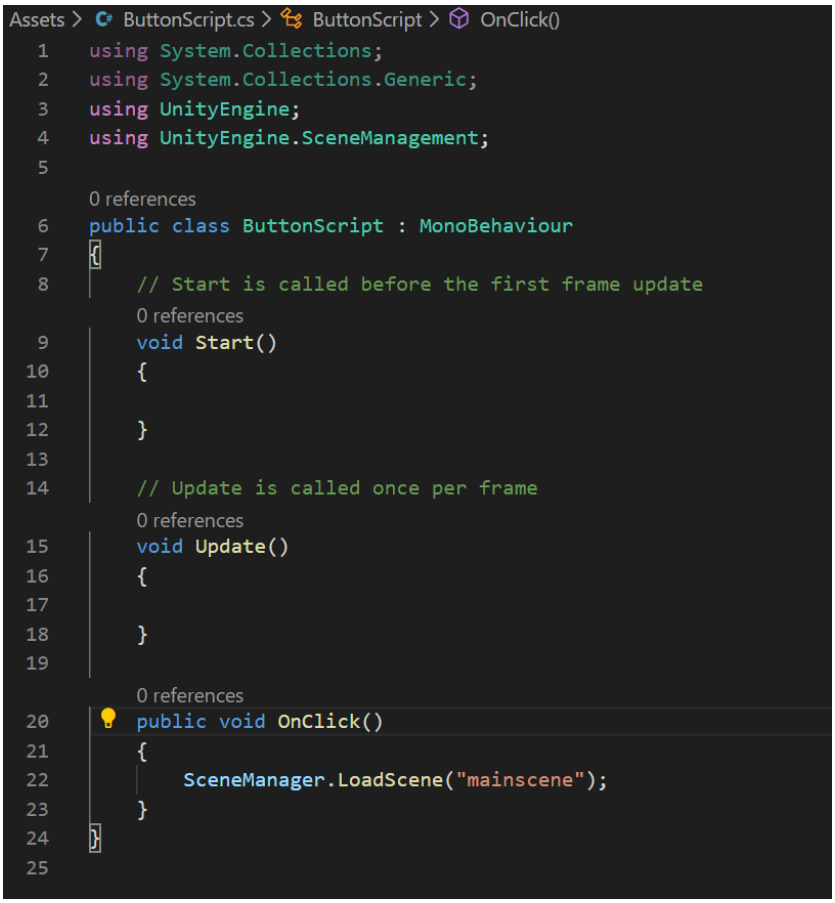
② Button の Rect Transform コンポーネントと子オブジェクトの Text の Text を編集します。



③ 実行するとボタンが表示されているのが分かります。しかし、押しても反応しません。



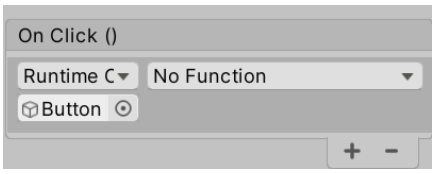
④ 押すと mainscene に移動するようにします。Button に Add Component>New Script から ButtonScript という名前でスクリプトを追加し、以下のように編集します。４行目にも追加されている命令があるので注意してください。また、OnClick には public がついていることにも注意してください。



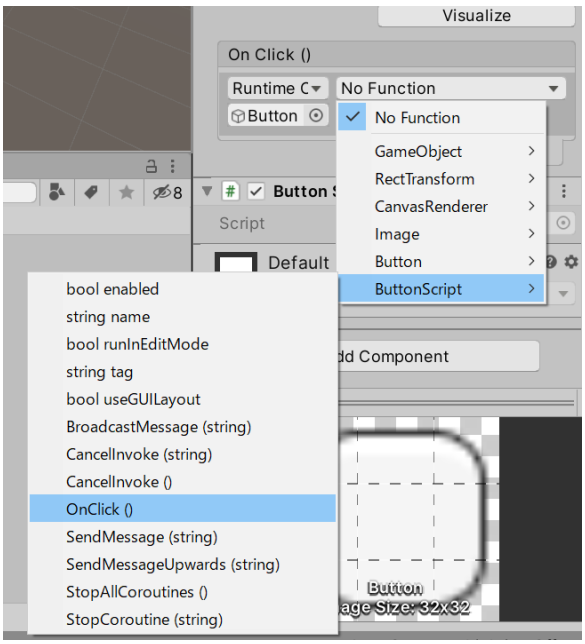
４行目：シーンに関する命令を書くときに必要な記述です。
OnClick（）のようなものを「メソッド」といいます。Update や OnTriggerEnter２D などもメソッドの一種ですが、これらは特別な意味をもち、特定の時に呼び出されます。Onclick は特別な意味を持たないので、このままだと呼び出される事無く終わってしまいますが、あとからボタンが押されたときに呼び出されるようにします。くわしいことはいつかやります。
SceneManager.LoadScene(シーン名);：カッコ内のシーンに移動できます。

⑤ Button の Button コンポーネントの一番下にある On Click の+マークをクリックします。

⑥ None となっているところにヒエラルキーの Button をドラッグアンドドロップします。



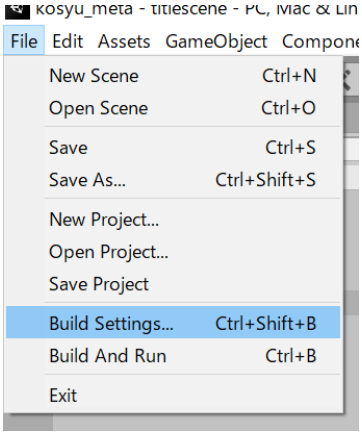
⑦ No Function となっているところを選択し、Button Script>OnClick()をクリックします。



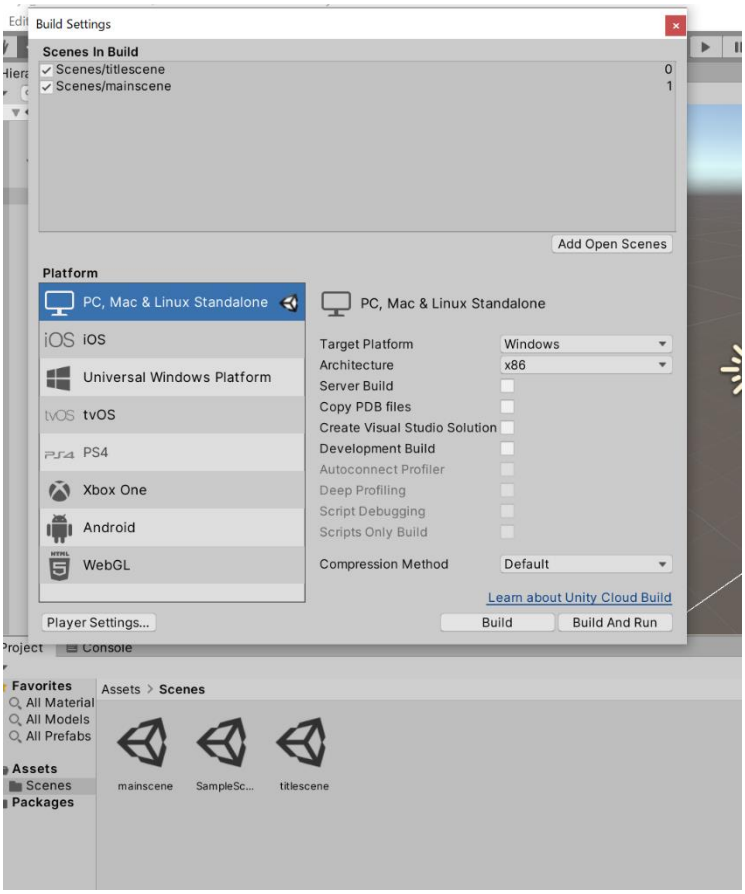
これで、ボタンが押されると ButtonScript の OnClick 内の処理が実行されるよう

になりました。

⑧ File>Build Settings を開きます。



⑨ Scenes in Build にプロジェクトビューから titlescene と mainscene をドラッグアンドドロップします。



これをしないとシーンを移動することができません。

⑩ 実行してボタンを押すと、シーンが変わりました。

２３． ゲームを終了させるシーンを作る

はじめのシーンを作ったので終わりのシーンも作りましょう。

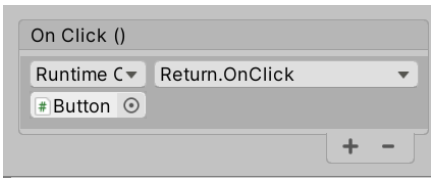
① 現在のシーンを保存し、File>New Scene でシーンを作り、Save As から endscene という名前で保存します。

② 以下のようになるように、Text と Button を作って配置してください。位置はだいたいで構いません。

③ もう一度、と書かれた方に Return という名前でスクリプトを追加して、以下のように編集します。

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

0 references
public class Return : MonoBehaviour
{
    // Start is called before the first frame update
    0 references
    void Start()
    {

    }

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

    }

    0 references
    public void OnClick()
    {
        SceneManager.LoadScene("titlescene");
    }
}
```

④ ボタンが押されるとこのスクリプトが実行されるようにします。



⑤ File>Build Settings からこのシーンを Scenes in Build に入れます。

⑥ 実行すると、もう一度ボタンが押されるとタイトルに移動するのが分かります。

２４．　ゲームの終了

終わるボタンを押すとゲームが終了するようにしましょう。

① 終わるボタンに Quit という名前のスクリプトを追加し、以下のように編集しましょう。

```
    }

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

    }

    0 references
    public void OnClick()
    {
        #if UNITY_EDITOR
            UnityEditor.EditorApplication.isPlaying = false;
        #else
            Application.Quit();
        #endif
    }
}

```

#if は if(){}とだいたい同じと考えてください。#if UNITY_EDITOR で、unity エディターで実行されているなら、という意味です。

UnityEditor.…… ＝false は、実行状態でなくす、という意味です。再生ボタンを押したときと同じです。

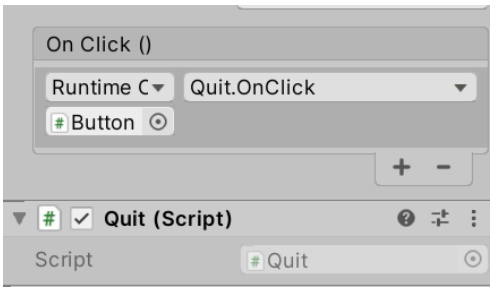
#else は、#if と同時に使われ、そうでないなら、という意味になります。この場合、unity エディター以外で実行されているなら、という意味です。

Application.Quit();はアプリを終了する、という意味です。ウィンドウ右上の×を

押したときと同じです。

以上より、このスクリプトは「Unity エディター上で実行されているなら実行状態でなくし、それ以外で実行されているならアプリを終了する」という意味になります。Unity エディター以外で実行することがあるのかよ、と思うかもしれませんが、ゲームが完成したときに実行するのは PC やスマホや switch です

② ボタンが押されたときにこのスクリプトが実行されるようにします。



③ 実行して終わるボタンを押すと、実行状態が解除されます。

２５．　シーン遷移

メインシーンでアイテムをすべて取ったらエンドシーンに移動するようにします。

① 現在のシーンを保存して、mainscene に移動してください。

② GameController スクリプトを以下のように編集します。

４行目の追加と、Count = 〜の位置の変更を忘れないで下さい。

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

0 references
public class Gamecontroller : MonoBehaviour
{
    1 reference
    public GameObject Item;
    3 references
    public Vector3[] ItemPosition;
    // Start is called before the first frame update
    0 references
    void Start()
    {
        for(int i = 0;i < ItemPosition.Length;i++)
        {
            ItemPosition[i] = new Vector3(Random.Range(-5,5),0,Random.Range(-5,5));
            Instantiate(Item,ItemPosition[i],Quaternion.Euler(0,0,0));
        }
    }

    // Update is called once per frame
    0 references
    void Update()
    {
        Count = GameObject.FindGameObjectsWithTag("Item").Length;
        if(Count == 0)
        {
            SceneManager.LoadScene("endscene");
        }
    }

    3 references
    public int Count;
    0 references
    void OnGUI()
    {
        GUI.TextField(new Rect(10,10,300,80),"残り" + Count.ToString() + "個");
    }
}

```

24 行目：「＝」は代入を意味しましたが、「＝＝」は左辺と右辺が同じかを判断する条件式となります。if(Count ==0)で「Count が 0 なら」の意味です

③ 実行すると、アイテムをすべて取ったときに endscene に移動します。

２６．　クリアできないことをなくす

このままだと、Player が足場から落ちるとクリアできないままになってしまいます。そのため、足場から落ちないようにします。

① Player スクリプトを以下のように編集します。

```
    if (Input.GetKey(KeyCode.DownArrow))
    {
        GetComponent<Rigidbody>().AddForce(0f, 0f, -Speed);
    }

    if(transform.position.x >5)
    {
        transform.position = new Vector3(5,transform.position.y,transform.position.z);
    }
    if(transform.position.x <-5)
    {
        transform.position = new Vector3(-5,transform.position.y,transform.position.z);
    }
    if(transform.position.z >5)
    {
        transform.position = new Vector3(transform.position.x,transform.position.y,5);
    }
    if(transform.position.z <-5)
    {
        transform.position = new Vector3(transform.position.x,transform.position.y,-5);
    }

}
```

transform.position：Transform コンポーネントの position の情報。

transform.position.x：Transform コンポーネントの position の情報のうち X の数値。

transform.position.x > 5：>は数学的な意味どおり、左辺のほうが大きいことを判断する条件式です。if の中で「X 座標が 5 より大きいなら」という意味になります。

以上から、X 座標が 5 以上なら X 座標を 5 にする、というような命令です。

なお、transform.position.x = 5;とは書けません。

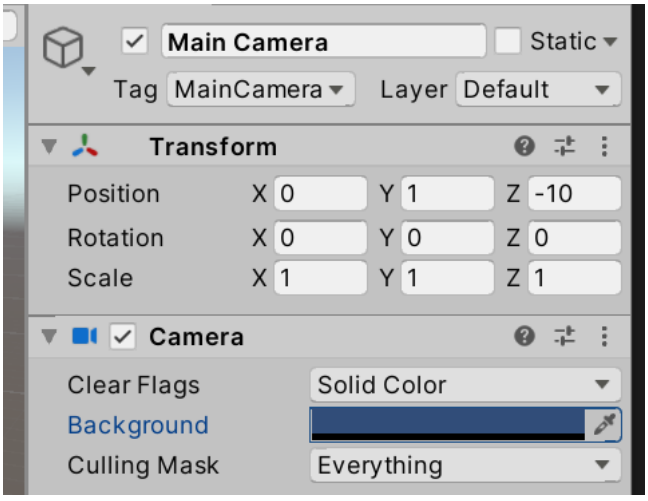
２７．　背景の変更

背景を変更します。

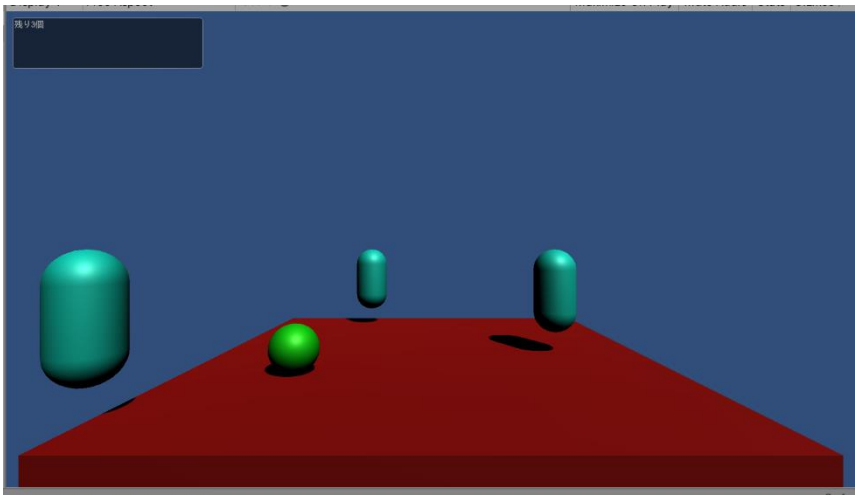
① Main Camera を選択します。

② Clear Frags を Sky Box から Solid Color に変更します。

③ Background で好きな色に変更します。



④ 実行すると、背景が変わります。

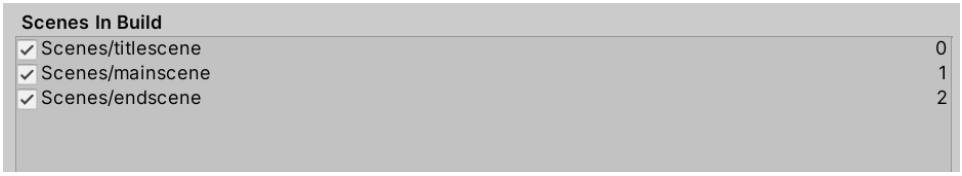


⑤ ほかの二つのシーンの背景も変更しましょう。

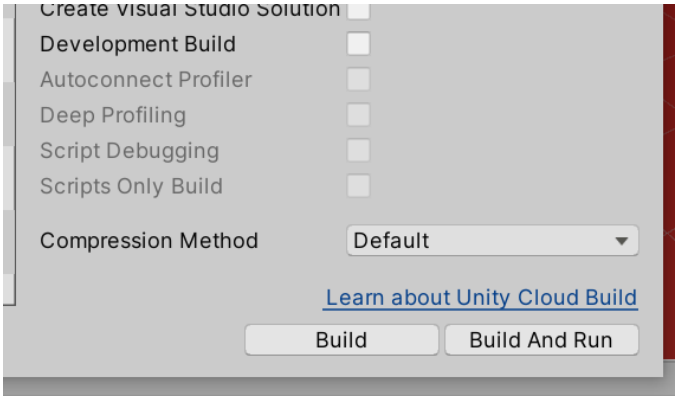
２８． ビルド

ゲームをアプリケーションにします。

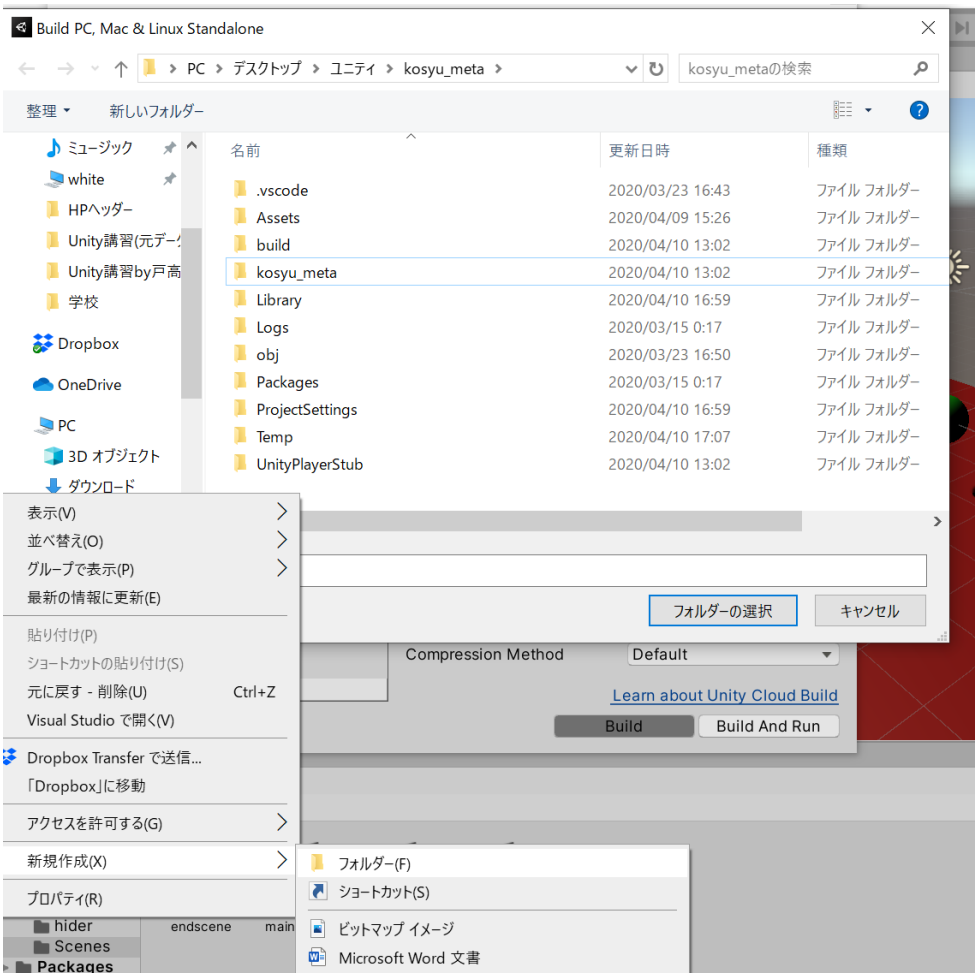
① File>Build Setting から順番が以下のようになっていることを確認します。なっていなければドラッグで順番を変更してください。



② Build を選択します。



③ 実行ファイルを作る位置を聞かれるので、右クリック>新規作成>フォルダーで EXE という名前のフォルダを作り、それを選択します。



④ しばらく待ちます。

⑤ ファイルが完成するので、その中の 「名前.exe」というものを実行します。

⑥ 先ほどまで作っていたゲームが遊べます。

⑦ これにてゲームの完成です！

これでゲームの完成です。お疲れさまでした。次回からは２Ｄゲームの作り方をやっていきます。