Unity講座2

~ブロック崩し編~

はじめに

　今回は、Unityを使ってブロック崩しを作りながら、Unityの基本的な操作を学びます。

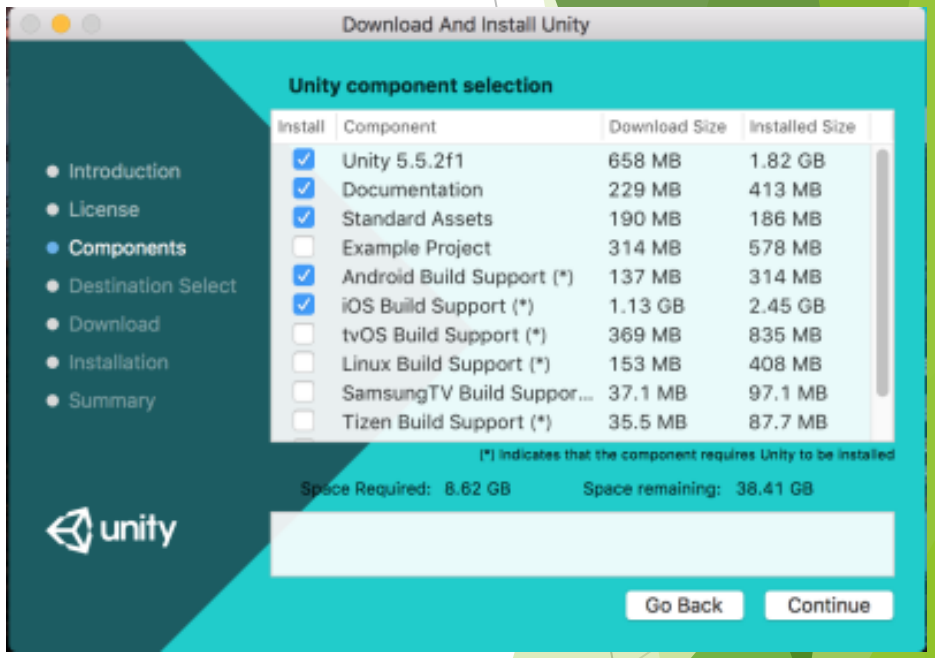
Unityの長所

- マルチプラットフォーム対応
  Android, iPhone, PCなど、対応する機種ごとに作り直す必要がない
- プログラミング無しで動かせる
- 3Dモデルを使ったゲームが簡単に作れる
- 豊富な”アセット”
  他の人が公開している ゲームのパーツ を利用して、高度なことも簡単に

Unityをインストール

Unity公式サイト(https://unity3d.com/jp/)から、Unityをダウンロードします。有料版もありますが、起動時に表示されるUnityのロゴが消せる・クラウドビルドサービスが利用できるなどの違いしかないため、無料版で十分です。

ダウンロードしたものを起動して、右図の画面まで進めます。この画面で、インストールするものを選択できます。Unity本体以外にも、Standard Assetsと、開発する可能性のあるデバイス用のBuild Supportは必ずインストールしましょう。一般的に利用することが多いので、Windows, Mac, iOS, Androidあたりはインストールしておくといいと思います。

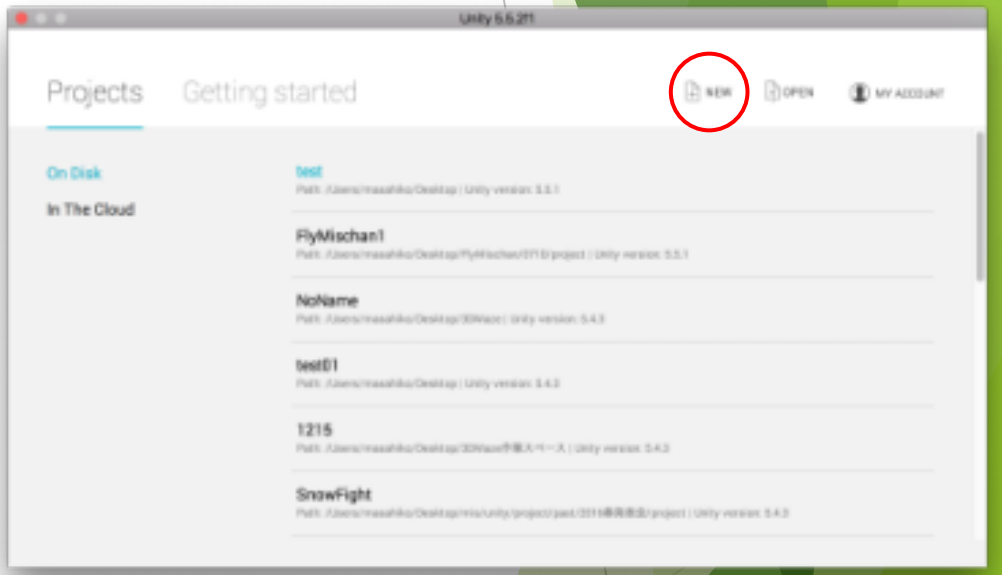


プロジェクトを作成

Unityを起動します。初回起動時にアカウント登録が必要なはずなので、必要事項を記入して登録しましょう。右図のような画面になったら、NEWボタンから、プロジェクト名と保存する場所を指定して、新しいプロジェクトを作成します。

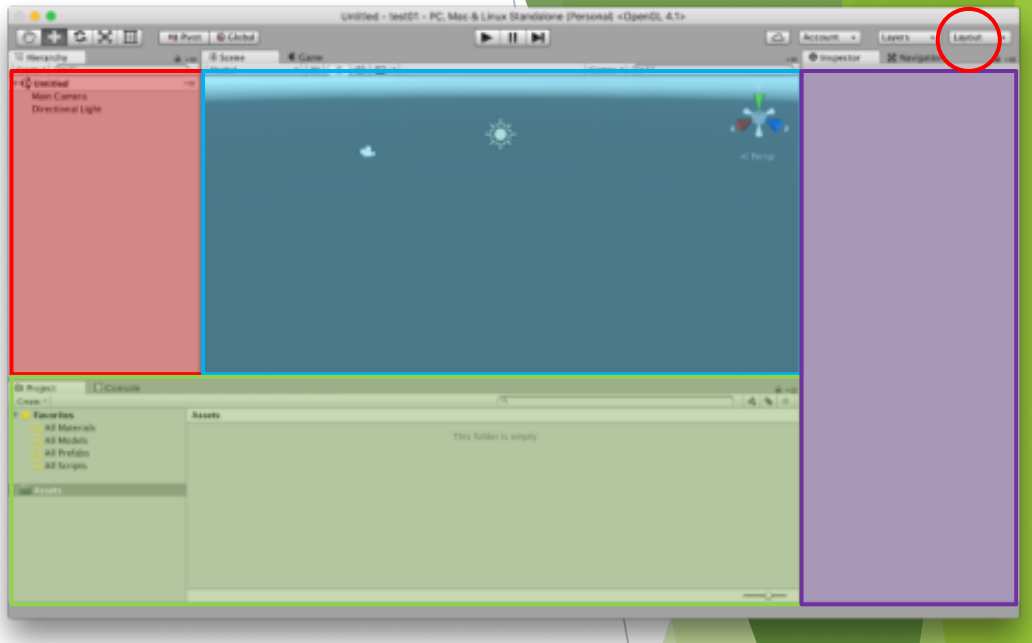
Unityでは、一つのゲームを”プロジェクト”と呼び、一つのプロジェクトは、複数の”シーン”によって構成されます。例えばRPGを作るなら、タイトル・町やダンジョン・戦闘画面などのシーンをそれぞれ作成し、それらを行き来することでゲームを構成します。

Unityの画面が開いたら、Ctrl+Sでシーンを保存しましょう。



画面の説明

- ヒエラルキーウィンドウ
  シーン内にあるオブジェクトの一覧が表示される
- シーンビュー
  ゲームの世界を自由に見て回れる画面。オブジェクトの移動や拡大縮小などの操作が行える。
- ゲームビュー
  ゲームのプレイ画面が表示される。シーンビューと同じ場所にあるが、オブジェクトの操作はできない。
- プロジェクトウィンドウ
  ゲーム全体の素材倉庫となるassetフォルダの中身が表示される。ここから素材を変更・追加することができる
- コンソールウィンドウ
  ゲーム実行時のエラーなどが表示される
- インスペクターウィンドウ
  ヒエラルキーやプロジェクトウィンドウなどで選択したものの詳細が表示される。ここから要素の追加や設定の変更の作業ができる。



起動すると上のような画面になると思いますが、もし構成が違っても、右上のLayoutボタンからDefaultを選択すると呼び出せます。

画面の説明

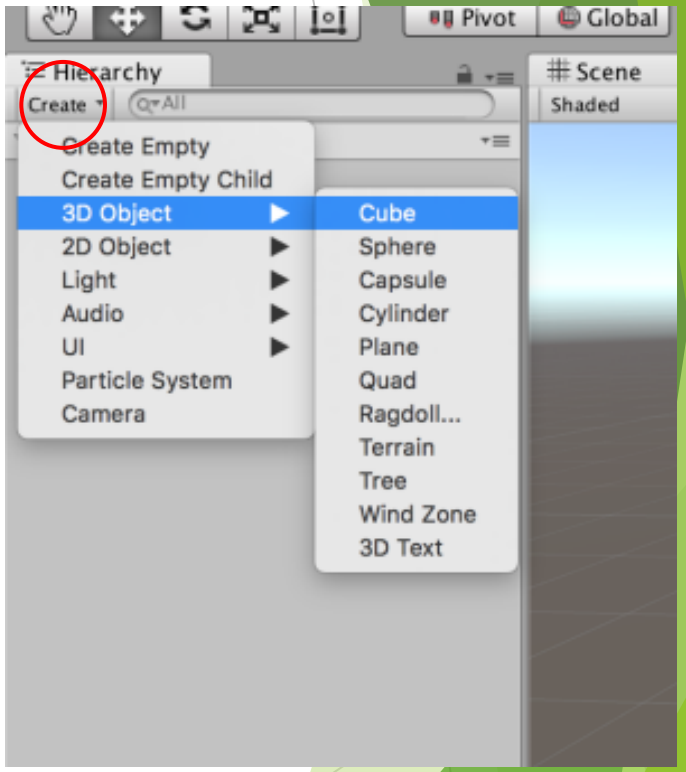
画面上方のこの部分は、ツールバーと呼ばれています。
一番左が操作モードの選択で、視点移動、オブジェクトの移動・回転・拡大などが選択できます。

中央の3つのボタンはゲームの再生・停止ができるボタンです。Unityでは、動作を確認するためにいちいちビルドして実行する必要はなく、この再生ボタンを押すだけで簡単に確認ができます。

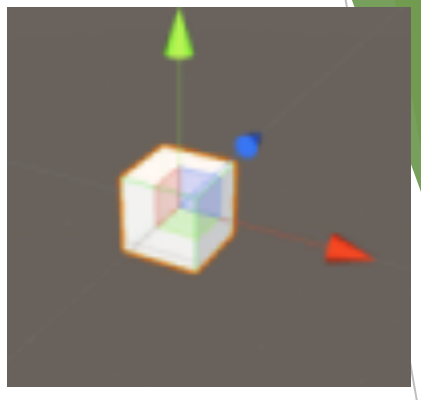
オブジェクトを置く

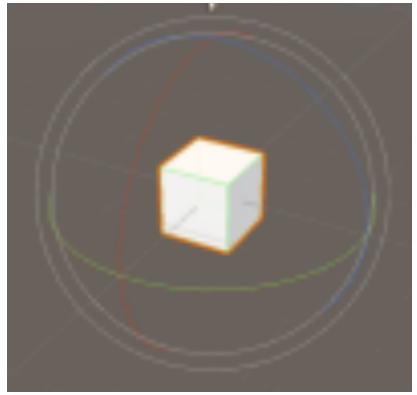
　新しいシーンには、カメラとライトだけが置いてあります(2Dで作成した時はカメラのみ)。このカメラによって映される映像が、プレイヤーが見るゲーム画面になります。

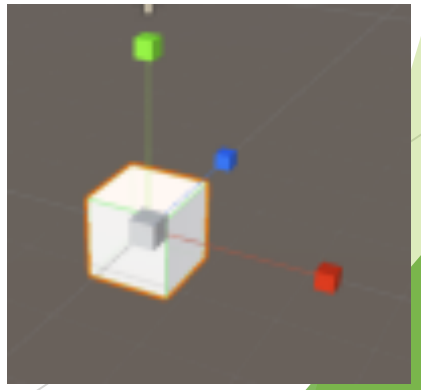
　ここに、新しいオブジェクトを置いてみましょう。右図のように、画面左側、ヒエラルキーウィンドウのCreateボタンから3DObject→Cubeと選択して、新しいCubeを作成します。



オブジェクトの操作







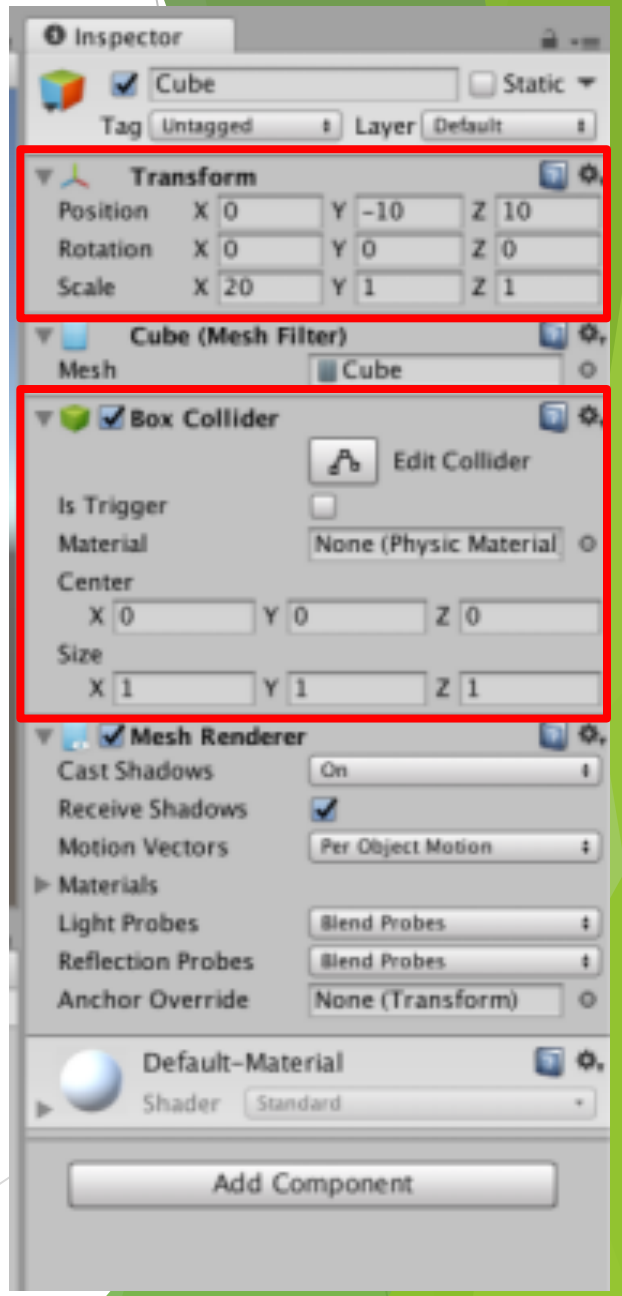
　ツールバーで操作モードを切り替えると、右図のように、オブジェクトの周辺に表示される軸(ギズモという)が変化します。この軸をドラッグすることで、移動・回転・拡大などの対応する操作が行えます。

　また、Altキーを押しながらドラッグすると視点が回転でき、Altを押しながらマウスの右ボタンでドラッグすると、視点のズームができます。

作るもの

- ブロック崩し
- キーボードの十字ボタンで操作
- ブロックを全て壊したらクリアー

壁を準備



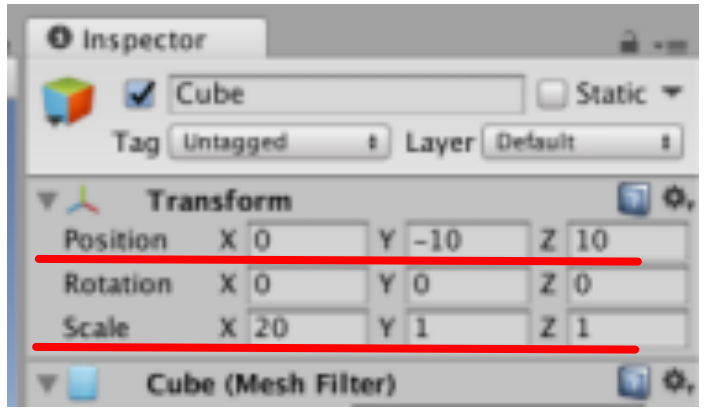
さっき作成したCubeを利用して枠を作ります。ヒエラルキーでCubeを選択すると、インスペクターにCubeの情報が表示されます。
　Unityでは、オブジェクトの持つ各要素をコンポーネントと呼び、コンポーネントを追加することで、そのオブジェクトができることを増やしてゲームを作成していきます。

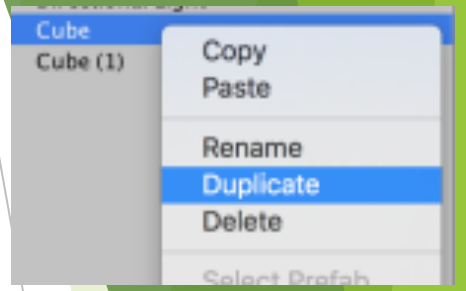
　Transformはこのオブジェクトの位置・角度・大きさを持つコンポーネントです。全てのオブジェクトがTransformを持っており、そのオブジェクトの本体のような役割も持っています。

　Colliderは当たり判定のコンポーネントです。Colliderをつけておくだけで、Collider同士の衝突を自動的に検知して反射してくれるのでとても便利です。

壁を準備

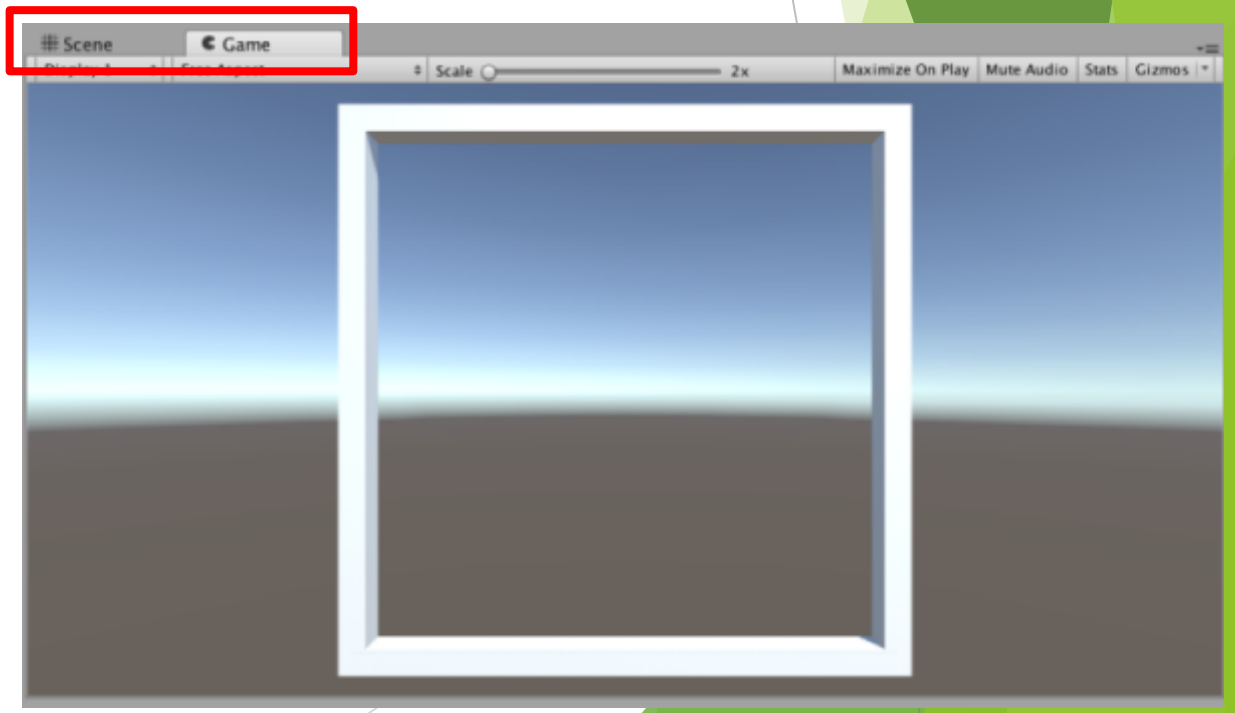
CubeのTransformを変更して、枠を形成します。Positionを(0,-10,10), Scaleを(20,1,1)として、下端にしました。ヒエラルキーでCubeを右クリックしてDuplicateを繰り返し、Cubeの複製を3つ作ります。Cubeの複製のTransformをそれぞれ次のように設定しました。(Rotationは全て0です)





	Cube(1)	Cube(2)	Cube(3)
Position	(0,10,10)	(-10,0,10)	(10,0,10)
Scale	(20,1,1)	(1,21,1)	(1,21,1)

Main CameraのPositionを(0,0,-10)にすると、ゲームビューでは右図のような画面になると思います。(タブをクリックするとシーンビューとゲームビューを切り替えられます。)

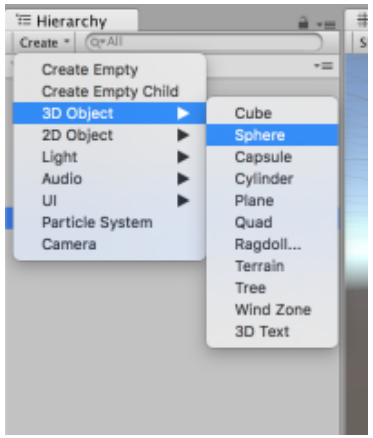


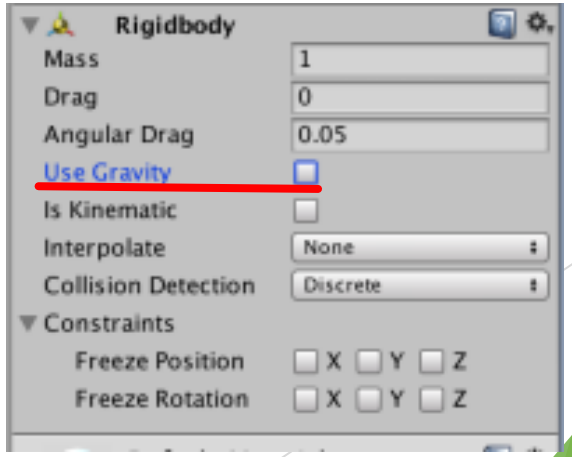
ボールの準備

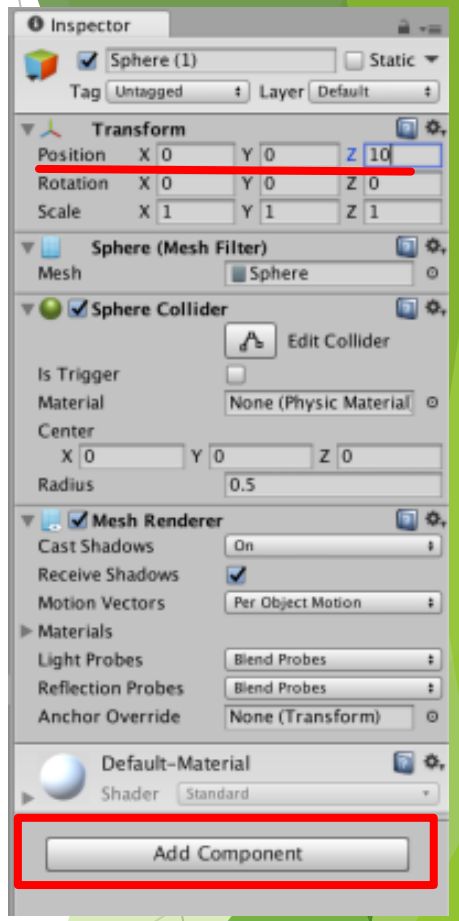
ヒエラルキーのCreateから3DObject→Sphereを選択し、球体を作成します。作成したSphereを選択した状態で、インスペクターの一番下にあるAdd Componentボタンから、Physics→Rigidbodyと選択して、Rigidbodyを追加します。

これは物理法則に関するコンポーネントで、持たせておくだけで物理法則に従うようになります。
今回は重力を無視するので、Rigidbodyの Use Gravityのチェックを外しておきます。

また、TransformのPositionの値も壁に合わせて(0,0,10)に設定しておきます。





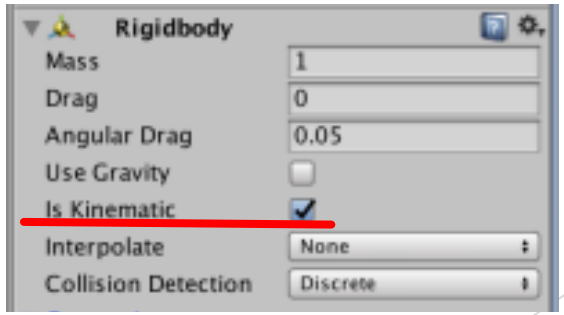


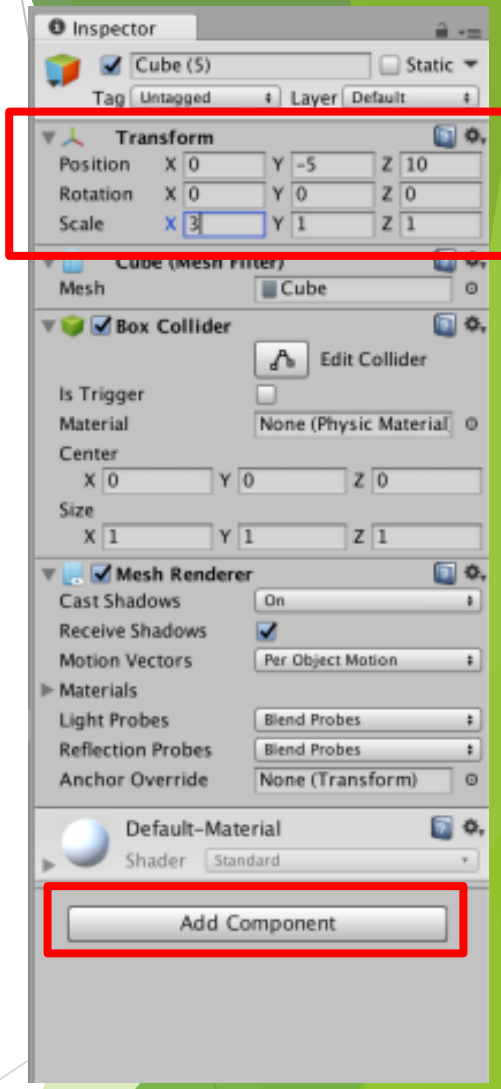
プレイヤー用のブロックの準備

ヒエラルキーのCreateボタンからCubeをもう一つ作成します。インスペクターで、Transformの Positionを(0,-5,10), Scaleを(3,1,1) にしました。

一番下のAdd Componentから、Physics→Rigidbodyを選択し、Rigidbodyを追加します。プレイヤー用のブロックも重力を無視するので、Use Gravityのチェックを外します。

次に、その下の is Kinematic にチェックを入れてください。これは物理法則を無視してその場にとどまるようになる項目です。このブロックは、プレイヤーの操作によって移動して、ボールとの衝突では移動しないようにする必要があります。衝突によって与えられる加速度を無視するためにis Kinematicをつけました。



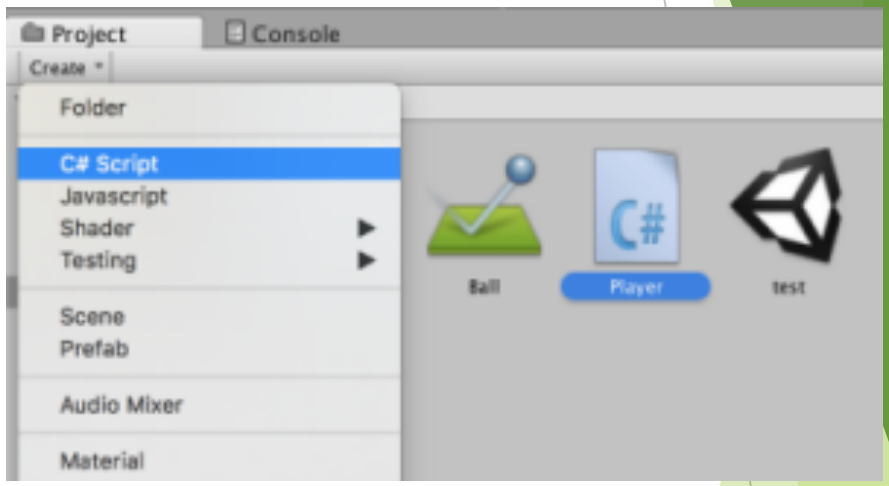


スクリプトを書く

　ここまでである程度の設定ができましたが、これだけではゲームは始まりません。ここから少しプログラミングをして、ボールに初速度を与え、プレイヤーの操作をゲームに反映させる必要があります。

　まずは、プロジェクトウィンドウのCreateからC#Scriptを選択し、スクリプトファイルを作成しましょう。

　作成したファイルを開きましょう。MacではUnityに付属しているMono Develop、Windowsでは自動的に連携されるVisual Studioで開かれると思います。



スクリプトを書く

ファイルを開くと、右のようなプログラムがすでに書かれていると思います。
上の方のusingの部分は、unityが用意している関数の中から、これから使用する関数の塊を宣言して、使えるようにしておくものです。
5行目の NewBehaviorScript の部分(クラス名)は、ファイル名と一致していないとエラーが出ます。ファイル名を変更したら、必ずここも変更しましょう。

8~10行目のStart関数は、シーン開始時に1度だけ実行されます。一方、13~15行目のUpdate関数は、毎フレーム繰り返し実行されます。
この2種類の関数の中に、オブジェクトに行わせたい処理を書いて、実行させることになります。

```
1    using System.Collections;
2    using System.Collections.Generic;
3    using UnityEngine;
4
5    public class NewBehaviourScript : MonoBehaviour {
6
7        // Use this for initialization
8        void Start () {
9
10       }
11
12       // Update is called once per frame
13       void Update () {
14
15       }
16   }
17
```

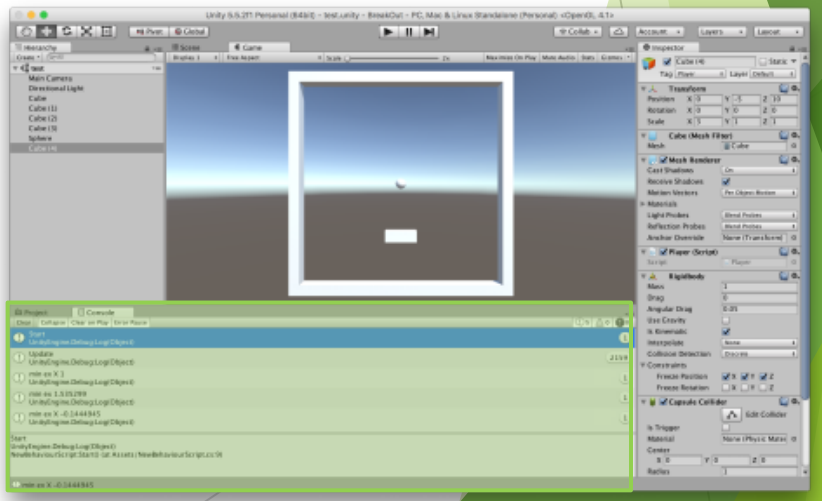
スクリプトを書く

練習として、右のようなスクリプトを書いてみました。一つ前からの変更点は赤線で示した2箇所です。先ほどのファイルを書き換えて保存しましょう。

Debug.Logという関数を使うと、unityのコンソールウィンドウに好きな文字列を出力することができます。(コンソールウィンドウは、プロジェクトウィンドウと同じスペースにあります。タブをクリックすると切り替えられます。)

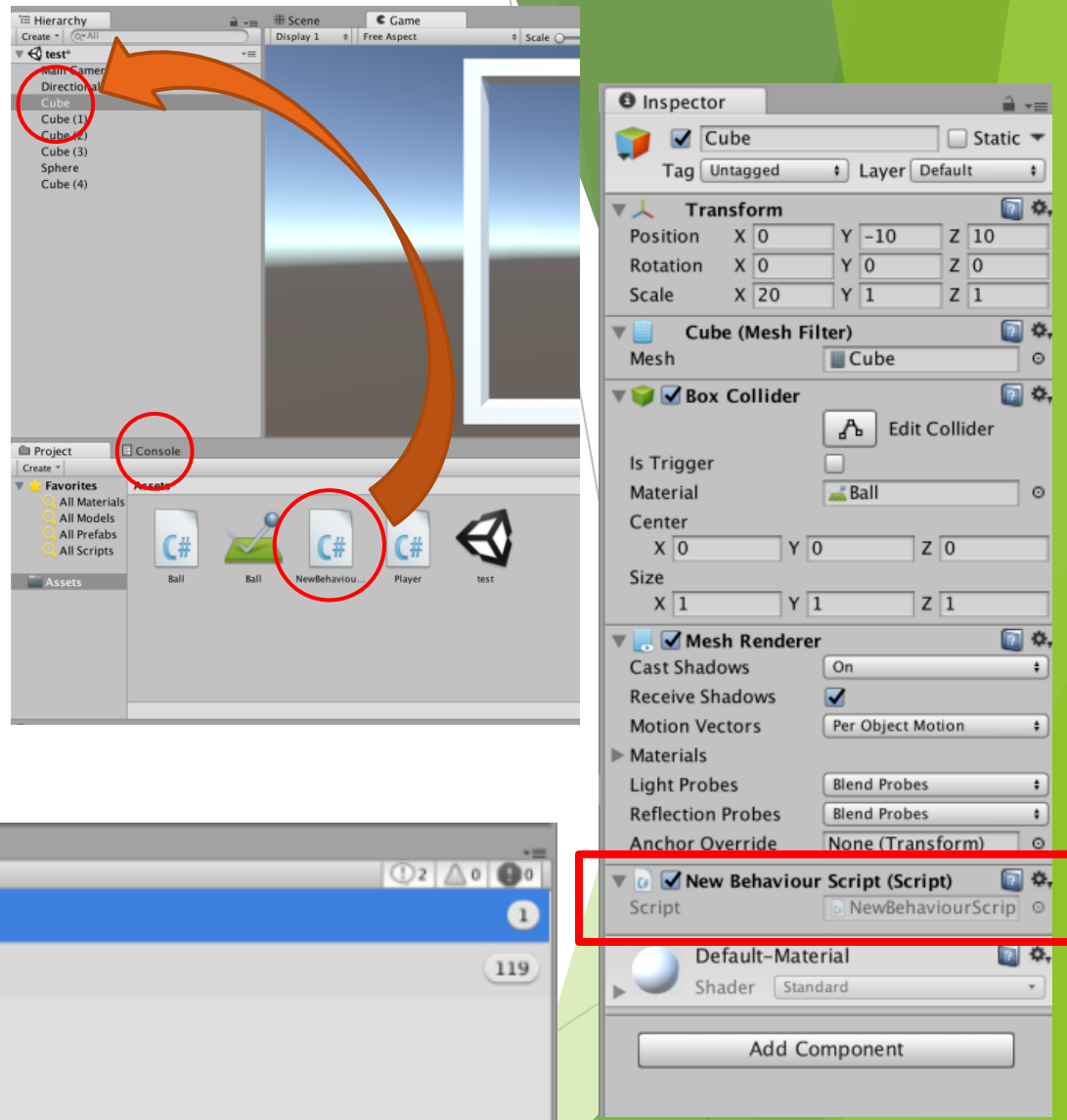
スクリプトをassetフォルダに保存しても、これは素材倉庫に入れただけのようなものなので、まだこのスクリプトは実行されません。

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class NewBehaviourScript : MonoBehaviour {
6
7       // Use this for initialization
8       void Start () {
9           Debug.Log ("Start");
10      }
11
12      // Update is called once per frame
13      void Update () {
14          Debug.Log ("Update");
15      }
16  }
17
```

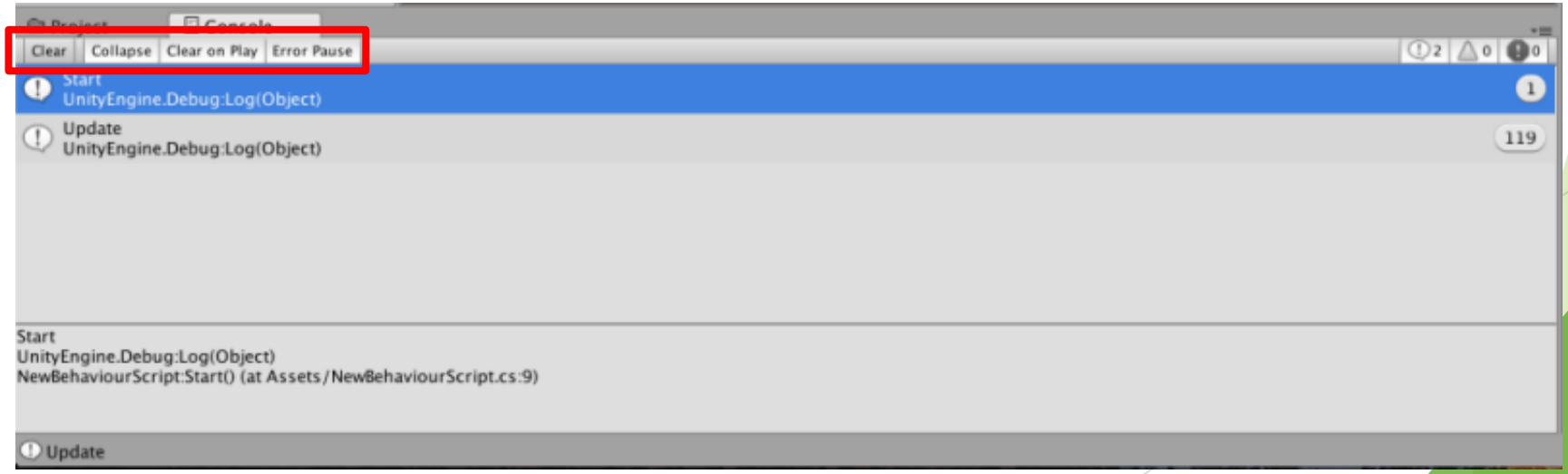


スクリプトを実行する

　スクリプトは、オブジェクトに持たせることで初めて実行されるようになります。
　プロジェクトウィンドウからヒエラルキーのCubeにドラッグ&ドロップして、スクリプトを持たせてみましょう。右図のようにInspecterにスクリプトが追加されます。



　コンソールウィンドウを表示して、画面上部の再生ボタンでゲームを実行してみましょう。下図のようにStartとUpdateが表示されます。コンソールを見るときは、CollapseとClear on Playをオンにしておくと見やすくなります。



スクリプト

　ここからボールとプレイヤーが操作するブロックのスクリプトを説明していきますが、C言語講座など、他の講座を受けてからの方がわかりやすいと思います。
　公式サイトのマニュアル
(https://docs.unity3d.com/jp/540/Manual/index.html)の検索機能などを使って、分からないものは逐一調べながら読んでください。

Ballのスクリプト

　スクリプトの実行方法が確認できたので、次はボールを動かすためのスクリプトを書きます。新しいC# Scriptファイルを作成し、Ballと名付けてください。
　右のようなスクリプトを書きます。まず、5行目のクラス名がファイル名と一致しているか必ず確認します。今回はBallです。

　ボールに初速度を与えて移動させるスクリプトを書くのですが、オブジェクトを移動させる方法は2通りあります。
　1つはRigidbodyのAddForceという関数を用いて力を与えて、あとは物理演算に任せて移動させる方法です。
　もう１つは、Trasnformの持つTranslateという関数で直接位置を動かす方法です。
　今回は物理法則に従って運動させたいので、Rigidbodyを用います。

```
1    using System.Collections;
2    using System.Collections.Generic;
3    using UnityEngine;
4
5    public class Ball : MonoBehaviour {
6
7        Rigidbody rigid;
8
9        // Use this for initialization
10       void Start () {
11           rigid = gameObject.GetComponent<Rigidbody> ();
12           rigid.AddForce (1,4,0,ForceMode.VelocityChange);
13       }
14
15       // Update is called once per frame
16       void Update () {
17
         }
     }
```

Ballのスクリプト

7行目は、Rigidbody型の変数rigidを宣言しています。11行目でこの変数にボールの持つRigidbodyコンポーネントを入れて、プログラムの中で扱えるようにします。

GetComponent関数は、オブジェクトの持つコンポーネントから、目当ての種類のものを探してきてくれる関数です。11行目でボールの持つRigidbodyをrigidに入れて、12行目で、オブジェクトに力を加えます。

AddForceの引数(かっこの中)は、(x,y,z,ForceMode)で、x,y,zの各方向に対して加える力と、力の加え方を指定しています。今回のようにForceModeをVelocityChangeにすると、オブジェクトの重さを無視して速度を直接変更します。

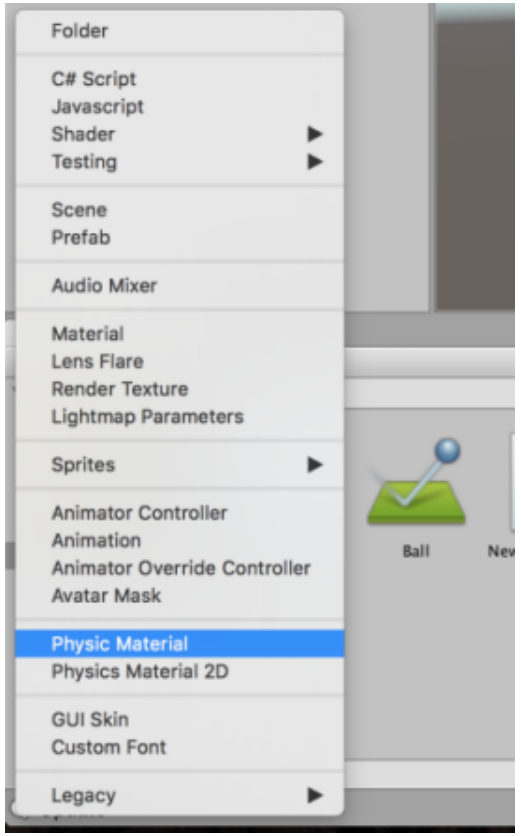
```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Ball : MonoBehaviour {
6
7       Rigidbody rigid;
8
9       // Use this for initialization
10      void Start () {
11          rigid = gameObject.GetComponent<Rigidbody> ();
12          rigid.AddForce (1,4,0,ForceMode.VelocityChange);
13      }
14
15      // Update is called once per frame
16      void Update () {
17
        }
    }
```

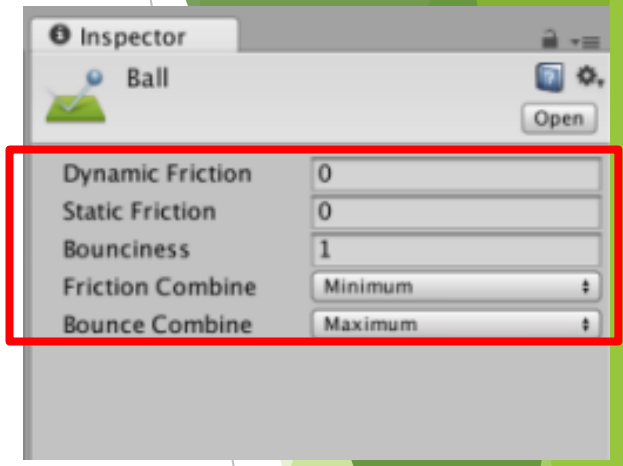
動作確認

　これでボールに初速が与えられるようになります。
先程と同様にボールにドラッグ&ドロップでスクリプトを持たせて、再生ボタンを押して実行してみましょう。
ボールが右斜め上に発射されます。

　しかし、上の壁に当たると反射されずに、吸い付いたような動きをしてしまうと思います。

ボールの反射の設定



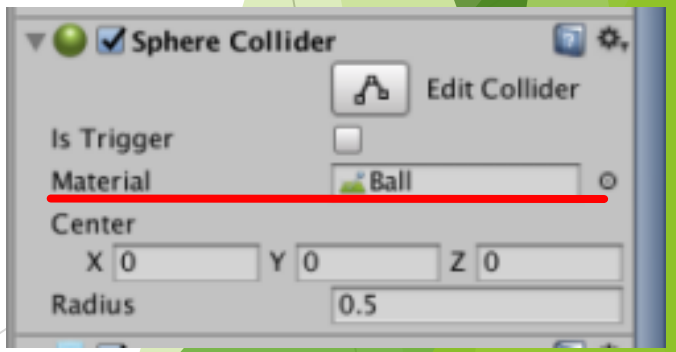


プロジェクトウィンドウのCreateから、Physic Materialを作成します。これは物体の摩擦と弾性を設定するものです。

力を失わずに反射させるために、Bouncinessを1にします。Bounce CombineはMaximumにして、衝突した2つの物体のBouncinessの大きい方を使って計算するようにします。ついでに摩擦が極力小さくなるように、Frictionに関する項目を右図のように設定しました。

Physic Materialは、オブジェクトのColliderに設定します。ボールのColliderの、Materialの項目に、今作ったものを設定してください。

これである程度角度がついた反射については解決します。(浅い角度で反射するときに勢いを殺されてしまう場合に関しては、少し後で対策します。)



Playerのスクリプト

ボールがきちんと反射するようになったので、今度はプレイヤーが操作するブロックのスクリプトを書きます。新たに作成したスクリプトに、右の赤線を引いた2行を追加します。

ボールの移動はRigidbodyで力を加えましたが、プレイヤーはキー入力を反映するだけでいいので、TransformクラスのTranslate関数を用いる方法を用います。

Input.GetAxisは、指定された軸方向の入力を数値にして取得できます。Horizontalなので、十字キーの横かA,Dキーの入力で操作できるようになります。

また、Time.deltaTimeは、Update関数が呼び出される間隔です。これをかけないと、Updateの間隔が変わるたびに移動速度が変わってしまうので、Translateで移動させるときは必ずこれをかけましょう。

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Player : MonoBehaviour {
6
7       float speed = 10.0f;
8
9       // Use this for initialization
10      void Start () {
11
12      }
13
14      // Update is called once per frame
15      void Update () {
16          transform.Translate (Input.GetAxis("Horizontal")*speed*Time.deltaTime,0,0);
17      }
18  }
```

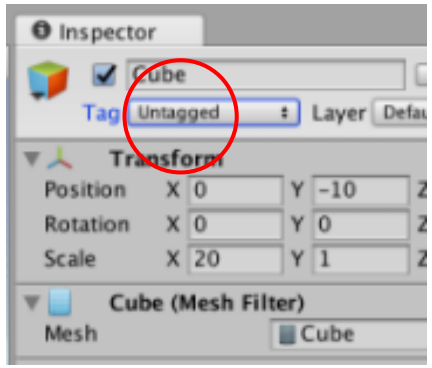
衝突の検知

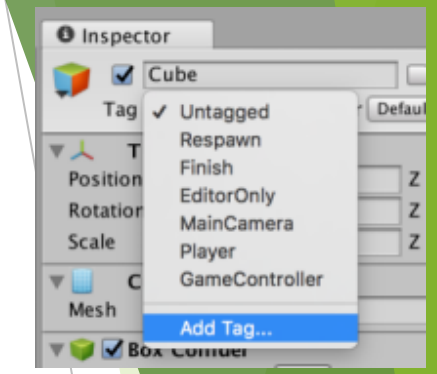
　ゲームオーバーの設定の準備として、“ボールが下の壁に当たったとき”を検知できるようにします。

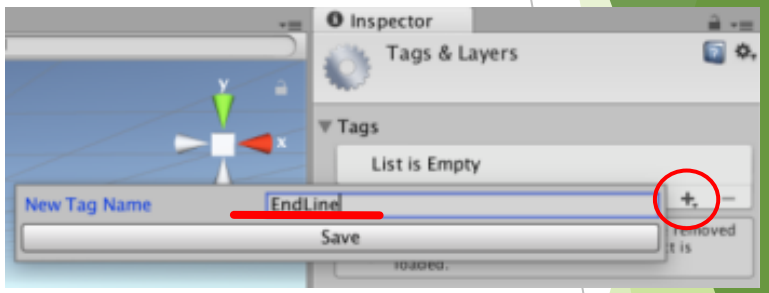
　衝突したものが何かを判別する方法として、オブジェクトに付いているタグを調べる方法があります。

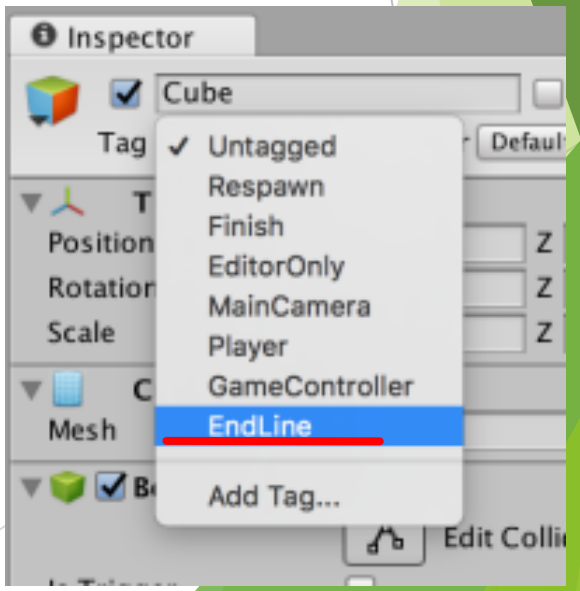
　下側の壁のオブジェクトを選択して、インスペクターのTagの項目を設定します。Untaggedとなっているボタンを押して、一番下のAdd Tagを選択。出てきた画面の+ボタンを押して、EndLineと入力してSaveボタンを押し、新しいタグを作成します。

　もう一度ヒエラルキーで下側の壁のオブジェクトを選択し、インスペクターのUntaggedのボタンを押すと、新しく作ったEndLineというタグが増えているので、これを選択します。









衝突の検知

コライダー同士が衝突した際、オブジェクトの持つOnCollisionEnterという関数が自動的に呼び出されます。

ボールのスクリプトを右のように書き換えました。OnCollisionEnterの引数には、衝突した相手のCollisionを取ることができます。右のようにすればここからtagを見ることができるので、これをEndLineという文字列と比較し、一致すれば処理を実行します。

今回はコンソールにGameOverと表示するだけにしました。

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Ball : MonoBehaviour {
6
7       Rigidbody rigid;
8
9       // Use this for initialization
10      void Start () {
11          rigid = gameObject.GetComponent<Rigidbody> ();
12          rigid.AddForce (1,4,0,ForceMode.VelocityChange);
13      }
14
15      // Update is called once per frame
16      void Update () {
17
18      }
19
20      void OnCollisionEnter(Collision coll){
21          if (coll.transform.tag == "EndLine") {
22              Debug.Log ("GameOver");
23          }
24      }
25  }
```

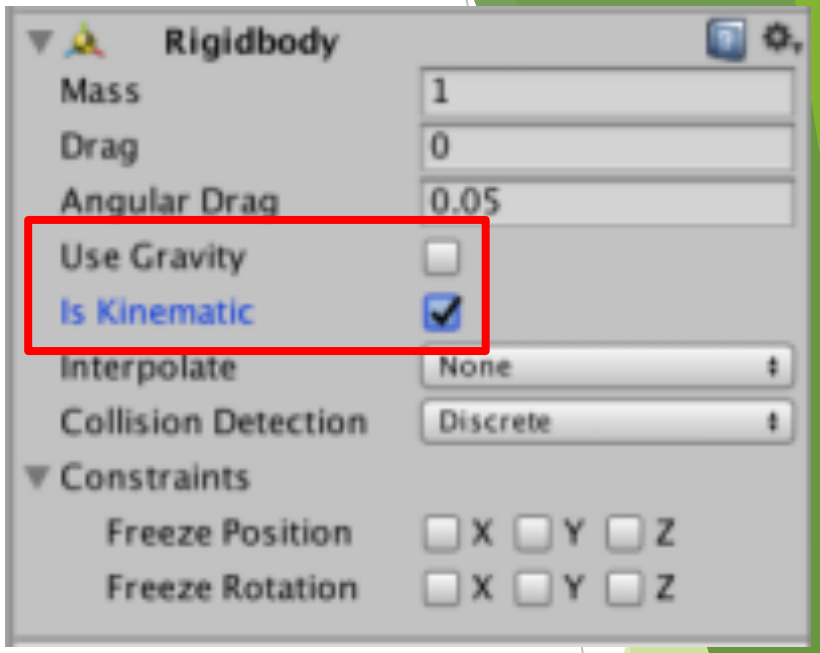
衝突の検知

今までのやり方では、横の壁に浅い角度で当たったときに垂直に反射し続けるようになり、ゲームの進行ができなくなるバグがありました。

これを解決するため、右のように、x軸方向の速度の最小値を決めておき、プレイヤーが触れた時の速度がそれを下回ったら無理やり加速するようにしました。
プレイヤーが操作するブロックにPlayerタグをつける必要がありますが、これで比較的簡単に解決できました。
今回は分かりやすく右向きに力を加えましたが、場合分けを丁寧にすればもっと自然な反射にできると思います。

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Ball : MonoBehaviour {
6
7       Rigidbody rigid;
8       float minVel = 3;
9
10      // Use this for initialization
11      void Start () {
12          rigid = gameObject.GetComponent<Rigidbody> ();
13          rigid.AddForce (1,4,0,ForceMode.VelocityChange);
14      }
15
16      // Update is called once per frame
17      void Update () {
18
19      }
20
21      void OnCollisionEnter(Collision coll){
22          if (coll.transform.tag == "EndLine") {
23              Debug.Log ("GameOver");
24          } else if (coll.transform.tag == "Player") {
25              if (rigid.velocity.x < minVel && rigid.velocity.x > -minVel) {
26                  rigid.AddForce (minVel, 0, 0, ForceMode.VelocityChange);
27              }
28          }
29      }
30  }
```

ブロックを置く



　ボールを当てて破壊するブロックを置きます。
　ヒエラルキーのCreateから3DObject→Cubeを作成します。

　衝突に必要なコンポーネントを追加します。インスペクターのAdd Componentから、Physics → Rigidbodyを追加します。Rigidbodyの useGravityのチェックを外し、isKinematicのチェックを入れます。これでボールに当たっても動かなくなりました。

　ボールとの衝突時に処理が必要なので、ブロックに持たせるスクリプトを書きます。C#Scriptファイルを新たに作成してください。

ブロックのスクリプト

今回も、OnCollisionEnter内でタグを比較し、Ballタグを持つオブジェクトと衝突したら破壊されるようにします。右のようなスクリプトになりました。

5行目のクラス名がファイル名と一致しているか必ず確認してください。

タグを使ってボールを認識するので、EndLineと同様にBallタグを新たに作成し、ボールにつけておきます。

Destroy関数は、引数に渡されたオブジェクトを破壊する関数です。強力すぎて使い勝手が悪いので、普通はSetActive関数でオブジェクトを無効化するなどの方が便利なのですが、今回はこの後の処理の関係で、Destroyして完全に消去してしまいます。

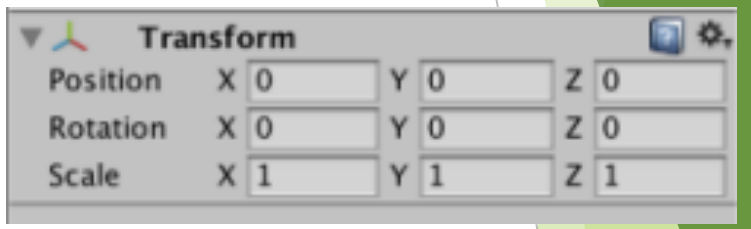
19行目に、gameObject と書いていますが、Unityでは、gameObjectと書くとこのスクリプトを持っているオブジェクト自身のことになります。つまり今回は、ブロック自身を渡して、自分を破壊するようにします。

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Block : MonoBehaviour {
6
7       // Use this for initialization
8       void Start () {
9
10      }
11
12      // Update is called once per frame
13      void Update () {
14
15      }
16
17      void OnCollisionEnter(Collision coll){
18          if (coll.transform.tag == "Ball") {
19              Destroy (gameObject);
20          }
21      }
22  }
```

ブロックを置く

Unityでは、オブジェクト間に親子関係というものをつけられます。これを使うと、親が移動・拡大縮小・回転すると子も追従して同じように変化するようになります。他にも親子関係を利用すると利用できる便利な機能があるので、今回は、ブロックが空のオブジェクトの子になるようにします。

ヒエラルキーのCreateからCreate Emptyを選択し、空のオブジェクトを一つ作成します。これをBlocksと名付けました。BlocksのTransformは、右図のようになっていることを必ず確認してください。親のTransformはそのまま子の配置にも反映されるため、これがずれていると、子のTransformの表記と実際の配置にずれが生じてしまいます。



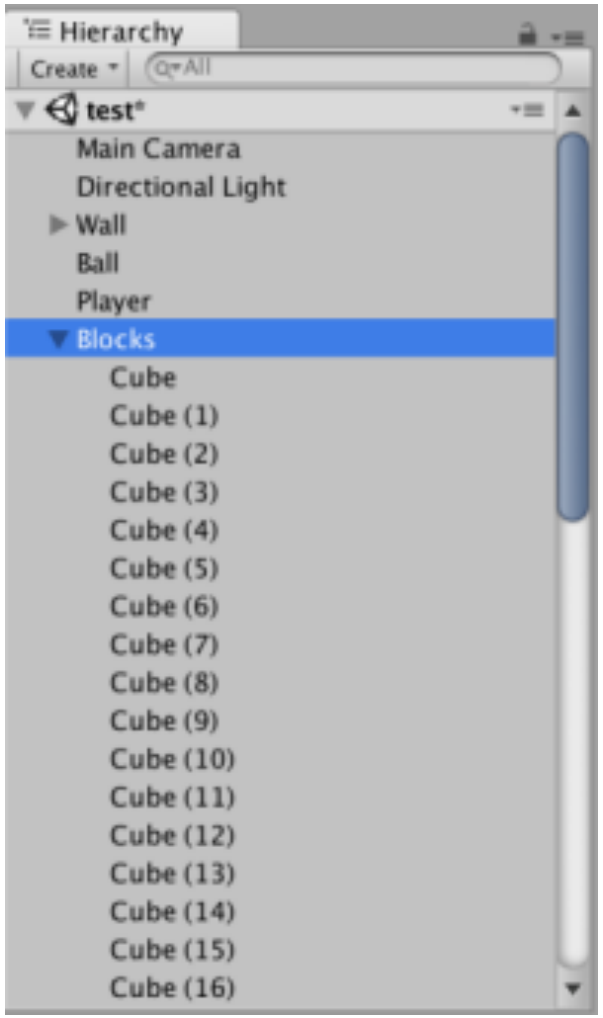
ブロックを置く

Rigidbodyとスクリプトをつけたブロックを、Blocksの子にします。

ヒエラルキーで、子にしたいオブジェクトを、親にしたいオブジェクトにドラッグ&ドロップしてください。右図のように、Blocksの中、一段下の階層にCubeが入ったようになります。

CubeをCtrl+C→Ctrl+Vで複製し、インスペクターで位置を調整して適当に並べてみましょう。

実行すると、ボールが当たったブロックが破壊されると思います。うまくいかない場合は、ボールのタグがBallになっているか見直してください。



クリアの検知

クリア時の処理をするために、ブロックが全て壊されたことを検知するスクリプトを用意します。新しくスクリプトを作成します。

16行目のように、transform.childCountとすると、オブジェクトの持つ子の数がわかります。ブロックの親に対してこれを使えば、残りのブロックの数がわかるので、クリアが検出できます。

Updateが呼び出されるたびに調べるので、クリア時の処理を何度も実行しないために、bool型の変数を使って、すでにクリアしている時は実行しないようにしています。

これを保存して、ブロックの親に持たせましょう。

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class CntBlock : MonoBehaviour {
6
7       bool isClear = false;
8
9       // Use this for initialization
10      void Start () {
11
12      }
13
14      // Update is called once per frame
15      void Update () {
16          if (!isClear && transform.childCount == 0) {
17              isClear = true;
18              Debug.Log ("Clear!");
19          }
20      }
21  }
22
```

ビルド

ここまででブロック崩しとして遊べるものは完成です。クリア・ゲームオーバー時の処理、ポーズ機能など、まだまだやることはたくさんありますが、ゲームとして出力する方法だけ紹介しようと思います。

その前に、このままPC版のゲームを作ると、ゲーム終了の処理を用意していないため、強制終了するしかなくなってしまいます。Escキーでゲームを終了する処理だけ追加します。

ブロックの親のスクリプトのUpdate内に、右のように21行目から24行目の処理を追加します。これでEscapeキーが押された時、ゲームを終了するようになります。

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class CntBlock : MonoBehaviour {
6
7       bool isClear = false;
8
9       // Use this for initialization
10      void Start () {
11
12      }
13
14      // Update is called once per frame
15      void Update () {
16          if (!isClear && transform.childCount == 0 ) {
17              isClear = true;
18              Debug.Log ("Clear!");
19          }
20
21          //ゲーム終了の処理
22          if (Input.GetKeyDown (KeyCode.Escape)) {
23              Application.Quit ();
24          }
25      }
26  }
```

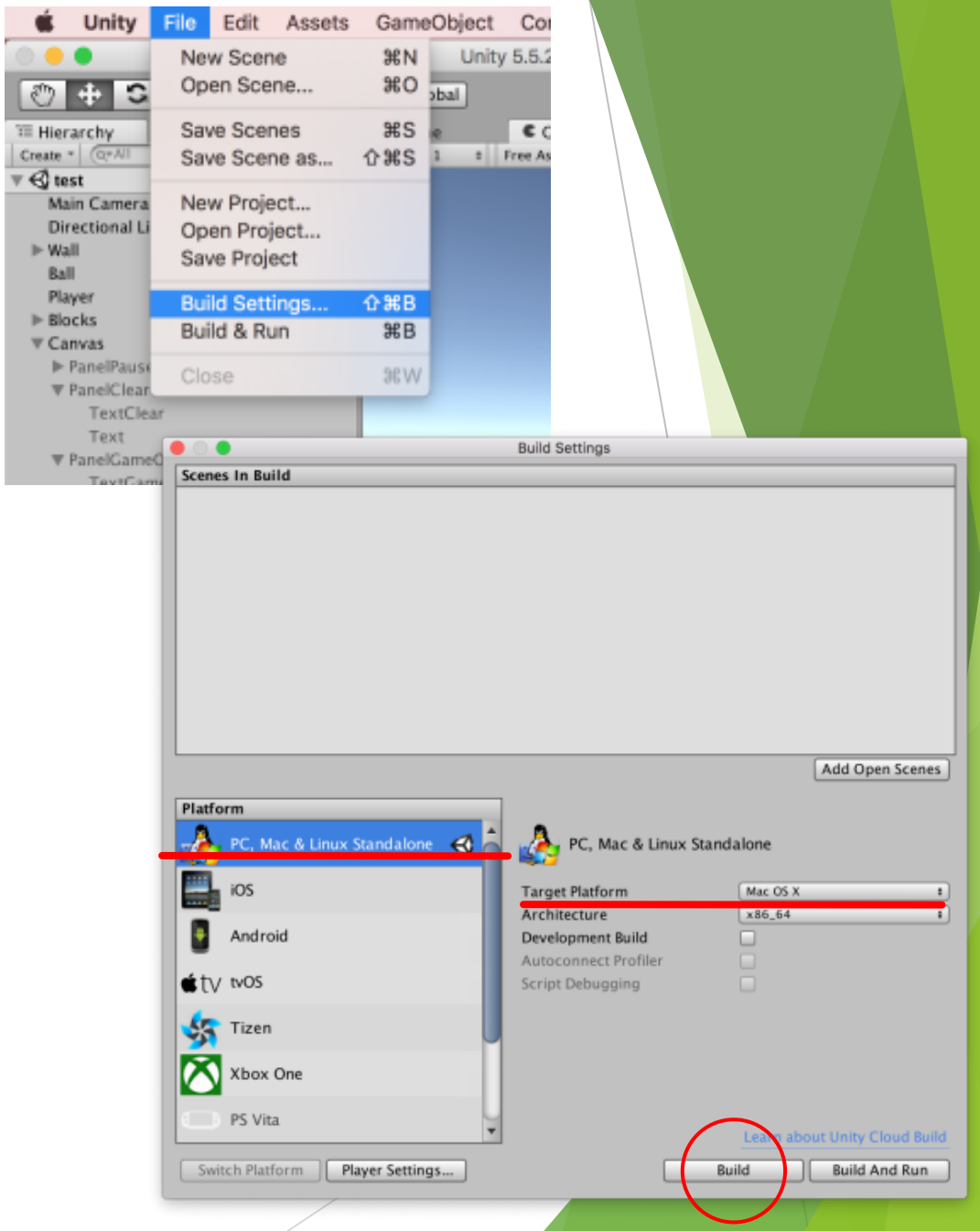
ビルド

右図のようにFileメニューからBuild Settingsを開きます。

開いたウィンドウの上半分は、今から出力するゲームに使うシーンの一覧です。今は何も入っていないと思うので、画面右端の、AddOpenScenesボタンを押し、今開いているシーンを追加します。

下半分は、どの環境で実行できるように出力するか選択する部分です。スマホ等で操作できるようにはしていないので、左側でPCを選択し、右側のTarget PlatformでMacかWindowsを選択しましょう。

画面下のBuildボタンを押して名前をつけると、ビルドが開始されます。



動作確認

　これで、ブロック崩しの基本的な部分は完成です。
　ここまででうまく動作しない場合は、タグやコンポーネントを見直してみてください。
　また、コンソールをみてみると、エラーや警告が出ていることがあります。

Unity公式のマニュアル
(https://docs.unity3d.com/jp/current/Manual/index.html)などで調べてみるのもいいと思います。