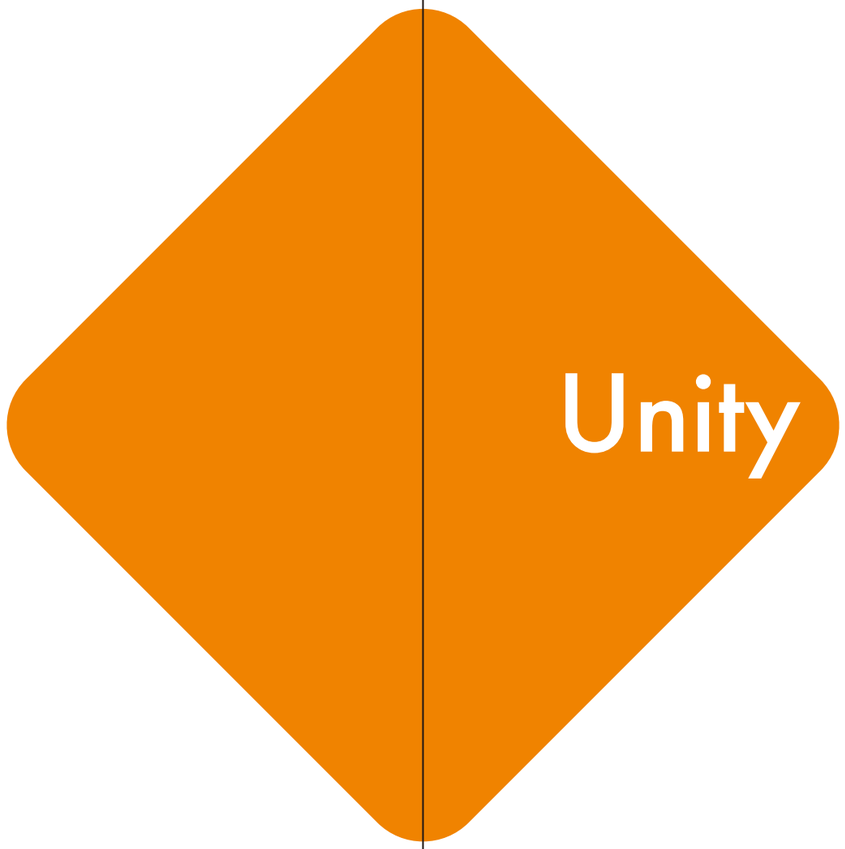


プログラミング
実践編

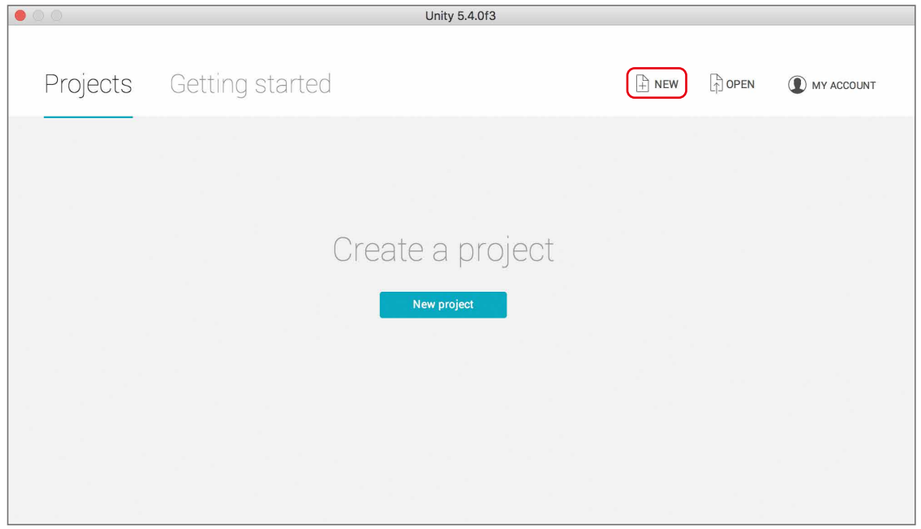
〈発行〉 Programming School ZERO ONE studio プログラミングスクール ゼロワンスタジオ

〒470-0136 日進市竹の山3丁目412 ザ・ネクステージ 3F
Tel／Fax.0561-73-0132　E-mail pgs@zero-one-studio.com

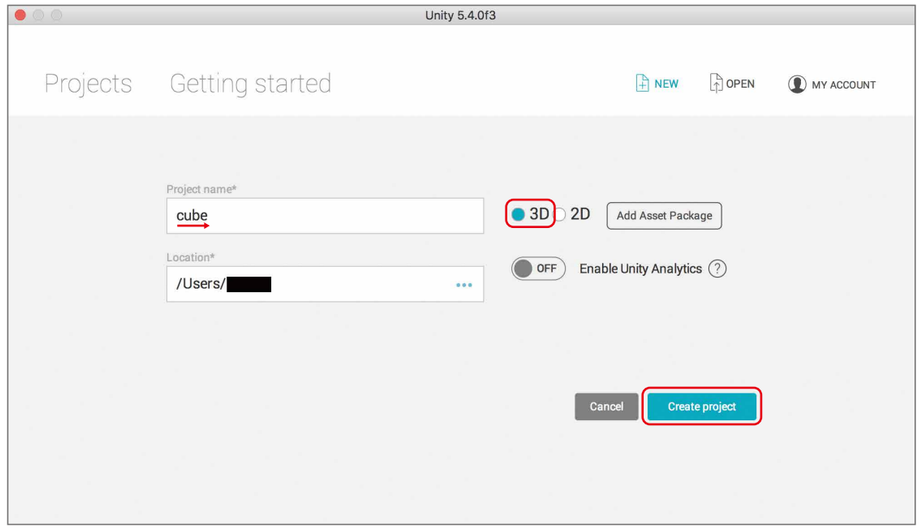
Programming School ZERO ONE studio プログラミングスクール ゼロワンスタジオ

プロジェクトを作る

Unityを起動後、プロジェクト選択ウインドウのNEW（ニュー）をクリック。

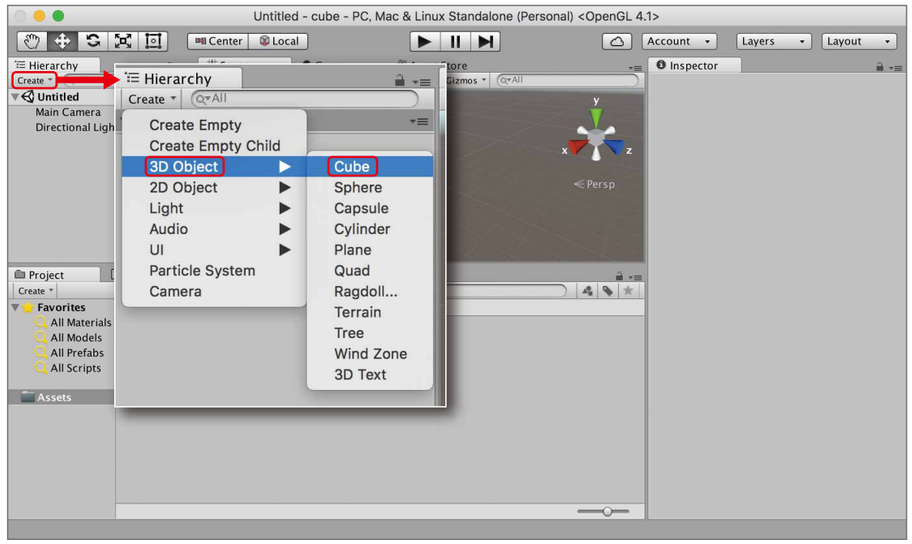


Project name（プロジェクト・ネーム）の欄に、「cube」と入力し、３Dにチェックをいれます。
その後、Create project（クリエイト・プロジェクト）をクリックすることで、プロジェクトを作成できます。

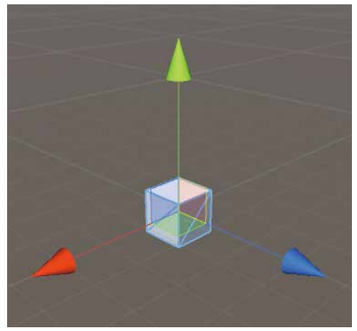


キューブを作る

Hierarchy（ヒエラルキー：階層）のCreate → 3D Object → Cubeをクリックします。



下の絵のように、Cube（キューブ：立方体）があらわれます。



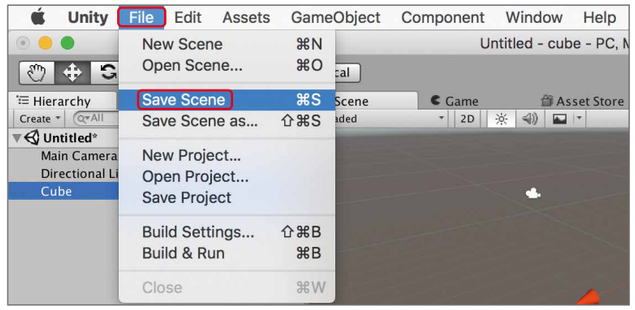
立方体は、たて、よこ、高さがすべて同じ長さのサイコロ状の立体です。
このような物体を、Unityでは「オブジェクト」と呼びます。

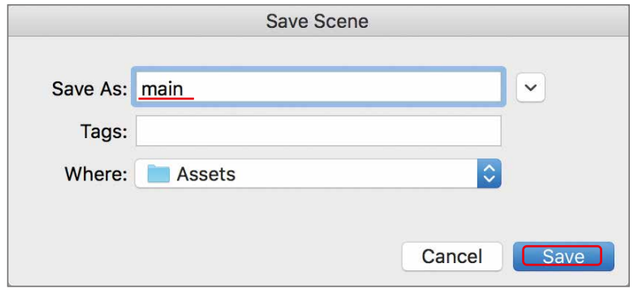
解説 オブジェクト **Objectとは**

Object（オブジェクト）とは、「物体」や「目的」を意味する言葉です。パソコン上では、操作する対象のことを指す言葉です。
Unityでは、Hierarchy（ヒエラルキー）に表示されるものを、オブジェクトと呼ぶことが多いです。また、先ほど作ったCubeは、オブジェクトに含まれます。

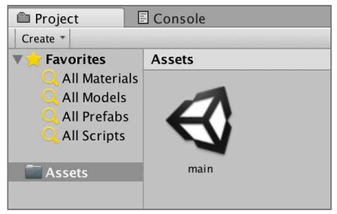
セーブする

現状をセーブしておきます。
画面上部のメニューのFile → Save Sceneをクリックします。もしくは、Command + Sキーを押します。
Save asの欄に、「main」と入力し、Saveをクリックします。
これで現状がセーブできます。今後も、こまめにセーブしますので覚えておきましょう。





Project（プロジェクト）に、mainがあれば成功です。



再生する

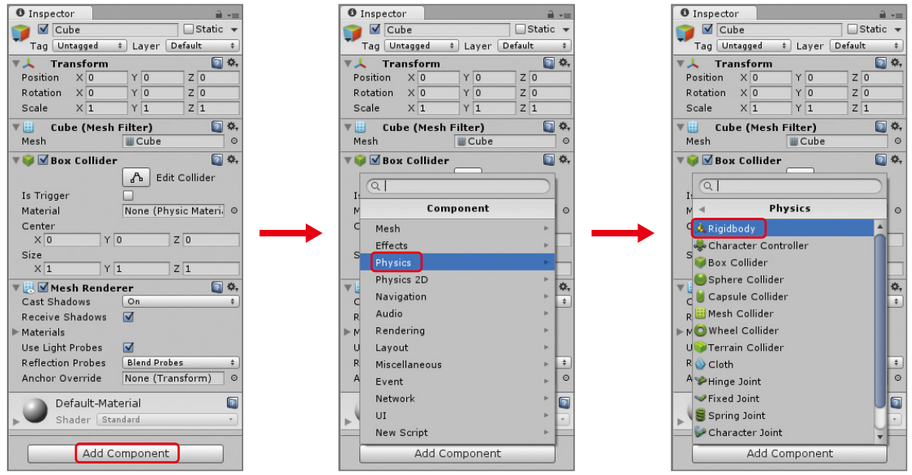
ここまで作ったゲームを実行してみましょう。
再生ボタンを押してみます。
画面中央上あたりにある右三角ボタンをクリックします。

再生ボタンは、一度押すことで再生状態となります。（右の図）　もう一度押すと再生が止められます。
まだ、何も変化はありません。次は、Cubeを重力により落下させます。
再生を止めてください。

Rigidbody（リジッドボディ）を付ける

Cubeに、Rigidbody（リジッドボディ：剛体）を付けます。
Hierarchy（ヒエラルキー：階層）のCubeをクリックし、選択します。するとInspector（インスペクター：調査）にCubeの情報が表示されます。
InspectorのAdd Component（アッド・コンポーネント：構成要素の追加）→ Physics（フィジックス：物理）→ Rigidbody（リジッドボディ：剛体）の順にクリックします。

再生ボタンを押してみましょう。キューブが下に落ちたら成功です。

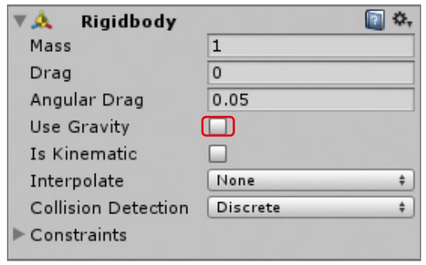


解説　Rigidbody（リジッドボディ）とは

Rigidbodyを付けることで、物理計算を行うようになり、重力により落下します。
また、Collider（コライダー：衝突装置）というものを一緒に付けると、他の物体とぶつかり、より現実的な挙動をするようになります。

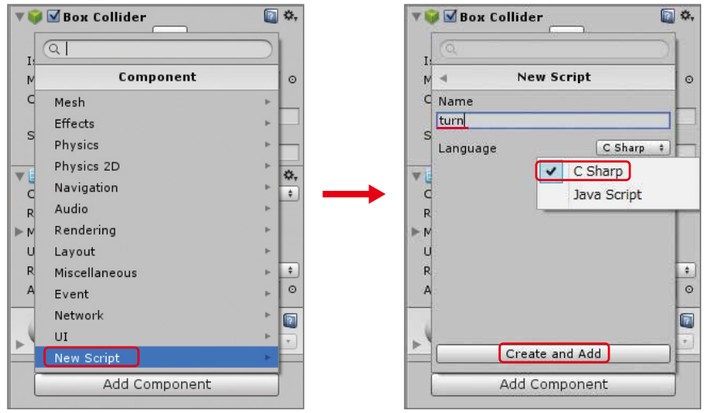
重力の影響を受けないようにする

InspectorのRigidbodyのUse Gravity（ユーズ グラビティ：重力を使う）のチェックを外します。これで、Cubeには重力が働かなくなります。
チェックを外したら、再生ボタンを押してみましょう。キューブが動かなければ成功です。



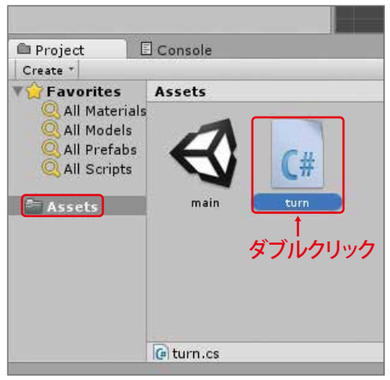
スクリプトを付ける

Cubeに、スクリプトを付けます。
Hierarchy（ヒエラルキー：階層）のCubeをクリックし、選択します。
Inspector（インスペクター：調査）のAdd　Component（アッド・コンポーネント：構成要素の追加）
→ New Script（ニュー・スクリプト：新しいスクリプト）とクリックします。
Name（ネーム：名前）を「turn」とし、Language（ランゲイジ：言語）を「C Sharp」として、Create and Add（クリエイト・アンド・アッド：生成して追加）をクリックします。
Inspector（インスペクター：調査）にTurn (Script) が追加されていれば成功です。



スクリプトを編集する

まずは、スクリプトを開きます。
Assetsの中のturnをダブルクリックします。スクリプトを編集するアプリケーションが起動します。



次の図のように打ち込んだら、Command＋Sでスクリプトをセーブします。

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class turn : MonoBehaviour {
5
6     // Use this for initialization
7     void Start () {
8
9     }
10
11    // Update is called once per frame
12    void Update () {
13        gameObject.transform.Rotate(0f,1f,0f);
14    }
15 }
16
```

キューブを回転させるときは、transform.Rotateを使います。

解説 トランスフォーム　ローテート
transform.Rotateとは

Transform(トランスフォーム：変換)は、Position（ポジション：座標）やRotation（ローテーション：回転)、Scale（スケール：大きさ)などの数値を持っています。
Rotate(ローテイト：回転させる)は、オブジェクトを回転させる命令です。引数（カッコ内に書く数値）は、順番にX軸の回転量、Y軸の回転量、Z軸の回転量です。
X、Y、Z軸とは、座標軸と呼ばれるもので、回転の基準となる軸です。
1fや0fなどfをつけるのは、float（フロート）つまり、実数型であることを示しています。

解説 アップデート
Update()とは

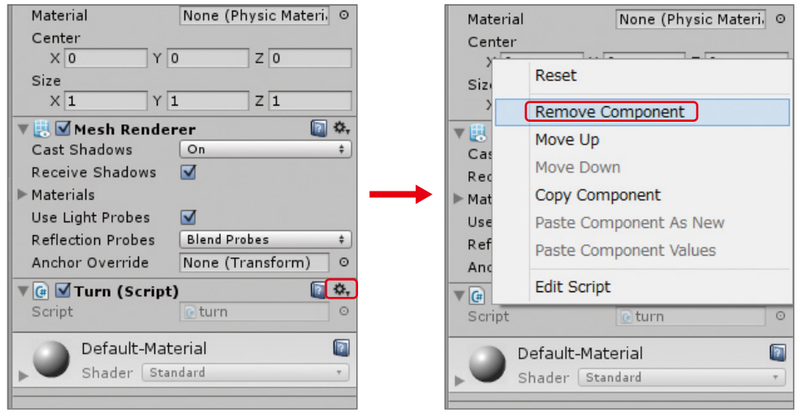
Updateの後の {} 中カッコ内には、1フレーム分の命令を書きます。ここに書いた命令は、1秒間に数十回処理されます。
例えば、映画は、一秒間に24回絵を切り替えて動いているように見せています。1回の切り替えのことを1フレームといいます。
Updateの後の中カッコ内は、1フレームの動きを書く場所になります。次のフレームまでどんな動きをさせるかという命令文を書いていきます。
先ほど書いたtransform.Rotateは、1フレームに1度回転させるという命令になります。
説明が理解できなかった人でも今は、繰り返しおこなう命令はここに書く、と覚えておきましょう。

★課題
Rotate関数のカッコ内の数値を変えて以下の課題に挑戦してみてください。
（1）回転スピードを上げてください。
（2）X軸回りに回転させてください。
（3）Z軸回りに回転させてください。

Rotate関数の引数を大きくするとどうなるでしょう、またマイナスの値を入れるとどうなるでしょう。

スクリプトを外す

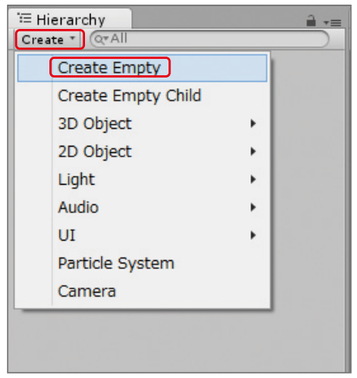
Cubeに付いているスクリプトを外します。
Hierarchy（ヒエラルキー：階層）でCubeを選択し、Inspector（インスペクター：調査）のTurn(Script)の歯車をクリックし、Remove Component（リムーブ・コンポーネント：構成要素の削除）をクリックします。



再生ボタンを押してみましょう。
スクリプトという構成要素をオブジェクトから取り外したため、回転しません。

空のオブジェクトを作る

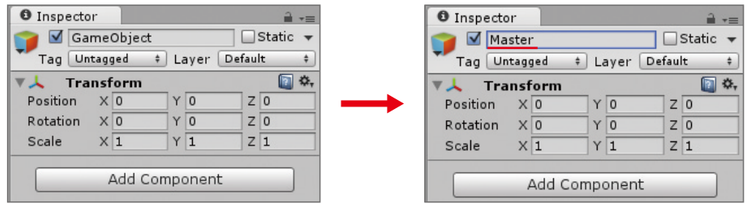
今度は、空のオブジェクトを作ります。
Hierarchy（ヒエラルキー：階層）のCreate（クリエイト：生成） → Create Empty（クリエイト・エンプティー：空オブジェクトの生成）をクリックします。



Hierarchy(ヒエラルキー:階層)にGameObjec(t ゲームオブジェクト)という名前のオブジェクトが作られます。

オブジェクトの名前を変更

Hierarchy（ヒエラルキー：階層）からGameObjectをクリックし、選択します。
Inspector（インスペクター：調査）の一番上に名前が表示されています。そこをクリックし「Master」と入力してEnter（エンター）キーを押して適用します。



解説 エンプティ **Emptyオブジェクト**

Empty(エンプティ)を日本語に訳すと「空の」という意味になります。
作成したオブジェクトは、何も付いていないオブジェクトになります。そのためCubeの立方体のように目に見えるものも持っていません。ただし、Transform(トランスフォーム)という位置情報などの基本的な情報は持っています。

Masterオブジェクトにスクリプトを付ける

次は、Master（マスター）に付けたスクリプトからCubeを回転させます。
Hierarchy（ヒエラルキー：階層）でMasterを選択してInspector（インスペクター：調査）のAdd Component（アッド・コンポーネント）→ New Script（ニュー・スクリプト）をクリックします。
スクリプトのName（ネーム：名前）は「turn2」としてLanguage（ランゲイジ：言語）は、「C sharp」とします。

スクリプトを開きます。Assets（アセット）の中にあるturn2をダブルクリックします。
スクリプトを次のように記述します。

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class turn2 : MonoBehaviour {
5
6     GameObject cubeobj;
7
8     // Use this for initialization
9     void Start () {
10        cubeobj = GameObject.Find ("Cube");
11    }
12
13    // Update is called once per frame
14    void Update () {
15        cubeobj.transform.Rotate (0f, 1f, 0f);
16    }
17 }
18
```

スクリプトを記述したら、Command + Sでセーブします。

解説 Start関数とは

Startの後の{}中カッコ内には、準備を行うための命令文を書きます。最初に一度だけ実行されます。

解説 GameObject.Findとは（ゲームオブジェクト　ファインド）

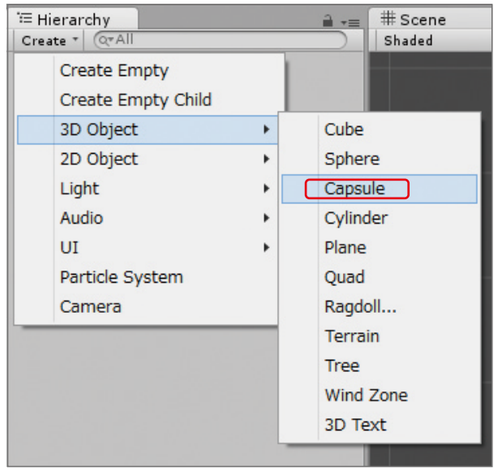
GameObject.Findとは指定した名前のオブジェクトを探し出し取得する命令です。

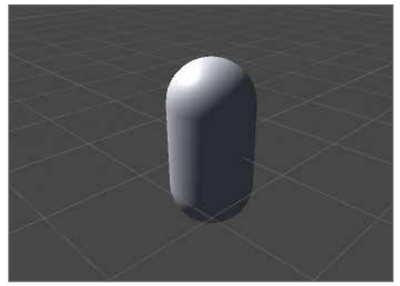
```
3
4 public class turn2 : MonoBehaviour {
5
6     GameObject cubeobj;   ← オブジェクトを入れるもの
7
8     // Use this for initialization
9     void Start () {       ← Cube という名前のオブジェクトを探す
10
11        cubeobj = GameObject.Find ("Cube");
12
13    }
14                          ← cubeobj に Cube を入れる
15    // Update is called once per frame
16    void Update () {      ← cubeobj に入っている Cuve を回転させる
17
18        cubeobj.transform.Rotate (0f, 1f, 0f);
19
20    }
21 }
22
```

GameObject.Findを使うことで、別のスクリプトからオブジェクトを取得して操作することができます。

Capsule（カプセル）を作る

Hierarchy（ヒエラルキー：階層）のCreate（クリエイト）→ 3D Object → Capsule（カプセル）をクリックします。





Capsuleを選択した状態で、Transform（トランスフォーム）のPosition（ポジション）のXを、2にします。
再生ボタンを押してみましょう。
キューブの横にカプセルがあります。

カプセルを回転させるスクリプトを追加する

Assetsの中のturn2をダブルクリックして開きます。
次のように、turn2のスクリプトを書き換えます。
書き換えたらセーブを行い、再生ボタンを押してみましょう。

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class turn2 : MonoBehaviour {
5
6     GameObject cubeobj;
7     GameObject capobj;
8
9     // Use this for initialization
10    void Start () {
11        cubeobj = GameObject.Find ("Cube");
12        capobj = GameObject.Find ("Capsule");
13    }
14
15    // Update is called once per frame
16    void Update () {
17        cubeobj.transform.Rotate (0f, 1f, 0f);
18        capobj.transform.Rotate (1f, 0f, 0f);
19    }
20 }
21
```

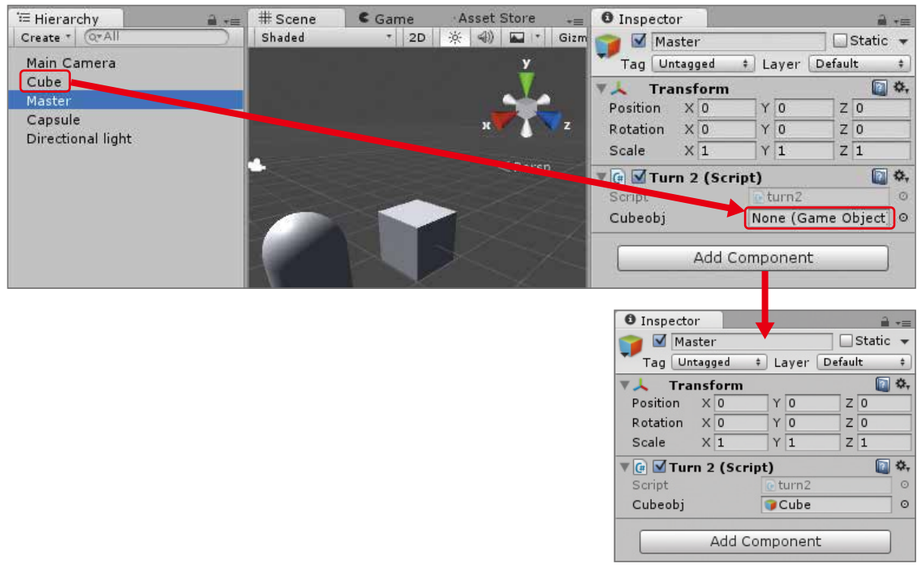
Inspectorから編集できるようにする

public（パブリック：公開）を書き加えます。
Unityでは、宣言文にpublic（パブリック）を付けることによってInspector（インスペクター）で、その値が編集できるようになります。
turn2を開いて、次のように書き換えます。Command＋Sでセーブします。

```
using UnityEngine;
using System.Collections;

public class turn2 : MonoBehaviour {

    public GameObject cubeobj;
    GameObject capobj;

    // Use this for initialization
    void Start () {

        capobj = GameObject.Find ("Capsule");
    }

    // Update is called once per frame
    void Update () {
        cubeobj.transform.Rotate (0f, 1f, 0f);
        capobj.transform.Rotate (1f, 0f, 0f);
    }
}

```

Cube = GameObject.Find（“Cube”）;　の一行を削除

Unityの画面に戻り、Hierarchy（ヒエラルキー）でMasterをクリックします。　すると、InspectorのTurn2(Script)に、Cubeobjという枠が増えています。
HierarchyからCubeを、Cubeobjの枠内にドラッグアンドドロップ（以下DDという）します。



Inspector(インスペクター)からCubeobjにCubeを入れているのでGameObject.Findの一行が不要になります。
再生ボタンを押してみましょう。　Cubeが回転すれば成功です。

★課題
（1）capsule（カプセル）を、パブリック変数で書き換えてみましょう。
課題（1）の答え
次のようにスクリプトを書き換えます。

```
using UnityEngine;
using System.Collections;

public class turn2 : MonoBehaviour {

    public GameObject cubeobj;
    public GameObject capobj;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        cubeobj.transform.Rotate (0f, 1f, 0f);
        capobj.transform.Rotate (1f, 0f, 0f);
    }
}

```

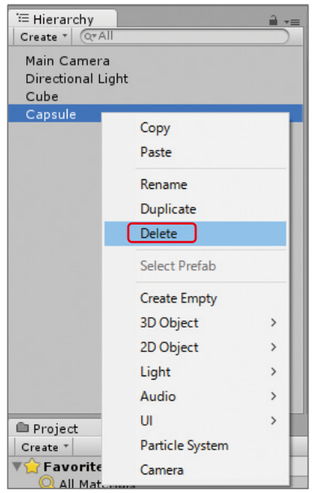
書き換えた後は、Command＋Sでセーブしてください。

Unityの画面に戻り、HierarchyのMasterをクリックします。
するとInspectorのTurn2(Script)に、Capobjという枠が増えています。
Hierarchyから、Capsuleを、Capobjの枠内にDDします。

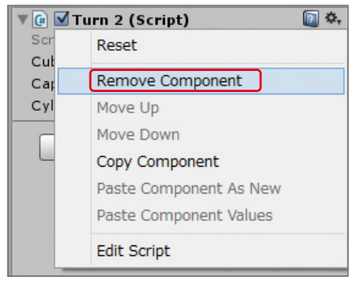
再生してみましょう。

ゲーム作りの準備

次にキューブを使ってゲームを作ります。
Capsule（カプセル）は、不要なのでDelete（デリート：削除）します。
Hierarchy（ヒエラルキー）のCapsuleを選択します。Hierarchy（ヒエラルキー）の枠の中のCapsuleを右クリックし、Deleteを選択します。



Cubeを回転させるためのスクリプト（turn2）は、不要となりますので、Masterからturn2を取り外します。
Hierarchy（ヒエラルキー）でMasterを選択し、Inspector（インスペクター）のTurn2(Script)の歯車をクリックし、Remove Component（リムーブ・コンポーネント：構成要素の削除）をクリックします。

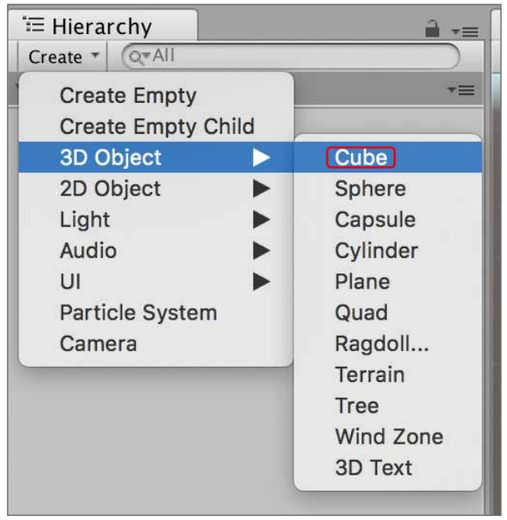


再生してみましょう。動かないキューブ(Cube)が表示されていれば成功です。

キューブの下に床（Floor）を作る

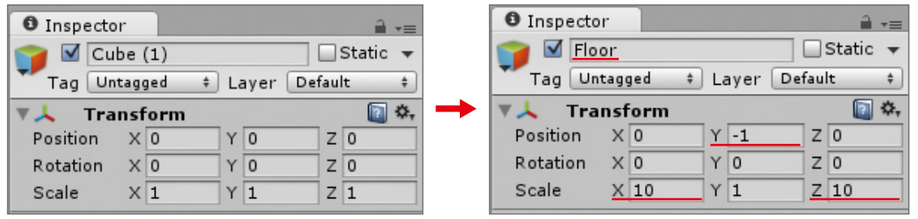
もう一つキューブを追加します。キューブの作り方を覚えていますか？

Hierarchy（ヒエラルキー）でCreate → 3D Object → Cubeの順にクリックします。

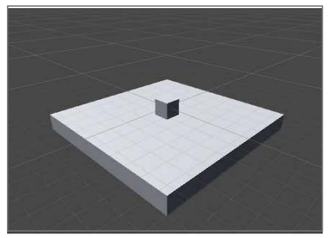


Cube (1)が作成されますが　Scene（シーン）を見ても新しいキューブがありません。これは、２つのキューブが重なっているからです。

Cube (1)のTransformを次のように設定します。
PositionのYを、-1とし、
ScaleのXとZを、10にします。
さらにCube (1)の名前を、「Floor」に書き換えます。

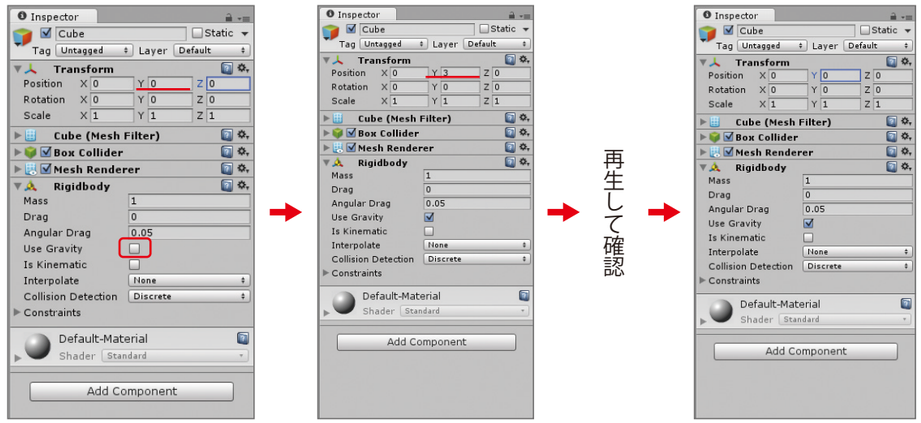


Cubeの下にFloorが出来ています。



キューブに重力をふたたび作用させる

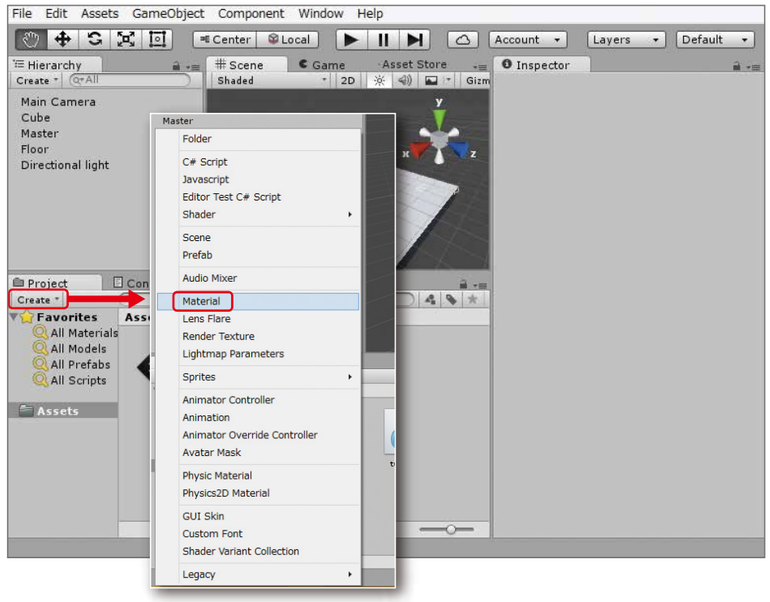
Hierarchy（ヒエラルキー）でCubeを選択し、Inspector（インスペクター）のRigidbody（リジッドボディ：剛体）のUse Gravity（ユーズ・グラビティー）にチェックを入れます。
Cubeが重力で落下するかを確認するためにTransform（トランスフォーム）のPosition（ポジション）のYを3にします。

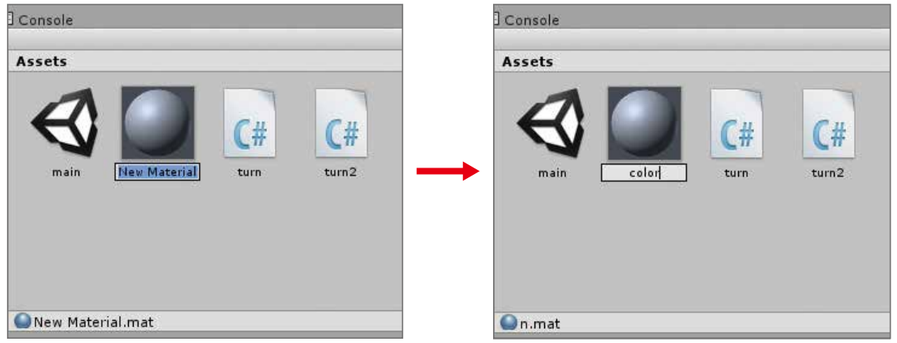


再生ボタンを押してみましょう。
キューブは少し落下した後、床の上に乗っていて動かなければ成功です。
落下するのを確認したらTransform（トランスフォーム）のPosition（ポジション）のYを0に戻します。

キューブ（Cube）に色を付ける

Unityでは、Material（マテリアル：材質）を使って色を付けます。
左下のProject（プロジェクト）で、Create（クリエイト）→ Material（マテリアル）をクリックします。
Materialの名前が入力できる状態となっていますので、「color」としましょう。
※Projectの方のCreateですので、間違えないようにしましょう。

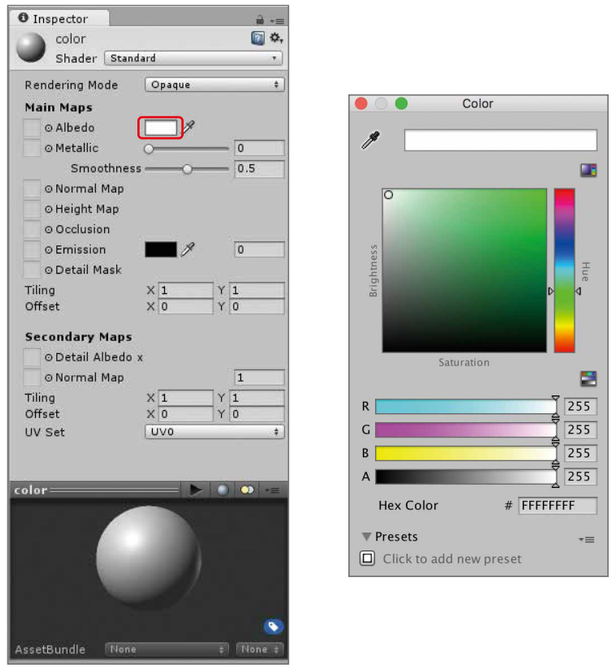




解説　Material（マテリアル）とは

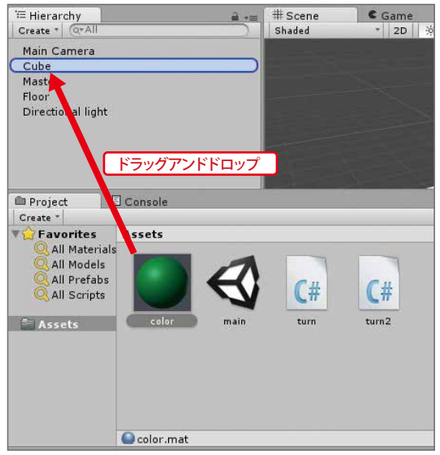
Materialとは、訳すと「素材」や「材質」という意味です。
Unityでは、物体の表面を表現するのに使います。物体の色の設定をしたり、テクスチャーと呼ばれる画像を張って模様などを表現します。

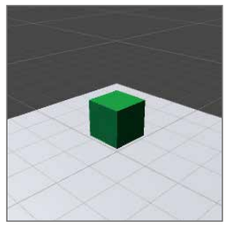
Assets（アセット：資産）の中のcolorを選択し、Inspector（インスペクター）に表示されている
Albedo（アルベド：反射率）の右側にある白色の枠をクリックします。すると、カラーパレットが表示
されます。



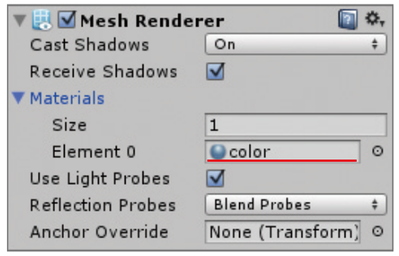
右側の縦長のバーで好きな色を選択し、左側の四角の枠の中をクリックして、色の明るさを選択します。
この教材では、緑色にしました。

色を変更すると、Assets（アセット）の中のcolorというMaterialが、設定した色になっています。
このcolorを、Hierarchy（ヒエラルキー）にあるCube（キューブ）にDDします。また、Scene（シーン）の
CubeにDDしても同じ操作ができます。
Sceneに表示されているCubeの色が変われば成功です。





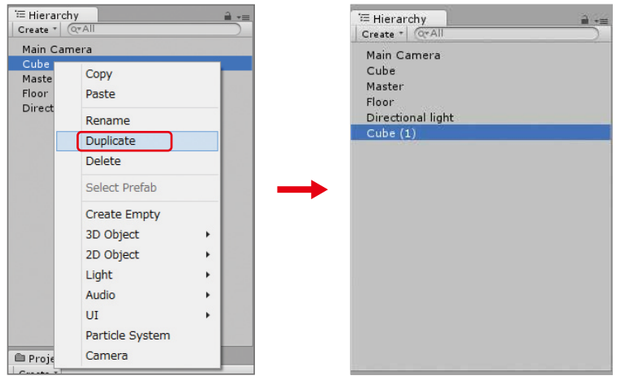
このcolorというMaterial（マテリアル）は、CubeのMesh Renderer（メッシュレンダラー）のMaterials
（マテリアル）に設定されています。
これを確かめてみましょう。
Hierarchy（ヒエラルキー）でCubeを選択します。Inspector（インスペクター）の中にMesh
Renderer（メッシュ レンダラー）が表示されます。その中のMaterialsをクリックします。
すると、次のように、colorが設定されていることが分かります。



※このElement 0となっている枠の中に、Assets（アセット）のcolorをDDしても同じように色の設定
ができます。

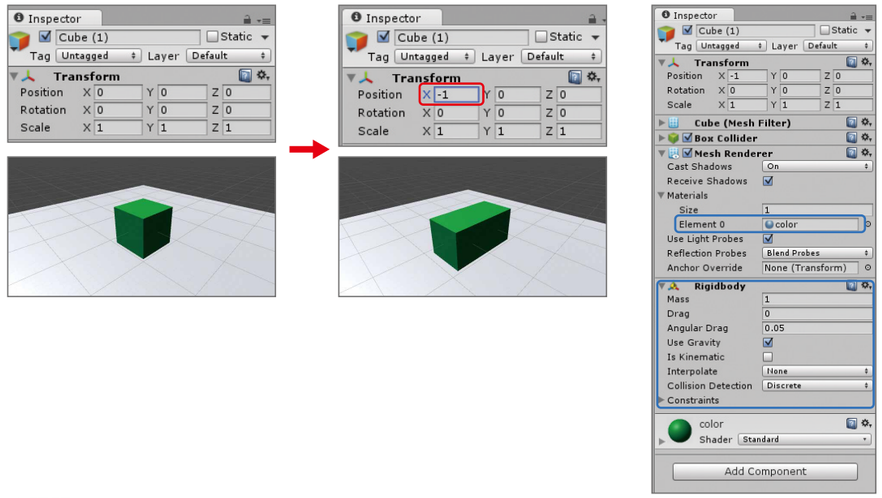
Cubeを増やす

次にキューブを増やしてみます。
Hierarchy（ヒエラルキー）のCubeを選択し、右クリックし、Duplicate（ディプリケート：複製）を選択します。



HierarchyにCube (1)が出来ていれば成功です。

Cube (1)は、Cubeに重なっていますので、位置を調整します。
　Hierarchy（ヒエラルキー）でCube　(1)を選択し、Inspector（インスペクター）のTransform（トランスフォーム）のPosition（ポジション）のXを、-1にします。
Material（マテリアル）とRigidbody（リジッドボディ）が設定されたCubeを複製しましたので、Cube (1)にも、色が付いていてRigidbody（リジッドボディ）も付いています。

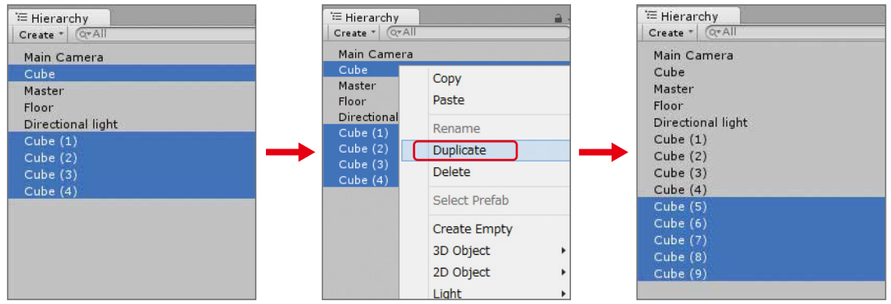


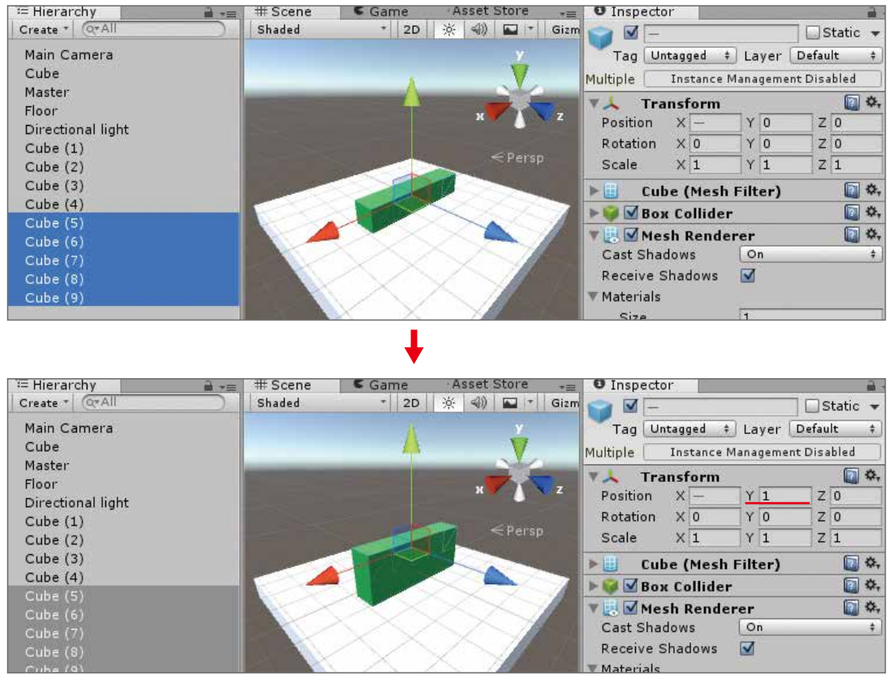
★課題
（１）Cubeを三つ複製して、Cube (2) Cube (3) Cube (4)を作成しましょう。
（２）複製したCubeの座標を以下の通りに設定しましょう。
　　Cube (2)のPositionのXを-2
　　Cube (3)のPositionのXを1
　　Cube (4)のPositionのXを2

再生ボタンを押してみましょう。　Cubeが横に並んでいたら成功です。

Cubeを積み上げる

五つ並んでいるCubeの上にCubeを積み上げます。
まずは、Cubeを五つまとめて複製します。
Hierarchy（ヒエラルキー）のCubeを選択します。その後、commandキーを押しながらCube　(1)　～ Cube (4)を順にクリックします。
右クリックをしてDuplicate（ディプリケート：複製）をクリックします。
すると、Cubeが五つ複製されてCube(5)、Cube (6)、Cube (7)、Cube (8)、Cube (9)がHierarchyに表示されます。
Cube(5) ～ Cube (9)を選択した状態でTransform（トランスフォーム）のPosition（ポジション）のYを1にします。



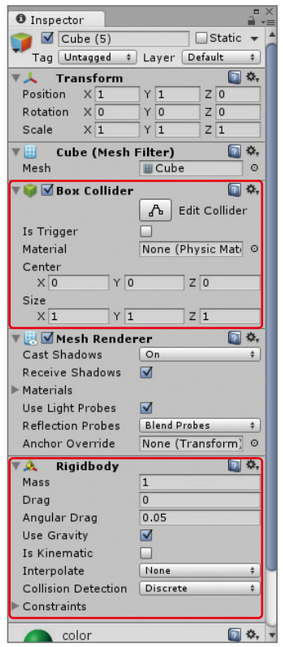


同じようにあと一段積み上げます。
Cube(5) ～ Cube (9)を選択し、右クリックしてDuplicate（ディプリケート：複製）をクリックします。
すると、Cube(10)、Cube (11)、Cube (12)、Cube (13)、Cube (14)がHierarchyに表示されます。

Cube(10) ~ Cube (14)を選択された状態で、TransformのPositionのYを、2にします。
★課題
（１）Cube (15)、Cube (16)、Cube (17)、Cube (18)、Cube (19)を作ってみましょう。
（２）Cube (15)、Cube (16)、Cube (17)、Cube (18)、Cube (19)が同時に選択された状態で、
TransformのPositionのYを、3にしてみましょう。

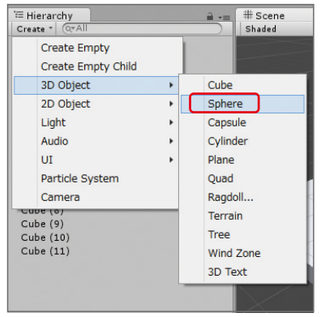
すべてのCubeには、Rigidbody（リジッドボディ）が付いています。
また、Hierarchy（ヒエラルキー）のCreateから作成するオブジェクトには、Collider（コライダー：衝突装置）が付いています。

適当なCubeをクリックし、Rigidbody（リジッドボディ）とBox Collider（ボックス・コライダー）がついていることを確かめましょう。



ボールを作る

Hierarchy（ヒエラルキー）で、Create → 3D Object → Sphere（スフィア：球）をクリックします。



Sphereの名前を「Ball」に変更します。名前はなんでもよいですが、練習の意味で「Ball」に変更しています。
Ball（ボール）に色を付けます。
Project（プロジェクト）から、Create（クリエイト） → Material（マテリアル：材質）を選択します。
名前を「color2」とします。
Assets（アセット：資源）のcolor2を選択し、Inspector（インスペクター：調査）のAlbedo（アルベド）の右側にある白色の枠をクリックします。
カラーパレットで、好きな色を選択します。
Assets（アセット）の中のcolor2をScene（シーン）にあるBallにDDします。

Ballを飛ばすために、Rigidbody（リジッドボディ）を付けておきます。
Hierarchy（ヒエラルキー）でBallを選択してInspector（インスペクター）のAdd Component（アッド・コンポーネント：構成要素の追加） → Physics（フィジックス：物理） → Rigidbody（リジッドボディ）をクリックします。

ボールをプレハブ化する

Hierarchy（ヒエラルキー）のBallを、Assets（アセット）の中へDDして、プレハブ化します。
Assets（アセット）にBallが表示されれば成功です。

解説 Prefab（プレハブ）とは

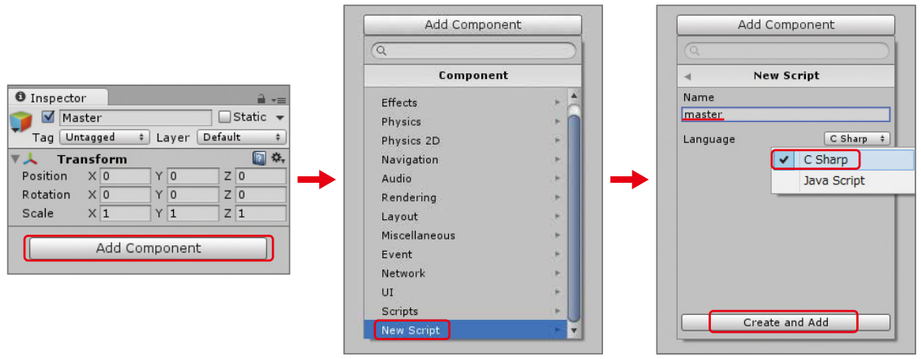
プレハブとは、オブジェクトの設計図のようなものです。プレハブをもとに同じオブジェクトを生成することができます。
HierarchyからProjectにDDすることによってプレハブを作ることができます。プレハブを作ることをプレハブ化といいます。

プレハブから複製したものは、プレハブの情報を共有する特徴を持っています。
Hierarchy上で単純に複製されたものは、同じ情報を持っていますが、全く関係のないものとして扱われます。片方のInspectorの情報を編集してももう片方には変化がありません。しかし、プレハブから作られたオブジェクトは、関連付けされています。プレハブのInspectorの情報が変化すると、それらのオブジェクトにも同じ変化が起こります。

Ballを削除します。
プレハブ化をし、Project（プロジェクト）にBallのプレハブがあるためHierarchy（ヒエラルキー）にあるBallは、不要になります。Hierarchy（ヒエラルキー）のBallを削除します。
Hierarchy（ヒエラルキー）のBallを選択して右クリックのメニューからDelete（デリート：削除）を選択します。

Masterに新しいスクリプトを付ける

Hierarchy（ヒエラルキー）のMasterに、新しいスクリプトをつけます。
HierarchyのMasterを選択してInspector（インスペクター）のAdd Component（アッド・コンポーネント）→ New Script（ニュー・スクリプト）とクリックします。
スクリプトのName（ネーム：名前）を「master」としてLanguage（ランゲイジ：言語）を「C Sharp」とします。
選択したらCreate and Add（クリエイト・アンド・アッド）をクリックして作成します。



Assetsのmasterをダブルクリックします。
次のように、スクリプトを記述します。

```
using UnityEngine;
using System.Collections;

public class master : MonoBehaviour {

    public GameObject ballpf;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        if (Input.GetMouseButtonDown (0)) {
            Ray ray = Camera.main.ScreenPointToRay (Input.mousePosition);
            Vector3 dir = ray.direction.normalized;
            GameObject ball = (GameObject)Instantiate (ballpf,
                Camera.main.transform.position, Quaternion.identity);
            ball.GetComponent<Rigidbody> ().velocity = dir * 50f;
        }
    }
}
```

Hierarchy（ヒエラルキー）のMasterを選択し、AssetsのBallをInspector（インスペクター）のMaster(Script)のBallpfの枠の中へDDします。

再生ボタンを押してみましょう。クリックしてボールが飛んでいき、cubeの壁を壊すことができれば成功です。

解説 スクリプトの解説

このスクリプトを一言で説明するなら、「左クリックを押すとBallを発射する」です。
もう少し明確に説明するならこうなります。

左クリックされたら、
マウスの座標からカメラ正面を向く直線の情報を取得しrayに入れる。
rayから方向を取り出しdirに入れる。
　ボールをカメラの位置に生成する。
　ボールにdirの方向に力を加えて飛ばす。

```
public class master : MonoBehaviour {

    public GameObject ballpf;

    void Start () {

    }

    void Update () {
        if (Input.GetMouseButtonDown (0)) {
            Ray ray = Camera.main.ScreenPointToRay (Input.mousePosition);
            Vector3 dir = ray.direction.normalized;
            GameObject ball = (GameObject)Instantiate (ballpf,
                Camera.main.transform.position, Quaternion.identity);
            ball.GetComponent<Rigidbody> ().velocity = dir * 50f;
        }
    }
}
```

ballのプレハブを入れる入れ物
もし、マウスの左クリックしているのであれば、{}中カッコ内を処理する
rayに入れる
マウスの座標からカメラ正面を向く直線の情報を作成
dirに入れる
線情報から方向を取り出す
ballに入れる
ボールを生成する
ballにdirの方向に50の力を加える
if文で条件を満たしたとき、処理する範囲

解説 if（イフ）文とは

if（〇〇）{××}とはif（イフ）文と呼ばれ、条件分岐処理に使われます。
ifは、和訳すると「もし〇〇ならば」という意味になります。
（）カッコ内の条件を満たしているならば {} 中カッコ内の命令文を処理します。

解説　各関数の説明

Input.GetMouseButtonDown（インプット・ゲットマウスボタンダウン）とは
マウスボタンが押されたかどうかを判別できます。
カッコ内の数字は、どのボタンかを指定します。
０で左クリック　１で右クリック　２でホイールクリックになります。

Input.mousePosition（インプット　マウスポジション）とは
マウスの位置座標（スクリーン座標）を取得できます。

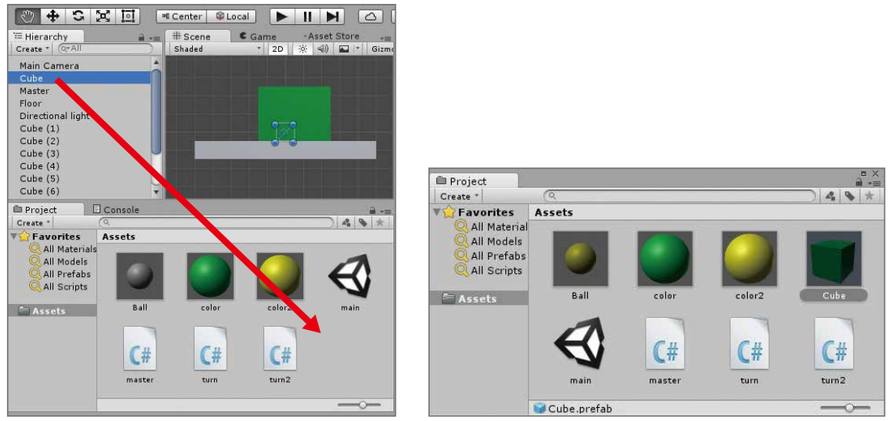
Camera.main.ScreenPointToRay（カメラ・メイン・スクリーンポイントトゥレイ）とは
スクリーン座標系の座標からカメラ方向に平行な線の情報を作ります

Instantiate（インスタンシエート）とは
オブジェクトを生成する命令です。
カッコ内には、生成するオブジェクト、生成する位置、生成したときの回転値を入れます。
今回は、Ballのプレハブであるballpf、カメラの位置、回転なしで指定しました。

ball.GetComponent<Rigidbody>().velocityとは
ballに付いているRigidbodyの速度という意味になります。velocity（ベロシティー：速度）に
方向と速度を渡すとその方向、速度で物体が動きます。

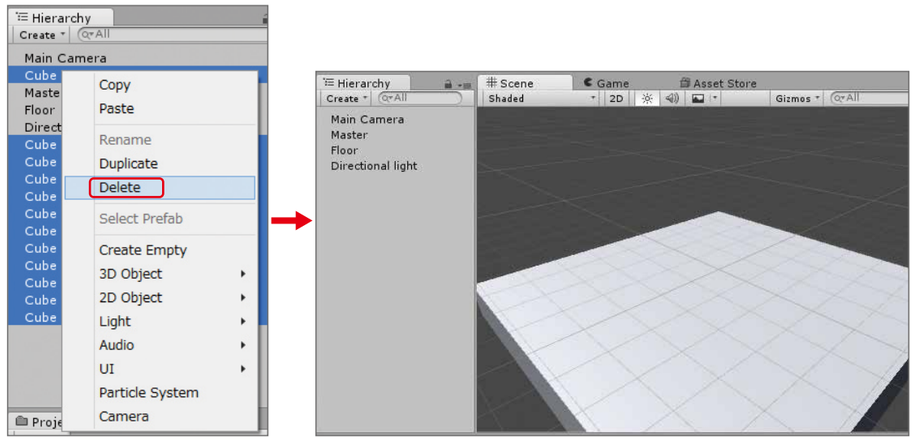
Cubeをプレハブ化する

ここまでは、Cubeを複製しPosition（ポジション：座標）を設定することで、Cubeを並べてきました。
これをスクリプトで並べるように変更します。
Ballを生成するのと同じように、Cubeをプレハブ化して、置きたい位置に生成します。
まず、Cubeをプレハブ化します。
Hierarchy（ヒエラルキー）にあるCubeを、Assets（アセット）の中へDDします。



HierarchyにあるCubeをデリートする

プレハブ化したCubeを使ってScene（シーン）にCubeを生成します。
したがって、Hierarchy（ヒエラルキー）にあるCubeをすべて削除します。
まず、Cubeを選択します。Commandを押しながら、Cube (1) ~ Cube (19)を選択します。
その後、右クリックからDelete（デリート）を選択します。



スクリプトを修正する

Assets（アセット）の中のMasterをダブルクリックして、スクリプトを開きます。
次のようにスクリプトを追加します。

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class master : MonoBehaviour {
5
6     public GameObject ballpf;
7     public GameObject cubepf;
8
9     // Use this for initialization
10    void Start () {
11        for (int i = 0; i < 5; i++) {
12            int x = i - 2;
13            Instantiate (cubepf, new Vector3 (x, 0, 0), Quaternion.identity);
14        }
15    }
16
17    // Update is called once per frame
18    void Update () {
19        if (Input.GetMouseButtonDown (0)) {
20            Ray ray = Camera.main.ScreenPointToRay (Input.mousePosition);
21            Vector3 dir = ray.direction.normalized;
22            GameObject ball = (GameObject)Instantiate (ballpf,
23                Camera.main.transform.position, Quaternion.identity);
24            ball.GetComponent<Rigidbody> ().velocity = dir * 50f;
25        }
26
27    }
28 }
29
```

解説　for（フォー）文とは

for(○○; ××; △△) { □□ } とは、for(フォー)文と呼ばれるものです。一般的に同じ処理、あるいはよく似た処理を、複数回行うために使われます。繰り返し処理、ループ処理などと呼ばれます。
カウンターを用意して、その値で何回繰り返すかを決めます。
書き方は、○○の部分に、カウンターの宣言と初期化。××には、繰り返す条件。△△は、カウンターの増え方です。
ここでは、iをカウンターとし初期値は、0 に設定、iが 5 未満の時、繰り返し処理を行い、一回処理を行うごとにiを＋ 1 するという繰り返し処理になります。

解説　変数（へんすう）とは

変数（へんすう）とは、数値や文字などを保持する入れ物のようなものです。
ここでは、ballpf、cubepf、i、xzahyo が変数に当たります。

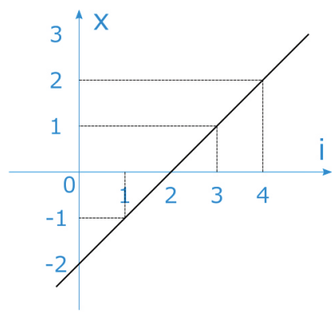
なぜ変数が必要なのか。
計算結果の数値などを複数の命令を使用したい場合に、一時的に保存するために使います。
変数とは、水（数値、文字など）を入れる容器なのです。
プログラムにおいて数値や文字は、水のようなものです。そのまま使えば何処かに流れてなくなります。しかし、容器に入れておけば無くなりません。複数の場所で使いたい場合、容器に入れて使います。

解説　変数の型（かた）とは

変数には、型と呼ばれる種類を限定するものがあります。
ここでは、 ballpf、cubepfは、GameObject（ゲームオブジェクト）型。i、xzahyoは、int（イント）型になります。
GameObject型とは、Unity特有の型でHierarchyに表示されているオブジェクトやプレハブが入ります。
int（イント）型とは、Integer（インテジャー）の略で整数を入れます。
float（フロート）とは、小数点以下が使える実数を入れる型です。

解説　スクリプトの解説

追記した文を一言で表すなら「cubeを 5 個並べて生成する」です。
int xzahyo = i – 2;
この一行は、キューブ生成する際のX座標を求める式です。
変数i は、for文によって＋ 1 ずつ加算されていきます。これを利用してCubeを並べます。



i	i-2
0	-2
1	-1
2	0
3	1
4	2

```
public class master : MonoBehaviour {

    public GameObject ballpf;
                                  cube を入れる入れ物
    public GameObject cubepf;

    vo
        for ( int i = 0;  i < 5;  i++ )
        {
            int x = i - 2;
            Instantiate (cubepf, new Vector3 (x, 0, 0) , Quaternion.identity);
        }
    }
```

カウンターの宣言と初期化
繰り返し条件 i が 5 未満
カウンターの増え方 ＋ 1 ずつ加算
変数 x を宣言し代入
x 座標を求める
キューブを生成
生成位置
for 文のループ範囲

再生ボタンを押してみましょう。Cubeが一列並んだら成功です。

高さ(Y)方向にも並べます。
スクリプトのStart関数を次のように修正します。

```
 4 public class master : MonoBehaviour {
 5
 6     public GameObject ballpf;
 7     public GameObject cubepf;
 8
 9     // Use this for initialization
10     void Start () {
11         for (int j = 0; j < 4; j++) {
12             for (int i = 0; i < 5; i++) {
13                 int x = i - 2;
14                 int y = j;
15                 Instantiate (cubepf, new Vector3 (x, y, 0), Quaternion.identity);
16             }
17         }
18     }
19
```

解説　for文の中にfor文｜二重ループ

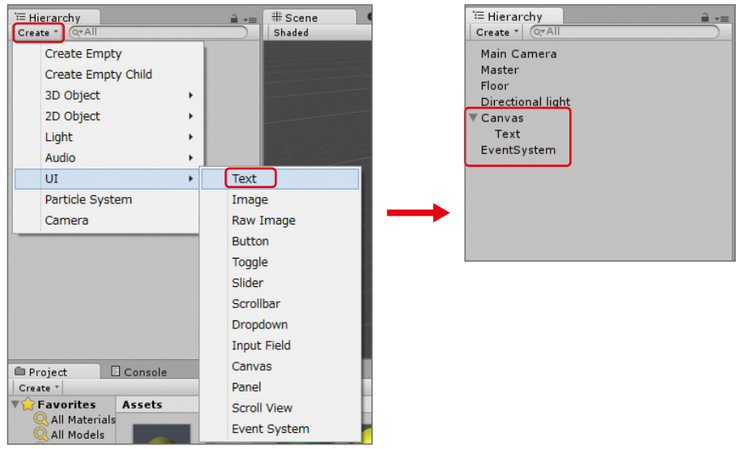
for文の{}中カッコ内にfor文を書いたものを、「二重ループ」といいます。
一つ目のfor文は、二つ目のfor文を指定回数繰り返す処理になります。
このスクリプトでは、横にCube並べて生成させるループ２をループ1で4回繰り返しています。

```
void Start () {

    for (int j = 0; j < 4; j++)
    {                                          ループ1:ループ2を4回繰り返す

        for (int i = 0; i < 5; i++)
        {                                      ループ2:キューブを5個生成

            int x = i - 2;
            int y = j;
            Instantiate (cubepf, new Vector3 (x, y, 0), Quaternion.identity);

        }
    }
}
```

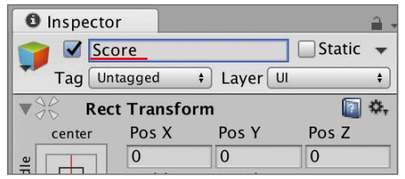
再生ボタンを押してみましょう。Cubeで壁が出来たら成功です。

スコアを表示する

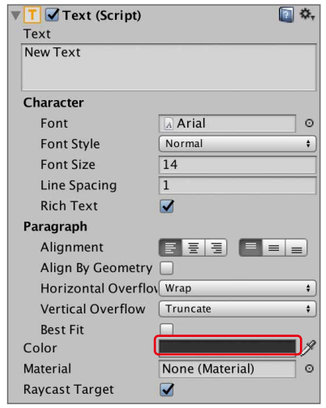
落としたCubeをスコアとして表示する機能を、これから作ります。
ますは、文字を表示するオブジェクトを用意します。
Hierarchy(ヒエラルキー)のCreate(クリエイト) → UI(ユーアイ) → Text(テキスト)を選択します。
Canvas(キャンバス)とText(テキスト)、EventSystem(イベントシステム)が生成されます。
Gameタブを押すと、New Textという黒い文字が表示されます。

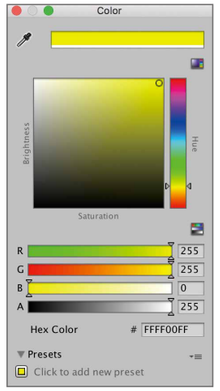


Textの名前を変えます。
HierarchyでTextを選択し、名前を「Score」とします。

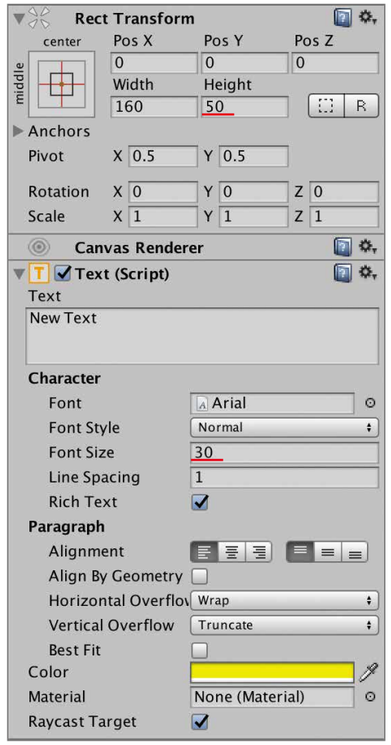


このままでは、見にくいので色と大きさを変更します。まずは、テキストの文字色を変更します。
Hierarchy(ヒエラルキー)でScoreを選択してInspector(インスペクター)のColorの枠内をクリックします。
カラーパレットで好きな色を選びます。
見やすくするのが目的のため明るく目立つ色にするとよいでしょう。

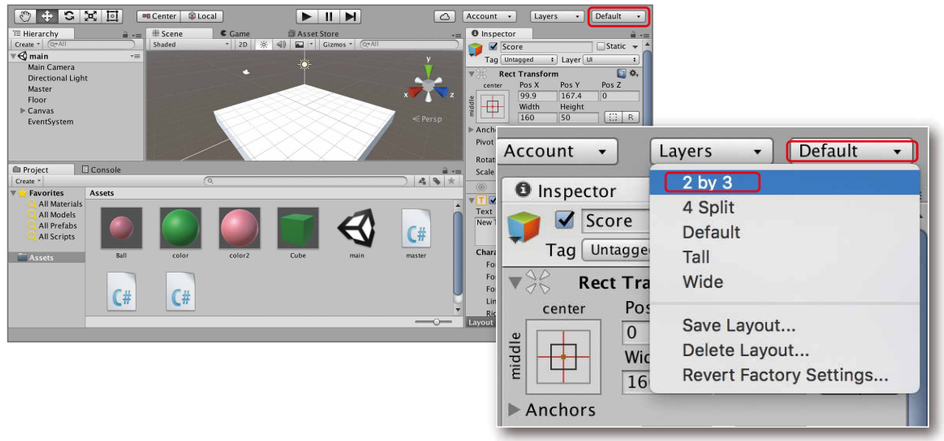




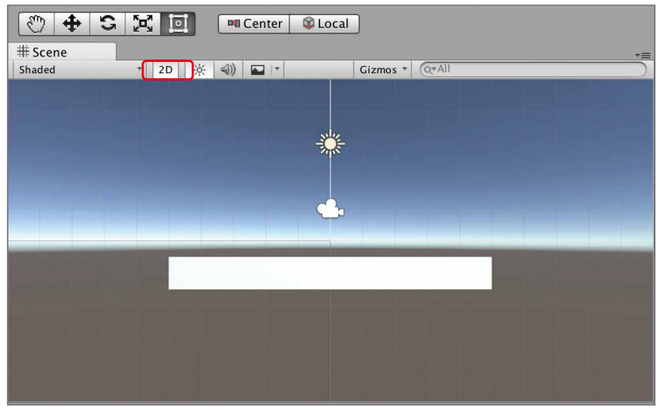
今回は、黄色にしました。
次に文字の大きさを変更します。
Hierarchy（ヒエラルキー）でScoreを選択し、Inspector（インスペクター）のRect Transform（レクト・トランスフォーム）のHeight（ハイト：高さ）を50にします。
Text(Script)のFont Sizeを30にします。



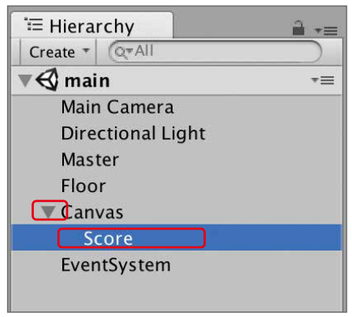
表示場所も変更します。
今回は、数値を入力する以外の方法で行います。
まずは、Unityのレイアウトを操作しやすいものに変更します。
右上のDefault（デフォルト）を押して、2 by 3（ツーバイスリー）を選択します。



すると左側に、Scene（シーン）ビューとGame（ゲーム）ビューが縦に並びます。
Sceneの2Dのボタンを押します。

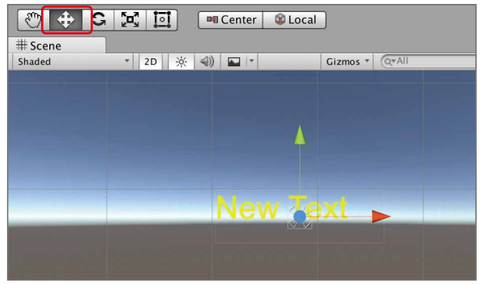


Scene（シーン）にScore（スコア）を表示させます。
Hierarchy（ヒエラルキー）のCanvas（キャンバス）の左側の右向きの三角をクリックし、下向きにします。
すると、Scoreが現れます。
このScoreをダブルクリックします。
※Hierarchyが真ん中あたりに移動しています。
※Scoreオブジェクトは、Canvasオブジェクトの子供になっています。



すると、Scene（シーン）にNew Textの文字が現れます。
※Sceneの中にオブジェクトが見えないときは、HierarchyにあるオブジェクトをダブルクリックすることでSceneに表示されます。

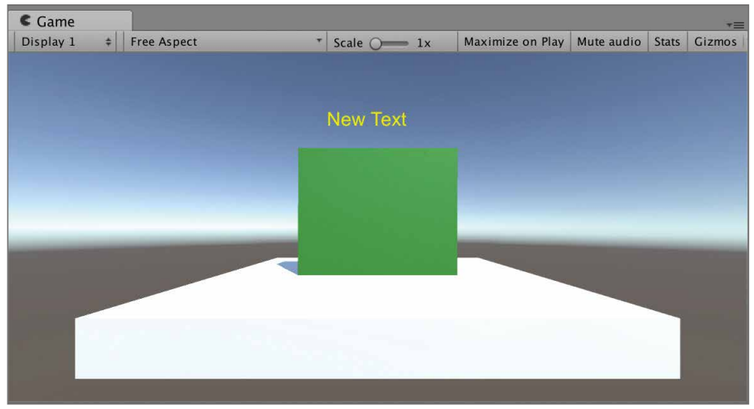
左上の十字のボタンを押します。



Sceneに出てきた矢印をクリックしたままドラッグすることで、New Textの位置（Scoreオブジェクト）の位置を調整します。

上向き矢印（Y方向）をクリックして引き上げます。
調節が終わったら、右上の2 by 3のボタンをクリックし、Defaultを選択します。

一度再生してみましょう。
下の図のようになっていれば、成功です。



文字を変更する

Cubeが落ちた数を数えてスコアとして表示するようにします。

まずは、スコアを表示する機能を作ります。
次のようにMasterを修正します。

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

public class master : MonoBehaviour {

    public GameObject ballpf;
    public GameObject cubepf;
    public static int score;
    public Text scoreText;

    // Use this for initialization
    void Start () {
        for (int j = 0; j < 4; j++) {
            for (int i = 0; i < 5; i++) {
                int x = i - 2;
                int y = j;
                Instantiate (cubepf, new Vector3 (x, y, 0), Quaternion.identity);
            }
        }
    }

    // Update is called once per frame
    void Update () {
        if (Input.GetMouseButtonDown (0)) {
            Ray ray = Camera.main.ScreenPointToRay (Input.mousePosition);
            Vector3 dir = ray.direction.normalized;
            GameObject ball = (GameObject)Instantiate (ballpf,
                Camera.main.transform.position, Quaternion.identity);
            ball.GetComponent<Rigidbody> ().velocity = dir * 50f;
        }

        scoreText.text = "Score:" + score.ToString ();

    }
}
```

入力が終えたらCommand＋Sで保存してUnityに戻ります。
masterにScoreオブジェクトを設定します。
Hierarchy（ヒエラルキー）のMasterを選択します。Inspector（インスペクター）のMaster（Script）に
Score Text（スコア・テキスト）の枠が増えています。
Score Textの枠にHierarchyのScoreをDDします。

再生してみましょう。
「New Text」が「Score：0」になっていれば成功です。
ただし、Cubeを落としても変化がありません。次は、落ちたCubeを数えて反映させる機能を作ります。

解説 スクリプトの解説

Scoreオブジェクトを扱う変数scoreTextを用意してInspectorでScoreオブジェクトを設定します。こうするとscoreTextを通じてScoreオブジェクトを扱うことができます。

scoreText.textに表示したい文字を代入すると変更できます。イコール（＝）で代入できます。
今回は、「Score:」とscoreをToString()で変換した文字を一つの文章にして代入しています。

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

public class mast
    public GameObject ballpf;
    public GameObject cubepf;

    public static int score;

    public Text scoreText;

    // Use this for initialization
    void Start () {
        for (int j = 0; j < 4; j++) {
            for (int i = 0; i < 5; i++) {
                int x = i - 2;
                int y = j;
                Instantiate (cubepf, new Vector3 (x, y, 0), Quaternion.identity);
            }
        }
    }

    // Update is called once per frame
    void Update () {
        if (Input.GetMouseButtonDown (0)) {
            Ray ray = Camera.main.ScreenPointToRay (Input.mousePosition);
            Vector3 dir = ray.direction.normalized;
            GameObject ball = (GameObject)Instantiate (ballpf,
                Camera.main.transform.position, Quaternion.identity);
            ball.GetComponent<Rigidbody> ()
                                                    50f;
        }

        scoreText.text = "Score:" + score.ToString ();

    }
}
```

Text 型を使用するのに必要

スコアを記録する変数

Score オブジェクトを扱う Text

文字を作成して表示

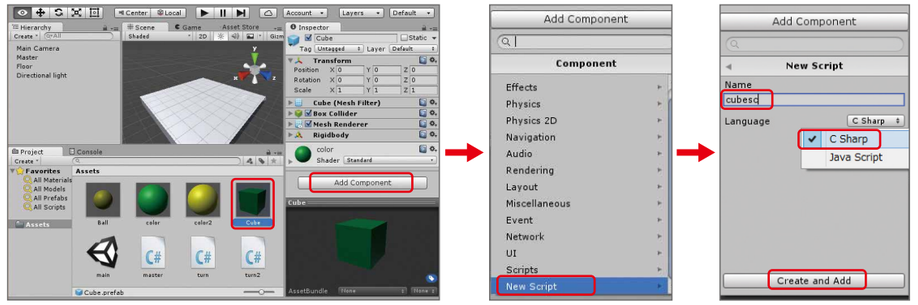
解説 ToString（トゥストリング）とは

ToStringとは、変数が持つデータを文字に変換し取得する命令です。変数の情報は、変化することはありません。イコール（＝）で左辺の変数に代入するなどして使います。
今回は、int型であるscoreの数値を文字に変換しtextに代入しています。

落ちたCubeを数える

今回は、Cubeが画面外に出たら落ちたということにして数えます。
まずは、Cubeにスクリプトを付けます。

Assets（アセット）のCubeをクリックします。
Inspector（インスペクター）の下のほうのAdd Component（アッド・コンポーネント）をクリックし、New Script（ニュー・スクリプト）をクリックします。
Name（ネーム：名前）を、「cubesc」としてLanguage（ランゲイジ：言語）は、「C Sharp」を選択します。
最後に、Create and Add（クリエイト・アンド・アッド）をクリックします。



Assetsの中のcubescをダブルクリックしてスクリプトを編集します。
次のように、記述してください。

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class cubesc : MonoBehaviour {
5
6      // Use this for initialization
7      void Start () {
8
9      }
10
11     // Update is called once per frame
12     void Update () {
13
14     }
15
16     void OnBecameInvisible() {
17         Destroy (gameObject);
18         master.score++;
19     }
20
21 }
22
```

入力を終えたらCommand＋Sで保存します。
再生して見ましょう。Cubeを落とした時、「Score：0」が 1 加算されれば成功です。

解説 OnBecameInvisible（オンビケームインビジブル）とは

OnBecameInvisibleとは、目に見えなくなったという意味です。
中カッコ内に書いた命令は、カメラから見えなくなったときに処理されます。

解説 Destroy（デストロイ）とは

Destroy（デストロイ）とは、和訳すると「破壊」という意味です。
カッコ内に指定したオブジェクトを削除します。
また、数秒後に削除するという命令も可能です。
,（カンマ）で区切って二つ目に数値を指定すると、指定した秒数後に削除という命令になります。

解説 gameObjectとは

最初のｇが小文字であるgameObjectは、このスクリプトが付いているオブジェクトのことを指します。
今回のcubescは、Cubeオブジェクトに付けてあるのでCubeオブジェクトのことになります。

解説 スクリプトの解説

Cubeが画面外に出た時の処理を書きました。処理内容は、二つ。
自分自身を削除する処理とmasterが持っているscoreに１加算する処理です。

ボールを削除する

Ballも削除するようにします。
AssetsのBallにスクリプトを付けます。手順は、今までと同じです。
スクリプト名は、「ballsc」とします。
ballscを次のように記述してください。

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

再生してボールを打ってみましょう。5秒後にHierarchy(ヒエラルキー)からBall(clone)が消えたら成功です。

ボールの個数を制限する

Assete（アセット）の中のmasterをダブルクリックして開きます。
次のように修正します。

```
 5 public class master : MonoBehaviour {
 6
 7     public GameObject ballpf;
 8     public GameObject cubepf;
 9     public static int score;
10     public Text scoreText;
11     public int count = 3;
12
13     // Use this for initialization
14     void Start () {
15         for (int j = 0; j < 4; j++) {
16             for (int i = 0; i < 5; i++) {
17                 int x = i - 2;
18                 int y = j;
19                 Instantiate (cubepf, new Vector3 (x, y, 0), Quaternion.identity);
20             }
21         }
22     }
23
24     // Update is called once per frame
25     void Update () {
26         if (Input.GetMouseButtonDown (0) && count > 0) {
27             Ray ray = Camera.main.ScreenPointToRay (Input.mousePosition);
28             Vector3 dir = ray.direction.normalized;
29             GameObject ball = (GameObject)Instantiate (ballpf,
30                 Camera.main.transform.position, Quaternion.identity);
31             ball.GetComponent<Rigidbody> ().velocity = dir * 50f;
32             count--;
33         }
34
35         scoreText.text = "Score:" + score.ToString ();
36
37     }
38 }
39
```

記述したらCommand＋Sでセーブします。

再生してみましょう。ボールが３発しか発射できないなら成功です。

比較演算子（ひかくえんざんし）

比較演算子とは、変数の値や数値を比較する演算子（記号）のことです。
if文と組み合わせると、比較の結果から処理を変えることができます。
条件が成立したときは「真（true）」、成立しないときは「偽（false）」といいます。

演算子	意味	使い方	使い方の意味
==	＝（等しい）	a==b	aとbは等しい
!=	≠（等しくない）	a!=b	aとbは等しくない
<	＜（小なり）	a<b	aはbよりも小さい値
>	＞（大なり）	a>b	aはbよりも大きい値
<=	≦（以下）	a<=b	aはb以下
>=	≧（以上）	a>=b	aはb以上

論理演算子（ろんりえんざんし）

複数の条件を組み合わせて使うのが論理演算子です。
２つ以上の条件が成り立ったときやどちらかが成り立ったときといった表現が可能になります。

演算子	意味	使い方	使い方の意味
&&	両方が真	(a>=1)　&&　(a<=2)	aが1以上かつaが2以下
||	片方が真	(a>=1)　||　(b>=1)	aが1以上またはbが1以上
!	真ではない	!(a==1)	aは1ではない

解説　スクリプトの解説

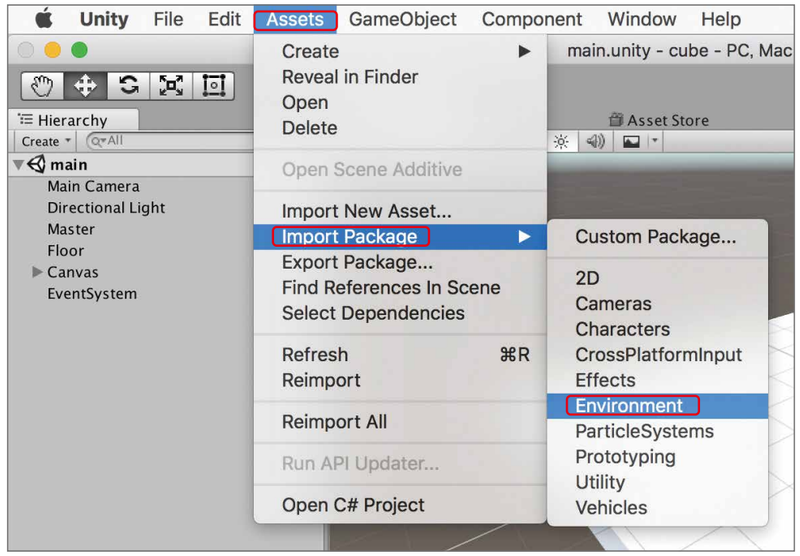
ボールの残数をmasterに管理させています。変数countがボールの残数になります。
if文の一文の意味は、今まで「左クリックした時」でしたが、「&&」と「count > 0」を付け加えることによって「左クリックした時且つ、残数が０より多い時」になりました。
if文が成立した時のみ処理させる{}中カッコ内に「count--」は、その変数の値を１減らす処理です。

```
public class master : MonoBehaviour {

    public GameObject ballpf;
    public GameObject cubepf;
    public static int score;
    public Text scoreText;
    public int count = 3;

    // Use this for initialization
    void Start () {
        for (int j = 0; j < 4; j++) {
            for (int i = 0; i < 5; i++) {
                int x = i - 2;
                int y = j;
                Instantiate (cubepf, new Vector3 (x, y, 0), Quaternion.identity);
            }
        }
    }

    // Update is called once per frame
    void Update () {
        if (Input.GetMouseButtonDown (0) && count > 0) {

            Ray ray = Camera.main.ScreenPointToRay (Input.mousePosition);
            Vector3 dir = ray.direction.normalized;
            GameObject ball = (GameObject)Instantiate (ballpf,
                Camera.main.transform.position, Quaternion.identity);
            ball.GetComponent<Rigidbody> ().velocity = dir * 50f;

            count--;
        }

        scoreT                ore:" + score.ToString ();

    }
}
```

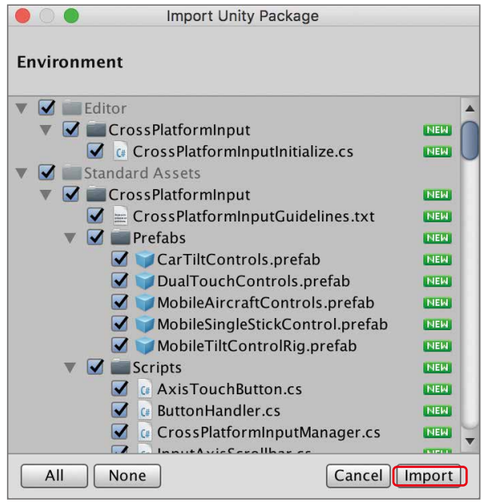
ボールの残数
左クリックした時且つ残りが０より多い時
残数を1減らす

水に浮かぶ島にする

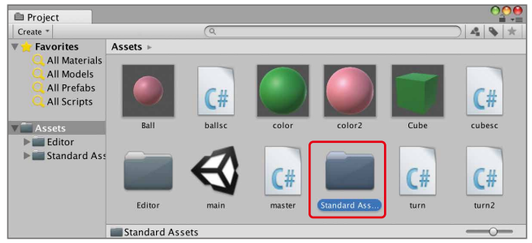
次は見た目を少し変えましょう。
一番上のメニューのAssets（アセット）　→　Import　Package（インポート・パッケージ）　→　Environment（エンバイロンメント：環境）を選択します。

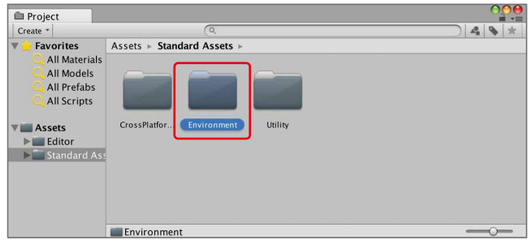


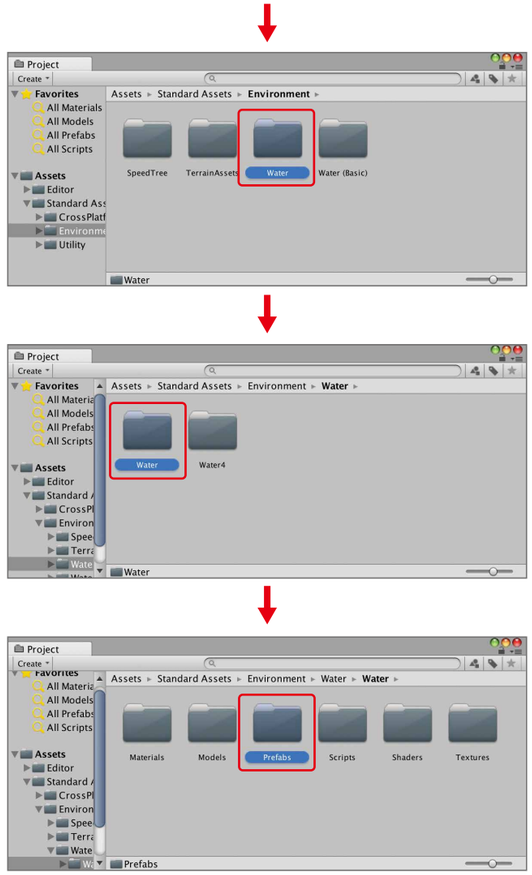
出てきたウインドウのImport（インポート）をクリックします。



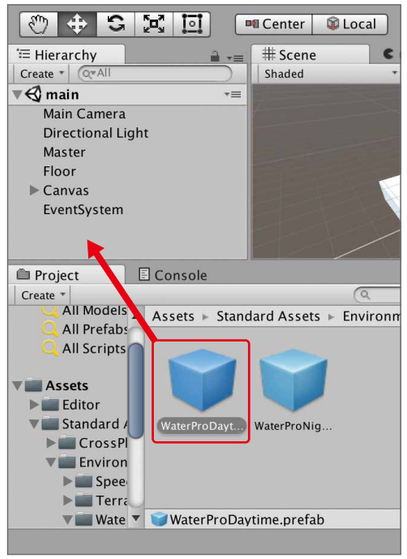
Project（プロジェクト）のAssets（アセット）にあるフォルダを以下の順でダブルクリックします。
　Standard Assets（スタンダード・アセット） → Environment（エンバイロンメント：環境） → Water（ウォーター） → Water（ウォーター） → Prefabs（プレハブ）



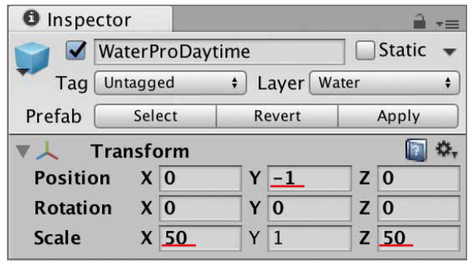




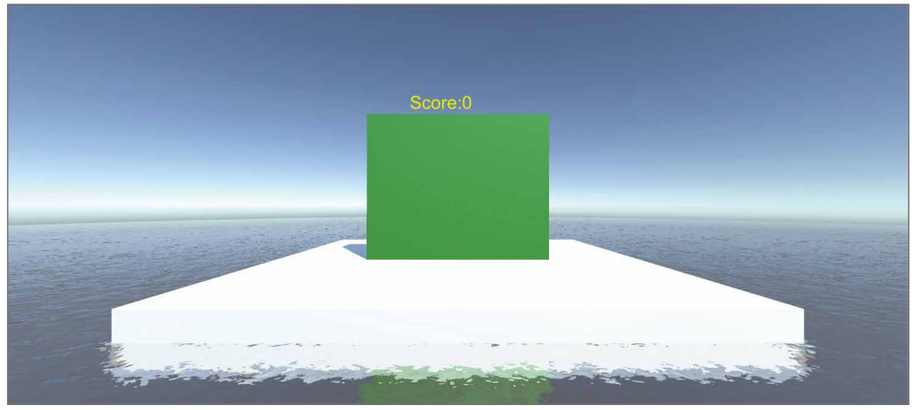
Prefabsの中のWaterProDaytime（ウォーター・プロ・デイタイム）をHierarchyにDDします。



Hierarchy（ヒエラルキー）でWaterProDaytime（ウォーター・プロ・デイタイム）を選択し、Transform（トランスフォーム）を以下のように設定します。

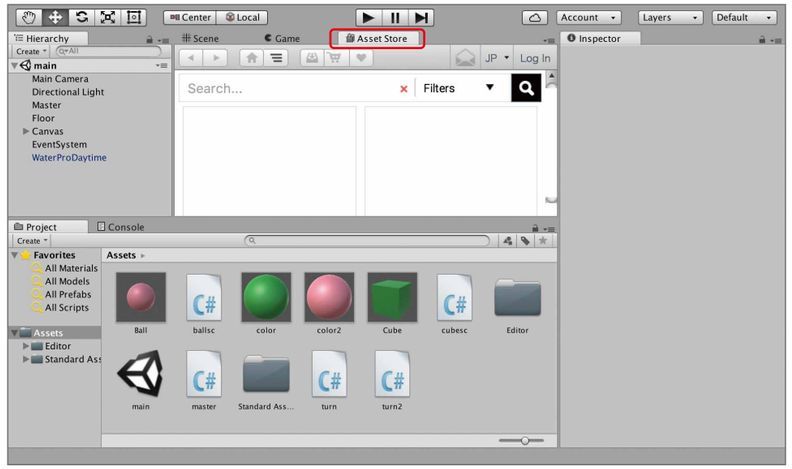


再生してみましょう。次のようになっていれば成功です。

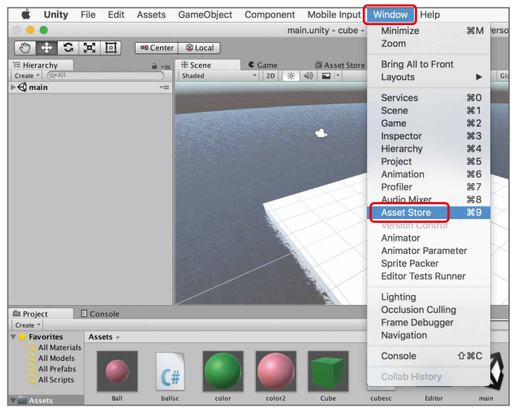


スカイボックスを設定する

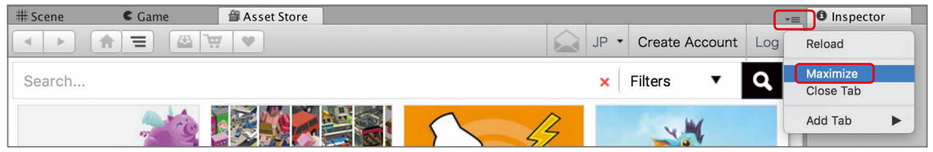
Asset Store（アセット・ストア）を開きます。
SceneとGameに並んでいるAsset Store（アセットストア）のタブをクリックします。



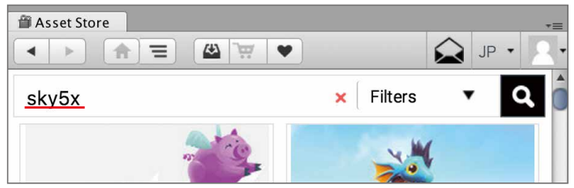
タブがない場合は、一番上のメニューのWindow → Asset Storeを選択します。



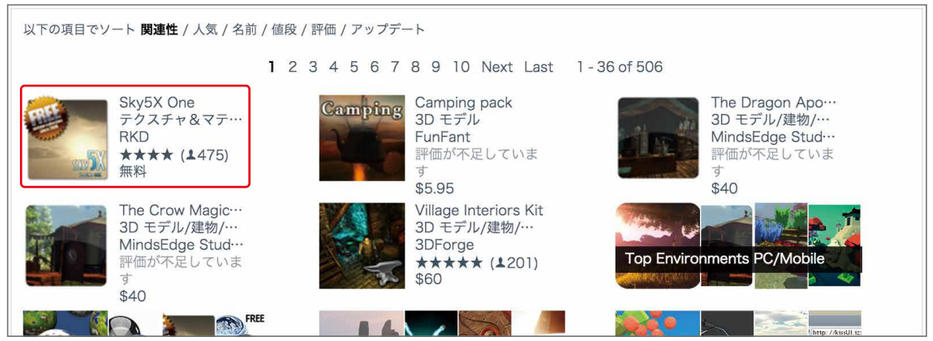
アセットストアの画面が表示されます。
ウインドウが小さくて見にくいときは、右上のメニューをクリックし、Maximizeをクリックします。



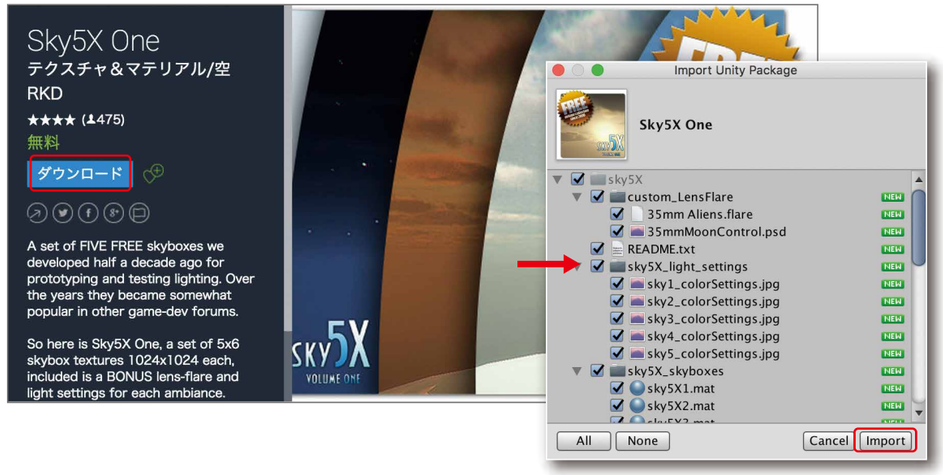
Asset Storeの一番上の検索欄に、「sky5x」と入力してエンターキーを押します。



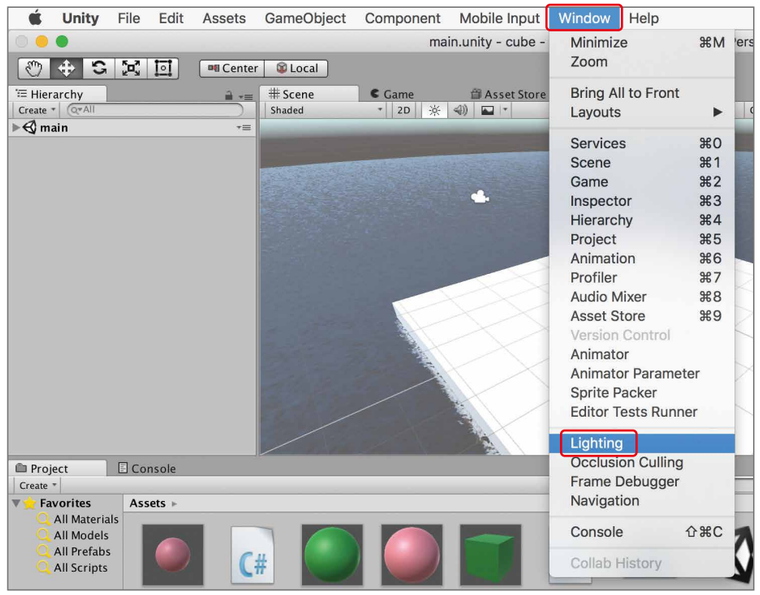
検索結果から「Shy5X One」を探しクリックします。



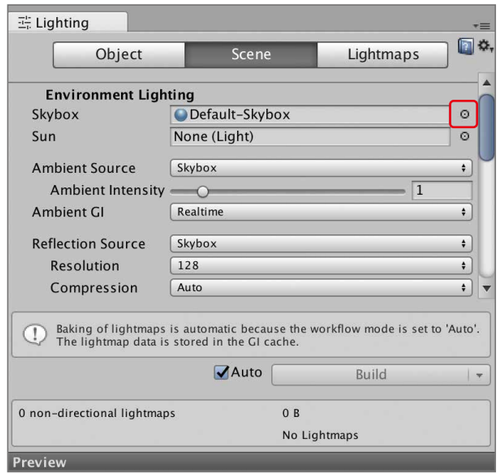
次に、ダウンロードボタンを押します。
出てきたウインドウのImport（インポート）ボタンを押します。



次に、Gameタブをクリックし、ゲーム画面を表示します。
Window（ウィンドウ）→ Lighting（ライティング：光）を選択します。



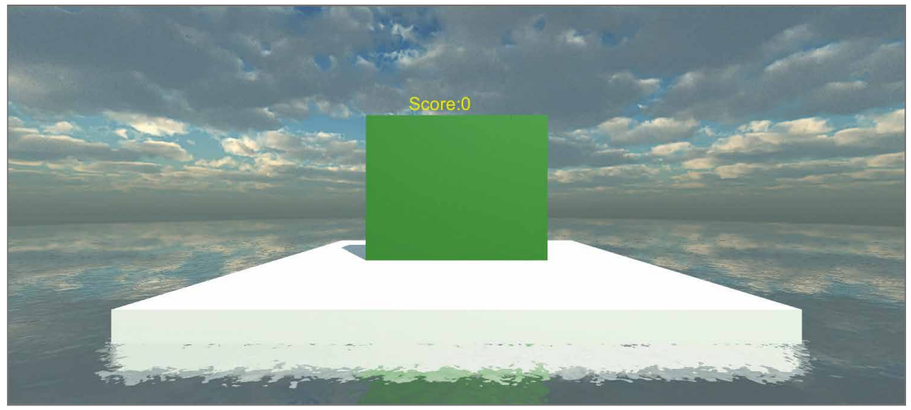
Lighting（ライティング）のウインドウが現れます。
Skybox（スカイボックス）の一番右側の小さな丸印をクリックします。



ウインドウが出てきます。
好みでsky5x1～sky5x5のいずれかを選択します。
また、右上のスライドバーを動かすとアイコンのサイズが変わります。

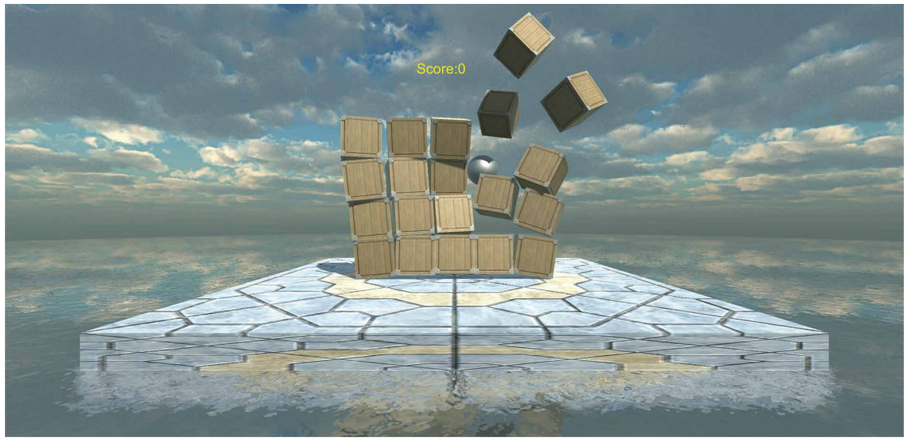
ここでは、sky5x2を選択しました。

再生してみましょう。背景が変わっていれば成功です。



memo

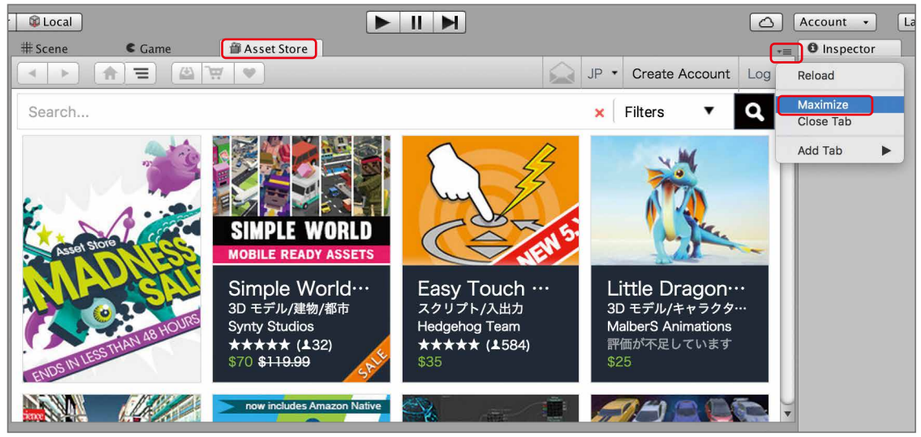
Cube番外編



P50　ボールの発射音を鳴らす
P56　破壊音を入れる
P58　ボールを金属っぽくする
P59　Cubeに模様をつける

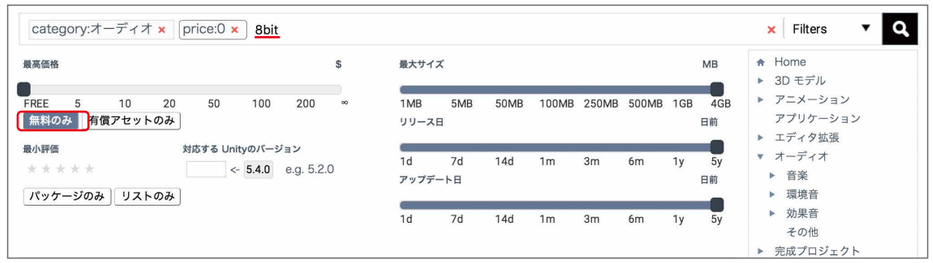
ボールの発射音を鳴らす

Asset Store（アセット・ストア）から効果音をダウンロードしてインポートします。
Asset Store（アセット・ストア）のタブをクリックします。
右上のメニューをクリックしてMaximizeを選択します。



右側のカテゴリー欄のオーディオをクリックします。

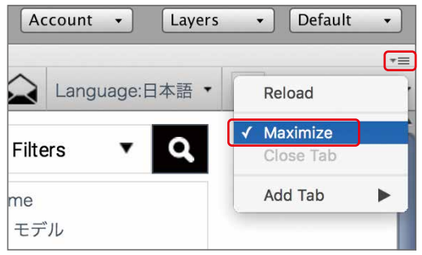
無料のみをクリックし、検索欄に「8bit」と入力して検索してください。



検索結果から「8-bit Sound Free Package」を探してクリックします。

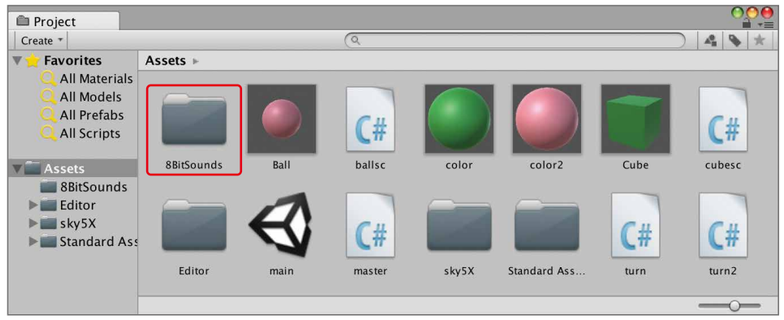
次に、ダウンロードボタンを押します。
出てきたウインドウのImportボタンを押します。

Asset Store（アセット・ストア）を小さな画面にもどします。
右上のシンボルをクリックし、Maximizeをもう一度選択します。

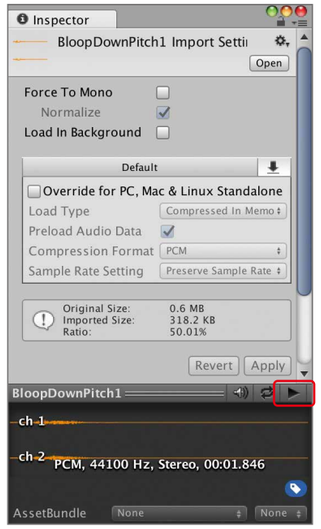


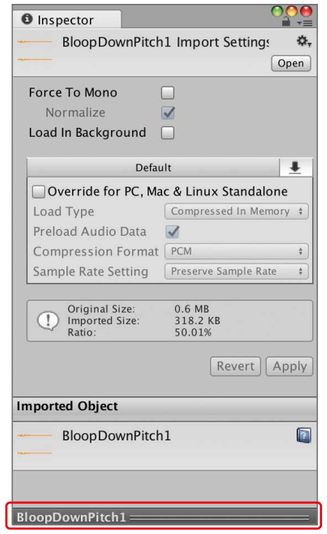
Assets（アセット）の中に8BitSounds（８ビットサウンド）が入っていれば、インポート成功です。
Sceneタブをクリックして、タブを戻しておきます。

8BitSoundsをダブルクリックして開きます。

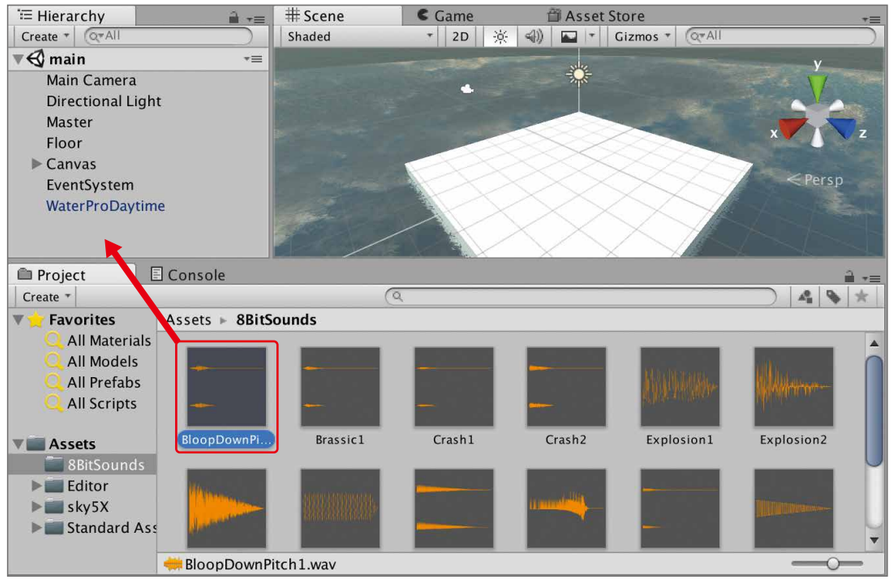


音ファイルが何種類か入っています。
音ファイルを一つ選択するとInspector（インスペクター）に情報が出てきます。Inspector（インスペクター）の下のほうにある再生ボタンを押すと、音が再生されます。
また、Inspector（インスペクター）下に黒いウインドウがない場合、一番下の黒いバーをクリックすることで出てきます。

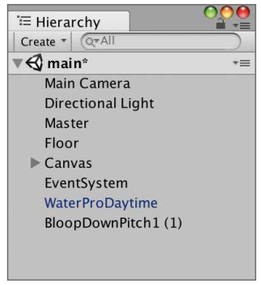




ボールの発射音に好きな音を選択してください。
ここでは、BloopDownPitch1を選択しました。
BloopDownPitch1をHierarchyにDDします。

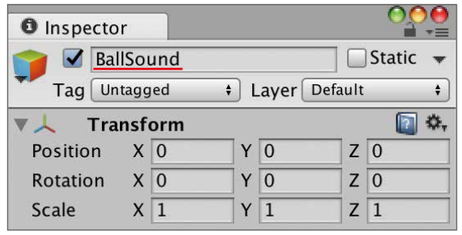


Hierarchyに、BloopDownPitch1 (1)が生成されています。

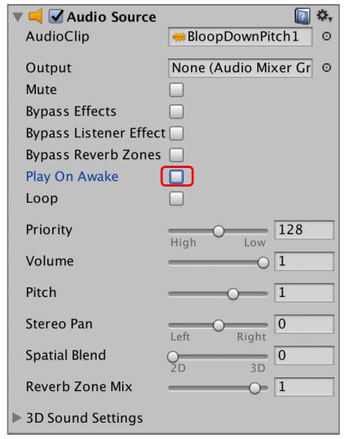


これは、音ファイルそのものではありません。音ファイルが設定されたオブジェクトをUnityが自動的に作ってくれたものです。

名前を変更します。
BloopDownPitch (1)を選択して、Inspector（インスペクター）で名前を「BallSound」とします。最後にエンターキーを押すのを忘れないようにしてください。



Play On Awake（プレイ・オン・アウェイク：起動時に再生）のチェックを外します。



解説　音を鳴らすオブジェクトと音ファイル

Unityでは、音のファイルのことをAudio　Clip（オーディオクリップ）といい、音を鳴らすコンポーネント（構成要素）は、Audio Source（オーディオソース：音源）です。
Audio　Sourceは、設定されたAudio　Clipを再生します。Inspectorに表示されているAudio Cripの欄に再生したい音データを設定するだけで再生する音が変更できます。

今回DDしたのは、プレハブではなく、Audio Cripです。
ProjectにあるAudio CripをHierarchyにDDすると、Audio Cripが設定されたAudio Sourceが付いたオブジェクトが自動的に作られます。

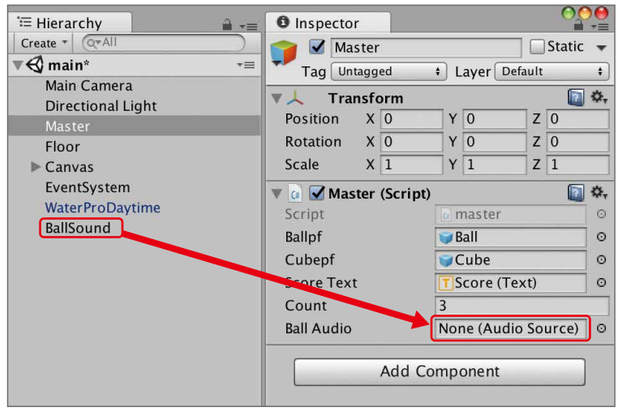
Play On Awakeにチェックを入れると、再生時に自動的に音がなるようになります。最初からずっと流し続けるBGMの場合、Play On Awakeにチェックを入れておけばプログラムを書く必要がありません。

次は、ボールが発射された時、音を鳴らすようにします。
Masterを修正します。

```
public class master : MonoBehaviour {

    public GameObject ballpf;
    public GameObject cubepf;
    public static int score;
    public Text scoreText;
    public int count = 3;
    public AudioSource ballAudio;

    // Use this for initialization
    void Start () {
        for (int j = 0; j < 4; j++) {
            for (int i = 0; i < 5; i++) {
                int x = i - 2;
                int y = j;
                Instantiate (cubepf, new Vector3 (x, y, 0), Quaternion.identity);
            }
        }
    }

    // Update is called once per frame
    void Update () {
        if (Input.GetMouseButtonDown (0) && count > 0 ) {

            Ray ray = Camera.main.ScreenPointToRay (Input.mousePosition);
            Vector3 dir = ray.direction.normalized;
            GameObject ball = (GameObject)Instantiate (ballpf,
                Camera.main.transform.position, Quaternion.identity);
            ball.GetComponent<Rigidbody> ().velocity = dir * 50f;

            count--;
            ballAudio.Play ();

        }

        scoreText.text = "Score:" + score.ToString ();

    }
}

```

入力したらCommand＋Sでセーブし、Unityに戻ります。

Hierarchy（ヒエラルキー）からMasterを選択し、Inspector（インスペクター）のBallAudioの欄にHierarchyのBallSoundをDDします。



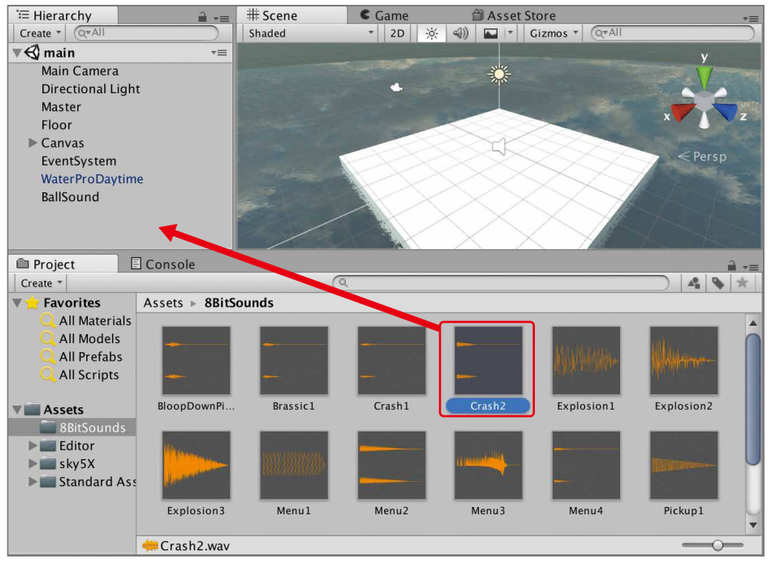
再生してみましょう。ボールを発射したときに、音が出たら成功です。

解説 スクリプトの解説

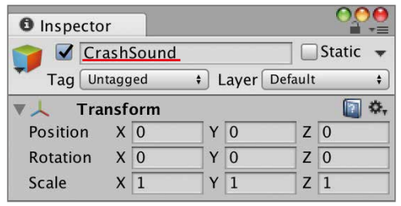
AudioSource型の変数ballAudioは、BallSoundオブジェクトを入れるためのものです。
publicが書いてあるので、Inspectorから設定できます。設定すると、この変数を通じてBallSoundオブジェクトを操作するとこができます。
ballAudio.Play()は、音を再生する命令です。AudioClipに設定してある音を再生します。

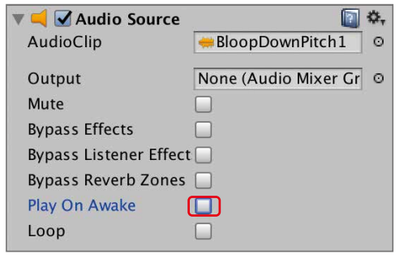
破壊音を入れる

Assets（アセット）の中の8BitSoundsにあるCrash2(クラッシュ2)を、HierarchyにDDします。



このオブジェクトの名前を、「CrashSound」にし、Play On Awake（プレイ・オン・アウェイク）のチェックを外します。





この音は、Ballが物にぶつかった時になる音です。
ballscを次のように書き換えます。

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class ballsc : MonoBehaviour {
5
6      AudioSource crashAudio;
7
8      // Use this for initialization
9      void Start () {
10         Destroy (gameObject, 5f);
11
12         GameObject clasObj = GameObject.Find ("CrashSound");
13         crashAudio = clasObj.GetComponent<AudioSource> ();
14     }
15
16
17     // Update is called once per frame
18     void Update () {
19
20     }
21
22     void OnBecameInvisible() {
23         Destroy (gameObject);
24         master.score++;
25     }
26
27     void OnCollisionEnter(Collision c) {
28         crashAudio.Play ();
29     }
30 }
31
```

解説 スクリプトの解説

物にぶつかった時、音を鳴らす機能を書き加えました。
AudioSource型の変数crashAudioは、HierarchyのCrashSoundオブジェクトを扱うものです。
今回は、publicを付けておらず、Inspectorで設定することができません。なぜなら、このスクリプトが付いているBallは、プレハブだからです。プレハブは、Hierarchyのオブジェクトを Inspectorに設定することができないからです。
今回は、Ballが生成された時、CrashSoundオブジェクトを探して設定する処理を書きました。その命令文は、Startの {} 中カッコ内です。
Startは、Unityが再生された時、もしくは、このオブジェクトが生成された時に処理されます。その特性を利用して、生成された瞬間にCrashSoundを探しだし、crashAudioに入れています。

あとは、OnCollisionEnter関数内で音を再生しています。
OnCollisionEnter関数の {} 中カッコ内でPlayを書けば、ぶつかった時に音を再生できます。

解説 GameObject.Find（ファインド）とは

Faindとは、和訳すると「探す」という意味になります。
GameObject.Faind関数は、Hierarchyの中から指定された名前のオブジェクトを探し取得する命令です。
イコール（＝）で変数に渡すことが出ます。

書き方は、以下の通りです。
GameObject.Faind("探すオブジェクト名")

解説 GetComponent（ゲットコンポーネント）とは

GetComponentとは、和訳すると「構成要素を追加する」という意味になります。
コンポーネント（構成要素）とは、オブジェクトに付けることのできるもの、RigidbodyやAudioSource、自分で作ったスクリプトなどのことです。
GetComponent関数は、オブジェクトに付いている指定したコンポーネントを取得することができます。
イコール（＝）で変数に渡すことが出ます。

書き方は、以下の通りです。
GameObject型変数.GetComponent<取得したいコンポーネント名>()

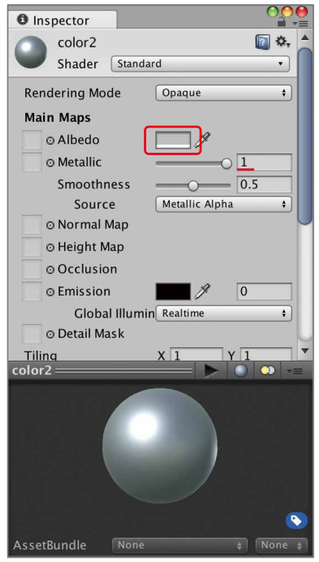
解説 OnCollisionEnter（オンコリジョンエンター）とは

何かに衝突した時に処理される部分です。
処理される条件は、自身と衝突した相手にCollisionが付いていて尚且つどちらかにRigidbodyが付いていることです。
カッコ内に定義してある c には、衝突したオブジェクトのデータが自動で入り、それをもとに衝突したオブジェクトを判別できます。
似た命令にOnCollision　とOnCollisionExitがあります。これらは、処理するタイミングが違います。タイミングは、以下の通りです。

OnCollisionEnter	衝突した瞬間
OnCollision	接触している間
OnCollisionExit	離れた瞬間

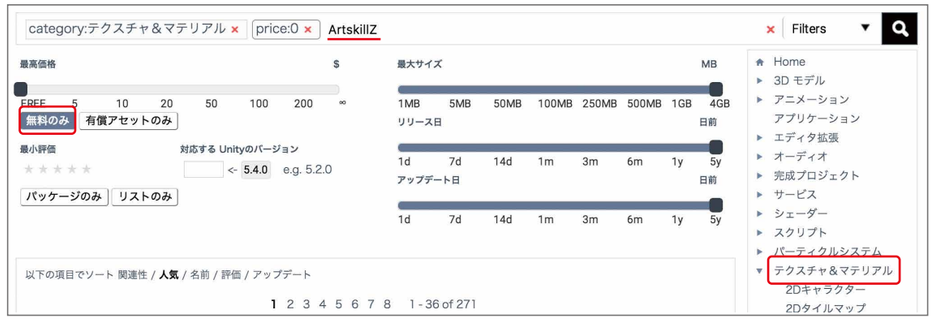
ボールを金属っぽくする

Project（プロジェクト）のAssets（アセット）の中のcolor2を選択します。
Inspector（インスペクター）のAlbedo（アルベド）を薄い灰色にし、Metallic（メタリック）を 1 にします。



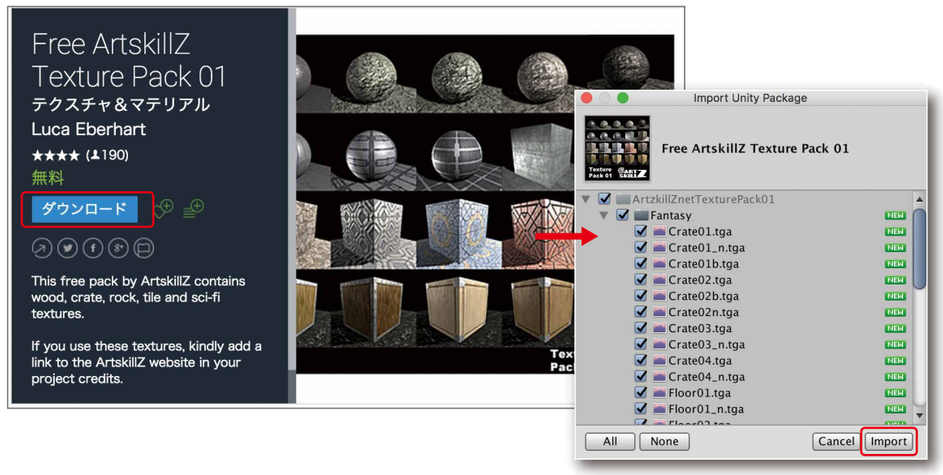
cubeに模様を付ける

まずは、素材をダウンロードしてインポートします。
Asset Store（アセット・ストア）を開きます。
右側のメニューのテクスチャ&マテリアル、無料のみをクリックし、「ArtskillZ」と入力して検索します。



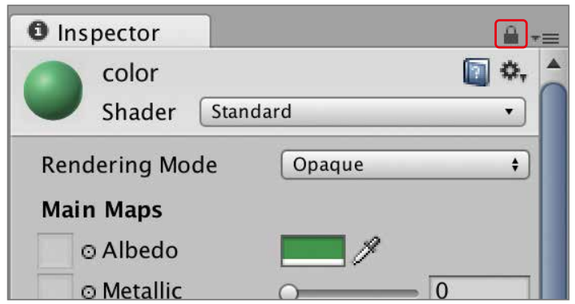
検索結果から「Free ArtskillZ Texture Pack 01」を探してクリックします。

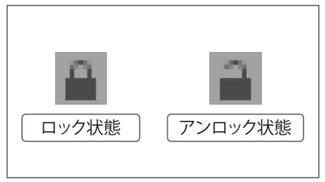
ダウンロードボタンを押して、出てきたウインドウのImportを押します。



Assetsの中にArtzkillZnetTexturePack01というフォルダがあればインポート成功です。

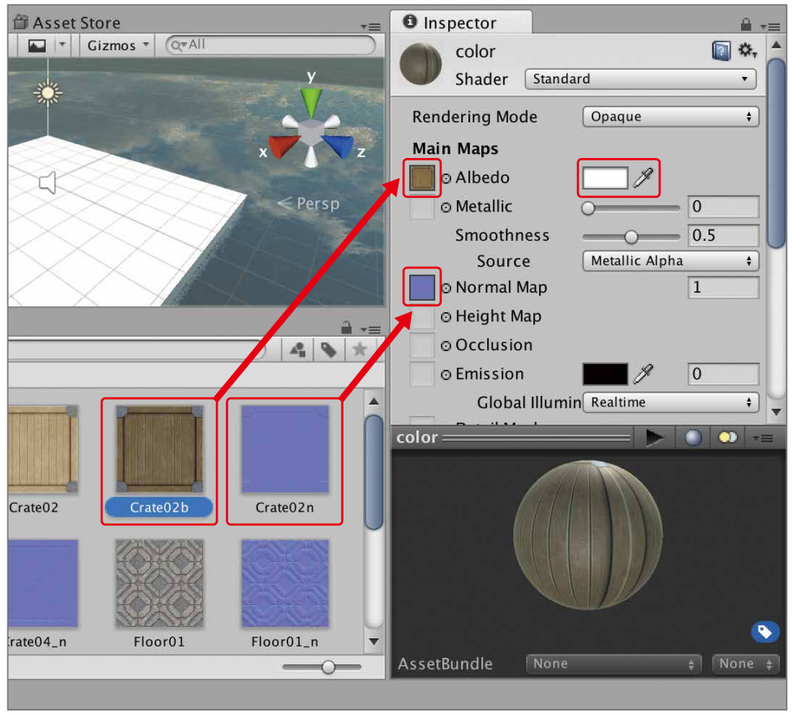
Assetsの中にあるcolorを選択します。
Inspector（インスペクター）の右上、南京錠マークをクリックし、ロック状態にします。
ロック状態にすることで、表示を維持します。他のオブジェクトを選択しても変わらなくなります。





これから模様を付けます。
colorをロックした状態で、ArtzkillZnetTexturePack01をクリックし、Fantasy（ファンタジー）、Rocks（ロックス：岩）、Scifi（サイフィ：SF）のうち好きなフォルダを開き好きな模様を決めてください。
教材では、「Crate02b」にしました。

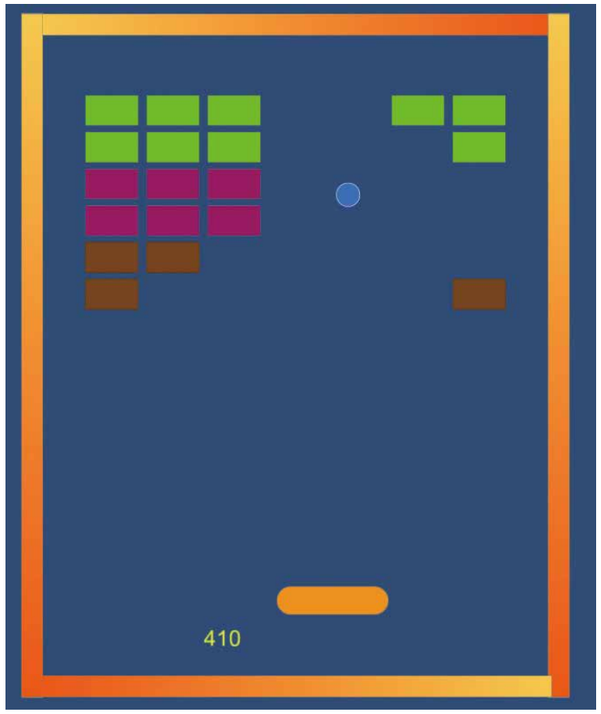
選んだ画像をAldedoにDDし、色を白にします。
選んだ画像に似た名前で最後にnが付いている青っぽい画像をNormalMapにDDします。
教材では、「Crate02n」のことです。



★課題
（1）Floorにも、模様を設定してみましょう。
　　Project → Create → Materialでcolor3を作ってやってみましょう。

memo

Block編



P63　準備する
P65　一枚の絵を複数の絵として扱う
P67　壁を作る
P69　バーを作る
P70　Barを操作するスクリプトを作る
P72　ボールを作る
P74　Physics 2D Materialを設定する
P75　BallにTag（タグ）を付ける
P76　ブロックを作る
P78　マスターを作ってブロックを並べる
P81　Wall4にボールが当たったらミスにする
P82　スコアを表示する
P86　GameOverの文字を出す
P88　リプレイできるようにする
P90　効果音を付ける

準備する

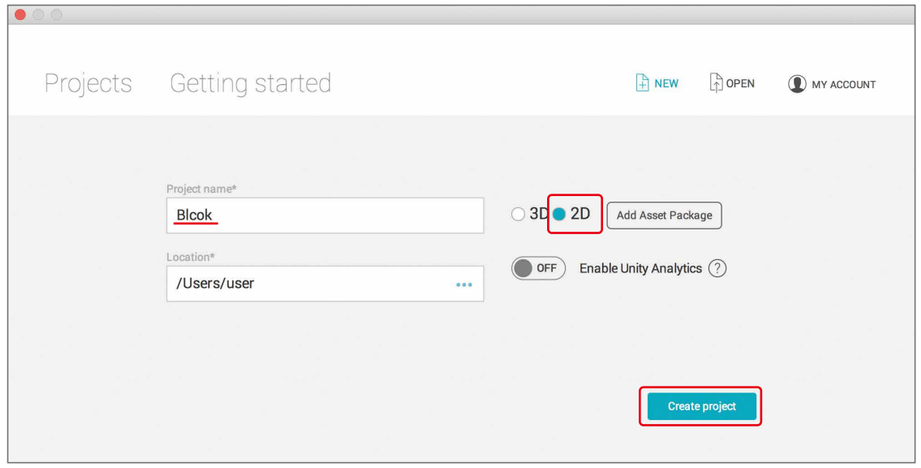
素材となる絵をダウンロードします。
ネット上に素材を用意しましたので、ダウンロードします。
下のアイコンから、Safariをクリックして起動します。

上の検索枠に、「gameprograming.jp/block.png」と打ち込みエンターキーを押します。
すると、次のような絵が出てきます。

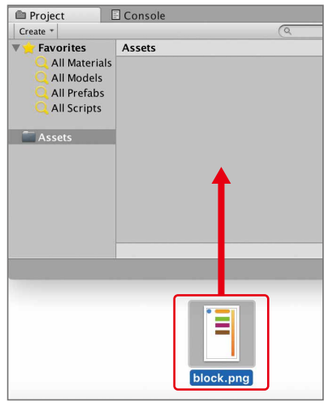
画像の上で右クリックをし、イメージを別名で保存…を選択。

場所をデスクトップに変更して保存します。

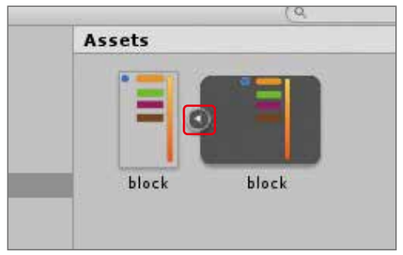
Unityを起動して、新しいプロジェクトを作ります。
Project name（プロジェクト・ネーム）に「Block」と入力し、２Ｄにチェックを入れます。その後Create project（クリエイト・プロジェクト）をクリックします。



unityに素材を追加します。
デスクトップのblock.pngをAssetsの中へDDします。



Assets（アセット）の中にblockのファイルが出来ていれば成功です。
右側の矢印マークを押してみます。

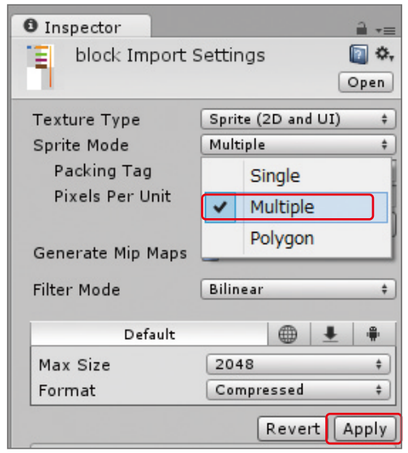


これは、一枚の絵として扱っていることを示しています。

これを切り取って複数の絵として扱うように設定します。

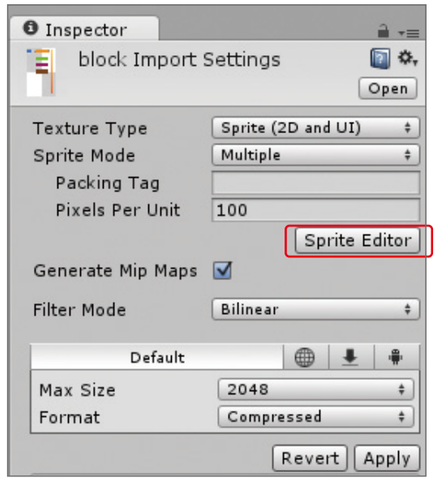
一枚の絵を複数の絵として扱う

Assetsの中のblockを選択し、InspectorのSprite　Mode（スプライトモード）をMultiple（マルチプル）にします。その後、Apply（アプライ：適用）ボタンを押し、Sprite　Editor（スプライト・エディター）ボタンを押します。

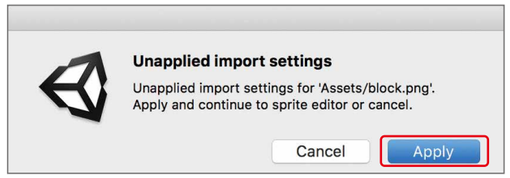


解説 スプライト　モード　マルチプル
Sprite Mode　Multipleとは

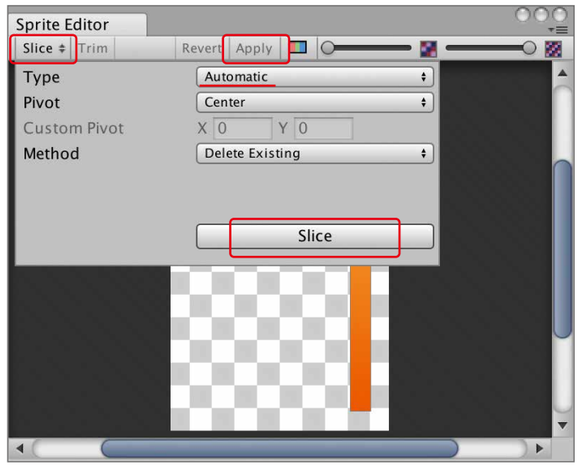
Multiple（マルチプル）とは、日本語で「複数」という意味です。
一つの画像を切り抜いて複数のパーツとして扱うモードです。



Unapplied import settings というウインドウが出てくる場合があります。Applyを押します。

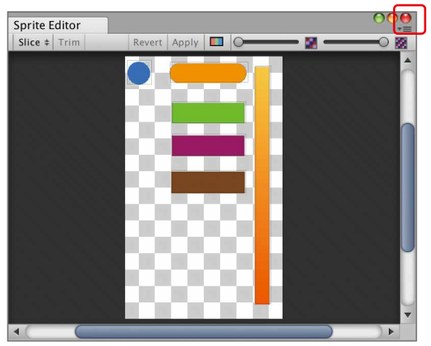


Sprite Editorのウインドウが出てきます。
左上のSlice（スライス：切り身）ボタンを押します。
TypeがAutomatic（オートマチック：自動的）であることを確認し、Sliceボタンを押し、上部のApply
を押します。



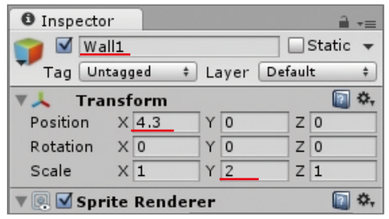
これによって、一枚の絵から複数枚の部品が切り出されます。
Assets（アセッツ）の中の右側の三角アイコンを押して、次のようになれば成功です。

Sprite Editorのウインドウは、もう必要ないので右上のバツボタンで閉じます。

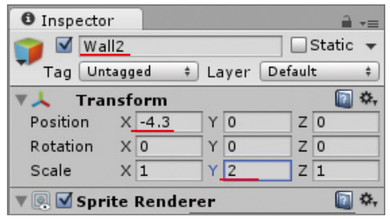


壁を作る

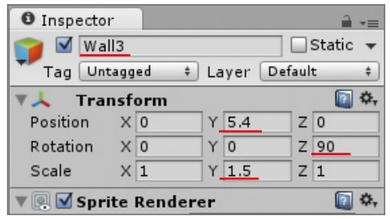
○ Wall1を作ります。
block_2をHierarchy（ヒエラルキー：階層）へDDします。
block_2の名前を「Wall1」とし、位置を調整します。
PositionのXを4.3とし、ScaleのYを2とします。



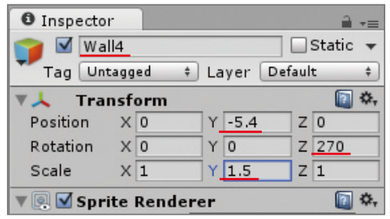
○ Wall2を作ります。
block_2をHierarchyへDDします。
block_2の名前を「Wall2」とし、位置を調整します。
PositionのXを-4.3とし、ScaleのYを2とします。



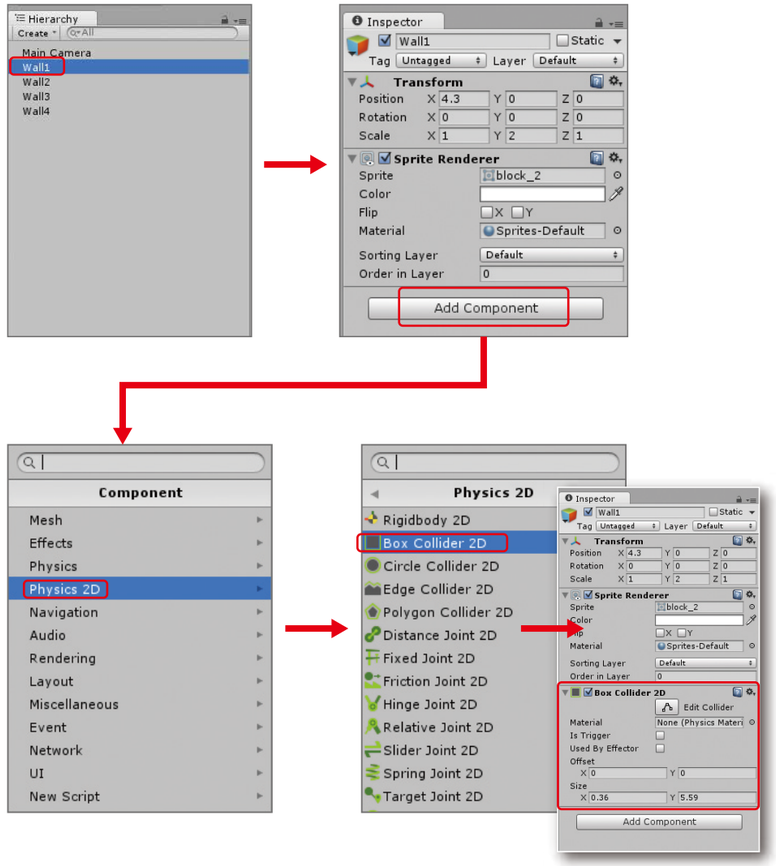
○ Wall3を作ります。
block_2をHierarchyへDDします。
block_2の名前を「Wall3」とし、位置を調整します。
PositionのYを5.4とし、RotationのZを90とし、ScaleのYを1.5とします。



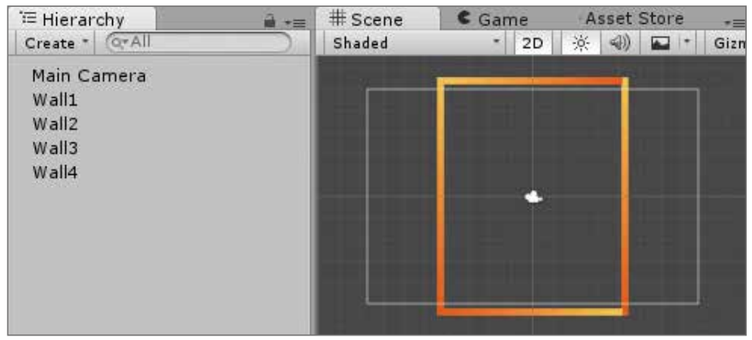
○ Wall4を作ります。
block_2をHierarchyへDDします。
block_2の名前を「Wall4」とし、位置を調整します。
PositionのYを-5.4とし、RotationのZを270とし、ScaleのYを1.5とします。



○Wall1～4にBox Collider2D（ボックス・コライダー2D）を付けます。
Hierarchy（ヒエラルキー）でWall1を選択し、Inspector（インスペクター）のAdd Component（アッド・コンポーネント）→ Physics2D（フィジックス2D）→ Box Collider2Dとクリックします。

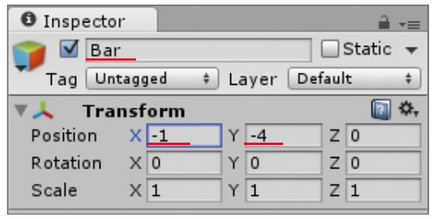


同様に、Wall2～Wall4にも、Box Collider2Dを付けます。
※HierarchyでWall1～Wall4を同時に選択し、一気にBox Collider2Dを付けることもできます。



バーを作る

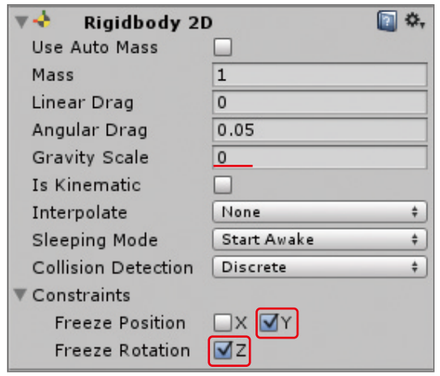
○ボールを跳ね返すバーを作ります。
block_1を、Hierarchy（ヒエラルキー：階層）にDDします。
block_1の名前を、「Bar」に変更します。
BarのPositionのXを-1、Yを-4とします。



課題
（1）BarにBox Collider2Dを付けましょう。
（2）BarにRigidbody2Dを付けましょう。

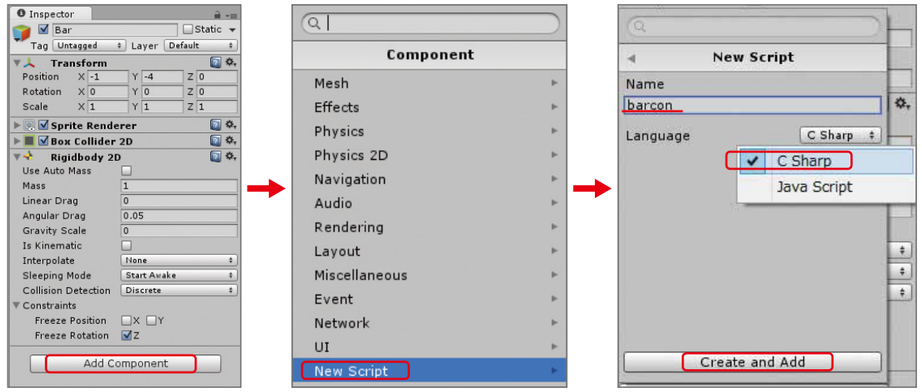
再生してみましょう。Barが下へ落ちたら成功です。

○Rigidbody 2D（リジッドボディ2D）を設定します。
Hierarchy（ヒエラルキー）でBarを選択し、Inspector（インスペクター）のRigidbody2Dを設定します。
Gravity Scale（グラビティ スケール：重力の大きさ）を0にします。
Constraints（コンストレイン：制約）の左側の三角マークをクリックし、Freeze Position Y と Freeze Rotation Zにチェックを入れます。
これにより、重力によるY座標の増減とZ軸まわりの回転が規制されます。Freeze（フリーズ）は、凍る、じっと動かないでいる、という意味です。



Barを操作するスクリプトを作る

○Barにスクリプトを付けます。
Hierarchy（ヒエラルキー）でBarを選択し、Inspector（インスペクター）のAdd　Component（アッド・コンポーネント）をクリック、New Script（ニュー・スクリプト）を選択します。
名前を「barcon」とし、言語は「CSharp」とします。



Assetsの中のbarconをダブルクリックします。
barconを以下のように記述します。記述したら、Command＋Sでセーブします。

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class barcon : MonoBehaviour {
5
6     public float speed = 10;
7
8     // Use this for initialization
9     void Start () {
10
11    }
12
13    // Update is called once per frame
14    void Update () {
15        float x = Input.GetAxisRaw ("Horizontal");
16        GetComponent<Rigidbody2D> ().velocity = new Vector2 (x * speed * Time.deltaTime, 0);
17    }
18 }
19
```

再生してみましょう。Barが左右に動き、Wallにぶつかれば成功です。
遅いと感じたら、再生と止めてInspector（インスペクター）のspeed（スピード）を調節してください。

解説 タイム　デルタタイム
Time.deltaTimeとは

Time.deltaTimeとは、1フレームを処理するのにかかった時間です。
使い方は、１秒当たりＮメートル動かしたいというときに使います。
この値とスピードをかけることによって、秒速Ｎメートル動く処理を作ることが出来ます。

解説 インプット　ゲットアクシス ロー
Input.GetAxisRawとは

Input.GetAxisRawは、ゲームパットのスティックもしくは、キーボードの矢印キーの入力を取得するのに使います。カッコ内に"Horizontal"と書くと左右の入力を取得でき、"Vertical"と書くと上下の入力を取得できます。

+1, 0, -1 のいずれかの値が取れます。Horizontalの場合、右に入力した時は+1が取得でき、入力していない時は0が取得でき、左に入力した時は-1が取得できます。
よく似た命令にInput.GetAxisがあります。こっちは、ゲームパットのスティックの倒し具合によって+1～-1の値が取れます。Input.GetAxisをキーボード操作で使うと、いきなり+1や-1とならず、0から徐々に+1や-1に近づいていきます。

解説 スクリプトの解説

左右の入力方向に秒速speedメートルで移動するスクリプトです。
speedとTime.deltaTimeをかけることによって秒速speedメートルの値にします。
ｘは、右入力の時は+1で、左力の時は-1、入力なしの時は0です。秒速speedメートルの値にかけることによって、左右入力は、それぞれ＋とーの値にして、入力なしの時は、0にしています。

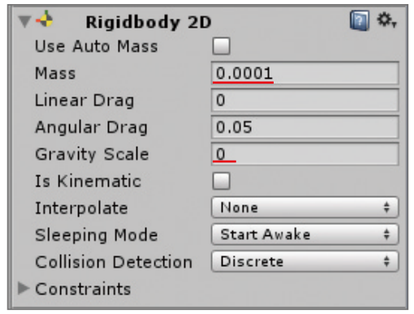
```
public class barcon : MonoBehaviour {

    public float speed = 10;   // Bar の速度　単位は秒速

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        float x = Input.GetAxisRaw ("Horizontal");   // ＋１、０、－１のいずれか / 左右の入力を取得

        GetComponent<Rigidbody2D> ().velocity = new Vector2 (x * speed * Time.deltaTime, 0);   // 入力があった場合その方向に秒速 speed メートルで移動
    }
}
```

ボールを作る

○block_0をHierarchy（ヒエラルキー）へDDします。
block_0の名前を、「Ball」に変更します。
BallのScale（スケール：大きさ）のX を0.7、Yを0.7とします。

課題
（１）BallにCircle Collider 2Dを付けましょう。
（２）BallにRigidbody 2Dを付けましょう。

○ボールの調整をします。
Hierarchy（ヒエラルキー）でBallを選択し、Rigidbody　2DのMass（マス：重量）を0とし、エンターキーを押します。
※Massは、0.0001となります。
Gravity Scale（グラビティ スケール：重力の大きさ）を、0とします。



○ボールにスクリプトをつけます。
HierarchyでBallを選択し、InspectorのAdd Component → New Scriptとクリックします。
名前を「ballcon」とし、言語を「C Sharp」とし、Create and Add をクリックします。

Assetsの中のballconをクリックしてスクリプトを開きます。
次のようにスクリプトを記述します。

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class ballcon : MonoBehaviour {
5
6     public float speed = 2f;
7
8     // Use this for initialization
9     void Start () {
10        GetComponent<Rigidbody2D> ().AddForce (new Vector2(speed, speed) * 0.01f);
11    }
12
13    // Update is called once per frame
14    void Update () {
15
16    }
17 }
18
```

記述したら、Command + Sでセーブします。

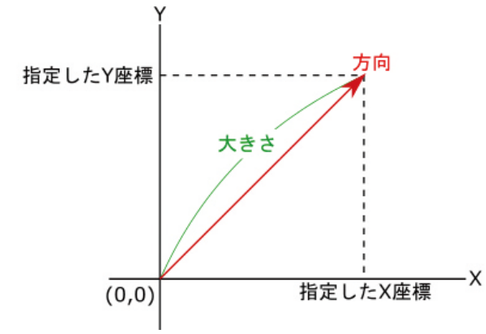
再生してみましょう。ボールが飛んでいきますが、動きがおかしいです。次は、これを改善します。

解説 AddForce（アドフォース）とは

AddForce（アドフォース）はRigidbody2Dが持っている命令文で、和訳すると「力を加える」という意味です。カッコ内で指定した方向に力を加えます。

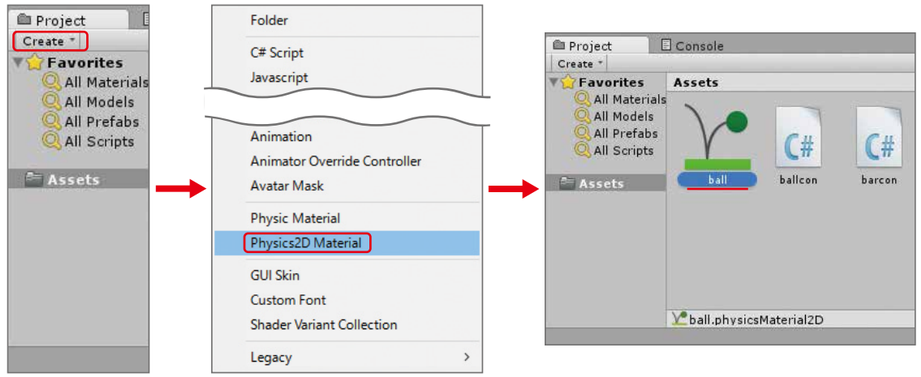
解説 Vector2（ベクターツー）とは

Vecter2（ベクターツー）とは、数学で出てくるベクトルを扱う型です。
ベクトルとは、方向と大きさを持つもので、座標(0, 0)の点から指定した点まで矢印を引きます。その矢印の向きが方向で長さが大きさを表します。

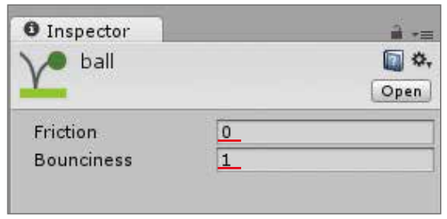


Physics 2D Materialを設定する

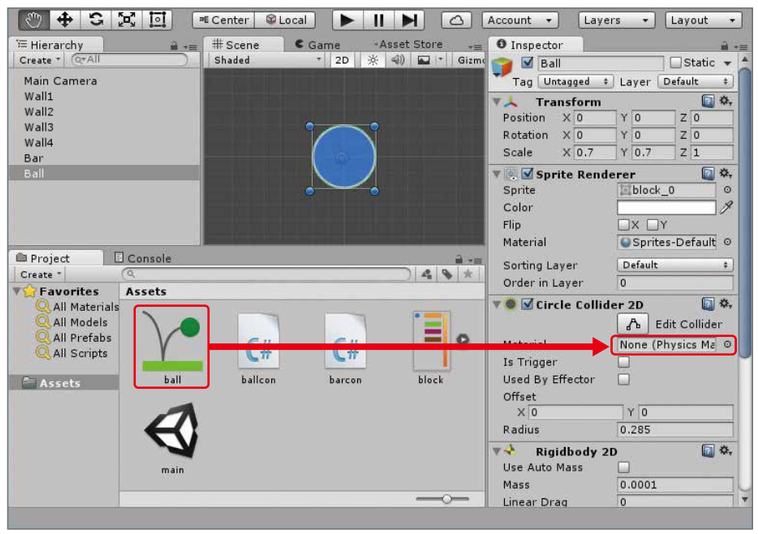
Project（プロジェクト）のCreate → Physics2D Materialの順でクリックします。
名前が付けられる状態となっているので、「ball」とします。



Assetsのballをクリックしし、InspectorのFriction（フリクション：摩擦）を0とし、
Bounciness（バウンシネス：跳ね返り）を1とします。



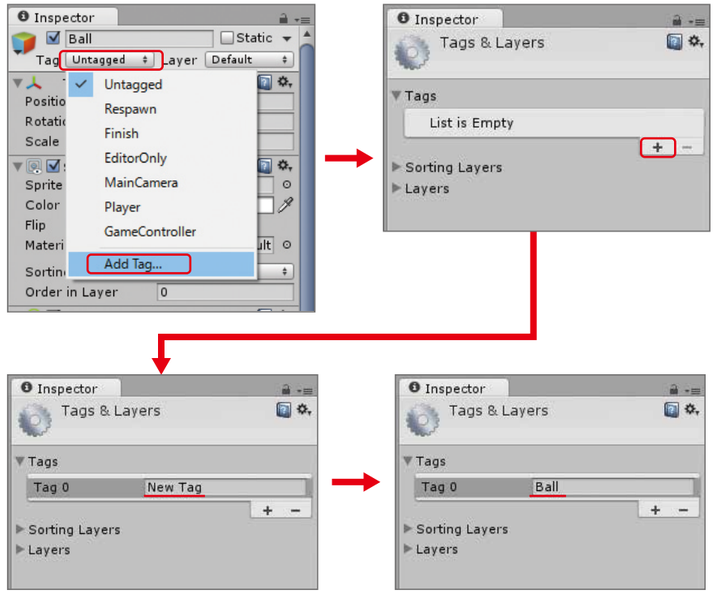
HierarchyでBallを選択し、Circle Collider 2DのMaterialの枠の中に、マテリアルballをセットします。



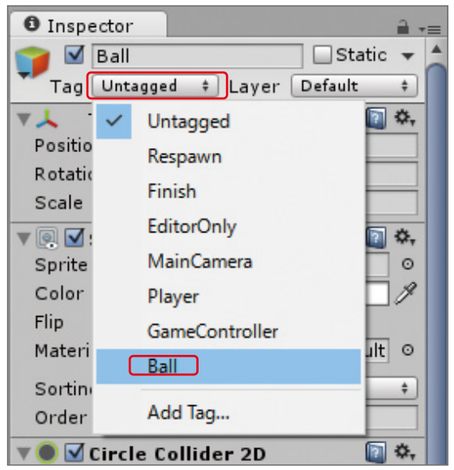
再生ボタンを押してみましょう。壁でボールが跳ね返れば成功です。

BallにTag（タグ）を付ける

この後、ブロックを作りますが、準備をします。
○Ball（ボール）にTag（タグ：名札）を設定します。
InspectorのTagの横のUntagged（アンタグド：タグなし）をクリックします。
一番下のAdd Tag（アドタグ：タグ追加）を選択します。
Tagsの＋マークをクリックします。
New Tagを、「Ball」と書き換えます。
※BallのBは大文字としてください。

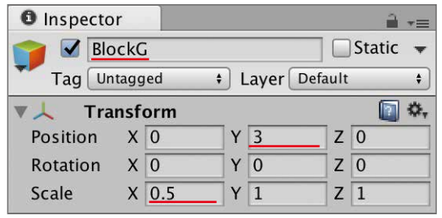


HierarchyでBallを選択し、Inspectorの横のUntaggedをクリックします。
Ballをクリックします。
これで、Ballオブジェクトに、Tag（タグ：名札）がBallとなります。



ブロックを作る

ボールがぶつかると壊れるブロックを作ります。
block_3をHierarchy（ヒエラルキー：階層）にDDします。
名前を「BlockG」とします。
Position（ポジション）のYを3とします。
Scale（スケール）のXを0.5とします。



BlockGに、Box Collider 2Dを付けます。

○BlockGにスクリプトを付ける
スクリプトの名前は、「blocksc」とします。
blockscを開いて、次のように記述します。

```
using UnityEngine;
using System.Collections;

public class blocksc : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }

    void OnCollisionEnter2D(Collision2D col) {
        if (col.gameObject.tag == "Ball") {
            Destroy (gameObject);
        }
    }
}
```

再生してみましょう。ボールが当たってブロックが消えたら成功です。

解説 スクリプトの解説

衝突したオブジェクトのTagが"Ball"であった場合、自分を削除するスクリプトです。ここでいう自分とは、このスクリプトが付いているオブジェクトのことを指します。
衝突したオブジェクトがBallかどうかを判断するには、Tagを使います。Tagは、gameObject.tagで取得できます。

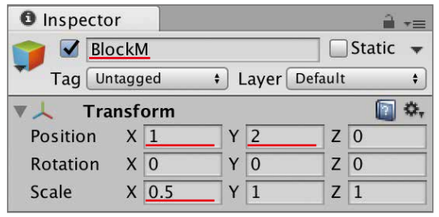
解説 オンコリジョンエンターツーディー OnCollisionEnter2Dとは

何かに衝突した時に処理される部分です。
処理される条件は、自身と衝突した相手にCollision2Dが付いていて尚且つどちらかにRigidbody2Dが付いていることです。
カッコ内に定義してある col には、衝突したオブジェクトのデータが自動で入り、それをもとに衝突したオブジェクトを判別できます。
似た命令にOnCollision2D とOnCollisionExit2Dがあります。これらは、処理するタイミングが違います。タイミングは、以下の通りです。

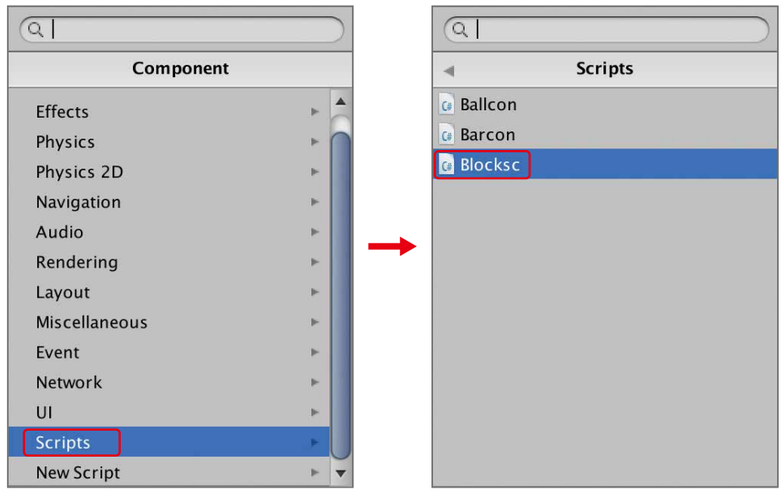
OnCollisionEnter2D	衝突した瞬間
OnCollision2D	接触している間
OnCollisionExit2D	離れた瞬間

○BlockGをプレハブ化します。
Hierarchy（ヒエラルキー）のBlockGをAssets（アセット）の中へDDします。

○色違いのブロックを作ります。
Block_4をHierarchy（ヒエラルキー）にDDします。
名前を、「BlockM」とします。
PositionのXを 1 とし、Yを2とします。
ScaleのXを0.5とします。



BlockMに、Box Collider 2Dをつけます。
BlockMにスクリプトblockscを付けます（以下アタッチという）。
InspectorのAdd Component → Scriptsの順にクリックし、blockscを選択します。
BlockMをプレハブ化します。



★課題
（1）色違いのブロックをあと一つ作ります。
　block_5をHierarchyにDDし、名前を「BlockB」としてください。
　ScaleのXを0.5とし、PositionのYを1.5とします。
　Box Collider2Dとスクリプトblockscをアタッチしてください。
　プレハブ化してください。

再生して、ボールが当たってブロックが消えたら成功です。

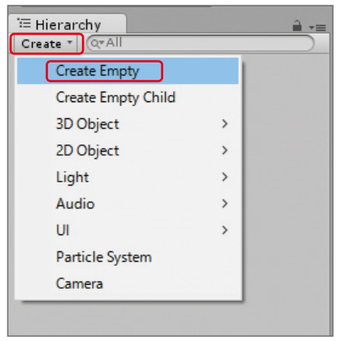
解説　アタッチとは

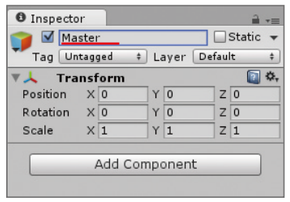
attach（アタッチ）とは、和訳すると「付着する」や「所属させる」という意味です。
Unityでは、オブジェクトにコンポーネント（スクリプトやRigidbody、Collderなど）を付けることをアタッチといいます。

マスターを作ってブロックを並べる

○Hierarchy（ヒエラルキー）から、ブロックを消します。
　BlockG、BlockM、BlockBをDelete（デリート：削除）します。
　ブロックを右クリックして、Delete選択します。
　※Shiftキーを押しながら同時に選択し右クリックすると同時にDeleteできます。
　※ブロックはプレハブ化してあり、スクリプトから生成します。したがって、Hierarchyにあるブロックを消してしまいます。

○Masterオブジェクトを作ります。
　HierarchyのCreate → Create Emptyの順にクリックします。
　名前を、「Master」にします。





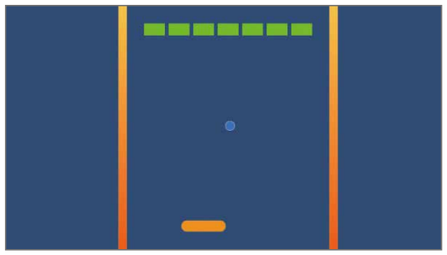
○スクリプトをMasterに付けます。
　スクリプトの名前を、「master」にします。

○次のようにスクリプトを記述します。

```
using UnityEngine;
using System.Collections;

public class master : MonoBehaviour {

    public GameObject blockG;
    public GameObject blockM;
    public GameObject blockB;

    // Use this for initialization
    void Start () {
        for (int i = 0; i < 7; i++) {
            Instantiate (blockG, new Vector3 (i - 3f, 4.0f, 0f), Quaternion.identity);
        }
    }

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

    }
}

```

記述したら、Command + Sでセーブします。

Unityの画面に戻り、HierarchyのMasterを選択します。
InspectorのBlockGにプレハブのBlockGをDDします。
InspectorのBlockMにプレハブのBlockMをDDします。
InspectorのBlockBにプレハブのBlockBをDDします。
この操作によって、プレハブ化したオブジェクトがスクリプトのパブリック変数に代入されます。
再生ボタンを押してみましょう。緑のブロックが一列並んだら成功です。



解説　スクリプトの解説

ブロックを複数生成する処理です。内容はCube編と同じで、横に並べるようにブロックを生成します。
for文を使って７回処理します。カウンター ｉ が0から始まり、ｉ が７未満である間繰り返し処理し、そして、一回処理するごとに、＋１されるので７回同じ処理をすることになります。
生成されるXの座標は、i - 3 です。i は0から始まるので一個目の座標は、(-3, 4, 0) です。二個目は、i が1なので (-2, 4, 0) です。最後は、i　が6なので (3, 4, 9) です。

○同じ要領でブロックを並べます。
スクリプトmasterを次のように編集します。

```
using UnityEngine;
using System.Collections;

public class master : MonoBehaviour {

    public GameObject blockG;
    public GameObject blockM;
    public GameObject blockB;

    // Use this for initialization
    void Start () {
        for (int i = 0; i < 7; i++) {
            Instantiate (blockG, new Vector3 (i - 3f, 4.0f, 0f), Quaternion.identity);
            Instantiate (blockG, new Vector3 (i - 3f, 3.4f, 0f), Quaternion.identity);
            Instantiate (blockM, new Vector3 (i - 3f, 2.8f, 0f), Quaternion.identity);
            Instantiate (blockM, new Vector3 (i - 3f, 2.2f, 0f), Quaternion.identity);
            Instantiate (blockB, new Vector3 (i - 3f, 1.6f, 0f), Quaternion.identity);
            Instantiate (blockB, new Vector3 (i - 3f, 1.0f, 0f), Quaternion.identity);
        }
    }

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

    }
}

```

記述したら、Command + Sでセーブします。
Unityに戻って再生ボタンを押してみましょう。

解説 スクリプトの解説

生成する種類とYの座標を変えて、生成する命令を増やしました。すべてループ文の中に書いてあるので、それぞれ7個ずつ生成されます。

Wall4にボールが当たったらミスにする

○Wall4にスクリプトをアタッチします。
HierarchyでWall4を選択し、InspectorのAdd Component → New Scriptの順でクリックします。
スクリプトの名前は、「wall4sc」とします。
スクリプトを次のように記述します。

```
using UnityEngine;
using System.Collections;

public class blocksc : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }

    void OnCollisionEnter2D(Collision2D col) {
        if (col.gameObject.tag == "Ball") {
            Destroy (gameObject);
        }
    }
}

```

解説 スクリプトの解説

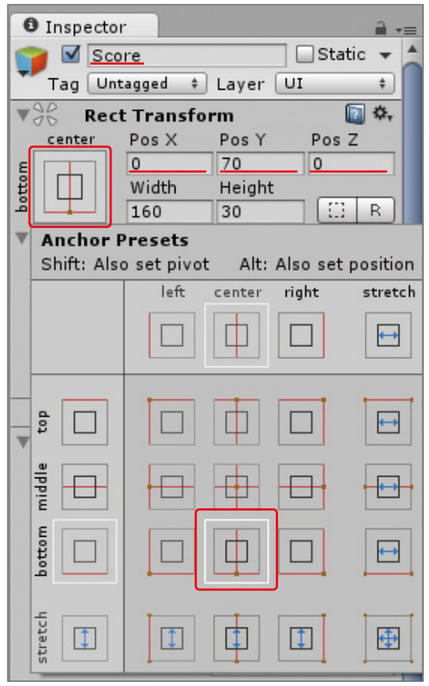
スクリプトblockscと似ていますが大きく違います。このスクリプトは、衝突したオブジェクトを消す処理です。
col には衝突したオブジェクトのデータが入っています。相手を消すにはDestroy(col.gameObject)で消せます。
衝突したとき自身を消すとのと相手を消すのでは、意味が違うので注意しましょう。

再生してみましょう。下の壁（Wall４）にボールが当たったときにボールが消えたら成功です。

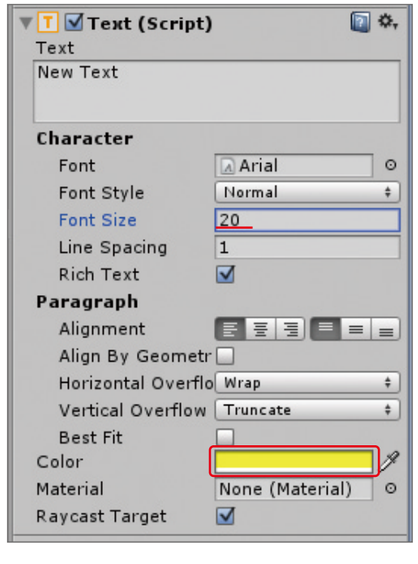
スコアを表示する

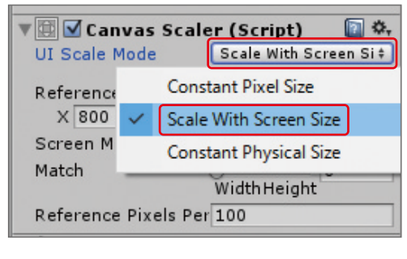
○スコアを表示するText（テキスト）オブジェクトを作ります。
HierarchyのCreate → UI → Textの順でクリックします。
HierarchyにCanvasとTextそしてEventSystemが作成されます。

Canvas（キャンバス）の子供となっているTextの名前を「Score」に変更します。
HierarchyでScoreを選択し、Inspectorで調整をします。
Rect Transform（レクトトランスフォーム）でcenter-bottomとします。
Rect Transform（レクトトランスフォーム）のPos Xを0、Pos Yを70にします。



Text(Script)のFont Size（フォントサイズ：字体の大きさ）を20にします。
Text(Script)のColor（カラー：色）を見やすい色に設定します。ここでは黄色にしました。
CanvasのCanvas Scaler（キャンバス スケーラー）のUI Scale Mode（UIスケールモード）をScale With Screen Size（スケールウィズ スクリーンサイズ：画面サイズに合わせて拡張）にします。





○ブロックに点数を設定します。
スクリプトblockscを開きます。
次のようにpublic（パブリック）変数blockScoreを追加します。

```
using UnityEngine;
using System.Collections;

public class blocksc : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }

    void OnCollisionEnter2D(Collision2D col) {
        if (col.gameObject.tag == "Ball") {
            Destroy (gameObject);
        }
    }
}

```

Unityに戻り、Assetsの中のBlockGを選択します。
InspectorのBlock(Script)にBlock Scoreが増えていますので、30とします。

同様に、BlockMとBlockBにも入力します。
BlockMを選択し、Block Scoreを20とします。
BlockBを選択し、Blcok Scoreを10とします。

BlockG、BlockM、BlockBには同じスクリプトblockscがアタッチされていますが、パブリック変数blockScoreを用意してInspectorから設定することで、ブロックの点数をそれぞれ設定しています。

○スクリプトmasterを編集します。
　次のようにmasterを編集します。

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

public class master : MonoBehaviour {

    public GameObject blockG;
    public GameObject blockM;
    public GameObject blockB;

    public static int score;
    public Text scoreText;

    // Use this for initialization
    void Start () {
        for (int i = 0; i < 7; i++) {
            Instantiate (blockG, new Vector3 (i - 3f, 4.0f, 0f), Quaternion.identity);
            Instantiate (blockG, new Vector3 (i - 3f, 3.4f, 0f), Quaternion.identity);
            Instantiate (blockM, new Vector3 (i - 3f, 2.8f, 0f), Quaternion.identity);
            Instantiate (blockM, new Vector3 (i - 3f, 2.2f, 0f), Quaternion.identity);
            Instantiate (blockB, new Vector3 (i - 3f, 1.6f, 0f), Quaternion.identity);
            Instantiate (blockB, new Vector3 (i - 3f, 1.0f, 0f), Quaternion.identity);
        }
    }

    // Update is called once per frame
    void Update () {
        scoreText.text = score.ToString ();
    }
}

```

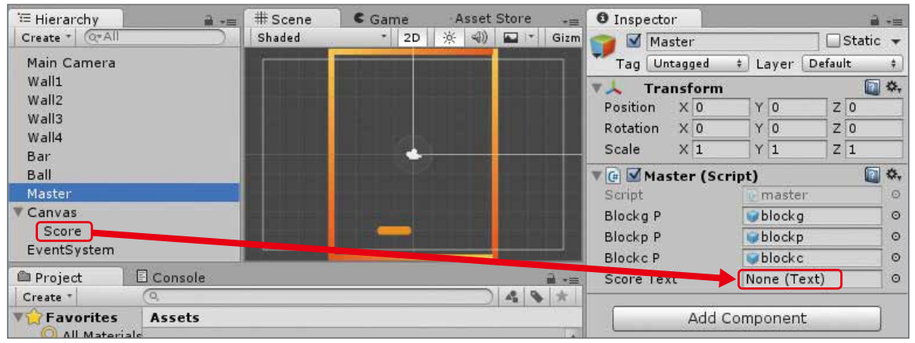
解説　スクリプトの解説

スコアを表示する処理を追加しました。
文字を表示するには、Text型の変数が持つtextに文字を入れておけばよいです。
scoreText.text = score.ToString(); の一行がそれに当たります。

解説　ToString（トゥストリング）とは

ToStringとは、変数が持つデータを文字に変換し取得する命令です。変数の情報は、変化することはありません。イコール（＝）で左辺の変数に代入するなどして使います。
ここでは、int型であるscoreの数値を文字に変換しtextに代入しています。

HierarchyでMasterを選択し、InspectorのMaster(Script)のScore Textに、HierarchyのCanvasの下のScoreをDDします。



○スクリプトblockscを編集します。
　次のようにblockscを編集します。

```

    void OnCollisionEnter2D(Collision2D col) {
        if (col.gameObject.tag == "Ball") {
            master.score += blockScore;
            Destroy (gameObject);
        }
    }
}

```

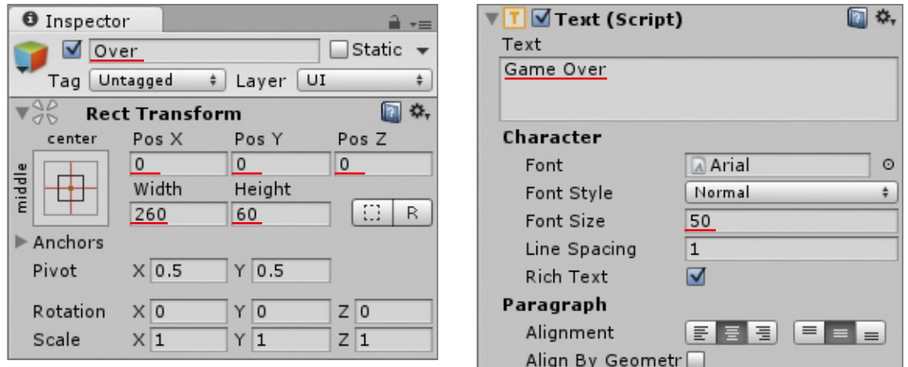
再生してみましょう。ブロックに当たって点数が増えていけば成功です。

GameOverの文字を出す

GameOverと表示するようにします。
○文字を表示するためにTextを追加します。
Hierarchyで、Create → UI → Textの順にクリックします。

Textという名前のオブジェクトが出来ますので、「Over」という名前にします。
Rect TransformのPos X、Pos Y、Pos Zをすべてに0とします。
Width（ウィズ：幅）を260、Height（ハイト：高さ）を60にします。

「New Text」を「Game Over」と書き換えます。
Font Size（フォントサイズ：書体の大きさ）を50にします。
Colorを見やすい色に設定します。ここでは赤色にしました。



○Wall4scのスクリプトを編集します。wall4scを開いて、次のように編集します。

```
1 using UnityEngine;
2 using System.Collections;
3 using UnityEngine.UI;
4
5 public class wall4sc : MonoBehaviour {
6
7     public Text overText;
8
9     // Use this for initialization
10    void Start () {
11        overText.enabled = false;
12    }
13
14    // Update is called once per frame
15    void Update () {
16
17    }
18
19    void OnCollisionEnter2D(Collision2D col) {
20        if (col.gameObject.tag == "Ball") {
21            overText.enabled = true;
22            Destroy (col.gameObject);
23        }
24    }
25 }
26
```

HierarchyでWall4を選択し、HierarchyのCanvasの下のOverを、Inspectorの
Wall 4（Script）のOver TextにDDします。
再生してみましょう。ボールを取り損ねたときにGame Overと表示されたら成功です。

解説 enabledとは（イネーブル）

和訳すると、「使用可能」という意味です。使用するかどうかを決定します。
enabledには、bool型（true（トゥルー）かfalse（ファルス））を入れます。true（真）を入れると使用するという意味になり、反対にfalse（偽）を入れると使用しないという意味になります。

解説 スクリプトの解説

TextのenabledにfalseをAを入れると文字が表示されなくなります。それを利用してStart関数内でfalseを代入し表示しない状態（非表示）にしておきます。そしてOnCollisionEnter2D関数内でtrueを代入することによってボールが衝突したときGameOverの文字を表示させる仕組みを作っています。

リプレイできるようにする。

GameOverと表示するようにします。

```
1 using UnityEngine;
2 using System.Collections;
3 using UnityEngine.UI;
4 using UnityEngine.SceneManagement;
5
6 public class wall4sc : MonoBehaviour {
7
8     public Text overText;
9     bool gameOvaer;
10
11    // Use this for initialization
12    void Start () {
13        overText.enabled = false;
14    }
15
16    // Update is called once per frame
17    void Update () {
18        if (gameOvaer && Input.GetKeyDown ("n")) {
19            gameOvaer = false;
20            master.score = 0;
21            SceneManager.LoadScene (0);
22        }
23    }
24
25    void OnCollisionEnter2D(Collision2D col) {
26        if (col.gameObject.tag == "Ball") {
27            gameOvaer = true;
28            overText.enabled = true;
29            Destroy (col.gameObject);
30        }
31    }
32 }
33
```

解説 bool型（ブール）とは

bool型とは、true（トゥルー）かfalse（ファルス）どちらかの値を持つ型です。trueは、真。falseは、偽という意味を持っています。

解説 if文とbool型

if文のカッコ内がtrueなら中カッコ内の処理を行う仕組みになっていてbool型の変数をif文のカッコ内に入れることができます。

解説 Input.GetKeyDown関数

Input,GetKeyDown関数は、カッコ内に指定したキーが押された瞬間だけtrueが取得できそれ以外は、falseが取得できるものです。

解説 スクリプトの解説

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

public class wall4sc : MonoBehaviour {

    public Text overText;
    bool gameOvaer;

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

    // Update is called once per frame
    void Update () {
        if (gameOvaer && Input.GetKeyDown ("n")) {
            gameOvaer = false;
            master.score = 0;
            SceneManager.LoadScene (0);
        }
    }

    void OnCollisionEnter2D(Collision2D col) {
        if (col.gameObject.tag == "Ball") {
            gameOvaer = true;
            overText.enabled = true;
            Destroy (col.gameObject);
        }
    }
}
```

SceneManagerを使うのに必要

ゲームオーバー状態かどうかを記憶

ゲームオーバーの状態で
Nキーが押された時のみ処理する

ゲームオーバーの判定をリセット

スコアをリセット

このゲームの再読み込み

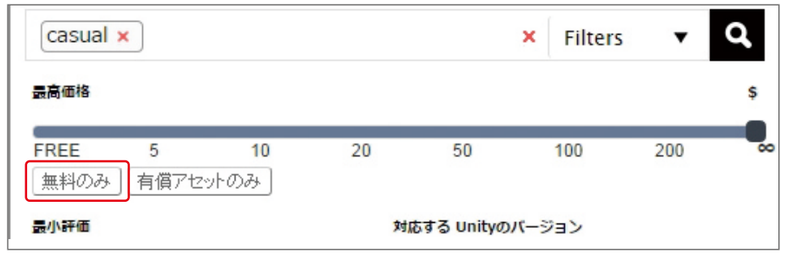
ゲームオーバーになった

再生してみましょう。GameOverの後、nボタンを押して再度プレイできれば成功です。

効果音を付ける

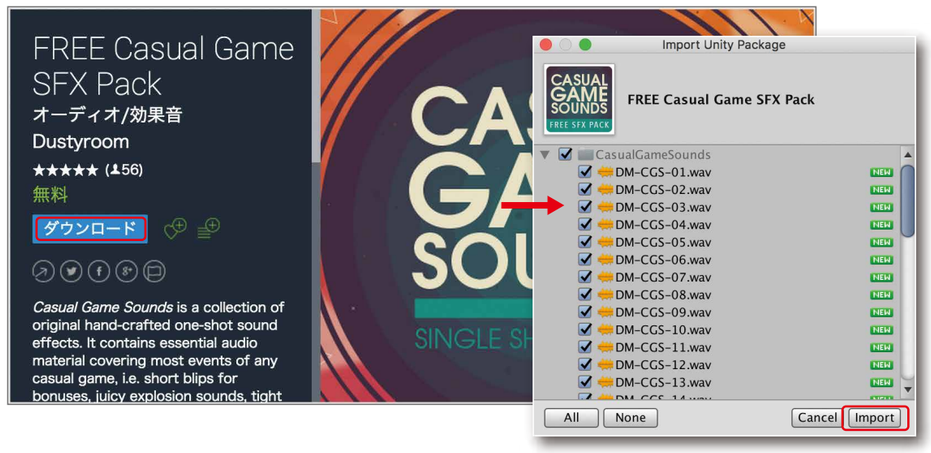
○Asset Storeから効果音をダウンロードしてインポートします。
Asset Store（アセットストア）は、Gameタブの隣にあります。
※ない場合は、Command + 9（Commandを押しながら9のキーを押す）と出てきます。
Asset Store（アセットストア）の検索窓に、「casual」と入力して虫眼鏡ボタンを押します。

次に、無料のみ、のボタンを押します。



下の方へスクロールして、「FREE Casual Game SFX Pack」をクリックします。

次にダウンロードボタンを押し、出てきたウインドウのImport（インポート）を押します。



Assetsの中にCasualGameSoundsが現れますので、クリックします。
つぎに、DM-CGS-03と、DM-CGS-10をHierarchyへDDします。

DM-CGS-03は、Barに当たった音です。
DM-CGS-10は、Blockを破壊した音です。

HierarchyのDM-CGS-03 (1)の名前を、「Bound」（バウンド）に変更します。
InspectorでAudioSouceのPlay On Awakeのチェックを外します。
HierarchyのDM-CGS-10 (1)の名前を、「Hit」（ヒット）に変更します。
InspectorでAudioSouceのPlay On Awakeのチェックを外します。

○バーへの衝突音をつけます。
Assetsの中のbarconを開いて、次のように編集します。

```
using UnityEngine;
using System.Collections;

public class barcon : MonoBehaviour {

    public float speed = 10;
    AudioSource boundSound;

    // Use this for initialization
    void Start () {
        boundSound = GameObject.Find ("Bound").GetComponent<AudioSource> ();
    }

    // Update is called once per frame
    void Update () {
        float x = Input.GetAxisRaw ("Horizontal");
        GetComponent<Rigidbody2D> ().velocity = new Vector2 (x * speed * Time.deltaTime, 0);
    }

    void OnCollisionEnter2D(Collision2D col) {
        boundSound.Play ();
    }
}
```

再生してみましょう。Barにボールがあたったときに音がすれば成功です。
ただし、Barが左右の壁に当たっても音がします。これを改善します。

スクリプトbarconを次のように変更します。

```
19
20      void OnCollisionEnter2D(Collision2D col) {
21          if (col.gameObject.tag == "Ball") {
22              boundSound.Play ();
23          }
24      }
25  }
26
```

これは、Bar（バー）に衝突したのがBallだったら音を鳴らす処理です。

○Wall1～3への衝突音を付けます。
Wall1に新しいスクリプトをアタッチします。
HierarchyでWall1を選択し、InspectorのAdd Component → New Scriptの順にクリックします。
スクリプトの名前を「wallsc」とします。
wallscを開いて次のように記述します。
先ほどのプログラムと同じです。 そこで、コピー＆ペーストで記述します。
やり方は、barconの中の必要な部分をマウスでなぞり（クリックしながら動かします）、
Command＋Cを押してコピーします。
次に、wallの中で記述したい部分にカーソルを持っていき、Command＋Vでペーストします。
これを何回か繰り返して、下のように記述します。

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class wallsc : MonoBehaviour {
5
6      AudioSource boundSound;
7
8      // Use this for initialization
9      void Start () {
10         boundSound = GameObject.Find ("Bound").GetComponent<AudioSource> ();
11     }
12
13     // Update is called once per frame
14     void Update () {
15
16     }
17
18     void OnCollisionEnter2D(Collision2D col) {
19         if (col.gameObject.tag == "Ball") {
20             boundSound.Play ();
21         }
22     }
23 }
24
```

記述したらセーブして、Unityの画面に戻ります。

Wall2、Wall3にスクリプトwallscをアタッチします。
再生して確認しましょう。WallにBallがあった時、音が鳴れば成功です。

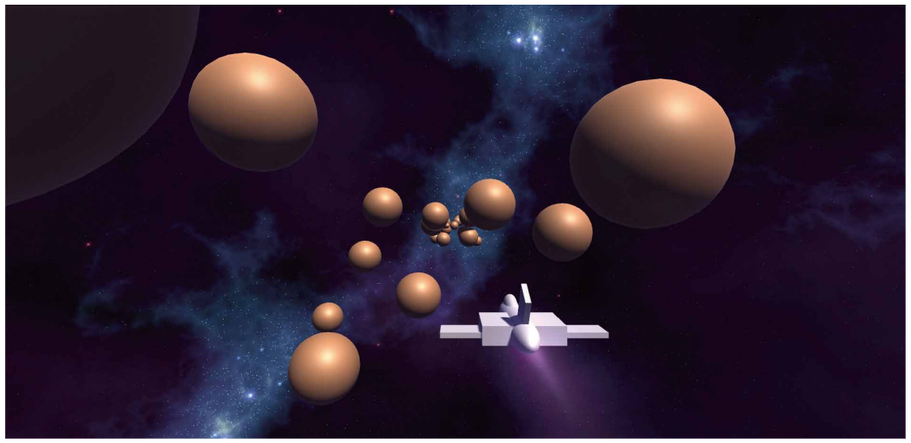
○ブロックの破壊音を付けます。
Assetsの中のblockscを開いて、次のように編集します。

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class blocksc : MonoBehaviour {
5
6      public int blockScore;
7      AudioSource hitSound;
8
9      // Use this for initialization
10     void Start () {
11         hitSound = GameObject.Find ("Hit").GetComponent<AudioSource> ();
12     }
13
14     // Update is called once per frame
15     void Update () {
16
17     }
18
19     void OnCollisionEnter2D(Collision2D col) {
20         if (col.gameObject.tag == "Ball") {
21             hitSound.Play ();
22             master.score += blockScore;
23             Destroy (gameObject);
24         }
25     }
26 }
27
```

再生してみましょう。ブロックに当たったときに音が出たら成功です。

memo

Space編



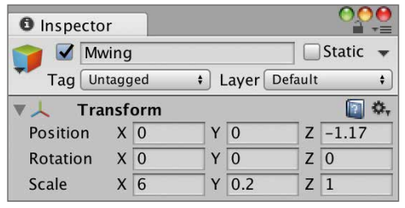
P96　プロジェクトを作成する
P96　飛行機を作ります
P99　飛行機が左右キー操作で左右に傾くようにする
P101　飛行機が上下キー操作で前後に傾くようにする
P102　飛行機を画面の中で移動させる
P106　Skybox（スカイボックス）を設定する
P107　エフェクトを付ける
P109　隕石を作る
P115　レーザーを発射できるようにする
P119　レーザーに当たった隕石が破壊されるようにする

プロジェクトを作成する

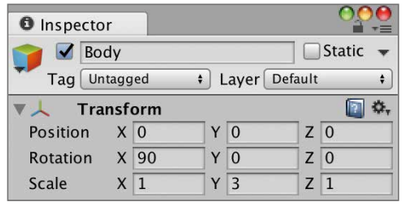
プロジェクト名は、「Space」（スペース：宇宙）とします。
3Dにチェックを入れて、Create Projectを押してでプロジェクトを作成します。

飛行機を作ります

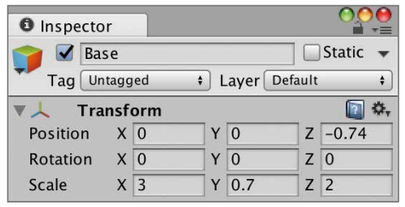
○Mwing（Mウィング）を作ります。
Cubeを作ります。
HierarchyのCreate → 3D Object → Cubeの順でクリックします。
名前を、「Mwing」とし、Transformを次のように設定します。
Position（ポジション：位置）のZを-1.17
Scale（スケール：大きさ）のXを6、Yを0.2



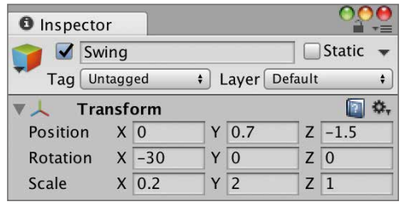
○Body（ボディ：胴体）を作ります。
Capsuleを作ります。
HierarchyのCreate → 3D Object → Capsuleの順でクリックします。
名前を、「Body」とし、Transformを次のように設定します。
Rotation（ローテーション：回転）のXを60
Scale（スケール：大きさ）のYを3



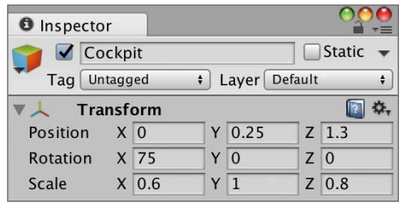
○Base（ベース：基礎）を作ります。
Cubeを作ります。
HierarchyのCreate → 3D Object → Cubeの順でクリックします。
名前を、「Base」とし、Transformを次のように設定します。
Position（ポジション：位置）のZを-0.74
Scale（スケール：大きさ）のXを3、Yを0.7、Zを2

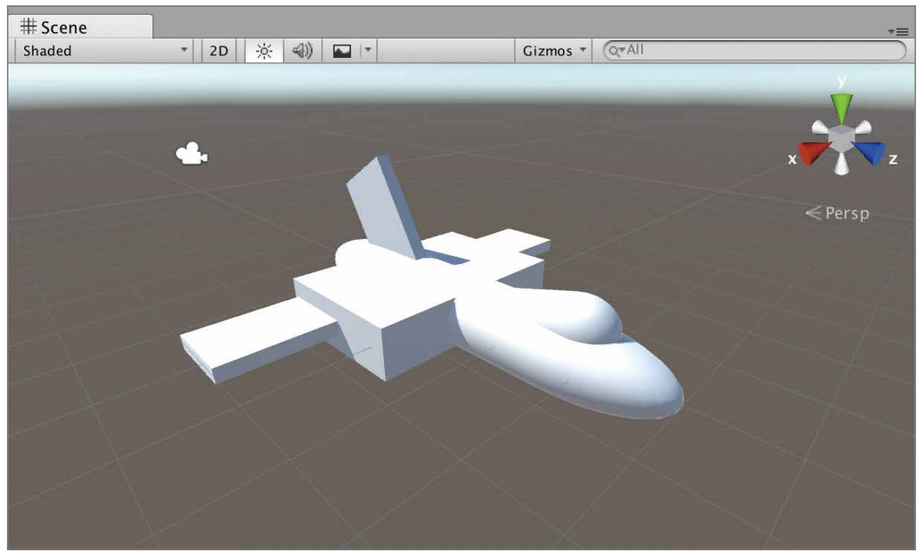


○Swing（Sウィング）を作ります。
Cubeを作ります。
名前を、「Swing」とし、Transformを次のように設定します。
Position（ポジション：位置）のYを0.7、Zを-1.5
Rotation（ローテーション：回転）のXを-30
Scale（スケール：大きさ）のXを0.2、Yを2

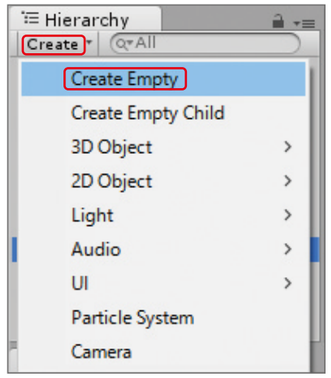


○Cockpit（コックピット：操縦席）を作ります。
Capsuleを作ります。
名前を、「Cockpit」とし、Transformを次のように設定します。
Position（ポジション：位置）のYを0.31、Zを-1.5
Rotation（ローテーション：回転）のXを75
Scale（スケール：大きさ）のXを0.6、Zを0.8

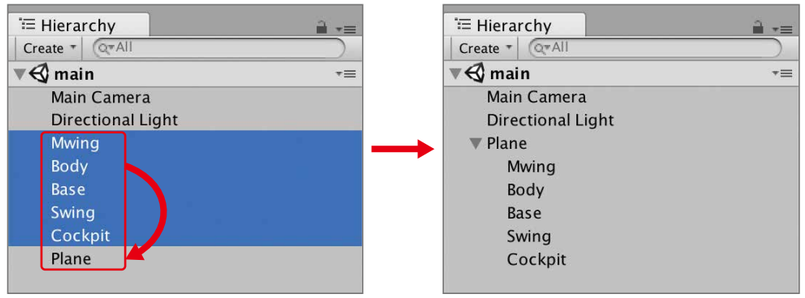




○作った飛行機をPlaneの子オブジェクトにします。
HierarchyのCreate → Create Emptyの順にクリックします。
名前を「Plane」とします。



先ほど作った５つの部品を同時に選択し、Planeの中へDDします。
同時に選択する場合、Commandキーを押しながら順にクリックしていきます。



これで5つの部品が、Planeオブジェクトの子オブジェクトとなりました。
Planeを動かすと、5つの部品が一緒に動きます。

ここまで出来たら、Command + Sでシーンをセーブします。
シーンの名前は、「main」とします。

飛行機が左右キー操作で左右に傾くようにする

○Planeにスクリプトを付けます（アタッチします）。
HierarchyのPlaneを選択し、InspectorのAdd Componentをクリックします。
スクリプト名は、「PlaneCon」とします。
次のように、PlaneConを編集します。

```
using UnityEngine;
using System.Collections;

public class PlaneCon : MonoBehaviour {

    Vector3 center;
    Vector3 turn;
    float hori;
    int sign;

    // Use this for initialization
    void Start () {
        center = new Vector3 (0f, 0f, 0f);
        turn = new Vector3 (0f, 0f, -70f);
    }

    // Update is called once per frame
    void Update () {
        hori = Input.GetAxis ("Horizontal");
        if (hori > 0f) {
            sign = 1;
        }
        if (hori < 0f) {
            sign = -1;
        }
        transform.eulerAngles = Vector3.Lerp (center, turn * sign, hori * sign);
    }
}

```

記述できたらCommand + Sでセーブします。
再生してみましょう。矢印の右左を押して飛行機が左右に傾けば成功です。

解説 トランスフォーム オイラーアングル
transform.eulerAnglesとは

transformは、ゲームオブジェクトが持っているもので、座標、回転値、大きさを管理します。
transform.eulerAnglesは、角度を０～３６０度で表現するオイラーで指定するものです。
XYZそれぞれ指定する必要があり受け渡しには、Vector3型を使用します。
代入例）　transform.eulerAngles = new Vector3(0f, 45f, 225f);

解説 ベクタースリー ラープ
Vector3.Lerp関数

Vector3.Lerpとは、２点を結んだ直線上の点を作成するものです。
始点と終点、その間の割合で点を作成します。割合は、始点を0.0とし終点を1.0とした0.0～1.0の値です。
この関数は次のように書きます。　Vector3.Lerp(始点 , 終点 , 割合);

解説 スクリプトの解説

このスクリプトは、オブジェクトを入力により回転させるものです。
Input.GetAxisで左右の入力を受け取ります。そのデータは、左入力なら－１～０、右入力なら０～＋１です。押した時間に応じて徐々に０から±１に近づきます。
左右の入力の値をVector3.Lerp関数の割合に使いますが、マイナスの値を受け付けてくれないので、マイナスをプラスに反転必要があります。hugo変数に－１の値を入れてyokoとかけて反転させます。その際に終点であるturnも反転させています。
入力が左の時のみ反転する必要がありますが、if文を用いてyoko変数の符号を判断し、hugo変数に代入する値を決定しています。

```
public class PlaneCon : MonoBehaviour {

    Vector3 center;
    Vector3 turn;
    float hori;
    int sign;

    // Use this for
    void Start () {
        center = new Vector3 (0f, 0f, 0f);
        turn = new Vector3 (0f, 0f, -70f);
    }

    // Update is called once per frame
    void Update () {
        hori = Input.GetAxis ("Horizontal");
        if (hori > 0f) {
            sign = 1;
        }
        if (hori < 0f) {
            sign = -1;
        }
        transform.eulerAngles = Vector3.Lerp (center, turn * sign, hori * sign);
    }
}
```

入力情報を一時記録する変数
Horizontal の略で hori

入力情報を一時記録する変数
Sign（サイン：符号）

始点

終点

左右の入力　値は、0～±1

左右入力から hori と sign の符号を反転させるための値を決定

オブジェクトを回転　sign によって　turn と hori の符号が反転する

飛行機が上下キー操作で前後に傾くようにする

○pconを編集します。

```
using UnityEngine;
using System.Collections;

public class PlaneCon : MonoBehaviour {

    Vector3 center;
    Vector3 turn;
    float hori;
    int sign;
    Vector3 upDown;
    float vert;

    // Use this for initialization
    void Start () {
        center = new Vector3 (0f, 0f, 0f);
        turn = new Vector3 (0f, 0f, -70f);
        upDown = new Vector3 (-30f, 0f, 0f);
    }

    // Update is called once per frame
    void Update () {
        hori = Input.GetAxis ("Horizontal");
        if (hori > 0f) {
            sign = 1;
        }
        if (hori < 0f) {
            sign = -1;
        }
        transform.eulerAngles = Vector3.Lerp (center, turn * sign, hori * sign);

        if (hori != 0f) {
            return;
        }

        vert = Input.GetAxis ("Vertical");
        if (vert > 0f) {
            sign = 1;
        }
        if (vert < 0f) {
            sign = -1;
        }
        transform.eulerAngles = Vector3.Lerp (center, upDown * sign, vert * sign);
    }
}

```

再生してみましょう。矢印の上下を押して飛行機が前後に傾けば成功です。

解説 returnとは（リターン）

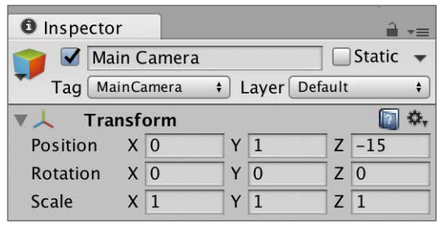
処理を中断させる命令です。
returnより下の命令は、処理されません。今回の場合は、if文で分岐させてあるので、if文の条件が満たされた時（yokoが0以外の時）処理が中断されてreturnより下の命令は処理されません。

解説 スクリプトの解説

上下入力でも回転するように追記しました。
if (yoko != 0f) というif文があり{}中カッコ内には、return命令が書かれています。これは、「横入力があった時に処理を中断する」という意味になります。なぜ書かれているかというと、横の入力による回転値を代入したtransform.eulerAnglesが上書きされるのを防いでいます。もし、防がなかったら問題が起きます。右入力のみをして、yokoが1.0だった場合、1回目のtransform.eulerAnglesに代入される値は (0, 0, -70) になります。ここでreturnがなかった場合、縦入力の処理に入ります。tateが0.0であったら2回目のransform.eulerAnglesに代入される値は (0, 0, 0) になり結果として回転していない状態になってしまいます。

飛行機を画面の中で移動させる

飛行機を画面の中で移動させる



○Planeを動くようにする
スクリプト名を、「PlaneMove」とします。
次のように編集します。

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class PlaneMove : MonoBehaviour {
5
6     public float speed = 0.5f;
7     Vector3 pos;
8     float hori;
9
10    // Use this for initialization
11    void Start () {
12
13    }
14
15    // Update is called once per frame
16    void Update () {
17        pos = transform.position;
18        hori = Input.GetAxis ("Horizontal");
19        transform.position = pos + new Vector3 (speed * hori, 0f, 0f);
20    }
21 }
22
```

再生してみましょう。飛行機が左右に動けば成功です。
ただし、画面からはみ出しても移動し続けてしまいます。次は、これを修正します。

解説 transform.position（トランスフォーム ポジション）とは

オブジェクトの座標で、Inspectorに表示されているものです。
この変数に値を入れるとオブジェクトの位置を変更することができます。また、現在の座標に移動距離を足した値を代入することで動いているように見せることができます。今回のスクリプトでは、この手法を使って動かしています。

解説 スクリプトの解説

左右入力で左右に動かすプログラムです。
Input.GetAxis("Horizontal")で取得できる値は、右入力は０～＋１左入力は－1～０です。
この値で移動方向を決定して、変数speedと掛けて速度を調節しています。

現在の位置と移動する距離を足すことで移動先の位置がわかります。その値をtransform.positionに代入することで、オブジェクトを移動させています。

```
using UnityEngine;
using System.Collections;

public class PlaneMove : MonoBehaviour {

    public float speed = 0.5f;
    Vector3 pos;          ← 位置を一時記録する用の変数
    float hori;           ← 入力情報を記録する用の変数

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        pos = transform.position;          ← 現在の位置を記録
        hori = Input.GetAxis ("Horizontal");
        transform.position = pos + new Vector3 (speed * hori, 0f, 0f);
    }
}
```

代入することで移動させる / 移動先の位置 / 移動する距離

○Myshipが画面内で左右に移動するようにします。
スクリプトPlaneMoveを次ように編集します。

```
using UnityEngine;
using System.Collections;

public class PlaneMove : MonoBehaviour {

    public float speed = 0.5f;
    Vector3 pos;
    float hori;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        pos = transform.position;
        hori = Input.GetAxis ("Horizontal");
        if (pos.x > 10f && hori > 0f) {
            hori = 0f;
        }
        if (pos.x < -10f && hori < 0f) {
            hori = 0f;
        }
        transform.position = pos + new Vector3 (speed * hori, 0f, 0f);
    }
}
```

編集が終わったら、Command + Sでセーブします。
再生してみましょう。左右に移動すると、見えない壁があるかのように止まれば成功です。

解説 スクリプトの解説

X座標が一定範囲外に出そうになった時に移動処理を行わないようにしました。

追記部分を解説します。
まず、前提として、変数posには現在の座標が、変数horiには左入の入力情報が入っています。
一つ目のif文では、posのｘが10より高く、尚且つhoriが0より大きい場合は、horiに0を入れる。という処理をしています。言葉を変えると、現在の座標のｘが10より高く、右入力をしていると時、入力をしていないことにする。ということになります。
horiを0にすることにより、最後のspeed * horiの掛け算の結果が0になります。結果として移動しないという処理になっています。

○Myshipが画面内で上下に移動するようにします。
スクリプトshipのUpdate関数を、次のように編集します。

```
using UnityEngine;
using System.Collections;

public class PlaneMove : MonoBehaviour {

    public float speed = 0.5f;
    Vector3 pos;
    float hori;
    float vert;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        pos = transform.position;
        hori = Input.GetAxis ("Horizontal");
        if (pos.x > 10f && hori > 0f) {
            hori = 0f;
        }
        if (pos.x < -10f && hori < 0f) {
            hori = 0f;
        }

        vert = Input.GetAxis ("Vertical");
        if (pos.y > 6f && vert > 0f) {
            vert = 0f;
        }
        if (pos.y < -6f && vert < 0f) {
            vert = 0f;
        }

        transform.position = pos + new Vector3 (speed * hori, speed * vert, 0f);
    }
}
```

編集が終わったら、Command + Sでセーブします。
再生してみましょう。上下にも見えない壁があるかのように止まれば成功です。

解説 スクリプトの解説

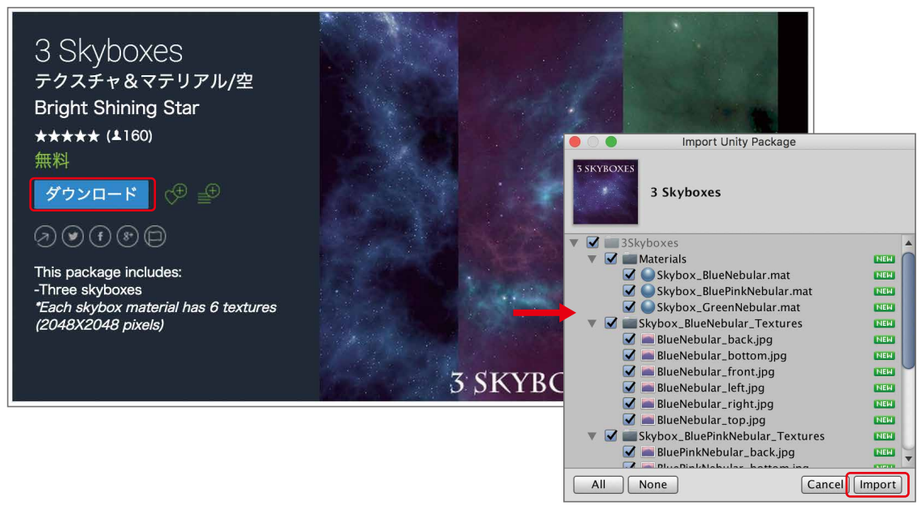
上下に移動する処理と一定範囲外に出ないようにする制御処理を追加しました。
考え方は、左右の時と変わりません。使う変数や値が違うだけです。

Skybox（スカイボックス）を設定する

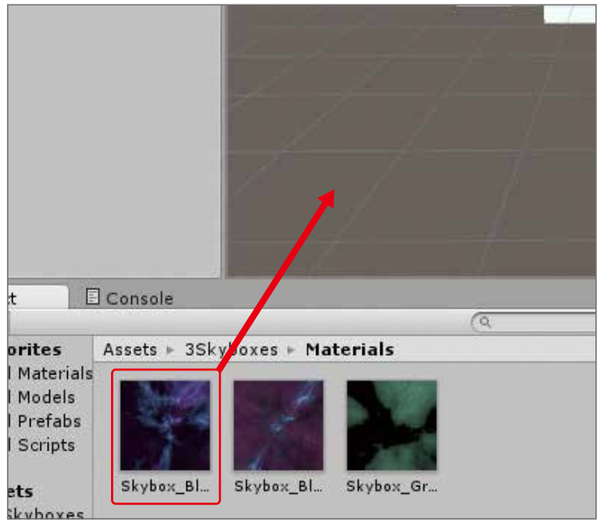
○スカイボックスを設定します。
Asset Store（アセットストア）の検索枠に、skyboxと入力し、検索します。
無料のみ、を押します。

下のほうへスクロールし、3 Skyboxesをクリックします。

次にダウンロードボタンを押し、出てきたウインドウのImport（インポート）を押します。



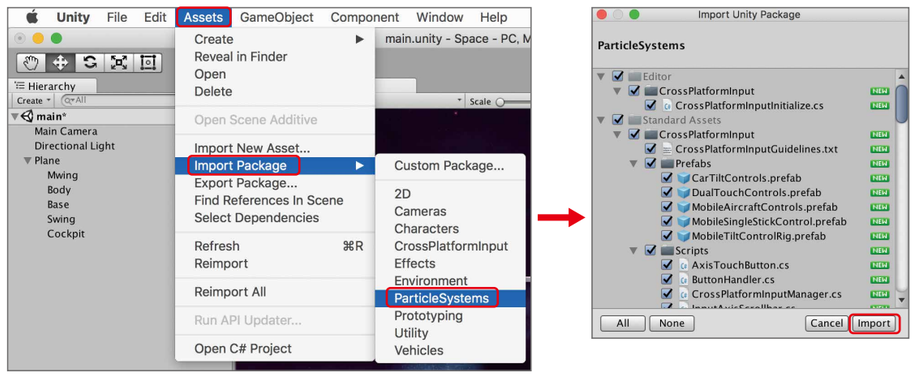
Assetsの中の3Skyboxes → Materialsのフォルダを開きます。
Materialsの中に3つの画像があります。
どれでもよいので、Scene（シーン）へDDします。



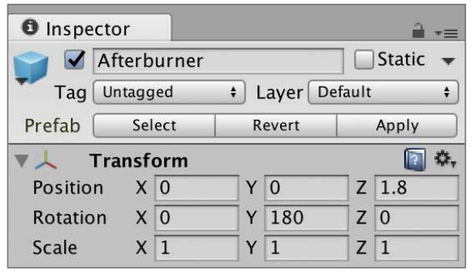
再生してみましょう。背景が宇宙空間となれば成功です。

エフェクトを付ける

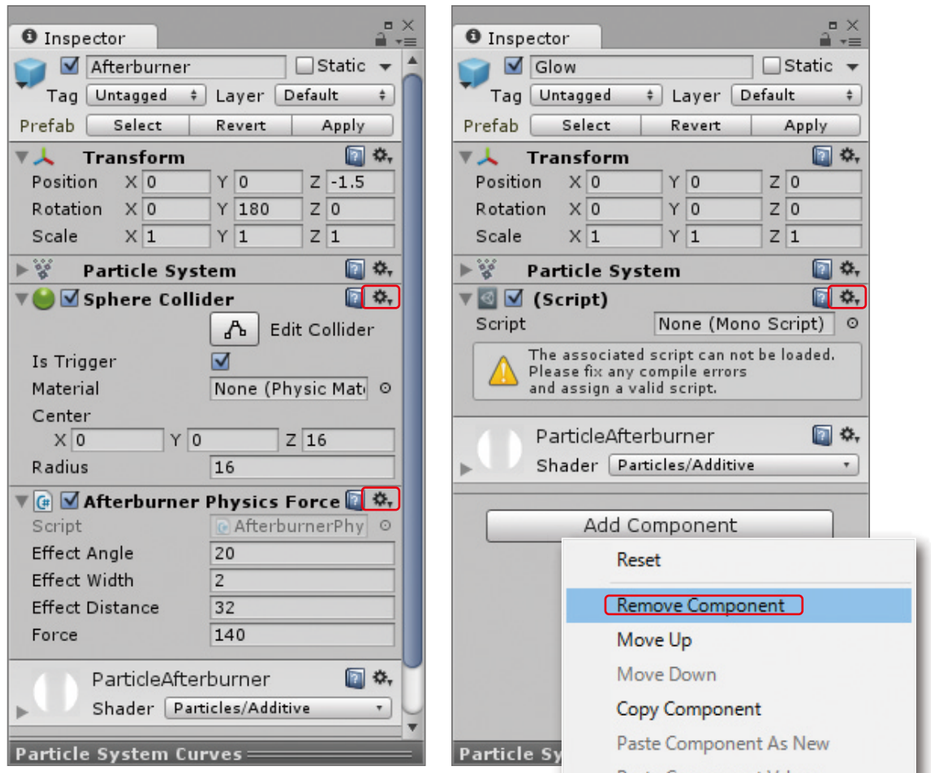
ジェットのバーナーエフェクトを付けます。
一番上のメニューで、Assets → Import Package → PerticleSystemsの順にクリックします。
出てきたウインドウでImportをクリックします。



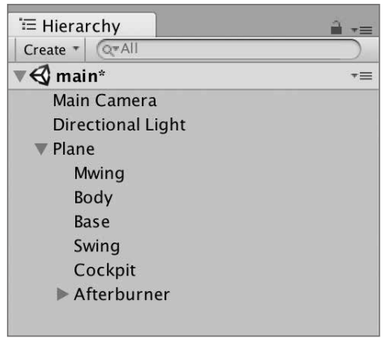
Assetsの中のStandard Assets → ParticleSystems → Prefabsのフォルダを開きます。
そして、Afterburner（アフターバーナー）をHierarchyにDDします。
Transformを次のように設定します。
PositionのZを-1.5
RotationのYを180



Afterburner に付いているAfterburner Physics Forceを外します。
Afterburner Physics Forceの右上の歯車をクリックし、Remove Componentをクリックします。
同様に、Afterburnerに付いているSphere Colliderを削除します。
同様に、Afterburnerの子オブジェクトGlowについているスクリプトを削除します。



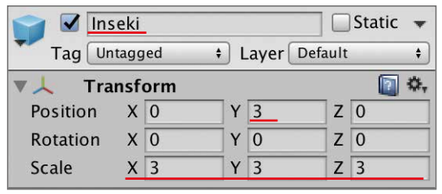
HierarchyのAfterburnerを、PlaneにDDします。
これでAfterburnerもPlaneの子オブジェクトになります。



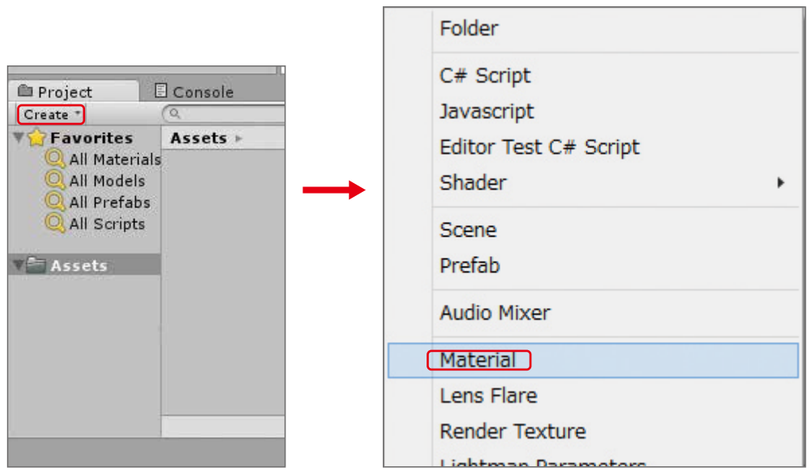
再生してみましょう。AfterburnerとPlaneが一緒に動けば成功です。

隕石を作る

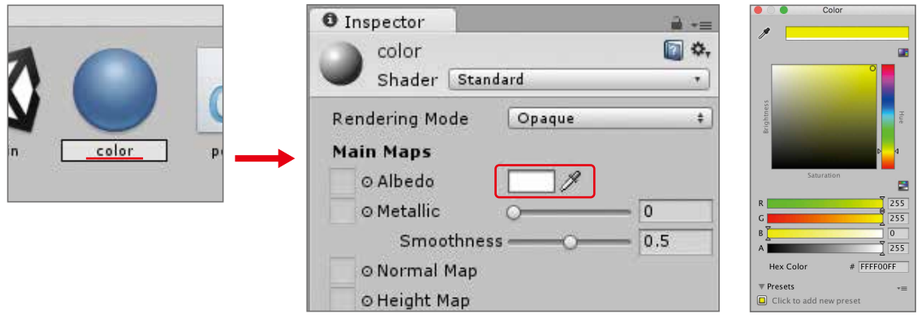
○Sphere（スフィア：球）を作ります。球の作り方を覚えていますか。
Hierarchyの、Create → 3D Object → Sphereの順にクリックします。
名前を、「Inseki」とします。
TransformのPositionのYを3、Scaleをすべて3にします。



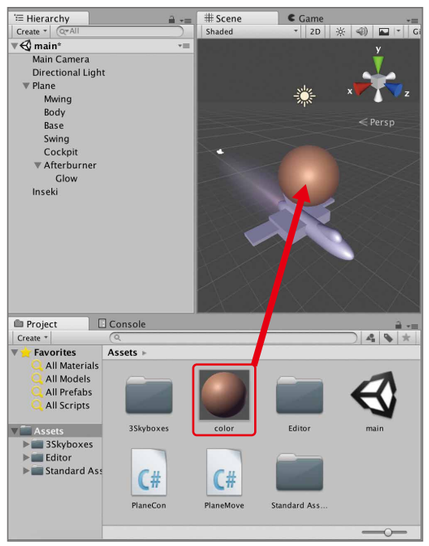
○色を付けます。
ProjectのAssetsをクリックし、Create → Materialとクリックします。



名前を「color」とします。
スポイトマークの横の枠をクリックし、好きな色にします。

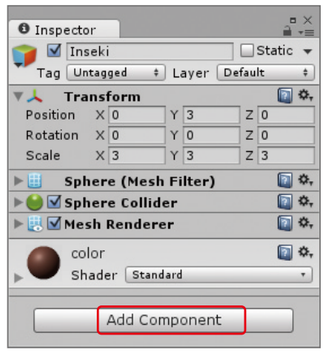


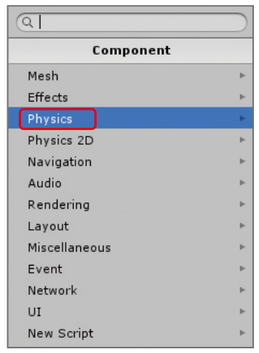
出来たColorをシーンのInsekiにDDします。



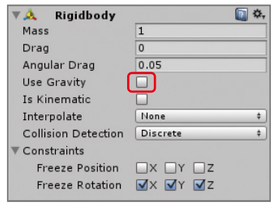
ここでは、茶色の隕石を作りました。

次に、InsekiにRigidbody（リジッドボディ：剛体）を付けます。
HierarchyのInsekiを選択し、InspectorのAdd Component → Physics → Rigidbodyとクリックします。

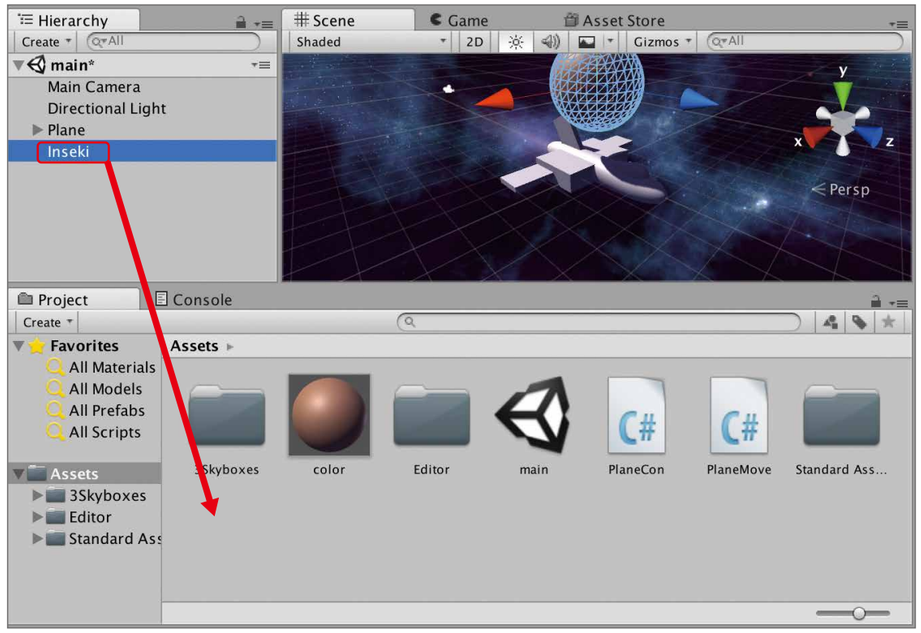




RigidbodyのUse Gravity（ユーズ・グラビティー）のチェックを外します。



○Inseki（隕石）をプレハブ化します。
ProjectのAssetsをクリックし、InsekiをAssetsの直下へDDします。



HierarchyのInsekiは不要ですので、右クリックしてDelete（デリート）します。

○隕石を生成します。
空のオブジェクトを作ってスクリプトをアタッチします。
まずは、空のオブジェクトを作ります。
HierarchyのCreate → Create Emptyをクリックします。
オブジェクト名を「Master」とします。
次にスクリプトを作成してアタッチします。
Inspectorで、Add Component → New Scriptとクリックします。
スクリプトの名前を、「master」とします。

masterを次のように編集します。

```
using UnityEngine;
using System.Collections;

public class master : MonoBehaviour {

    public GameObject insekiPf;
    float timeCount;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        timeCount += Time.deltaTime;
        if (timeCount > 2f) {
            timeCount = 0;

            for (int i = 0; i < 10; i++) {
                float rndx = Random.Range (-10f,  10);
                float rndy = Random.Range ( -6f,   6);
                float rndz = Random.Range (200f, 250);
                GameObject inseki = (GameObject)Instantiate
                    (insekiPf, new Vector3 (rndx, rndy, rndz), Quaternion.identity);
                inseki.GetComponent<Rigidbody> ().velocity = new Vector3 (0f, 0f, -50f);
            }
        }

    }
}

```

解説 Random.Range（ランダムレンジ）とは

Random.Range関数は、指定した範囲内からランダムで数値を作ります。
この関数は、次のように書きます。　Random.Range（最小, 最大）;

解説 Time.deltaTime（タイム デルタ タイム）とは

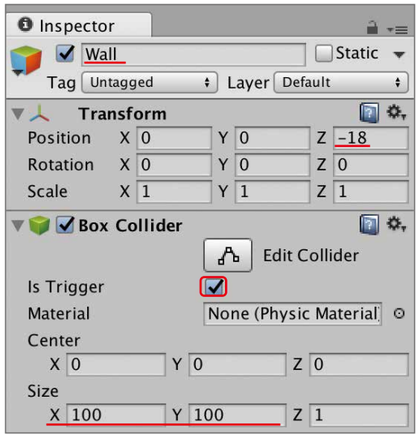
Time.deltaTimeとは、1フレームを処理するのに要した時間です。
この数値をUpdate関数内で加算し続けると経過時間を計ることができます。

解説 スクリプトの解説

２秒ごとに隕石を、一定範囲内に１０個生成して飛ばすスクリプトです。
変数timeCountに、Time.deltaTimeを加算し続けて時間を計ります。if文で２秒たったら処理を分岐して、変数timeCountの値を０にリセットします。その後、隕石を生成する処理を行います。
隕石生成は、Instantiate関数で生成します。このとき、必要な生成位置は、ランダムで作っています。
生成したのち生成した隕石を取得し、まっすぐ飛ばすように力を加えています。
これらの生成位置の値づくり、生成処理、飛ばす処理をfor文で囲い１０回行う事で隕石を１０個飛ばしています。

```
public class master : MonoBehaviour {
    public GameObject insekiPf;                 // Inspector で設定したプレハブが入っている
    float timeCount;                            // 時間を数える変数　Time Count（タイムカウント）
    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {                            // 時間を計る
        timeCount += Time.deltaTime;
        if (timeCount > 2f) {                   // 2秒に一度処理する
            timeCount = 0;
            for (int i = 0; i < 10; i++) {      // 座標作成、生成、飛ばす処理を10回行う
                float rndx = Random.Range (-10f,  10);      // ランダムで座標を作る
                float rndy = Random.Range ( -6f,   6);
                float rndz = Random.Range (200f, 250);
                GameObject inseki = (GameObject)Instantiate     // 生成して inseki に代入
                    (insekiPf, new Vector3 (rndx, rndy, rndz), Quaternion.identity);
                inseki.GetComponent<Rigidbody> ().velocity = new Vector3 (0f, 0f, -50f);    // inseki を飛ばす
            }
        }
    }
}
```

○不要な隕石を消去します。
飛行機の後ろまで飛んだ隕石を消す処理を作ります。
空のオブジェクトを作成します。
Create → Create Emptyとクリックします。
名前を、「Wall」とします。
InspectorのAdd Component
→ Physics → Box Colliderとクリックします。
右のように設定します。



Wallにスクリプトをアタッチします。
スクリプトの名前を、「Wall」とします。
スクリプトWallを次のように編集します。

```
using UnityEngine;
using System.Collections;

public class Wall : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }

    void OnTriggerEnter(Collider col) {
        Destroy (col.gameObject);
    }
}

```

再生してみましょう。HierarchyからInseki (clone)が消えていれば成功です。

解説 スクリプトの解説

Wallに衝突したオブジェクトを削除する処理です。
OnTriggerEnter(Collider col) のcolには、衝突したオブジェクトのデータが入っています。なので、col.gameObjectで取得できます。
Destroy関数を使い、衝突したものを削除しています。

○隕石の大きさをランダムにします。
スクリプトmasterを次のように編集します。

```
    // Update is called once per frame
    void Update () {
        timeCount += Time.deltaTime;
        if (timeCount > 2f) {
            timeCount = 0;

            for (int i = 0; i < 10; i++) {
                float rndx = Random.Range (-10f,  10);
                float rndy = Random.Range ( -6f,   6);
                float rndz = Random.Range (200f, 250);
                GameObject inseki = (GameObject)Instantiate
                    (insekiPf, new Vector3 (rndx, rndy, rndz), Quaternion.identity);
                inseki.GetComponent<Rigidbody> ().velocity = new Vector3 (0f, 0f, -50f);

                float rnds = Random.Range (2f, 8f);
                inseki.transform.localScale = new Vector3 (rnds, rnds, rnds);
            }
        }
    }
```

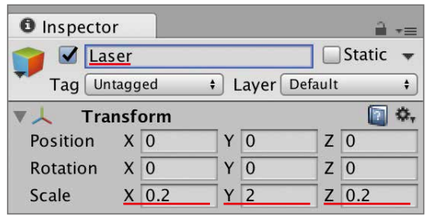
再生してみましょう。ランダムな大きさの隕石が生成されれば成功です。

解説 トランスフォーム ローカルスケール transform.localScaleとは

transfom.localScaleは、大きさの値です。これに代入することでゲームオブジェクトの大きさを変えることができます。
代入例　transfom.localScale = new Vector3(x サイズ, y サイズ, z サイズ);

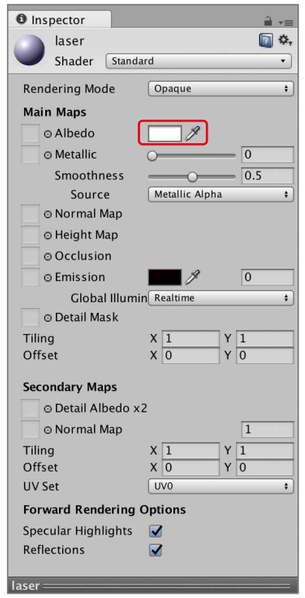
レーザーを発射できるようにする

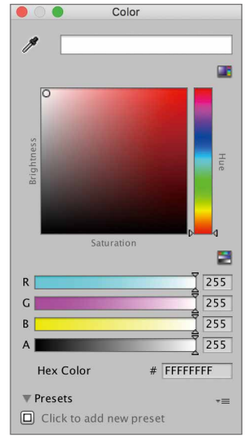
○Laser（レーザー）を作ります。
Capsule（カプセル）を作ります。
HierarchyのCreate → 3D Object → Capsuleとクリックします。
名前を、「Laser」とします。
Transform（トランスフォーム）のScaleのXを0.2、Yを2、Zを0.2にします。



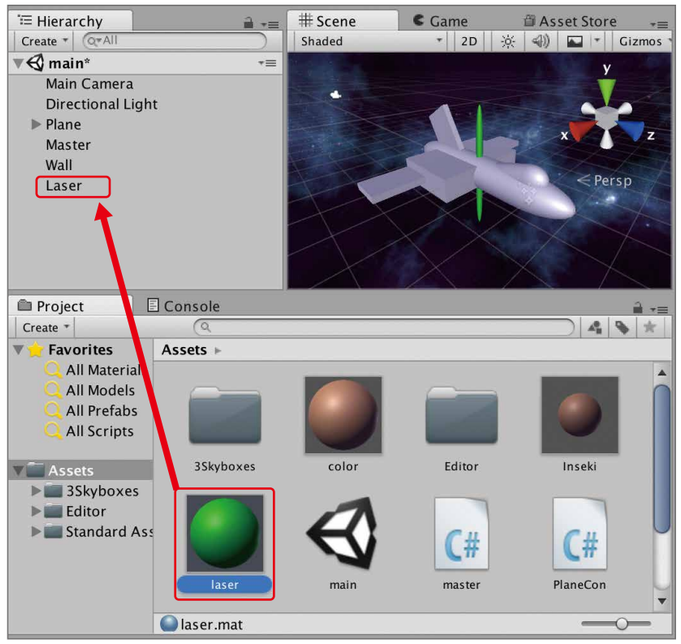
HierarchyのLaserをダブルクリックすると、飛行機の前方に弾らしきものが見えるはずです。

○Laserに色を付けます。
ProjectのCreate → Material（マテリアル：材質）とクリックします。
Assetsの中に新しいMaterialが出来ています。
名前を、「laser」（レーザー）とします。名前はなんでも構いません。
マテリアルlaserをクリックし、スポイトの横の枠をクリックして色を設定します。

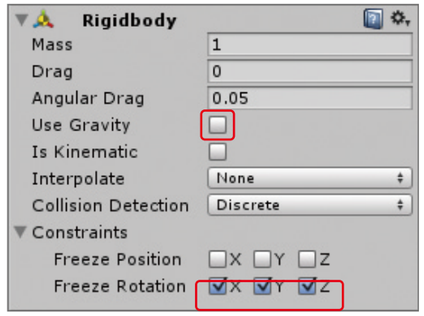




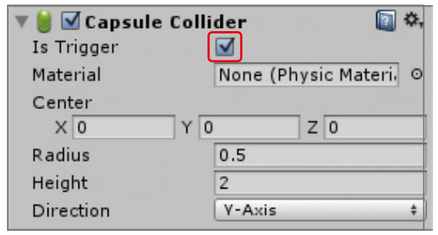
このマテリアルlaserを、HierarchyのLaserにDDします。



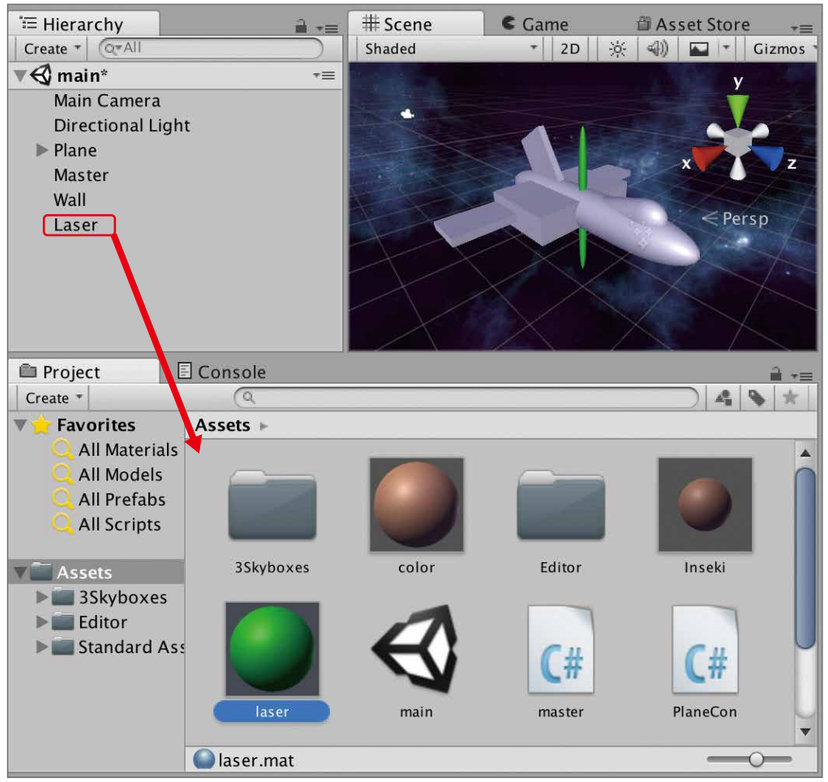
○オブジェクトLaserにRigidbody（リジッドボディ）をアタッチします。
HierarchyのLaserをクリックし、InspectorのAdd Componentをクリックします。
Physics（フィジックス：物理）　→ Rigidbody（リジッドボディ：剛体）とクリックします。
RigidbodyのUse Gravity（ユーズ・グラビティー）のチェックを外します。
これで、重力が作用しなくなります。
また、Constraints（コンストレイン）の左側の三角をクリックし、Freeze Rotation（フリーズローテーション）のX,Y,Zの全てにチェックを入れます。



○LaserのcolliderのIs Triggerのチェックを入れます。
HierarchyのLaserをクリックし、Capsule ColliderのIs Triggerにチェックを入れます。



○オブジェクトLaserをプレハブ化します。
分かり易いように、Assetsの直下にプレハブ化します。
ProjectのAssetsをクリックし、HierarchyのLaserをAssetsの中へDDします。



HierarchyのLaserを削除します。

○新しいスクリプトを作りPlaneにアタッチします。
スクリプト名は、「LaserGan」として、次のように編集します。

```
using UnityEngine;
using System.Collections;

public class LaserGan : MonoBehaviour {

    public GameObject laserPf;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        if(Input.GetKeyDown("space")) {
            GameObject laser = (GameObject)Instantiate
                (laserPf, transform.position, Quaternion.Euler(90f, 0f, 0f));
            laser.GetComponent<Rigidbody> ().velocity = new Vector3 (0f, 0f, 30f);
            Destroy (laser, 5f);
        }
    }
}
```

Unityに戻り、Planeをクリックして、InspecterのLaserPfの欄にプレハブのLaserを設定します。
再生してみましょう。スペースキーを押してレーザーが発射されたら成功です。

解説 スクリプトの解説

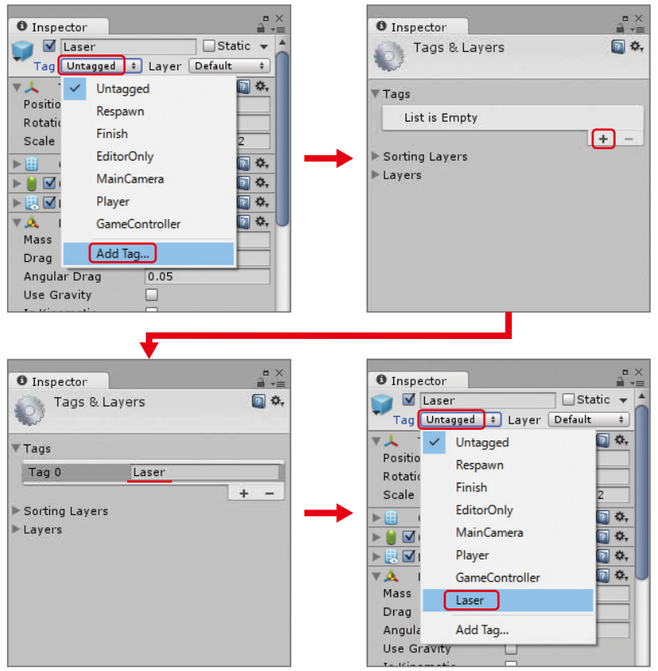
スペースキーを押した時にレーザーを生成する処理を書きました。
if文で、スペースキーを押した時に処理を分岐するようにしています。
分岐したときにInstantiate関数でレーザーを生成し、生成する座標は、Planeと同じ座標です。回転値は、X軸に90度です。
生成したレーザーにrigidbodyのvelocityに力を加えて飛ばします。
最後に生成したレーザーを5秒後に消すように設定しています。

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

    if(Input.GetKeyDown("space")) {
        GameObject laser = (GameObject)Instantiate
            (laserPf, transform.position, Quaternion.Euler(90f, 0f, 0f));

        laser.GetComponent<Rigidbody> ().velocity = new Vector3 (0f, 0f, 30f);

        Destroy (laser, 5f);
    }

}
```

座標はXとYはShip
回転値は、X軸に90度
力を加えて飛ばす
生成したレーザーを5秒後に消す

レーザーに当たった隕石が破壊されるようにする

○プレハブLaserに、Tag（タグ：名札）を付けます。
InspectorのTagの横のUntaggedをクリックし、 Add Tagをクリックします。
＋マークをクリックして、「Laser」というタグを新たに作ります。
Assetesの中のLaserをクリックし、Untaggedをクリックして、Laserを選択します。



○プレハブInsekiに、新しいスクリプトをアタッチします。
スクリプト名を、「Inseki」とし、次のように編集します。

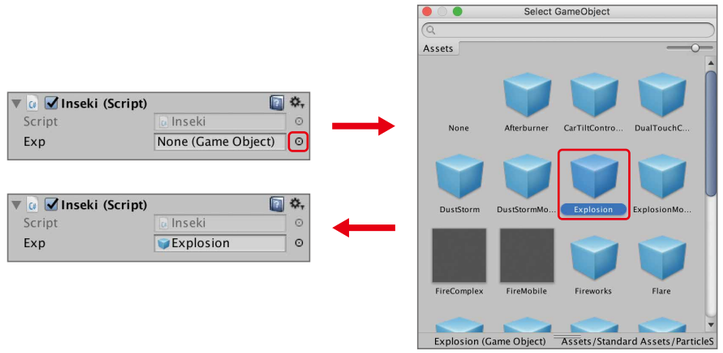
```
using UnityEngine;
using System.Collections;

public class Inseki : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }

    void OnTriggerEnter(Collider col) {
        if (col.gameObject.tag = "Laser") {
            Destroy (gameObject);
            Destroy (col.gameObject);
        }
    }
}
```

再生して見ましょう。レーザーと隕石が衝突したとき両方消えれば成功です。

○隕石の破壊を演出します。
スクリプトInsekiを次のように編集します。

```
using UnityEngine;
using System.Collections;

public class Inseki : MonoBehaviour {

    public GameObject exp;

    // Use this for initialization
    void Start () {

    }

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

    }

    void OnTriggerEnter(Collider col) {
        if (col.gameObject.tag == "Laser") {
            Destroy (gameObject);
            Destroy (col.gameObject);
            Instantiate (exp, transform.position, Quaternion.identity);
        }
    }
}
```

Unityに戻り、Assetsの中のプレハブInsekiをクリックします。
InspectorのInssekiコンポーネントからExpの右端の丸をクリックします。
Explosionプレハブをダブルクリックします。ExpにExplosionが入ったのを確認します。



解説 OnTriggerEnter（オントリガーエンター）とは

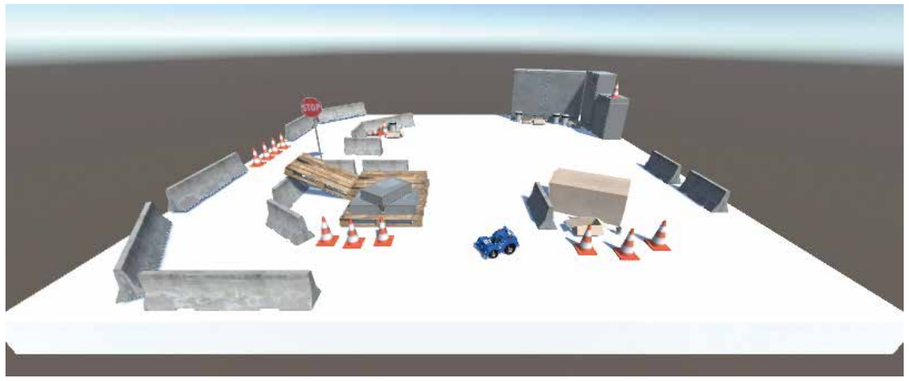
Colliderコンポーネントの Is Trigger にチェックが入っているものに衝突した時に処理されるものです。
(Colloder col) のcolには、衝突した相手のデータが入っています。col.gameObjectで相手のゲームオブジェクトを取得することができます。

解説 スクリプトの解説

OnTriggerEnter関数により何かしらのオブジェクトに衝突したときに処理され、その衝突相手がLaserで会った場合のみ処理するようにif文で分岐しています。
Destroy関数を２回使って自分と相手を削除し、Instantiateで変数expに入っている爆発を生成しています。

memo

RadioControl編

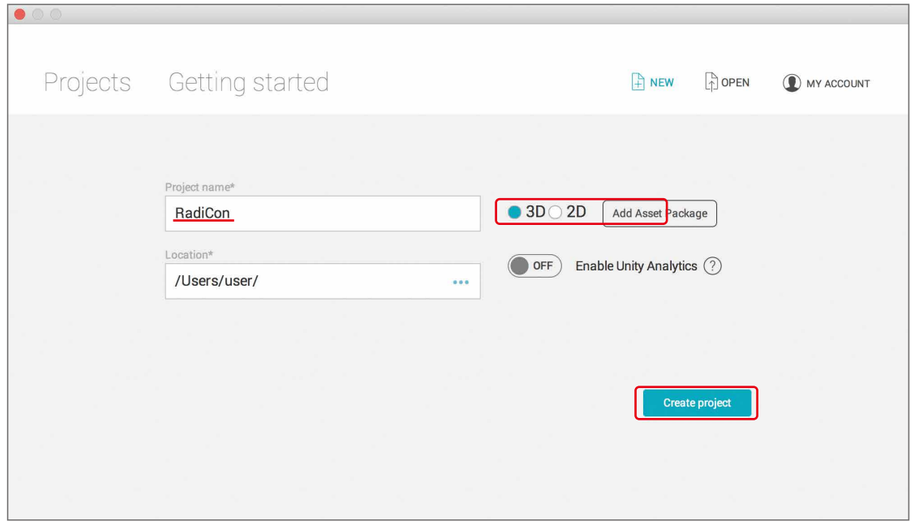


P123 プロジェクトを作る
P124 車の3Dモデルを入手する
P126 車を作る
P127 車を走らせる準備をする
P128 床を作る
P129 車を走らせる
P131 コントローラを作る
P132 曲がれるようにする
P134 摩擦抵抗を設定する
P135 カメラの位置を修正する
P138 障害物を設置する
P140 衝突判定を付ける
P143 Rigidbodyを付ける

車のラジコンを走らせるゲームを作ります。

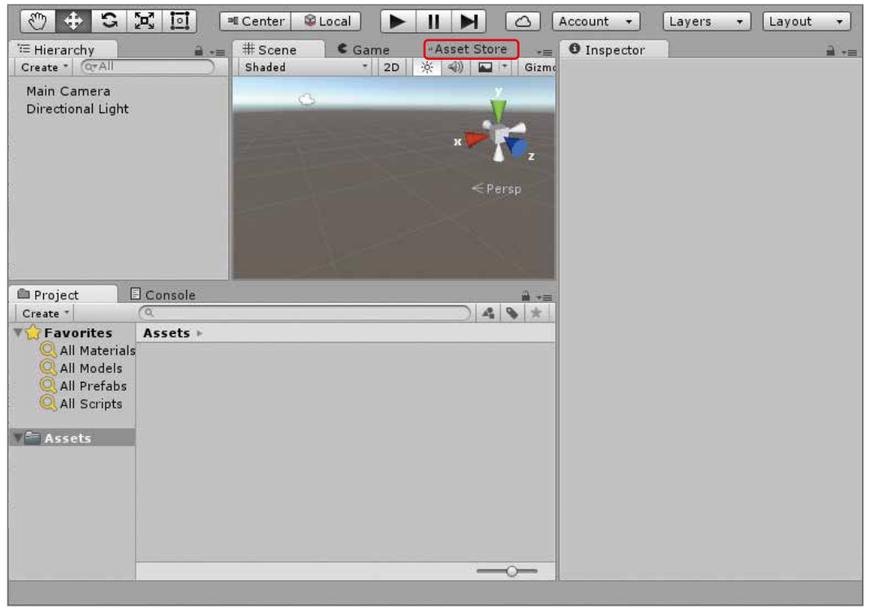
プロジェクトを作る

Project name（プロジェクトネーム）に「RadiCon」入力し、３Ｄにチャックを入れます。
Create projectをクリックしてプロジェクトを作成します。

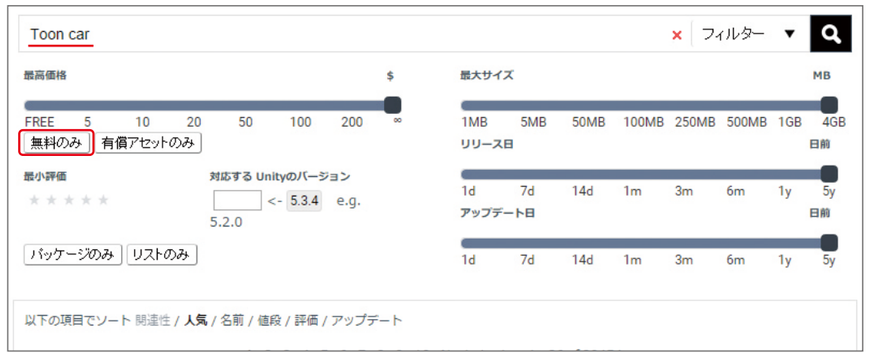


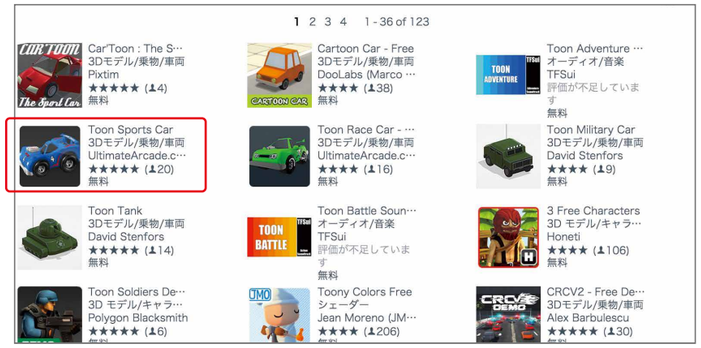
車の3Dモデルを入手する

Asset Storeから車の３Ｄモデルをダウンロードします。
画面中央上のAsset Storeタブをクリックします。

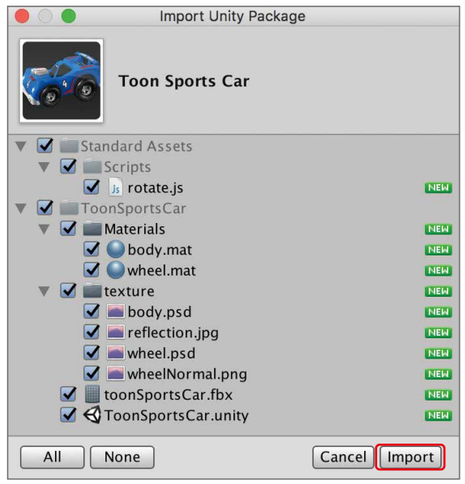


検索枠に「Toon Car」と入力して、無料のみをクリックします。
検索結果から「Toon Spots Car」を探してクリックします。



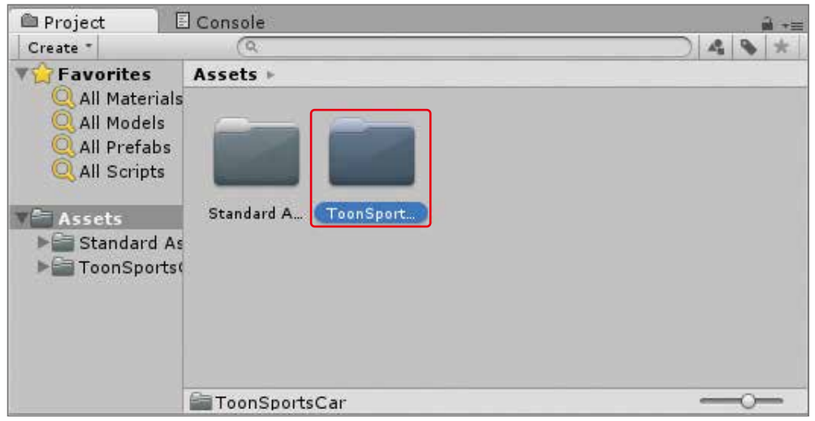


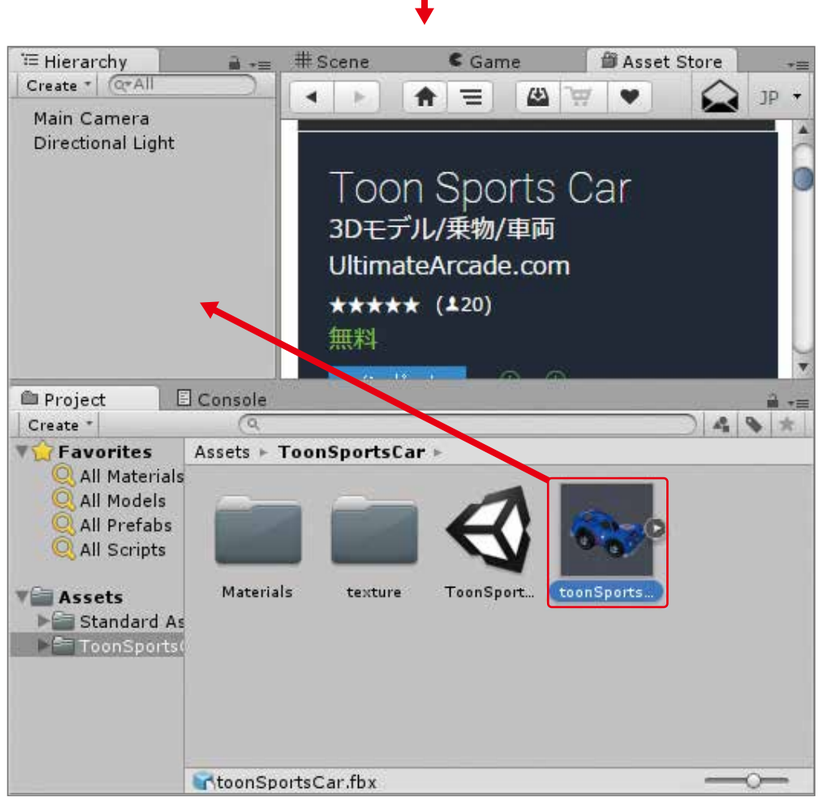
インポートをクリックします。
ダウンロードが開始され終わると、Import Unity Packageというウインドウが開くのでImportをクリックします。



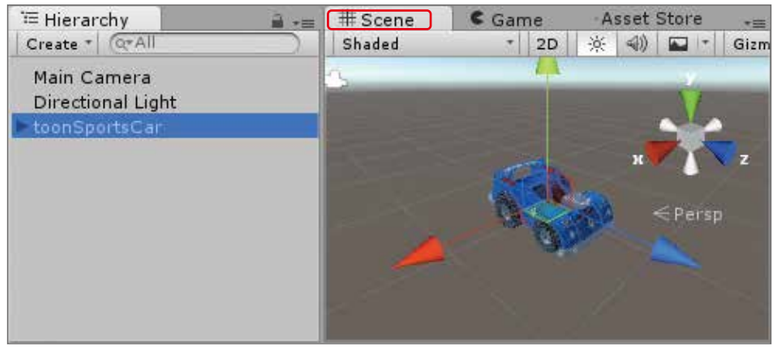
車を作る

車のモデルを手に入れたので、車を走らせます。
まずは、Scene上に車を設置します。
ToonSportsCarフォルダをダブルクリックして、toonSportsCarというモデルをHierarchy（ヒエラルキー）にDDします。





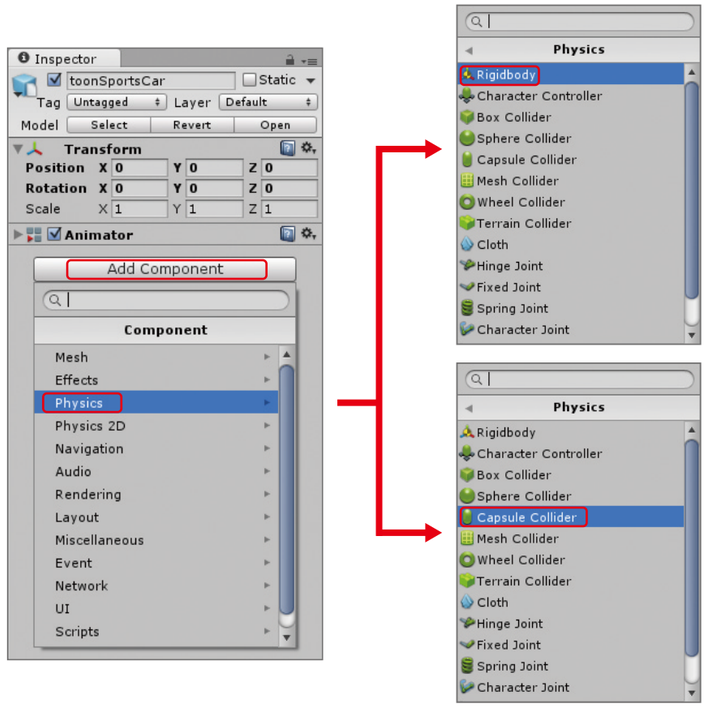
Sceneタブに切り替えて車が設置されてことを確認します。



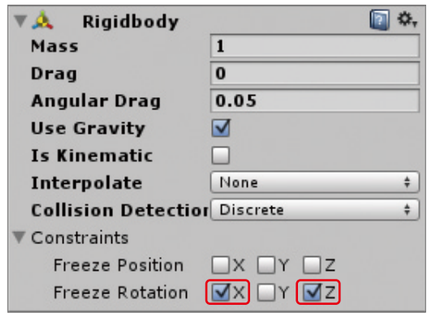
車を走らせる準備をする

今回は、Rigidbody（リジッドボディ）を利用して車を走らせます。そのために車にRigidbody（リジッドボディ）とCapsuleCollider（カプセルコライダー）を付けます。

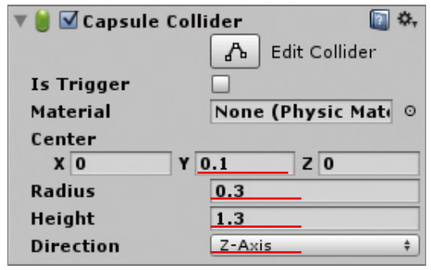
HierarchyからtoonSpotsCarをクリックして、InspectorのAddComponentをクリックします。
PhysicsからRigidbodyをクリックします。同じ手順でCapseuleColliderも付けます。



追加したRigidbodyのFreeze RotationのXとZにチェックをします。



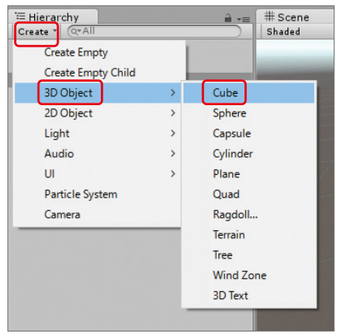
CapseuleColliderは、CenterのYを0.1にし、Radiusを0.3、Heightを1.3、DirectionをZ-Axisに設定します。



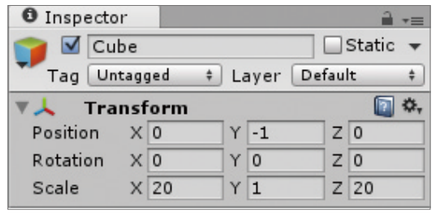
再生して車が落下するのを確認します。

床を作る

HierarchyのCreateボタンをクリックして、3DObjectからCubeをクリックして、Cubeオブジェクトを作ります。



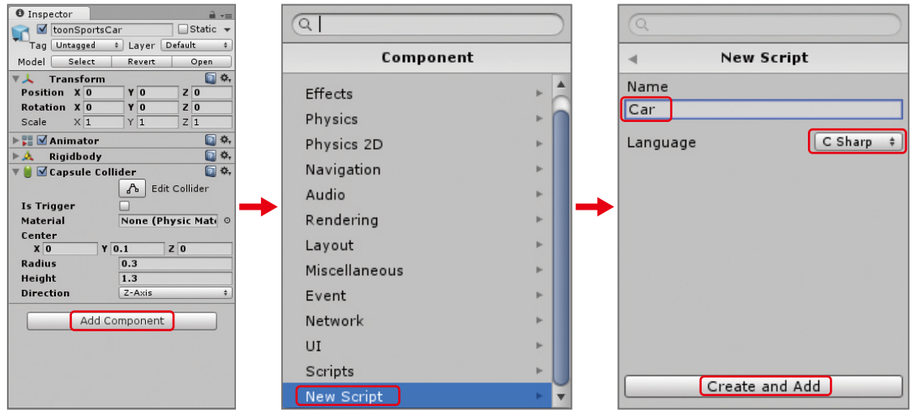
CubeのTransformのPositionのYを－1にしてSizeのXとZを20に設定します。



現状を保存しておきましょう。
Command＋Sキーまたは、左上のFileからSave Sceneを押してファイル名は「main」として保存します。

車を走らせる

車にスクリプトと付けます。
Hierarchy（ヒエラルキー）でtoonSportsCarを選択し、Inspecter（インスペクター）のAddComponentをクリック、New Scriptを選択します。名前は「Car」とし、言語は「CSharp」とします。



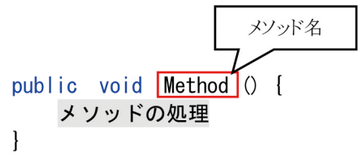
ProjectでAssetsを選択し、Carスクリプトをダブルクリックして開きます。
開いたら、以下のように記述します。

```
using UnityEngine;
using System.Collections;

public class Car : MonoBehaviour {

    public float speed = 10f;
    Rigidbody rig;

    // Use this for initialization
    void Start () {
        rig = GetComponent<Rigidbody> ();
    }

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

    }

    public void Forward() {
        rig.AddForce (transform.forward * speed);
    }

    public void Back() {
        rig.AddForce (transform.forward * speed * -1);
    }
}
```

再生しても何も変化ありません。
今回は、ラジコンという設定なので別のスクリプトからこのスクリプトに記述したものを呼び出し、遠隔操作をします。

解説 メソッド｜メンバ関数

メソッドとは、処理の一塊のことを指します。また、メンバ関数と呼ばれることもあります。
メソッドは、外部から呼出し操作行うものです。
先ほどのプログラムでは、Start()、Update()、Forward()、Back()がメソッドに当たります。
メソッドは以下のように記述します。

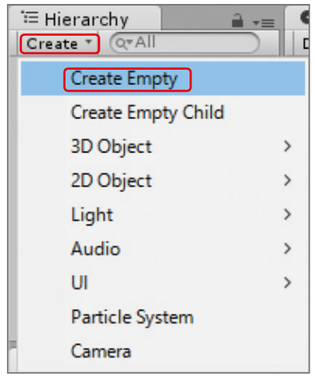


解説 スクリプトの解説

記述したForwardメソッドとBackメソッドを別のスクリプトから呼び出し、Forwardは、車を前進させ、Backは、車を後進させる処理です。
変数rigは、Start関数で車にアタッチされているRigidbodyを取得し代入しています。
ForwardとBackは、Rigidbodyに力を加える命令addForceメソッドで前進と後退をさせています。

コントローラを作る

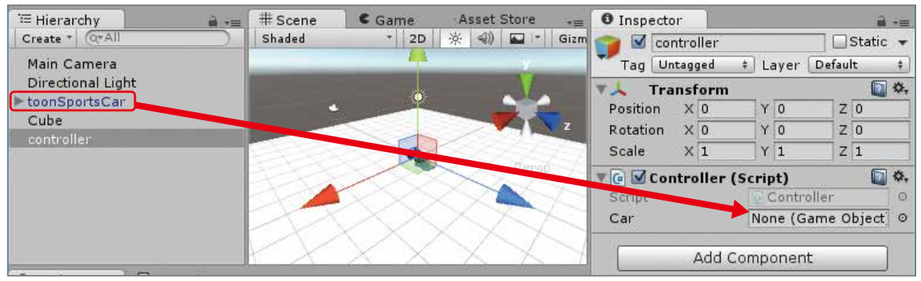
次は、遠隔操作をさせるコントローラを作ります。
Hierarchy（ヒエラルキー）のCreate（クリエイト）ボタンをクリックして、Create Empty（クリエイトエンプティ）をクリックします。名前を「controller」とします。



controllerオブジェクトに新規スクリプトを作成して付けます。
スクリプト名は、「Controller」とします。
Controllerスクリプトを以下のように記述します。

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class Controller : MonoBehaviour {
5
6     public GameObject car;
7
8     // Use this for initialization
9     void Start () {
10
11    }
12
13    // Update is called once per frame
14    void Update () {
15
16        if (Input.GetAxis ("Vertical") > 0) {
17            car.SendMessage ("Forward");
18        }
19
20        if (Input.GetAxis ("Vertical") < 0) {
21            car.SendMessage ("Back");
22        }
23    }
24 }
25
```

InspectorにCarという項目が追加された。そこにHierarchy（ヒエラルキー）のtoonSportsCarをDDしましょう。



解説 SendMessage関数

SendMessage（センドメッセージ）とは、GameObject型が持つメソッドでそのオブジェクトにアタッチされているスクリプトの中から指定した名前のメソッドを呼び出すものです。SendMessageでメソッドを呼び出すのは、Unity独特な手法です。今回は、スクリプトCarが持つForwardとBackメソッド呼び出しています。

曲がれるようにする

次は、曲がれるようにします。まずは、Carスクリプトを以下のように改変します。

```
4 public class Car : MonoBehaviour {
5
6     public float speed = 10f;
7     Rigidbody rig;
8
9     // Use this for initialization
10    void Start () {
11        rig = GetComponent<Rigidbody> ();
12    }
13
14    // Update is called once per frame
15    void Update () {
16
17    }
18
19    public void Forward() {
20        rig.AddForce (transform.forward * speed);
21    }
22
23    public void Back() {
24        rig.AddForce (transform.forward * speed * -1);
25    }
26
27    public void Handle(float hor) {
28        if (hor > 0) {
29            transform.Rotate (0, 2, 0);
30        } else if (hor < 0) {
31            transform.Rotate (0, -2, 0);
32        }
33    }
34
35 }
36
```

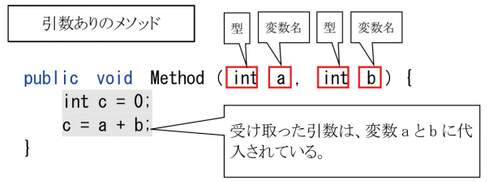

Controllerスクリプトも以下のように改変します。

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class Controller : MonoBehaviour {
5
6      public GameObject car;
7
8      // Use this for initialization
9      void Start () {
10
11     }
12
13     // Update is called once per frame
14     void Update () {
15
16         if (Input.GetAxis ("Vertical") > 0) {
17             car.SendMessage ("Forward");
18         }
19
20         if (Input.GetAxis ("Vertical") < 0) {
21             car.SendMessage ("Back");
22         }
23
24         car.SendMessage ("Handle", Input.GetAxisRaw ("Horizontal"));
25     }
26 }
27
```

再生して矢印キー左右で回転すれば成功です。

解説 メソッドの引数

引数とは、パロメータとも呼ばれ、処理の材料となるデータのことです。
メソッド宣言するときにカッコ内に変数宣言のように書いてでデータ型と数を指定します。
また、引数を使わない場合は、カッコ内に何も書きません。



解説 SendMessageで引数を渡す

SendMessage関数でメソッドを呼ぶときに呼び出すメソッドに1つだけ引数を渡すことが可能です。
SendMessage関数の引数は、一つ目に呼び出すメソッド名、二つ目に呼び出すメソッドに渡す引数を書きます。
GameObject型の変数.SendMessage(メソッド名, 引数);

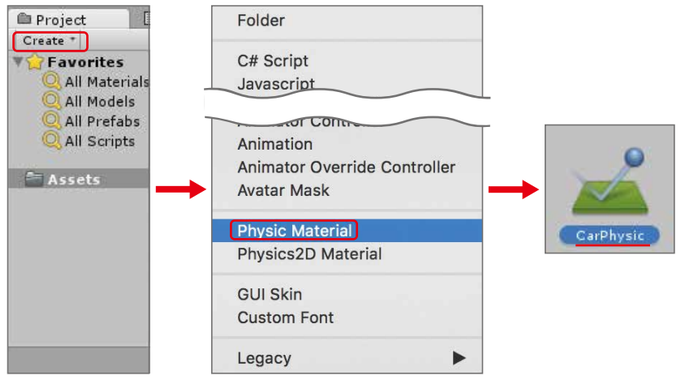
解説 スクリプトの解説

Car スクリプトで追記したHandleメソッドは、引数としてfloat型を受け取ります。
このメソッドは、受け取る値がInput.GetAxisであるという前提で書かれており、受け取ったデータが入った変数horの値をもとに左右回転する処理です。

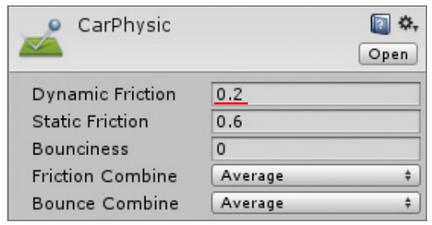
Controllerスクリプトで追記した命令文は、Handleメソッドを呼び出し、引数としてInput.GetAxisRaw("Horizontal")の値を渡しています。

摩擦抵抗を設定する

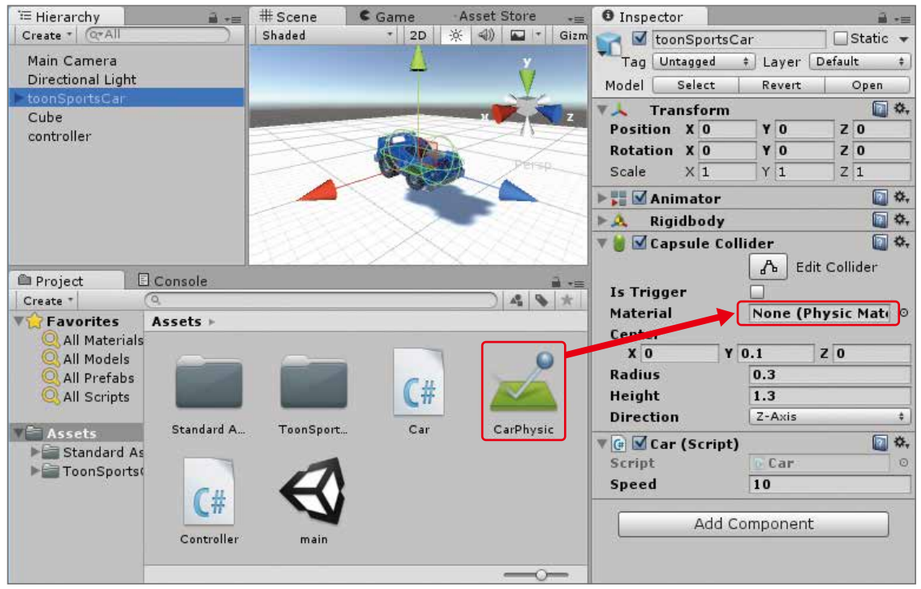
車の速度が遅いです。InspectorのSpeedを変更するのも一つの手ですが、今回は、摩擦抵抗を設定して調整します。
HierarchyのCreateからPhysic Materialをクリックします。名前は、分かり易く「CarPhysic」としておきます。



動いているときの摩擦抵抗を示すDynamic Frictionを0.2に設定します。



HiearchyからTroonSportsCarを選択してInspectorのCapsuleColliderコンポーネントのMaterialにCarPhysicをDDします。



再生して動きを確認しましょう。
もし動きが気に入らなかったら、SpeedやDynamic Frictionを変更して調整してみましょう。

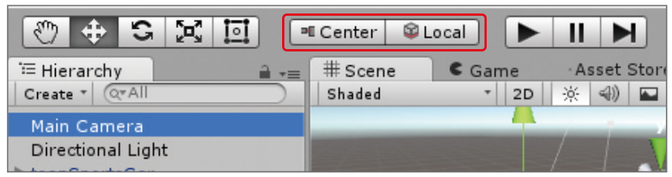
解説 Physic Materialのパラメータ

Dynamic Friction （ダイナミック フリクション）	移動中の摩擦抵抗。0 ～ 1 の間の値を使用する。値が小さいほどよく滑る。
Static Friction （スタティック　フリクション）	停止中の摩擦抵抗。0 ～ 1 の間の値を使用する。値が小さいほどよく滑りやすく、小さな力で動かせる。
Bounciness （バウンシーケンス）	跳ね返し度合。0 ～ 1 の間の値で使用する。0 の場合、跳ね返らない。1 の場合、力が衰えることなく跳ね続けます。
Fiction Combine （フリクション コンバイン）	衝突したオブジェクト同士の摩擦の値の計算。 - Average　2 つの摩擦力が平均化される。 - Minimum　2 つの摩擦力のうち小さい方の値が使用される。 - Maximum　2 つの摩擦力のうち大きい方の値が使用される。 - Multiply　2 つの摩擦力が互いに乗算される。
Bounce Combine （バウンス コンバイン）	衝突したオブジェクト同士の跳ね返し度合の値の計算。

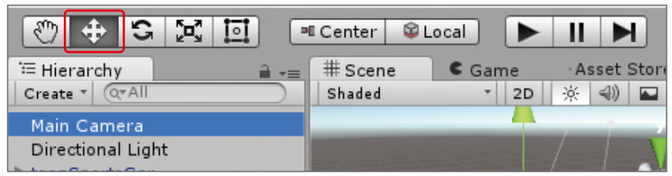
カメラの位置を修正する

見にくいのでカメラの位置を変更します。
HierarchyからMain Cameraをダブルクリックします。するとScene上でMain Cameraを注視します。
Shift + F キーでも同じことができます。

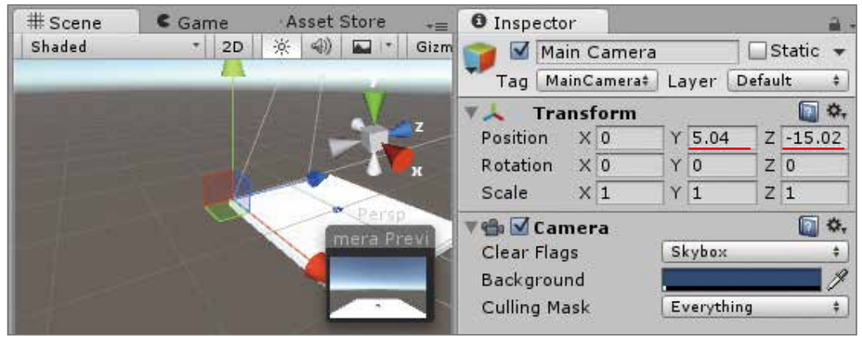
再生ボタンの左側が「Center」と「Local」にします。それぞれクリックして変更できます。



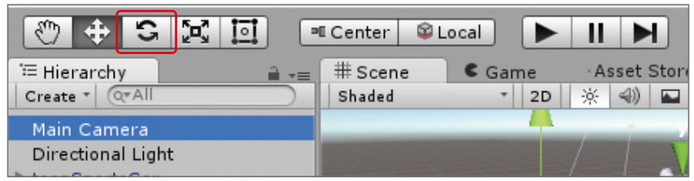
その左側、十字矢印（移動ツール）のボタンをクリックします。



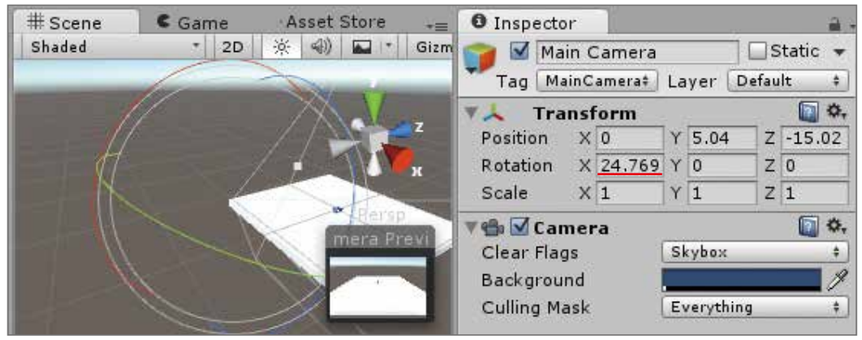
三色の矢印が表示されます。
青色をドラッグしてPositionのZが-16～-14くらいになるまで移動させます。
同様に緑色をドラッグしてPositionのYが4～6くらいになるまで移動させいます。



次に矢印の円（回転ツール）のボタンをクリックします。



三色の円が表示されます。
赤い円をドラッグしてRotationのXを25くらいになるまで回転させます。



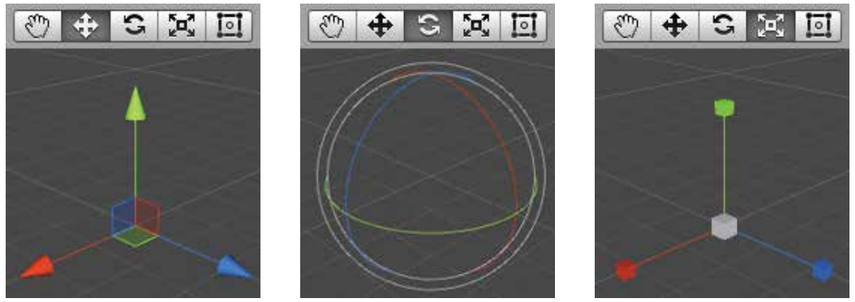
解説 トランスフォームツール

トランスフォームツールとは、オブジェクトの座標や大きさを直感的に操作しTransfromの値を変更するツールです。

■移動ツール
Position（座標）変更する。
矢印をドラッグするとその矢印と同じ直線上で移動させることができ、矢印の根元にある面をドラッグするとその面に平行に動かすことができる。

■回転ツール
Rotation（回転値）を変更する。
円をドラッグするとその円に沿った回転をします。球をドラッグすると自由に回転して直感的です。

■拡大ツール
Ｓｃａｌｅ（大きさ）を変更する。
赤がＸ、緑がＹ、青がＺのスケールに対応している。真ん中の灰色ををドラッグするとＸＹＺすべて拡縮する。



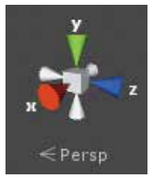
左から、移動ツール、回転ツール、拡大ツール

解説 シーンのカメラ操作

Scene上の視点操作についてです。これを知っているとScene上での作業が楽になります。覚えておきましょう。
右ドラッグをすると、カメラの向きを変更できます。
Alt＋左ドラッグで注視している場所を中心に回転します。
Alt＋右ドラッグでカメラを前後移動させることができます。

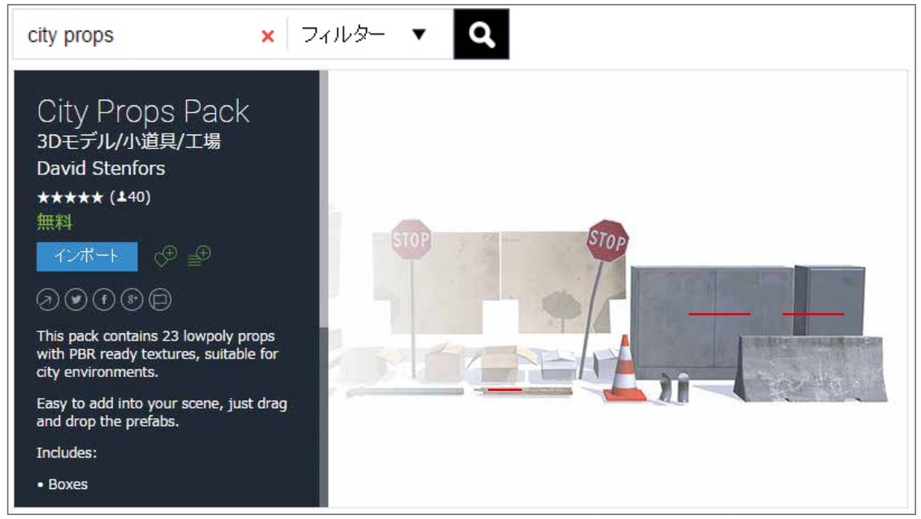
解説 シーンギズモ

矢印をクリックするとその方向に視点が移動します。またアイコンの下の文字をクリックすると平行投影モードに変更されます。もう一度クリックすると戻ります。
平行投影モードとは、奥行感がなく平面的なモードです。平面的なものの操作や、コライダーの細かい設定をするときに役立ちます。

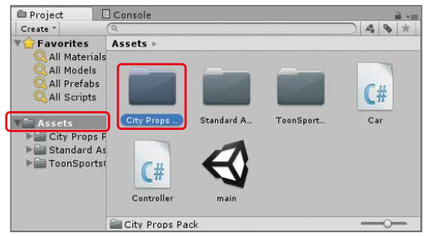


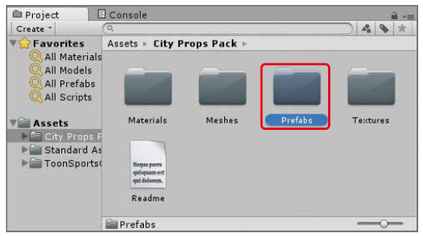
障害物を設置する

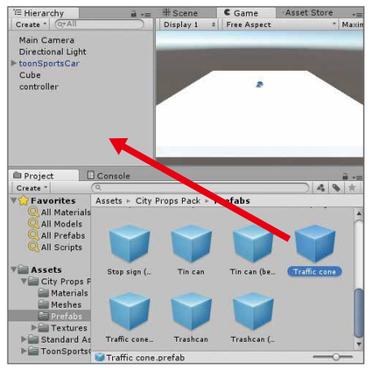
アセットストアから「City Props Pack」をインポートします。
検索ボックスに「city props」と入力、無料をクリックして検索します。検索結果から「City Props Pack」を探しインポートします。



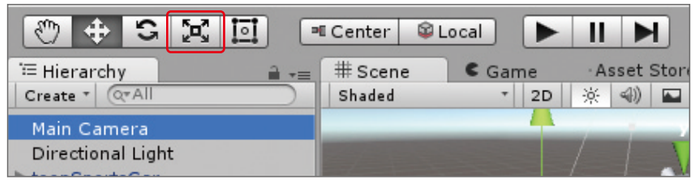
○モデルを出す
ProjectからAssetsをクリックします。
City Props Packフォルダの中のPrefabsフォルダをクリックして開きます。
この中から一つ好きなものを選びHierarchyにDDします。



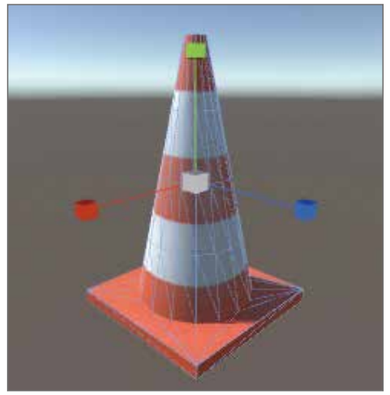


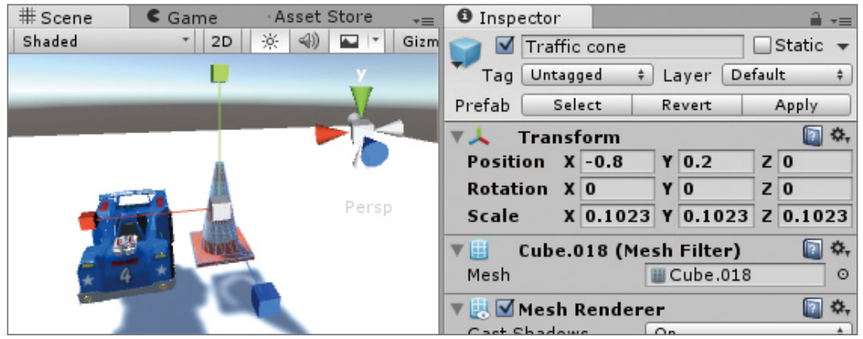


○サイズを変更する
大きいのでサイズを変えます。
HierarchyからDDしたオブジェクトを選択し、ダブルクリックまたは、Shift＋Fキーを押してScene上で注視します。
左上のトランスフォームツールから矢印のバッテン（拡大ツール）を選択します。



赤緑青の四角と、灰色の四角が出てきました。灰色の四角をドラッグしてお好みの大きさに調整します。





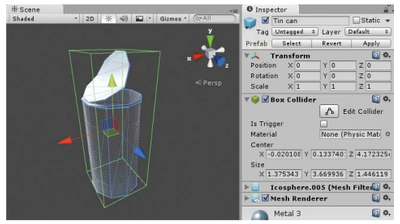
衝突判定を付ける

このままでは、すり抜けてしまうので衝突判定であるCollider（コライダー：衝突装置）を付けます。ここで使うColliderは、４種類です。モデルの形に合わせて使い分けます。また、複数付けることが可能で組み合わせて複雑な形を作ることもできます。

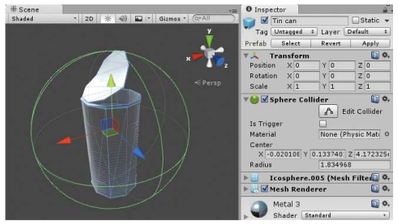
Add　Componentをクリックし Physics（フィジカル：物理）から使いたいColliderを選択し、アタッチします。InspectorからアタッチしたColliderのCenterやSizeなどのパラメータを変更し調整します。パラメータについては、下記の（解説）を見てください。

解説　4種類のCollider（コライダー）

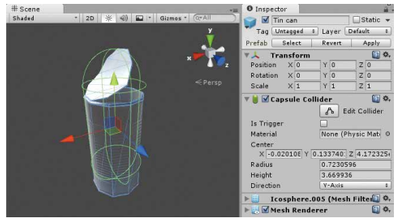
箱型　Box Collider（ボックス コライダー）



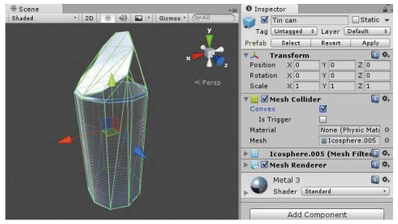
球体　Sphere Collider（スフィア コライダー



カプセル型
Capsule Collider（カプセル コライダー）



モデルに合わせて形を変える
Mesh Collider（メッシュ コライダー）



緑の線がColliderの持つ衝突判定です。

解説　Box Collider（ボックス コライダー）のパラメータ

Is Trigger （イズ トリガー）	有効にすると壁としての判定がなくなります。有効の場合、このコライダーに接触した際の処理をOnTrigger関数等で定義することができる。 無効にすると壁として機能します。無効の場合、このコライダー接触した際の処理をOnCollider関数等で定義することができる。
Material （マテリアル）	Physic　Materialの関連付け。他のコライダーと衝突した時の摩擦や跳ね返しなどの条件を定義する。
Center（センター）	コライダーの中心位置。
Size（サイズ）	コライダーの大きさ。（ＸＹＺの大きさ）

解説　Sphere Collider（スフィア コライダー）のパラメータ

Is Trigger （イズ トリガー）	有効にすると壁としての判定がなくなります。有効の場合、このコライダーに接触した際の処理をOnTrigger関数等で定義することができる。 無効にすると壁として機能します。無効の場合、このコライダー接触した際の処理をOnCollider関数等で定義することができる。
Material （マテリアル）	Physic　Materialの関連付け。他のコライダーと衝突した時の摩擦や跳ね返しなどの条件を定義する。
Center（センター）	コライダーの中心位置。
Radius（ラジウス）	コライダーの大きさ。（半径）

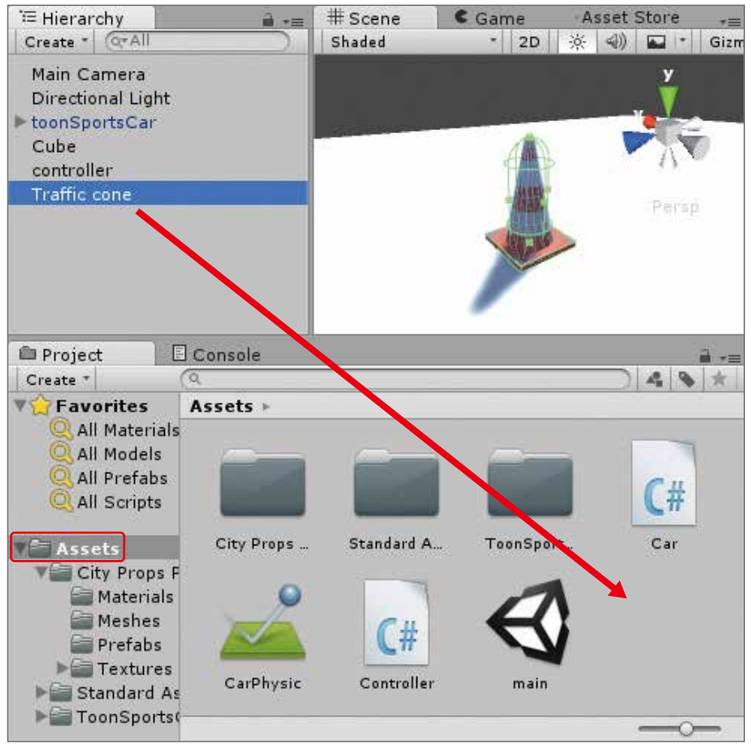
解説　Capsule Collider（カプセル コライダー）のパラメータ

Is Trigger （イズ トリガー）	有効にすると壁としての判定がなくなります。有効の場合、このコライダーに接触した際の処理をOnTrigger関数等で定義することができる。 無効にすると壁として機能します。無効の場合、このコライダー接触した際の処理をOnCollider関数等で定義することができる。
Material （マテリアル）	Physic　Materialの関連付け。他のコライダーと衝突した時の摩擦や跳ね返しなどの条件を定義する。
Center（センター）	コライダーの中心位置。
Radius（ラジウス）	コライダーの大きさ。（円柱の半径）
Height（ハイト）	コライダーの高さ。
Direction（ディレクション）	長辺方向の向き。

解説　Mesh Collider（メッシュ コライダー）のパラメータ

Is Trigger （イズ トリガー）	有効にすると壁としての判定がなくなります。有効の場合、このコライダーに接触した際の処理をOnTrigger関数等で定義することができる。 無効にすると壁として機能します。無効の場合、このコライダー接触した際の処理をOnCollider関数等で定義することができる。
Material （マテリアル）	Physic　Materialの関連付け。他のコライダーと衝突した時の摩擦や跳ね返しなどの条件を定義する。
Mesh（メッシュ）	衝突判定に使用するメッシュ（モデル）。
Convex （コンベックス）	有効にした場合、他のメッシュコライダーと衝突する。ただし、衝突判定は、実際のモデルより大まかになる。 無効状態ではシーン上で緑の線が表示されませんが衝突判定は、存在します。

Colliderを設定したオブジェクトをプレハブ化します。
ProjectのAssetsをクリックし、Hierarchyから先ほどのオブジェクトを選択しProjectの右側にDDします。



作成したプレハブをHierarchyやSceneにDDすると複製できます。

地面の上に設置して自分だけのフィールドを作りましょう。
必要ならば他のモデルもコライダーの設定とプレハブ化をしましょう。作業中はCommand+Sでこまめにセーブすることをお勧めします。

Rigidbodyを付ける

一部オブジェクトは、押せるようにします。
設置したオブジェクトに、一つ一つにRigidbodyを付けるのは面倒です。そこで、プレハブに付けます。
プレハブを変更すると、そのプレハブから複製したすべてのオブジェクトに適用されます。

Projectの先ほど作成したプレハブを選択します。InspectorからApp Componentをクリックします。
PhysicsからRigidbodyを選択します。

再生してRigidbodyが設置したオブジェクトに適用されていて、落下したり押せたりできれば成功です。
車とそれ以外のオブジェクトの重量が同じで、挙動が不自然です。RigidbodyのMass（マス：重さ）などを調整して完成です。

解説 Rigidbody（リジッドボディ）のパラメータ

Mass（マス）	物体の質量。単位はkg
Drag（ドラッグ）	空気抵抗の大きさ。空気抵抗が小さいほど空中で減速しにくい。
Angular Drag （アンギュラ ドラッグ）	回転する際の空気抵抗の大きさ。
Use Gravity （ユーズ グラビティ）	有効にすると、重力の影響を受けます。 無効にすると、重力の影響を受け無くなり、無重力状態になります。
Is Kinematic （イズ キネマティック）	有効にすると、物理計算を行わなくなります。 他のオブジェクトが衝突しても影響を受けることはありません。
Interpolate （インターポレート）	Rigidbodyの動きがぎこちないとき、変更して試してください。 ■None 補間を適用しない。 ■Interpolate 前フレームの Transform にもとづいてスムージング。 ■Extrapolate 次フレームの Transform を予測してスムージング。
Collision Detection （コリジョン ディテクション）	高速で動いた際、コライダーをすり抜けることを防止します。
Constraints （コンストレーン）	リジッドボディの動きに関する制限。 ■Freeze Position ワールド座標系の XYZ 軸で移動するリジッドボディを選択的に停止します。 ■Freeze Rotation ワールド座標系の XYZ 軸で回転するリジッドボディを選択的に停止します。

memo