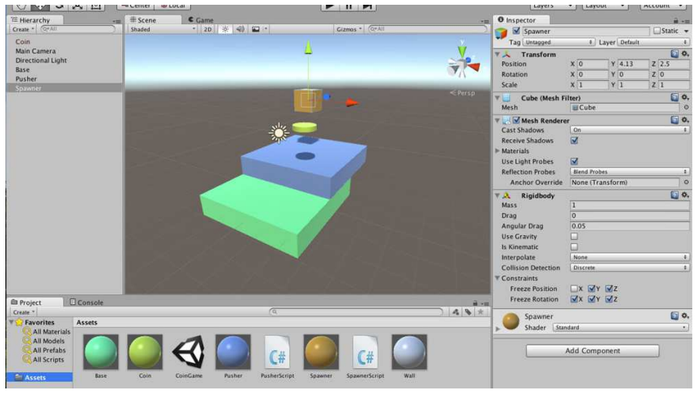
１ページずつ学ぶ unity Ver 2021



Unity とは、ユニティ・テクノロジーズ社が提供するゲーム開発プラットフォームです。これは描画、サウンド再生、UI(ユーザーインターフェイス)管理、データ作成，管理などの要素を揃えたゲームを作るための統合開発環境のことです。
2005 年の発表以来、日本を含む世界中で開発者が増え続け、現在では 100 万人を超える人々が使用しています。

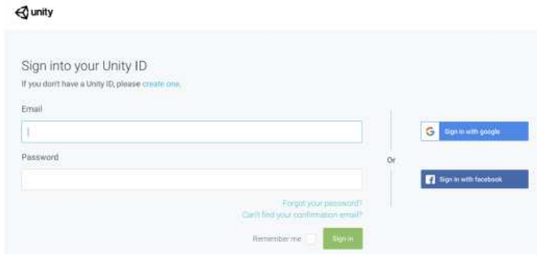
iOS や Android のようなスマホ向けアプリ、Windows、MacOS、Linux 向けのデスクトップアプリ、さらに PlayStation や Xbox、WiiU などの家庭用ゲーム機、アーケードゲーム、パチスロまで、非常に幅広いプラットフォームに向けたゲームやツールを開発でき、さらに、ゲームのみならず建築、医療、自動車、教育、プロジェクションマッピングなどさまざまな分野で活用されるようになってきています。

Unity のすごいところは個人での使用はライセンス料がかからず全て無料で開発ができ、ノンコーディングで 3D キャラクターを動かしたり、ゲームステージを設置したり、専門知識がなくても物理エンジンをすぐに導入できたりすることだろう。この手軽さが従来の「複雑で困難」とされてきた本格的な 3D ゲーム開発のハードルを下げ新たなゲーム開発者を次々と生み出しました。また、3D モデルやテクスチャ、マテリアルはもちろん、画面演出に使うパーティクルシステムや音楽、効果音などのゲーム開発に必要な素材が手軽に入手可能になっています。

実際に Unity で作られたゲームとしては 2016 年にリリースされたポケモンを捕まえるゲーム「Pokemon GO」、任天堂の「Super Mario Run(スーパーマリオラン)」、人気アクション RPG ゲーム「白猫プロジェクト」、元々はプレイステーションのソフトとして発売されたゲームがスマホでも遊べるようになった、スクエア・エニックスの「ドラゴンクエストⅧ」などがあります。

インストールとアクティベーション

授業ではすでにＵｎｉｔｙインストールされている状態から学習を始める。
基本的に制限なく使用できるが、アセットストアを使用するためにはアカウントの登録が必要となるため、個人のメールアドレスが必要となる。



例題１　起動とプロジェクトの作成

プロジェクト名　prog1

Unity を起動すると「Unity HUB」という管理用のアプリケーションが立ち上がる。
「新規作成」をクリックし、プロジェクト名と保存先フォルダ、テンプレートを指定する。
（今回は「２Ｄ」を選択）

プロジェクト名（基本的に半角英数字）

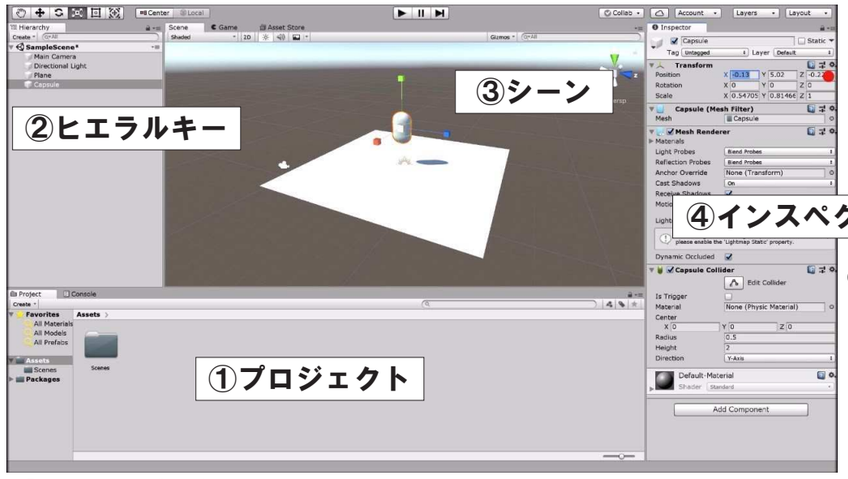
保存先フォルダ（自身のネットワークフォルダ）

「作成」ボタンをクリックすると少し時間がかかるが「テンプレート」が作成され編集画面が起動します。
※バージョンにより画面は異なる場合があります。

Unity の画面構成

②Hierarchy（ヒエラルキー）ウインド
ゲームの世界に配置したオブジェクト（物体）をツリー（階層）表示します。

③Scene（シーン）ビュー
ゲーム画面に表示する画像やボタンや文字などの部品を配置します。実行時には**ゲームビュー（Game）**として切り替わり、プレイ時に見える画面が表示されます。



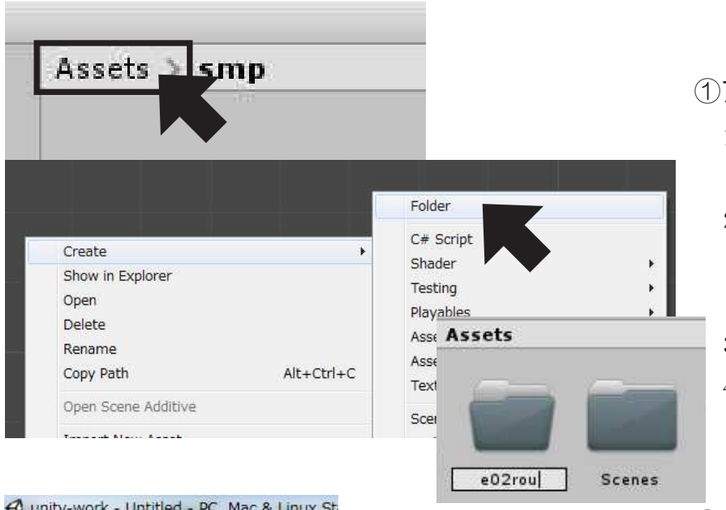
④ Inspector ウインド（インスペクター）
各オブジェクトの詳細な設定をを行います。

①Project（プロジェクト）ウインド
スクリプトや画像など、ゲームで使用するファイルが表示されます。
実行時には**Console（コンソール）ウインド**として機能しスクリプトにエラーがあったときとかにエラーメッセージが表示されます。

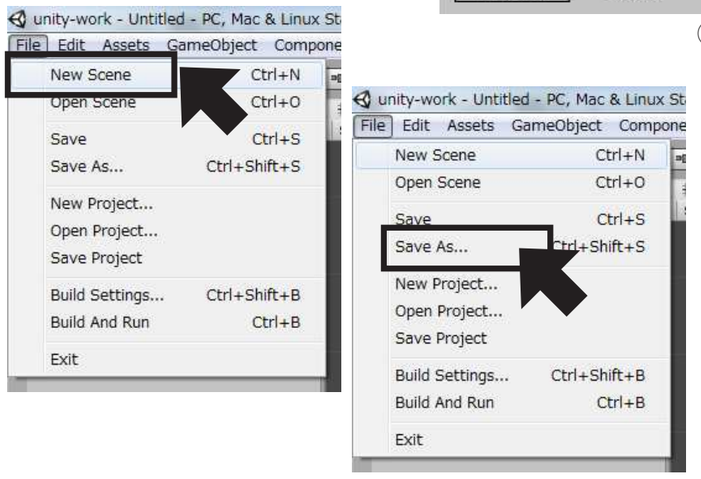
プロジェクトの作成は大掛かりで、最初の設定にも時間がかかります。この冊子の例題、演習はとても小さなものなので、そのたびに新たなプロジェクトを作成していると非効率です。そこで今回は、１つ１つの例題､演習をひとつの「シーン」として作成し､学習を進めていきます。それぞれに使用するアセット（素材）はシーンごとにフォルダを作って管理していくことにします。

重要　新しいシーン（課題・例題）の追加方法

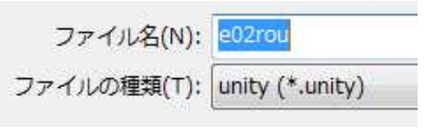
シーン名　r01ball

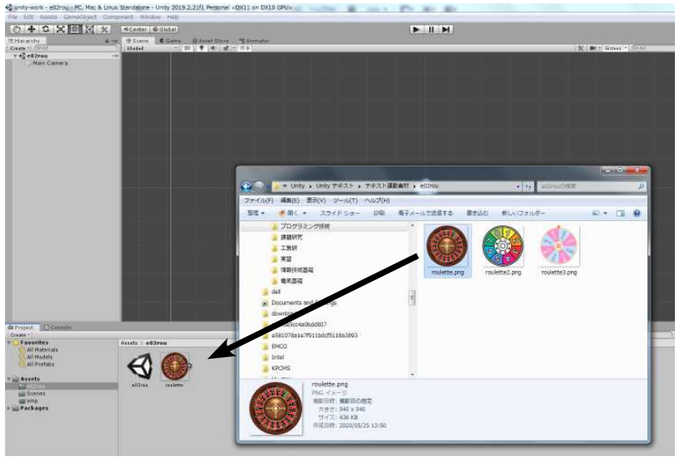


①フォルダの作成
1）プロジェクトビューの「Assets」で左クリック
2）プロジェクトビューで右クリック→メニューから「Create」→「Folder」選択
3）フォルダの名前を変更
4）作成したフォルダをダブルクリックして開く



②シーンの保存
1）「File」→「New Scene」
新しいシーンが作られます
2）「File」→「Save As」選択
保存先に作成したフォルダを指定し、シーン名をつけて保存する。

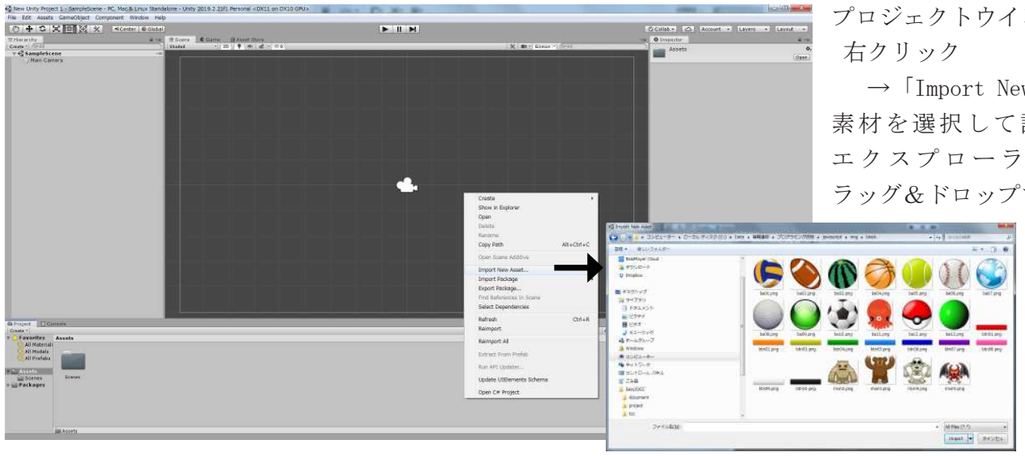




③アセット（部品）の登録
プロジェクトウインドの、作成したフォルダに画像やスクリプトを登録しシーンを作成していきます。

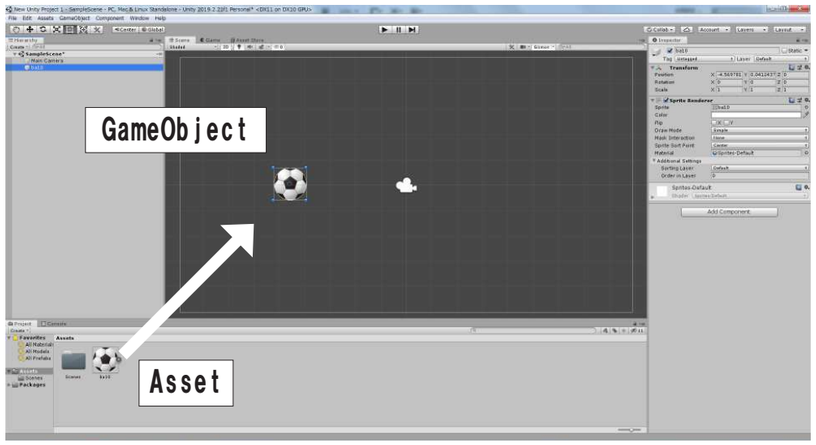
※１つのプロジェクト内で同じ名前のクラスは使用できないため、スクリプト名の先頭には「シーン名」を付ける事にします。

プロジェクトにアセット（資産）を追加する



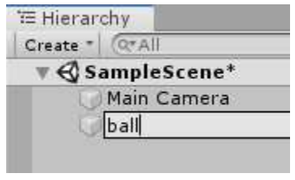
プロジェクトウインドで
右クリック
→「Import New Asset」
素材を選択して読み込む。
エクスプローラーからドラッグ＆ドロップでも可

ヒエラルキーウインド



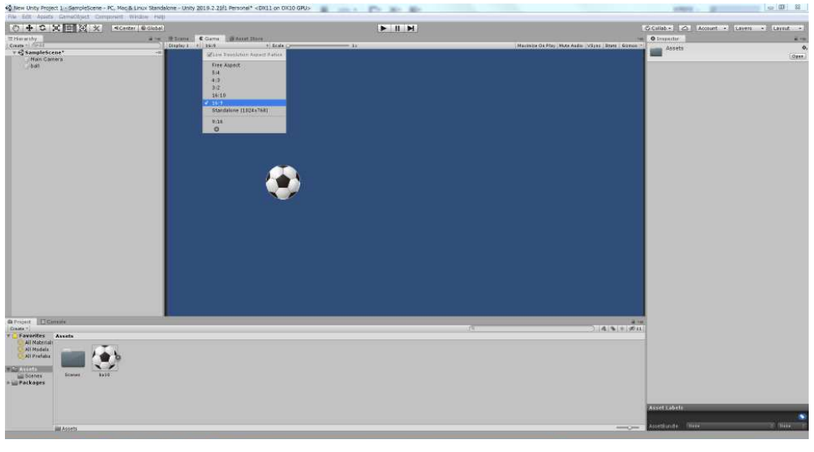
読み込んだアセットをプロジェクトウインドからシーンへドラッグ＆ドロップする。アセットがオブジェクト化（GameObject）されます。

ヒエラルキーウインドで分かりやすい名前「ball」に変更しておきます。

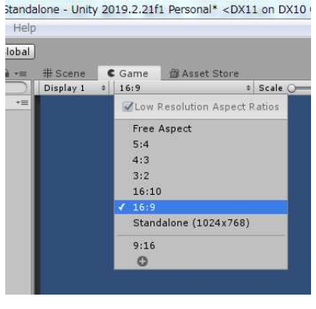


※画像は「スプライト」と呼ばれる形で処理されます

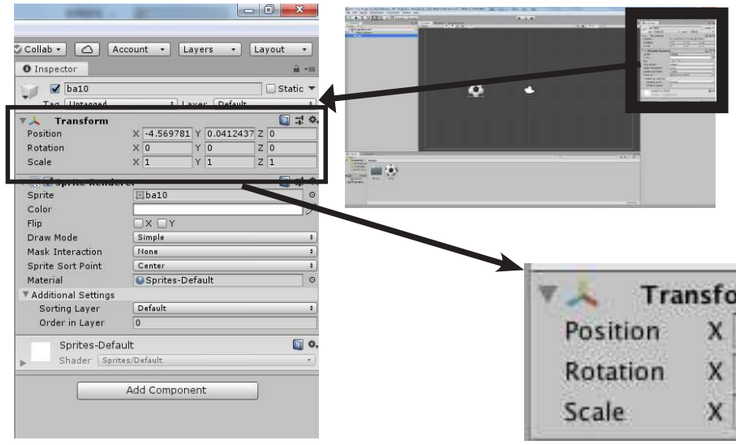
ゲームビューと画面サイズの設定



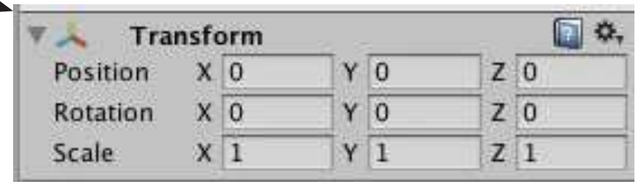
ゲームビューに切り替えて画面サイズを
「16:9」に設定する。



インスペクターウインド

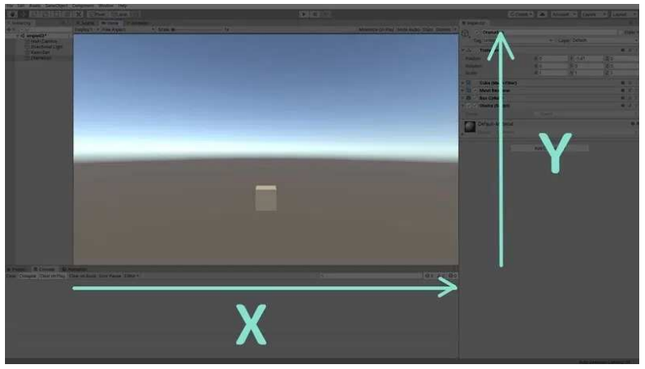


配置した素材（ゲームオブジェクト）の情報を設定するインスペクターウインドを確認する。
「Transform」の
「Position」
「Rotation」
「Scale」の値をを変更してみる

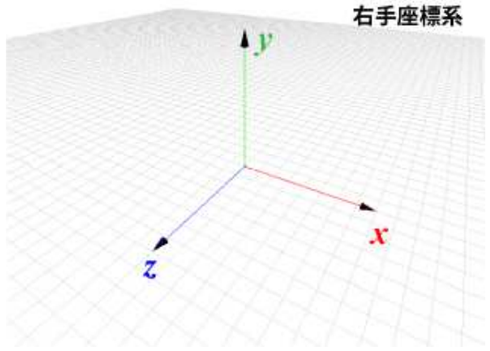


Unity 画面の座標

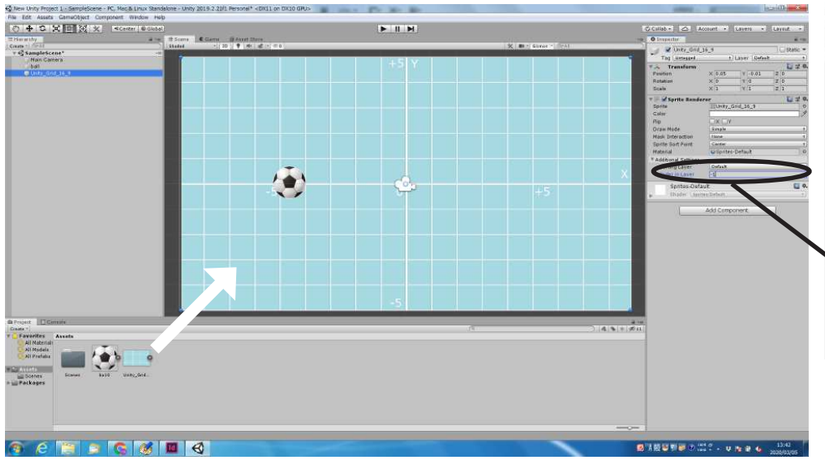
２Ｄ

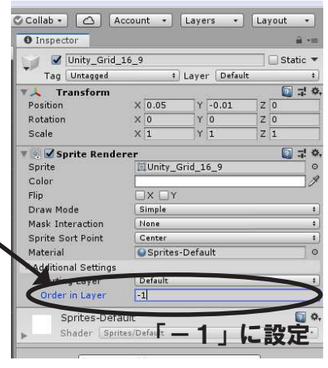


３Ｄ

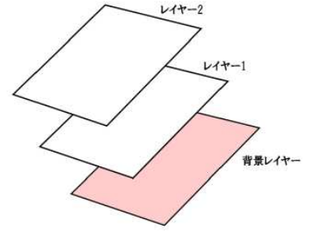


オブジェクト（GameObject）の追加と重なり順の確認

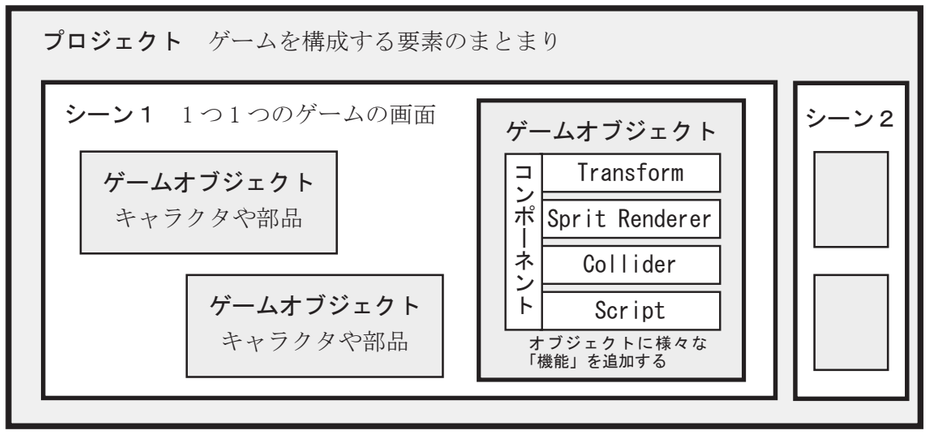




座標の確認をするために、座標軸が描かれた画像「Unity_Grid_16_9.png」をインポートします。先に読み込んだ画像が下に隠れてしまうので、インスペクターの**「Order in Layer」の値を「-1」**に設定します。
「Transform」の各値を変更して、２Ｄ座標の理解を深めましょう。



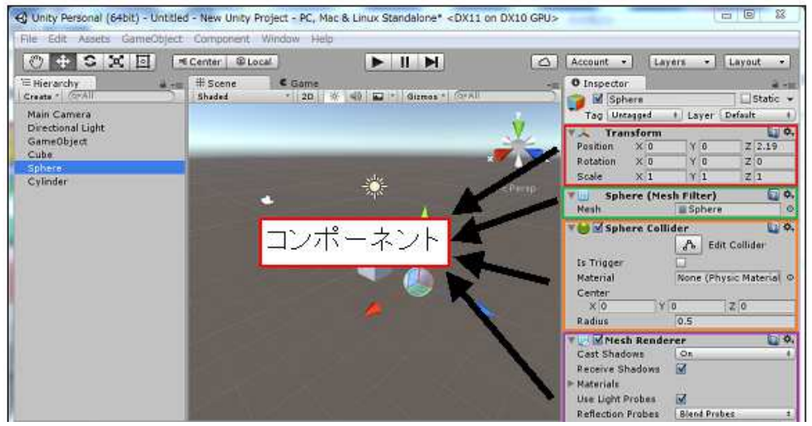
Unity の構成要素



用語の直訳

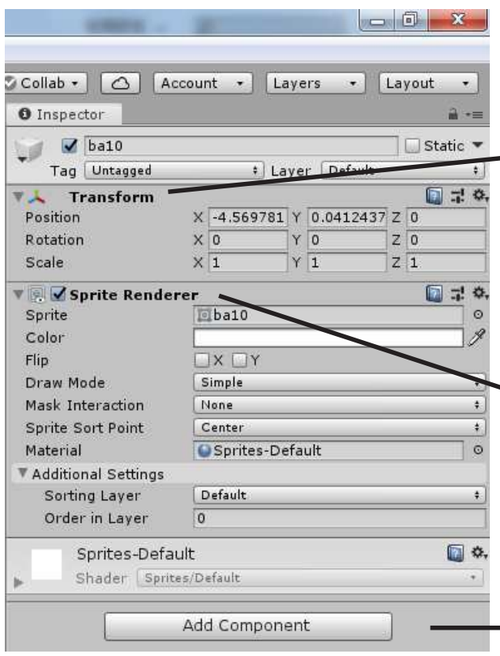
Hierarchy	階層
Project	事業
Inspector	検査官
Component	成分
Object	物体
Script	脚本
Scene	場面

コンポーネント



Unity ではゲームオブジェクトの様々な機能を「コンポーネント」と呼ばれる形で追加します。コンポーネントとは直訳すると「部品」とか「構成要素」という意味になります。

位置などを示す「Transform」は最初から標準で設定されています。



例　画像を配置した時、標準で設定されているコンポーネント

Transform　位置・回転・スケール

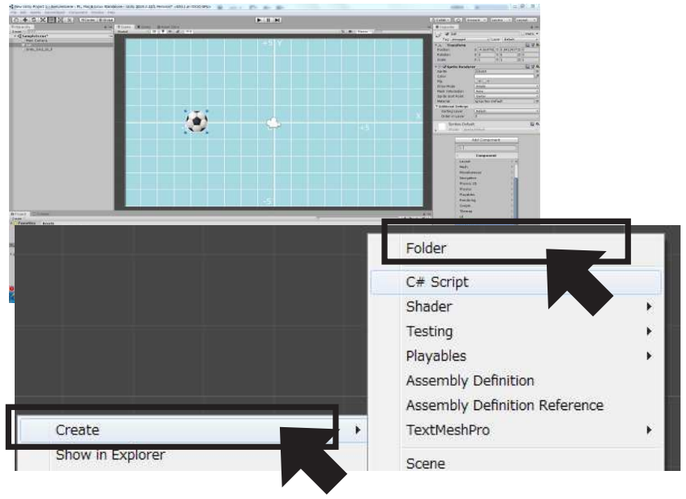
型	変数名	概　要	Inspector上での名称
Vector3	position	ワールド空間における位置情報	Position
Vector3	localScale	スケール値。2Dゲームでは x y 成分のみ変更する	Scale
Vector3	eulerAngles	回転角度。2Dゲームでは通常 Z軸 の回転のみ行う	Rotation

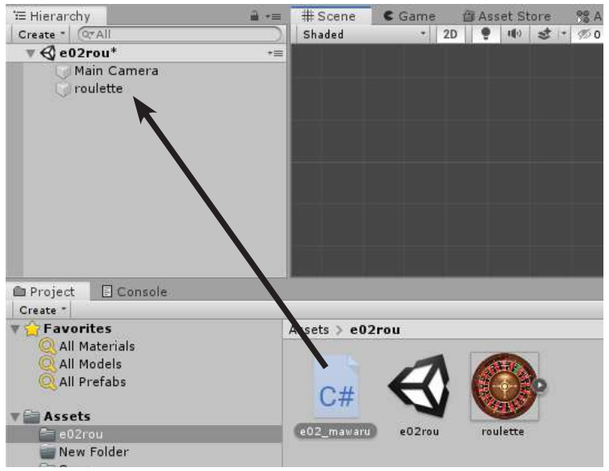
Sprite Renderer　スプライト表示

型	変数名	概　要	Inspector上での名称
bool	enabled	スプライトの表示(true)・非表示(false)を制御する	チェックボックス
Sprite	sprite	描画対象のスプライト	Sprite
Color	color	描画するスプライトの頂点カラー（色）	Color
string	sortingLayerName	ソーティングレイヤー名	Sorting Layer
int	sortingOrder	ソーティングレイヤー内での描画順	Order in Layer

新たなコンポーネントの追加

重　要　スクリプトの作成とアタッチ





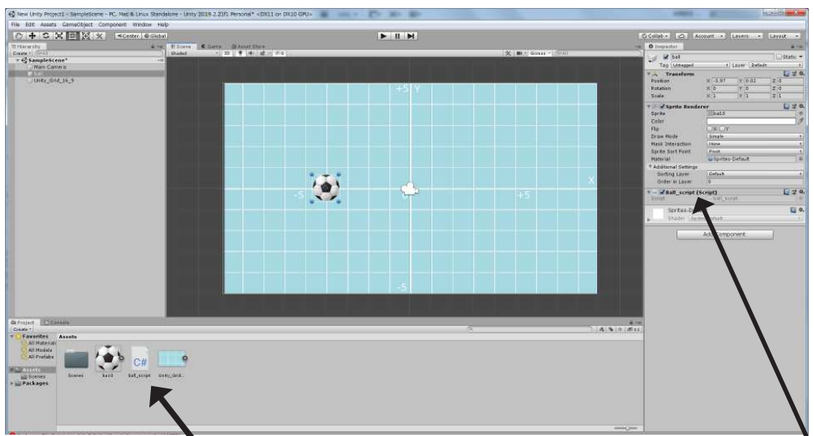
プロジェクトウインドから作成

1）プロジェクトウインドで保存するフォルダを開き、空いている場所で右クリック

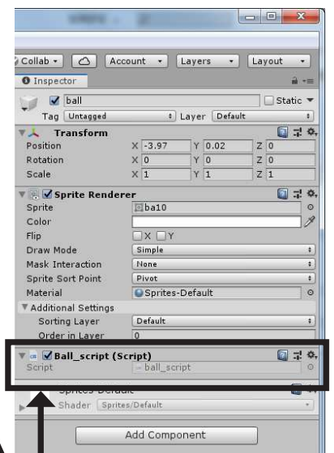
2）ポップアップメニューから「Create」 →「C# Script」を選択

3）スクリプト名をつける

4）スクリプト作成後、ヒエラルキーウインドのオブジェクトにアタッチする。

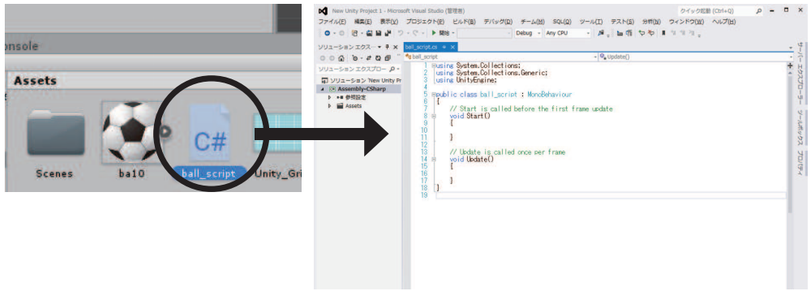


プロジェクトウインドにスクリプトが追加されます。



インスペクターウインドに「Script」コンポーネントが追加されます。

コードの編集



プロジェクトウインドのスクリプトのアイコンをダブルクリックすると編集用のエディタが起動します。
今回スクリプトは「Ｃ＃」で記述していきます。
最初にベースとなる「ひな形」が表示されます。

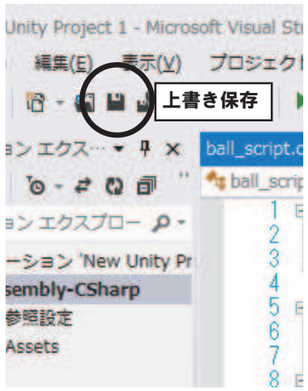
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class r01_ball : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        transform.Translate(0.1f, 0, 0);
    }
}
```

ファイル名と同じクラス名

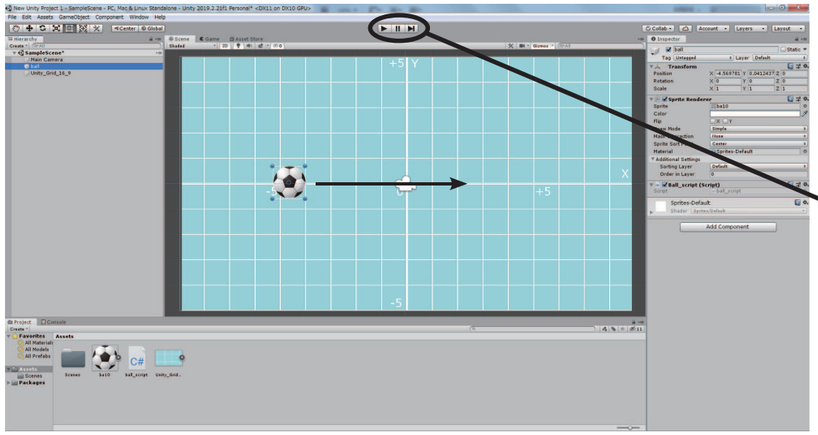
ゲームスタート時に１回だけ実行

フレーム毎に実行

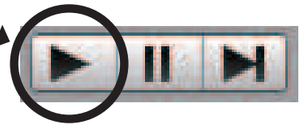
左のようにスクリプトを追加入力し、「上書き保存」ボタンをクリックします。



ゲームの実行



Unityエディタに戻り、中央上部の「実行」ボタンをクリックします。

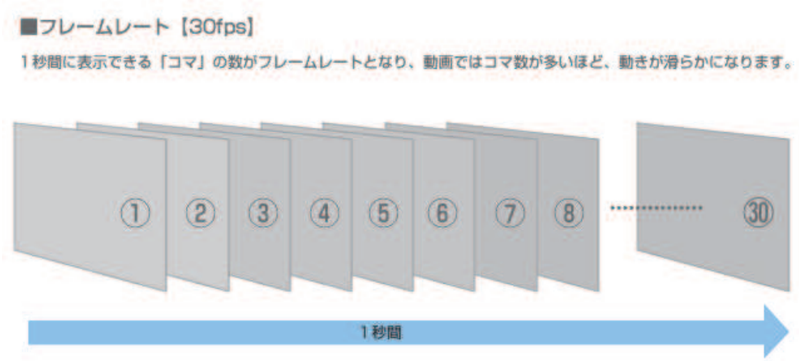


プログラムを停止するためにはもう一度「実行」ボタンをクリックします。

重要 動くオブジェクトの作り方

①シーンビューにオブジェクトを配置します。
②オブジェクトを動作させるスクリプトを作成します。
③作成したスクリプトをオブジェクトにアタッチします。

フレームレート



フレームレートとは動画における画面表示速度のことで、１秒間に何回画面を書き換えるかを表す数値です。動画１秒間当たりに表示されるフレーム数で表され、単位はfps（frame per second）です。映画の場合は24fps・テレビやビデオの場合は30fps、ゲーム（Unity）の場合は60fpsが標準です。

クラスとプロパティ・メソッド

```
・・・

    void Update()
    {
        transform.Translate( 0.1f, 0, 0 );
    }
・・・
```

X Y Z 座標（↓ ↓ ↓）

transform クラスの Translate メソッドを実行
１／６０秒毎にＸ方向へ「０．１ｆ」移動する

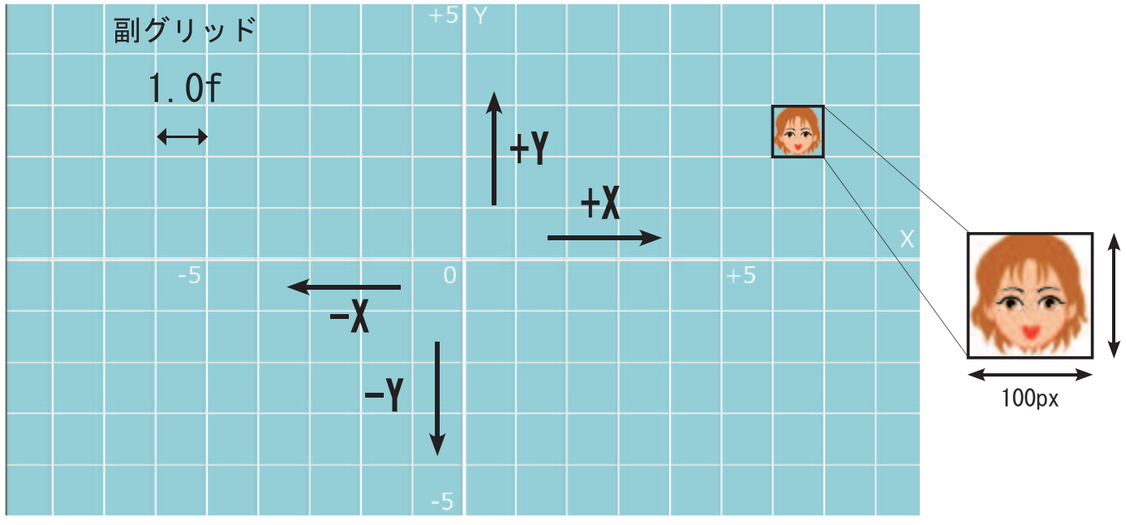
クラスはオブジェクトの設計図のようなもので、プロパティはそのオブジェクトの様々な性質を表します。メソッドはそのプロパティを変更したり、オブジェクトを操作する命令です。

Unity の単位系

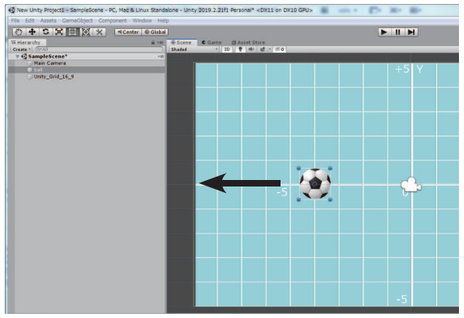
主グリッド	1m×1m
副グリッド	10cm×10cm
距離1.0f	10cm
重さ(Mass) 1.0f	1.0kg

Unity の長さや重さは実在の物体に対応しています。実際にはこれらを考慮してオブジェクトのサイズを設計します。

Unity 内で使用する「1.0f」は「１０ｃｍ」に相当します。画像を配置する場合、初期設定では「1.0f」が「100px」となっています。（通常「Pixels per Unit」は 100 に設定済　）

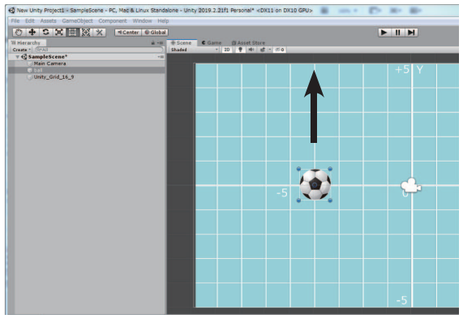


オブジェクトに次の動きをさせるように、スクリプトを変更して実行せよ。



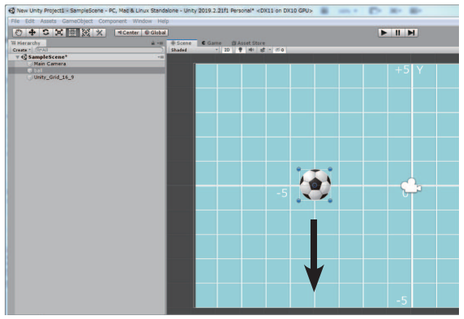
①「左」へ移動

```
・・・

    void Update()
    {
        transform.Translate(　　　　　　　　　　　　)
    }
・・・
```



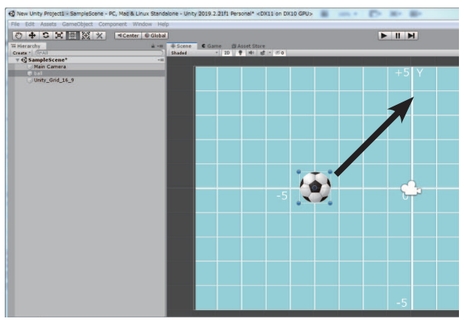
②「上」へ移動

```
・・・

    void Update()
    {
        transform.Translate(　　　　　　　　　　　　)
    }
・・・
```



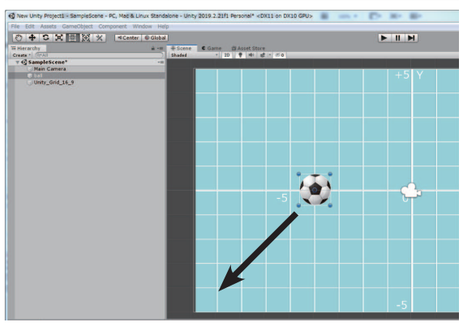
③「下」へ移動

```
・・・

    void Update()
    {
        transform.Translate(　　　　　　　　　　　　)
    }
・・・
```



④「斜め左上」へ移動

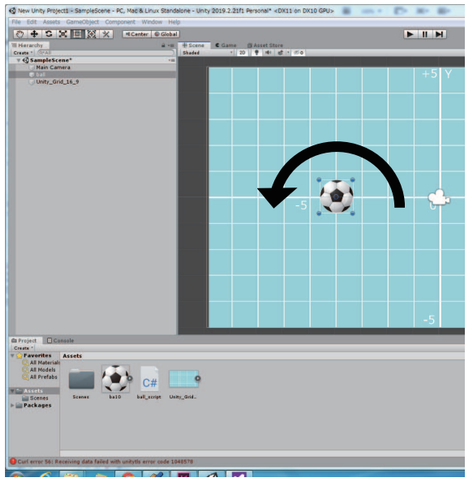
```
・・・

    void Update()
    {
        transform.Translate(　　　　　　　　　　　　)
    }
・・・
```



⑤「斜め右下」へ移動

```
・・・

    void Update()
    {
        transform.Translate(　　　　　　　　　　　　)
    }
・・・
```

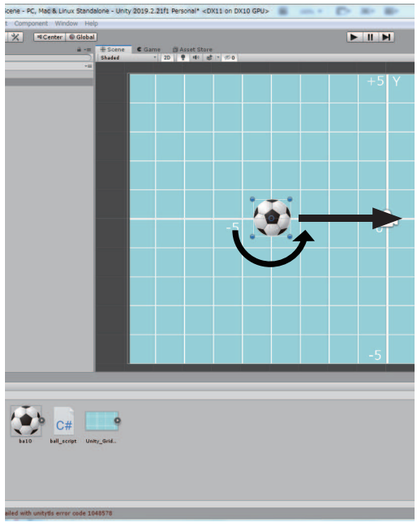

オブジェクトをその場で回転させるスクリプトを作る（正転、　　逆



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

public class r01_ball : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        transform.Rotate( 0 , 0 , 5.0f );
    }
}
```

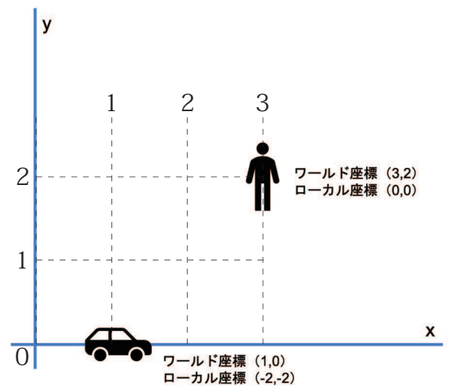
X　Y　Z　軸

オブジェクトを回転させながら移動するクリプトを作る



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

public class r01_ball : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }
     // Update is called once per frame
    void Update()
    {
        transform.Translate　　　　　　　　　　　　
        transform.Rotate(0, 0, 5.0f);
    }
}
```

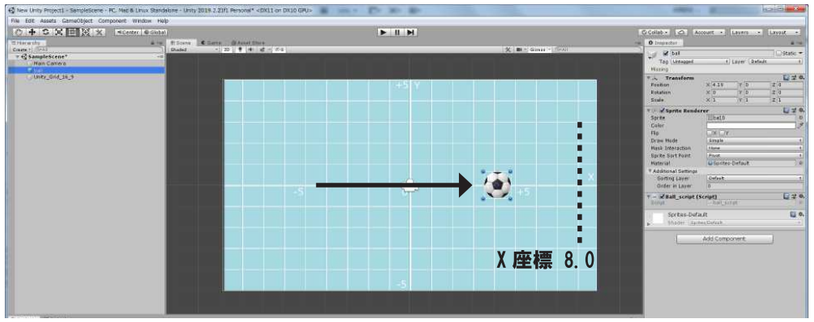
ワールド座標とローカル座標



Translate メソッドは標準で自分自身を基準とした「ローカル座標」を使用します。

シーンの原点を基準とした「ワールド座標」を使用するためには、第４パラメータに「Space.World」を追加設定します。

条件判断（ｉｆ）



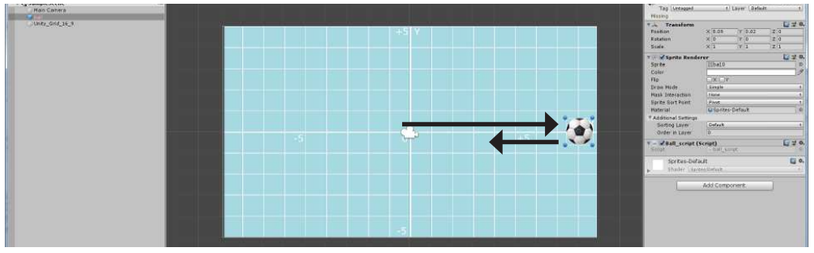
オブジェクト（スプライト）を移動し、指定された座標より大きくなったら動作を停止するスクリプトを追加します。

```
. .
. . .
     void Update()
    {
        transform.Translate(0.1f, 0, 0,Space.World);
        transform.Rotate(0, 0, 5f);
        Debug.Log(transform.position.x);
        if ( transform.position.x >= 8.0f )
        {
            Debug.Break();        一時停止
        }
    }
}
```

transform.position.x で現在のｘ座標位置を取得しｘ座標が「8.0」以上になっところでスクリプトの実行を一時停止「Debug.Break();」している。

コンソールウインドに変数の内容などを表示する **Debug.Log()** というメソッドもある。

オブジェクトの移動方向を変える



ｘ座標が「8.0」以上になったらところでスクリプトの実行を移動方向を変えるスクリプトを追加する。

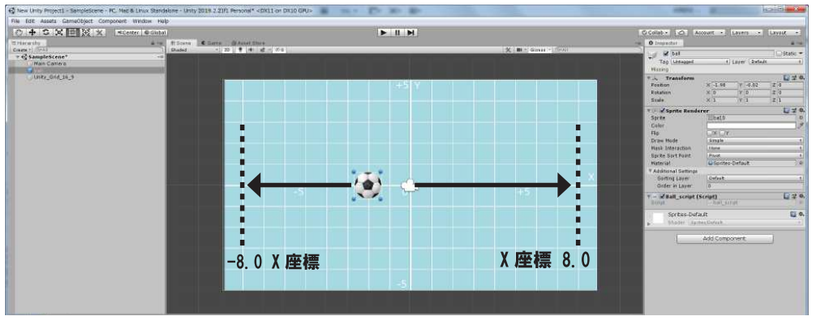
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class r01_ball : MonoBehaviour
{
    float ax = 0.1f;  ←――― // 一度の移動量

    void Start()
    {

    }
     void Update()
    {
        transform.Translate( ax , 0, 0,Space.World);
        transform.Rotate(0, 0, 5f);
        if (transform.position.x >= 8.0f)
        {
            ax = [          ]      // 移動する方向を変える
        }
    }
}
```

○比較演算子

条件式において、大きい、小さい、等しいなどの大小比較を行う演算子を比較演算子と呼び、次の６つがあります。等しいは == と = を２つ書くことに注意してください。

比較演算子	意　味
>	左辺は右辺より大きい
>=	左辺は右辺より大きいか等しい
<	左辺は右辺より小さい
<=	左辺は右辺より小さいか等しい
==	左辺と右辺は等しい
!=	左辺と右辺は等しくない

オブジェクトを左右ではね返す



ゲーム画面の左右
　左（Ｘ座標　＋８，０）
　右（Ｘ座標　－８．０）
でオブジェクトの移動方向を変化させ、はねかえっているように見せるスクリプトを記述する。

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

public class r01_ball : MonoBehaviour
{
    float ax = 0.1f;

    void Start()
    {

    }
     void Update()
    {
        transform.Translate( ax , 0, 0,Space.World);
        transform.Rotate(0, 0, 5f);
        if (transform.position.x >= 8.0f)      // 右側
        {
            ax = [          ]
        }
        if ([                          ])      // 左側
        {
            ax = [          ]
        }
    }
}
```

比較演算子と論理演算子

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

public class r01_ball : MonoBehaviour
{
    float ax = 0.1f;
    void Start()
    {

    }
     void Update()
    {
        float x;
        transform.Translate( ax , 0, 0,Space.World);
        transform.Rotate(0, 0, 5f);
        x = transform.position.x;
        if( x <= -8.0 || 8.0f <= x ) ax = -ax;
    }
}
```

ax = -ax は
ax = (-1) × ax
の数式の意味

いくつかの条件式の場合

２つ以上の条件が成り立った時に分岐させたい場合は「論理演算子」を使用して条件式を構成します。

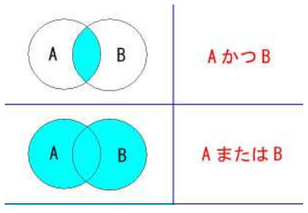
　＆＆　かつ
　｜｜　または

0<= a && b<10

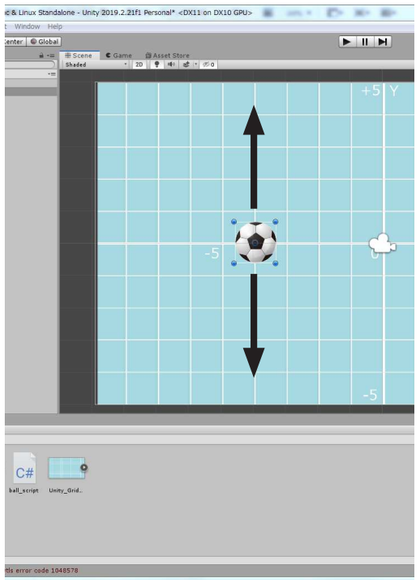
aが０より大きく
かつ
bが１０より小さい

a==1 || b==1

aが１と等しい
または
bが１と等しい



オブジェクトを縦に移動させ、上下ではねかえるような処理をせよ。

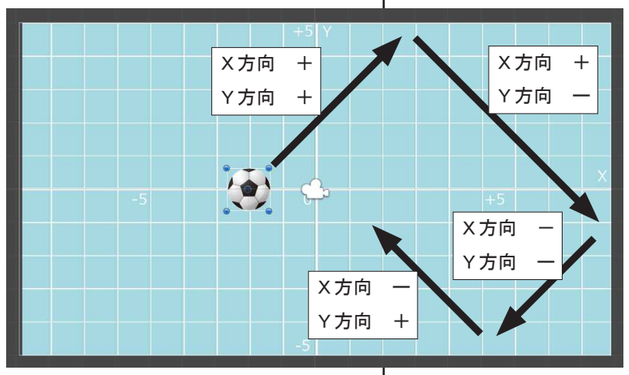


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class r01_ball : MonoBehaviour
{
    float ay = 0.1f;

    void Start()
    {

    }
     void Update()
    {
         float y;

         transform.Translate( 0, ay ,0,Space.World);
         transform.Rotate(0, 0, 5f);
         y = transform.position.y;
         if(                                    )
     }
}
```

オブジェクトを斜めに移動させ、左右、上下ではねかえるような処理をせよ。

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

public class r01_ball : MonoBehaviour
{
    float ax = 0.1f;     // X方向移動量
    float ay = 0.1f;     // Y方向移動量

    void Start()
    {

    }
     void Update()
    {
         float x, y;

         transform.Translate( ax , ay , 0,Space.World);
         transform.Rotate(0, 0, 5f);
         x =                                    // 現在位置　X座標
         y =                                    // 現在位置　Y座標
         if(                                    ) // 左右の処理
         if(                                    ) // 上下の処理
    }
}
```

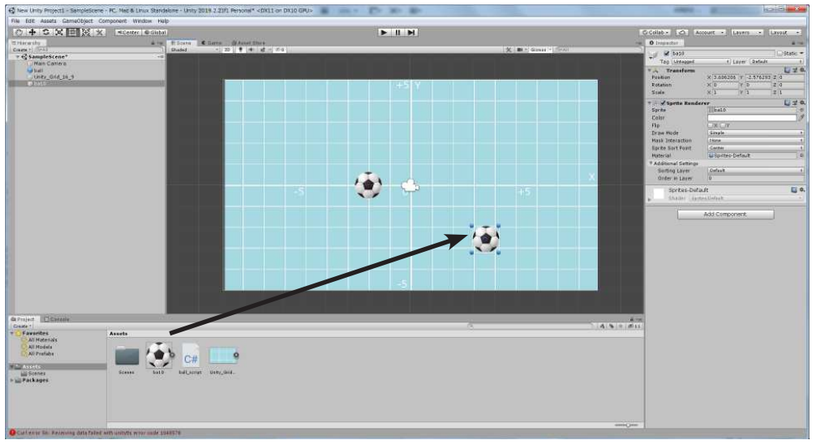
上下左右の壁に当たった時にX，Y方向それぞれの移動量はどう変化するか？

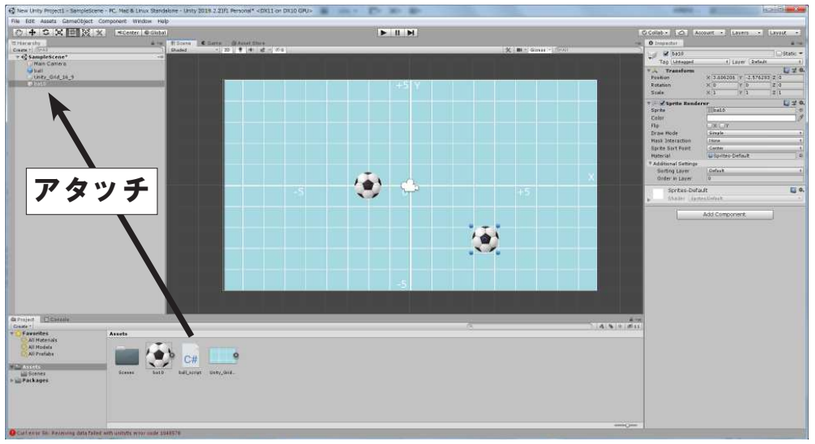


左右の壁に当たった時には（　　）方向の移動量が変化

上下の壁に当たった時には（　　）方向の移動量が変化

スクリプトのアタッチ





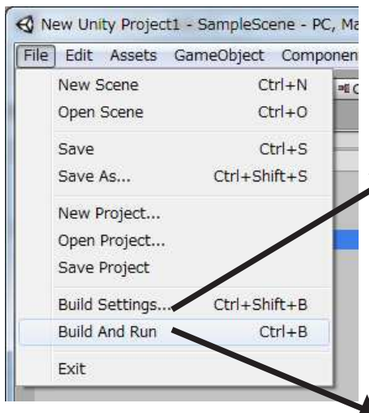
スクリプトは一つのコンポーネントとして、オブジェクトに追加されますが、通常は作成したスクリプトをヒエラルキーウインドのオブジェクトに「**アタッチ**」して関連付けます。

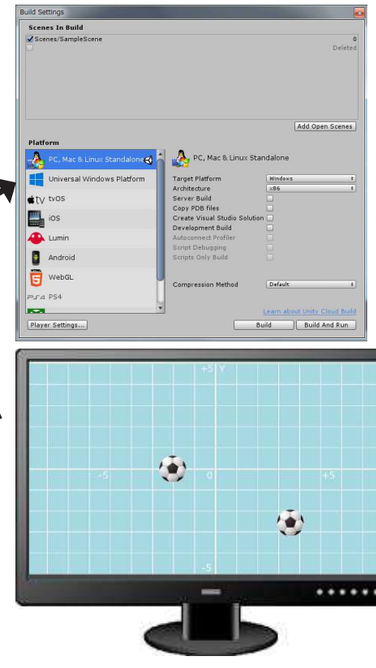
プロジェクトウインドからオブジェクトをシーンビューにドラッグ＆ドロップします。
この状態で実行してみます。
　→　動きません…

プロジェクトウインドから作成済みのスクリプトをヒエラルキーウインドのオブジェクトにドラッグ＆ドロップして「アタッチ」します。インスペクターウインドに「Script」が追加されるのを確認します。

この状態で実行してみます。
　→　動作します！

ビルド　＆　ラン





今までの実行は「Unity Editor」上でのものでした。実際にWindowsやスマホで実行するためにはプロジェクトを「ビルド」する必要があります。

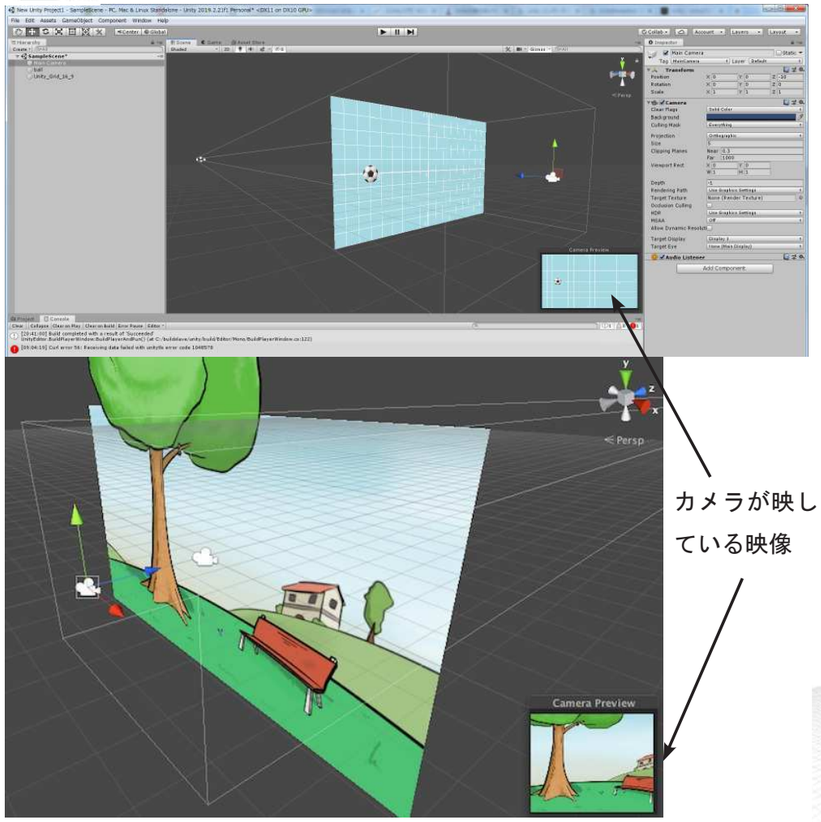
「File」メニューから
「Build Settings…」確認

「Build And Run」で全画面で実行されます。

実行の終了は Alt + F4

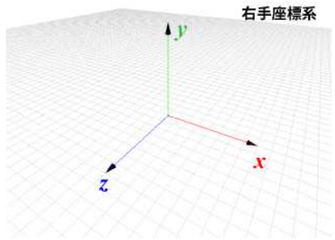
※エラーになる場合はビルドするフォルダを変更すると良い場合があります。

カメラ



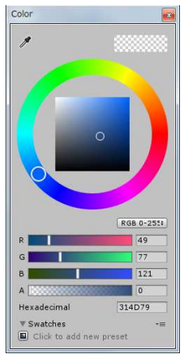
Unity のゲーム実行時にディスプレイに表示される画面は、シーン内に設定された「カメラ」が映している映像になります。

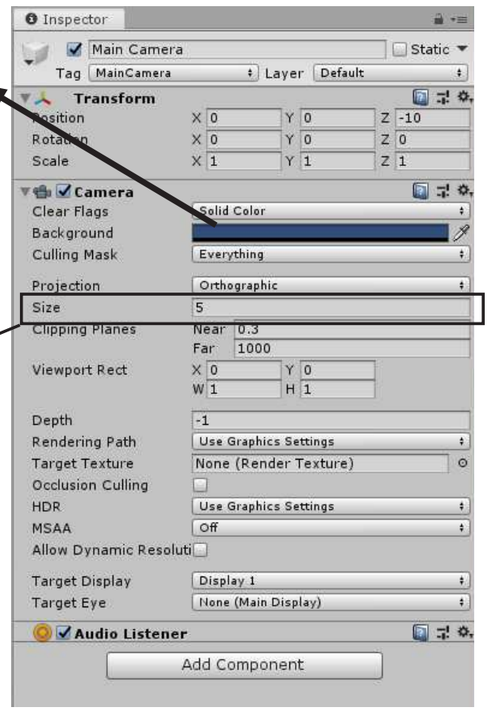
２Ｄの場合にはＸＹ座標は（０，０）、Ｚ軸は「－１０」にセットされています。
ヒエラルキーウインドの「Main Camera」を選択すると、右下にカメラの映像が確認できます。
設定はインスペクターウインドで確認できます。



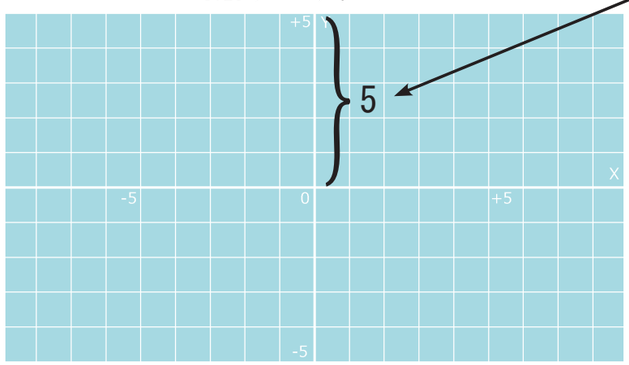
「Camera」コンポーネントの「Background」をダブルクリックするとゲーム実行時の背景色を設定することが出来ます。
右下のカメラビューを確認しながら設定します。

また「Size」プロパティはＹ軸の範囲を指定する値です。



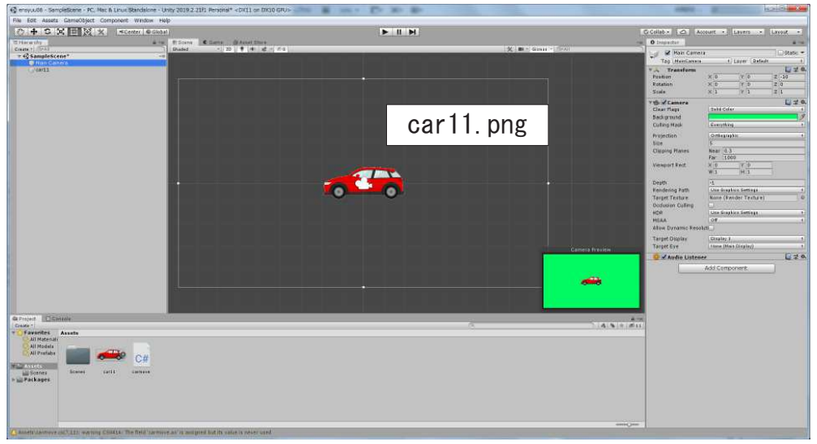


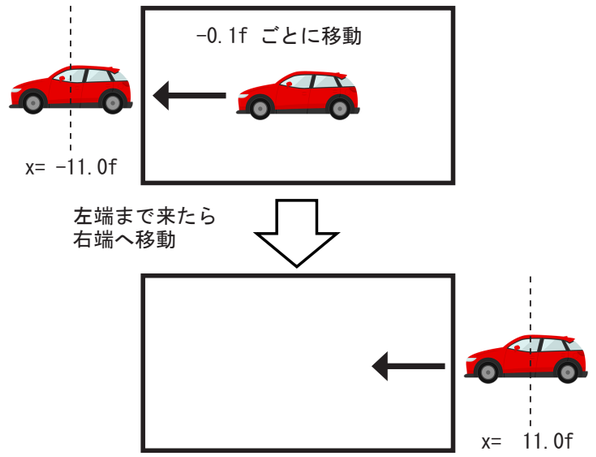
Camera コンポーネントの
Size プロパティ



演習１ Vector3

位置を表す変数型「Vector3」の使用方法を学習します。Vector2 もあります。





シーン名	e01car

自動車の画像を右から左に移動させ、画面外に出たら、右側から再び出現させるシーンを作成する。

①車の画像「car11.png」を中心（0,0,0）に配置し、大きさを整える。オブジェクト名を「car」とする。

②「Main Camera」を選択し背景色を適当なものに変更する。

③オブジェクトを移動させるスクリプトを作成する。

④車のオブジェクトにスクリプトをアタッチする。

現在位置からの相対移動ではなく、直接指定した場所へポジションを移すには Vector3 クラスを使用します。例えばオブジェクトを原点（0,0,0）へ移動するには次のようにします。

```
transform.position = new Vector3( 0,0,0 );
```

```
Vector3( 0,0,0 )
         ↑ ↑ ↑
         x y z
```

スクリプト	e01_move.cs
アタッチ先	車（car）

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

public class e01_move : MonoBehaviour
{

    void Update()
    {
        float x;
        transform.Translate( -0.1f , 0, 0);          // x座標を -0.1 ごとに移動
        x = transform.position.x;
        if (x <= -11.0f )                            // もし左端（x <= -11）まで来たら
        {
            transform.position =                     // 右端（x =11）に移動する
        }
    }
}
```

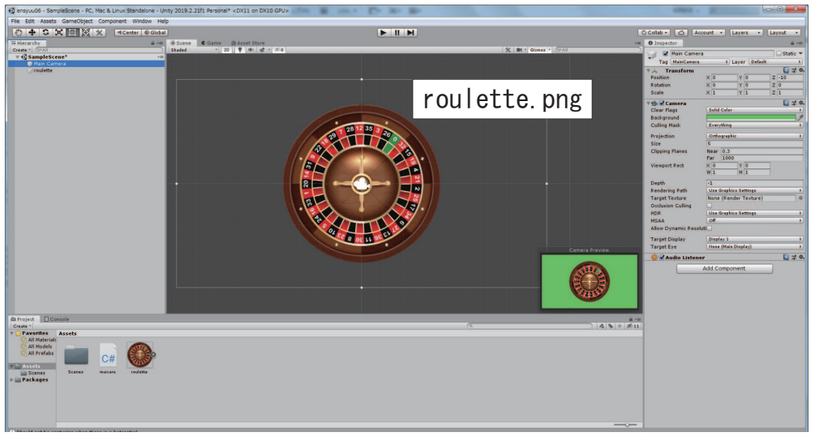
重要 動くオブジェクトの作り方

①シーンビューにオブジェクトを配置します。
②オブジェクトを動作させるスクリプトを作成します。
③作成したスクリプトをオブジェクトにアタッチします。

演習２

ルーレットの画像を回転させるシーンを作成せよ。

シーン名　e02rou



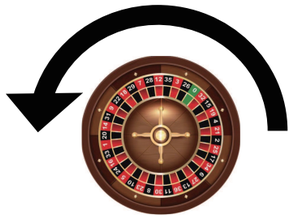
①ルーレットの画像「roulette.png」を中心に配置し、大きさを整える。

②「Main Camera」を選択し背景色を適当なものに変更する。

③オブジェクトを回転させるスクリプトを作成する。

④ルーレットのオブジェクトにスクリプトをアタッチする。


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

public class e02_mawaru : MonoBehaviour
{
    void Update()
    {
        [          ]    // ルーレットを回転
    }
}
```



スクリプト	e02_mawaru.cs
アタッチ先	ルーレット

発展

ルーレットの回転スピードが、だんだんゆっくりになるようにスクリプトを変更する。

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

public class e02_mawaru : MonoBehaviour
{
    float speed = 20.0f;   // 初速度（一度に回転する角度）

    void Update()
    {
        [                    ]    //speed 分だけ回転
        speed = [          ]      // 回転角度を減らす
    }
}
```

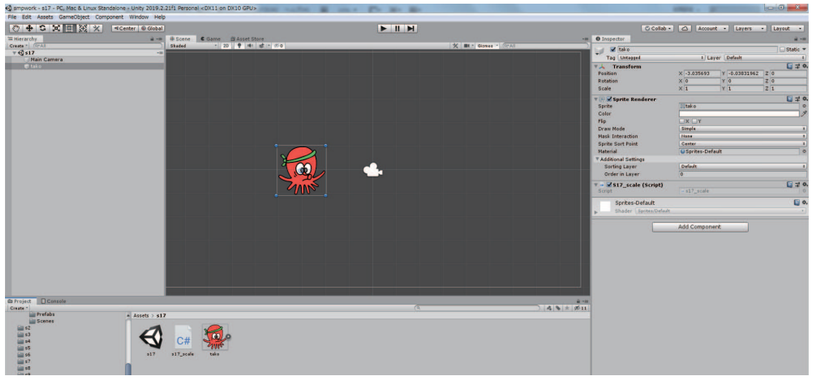
角度を減らす計算方法例

	1回の回転角度	
計算式	0.05減算	減衰係数 0.98乗算
フレーム	20.00	20.00
1	19.95	19.80
2	19.90	19.60
3	19.85	19.41
4	19.80	19.21
5	19.75	19.02
6	19.70	18.83
7	19.65	18.64
8	19.60	18.45
9	19.55	18.27
10	19.50	18.09
・・・	・・・	・・・

演習３

オブジェクトのスケールの拡大縮小を繰り返すシーンを作成する。

シーン名　e03scal



①タコの画像「tako.png」を配置する。

②オブジェクトを拡大縮小するスクリプトを作成しアタッチする。

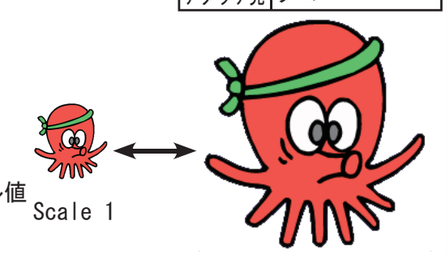
③実行を確認します。

オブジェクトのスケール変更にも「Vector3」クラスを使用します。

```
transform.localScale = new Vector3( 2 ,2 ,1 );
                                    ↑  ↑  ↑
                                    x  y  z  方向のスケールを設定
```

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

public class s17_scale : MonoBehaviour
{
    float s = 1.0f;              // スケールを表す変数
    float sx = 0.05f;            // 一度に変化するスケール値

    void Update()
    {
        transform.localScale = new Vector3(s, s, 1);      // x y方向のスケールを「s」に設定
        s = s + sx;                          // スケールを sx 分だけ増減する。
        if([                    ])  // もしスケールが３より大きくなったら sx をマイナスに (-0.05f)
        if([                    ])  // もしスケールが１より小さくなったら sx をプラスに (+0.05f)
    }
}
```

スクリプト	e03_scale.cs
アタッチ先	タコ



発展

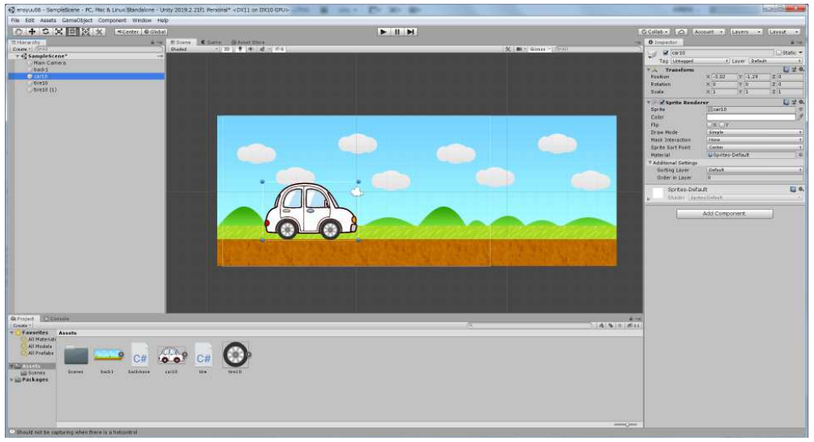
論理演算子を使用して if 文を１行で記述しなさい。

```
・・・
    void Update()
    {
        transform.localScale = new Vector3(s, s, 1);
        s = s + sx;
        if([                                ])
    }                                  // もしスケールが３より大きか、１より小さくなったら sx を反転する
```

演習４

背景を右から左へ横スクロールさせながら、。車のタイヤを回転させるシーンを作成する。

シーン名　e04car

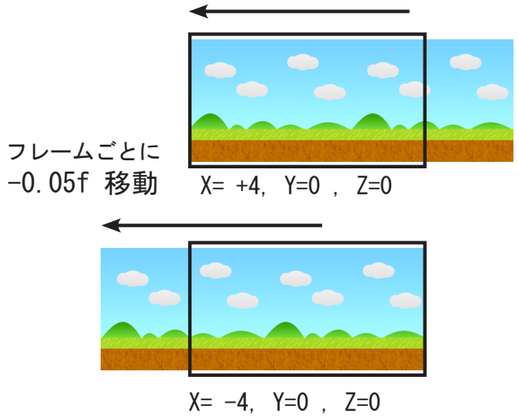


①背景画像「back1.png」をスケール調整しながら左端に揃える様に配置する。
②車の画像「car10.png」を配置する。
③タイヤの画像「tire.png」を車の画像と合うように片側だけ配置する。
④背景オブジェクトを右から左にスクロールさせるスクリプトを作成しアタッチする。
⑤タイヤオブジェクトを回転させるにスクリプトを作成しアタッチする。

背景を移動するスクリプトを作り背景画像にアタッチする（e03_move.cs）

スクリプト	e04_move.cs
アタッチ先	背景画像

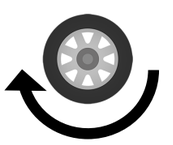
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class e04_move : MonoBehaviour
{
    void Update()
    {
        transform.Translate(-0.05f, 0, 0);
        if(                                    )
            {                                  }
        }
    }
}
```

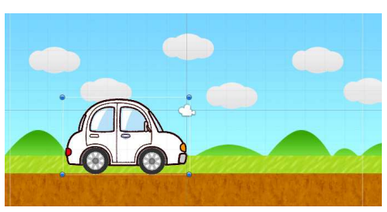


タイヤを回転するスクリプトを作り、タイヤの画像にアタッチする（e03_tire.cs）
タイヤは２つあるのでオブジェクトをコピーして、もう一つの位置に配置する。

スクリプト	e04_tire.cs
アタッチ先	タイヤ

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

public class e04_tire : MonoBehaviour
{
    void Update()
    {
        transform.Rotate(0, 0, -5);
    }
}
```

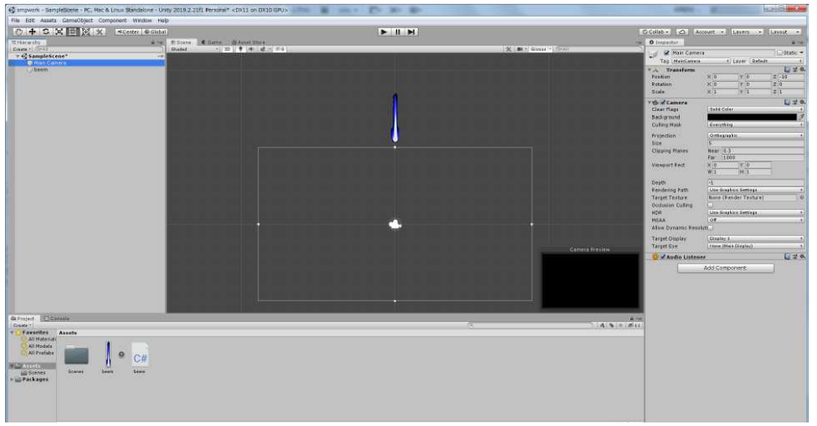




例題２　乱数

流星が上から下へ流れるシーンを作成する。出現位置をランダムにする。

シーン名　r02beem



①流星画像「beem.png」をスケール調整しながら上方に配置する。

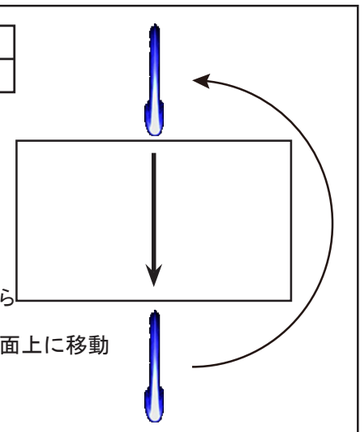
②「Main Camera」を選択し背景色を黒に変更する。

③流星オブジェクトを上から下へ移動するスクリプトを作成しアタッチする。

④流星に横位置をランダムになるようスクリプトを修正する。

スクリプト	r02_beem.cs
アタッチ先	流星

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

public class r02_beem : MonoBehaviour
{
    int x = 0;

    void Update()
    {
        transform.Translate(0,-0.2f,0);
        if (transform.position.y < -6.0f)          // 画面の外に出たら
        {
            transform.position = new Vector3(x, 6.0f, 0);  // 画面上に移動
        }
    }
}
```



乱数を得る

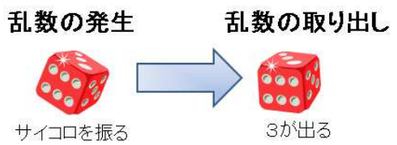
乱数とは、ある範囲の数値から任意に（ランダムに）取り出した数値のことです。ゲームにおける乱数といえば、敵と遭遇するかどうか、宝箱の出現確率、魔法攻撃の状態異常が発生するか、など様々なシーンで使われています。Unity で乱数を得るためには次のようにします。

Ａ以上Ｂ未満の乱数を得る（整数型か実数型）

```
Random.Range(A, B)
```

例　int x;

x = Random.Range(1,7);　１～６までの乱数（サイコロ）




```
・・・１行追加する
        if (transform.position.y < -6.0f)
        {
            x = Random.Range(-7, 8);       //-7 ～ +7 の乱数を発生する
            transform.position = new Vector3( x, 6.0f, 0 );     // 横位置は乱数
        }
・・・
```

乱数の種

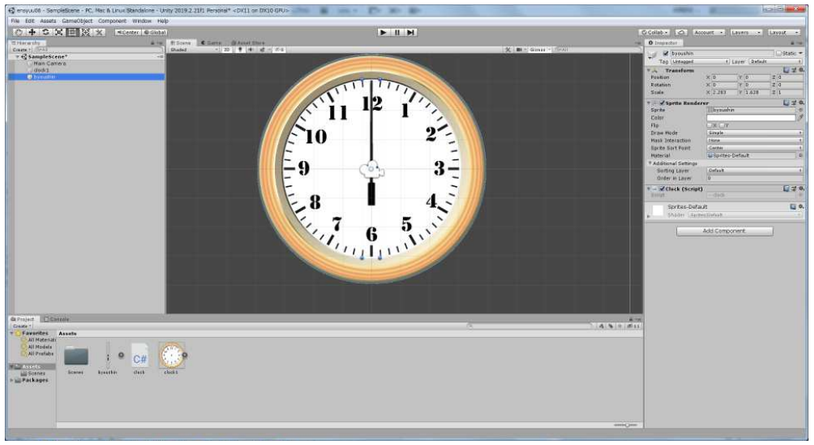
コンピュータの乱数は数式で擬似的に作り出したものなので、いつも同じ出現パターンになってしまうことがあります。そんなときには乱数の種（シード）を再設定させます。

```
Random.InitState( System.DateTime.Now.Millisecond )
```

…現在時刻（ミリ秒）をシードに設定

例題３ 一定時間ごとに処理する

シーン名　r03clk



時計の秒針を動かすシーンを作成せよ。

①時計画像「clock1.png」を原点に配置する。

②秒針の画像「byoushin.png」を原点に配置し、スケールを調整する。

③秒針が１秒で６度回転するスクリプトを作成しアタッチする。

一定時間ごとに処理する

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

public class NewBehaviourScript : MonoBehaviour
{
    void Start()              ゲームスタート時に１回だけ実行される
    {
        InvokeRepeating("move",1f, 1f);   １秒毎に「move」を呼び出す
    }

    void Update()             フレーム毎（１／６０秒）に実行される
    {

    }

    void move()               自分で追加作成したメソッド「move」
    {
        transform.translate( 0.1f, 0, 0);    移動処理
    }
}
```

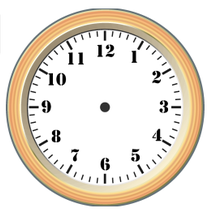
UnityのＣ＃の初期スクリプトには Start() と Update() の２つのメソッドが用意されています。

自分で機能を追加したい場合には新たに自作のメソッドを追記します。例えばオブジェクトを１秒ごとに移動させたい場合は左のようにします。

「move」は自分で作成したメソッドです。

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

public class r03_clock : MonoBehaviour
{
    void Start()
    {
        InvokeRepeating("clk",1.0f, 1.0f); // １秒ごとに clk() を呼び出す設定
    }

    void clk()
    {
        transform.Rotate( 0, 0, -6.0f );     // 秒針を６度 回転する
    }
}
```

clock1.png



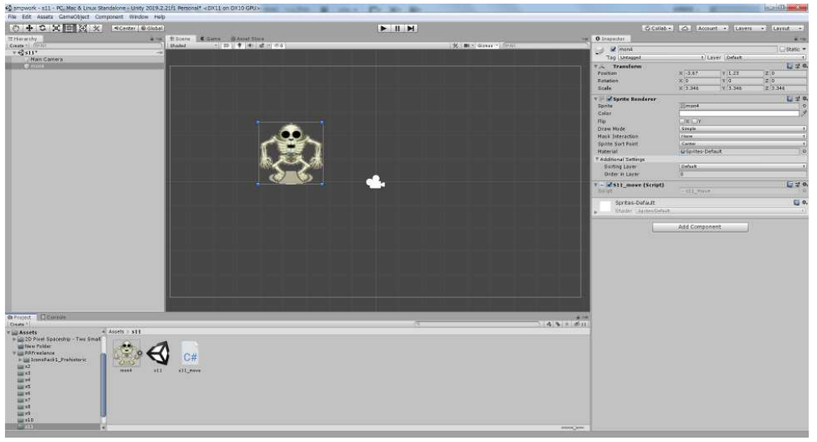
byoushin.png

スクリプト	r03_clock.cs
アタッチ先	秒針

演習５

キャラクタが０．５秒ごとにランダムに移動するシーンを作成しなさい。

シーン名　e05rnd



①画像「mon4.png」を配置し、スケールを調整します。

②０．５秒ごとにランダムに移動するスクリプトを作成しアタッチします。

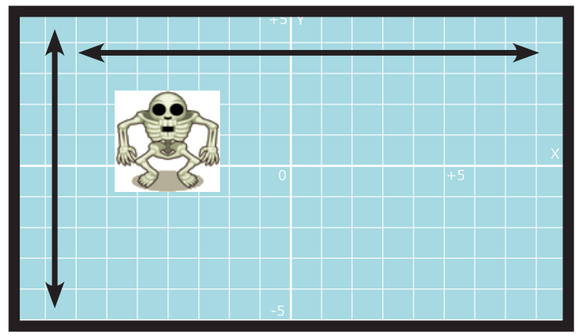
呼び出し間隔などを変更して実行してみましょう。

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

public class e05_move : MonoBehaviour
{
    void Start()
    {
        InvokeRepeating(                    )     // ０．５秒ごとに「move」呼び出し
    }

    void move()
    {
        int x =                    
        int y =                    
        transform.position = new Vector3(x, y, 0);  // ランダムな場所に移動
    }
}
```

スクリプト	e05_move.cs
アタッチ先	モンスター

－７　～　＋７　の乱数

－４～＋４の乱数



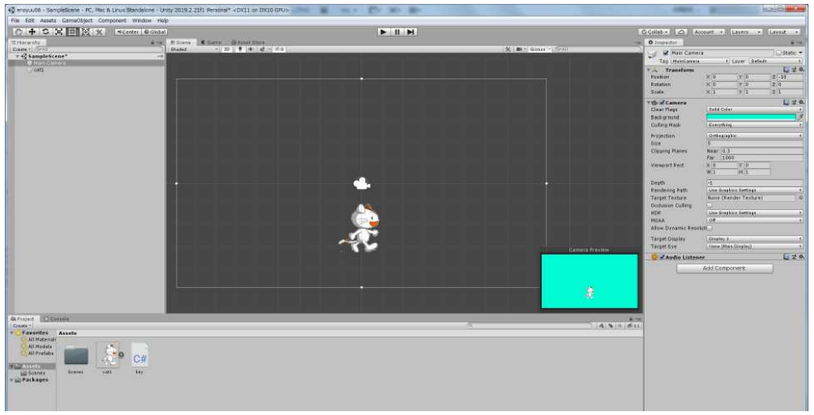
重要 動くオブジェクトの作り方

①シーンビューにオブジェクトを配置します。
②オブジェクトを動作させるスクリプトを作成します。
③作成したスクリプトをオブジェクトにアタッチします。

例題４ キー入力処理

キー入力によってオブジェクトを左右に移動する。

シーン名　r04cat



①猫の画像「cat1.png」を配置する。

②「Main Camera」を選択し背景色を適当なものに変更する。

③猫オブジェクトをキー入力によって移動するスクリプトを作成し、アタッチする。

スクリプト	r04_key.cs
アタッチ先	猫

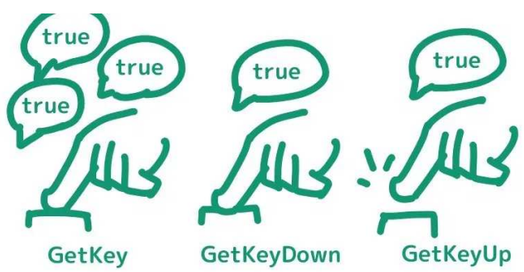
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class r04_key : MonoBehaviour
{
    void Update()
    {
        if (Input.GetKey(KeyCode.LeftArrow))        //押されたキーが「←」なら
        {
            transform.Translate(-0.1f, 0, 0);       //      左へ移動
        }
        if (Input.GetKey(KeyCode.RightArrow))       //押されたキーが「→」なら
        {
            transform.Translate(0.1f, 0, 0);        //      右へ移動
        }
    }
}
```

localScale をマイナス値にすると左右反転

```
transform.localScale = new Vector3(-2.0f, 2, 2);
```

Input クラスはさまざまな入力を取得することができるクラス。キーボードやマウス、ゲームパッドなどから入力を受け取ることができます。おもなキーコードは右表の通り。
次のメソッドで３つの状態を取得することができます。

名前	内容
Input.GetKey	キーを押している間は常に(連射状態)
Input.GetKeyDown	キーを押していない状態から押した時
InputGetKeyUp	キーを押している状態から離した時

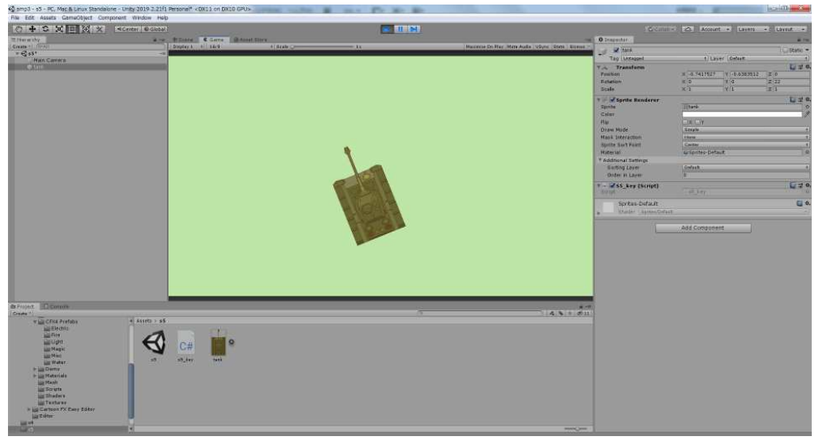


キーボード	値
Return	Return
Space	Space
上矢印キー	UpArrow
下矢印キー	DownArrow
右矢印キー	RightArrow
左矢印キー	LeftArrow
F1	F1
F15	F15
キーボード上の 0	Alpha0
キーボード上の 9	Alpha9
テンキー 0	Keypad0
テンキー 9	Keypad9
a	A
z	Z
Left mouse button	Mouse0
Right mouse button	Mouse1

演習６

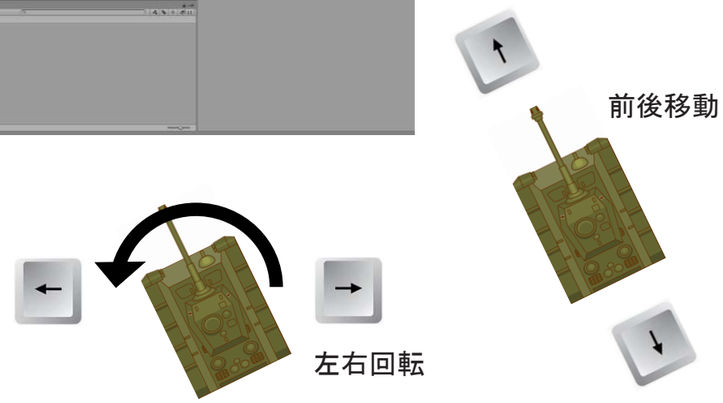
戦車のオブジェクトをキー入力（↑ ↓）で前後に移動、（← →）で左右に回転させるシーンを作成せよ。

シーン名　e06roc



①戦車画像「tank.png」を配置し、スケールを調整する。

②キー入力により戦車を移動、回転させるスクリプトを作成しアタッチする。



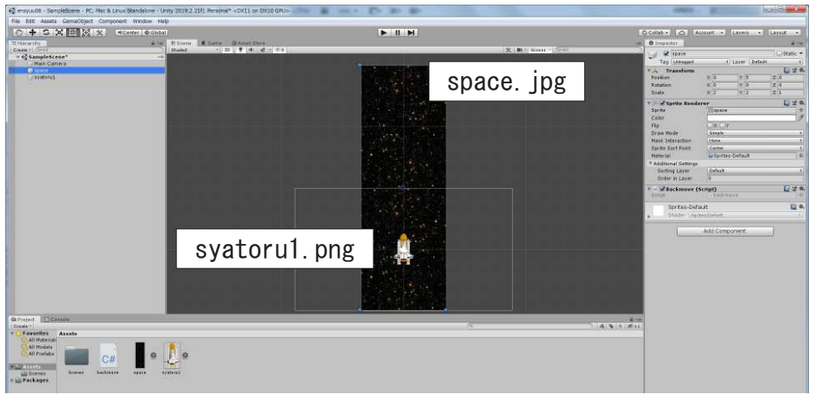
スクリプト	e06_key.cs
アタッチ先	戦車

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

public class e06_key : MonoBehaviour
{
    void Update()
    {

        if [                                ]       //押されたキーが「←」なら
        {
            [                                ]      //      戦車を「左に１度」回転
        }
        if [                                ]       //押されたキーが「→」なら
        {
            [                                ]      //      戦車を「右に１度」回転
        }
        if [                                ]       //押されたキーが「↑」なら
        {
            [                                ]      //      戦車を「前に 0,1f」移動
        }
        if [                                ]       //押されたキーが「↓」なら
        {
            [                                ]      //      戦車を「後に 0,1f」移動
        }
    }
}
```

演習７

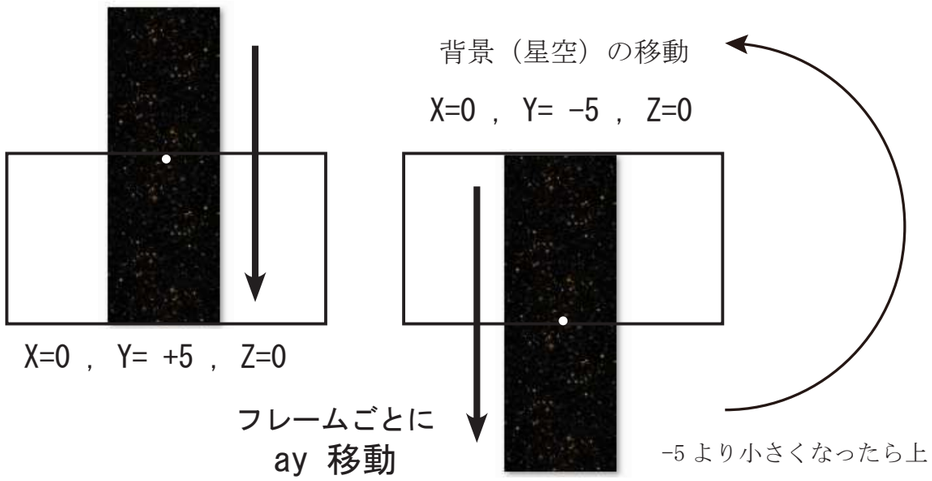
上から下へスクロールする星空を背景にして、ロケットがキー入力で左右に移動するシーンを作成せよ。

シーン名	e07roc



①星空画像「space.jpg」を配置し、スケールを縦横２倍にする。

②星空が上から下へスクロールするスクリプトを作成しアタッチする。

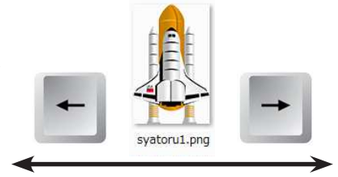


スクリプト	e07_move.cs
アタッチ先	星空背景

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

public class e07_move : MonoBehaviour
{
    float ay = -0.02f;      // １回の移動量

    void Update()
    {
        [                              ]      // Ｙ座標を ay ごと移動
        if ( transform.position.y <= -5.0f )      // もし背景Ｙが -5.0 より小さくなったら
        {
            transform.position = [                              ]  // スタート（ 0, 5.0f, 0 ）に戻す
        }
    }
}
```


スクリプト	e07_key.cs
アタッチ先	宇宙船

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

public class e07_key : MonoBehaviour
{
    void Update()
    {
        if (Input.GetKey(KeyCode.LeftArrow))    //「←」キー
        {
            transform.Translate(-0.1f, 0, 0);     // 左へ移動
        }
        if ([                              ])    //「→」キー
        {
            [                              ]      // 右へ移動
        }
    }
}
```



③ロケット画像「syatoru1.png」を配置し、スケールを縦横２倍にする。

④ロケットがキー入力で左右に移動するスクリプトを作成しアタッチする。

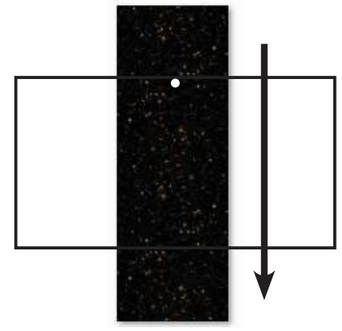
⑤上下キーによって星空のスクロールスピードをアップダウンするスクリプトを追加せよ。

キー入力で背景のスクロールスピードを変える（e07_move.cs に追加）

スクリプト	e07_move.cs
アタッチ先	星空背景

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

public class e07_move : MonoBehaviour
{
    float ay = -0.02f;

    void Update()
    {
        transform.Translate(0, ay, 0);
        if ( transform.position.y <= -5.0f )
        {
            transform.position = new Vector3(0, 5.0f, 0);
        }

        if (Input.GetKeyDown(KeyCode.UpArrow))      //「↑」キーなら
        {
            ay = [                              ]      // １回の移動量を -0.01
        }
        if (Input.GetKeyDown(KeyCode.DownArrow))    //「↓」キーなら
        {
            ay = [                              ]      // １回の移動量を +0.01
        }
    }
}
```

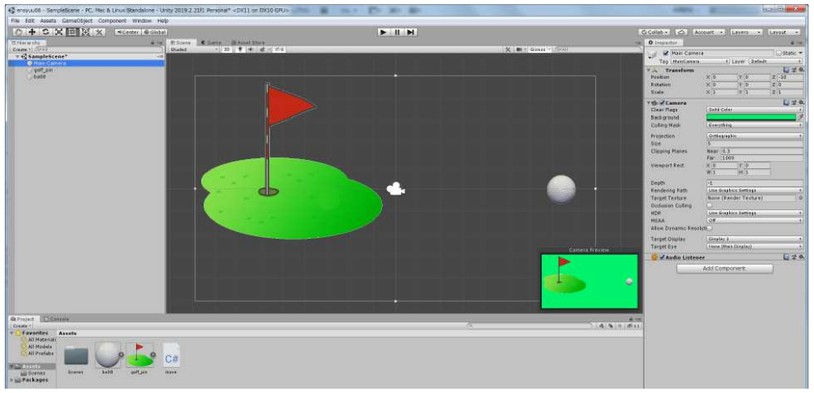
背景（星空）の
スクロールスピード変化



例題５ マウス入力

マウスのドラッグによる簡単なゴルフゲーム。

シーン名　r05golf



①ピンの画像「golf_pin.png」とボールの画像「ba08.png」を配置する。

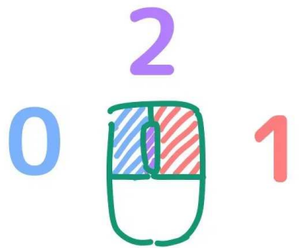
②「Main Camera」を選択し背景色を適当なものに変更する。

③マウスのドラッグ距離によってボールの初速度を計算し、ボールを減速しながら移動するスクリプトを作成、アタッチする。

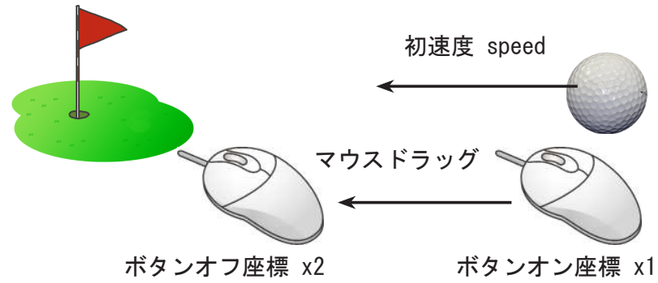
マウスによる入力も Input クラスで実装することが出来ます。
スマホやタブレットで実行するときは「タップ」の動作に該当します。
ボタンの種類は呼び出すときの引数によって指定します。

０…左ボタン　　１…右ボタン　　２…中央ボタン

名前	内容
Input.GetMouseButton	ボタン押している間は常に(連射状態)
Input.GetMouseButtonDown	ボタンを押していない状態から押した時
Input.GetMouseButtonUp	ボタンを押している状態から話した時
Input.mousePosition.x	マウスのX座標
Input.mousePosition.y	マウスのY座標




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

public class r05_move : MonoBehaviour
{
    float speed = 0.0f;
    float x1,x2;

    void Update()
    {
        if (Input.GetMouseButtonDown(0))      // 左ボタンが押されたら
        {
            x1 = Input.mousePosition.x;       //     x1 にマウスＸ座標を代入
            Debug.Log(x1);                    //     コンソールに座標を表示
        }
        if (Input.GetMouseButtonUp(0))        // 左ボタンが離されたら
        {
            x2 = Input.mousePosition.x;       //     x1 にマウスＸ座標を代入
            speed = (x2-x1)/500.0f;           //     初速度を計算（ドラッグ距離を 500 で割る例）
        }
        transform.Translate(speed,0,0);       // ボール移動
        speed = speed *0.98f;                 // 減速処理（移動距離をだんだん少なく）
    }
}
```

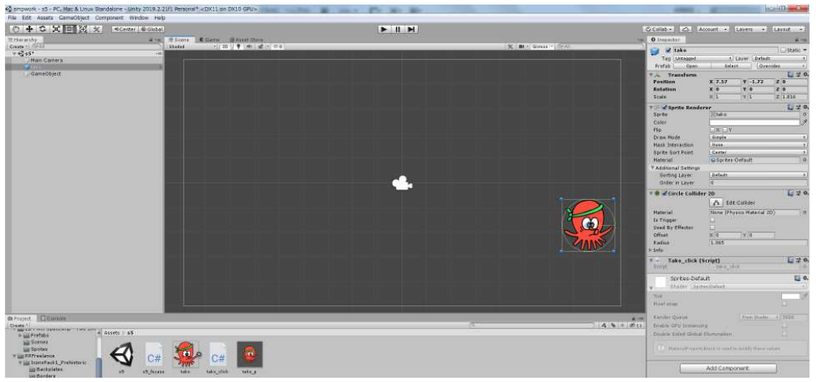


スクリプト	r05_move.cs
アタッチ先	ゴルフボール

例題６ オブジェクト上でのマウスクリック

シーン名　r06clk

「Collider」を設定するとオブジェクト上でのマウスクリックを取得できます。



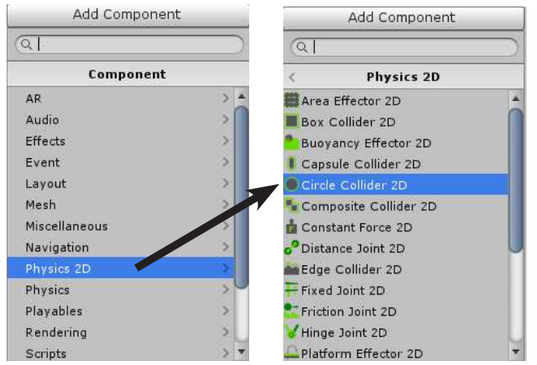
①画像（Sprite）をシーンに配置します。スケールは１のままにしておきます。

②インスペクターの「Add Component」ボタンをクリックし「Physics 2D」→「Circle Collider 2D」と選択します。

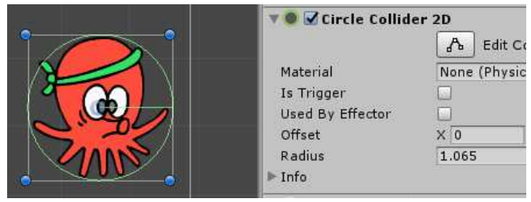
オブジェクトの周りに円形の領域が示されます。

③スクリプトを作成しアタッチ、実行します。

Collider 2D コンポーネントは、オブジェクトの衝突を判定するための領域を設定します。領域の形は円形（Circle）、四角形（Box）など設定できます。また、マウスの動作を感知するための領域としても使用できます。



Circle Collider 2D	円形の衝突エリアに使用
Box Collider 2D	正方形、長方形の衝突エリアに使用
Polygon Collider 2D	自由形状の衝突エリアに使用
Edge Collider 2D	自由形状の衝突エリアと完全に囲われていない衝突エリアに使用
Capsule Collider 2D	円形、または、ひし形の衝突エリアに使用
Composite Collider 2D	Box Collider 2D と Polygon Collider 2D を結合



動作が確認できたらコピーして増やしてみよう！

Box Collider 2D

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

public class r06_click : MonoBehaviour
{
    void OnMouseDown()  // マウスボタンが押されたら
    {
        transform.localScale = new Vector3(2, 2, 1);  //XY のスケールを２に
    }

    void OnMouseUp()    // マウスボタンが離されたら
    {
        transform.localScale = new Vector3(1, 1, 1);  //XY のスケールを１に
    }
}
```

関数	呼び出されるタイミング
OnMouseDown()	マウスボタンが押された時にコール
OnMouseDown()	マウスボタンを離した時にコール
OnMouseExit()	マウスカーソルが対象オブジェクトから退出した時にコール
OnMouseEnter()	マウスカーソルが対象オブジェクトに進入した時にコール
OnMouseOver()	マウスカーソルが対象オブジェクトに重なっている間コール
OnMouseEnter()	マウスカーソルが対象オブジェクトに進入した時にコール
OnMouseDrag()	マウスをドラッグしている間コール

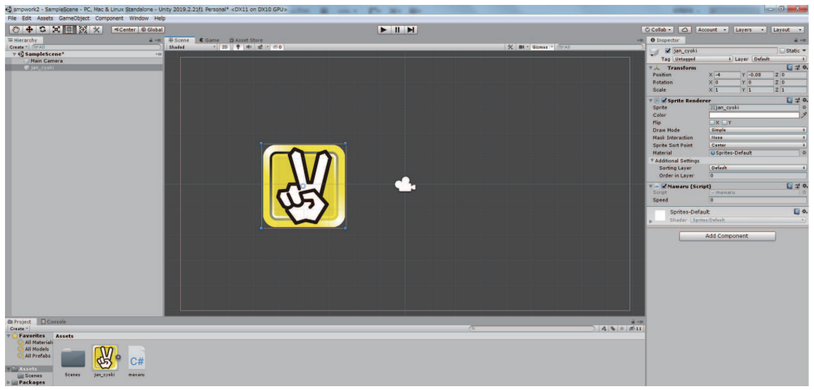
スクリプト	r06_click.cs
アタッチ先	タコ

例題６ 発展①

例題６に演習５を組み合わせて、ランダムに移動するキャラクタをクリックしたら大きさが変化するシーンを作成せよ。

シーン名　r06clk

例題７ public 変数



外部から参照できる変数「public 変数」について動作を確認します。

①画像（sprite）を配置する。

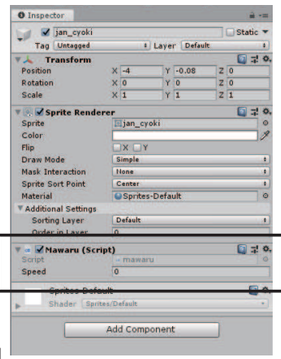
②その場で回転させるスクリプトを作成しアタッチする。この時、回転スピード(一度に回転する角度)を「public」として宣言する。

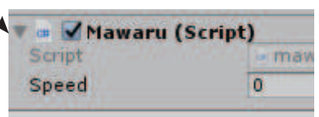
インスペクターの script コンポーネントに public 変数の項目が追加されています。

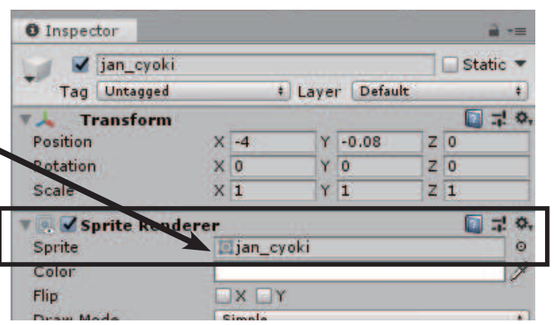
③インスペクターで public 変数を確認し実行、リアルタイムに数値を変更してみる。

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

public class r07_mawaru : MonoBehaviour
{
    public float speed;

    void Update()
    {
        transform.Rotate(0, 0, speed );
    }
}
```

スクリプト	r07_mawaru.cs
アタッチ先	画像





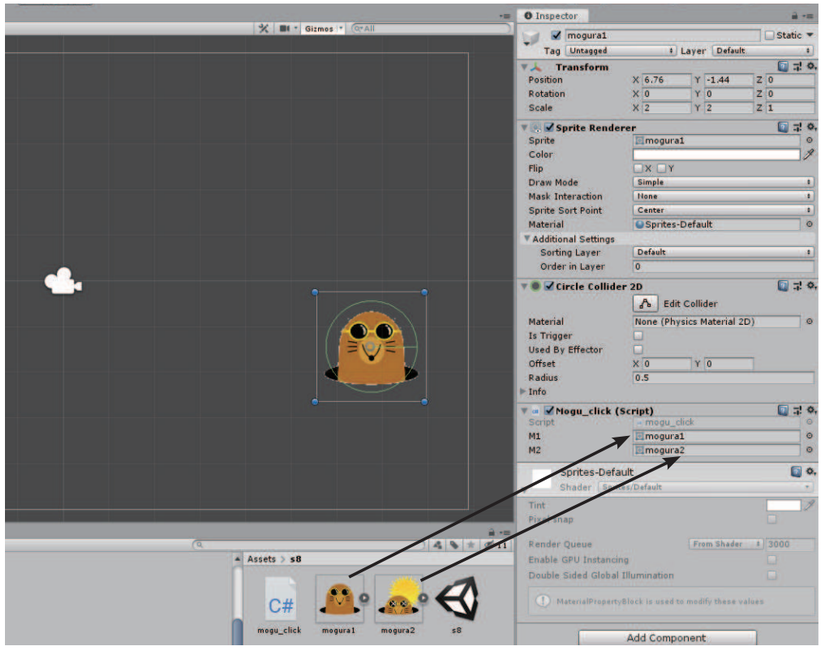


今まで意識してきませんでしたが、これと同じように、使用している画像は「sprite Renderer」コンポーネントの「Sprite」の項目に設定されています。

ここに別の画像をドラッグすればシーンに表示されている画像も変更されます。

例題８ スプライトの表示切替

シーン名　r08mogu



モグラの画像を２つ使用し、マウスクリックで画像を変更してみます。

①画像「mogura1.png」を配置、スケールを調整し、マウス入力を取得するため「Circle Collider 2D」をアタッチします。

②例の様なスクリプトを作成しアタッチします。

③インスペクターのスプライトコンポーネントに追加されている public 変数の内容を設定します。

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

public class r08_click : MonoBehaviour
{
    SpriteRenderer spr;   //SpriteRenderer 型の変数を準備

    public Sprite m1;     //Sprite 型の外部変数を準備
    public Sprite m2;

    void Start()
    {
        spr = GetComponent<SpriteRenderer>();  //オブジェクトから SpriteRenderer コンポーネントを取得
    }

    void OnMouseDown()    //マウスボタンがダウンしたら
    {
        spr.sprite = m2;  //スプライトに変数 m2 の内容を設定
    }

    void OnMouseUp()      //マウスボタンがアップしたら
    {
        spr.sprite = m1;  //スプライトに変数 m1 の内容を設定
    }
}
```

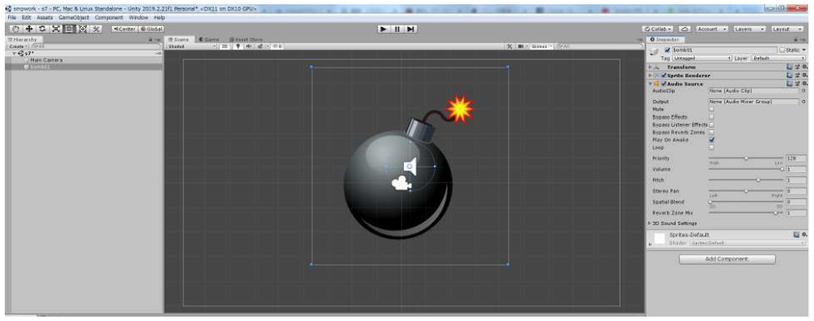
スクリプト	r08_click.cs
アタッチ先	モグラ

発展① コピー&ペーストして複数個を配置してみよう！

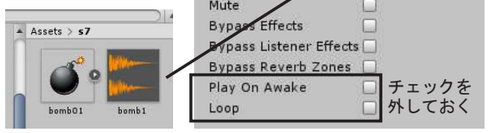
発展② 演習５と組み合わせてランダムに移動させるようにしてみよう！

例題９　効果音・ＢＧＭを再生

シーン名　r09snd



効果音は「Audio Source」コンポーネントに登録されたサウンドデータを操作することにより再生ができます。データの種類は mp3,wav,aif などが使用可能です。



①使用する画像とサウンドデータをアセットに追加します。

②画像（sprite）を配置し、「Box Collider 2D」コンポーネントをアタッチします。

③インスペクターの「Add Component」ボタンをクリックして「Audio」→「Audio Source」と選択します。

④「Audio Source」コンポーネントの「Audio Clip」にサウンドデータをドラッグして登録します。

⑤スクリプトを作成してアタッチ、実行します。

スクリプト	r09_bomb.cs
アタッチ先	爆弾

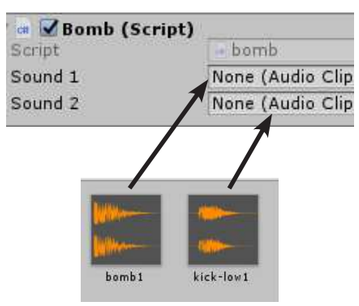
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class r09_bomb : MonoBehaviour
{
    AudioSource aud;     // オーディオソース型の変数

    void Start()
    {
        aud = GetComponent<AudioSource>();  // コンポーネントから取得
    }

    void OnMouseDown()      // もしマウスボタンがダウンしたら
    {
        aud.Play();         // コンポーネントに設定されているサウンドを再生
    }
}
```

１つのオブジェクトの中で、２つ以上の音を再生したい場合は、スクリプトに必要な数だけ AudioClip 型の外部変数を準備しアタッチ、スクリプトコンポーネントの外部変数項目にサウンドデータをドラッグ＆ドロップで登録して使用します。

複数のサウンドデータを使用したい場合（単独でも可）

スクリプト	r09_bomb.cs
アタッチ先	爆弾

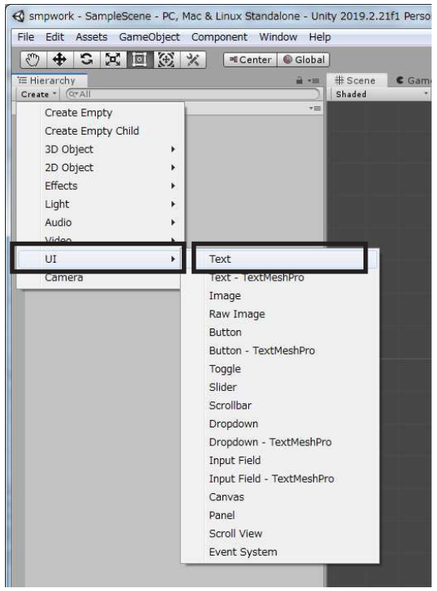
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class r09_bomb : MonoBehaviour
{
    public AudioClip sound1;  //Script コンポーネントに設定されたファイル１
    public AudioClip sound2;  //Script コンポーネントに設定されたファイル２

    AudioSource aud;

    void Start()
    {
        aud = GetComponent<AudioSource>();
    }

    void OnMouseDown()
    {
        aud.PlayOneShot( sound1 );  // 変数に格納されているサウンドファイルを再生
    }
}
```



例題 10　ＵＩ（ユーザーインターフェイス）

シーン名　r10text

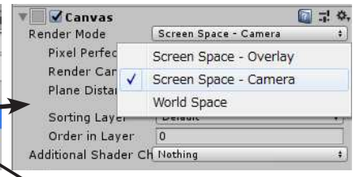
GUI（グラフィカルユーザインターフェース）とは、ユーザが視覚的に操作できる入出力機能です。各種メニューボタンやテキスト入力エリアなどがあります。他の素材と同じようにオブジェクトとして扱えます。例題としてマウスの現在位置を表示する「テキストボックス」を作ります。

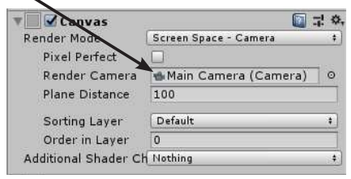


①ヒエラルキーウインド上の「Create」ボタンから「UI」→「Text」を選択。

②ヒエラルキーウインドに「Canvas」「Text」などが追加されます。

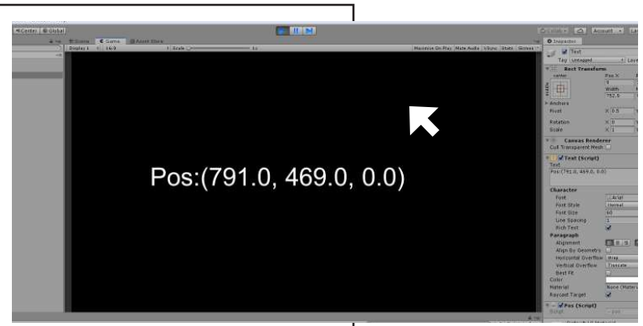
③「Canvas」を選択しインスペクターの「Render Mode」を「Camera」に変更します。





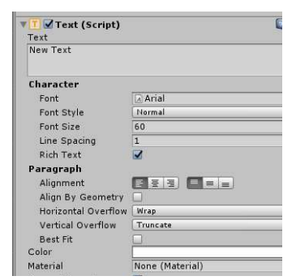
④続いて「RenderCamera」にヒエラルキーの「MainCamera」をドラッグして設定します。シーンの表示とＵＩの表示位置を合わせる事が出来ます。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;  // 追加する

public class r10_pos : MonoBehaviour
{
    GameObject mpos;

    void Start()
    {
        mpos = GameObject.Find("Text");  //Text オブジェクトを探す
    }

    void Update()
    {
        Vector3 mp = Input.mousePosition;  // マウス位置を Vector3 変数に取得
        mpos.GetComponent<Text>().text = "Pos:" + mp;
    }       // オブジェクト mpos の「Text」コンポーネント中の text を設定
}
```



⑤ヒエラルキーウインドの「Text」を選択し、テキストボックスの位置、文字サイズ、文字色などを変更します。

スクリプト	r10_pos.cs
アタッチ先	Text



⑥スクリプトを記述しアタッチします。

このスクリプトではマウス座標はスクリーン基準のピクセル値が表示されます。これを transform で使用できるワールド座標に変換するためには、取得した座標値「mp」を次のようにします。

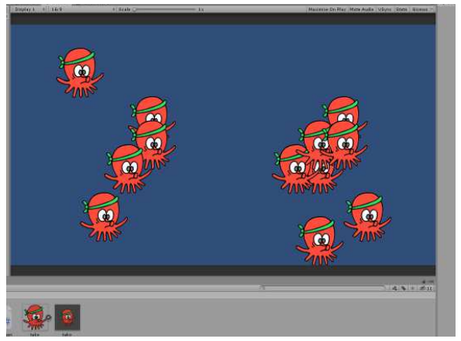
```
Camera.main.ScreenToWorldPoint( mp )
```

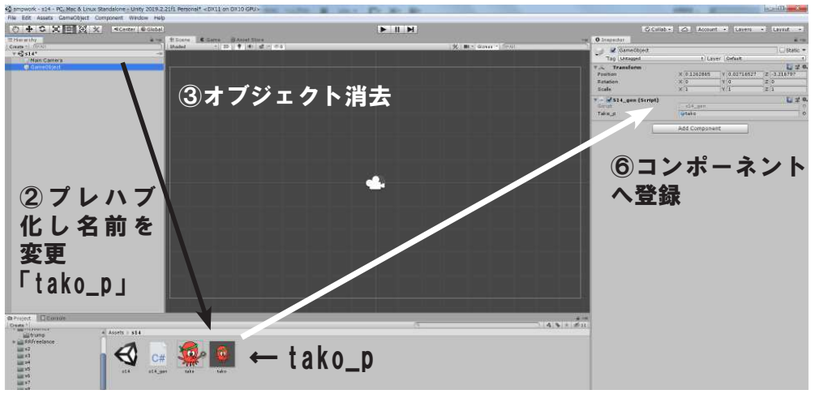
例題 11　プレハブとインスタンス

同じ種類のオブジェクトを複数扱いたい時、編集画面でひとつひとつコピーするのは大変です。スクリプトの中で複製する方法を学習します。

プレハブ（Prefabs）とは作成済みのオブジェクトを複製する機能のことで、オブジェクトの設計図とも呼ばれます。元となるプレハブの Inspector 上の設定をも含めて、簡単に同じ性質のオブジェクトを多数作成することができます。また、プレハブからオブジェクトを作ることをインスタンス化と呼びます。インスタンスとは「実体」と言う意味です。

タコのキャラクタが１秒ごとにランダムな場所に増殖するシーンを作成します。





シーン名　r11tako

①タコ画像「tako.png」を配置する。

②オブジェクト「tako」をヒエラルキーウインドからプロジェクトウインドへドラッグ「プレハブ化」し、名前を「tako_p」と変更する。

③元のオブジェクト「tako」を削除する。

④１秒ごとにタコを増殖させるスクリプトを作成する。(r11_gen.cs)

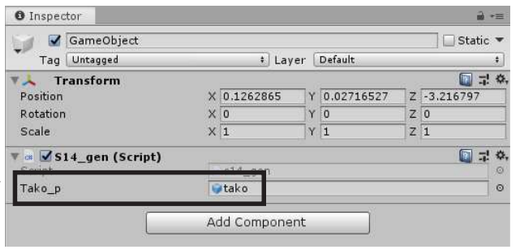
⑤ヒエラルキーウインドで空のオブジェクト「GameObject」(管理用オブジェクト）を作成し、「r11_gen.cs」をアタッチする。

⑥で作成した空のオブジェクトのコンポーネントの外部変数にタコのプレハブを登録する。

⑦実行を確認する。

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

public class r11_gen : MonoBehaviour
{
    public GameObject tako_p;

    void Start()
    {
        InvokeRepeating("tako_fuyasu",1f, 1f);     // １秒ごとに「tako_fuyasu()」を実行
    }

    public void tako_fuyasu()
    {
        int x,y;
        GameObject tk = Instantiate( tako_p );     // プレハブ tako_p からインスタンス tk を複製
        x = Random.Range(-7, 8);                        // X座標位置を -7 - 7 の乱数で取得
        y = Random.Range(-4, 5);                        // Y     〃     -4 - 4     〃
        tk.transform.position = new Vector3( x, y, 0);    // インスタンス tk 位置指定
    }
}
```

コピー元のプレハブ（コンポーネントで登録）



スクリプト	r11_gen.cs
アタッチ先	管理オブジェクト

発展①　キャラクタを回転させるようにして、それを増殖するようにしてみよう！

(スクリプトを作成しタコのプレハブにアタッチします)

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

public class r11_tako : MonoBehaviour
{
    void Update()
    {
        transform.                                  // オブジェクト回転
    }
}
```

スクリプト	r11_tako.cs
アタッチ先	タコのプレハブ

発展②　オブジェクトをクリックしたら消す（Destroy）ようにしてみよう！（例題６を参考に考えます）

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

public class r11_tako : MonoBehaviour
{
    void Update()
    {
        transform.                                  ; // オブジェクト回転
    }

    void OnMouseDown()              // マウスボタンが押されたら
    {
        Destroy(gameObject);        // オブジェクト消去
    }
}
```

スクリプト	r11_tako.cs
アタッチ先	タコのプレハブ

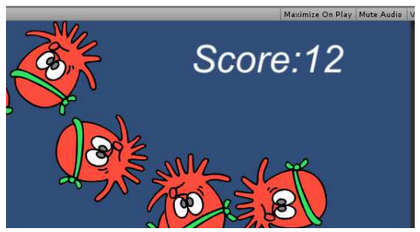
①インスペクターの「Add Component」ボタンをクリックし「Physics 2D」　→「Circle Collider 2D」と選択します。オブジェクトの周りに「円形の領域」が示されます。

②スクリプトを追加し、たこの「プレハブ」にアタッチ、実行します。

発展③　例題１０を参考に、クリックしたタコの数をカウントする機能を追加します。テキスト表示ＵＩを作成しスクリプトをアタッチします。

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

public class r11_score : MonoBehaviour
{
    int score = 0;          // スコア初期値０
    GameObject scoreText;

    void Start()
    {
        scoreText = GameObject.Find("Text");
    }

    void Update()
    {
        scoreText.GetComponent<Text>().text = "Score:" + score; // スコアを表示
    }

    public void AddScore()      // スコアを増やす関数
    {
        score += 1;
    }
}
```

スクリプト	r11_score.cs
アタッチ先	Canvas

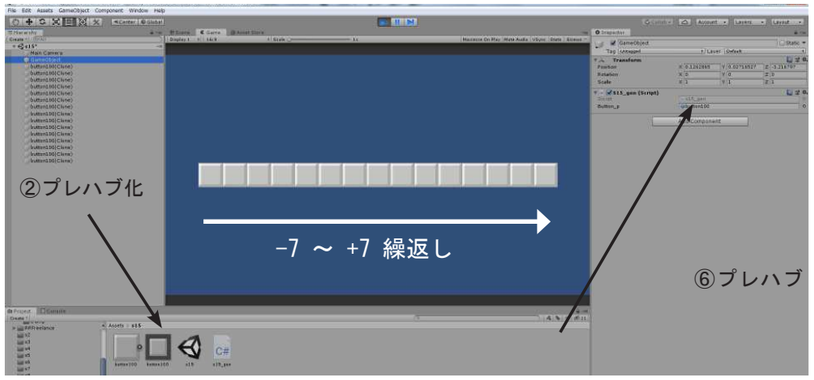



```
    void OnMouseDown()  // マウスボタンが押されたら
    {
        Destroy(gameObject);  // オブジェクト消去
        GameObject.Find("Canvas").GetComponent<r11_score>().AddScore();
    }       // r11_score の AddScore 関数を呼び出す
```

スクリプト	r11_tako.cs
アタッチ先	タコのプレハブ

例題 12　繰返し処理（インスタンスの複製）

規則的な繰り返しでスプライトを配置する手法を学びます。



シーン名　r12roop

①ボタン画像「button100.png」を配置する。

②オブジェクト「button100」をヒエラルキーウインドからプロジェクトウインドへドラッグ「プレハブ化」し、名前を「button100_p」と変更する。

③元のオブジェクト「button100」を削除する。

④ボタンを横一列に増やすスクリプトを作成する。(r12_gen.cs)

⑤ヒエラルキーウインドで空のオブジェクト「GameObject」(管理オブジェクト)を作成し、「r13_gen.cs」をアタッチする。

⑥で作成した空のオブジェクトのコンポーネントの外部変数「button_p」にボタンのプレハブを登録する。

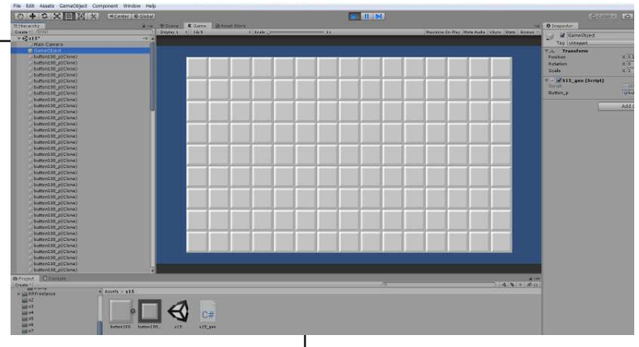
⑦実行を確認する。

横一列に配置

スクリプト	r12_gen.cs
アタッチ先	管理オブジェクト

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

public class r12_gen : MonoBehaviour
{
    public GameObject button_p;

    void Start()
    {
        int x,y;
        for (x = -7; x <= 7; x++)    //X 座標を -7 ～ +7 まで繰り返す
        {
            GameObject bt = Instantiate(button_p);
            bt.transform.position = new Vector3(x, 0, 0);
        }
    }
}
```

縦横に配置（二重ループ）



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

public class r12_gen : MonoBehaviour
{
    public GameObject button_p;

    void Start()
    {
        int x,y;
        for (y = 4; y >= -4; y--)  // 縦の繰り返し
        {
            for (x = -7; x <= 7; x++)  // 横の繰り返し
            {
                GameObject bt = Instantiate(button_p);
                bt.transform.position = new Vector3(x, y, 0);
            }
        }
    }
}
```

スクリプト	r12_gen.cs
アタッチ先	管理オブジェクト

⑧縦横に配置する様にスクリプトを変更して（二重ループ）実行を確認する。

発展①

ボタンをクリックしたら消える様にしてみよう！

演習 8

スペースキーを押すごとに、宇宙船からビームを発射するシーンを作成せよ。

シーン名　e08beem

①宇宙船の画像「syatoru1.png」とビームの画像「beem2.png」を大きさと位置を調整して配置する。

②ビームのオブジェクトを移動させるスクリプト「e06_beem.cs」を作成し、実行を確認する。

③オブジェクト「beem2」をヒエラルキーウインドからプロジェクトウインドへドラッグし「プレハブ化」し、名前を「beem2_p」と変更する。

④プレハブ化したら元のオブジェクトは消去します。

⑤ビームを複製するスクリプトを作成して、宇宙船にアタッチします。

⑥プレハブをコンポーネントに登録します。

スクリプト	e08_beem.cs
アタッチ先	ビーム

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

public class e06_beem : MonoBehaviour
{
    void Update()
    {
        [                                    ] // ビームを上方に移動
    }
}
```

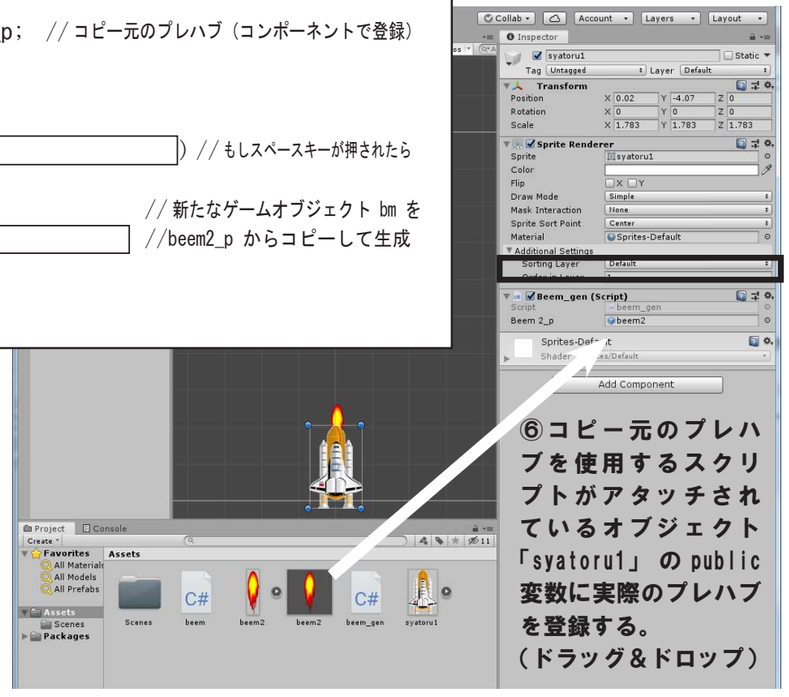
スクリプト	e08_gen.cs
アタッチ先	宇宙船

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

public class e06_gen : MonoBehaviour
{
    public GameObject beem2_p;  // コピー元のプレハブ（コンポーネントで登録）

    void Update()
    {
        if ([                                    ]) // もしスペースキーが押されたら
        {
            GameObject bm;                  // 新たなゲームオブジェクト bm を
            bm = [                          ]; //beem2_p からコピーして生成
        }
    }
}
```

複製されたインスタンスは画面外へ移動しても存在し続けます。これがあまり多くなるとシステムの負担となり動作が遅くなります。使わなくなったインスタンスは

Destroy(gameObject);

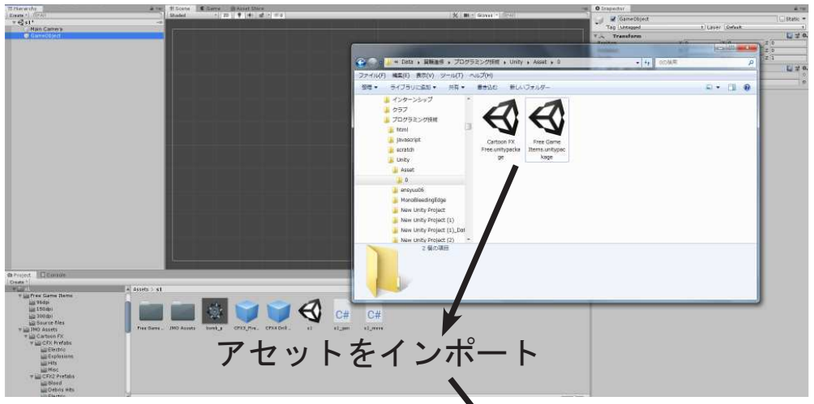
で消しておく必要があります。



例題 13　アセットストア（AssetStore）を使う

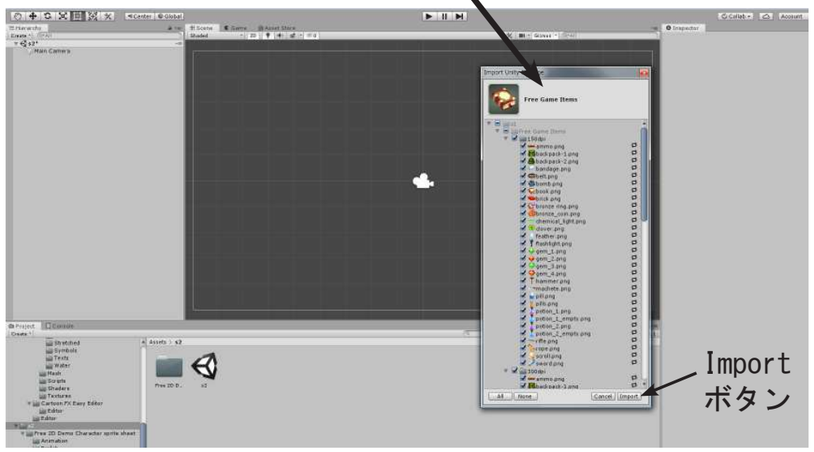
シーン名　r13aset

アセットストアとは、Unity で使用できる素材や画像などを購入できるショップです。ストアでダウンロードできるものは、有料が多いですが、無料で使える素材もあります。ストアを使用するためにはアカウントの登録が必要になりますが、ダウンロード済のものをインポートすることも可能です。

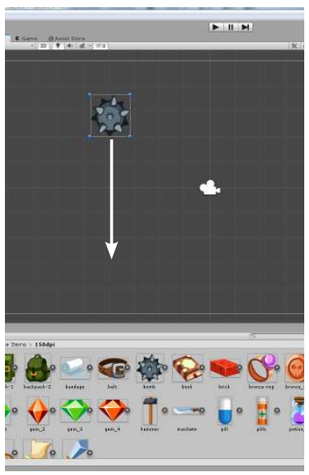


①ダウンロード済のアセット「Free Game Items.unitypackage」をプロジェクトウインドにインポートします。

フォルダ Asset 直下にインポートされるので、わかりやすいように、シーンのフォルダへ移動しておきます。
「FreeGameItems/150dpi」フォルダの内容を確認してみましょう。



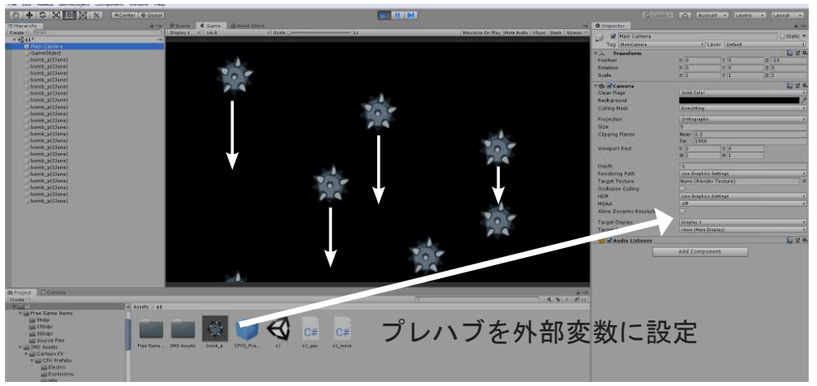
②Asset の中の「bomb」をシーンへ配置しマウスクリックを検出するために「Circle Collider」コンポーネントを設定、下方へ移動するスクリプト「r1_move.cs」をアタッチします。



③オブジェクトをプレハブ化し元のオブジェクトは消去しておきます。

スクリプト：r13_move.cs
アタッチ先：爆弾

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

public class r13_move : MonoBehaviour
{
    void Update()
    {
        transform.Translate(0, -0.05f, 0);
    }
}
```

スクリプト：r13_gen.cs
アタッチ先：管理オブジェクト

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

public class r13_gen : MonoBehaviour
{
    public GameObject bomb_p;   // 外部変数

    void Start()
    {
        InvokeRepeating("bakudan", 0.5f, 0.5f);  // １秒毎に実行
    }

    public void bakudan()
    {
        int x;
        GameObject bo = Instantiate(bomb_p);   // プレハブから複製
        x = Random.Range(-7, 7);               // 横位置は乱数
        bo.transform.position = new Vector3(x, 5, 0);
    }
}
```

スクリプト：r13_move.cs
アタッチ先：爆弾

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

public class r13_move : MonoBehaviour
{
    public GameObject ex_p;    // 爆発プレハブの外部変数

    void Update()
    {
        transform.Translate(0, -0.05f, 0);
    }

    void OnMouseDown()    // マウスがクリックされたら
    {                     // 爆発エフェクトを複製
        GameObject ex = Instantiate(ex_p, transform.position,
                                    Quaternion.identity); ⑦
        Destroy(gameObject);          // 爆弾オブジェクトを削除
        Destroy(ex.gameObject, 1.0f); // エフェクトを１秒後に削除
    }
}
```

GameObject ex = Instantiate(ex_p , transform.position , Quaternion.identity);

⑦第一引数に Prefab、第二引数にインスタンスを生成する位置、第三引数には回転角を指定します。

④０．５秒ごとにランダムな位置から落下するスクリプト「s1_gen.cs」を作成し、空のオブジェクト（管理オブジェクト）にアタッチします。

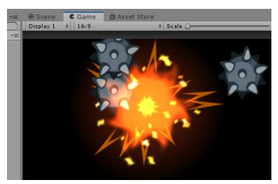
⑤スクリプトのコンポーネントの外部変数にプレハブを設定し、実行を確認します。

⑥ダウンロード済のアセット「Cartoon FX Free.unitypackage」をプロジェクトにインポートします。これは「爆発系のエフェクト」を集めたアセットになります。インポート後は「JMO Asset」という名前で登録されます。

フォルダ Asset 直下にインポートされるので、わかりやすいように、シーンのフォルダへ移動しておきます。

フォルダの内容を確認してみましょう。

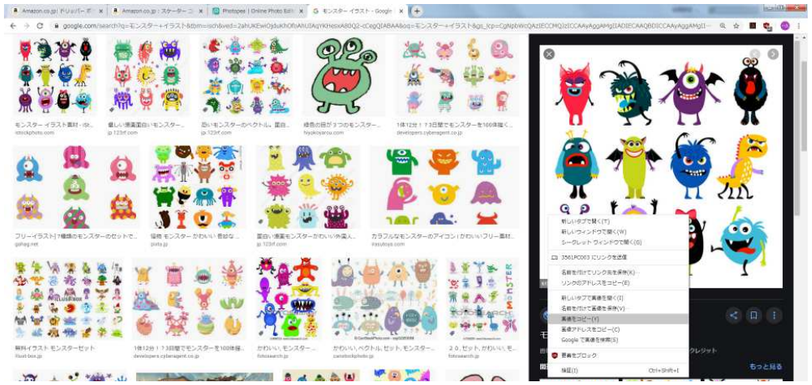
⑦落下するオブジェクトがクリックされたら、オブジェクト位置に、爆発するエフェクト「CFX3_Fire_Explosion」を複製するスクリプトを追加します。

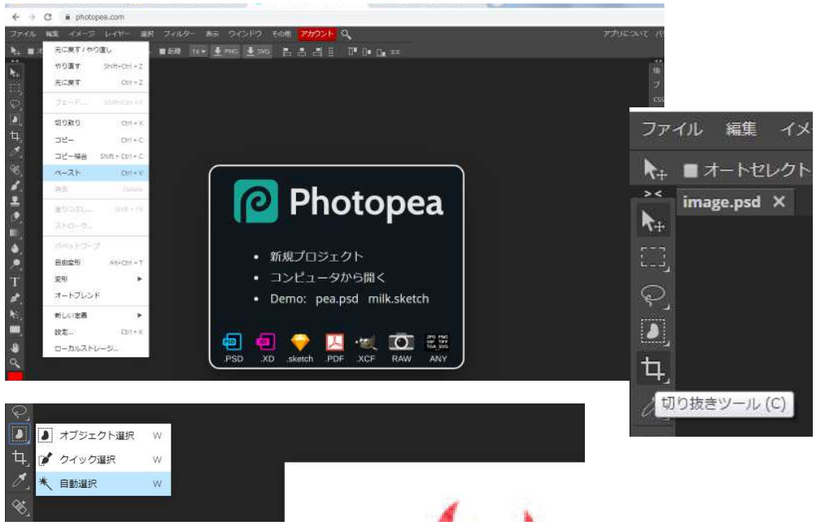


「CFX3_Fire_Explosion」エフェクトはプレハブとして提供されています。

⑧スクリプトコンポーネントの外部変数にエフェクトのプレハブをアタッチし実行してみましょう！

画像編集ソフト「Photopea」の使い方１





オリジナルゲームを作成するには素材集め、加工も大切です。
Web 上で画像の加工ができる「Photopea」を使ってみましょう。

透過 PNG の作成

①インターネットで画像を探します。

②画像上で右クリック「コピー」を選択します。

③ Photopea に切替え、編集メニューから「ペースト」を実行します。

④「切抜きツール」を使用して必要な部分を切り抜きます。

⑤「自動選択ツール」で背景となる部分を選択し、消去（ＤＥＬキー）します。

背景部分がチェックパターンになっていれば透過の状態です

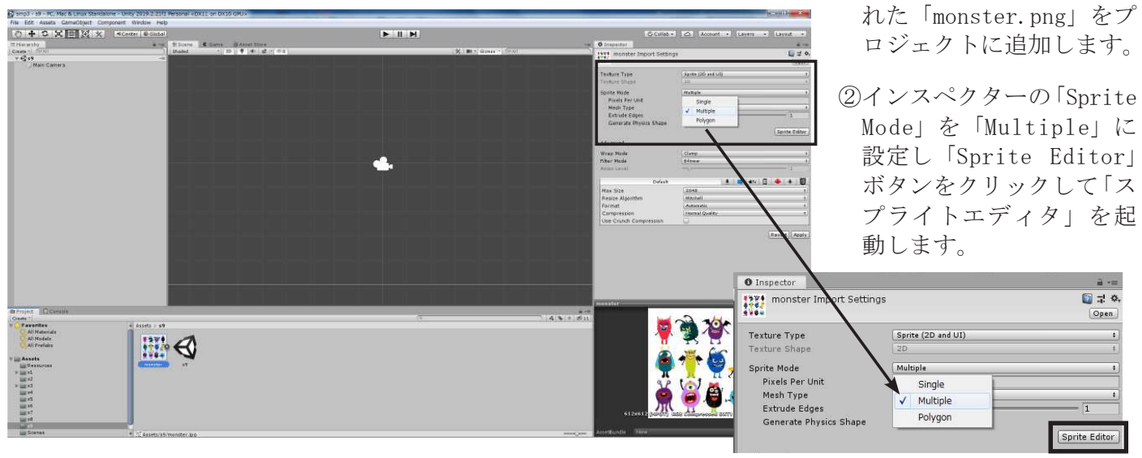
⑥「ファイル」メニューから「別名で保存」を選択し、ファイルの種類で「PNG」を選びます。

⑦名前をつけて保存します。

例題 14 スプライトエディタ（画像の切り分け）

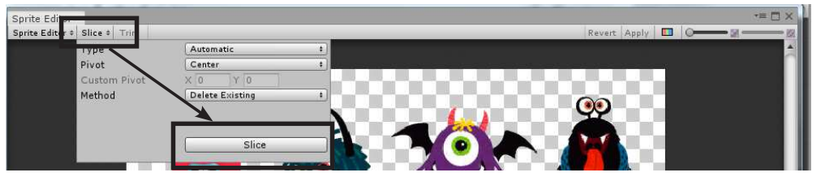
シーン名　r14roop

いくつかの画像をひとつにまとめたものをアトラス画像と呼びます。それを切り分け、それぞれの単独のスプライトとして使用できます。



①いくつかのモンスターの画像がひとつにまとめられた「monster.png」をプロジェクトに追加します。

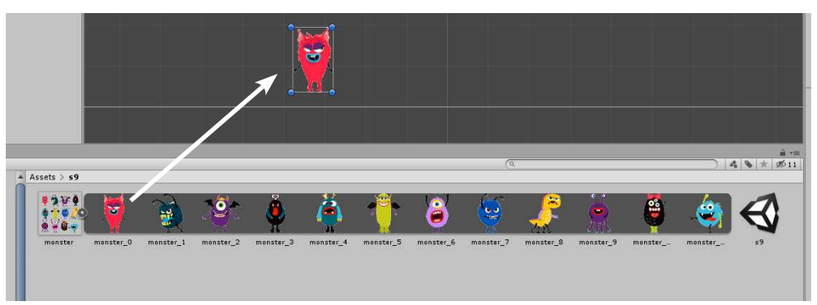
②インスペクターの「Sprite Mode」を「Multiple」に設定し「Sprite Editor」ボタンをクリックして「スプライトエディタ」を起動します。



③スプライトエディタの「Slice」メニューを選択し「Slice」ボタンをクリックします。

④右上の「Appry」ボタンをクリックすると、自動で画像が切り分けられます。それぞれのスプライトを確認してみましょう。

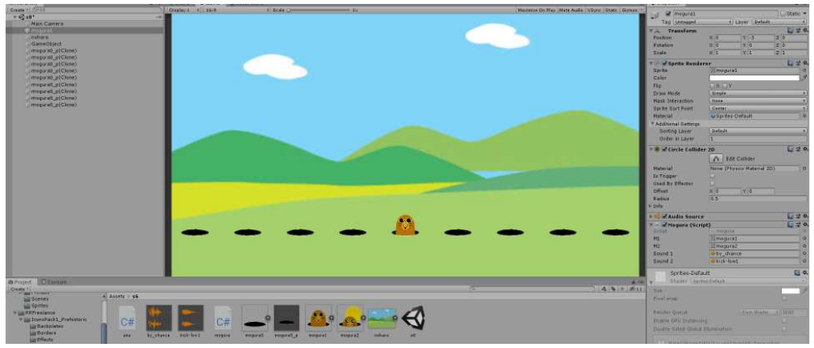
⑤スプライトエディタを閉じてプロジェクトウインドの画像の「▶」をクリックすると、切り分けられたスプライトが展開され、ひとるひとつのスプライトとして扱うことができます。

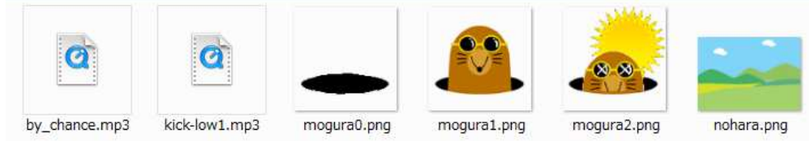


※画像ファイルが「透過ＰＮＧ」でないとうまくスライスできません…

ミニゲーム１

ここまでの例題を組み合わせて、簡単な「モグラたたき」ゲームを作ってみましょう！





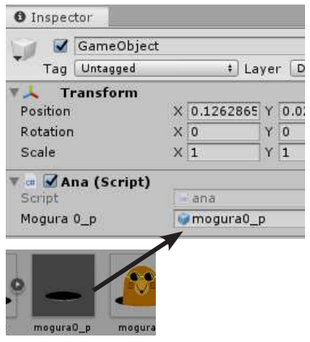
シーン名　g01mogu

①使用する画像と効果音ファイルをアセットに追加します。

②穴の画像「mogura0.png」をシーンへ配置した後プレハブ化し「mogura9_p」と名前をつけます。元のオブジェクトは消します。

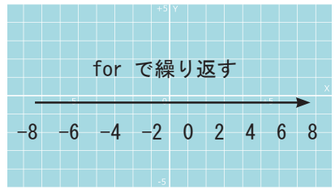
③プレハブを複製して穴を９つ生成するスクリプトを作成し、空のオブジェクトにアタッチします。

④空の「GameObject」を作成、インスペクターの外部変数「mogura0_p」に「mogura0.png」のプレハブを設定し、実行を確認しましょう。



穴を並べて生成する

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

public class g01_ana : MonoBehaviour
{
    public GameObject mogura0_p;

    void Start()
    {
        for (int x = -8; x <= 8; x+=2)      // 穴を連続して配置
        {
            GameObject tk = Instantiate(mogura0_p);
            tk.transform.position = new Vector3(x, -3, 0);
        }
    }
}
```

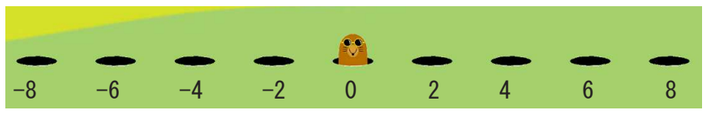


スクリプト	g01_ana.cs
アタッチ先	管理オブジェクト

１秒毎ランダムに移動

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

public class g01_mogura : MonoBehaviour
{
    void Start()
    {
        InvokeRepeating("                    ", 1f, 1f);   // １秒ごとに mogu_move() を呼び出す
    }

    void mogu_move()
    {
        int x;

        x = 
        transform.position = new Vector3( x*2, -3, 0);    //-4 ～ +4 までの乱数を発生
                                                          // モグラのＸ座標値を -8 ～ +8 へ移動
    }
}
```

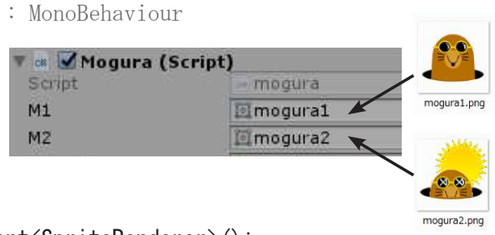
スクリプト	g01_mogura.cs
アタッチ先	モグラ



⑤モグラが出現した画像「mogura1.png」を配置し、１秒ごとにランダムに移動するスクリプトを作成し、アタッチ、実行を確認します。

クリックでスプライト切替

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

public class g01_mogura : MonoBehaviour
{
    SpriteRenderer spr;

    public Sprite m1;
    public Sprite m2;

    void Start()
    {
        spr = GetComponent<SpriteRenderer>();
        InvokeRepeating(                    );
    }
    void mogu_move()
    {
        int x;
        x = 
        transform.position = new Vector3( x*2, -3, 0);
        spr.sprite = m1;     // 移動したら元の画像に切替
    }
    void OnMouseDown()       // ボタンクリックされたら
    {
                             // スプライトを叩かれているものに切換
    }
}
```

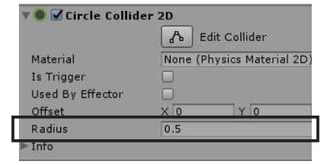


スクリプト	g01_mogura.cs
アタッチ先	モグラ

⑥クリック時にスプライトを切り替えるために、スクリプトを追加し、２つの画像を Sprite 型の外部変数として登録します。

⑦モグラのオブジェクトに「Circle Collider 2D」をアタッチし、大きさを調整します。

⑧実行して動作を確認します。



Radus（半径）

効果音を再生する

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

public class g01_mogura : MonoBehaviour
{
    SpriteRenderer spr;

    public Sprite m1;
    public Sprite m2;

    public AudioClip sound1;
    public AudioClip sound2;
    AudioSource aud;

    void Start()
    {
        spr = GetComponent<SpriteRenderer>();
        InvokeRepeating("mogu_move", 1f, 1f);
        aud = GetComponent<AudioSource>(); //AudioSource コンポーネントを取得
    }
    void mogu_move()
    {
        int x;
        x = Random.Range(-4, 4);
        transform.position = new Vector3( x*2, -3, 0);
        spr.sprite = m1;
        aud.PlayOneShot(sound1);  // モグラ出現サウンドを再生
    }
    void OnMouseDown()
    {
        spr.sprite = m2;
        aud.PlayOneShot(sound2);  // 叩くサウンドを再生
    }
}
```

スクリプト	g01_mogura.cs
アタッチ先	モグラ

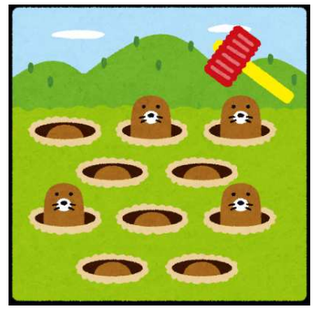
効果音を再生するスクリプトを追加します。
　移動時 … by_chance.mp3
　叩く時 … kick-low1.mp3

⑨AudioClip コンポーネントを追加します。

⑩AudioClip 型の外部変数を２つ用意したスクリプトを追加し、コンポーネントで実際のサウンドファイルと関連付けます。

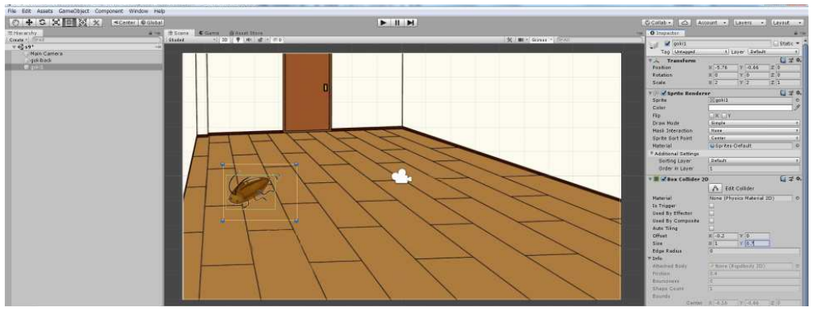
⑪実行して動作を確認します。

移動スピード等を変更してみましょう！



ミニゲーム２

「ゴキブリバトル」ゲーム
ランダムに増殖するゴキブリと戦え！！




gokiback.png


goki2.png


goki1.png


kick-low1.mp3


gasagasa.mp3

ゴキブリをランダムに移動

スクリプト	g02_goki.cs
アタッチ先	ゴキブリ

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

public class g02_goki : MonoBehaviour
{
    void Update()
    {
        float x, y;            // 現在位置から±0.3の範囲でランダムに移動する

        x = Random.Range(-0.3f, 0.3f);        // ±0.3の乱数
        y =                                   // ±0.3の乱数
        transform.                            // 乱数分だけ移動
    }
}
```

クリックでスプライトを変更

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

public class g02_goki : MonoBehaviour
{
    SpriteRenderer spr;
    public Sprite m1;
    public Sprite m2;

    void Start()
    {
        spr = GetComponent<SpriteRenderer>();
    }

    void Update()
    {
      ・・・省略
    }

    void OnMouseDown()                    // マウスクリックしたら
    {
        spr.sprite = m2;                  // スプライトを「叩かれ画像」に
        Destroy(gameObject, 0.5f);        //0.5秒後に消滅
    }
}
```


goki1.png


goki2.png

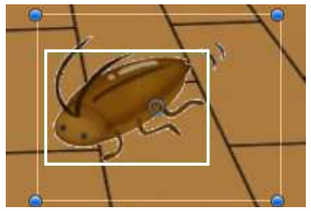
スクリプト	g02_goki.cs
アタッチ先	ゴキブリ

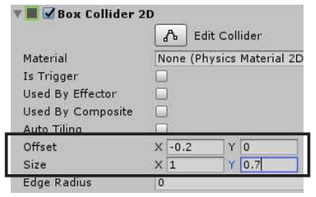
シーン名 g02goki

①使用する画像と効果音ファイルをアセットに追加します。

②床の画像「gokiback.png」をシーンへ配置します。

③ゴキブリの画像「goki1.png」を配置し、マウスクリックを取得するための「Box Collider 2D」をアタッチし、領域の位置と大きさを調整します。





④ゴキブリオブジェクトをランダムに移動するスクリプトを作成し、ゴキブリにアタッチします。

⑤クリックしたらスプライトを変更するスクリプトを追加し、コンポーネントの外部変数に画像ファイルを登録します。

⑥マウスをクリックしたらスプライトを変更し、0.5秒後に消滅するスクリプトを付け加えます。

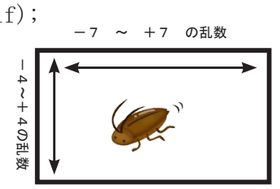
効果音を再生

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

public class g02_goki : MonoBehaviour
{
    SpriteRenderer spr;
    public Sprite m1;
    public Sprite m2;

    public AudioClip sound1;
    public AudioClip sound2;
    AudioSource aud;        AudioSource 型の変数

    void Start()
    {
        spr = GetComponent<SpriteRenderer>();
        aud = GetComponent<AudioSource>();    //AudioSource を取得
        aud.PlayOneShot(sound1);              // 出現した時に再生
    }

    void Update()
    {
        float x, y;
        x = Random.Range(-0.3f, 0.3f);
        y = Random.Range(-0.3f, 0.3f);
        transform.Translate(x, y, 0);
    }

    void OnMouseDown()
    {
        aud.PlayOneShot(sound2);          // クリックされた時に再生
        spr.sprite = m2;
        Destroy(gameObject, 0.5f);
    }
}
```

スクリプト	g02_goki.cs
アタッチ先	ゴキブリ

１秒ごとにゴキブリを増殖

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

public class g02_gen : MonoBehaviour
{
    public GameObject goki_p;
    void Start()
    {                    // １秒ごとにプレハブからインスタンスを生成
        InvokeRepeating("goki_gene", 1f, 1f);
    }
    void goki_gene()
    {
        int x, y;
        x =                                   －７ ～ ＋７ の乱数
        y =                                   －４～＋４の乱数
        GameObject gk = Instantiate(goki_p);
        gk.transform.position = new Vector3(x, y, 0);
    }
}
```



スクリプト	g02_gen.cs
アタッチ先	管理オブジェクト

⑦AudioClip コンポーネントを追加します。

⑧ゴキブリが出現した時とクリックされた時に効果音を再生するスクリプトを追加します。


gasagasa.mp3
出現時


kick-low1.mp3
クリック時

AudioSource 型の外部変数に効果音ファイルを設定。

以上で１匹のゴキブリに対する処理設定は完了です。

続いてゴキブリを増殖する処理を追加します。

⑧ゴキブリをプレハブ化し「goki_p」と名前をつけ、元のオブジェクトは削除しておきます。

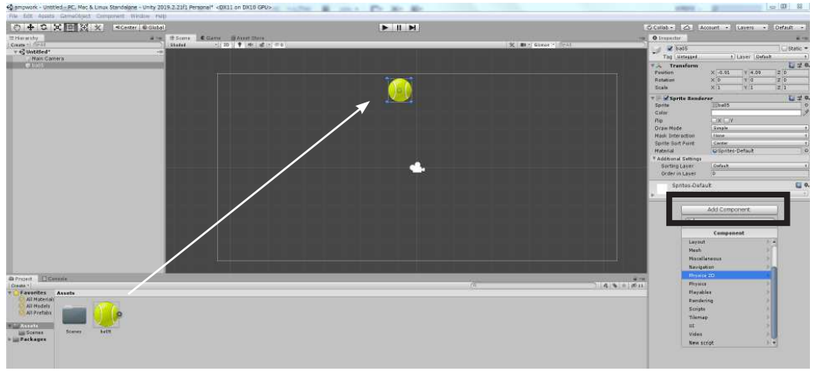
⑨１秒ごとにゴキブリを増殖するスクリプトを作成、空のオブジェクト「GameObject」(管理オブジェクト)を作成しアタッチします。

⑩空の「GameObject」のインスペクターの外部変数「goki_p」にプレハブを設定し、実行を確認します。

例題 15　物理エンジン（Physics）

シーン名　r15phys

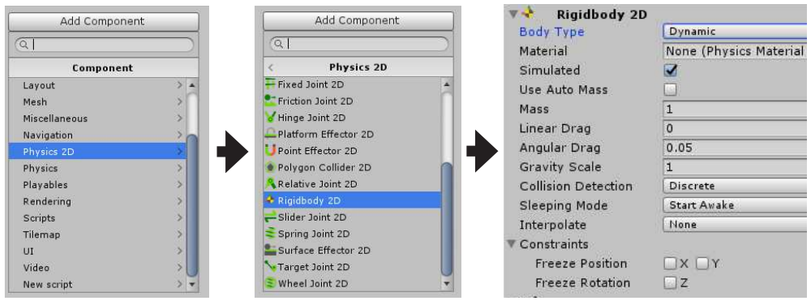
物理エンジンとは、オブジェクトに物理的な挙動（落下、衝突など）をさせるためのシミュレーションライブラリです。物理エンジンを使用するとオブジェクトの質量や摩擦係数、重力などを考慮して動きを計算するため、オブジェクトにリアルな挙動をさせることができ、Ｕｎｉｔｙの大きな特徴となっています。



①ボールの画像をシーンに配置し、インスペクターの「Add Component」ボタンをクリックします。

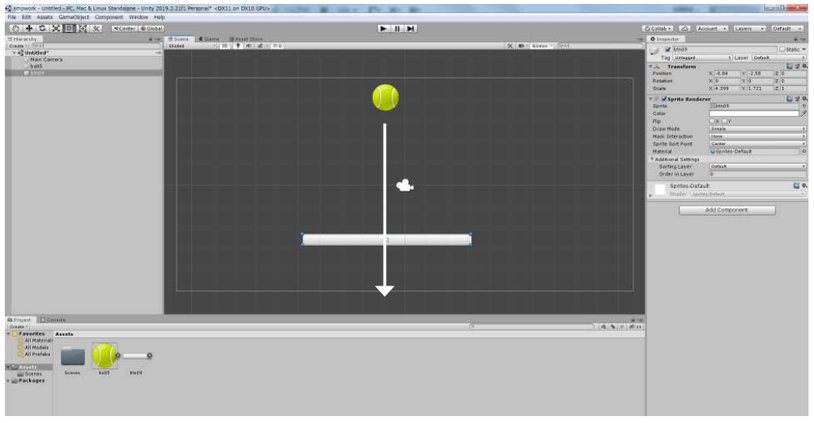
②「Physics 2D」 から「Rigidbody 2D」を選択し、実行してみましょう！

オブジェクトに重力が設定され自然落下します。ただし、このままでは画面の外へ出てもずっと落ち続けたままです。



③ボールの下にブロックの画像を配置し、大きさを調整し、実行します…

…ボールがブロックをすり抜けてしまいます。



④２つのオブジェクトに「衝突判定」のコンポーネント「Collder 2D」をアタッチします。

Colliderは「衝突検知」のための「領域指定」です。

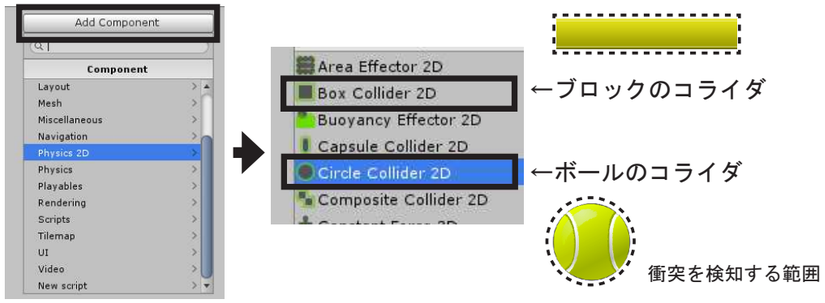
ボールへは円形のコライダ、ブロックへは四角形のコライダを設定します。

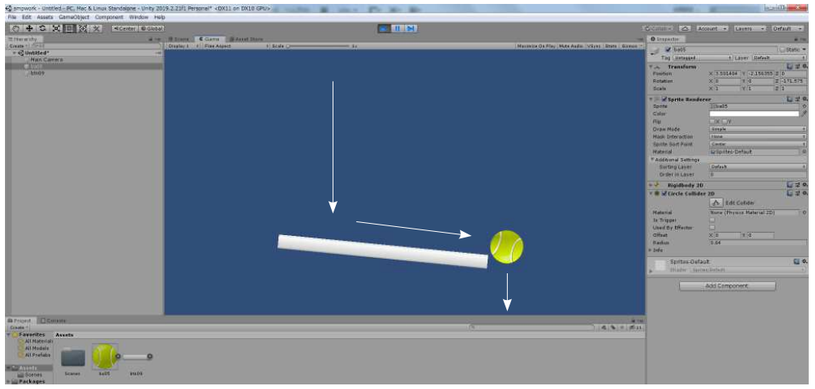
⑤実行してみましょう。

ボールがブロックで止まりました！！

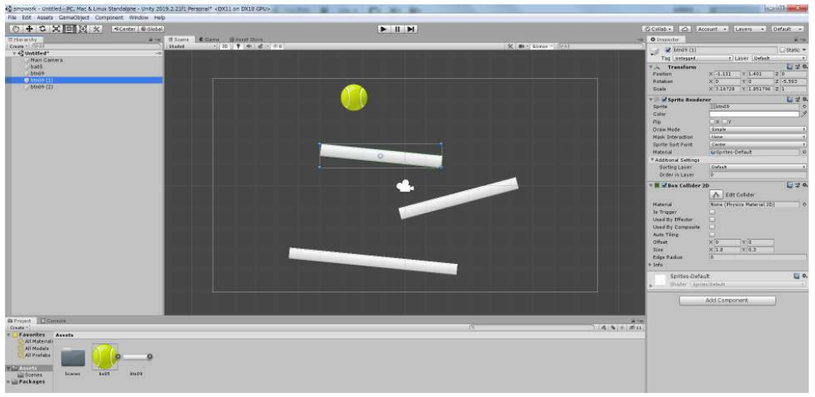
Unityではこのように、ノンコーディング（スクリプトなし）で物体を動かすことができます。

衝突検知（コライダ）の設定



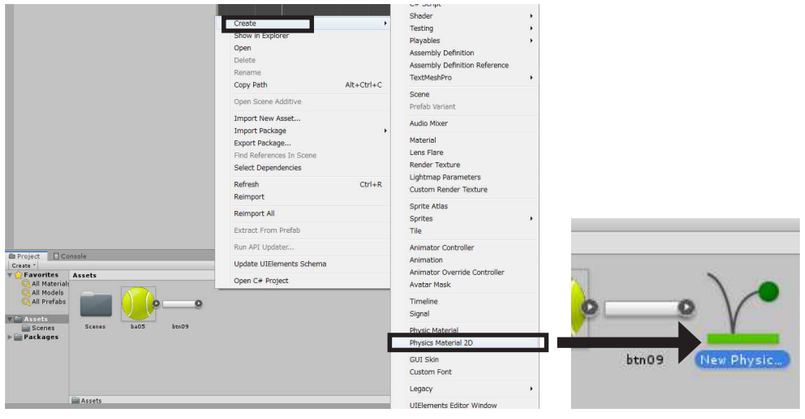


⑥ブロックをやや傾けて実行してみましょう。ボールはブロックに沿って転がります。



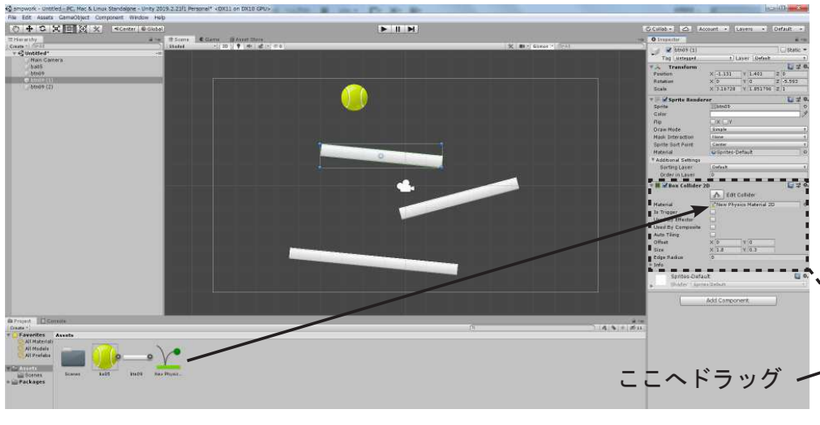
⑦ブロックを例のようにいくつか配置して、ボールが転がるようにしてみましょう。

続いてボールを弾ませてみましょう。



⑧プロジェクトウインドで右クリックし「Create」から「Physics Material 2D」を選択します。

⑨インスペクターウインドで「Bounciness」を「１」に設定します。１はエネルギー損失なしで弾む設定です。

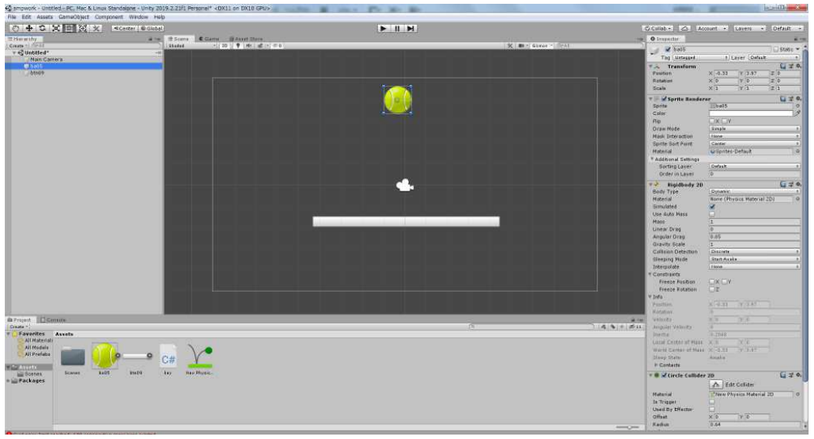


⑩ヒエラルキーウインドでボールオブジェクトを選択し、インスペクターの「Box Collider 2D」コンポーネントの「Material」に、プロジェクトウインドから設定した物理マテリアルをアタッチし、実行してみます。

演習９

ブロックをキー入力で回転させ、転がるボールをコントロールするシーンを作成せよ。

シーン名　e09ball



①ボール画像とブロックの画像を配置し、スケールを調整します。

②ボールオブジェクトに「Rigidbody 2D」と「Circle Collider 2D」コンポーネントをアタッチします。

③ブロックオブジェクトに「Box Collider 2D」コンポーネントをアタッチし、実行して動作を確認します。

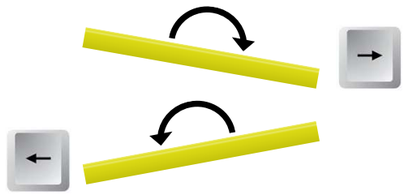
④「← →」のキー入力でブロックを左右に傾けるスクリプト（e09_key.cs）を作成、ブロックにアタッチし、実行する。

重要

Phsyics を使って衝突検知をする場合、
- 両方のオブジェクトに Collider コンポーネントをアタッチ
- 少なくとも一方には Rigidbody コンポーネントをアタッチ

する必要があります。

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

public class e09_key : MonoBehaviour
{
    void Update()
    {
        if (                                    )   // もし「←」が押されたら
        {
                                                    // 左回りに０．５度回転
        }
        if (                                    )   // もし「→」が押されたら
        {
                                                    // 右回りに０．５度回転
        }
    }
}
```

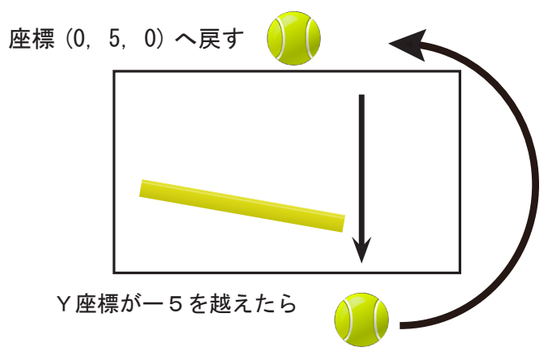


スクリプト	e09_key.cs
アタッチ先	ブロック

発展

ボールが画面から下へ落ちたら（Ｙ座標が－５より小さくなったら）上へ戻す（Ｙ座標＋５）スクリプト（ball.cs）を作成し、ボールオブジェクトにアタッチして実行してみよう。

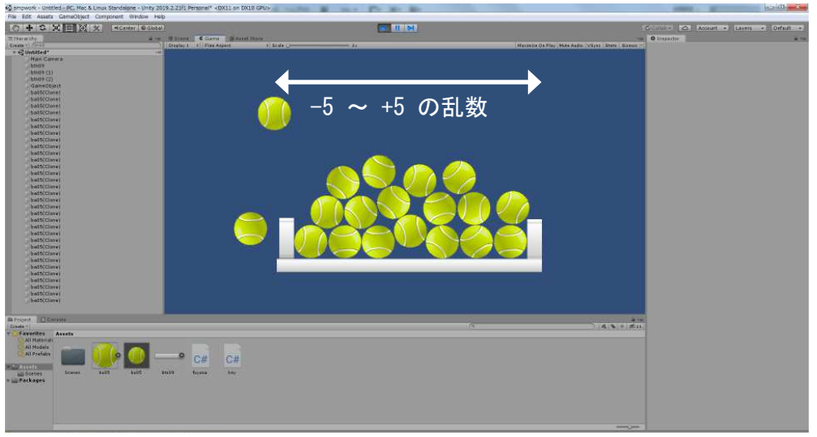
※Phisics Material 2D などは自由に設定してみよう



演習 10

下の様にブロックを配置し、上方からボールを０．５秒ごとに複製して落下させるシーンを作成する。

シーン名　e10ball



①ボール画像を配置し、スケールを調整して、ボールオブジェクトに「Rigidbody 2D」と「Circle Collider 2D」コンポーネントをアタッチします。

②ブロックオブジェクトを３つ、図のように配置しそれぞれに「Box Collider 2D」コンポーネントをアタッチ、実行して動作を確認します。先に Collider をアタッチしたブロックをコピーして作成しても構いません。

③ボールオブジェクトをプレハブ化し「ball_p」と名前をつけて、元のボールオブジェクトは削除します。

④空の「GameObject」（管理オブジェクト）を作成し、ボールを複製するスクリプト（e10_fuyasu.cs）を作成してアタッチします。

⑤「GameObject」の Script コンポーネントの外部変数「ball_p」に、プレハブ「ball_p」を設定します。

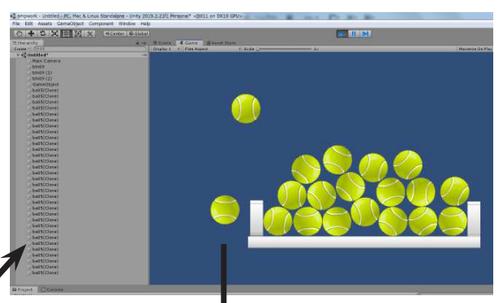
⑥実行してみましょう！。複製時間の変更や物理マテリアルの追加などもしてみましょう！

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

public class e10_fuyasu : MonoBehaviour
{
    public GameObject ball_p;

    void Start()
    {                           //0.5 秒ごとに ball_fuyasu を呼び出す
        InvokeRepeating(                                    );
    }

    public void ball_fuyasu()
    {
        int x, y;               // プレハブ ball_p をインスタンス化（複製）
        GameObject ba =                                     ;
        x = Random.Range(-5, 5);
        ba.transform.position =                             ;
    }                                   // ボールを上のランダム位置に移動
}
```

スクリプト	e10_fuyasu.cs
アタッチ先	管理オブジェクト

発展

このままでは無限にボールが増え続けてしまうので、ボールが画面外へ落ちたら（Ｙ座標が－５より小さくなったら）ボールオブジェクトを削除するスクリプト（ball.cs）を作成し、**ボールプレハブ**（ball_p）にアタッチして実行してみよう。

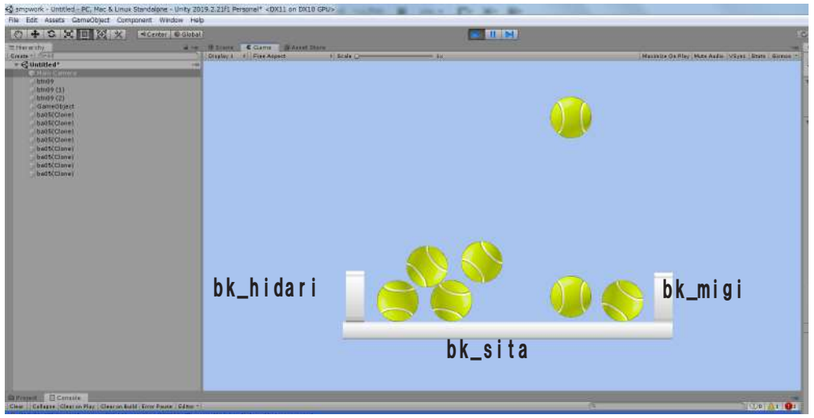


ヒエラルキーウインドでボールインスタンスの生成と削除される様子を確認しよう。

Ｙ座標が－５を越えたら削除（Destroy）する。

例題 16　親子オブジェクト

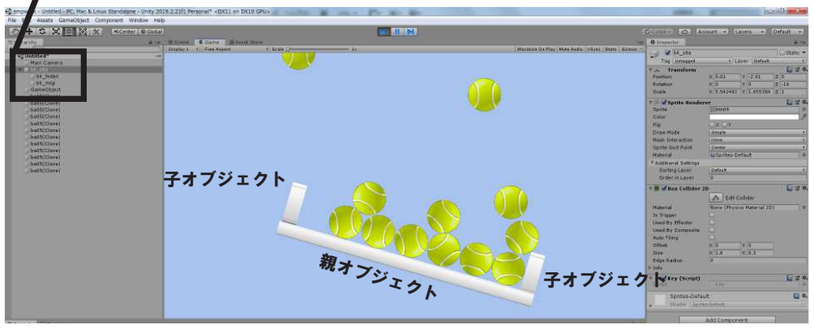
シーン名　e10ball



「演習１０」に３つのブロックを回転させるスクリプトを追加してみます。
分かりやすいように各ブロックのオブジェクト名を変更します。

①キー入力で回転させるスクリプトを作成（演習８と同じ）して「bk_sita」にアタッチして実行してみます。
左の様に bk_sita　だけが回転してしまいます。

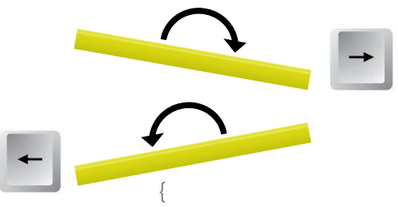
３つのブロックを連動して動作させるためにはオブジェクトに「親」と「子」の関係を設定します。



②ヒエラルキーで「bk_migi」と「bk_hidari」をそれぞれ「bk_sita」へドラッグ＆ドロップします。

→連動して動作します。

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

public class e09_key : MonoBehaviour
{
    void Update()
    {
        if (                                    )   // もし「←」が押されたら
        {
                                                    // 左回りに０．５度回転
        }
        if (                                    )   // もし「→」が押されたら
        {
                                                    // 右回りに０．５度回転
        }
    }
}
```

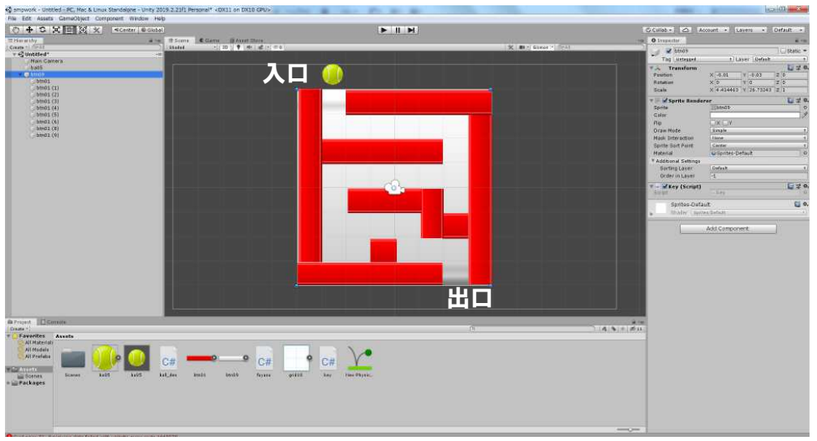


スクリプト	e09_key.cs
アタッチ先	ブロック

演習 11

ブロックを並べて迷路を作り、キー入力で回転させ、転がるボールを出口から出すシーンを作成せよ。

シーン名　e11maze



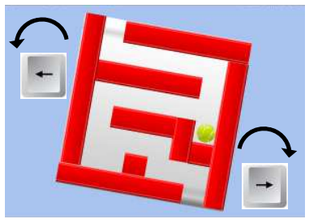
①ブロックの画像を拡大してベースとして配置します。

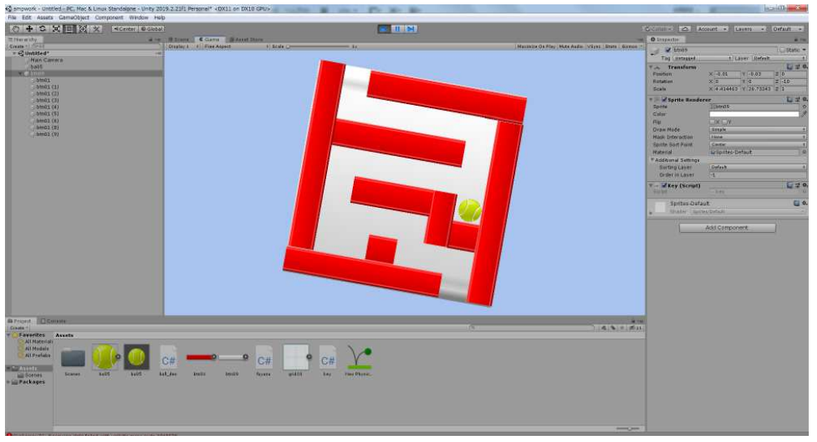
②別色のブロックをベースの上に、迷路の壁として配置し「Box Collider 2D」コンポーネントをアタッチします。

③ベースのブロックを「親」壁のブロックを「子」に設定します。


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

public class e11_key : MonoBehaviour
{
    void Update()
    {
        if (                                    )   // 「←」が押されたら
        {
                                                    // 左回りに１度回転
        }
        if (                                    )   // 「→」が押されたら
        {
                                                    // 右回りに１度回転
        }
    }
}
```

スクリプト	e11_key.cs
アタッチ先	ベースブロック





④ボール画像を配置し、スケールを調整して、「Rigidbody 2D」と「Circle Collider 2D」コンポーネントをアタッチします。

実行して動作を確認します。
Physics Material などは適宜設定します。

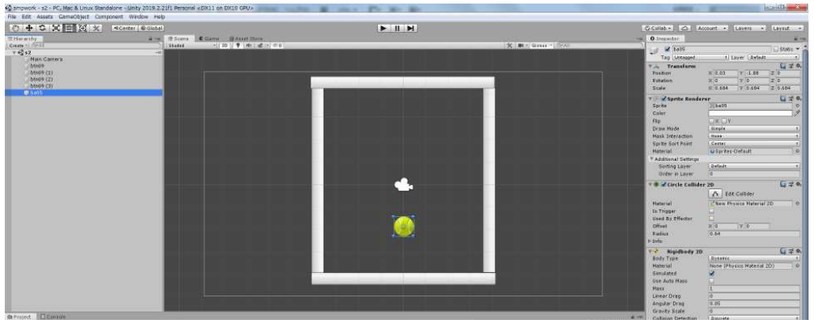
⑤キー入力で迷路を回転させるスクリプトを作成し、親オブジェクトにアタッチします。

実行してみましょう！

例題 17　物理演算によるオブジェクト制御

シーン名　r17ball

Rigidbody をアタッチしたオブジェクトを動かすには「力を与える」処理が必要になります。

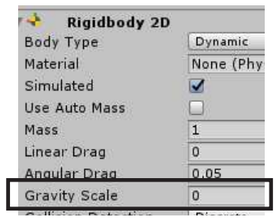


①ブロックを上下左右に壁として配置し、それぞれに「Box Collider 2D」コンポーネントをアタッチします。

②ボール画像を配置し「Rigidbody2D」と「Circle Collider 2D」コンポーネントをアタッチします。

③ボールに力を与えるスクリプトを作成し、ボールオブジェクトにアタッチします。実行してみましょう！

このままだとボールに通常の重力が作用して落下してしまいます。「GravityScale」の値を「０」にすると重力の働きをなくすことができます。
逆にこの値を大きくすると、動きに重さを出すことができます。

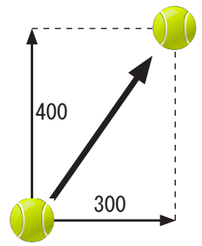


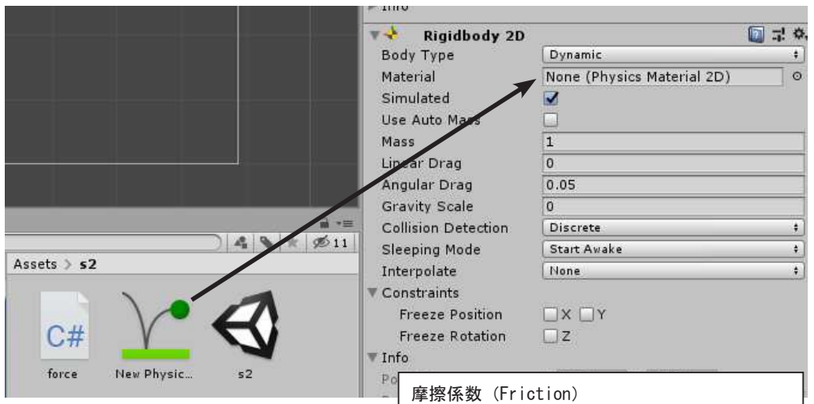

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class r17_ball : MonoBehaviour
{
    void Start()          // ゲームスタート時に実行
    {
        Rigidbody2D rb = GetComponent<Rigidbody2D>();     // ボールの Rigidbody を取得
        rb.AddForce(new Vector2(300, 400));               // ボールに力を与える
    }
}
```

スクリプト	r17_ball.cs
アタッチ先	ボール

力はＸＹ軸方向の別々の成分として合成されます。

Vector2(300,400)





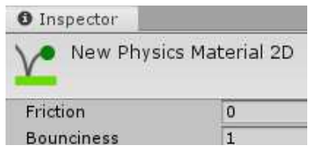
ボールと壁の衝突による摩擦も跳ね返りも損失がゼロの設定となってるので、ボールは永久に動き続けます。

摩擦係数（Friction）
→値を大きくすると摩擦力が強くなる

反発係数（Bounciness）
→値を大きくすると、跳ね返す力が強くなる

このままでは壁に当たっても跳ね返りません…

④「Phisics Material」を作成し、下図のように設定します。

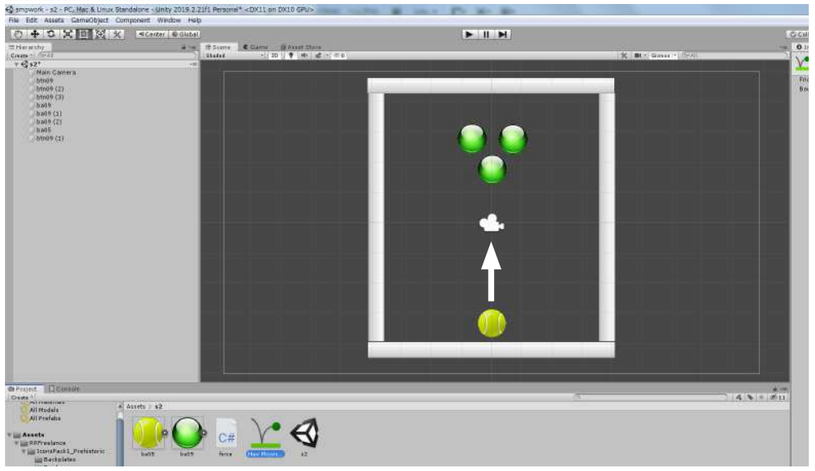


⑤ボールオブジェクトの「Rigidbody 2D」コンポーネントの「Material」にアタッチし、実行します。

演習 12

ビリヤードの玉の動きをシミュレーションするようなシーンを作成する。

シーン名　e12bill

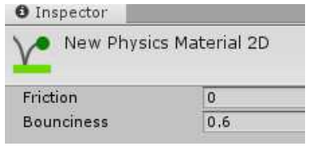


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class e12_force : MonoBehaviour
{
    public float power;  // 外部変数 power

    void Start()
    {
        Rigidbody2D rb = GetComponent<Rigidbody2D>();
        rb.                                  // Y方向に power を与える
    }
}
```

スクリプト	e12_force.cs
アタッチ先	ボール

①ブロックを上下左右に壁として配置し、それぞれに「Box Collider 2D」コンポーネントをアタッチします。

②「Physics Material」を作成し「Bounciness」を「0.6」に設定します。



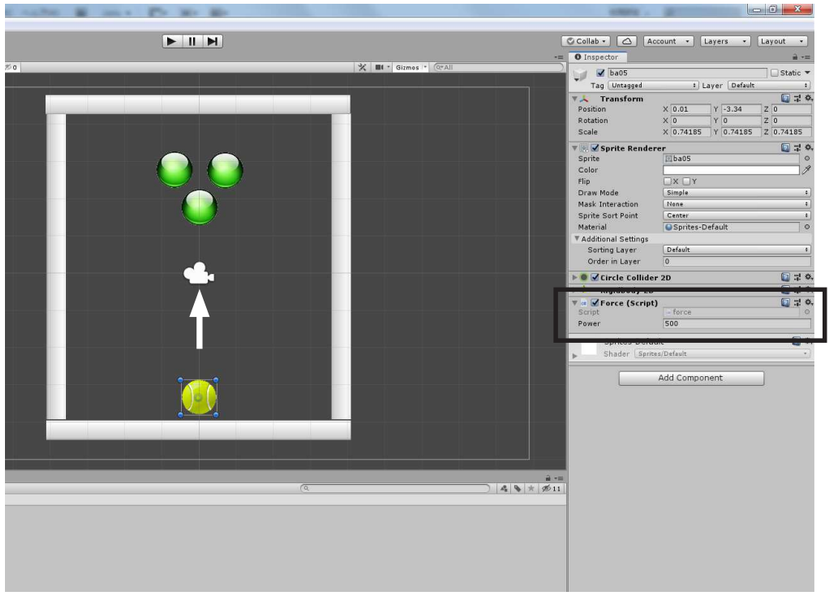
③ボールを１つ配置し「Rogidbody2D」と「Circle Collider2D」ををアタッチします。
「Rigidbody2d」の「GravityScale」を「０」「Material」に②で作成した「Physics Material」を設定します。

④ヒエラルキーでボールオブジェクトをコピー＆ペーストして例の様にボールを並べます。

⑤下方に配置したボールに力を与えるスクリプトをアタッチします。

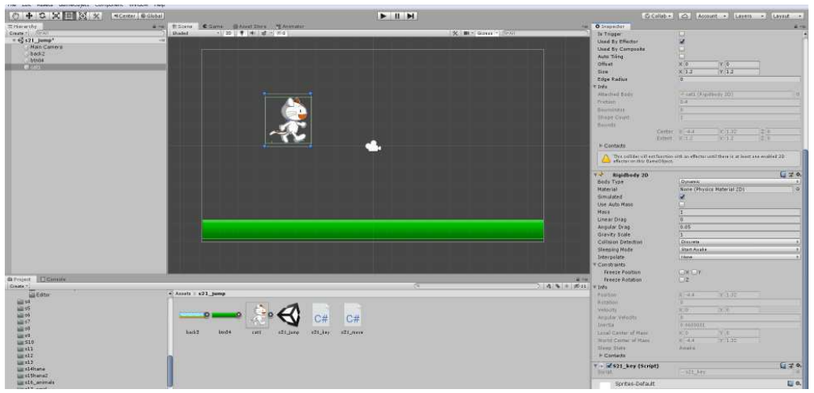
⑥インスペクターの「Script」で変数「power」の値を設定して実行してみます。

「power」や「Physics Material」の値を変更して何回か実行しよう！

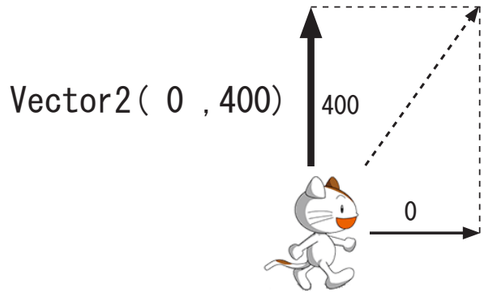


演習 13　猫のスプライトをキー入力でジャンプさせる

シーン名　e13cat



真上の場合は横方向（Ｘ）への力は「０」になります。



①ブロック画像を地面としてシーンの下方に配置し、「Box Collider 2D」コンポーネントをアタッチします。

②猫の画像をシーンに配置し、「BoxCollider2D」と「Rigisbody2D」コンポーネントをアタッチします。

③「PhysicsMaterial2D」を作成し「Bounciness」を「0.6」に設定、猫のオブジェクトの「Rigidbody2D」コンポーネントにアタッチします。

④実行して確認します。

⑤スペースキーを押したらジャンプする（上の向きに力を加える）スクリプトを作成して猫オブジェクトにアタッチします。

⑥実行を確認し、加える力や向きをを変更してみましょう！

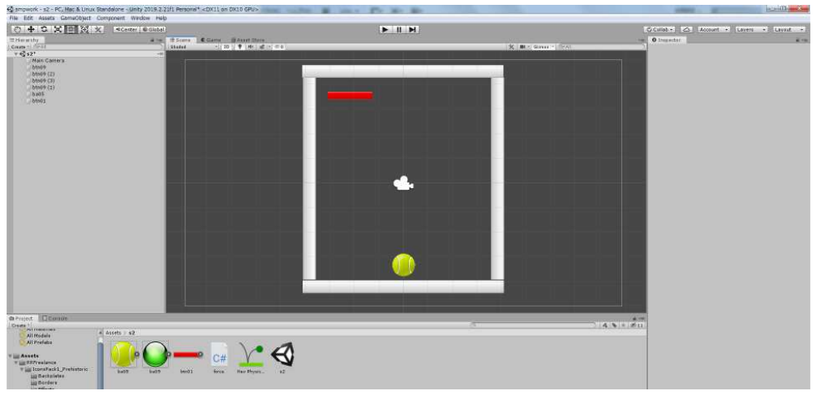
スクリプト	e13_key.cs
アタッチ先	猫

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

public class e13_key : MonoBehaviour
{
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))                    // スペースキーが押されたら
        {
            Rigidbody2D rb = [                              ]   //Rigidbody2D を取得
            [                                               ]   // 真上方向へ力を加える
        }
    }
}
```


例題 18　オブジェクトの衝突判定

シーン名　r18rock

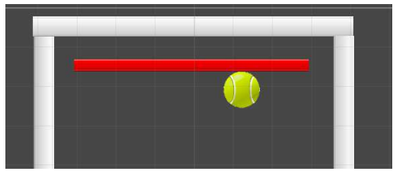
「Collider」「Rigidbody」を設定したオブジェクト同士は衝突の判定をする事ができます。



①「例題１７」に例の様にブロックを配置します。ブロックには「Box Collider」を設定します。

②ボールがブロックに当たったら消えるスクリプトを作成し、ブロックにアタッチします。

③ブロックのサイズを大きくして実行し、動作を確認（消える）します。



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

public class r18_ataru : MonoBehaviour
{
    void OnCollisionEnter2D(Collision2D obj)  // 衝突時に呼ばれる関数（衝突した相手オブジェクト）
    {
        Destroy(gameObject);    // 自分自身を消す
    }
}
```

スクリプト	r18_ataru.cs
アタッチ先	ブロック

オブジェクトの衝突には下記の様な種類があります。

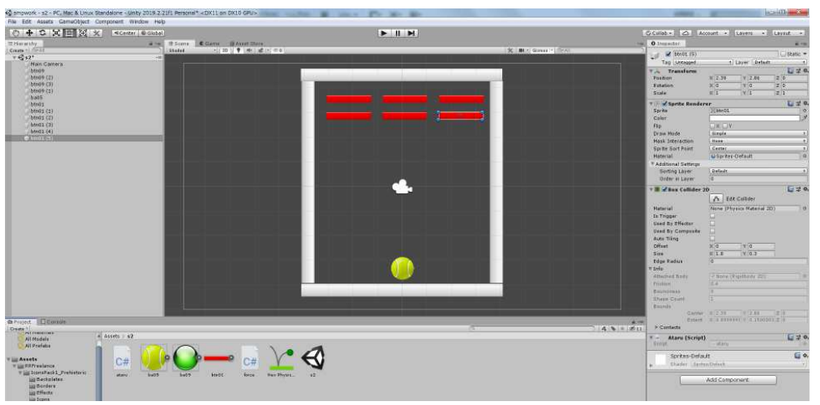
Trigger モード時はチェックを入れる

Collision系　跳ね返る

関数	呼び出されるタイミング
OnCollisionEnter	他のcollider/rigidbodyに触れたとき
OnCollisionExit	他のcollider/rigidbodyと触れ合うのをやめたとき
OnCollisionStay	他のrigidbody/colliderに触れている間

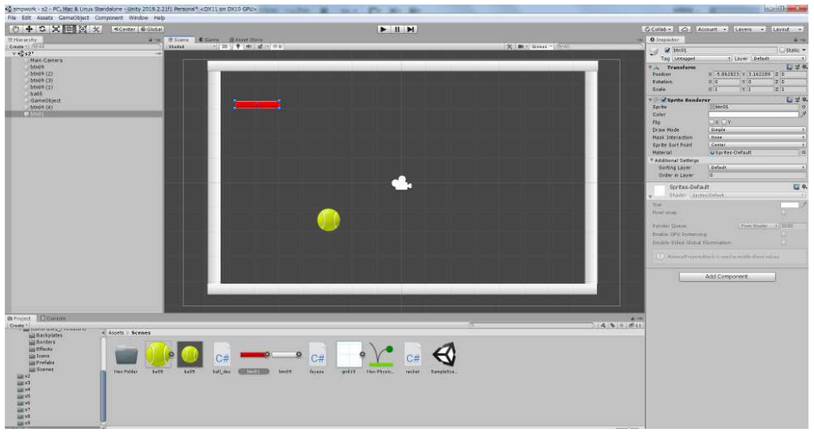
Trigger系　すり抜ける

関数	呼び出されるタイミング
OnTriggerEnter	Collider が他のトリガーイベントに侵入したとき
OnTriggerExit	Collider が他 のトリガーに触れるのをやめたとき
OnTriggerStay	Collider が他のトリガーに触れ続けている間



④ブロックサイズを小さくしてコピー、図のようにいくつか配置して実行してみます。

ミニゲーム３　ブロック崩しゲーム



シーン名	g03blck

①上下左右に壁を配置し、それぞれに「Box Collider 2D」コンポーネントを設定します。

②ボール画像を配置し「Circle Collider 2D」と「Rigidbody2D」コンポーネントをアタッチ、「GravityScale」の値を「０」します。

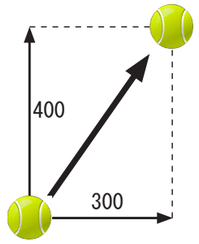
③ボールに力を与えるスクリプトを作成し、ボールオブジェクトにアタッチします。

④「Phisics Material」を作成し、「Furiction」を「０」、「Boundness」を「１」に設定します。

⑤１つだけ配置したブロックに「Box Colllider 2D」を設定し、プレハブ化します。

⑥ブロックを並べて複製するスクリプトを作成し、管理オブジェクトにアタッチします。

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

public class g03_force : MonoBehaviour
{
    void Start()    ゲームスタート時に実行
    {
        Rigidbody2D rb = GetComponent<Rigidbody2D>();
        rb.AddForce(new Vector2(300, 400)); // ボールに力を与える
    }
```

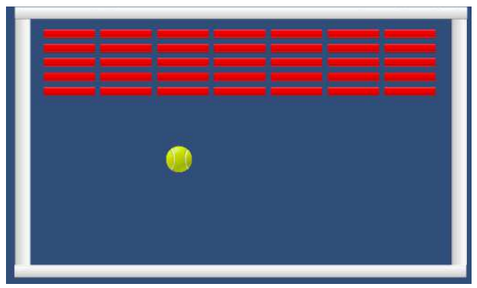


スクリプト	g03_force.cs
アタッチ先	ボール

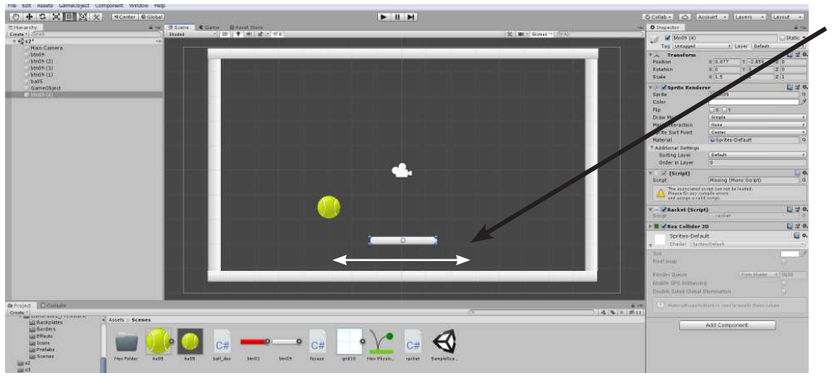
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class g03_block : MonoBehaviour
{
    public GameObject block_p;  // ブロックのプレハブ

    void Start()
    {
        float x,y;
        for(y=4.0f; y>=2.0f; y-=0.5f){                  // 縦（Ｙ座標）の繰り返し
            for(x=-6.0f; x<=6.0f; x+=2.0f){             // 横（Ｘ座標）の繰り返し
                GameObject bk = Instantiate(block_p);   // ブロックの複製
                bk.transform.position = new Vector3(x, y, 0);
            }
        }
    }
```

横方向 -6 ～ +6（７個並べる）

縦方向
+4 ～ +2
（5 個並べる）



スクリプト	g03_block.cs
アタッチ先	管理オブジェクト

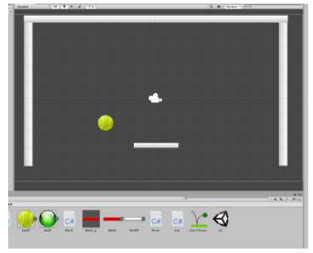


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class g03_key : MonoBehaviour
{
    void Update()
    {
        if (Input.GetKey(KeyCode.LeftArrow))
        {
            transform.Translate(-0.1f, 0, 0);
        }
        if (Input.GetKey(KeyCode.RightArrow))
        {
            transform.Translate(0.1f, 0, 0);
        }
    }
}
```

スクリプト	g03_key.cs
アタッチ先	ラケット

⑦ラケットに使用するブロックを下方に配置し「Box Collider 2D」を設定、キー入力で左右に移動するスクリプトを作成してアタッチします。
実行してみましょう！

※ゲームとして成立させるには、下側のブロックは消去するなどの設定も考えます。

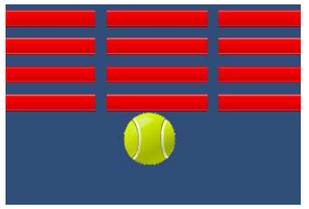


⑧ボールがブロックに衝突したら、ブロックを消去するスクリプトを作成し、ブロックのプレハブにアタッチします。

⑨ボールがブロックに衝突したときにエフェクトを表示させてみるのも面白いでしょう！

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

public class g03_ataru : MonoBehaviour
{
    public GameObject ex_p;     // エフェクトのプレハブ用外部変数

    void OnCollisionEnter2D(Collision2D obj)  // 衝突時に呼ばれる
    {
        Destroy(gameObject);    // 自分自身（ブロック）を消す
        GameObject ex = Instantiate(ex_p, transform.position, Quaternion.identity);
        Destroy(ex.gameObject, 1.0f);               // エフェクトを複製して、１秒後に消す
    }
}
```

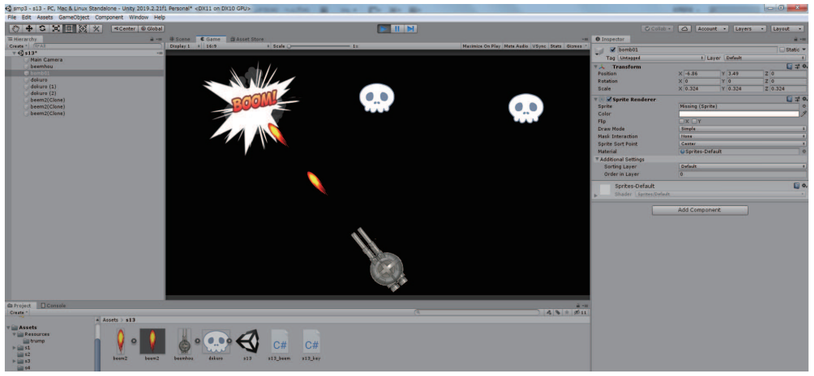
スクリプト	g03_ataru.cs
アタッチ先	ブロック



使用オブジェクトのまとめ

オブジェクト	オブジェクト名	追加コンポーネント	スクリプト	役　割
ボール		RigidBody2D　CircleCollider2D	g03_force.cs	ゲームスタート時にボールに力を与える
外壁		BoxCollider2dD	なし	上下左右に BoxCollider を設定した壁を設置
管理			g03_gen.cs	ブロックを縦横に繰り返して配置
ラケット		BoxCollider2D	g03_key.cs	キー入力でラケットを左右に移動
ブロック		BoxCollider2D	g03_ataru.cs	ボールがブロックに衝突した時の処理

例題 19 Collider による当たり判定



シーン名　r19coll

①砲台、ビーム、ドクロの画像をプロジェクトに追加します。

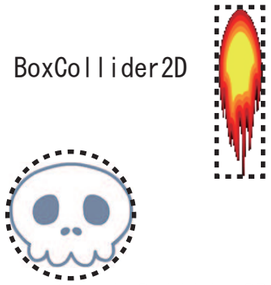
②砲台をシーンへ登録しキー入力（←→）で左右に回転するスクリプトをアタッチします。

※「Pivot」を設定すると画像の回転中心を変更できます。

③ビーム画像をシーンに配置し、BoxCollider2D を設定します。

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

public class r19_key : MonoBehaviour
{
    void Update()
    {
        if (Input.GetKey(KeyCode.LeftArrow))    //←キー入力で
        {
            transform.Rotate(0, 0, 1);          //  左へ１度回転
        }
        if (Input.GetKey(KeyCode.RightArrow)) //→キー入力で
        {
            transform.Rotate(0, 0, -1);         // 右へ１度回転
        }
    }
}
```

スクリプト	r19_key.cs
アタッチ先	砲台

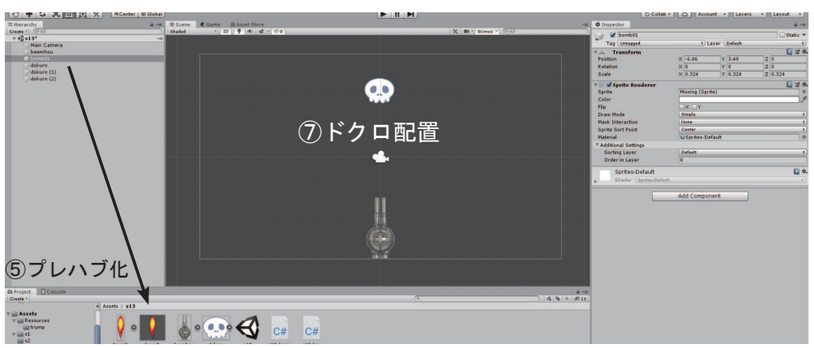



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class r19_beem : MonoBehaviour
{
    void Update()
    {
        transform.Translate(0, 0.2f, 0);
        if (transform.position.y > 5.0f)
            Destroy(gameObject);          //画面上外で消去
    }
}
```

スクリプト	r19_beem.cs
アタッチ先	ビーム

④ビームオブジェクトに上方（前方）に移動するスクリプトを作成しアタッチします。

⑤ビームオブジェクトの動きが確認できたら、プレハブ化し、元のオブジェクトは消去しておきます。



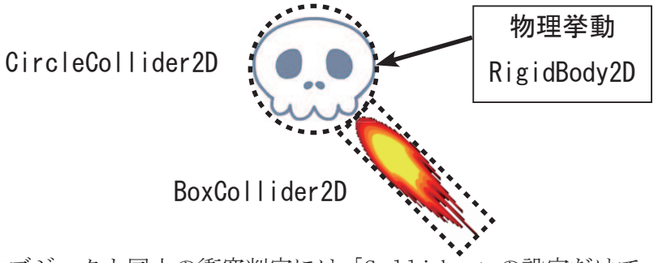
⑥ドクロの画像をシーンに配置し、CircleCollider2D を設定します。

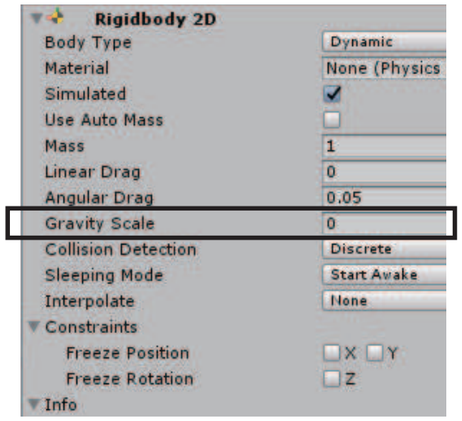

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class r19_key : MonoBehaviour
{
    public GameObject beem_p;   // プレハブを登録する外部変数

    void Update()
    {
        if (Input.GetKey(KeyCode.LeftArrow))
        {
            transform.Rotate(0, 0, 1);
        }
        if (Input.GetKey(KeyCode.RightArrow))
        {
            transform.Rotate(0, 0, -1);
        }
        if (Input.GetKeyDown(KeyCode.Space))        // スペースキーが押されたら
        {
            GameObject be = Instantiate(beem_p, transform.position, transform.rotation);
        }
    }                                               // 砲台の 位置 と 角度 でビームを複製する
```

スクリプト	r19_key.cs
アタッチ先	砲台

⑦スペースキーでビームを複製して発射するようにスクリプトを追加します。

⑧オブジェクトの外部変数にビームのプレハブを登録し、実行を確認します。





オブジェクト同士の衝突判定には「Collider」の設定だけでなく、どちらかのオブジェクトに物理挙動「RigidBody2D」コンポーネントの設定が必要になります。
今回はドクロオブジェクトに「RigidBody2D」を設定します。
「Gravity Scale」を「０」に設定し重力の影響を無しとします。

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

public class r19_dokuro : MonoBehaviour
{
    public GameObject ex_p;  // 爆発エフェクトのための外部変数

    void OnCollisionEnter2D(Collision2D obj) // オブジェクトの衝突判定
    {                                        // 爆発エフェクトの複製
        GameObject ex = Instantiate(ex_p, transform.position, Quaternion.identity);
        Destroy(gameObject);                 // ドクロ（自分自身）を消す
        Destroy(obj.gameObject);             // ビーム（衝突相手）を消す
        Destroy(ex.gameObject, 1.0f);        // 爆発エフェクトを１秒後に消す
    }
}
```

スクリプト	r19_dokuro.cs
アタッチ先	ドクロ

⑨衝突時に爆発のエフェクトが出現するようにスクリプトを作成します。

⑩爆発エフェクトのプレハブをスクリプトの外部変数に登録します。

⑪ドクロのオブジェクトをコピーして配置します。

ミニゲーム４　シューティングゲーム

シーン名	g04shot

今までの学習を組み合わせて簡単なシューティングゲームを、次のような手順で作ってみます。

①キー入力で宇宙船を移動
②ビームを発射する
③隕石の落下させる
④ビームと隕石の当り判定
⑤爆発エフェクトの追加
⑥効果音、背景など

「演習７」を参考にします。
背景は黒にしておきます。

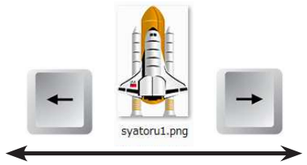
①キー入力で宇宙船を動かす

スクリプト	g04_key.cs
アタッチ先	宇宙船

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

public class g04_key : MonoBehaviour
{
    void Update()
    {
        if (Input.GetKey(KeyCode.LeftArrow)){    //「←」キー
            transform.Translate(-0.1f, 0, 0);    // 左へ移動
        }
        if (Input.GetKey(KeyCode.RightArrow)){   //「→」キー
            transform.Translate(0.1f, 0, 0);     // 右へ移動
        }
    }
}
```

①ロケット画像「syatoru1.png」を配置し、スケールを縦横２倍にする。

②ロケットがキー入力で左右に移動するスクリプトを作成しアタッチする。



「演習７」を参考にします。

②-1 ビームを移動する

スクリプト	g04_beem.cs
アタッチ先	ビーム

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

public class g04_beem : MonoBehaviour
{
    void Update()
    {
        transform.Translate( 0, 0.2f, 0);  // ビームを上方に移動
        if(transform.position.y>5.0f)
            Destroy(gameObject);           // 画面上まできたら削除
    }
}
```

③ビーム画像「beem2.png」を宇宙船の先端あたりに配置する。

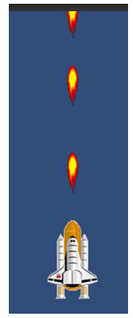
④ビームのオブジェクトが上方へ移動するスクリプトを作成、アタッチして実行を確認する。

⑤ビームオブジェクトをプレハブ化し名前を「beem_p」と変更する。元のオブジェクトは消去しておく。

②-2 キー入力でビームを発射（連射）する

スクリプト	g04_key.cs
アタッチ先	宇宙船

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

public class g04_key : MonoBehaviour
{
    public GameObject beem_p; // ビームプレハブのの外部変数
    void Update()
    {
        if (Input.GetKey(KeyCode.LeftArrow)) {
            transform.Translate(-0.1f, 0, 0);
        }
        if (Input.GetKey(KeyCode.RightArrow)){
            transform.Translate(0.1f, 0, 0);
        }
        if (Input.GetKeyDown (KeyCode.Space)) {
            Instantiate (beem_p, transform.position, Quaternion.identity);
        }                                          // 宇宙船の位置にビームを複製
    }
}
```



⑥「g04_key.cs」にスペースキーが押されたらビームを発射（複製）するスクリプト追加します。

⑦ビームの外部変数にプレハブをアタッチし、実行を確認します。

③隕石を落下（上から下へ移動）させる

スクリプト	g04_rock.cs
アタッチ先	隕石（岩石）

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

public class g04_rock : MonoBehaviour
{
    void Update()
    {
        transform.Translate(0, -0.02f, 0, Space.World);
        transform.Rotate(0, 0, 5.0f);
    }                                  // 回転させながら落下
}
```


⑧隕石（岩石）の画像を用意します。必要に応じてスプライトエディタでスライスしておきます。

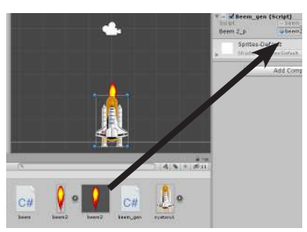
⑨隕石の画像をシーンに配置、落下するスクリプトを作成し、アタッチ、実行を確認します。

⑩隕石オブジェクトをプレハブ化「rock_p」し、元のオブジェクトは消去します。

⑪管理オブジェクトを作り、隕石を0.5秒ごとに複製するスクリプトを作成してアタッチします。

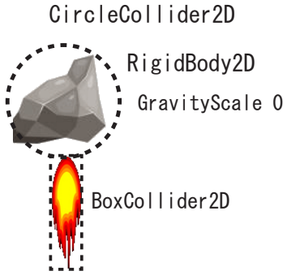
スクリプト	g04_gen.cs
アタッチ先	管理オブジェクト

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

public class g04_gen : MonoBehaviour
{
    public GameObject rock_p;   // プレハブ用外部変数

    void Start()
    {
        InvokeRepeating("rock_gen", 0.5f, 0.5f);  //0.5秒毎に実行
    }

    void rock_gen()
    {
        GameObject rc = Instantiate(rock_p);      // プレハブから複製
        int x = Random.Range(-7, 7);              // 横位置は乱数
        rc.transform.position = new Vector3(x, 5, 0);
    }
}
```



⑤ビームと隕石の当たり判定

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

public class g04_rock : MonoBehaviour
{
    public GameObject ex_p;

    void Update()
    {
        transform.Translate(0, -0.05f, 0, Space.World);
        transform.Rotate(0, 0, 5.0f);
    }

    void OnCollisionEnter2D(Collision2D obj)　衝突判定
    {//
        Destroy(gameObject);          //隕石オブジェクトを消去
        Destroy(obj.gameObject);      //ビームオブジェクトを消去
    }
}
```



スクリプト	g04_rock.cs
アタッチ先	隕石（岩石）

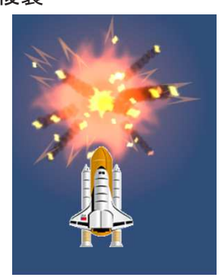
⑫ビームと隕石で当たり判定を行うためにビームのプレハブに「BoxCollider2D」、隕石のプレハブには「CircleCollider2D」と「RigidBody2D」コンポーネントをアタッチし、「RigidBody2D」の「GravityScale」を「0」に設定しておきます。

⑬隕石落下のスクリプトに当たり判定の処理を追加し、実行を確認します。

⑥爆発エフェクトの追加

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

public class g04_rock : MonoBehaviour
{
    public GameObject ex_p;     //プレハブ用の外部変数

    void Update()
    {
        transform.Translate(0, -0.05f, 0, Space.World);
        transform.Rotate(0, 0, 5.0f);
    }

    void OnCollisionEnter2D(Collision2D obj)
    {
        GameObject ex = Instantiate(ex_p, transform.position, Quaternion.identity);
        Destroy(gameObject);                  //隕石の位置に爆発エフェクトを複製
        Destroy(obj.gameObject);
        Destroy(ex.gameObject, 1.0f);         //爆発エフェクトを１秒後に消去
    }
}
```

スクリプト	g04_rock.cs
アタッチ先	隕石（岩石）



⑭衝突時に爆発のエフェクトを追加します。

アセットストアなどから爆発エフェクトを入手しインポートします。

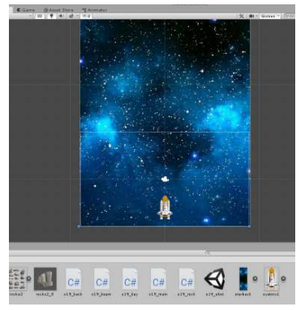
スクリプトを追加し、外部変数に、爆発エフェクトのプレハブを登録します。

⑦その他１　背景の表示

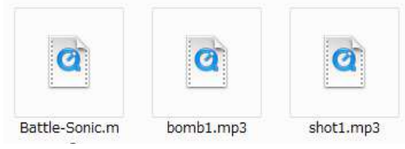
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class g04_back : MonoBehaviour
{
    void Update()
    {
        transform.Translate(0, -0.05f, 0);
        if (transform.position.y < -22.0f)
            transform.position = new Vector3(0, 22.0f, 0);
    }
}
```

スクリプト	g04_back.cs
アタッチ先	背景

⑮背景に星空の画像を配置し、背景が動くスクリプトを作成してアタッチします。



⑦その他２　ＢＧＭ・効果音をつける



ＢＧＭ　爆発音　ビーム音

⑯ビーム発射時に効果音を鳴らします。
宇宙船オブジェクトに「AudioSource」コンポーネントをアタッチし、「AudioClip」に効果音を設定します。
スクリプトを追加し実行を確認します。
他の効果音やＢＧＭも追加してみましょう！

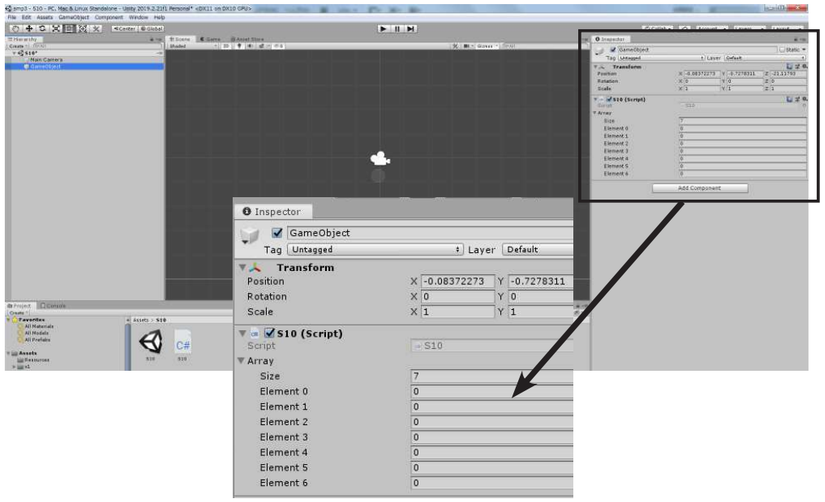
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class g04_key : MonoBehaviour
{
    public GameObject beem_p;
    AudioSource aud;

    void Start()
    {
        aud = GetComponent<AudioSource>();
    }

    void Update()
    {
        if (Input.GetKey(KeyCode.LeftArrow)) {
            transform.Translate(-0.1f, 0, 0);
        }
        if (Input.GetKey(KeyCode.RightArrow)){
            transform.Translate(0.1f, 0, 0);
        }
            if (Input.GetKeyDown (KeyCode.Space)) {
                aud.Play();
                Instantiate (beem_p, transform.position, Quaternion.identity);
            }
    }
}
```

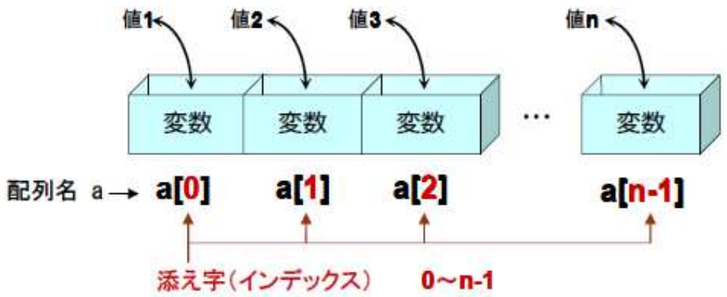
スクリプト	g04_key.cs
アタッチ先	宇宙船

オブジェクト	スクリプト	役　割
宇宙船	g04_key.cs	キー入力で宇宙船移動、ビームの発射（生成）を処理
ビーム	g04_beem.cs	下から上にビームを移動
隕石（岩石）	g04_rock.cs	上から下へ隕石を移動。ビームとの衝突判定を行い、爆発エフェクトも処理
管理	g04_gen.cs	指定時間ごとに隕石をランダムな位置に生成
背景	g04_back.cs	背景をスクロール

例題 20 配列

たくさんのデータを扱う配列を学びます。





シーン名　r20aray

サイコロを１０００回転がして１から６の目がそれぞれ何回出現したかをシミュレーションするプログラムを作ってみます。

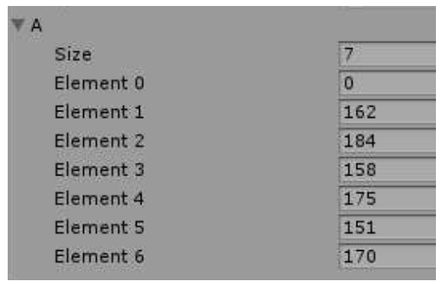
１～６の目をカウントする配列を用意し、全てを「0」に初期設定しておきます。

外部変数（public）にすることでインスペクターから値を確認することが出来ます。

①空のオブジェクト（管理オブジェクト）を作成しスクリプト「r20_aray.cs」をアタッチします。

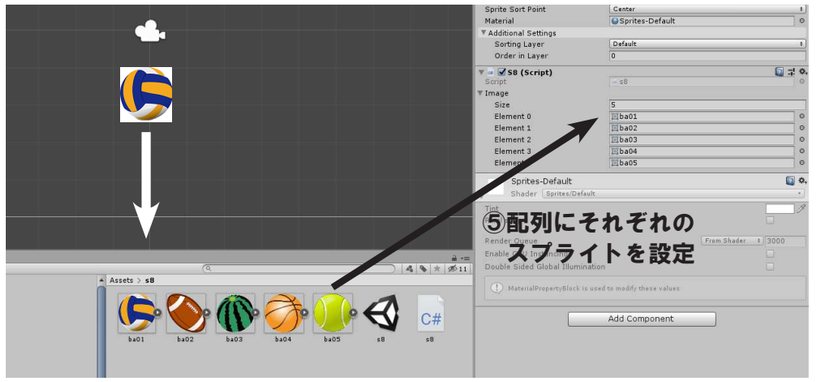
②実行するとインスペクターの外部変数欄にカウントされた数値が表示されます。

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

public class r20_aray : MonoBehaviour
{
    public int[] a = { 0,0,0,0,0,0,0 };
                  //a[0] ～ a[6] の配列　０を初期設定
    void Start()
    {
        for (int n = 0; n < 1000; n++)        // １０００回繰り返す
        {
            int r = Random.Range(1, 7);       // サイコロを振る（１～６の乱数）
            a[r] = a[r] + 1;                  // ｒ番の配列をカウントアップ
            //Debug.Log(r);                         // コンソールに乱数値を表示
        }
    }
}
```



スクリプト	r20_aray.cs
アタッチ先	管理オブジェクト

例題 21 スプライト型の配列



シーン名　r21aray

①異なるボールの画像を５つ用意しプロジェクトに追加します。

②１つのボールをシーンへ配置します。

③ボールが上から下へ移動し、画面下まで来たら画面上へ移動する、スクリプト記述しアタッチ、動作を確認します。

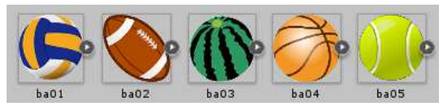
④例の様にスクリプトを追加します。ボールが画面上に移動するたびに異なったボールにランダムに切り替えます。

⑤外部変数として設定されたスプライト型の配列に５つのスプライト（画像）を設定します。

⑥実行して確認します。

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

public class r21_move : MonoBehaviour
{
    void Update()
    {
        transform.Translate( 0, -0.1f,0);
        if (transform.position.y < -5.0f)
        {
            transform.position = new Vector3(0, 5.0f, 0);
        }
    }
}
```

スクリプト	r21_move.cs
アタッチ先	ボール



img[0] img[1] img[2] img[3] img[4]

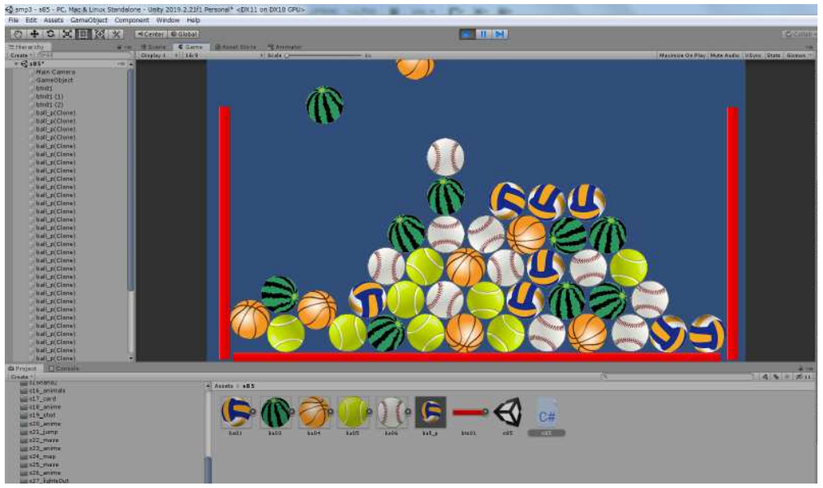
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class r21_move : MonoBehaviour
{
    public Sprite[] image = new Sprite[5];   // スプライト型の配列（外部変数）
    SpriteRenderer sr;                       //SpriteRenderer 型の変数

    void Start()
    {
        sr = gameObject.GetComponent<SpriteRenderer>();  //SpriteRenderer コンポーネントを取得
    }

    void Update()
    {
        transform.Translate( 0, -0.1f,0);
        if (transform.position.y < -5.0f)
        {
            transform.position = new Vector3(0, 5.0f, 0);
            int r = Random.Range(0, 5);                  // ０～４の乱数
            sr.sprite = image[r];                        // スプライトを配列の内容に変更
        }
    }
}
```

スクリプト	r21_move.cs
アタッチ先	ボール

例題 21 落ち物ゲーム風（配列使用）

発展①

シーン名　e10ball



演習１０をベースに例題２１を組み合わせて、種類の違うボールが連続落下するシーンを作成します。

①演習１０の動作を確認。

②例の様にスクリプトを追加します。

③外部変数として設定されたスプライト型の配列に５つのスプライト（画像）を設定します。

④実行を確認します。

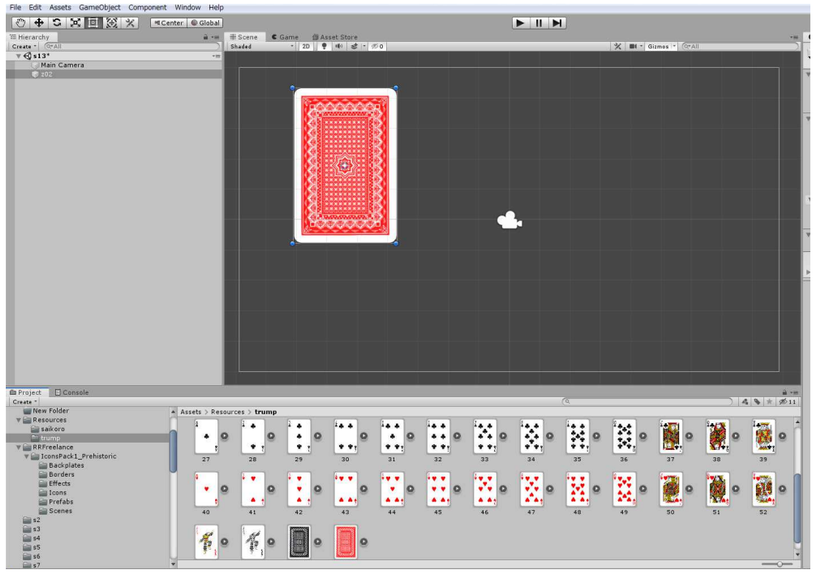
スクリプト	e10_fuyasu.cs
アタッチ先	管理オブジェクト

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

public class s85 : MonoBehaviour
{
    public GameObject ball_p;                //ボールのプレハブ変数
    public Sprite[] image = new Sprite[5];   //スプライト型の配列（外部変数）
    SpriteRenderer sr;                       //SpriteRenderer 型の変数

    void Start()
    {
        InvokeRepeating("ball_fuyasu", 0.5f, 0.5f);    //0.5 秒ごとに ball_fuyasu を呼び出す
    }

    public void ball_fuyasu()
    {
        int x,y,r;
        GameObject ba = Instantiate(ball_p);    //プレハブ ball_p をインスタンス化（複製）
        x = Random.Range(-5, 5);                //-5 ～ 5 の乱数
        r = Random.Range(0, 5);                 //0 ～ 4 の乱数
        sr = ba.GetComponent<SpriteRenderer>();  //SpriteRenderer コンポーネントを取得
        sr.sprite = image[r];                   //スプライトを配列の内容に変更
        ba.transform.position = new Vector3(x, 5, 0);  //ボールを上のランダム位置に移動
    }
}
```

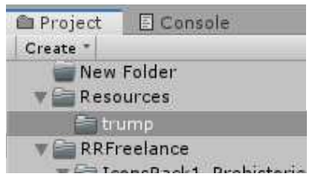

例題 22 多数の画像（スプライト）を配列に読み込む

シーン名　r22trump

たくさんのスプライト（画像）を配列にまとめて扱う手法を学びます。



①プロジェクトウインドの「Assets/Resources」フォルダにトランプ画像 50 数枚を収めた「trump」フォルダをコピーします。

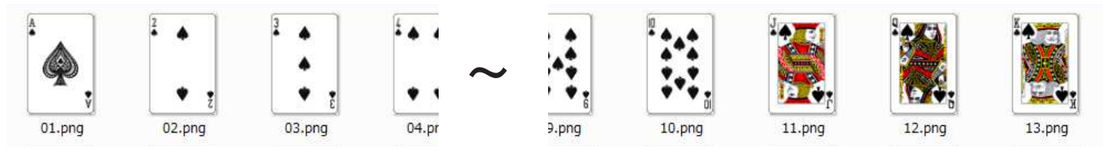


②トランプ裏面の画像を配置し、スケールを調整します。

③スクリプトを作成しトランプにオブジェクトにアタッチします。

④実行を確認します。

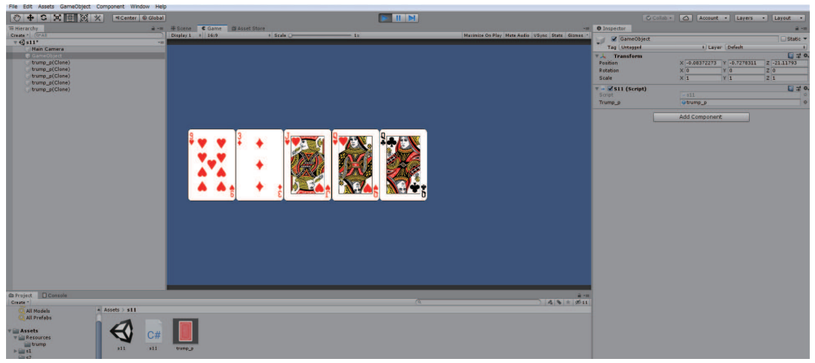
読み込まれた配列 image の内容



スクリプト	r22_trump.cs
アタッチ先	トランプ

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

public class r22_trump : MonoBehaviour
{
    SpriteRenderer sr;    //SpriteRenderer 型の変数を用意
    Sprite[] image;       //Sprite 型の配列 image を用意
    int n=0;              //配列の番号を示す変数

    void Start()
    {                                                    //Assets/Resources/trump フォルダに
        image = Resources.LoadAll<Sprite>("trump");      // ある全ての画像を配列に読み込む
        sr = gameObject.GetComponent<SpriteRenderer>();  //SpriteRenderer コンポーネントを取得
    }

    void Update()
    {
        if (Input.GetMouseButtonDown(0))     //もしマウスがクリックされたら
        {
            sr.sprite = image[n];            //スプライトにｎ番目の配列の内容を入れる
            n++;                             //配列番号をプラス１
        }
    }
}
```

演習 14　トランプ５枚をランダムに表示する



読み込まれた配列 image の内容

image[0]　image[1]　image[2]　・・・　image[11]　image[12]

シーン名　e14trump

①プロジェクトウインドウの「Assets/Resources」フォルダにトランプ画像 50 数枚を収めた「trump」フォルダをコピーします。

②トランプ裏面の画像を配置し、プレハブ化します。

③トランプ５枚分のオブジェクトを配列として用意し、読み込んだリソースからランダムでスプライトを設定します。

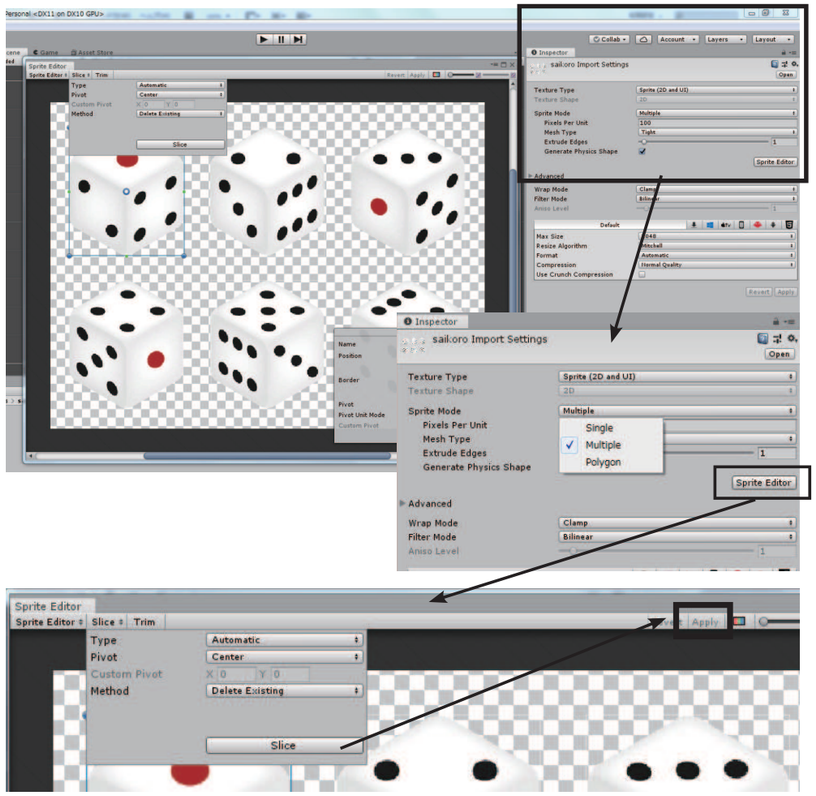
④実行を確認します。

スクリプト	e14_trump.cs
アタッチ先	トランプ

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

public class e14_trump : MonoBehaviour
{
    public GameObject trump_p;              // トランプのプレハブ用外部変数
    GameObject[] tp = new GameObject[5];    // トランプの５枚用のオブジェクト配列

    SpriteRenderer sr;                      //SpriteRenderer 型の変数
    Sprite[] image;                         // トランプ全ての画像を読み込む配列

    void Start()
    {
        for (int n = 0; n < 5; n++)         // ５枚分のトランプオブジェクトを作成
        {
            tp[n] = Instantiate(trump_p);
            tp[n].transform.position = new Vector3(n * 2.0f-7.0f, 0, 0);
        }
        image = Resources.LoadAll<Sprite>("trump");     // トランプ画像を Resources から読み込み
    }

    void Update()
    {
        if (Input.GetMouseButtonDown(0))    // もしマウスがクリックされたら
        {
            for[                        ]   // トランプ５枚分の繰返し
            {
                sr = tp[n].GetComponent<SpriteRenderer>();  // トランプ配列の SpriteRenderer 取得
                int r = [                   ]               // トランプ画像をランダムで設定
                sr.sprite = image[r];
            }
        }
    }
}
```

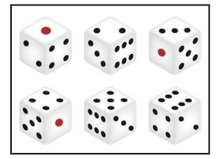
例題 23　アトラス画像を切り分け配列に読み込む

ひとつにまとめられた画像を切り分け、それぞれのスプライトとして配列で使用します。



シーン名　r23sai

①プロジェクトウインドウの「Assets/Resources」フォルダにサイコロの画像がまとめられた「saikoro.png」コピーします。



saikoro.png

②プロジェクトウインドウの「Assets/Resources」フォルダのサイコロの画像を選択、インスペクターの「Sprite Mode」を「Multiple」に設定し、「Sprite Editor」ボタンをクリックします。

③「Sprite Editor」の「Slice」メニューを選択し、下部の「Slice」ボタンをクリックします。

④右上の「Apply」ボタンをクリックし、スライスの状態を確認します。

スクリプト	r23_sai.cs
アタッチ先	サイコロ

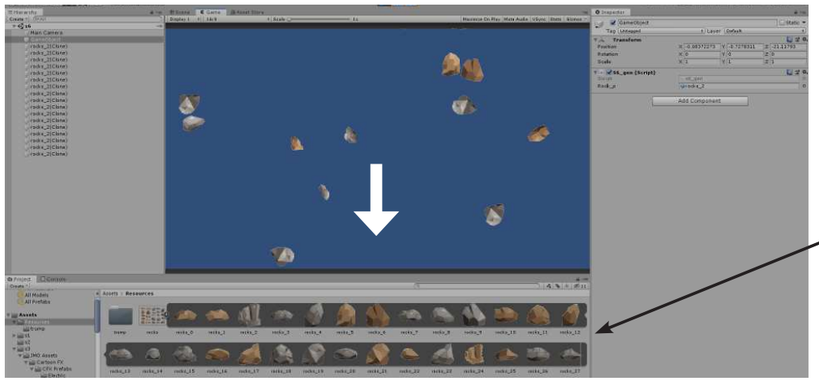
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class r23_sai : MonoBehaviour
{
    SpriteRenderer sr;    //SpriteRenderer 型の変数を用意
    Sprite[] image;       //Sprite 型の配列 image を用意

    void Start()
    {                                                   //Assets/Resources/trump フォルダに
        image = Resources.LoadAll<Sprite>("saikoro");     // ある全ての画像を配列に読み込む
        sr = gameObject.GetComponent<SpriteRenderer>();  //SpriteRenderer コンポーネントを取得
    }
    void Update()
    {
        if (Input.GetMouseButton(0))                  // マウスボタンが押されている間…
        {
            int r = Random.Range(0, image.Length);        // ０～５の乱数を発生
            sr.sprite = image[r];                         // 該当するサイコロ画像をスプライトに設定
        }
    }
}
```

⑤スライスした画像の一つをシーンに配置します。

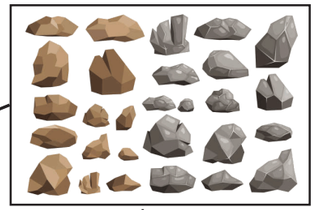
⑥スクリプトを作成しアタッチ、実行します。

例題 24　アトラス画像を配列に読み込む２

シーン名　r24atras



①岩の画像「rocks.png」をスプライトエディタで切り分け、ランダムに落下させるシーンを作成します。



rocks.png

②岩の画像ひとつをシーンに配置し、落下させるスクリプトを作成しアタッチ、動作を確認してプレハブ化します。

③切り分けられたスプライトを配列にまとめて読み込み、プレハブを複製してランダムに落下させるスクリプトを作成します。

④空のオブジェクトを作成しアタッチ、オブジェクトの外部変数にプレハブを設定します。

⑤実行を確認します。

スクリプト	r24_move.cs
アタッチ先	岩石

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

public class r24_move : MonoBehaviour
{
    float speed, rot;   // 落下スピードと回転

    void Start()
    {
        speed = Random.Range(-0.1f, -0.02f);        // 岩の落下スピード
        rot = Random.Range(-5.0f, 5.0F);            // 岩の回転方向
    }

    void Update()
    {
        transform.Translate(0, speed , 0,Space.World);  // 落下
        if (transform.position.y < -5)
            Destroy(gameObject);                    // 画面下まで落下したら消去
        transform.Rotate(0, 0, rot);                // 回転
    }
}
```

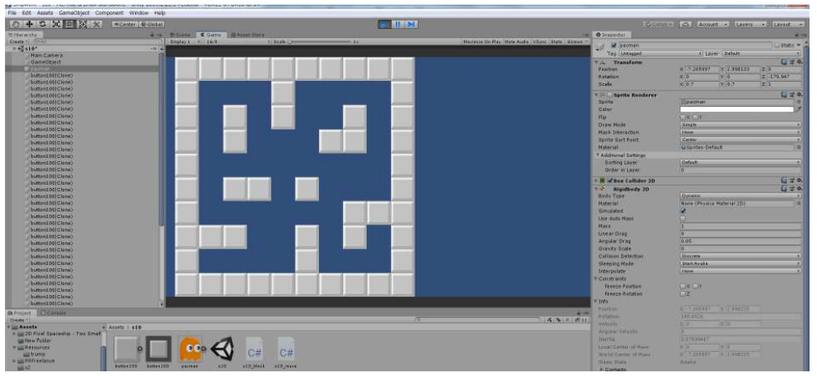

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class r24_gen : MonoBehaviour
{
    public GameObject rock_p;               // 岩オブジェクトのプレハブ
    SpriteRenderer sr;
    Sprite[] image;                         // 切り分けられたスプライトを入れる配列

    void Start()
    {
        image = Resources.LoadAll<Sprite>("rocks");    //Resources にある全てのスプライトを読み込む
        InvokeRepeating("rock_gen", 0.5f, 0.2f);
    }

    void rock_gen()
    {
        GameObject rk = Instantiate(rock_p);                    // プレハブからスプライトを複製
        float x = Random.Range(-11.0f, 11.0f);                  // 横位置は乱数
        rk.transform.position = new Vector3(x, 6, 0);
        int r = Random.Range(0, image.Length);                  // 岩の種類（配列番号）をランダムで設定
        sr = rk.gameObject.GetComponent<SpriteRenderer>();
        sr.sprite = image[r];
    }
}
```

スクリプト	r24_gen.cs
アタッチ先	管理オブジェクト

例題 25　２次元配列

２次元配列に設定されたデータから迷路を表示します。

シーン名　r25aray



①ボタン画像「button100.png」を配置する。

②オブジェクト「button100」をヒエラルキーウインドからプロジェクトウインドへドラッグ「プレハブ化」し、名前を「button100_p」と変更する。

③元のオブジェクト「button100」を削除する。

④迷路を表示するスクリプトを作成する。（r12_gen.cs）

⑤ヒエラルキーウインドで空のオブジェクト「GameObject」（管理オブジェクト）を作成し、「r12_gen.cs」をアタッチする。

⑥で作成した空のオブジェクトのコンポーネントの外部変数「button_p」にボタンのプレハブを登録する。

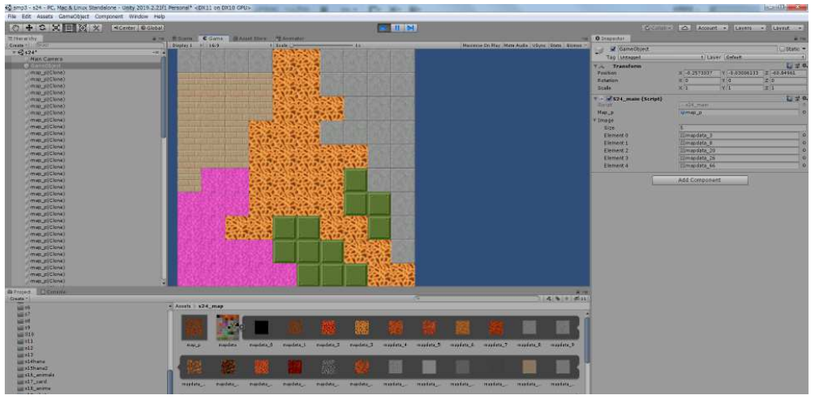
⑦実行を確認する。

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

public class r25_aray : MonoBehaviour
{
    public GameObject kabe_p;

    int[,] maze = {                 // ２次元配列の初期設定
        {1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
        {1, 0, 0, 0, 1, 0, 0, 0, 0, 1},
        {1, 0, 1, 0, 1, 0, 0, 1, 0, 1},
        {1, 0, 1, 0, 0, 0, 1, 1, 0, 1},
        {1, 0, 0, 0, 0, 0, 0, 0, 0, 1},
        {1, 0, 1, 1, 0, 1, 0, 0, 0, 1},
        {1, 0, 0, 0, 0, 0, 0, 1, 1, 1},
        {1, 1, 1, 0, 0, 1, 0, 1, 0, 1},
        {1, 0, 0, 0, 0, 1, 0, 0, 0, 1},
        {1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
    };                              // １が壁、０が通路

    void Start()
    {
        int x, y;
        for (y = 0; y < 10; y++)                // 縦方向の繰り返し
        {
            for (x = 0; x < 10; x++)            // 横方向の繰り返し
            {
                if (maze[y, x] == 1)            // もし配列の値が１なら
                {
                    GameObject kb = Instantiate(kabe_p);                    // プレハブをコピー
                    kb.transform.position = new Vector3(x-8, 4.5f-y, 0);    // 指定位置に配置
                }
            }
        }
    }
}
```

スクリプト	r25_aray.cs
アタッチ先	管理オブジェクト

例題 26 マップデータの表示

シーン名　r26aray



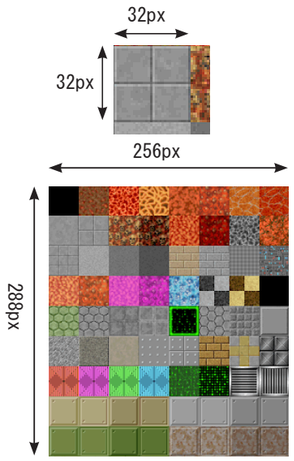
①マップデータがまとめられたアトラス画像をスライスします。(32px 32px)

②マップ１コマをプレハブ化します。32px を 100px に合わせるため Scale は「100/32」に設定します。

③管理オブジェクトを作成して、スクリプトをアタッチします。

スクリプト	r26_main.cs
アタッチ先	管理オブジェクト

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

public class r26_main : MonoBehaviour
{
    public GameObject map_p;
    Sprite[] image;
    SpriteRenderer sr;      //SpriteRenderer 型の変数

    int[,] map = {                          // ２次元配列の初期設定
        { 3,  3,  3,28,28,28,  3,  3,  3,  3},   // マップのパターン番号
        { 8,  8,  8,  8,28,28,  3,  3,  3,  3},
        { 8,  8,  8,  8,  8,28,  1,  3,  3,  3},
        {30,  8,  8,  8,  8,28,  1,  3,  3,  3},
        {30,30,  8,  8,28,28,  1,  1,  3,  3},
        {30,30,  8,  8,28,28,  1,  1,  3,  3},
        {30,30,  8,  8,28,28,  1,  1,  1,  3},
        {30,30,  8,  8,  8,28,  1,  1,  1,  1},
        {32,30,30,  8,  8,28,  1,  1,  1,  1},
        {32,32,32,32,32,28,28,28,28,  1},
    };

    void Start()
    {
        int x, y;

        image = Resources.LoadAll<Sprite>("mapdata");

        for (y = 0; y < 10; y++)          // 縦方向の繰り返し
        {
            for (x = 0; x < 10; x++)      // 横方向の繰り返し
            {
                GameObject mc = Instantiate(map_p);       // プレハブをコピー
                sr = mc.GetComponent<SpriteRenderer>();   //SpriteRenderer コンポーネントを取得
                sr.sprite = image[map[y, x]];             // マップのイメージをスプライトに設定
                mc.transform.position = new Vector3(x-8, 4.5f-y, 0);     // 指定位置に配置
            }
        }
    }

}
```



マップデータの配列番号

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	・	・	・			
			・	・	・	62	63
64	65	66	67	68	69	70	71

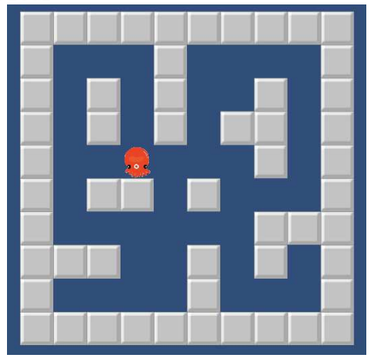
演習 15

カーソルキーの入力で、迷路の中をキャラクターが動かすシーンを作成してみましょう。

シーン名　e15maze

スクリプト	e15_main.cs
アタッチ先	管理オブジェクト

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

public class e15_main : MonoBehaviour
{
    public GameObject kabe_p;
    int[,] maze = {
        {1,1,1,1,1,1,1,1,1,1},
        {1,0,0,0,1,0,0,0,0,1},
        {1,0,1,0,1,0,0,1,0,1},
        {1,0,1,0,0,0,1,1,0,1},
        {1,0,0,0,0,0,0,0,0,1},
        {1,0,1,1,0,1,0,0,0,1},
        {1,0,0,0,0,0,0,1,1,1},
        {1,1,1,0,0,1,0,1,0,1},
        {1,0,0,0,0,1,0,0,0,1},
        {1,1,1,1,1,1,1,1,1,1},
    };

    void Start()
    {
        int x, y;
        for (y = 0; y < 10; y++){
            for (x = 0; x < 10; x++){
                if (maze[y, x] == 1){
                    GameObject kb = Instantiate(kabe_p);
                    kb.transform.position = new Vector3(x - 8, 5 - y, 0);
                }
            }
        }
    }
}
```



例題２５の迷路を表示するシーンをベースに作成していきます。

①迷路の壁となるブロックのプレハブに「BoxCollider2D」をアタッチします。

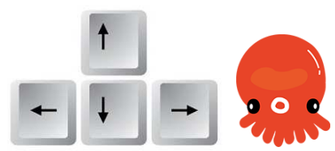
②キャラクタをシーンに配置し「BoxCollider2D」と「RigdBody2D」をアタッチします。大きさが壁内に収まるように、スケールと位置を調整します。

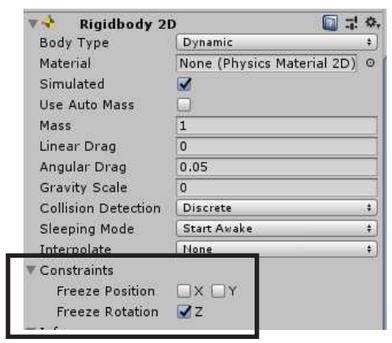
迷路を表示するスクリプトは例題２５と同じです。

③上下左右のキー入力でキャラクタを移動するスクリプトを記述して、アタッチします。

スクリプト	e15_key.cs
アタッチ先	キャラクタ

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

public class e15_key : MonoBehaviour
{
    void Update()
    {
        if (Input.GetKey(KeyCode.UpArrow))          ↑キーなら
        {
            transform.Translate(0, 0.1f, 0);
        }
        if (Input.GetKey(KeyCode.DownArrow))        ↓キーなら
        {
                                                    ]
        }
        if (Input.GetKey(KeyCode.LeftArrow))        ←キーなら
        {
            transform.Translate(-0.1f, 0, 0);
        }
        if                                          →キーなら
        {
            transform.Translate(0.1f, 0, 0);
        }
    }
}
```





RigidBody 2D コンポーネントの「Constraints」「Freze Rotation」の「Z」にチェックを入れるとキャラクタの回転を防ぐことができます。

演習 16 迷路の中をキャラクターが自動で動き回るシーンを作成してみましょう。

シーン名	e16maze

例題２５の迷路を表示するシーンをベースに作成していきます。

スクリプト	e16_main.cs
アタッチ先	管理オブジェクト

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

public class e16_main : MonoBehaviour
{
    public GameObject kabe_p;
    int[,] maze = {
        {1,1,1,1,1,1,1,1,1,1},
        {1,0,0,0,1,0,0,0,0,1},
        {1,0,1,0,1,0,0,1,0,1},
        {1,0,1,0,0,0,1,1,0,1},
        {1,0,0,0,0,0,0,0,0,1},
        {1,0,1,1,0,1,0,0,0,1},
        {1,0,0,0,0,0,0,1,1,1},
        {1,1,1,0,0,1,0,1,0,1},
        {1,0,0,0,0,1,0,0,0,1},
        {1,1,1,1,1,1,1,1,1,1},
    };

    void Start()
    {
        int x, y;
        for (y = 0; y < 10; y++){
            for (x = 0; x < 10; x++){
                if (maze[y, x] == 1){
                    GameObject kb = Instantiate(kabe_p);
                    kb.transform.position = new Vector3(x - 8, 5 - y, 0);
                }
            }
        }
    }
}
```

①迷路の壁となるブロックのプレハブに「BoxCollider2D」をアタッチします。

②キャラクタをシーンに配置し「BoxCollider2D」と「RigdBody2D」をアタッチします。「GravityScale」は０（ゼロ）に設定し、大きさが壁内に収まるように、スケールと位置を調整します。

BoxCollider2D
RigidBody2D

迷路を表示するスクリプトは例題２５と同じです。

③キャラクタを自身から見て、右方向に移動するスクリプトを作成してアタッチします。

スクリプト	e16_move.cs
アタッチ先	キャラクタ

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

public class e16_move : MonoBehaviour
{
    void Update()
    {
        ＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿  // 右方向へ 0.1f 移動
    }

    void OnCollisionEnter2D(Collision2D col)  // 壁と衝突したら
    {
        transform.Translate(-0.2f, 0, 0);     // 少し戻す
        int r = ＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿  // １～３の乱数発生
        transform.Rotate(0, 0, r * 90);       // 乱数分だけ回転
    }
}
```

④壁に衝突したら
・少し左に移動（戻す）
・１～３の乱数を発生
・乱数値× 90 度回転
する処理を加えます。

９０度回転

１フレームで移動する値などを変更して実行してみましょう。

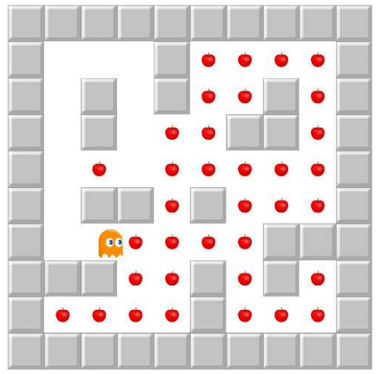
例題 27 迷路の中に置かれたりんごを消していく（ドットイート）シーンを作成します。

シーン名	r27maze

衝突したオブジェクトの種類によって処理を変える方法を学習します。演習１５をベースに作成します。

スクリプト	r27_main.cs
アタッチ先	管理オブジェクト

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

public class r27_main : MonoBehaviour
{
    public GameObject kabe_p;
    public GameObject apple_p;

    int[,] maze = {
        {1,1,1,1,1,1,1,1,1,1},
        {1,0,0,0,1,0,0,0,0,1},
        {1,0,1,0,1,0,0,1,0,1},
        {1,0,1,0,0,0,1,1,0,1},
        {1,0,0,0,0,0,0,0,0,1},
        {1,0,1,1,0,1,0,0,0,1},
        {1,0,0,0,0,0,0,1,1,1},
        {1,1,1,0,0,1,0,1,0,1},
        {1,0,0,0,0,1,0,0,0,1},
        {1,1,1,1,1,1,1,1,1,1},
    };

    void Start()
    {
        int x, y;
        for (y = 0; y < 10; y++){
            for (x = 0; x < 10; x++){
                switch (maze[y, x]){
                    case 0: GameObject ap = Instantiate(apple_p);     //「０」ならりんごを置く
                            ap.transform.position = new Vector3(x - 8, 5 - y, 0);
                            break;
                    case 1: GameObject kb = Instantiate(kabe_p);      //「１」なら壁を置く
                            kb.transform.position = new Vector3(x - 8, 5 - y, 0);
                            break;
                }}}
    }
}
```

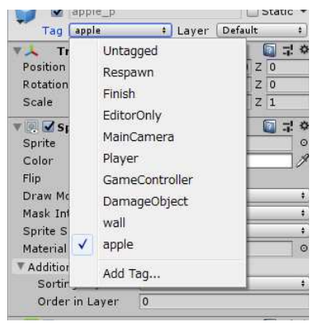


①りんごの画像をシーンに配置し「BoxCollider2D」をアタッチし、プレハブ化します。

②迷路データが「０」ならりんご、「１」なら壁を表示するようスクリプトを変更します。

スクリプト	r27_move.cs
アタッチ先	キャラクタ

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

public class r27_move : MonoBehaviour
{
    void Update()
    {
        transform.Translate(0.1f, 0, 0);
    }

    void OnCollisionEnter2D(Collision2D col)
    {
        if (col.gameObject.tag == "apple"){    //タグが「apple」なら
            Destroy(col.gameObject);           //オブジェクトを消す
        }
        else{                                  //そうでなければ（壁ならば）
            transform.Translate(-0.1f, 0, 0);  //方向を変える
            int r = Random.Range(1, 4);
            transform.Rotate(0, 0, r * 90);
        }
    }
}
```

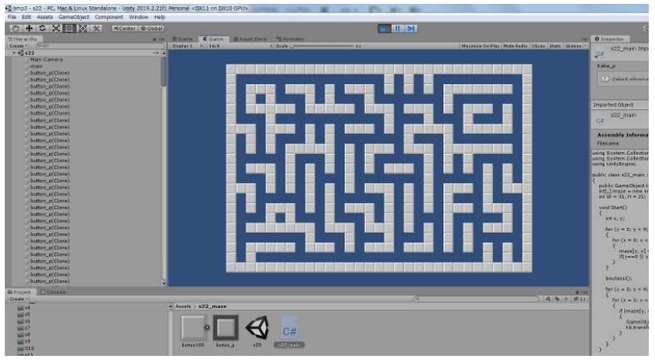


③オブジェクトを識別するためには「タグ」使います。「Add Tag」からタグに「apple」を追加し、設定します。

④衝突時のスクリプトでタグを判定して処理します。

例題 28 迷路の自動生成（棒倒し法）

シーン名　r28maze



迷路データは２次元配列とし「０」が道、「１」を壁として考えます。５×５の配列の場合には下図のようになります。

壁	壁	壁	壁	壁
壁	道	道	道	壁
壁	道	壁	道	壁
壁	道	壁	道	道
壁	道	壁	壁	壁

→

1	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	0	1	0	0
1	0	1	1	1

まずはこの配列を用意、外壁を設定し、これを表示させてみます。

スクリプト	r28_main.cs
アタッチ先	管理オブジェクト

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

public class r28_main : MonoBehaviour
{
    public GameObject kabe_p;           // ブロック用プレハブ変数
    int[,] maze = new int[100,100];     // 迷路用の２次元配列
    int W = 31, H = 21;                 // 迷路の横(W)縦(H)サイズ（奇数）

    void Start()
    {
        int x, y;

        for (y = 0; y < H; y++)         // 縦方向の繰返し
        {
            for (x = 0; x < W; x++)     // 横方向の繰返し
            {
                maze[y, x] = 0;         // 配列の内容を「０」に設定
                if(y==0 || y==H-1 || x==0 || x==W-1)
                    maze[y,x]=1;        // もし外側なら「１」に設定
            }
        }
        for (y = 0; y < H; y++)         // 配列の内容を迷路として表示
        {
            for (x = 0; x < W; x++)
            {
                if (maze[y, x] == 1)    // もし配列の内容が「１」なら
                {
                    GameObject kb = Instantiate(kabe_p);
                    kb.transform.position = new Vector3(x-15, 10-y, 0);
                }                       // ブロックを複製し配置
            }
        }
    }
}
```

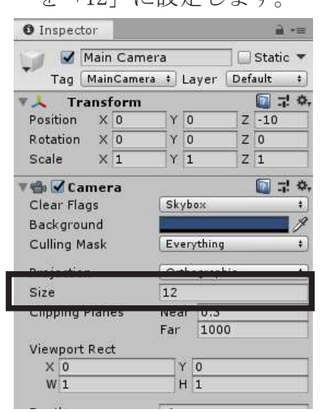
①ボタン画像「button100.png」を配置する。

②オブジェクト「button100」をヒエラルキーウインドからプロジェクトウインドへドラッグ「プレハブ化」し、名前を「button100_p」と変更する。
元のオブジェクト「button100」は削除する。

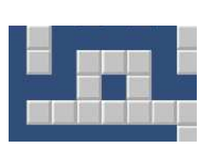
③空のオブジェクト（管理オブジェクト）を作成し、名前を「main」とします。

④迷路の大きさを横３１、縦２１とし、の外壁のみを表示するスクリプトを作成し、管理オブジェクトにアタッチします。

⑤通常より広い範囲を画面の範囲とするよう、カメラオブジェクト「MainCamera」の「Size」を「12」に設定します。

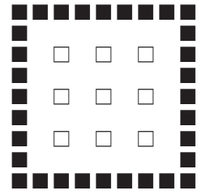



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class r28_main : MonoBehaviour
{
    public GameObject kabe_p;
    int[,] maze = new int[100,100];
    int W = 31, H = 21;

    void Start()
    {
        int x, y;

        for (y = 0; y < H; y++)
        {
            for (x = 0; x < W; x++)
            {
                maze[y, x] = 0;
                if(y==0 || y==H-1 || x==0 ||
            }
        }

        boutaosi();   // 迷路生成関数呼び出し

        for (y = 0; y < H; y++)
        {
            for (x = 0; x < W; x++)
            {
                if (maze[y, x] == 1)
                {
                    GameObject kb = Instantia
                    kb.transform.position = n
                }
            }
        }
    }

    void boutaosi()      // 棒倒し法による迷路生成
    {
        int x, y,r;
        Random.InitState(System.DateTime.Now.Millisecond);      // 乱数の種の初期化
        for (y = 2; y <= H-3; y += 2)                           // 縦方向の繰り返し(棒の場所)
        {
            for (x = 2; x <= W-3; x += 2)                       // 横方向の繰り返し(棒の場所)
            {
                maze[y,x] = 1;                                  // 棒の場所は「１」
                r = Random.Range(0,4);                          //0 ～ 3 の乱数
                switch (r)                                      // 乱数の値で分岐(switch ～ case 文)
                {
                    case 0: maze[y - 1,x    ] = 1; break;       // もし 0 なら 上に倒す
                    case 1: maze[y + 1,x    ] = 1; break;       //     1      下 〃
                    case 2: maze[y    ,x - 1] = 1; break;       //     2      左 〃
                    case 3: maze[y    ,x + 1] = 1; break;       //     3      右 〃
                }
            }
        }
    }
}
```

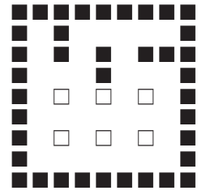


←囲まれた場所を作らないためには工夫が必要

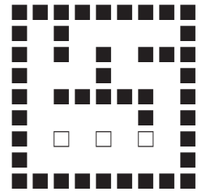
スクリプト	r28_main.cs
アタッチ先	管理オブジェクト



棒倒し法とは、迷路の盤上の基点から上下左右のどこかに壁を伸ばす事により、迷路を生成するアルゴリズムです。
基点から棒を倒していくような動作をするので、棒倒し法と呼ばれます。



棒倒し法では次のように迷路を生成します。



１．まず、外周の壁と１つ飛ばしに棒を倒す対象となる基点の壁があります。
（「□」が基点の壁になります）

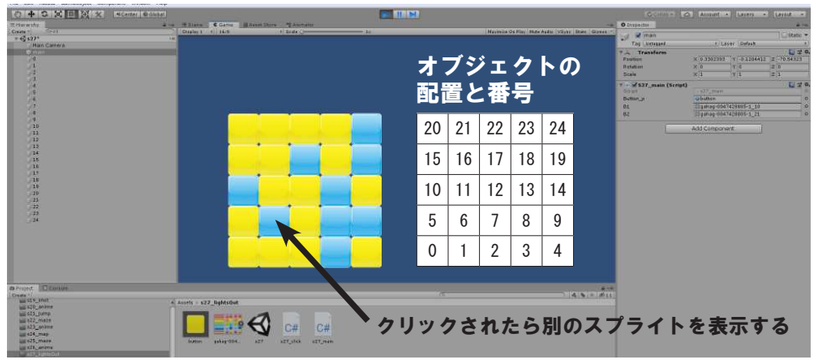
２．左上の基点から右の基点に向けて上下左右にランダムに棒（壁）を倒します。

３．続けて、２行目以降も棒を倒しますが、２行目以降については上方向以外にランダムに棒を倒します。
（発展課題の部分）

４．３行目も手順３と同様に棒を倒します。

５．一番下の行まで棒を倒し終わったら迷路の完成です。

例題 29 オブジェクト配列のクリック取得



スクリプト	r29_main.cs
アタッチ先	管理オブジェクト

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

public class r29_main : MonoBehaviour
{
    public GameObject button_p;                 //ボタンのプレハブ
    GameObject[] bt = new GameObject[25];   //オブジェクト型の配列
    public Sprite b1,b2;     //スプライトを切り替えるための外部変数

    void Start()
    {
        int n;
        for( n=0; n<25; n++ ){     //0 - 24 まで生成
            bt[n] = Instantiate(button_p);
            bt[n].transform.position = new Vector3(n%5, n/5, 0);
            bt[n].name = n.ToString();          //横５×縦５に配置
        }           //番号をオブジェクト名に設定
    }

    public void bt_click(int n)   //オブジェクト番号を受け取る
    {
        SpriteRenderer sr = bt[n].GetComponent<SpriteRenderer>();
        sr.sprite = b2;                 //スプライト（画像）を変更
    }
}
```

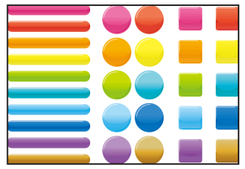
スクリプト	r29_click.cs
アタッチ先	ボタン プレハブ

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

public class r29_click : MonoBehaviour
{
    void OnMouseDown()   //このオブジェクトでマウスがクリックされたら
    {
        GameObject obj = GameObject.Find("main");   //メインを探し取得
        int n = int.Parse(gameObject.name);  //オブジェクト名を番号に変換
        obj.GetComponent<r29_main>().bt_click(n);
    }               //メイン(main)にある「sbt_click()」に番号を渡す
}
```

シーン名　r29click

プレハブから生成し配列に格納した多数のオブジェクトを扱う手法を学びます。

①元の画像をスライスしてボタン状のスプライトを用意します。



②シーンに配置しスケールを調整、マウスクリックを感知するための「BoxCollider2D」をアタッチし、プレハブ化します。

③プレハブを複製して、縦５×横５に配置するスクリプトを作成します。この時、オブジェクト名として生成した順に「番号」を付けておきます。

20	21	22	23	24
15	16	17	18	19
10	11	12	13	14
5	6	7	8	9
0	1	2	3	4

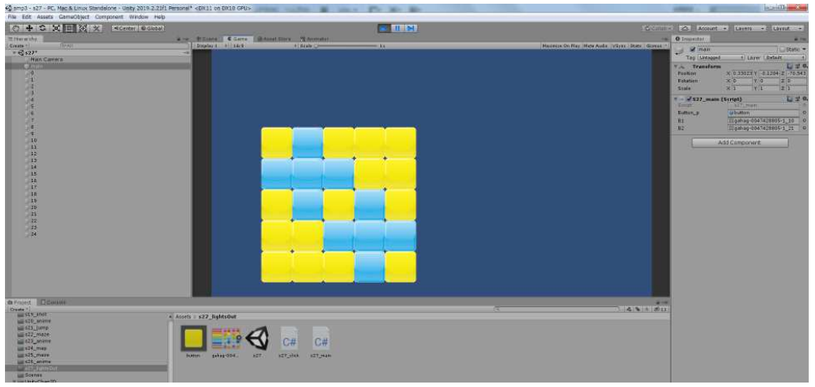
④管理オブジェクトを作り、オブジェクト名を「main」とし、スクリプトをアタッチします。

⑤メインの外部変数にボタンのスプライトを２つ登録します。

⑥オブジェクトがクリックされたらその番号（名前）を管理オブジェクトに知らせるスクリプトを記述して、プレハブにアタッチします。

ミニゲーム５ ライツアウト（パズルゲーム）

シーン名　g05light



ライツアウトは、５×５の形に並んだライトをすべて消灯（lights out）させることを目的としたパズル。あるライトを押すと、自身とその上下左右最大４個のライトが一緒に反転する。

例題２８を発展させてパズルゲーム「ライツアウト」を作成します。

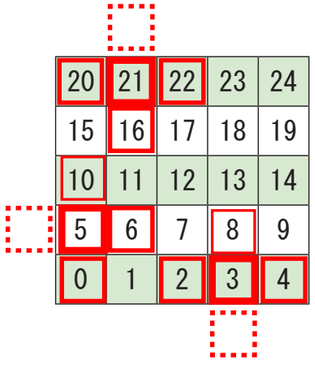
①例題２８のメインオブジェクトのスクリプトを次のように変更する。

受け取った番号の上下左右のボタンもスプライトを変更（反転）する。

②ボタンの画像（スプライト）を変更してみよう！

スクリプト	g05_main.cs
アタッチ先	管理オブジェクト

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

public class g05_main : MonoBehaviour
{
    public GameObject button_p;
    GameObject[] bt = new GameObject[25];
    public Sprite b1, b2;


    void Start()
    {
        int n;
        for( n=0;n<25;n++){
            bt[n] = Instantiate(button_p);
            bt[n].transform.position = new Vector3(n%5, n/5, 0);
            bt[n].name = n.ToString();
        }
    }

    public void bt_click(int n)
    {
        rev(n);                             //クリックされた位置を反転
        if ( n-5 >= 0   ) rev(n - 5); //      〃      下    〃
        if ( n+5 <= 24  ) rev(n + 5); //      〃      上    〃
        if ( n%5 - 1 >=0) rev(n - 1); //      〃      左    〃
        if ( n%5 + 1 <=4) rev(n +1);  //      〃      右    〃
    }               //いずれも枠内なら反転を呼び出す

    void rev(int n)       //スプライトを反転するメソッド
    {
        SpriteRenderer sr = bt[n].GetComponent<SpriteRenderer>();
        if(sr.sprite == b1 )    //もしスプライトが b1 なら
            sr.sprite = b2;     //    スプライト b2 に反転
        else                    //そうでなければ ( b2 なら)
            sr.sprite = b1;     //    スプライト b1 に反転
    }
}
```

ミニゲーム5-2 ライツアウト（画像 Ver）

シーン名　g05light

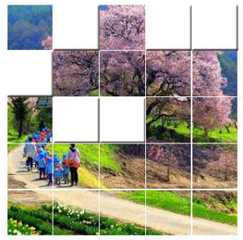
ミニゲーム５を発展させて「ライツアウト」の画像バージョンを作成します。

①画像ファイル（500px × 500px）を「Resources」にコピーする。

②スプライトエディタで画像を切り分けます。ピースの境界が分かるように 98px × 98px にします。

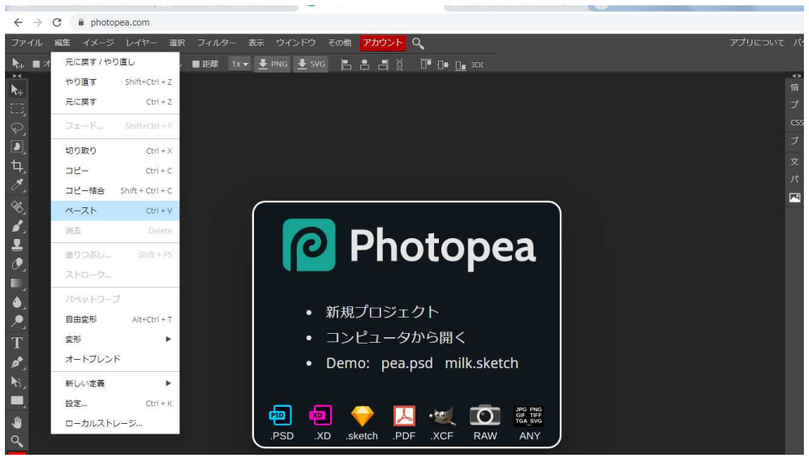
③スクリプトを次のように変更します。

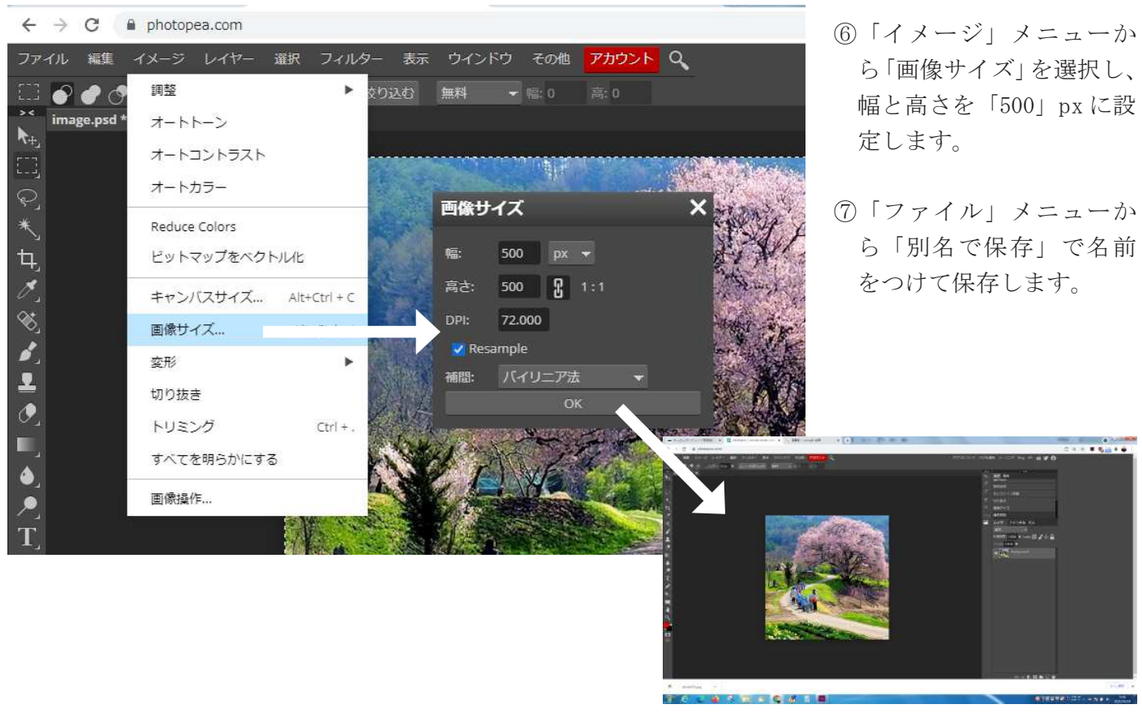
スクリプト	g05_main.cs
アタッチ先	管理オブジェクト



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

public class g05_main : MonoBehaviour
{
    public GameObject button_p;
    GameObject[] bt = new GameObject[25];
    public Sprite b1, b2;

    SpriteRenderer sr;    //SpriteRenderer 型の変数
    Sprite[] image;       // スライスした画像を読み込む配列

    void Start()
    {
        int n;
        image = Resources.LoadAll<Sprite>("sakura");  // スライス画像を Resources から読み込み
        for (n = 0; n < 25; n++)
        {
            bt[n] = Instantiate(button_p);
            bt[n].transform.position = new Vector3(n % 5, -(n / 5), 0);
            bt[n].name = n.ToString();
            sr = bt[n].GetComponent<SpriteRenderer>();  // 画像ピース配列の SpriteRenderer 取得
            sr.sprite = image[n];
        }
    }

    public void bt_click(int n)
    {
        rev(n);                         // クリックされた位置を反転
        if (n - 5 >= 0) rev(n - 5);     //      〃      下    〃
        if (n + 5 <= 24) rev(n + 5);    //      〃      上    〃
        if (n % 5 - 1 >= 0) rev(n - 1);     //      〃      左    〃
        if (n % 5 + 1 <= 4) rev(n + 1);     //      〃      右    〃
    }             // いずれも枠内なら反転を呼び出す

    void rev(int n)       // スプライトを反転するメソッド
    {
        SpriteRenderer sr = bt[n].GetComponent<SpriteRenderer>();
        if (sr.sprite == b1)    // もしスプライトが b1 なら
            sr.sprite = image[n];     //    スライス画像に反転
        else                    // そうでなければ ( b2 なら)
            sr.sprite = b1;     //    スプライト b1 に反転
    }
}
```


画像編集ソフト「Photopea」の使い方２





オリジナルゲームを作成するには素材集め、加工も大切です。
Web 上で画像の加工ができる「Photopea」を使ってみましょう。

画像のリサイズ

①インターネットで画像を探します。

②画像上で右クリック「コピー」を選択します。

③ Photopea に切替え、編集メニューから「ペースト」を実行します。

④「SHIFT」キーを押しながら「切抜きツール」を使用して「正方形」に選択します。

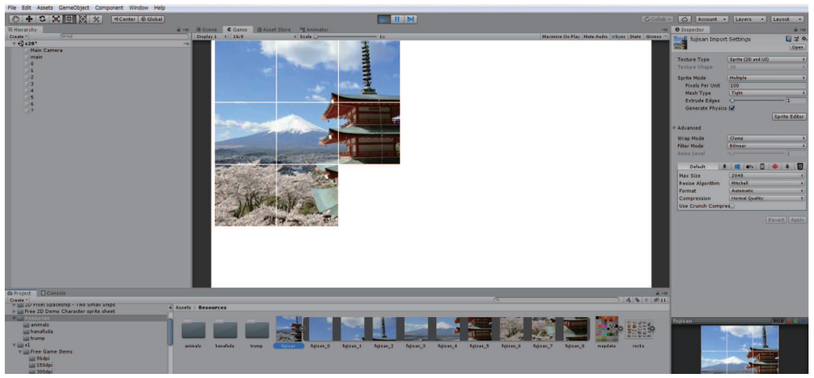
⑤「イメージ」メニューから「切り抜き」を実行。

⑥「イメージ」メニューから「画像サイズ」を選択し、幅と高さを「500」px に設定します。

⑦「ファイル」メニューから「別名で保存」で名前をつけて保存します。

ミニゲーム６ スライドパズル（８パズル）

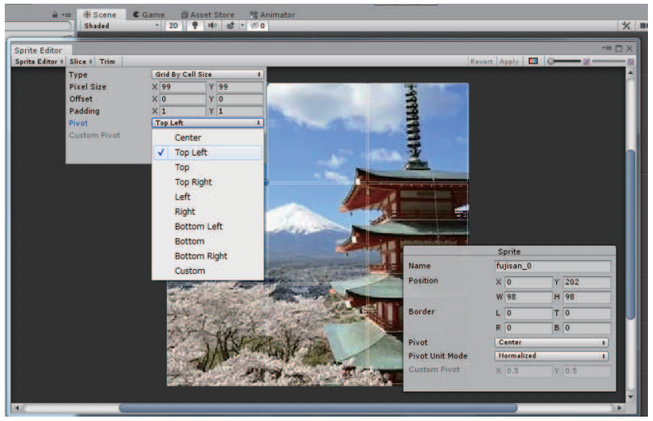
シーン名　g06puzz

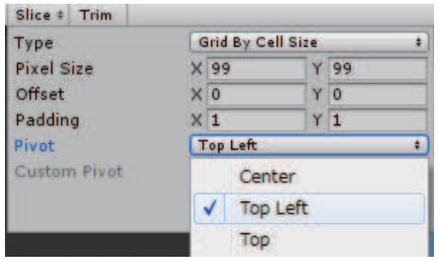


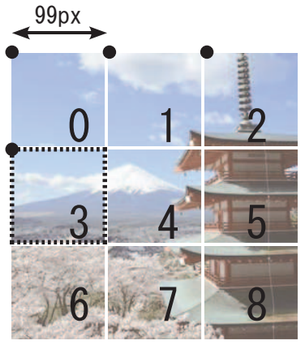
８枚のピースをスライドさせて並び換えるパズルを作成します。

①縦横３分割できるサイズの画像を用意し（例では 300px × 300px）
Resources に保存する。

②画像を９つのピースに分割するために、スプライトエディタで以下のような設定でスライスします。







オブジェクトの基準位置 →

この例の場合には「100px × 100px」にしますが、ピースとして見やすいよう、1pxの「隙間」を空けるために「99px × 99px」とします。

さらに、基準の位置（Pivot）を左上（Top left）に変更しておきます。

③スライスしたピースのひとつをシーンに配置し「Boxcollider2D」をアタッチ、プレハブ化します。

④ピース（プレハブ）を並べて配置するスクリプトを記述して、管理オブジェクトにアタッチします。

スクリプト：g06_main.cs
アタッチ先：管理オブジェクト

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

public class s28_main : MonoBehaviour
{
    public GameObject piece_p;              // ピースのプレハブ
    GameObject[] p = new GameObject[9];     // ピースのオブジェクト配列
    SpriteRenderer sr;
    Sprite[] image;

    void Start()
    { // リソースからスライスされたスプライトを読み込む
        image = Resources.LoadAll<Sprite>("fujisan");
        for (int n = 0; n < 8; n++ )          // ０～７のピースを配置
        {
            p[n] = Instantiate(piece_p);
            sr = p[n].GetComponent<SpriteRenderer>();
            sr.sprite = image[n];
            p[n].transform.position = new Vector3(n % 3, -(n / 3), 0);
            p[n].name = n.ToString();           // オブジェクト位置
        }
    }
}
```

リソースファイル名

８枚のスプライトはシーンの図のような位置に配置します。
カメラオブジェクトの位置とサイズを変更して、適当な大きさで表示するようにしておきます。

スクリプト：g06_click.cs
アタッチ先：ピースプレハブ

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

public class g06_click : MonoBehaviour
{
    void OnMouseDown()
    {
        GameObject obj = GameObject.Find("main");
        int n = int.Parse(gameObject.name);
        obj.GetComponent<g06_main>().click(n);
    }
}
```

⑤オブジェクトがクリックされたらその番号（名前）を管理オブジェクトに知らせるスクリプトを記述して、プレハブにアタッチします。

⑥管理オブジェクトのスクリプトにピースがクリックされた時の処理を追加します。

クリックされたピースの場所を番号（０～８）に変換して、その上下左右に空きピースがあるかどうかを調べます。空きピースがあれば、クリックされたピースをそこへ移動し、空きピースの番号も変更します。

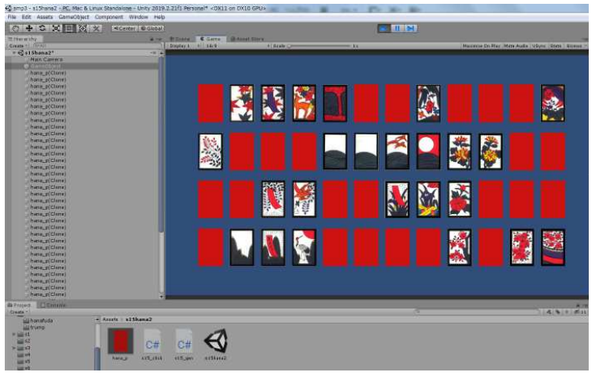
スクリプト：g06_main.cs
アタッチ先：管理オブジェクト


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

public class s28_main : MonoBehaviour
{
    public GameObject p_p;
    GameObject[] p = new GameObject[9];
    SpriteRenderer sr;
    Sprite[] image;
    int aki = 8;     // ピースの空いている場所の番号を記憶

    void Start()
    {
        image = Resources.LoadAll<Sprite>("fujisan");
        for (int n = 0; n < 8; n++ )
        {
            p[n] = Instantiate(p_p);
            sr = p[n].GetComponent<SpriteRenderer>();
            sr.sprite = image[n];
            p[n].transform.position = new Vector3(n % 3, -(n / 3), 0);
            p[n].name = n.ToString();
        }
    }
    public void click(int n)
    {
        float x = p[n].transform.position.x;        // ピースのＸ位置（左上）
        float y = p[n].transform.position.y;        // ピースのＹ位置（左上）
        int a =(int)( -(y * 3.0f)+x);               // ピースが現在ある場所（番号）を求める
        if (a - 1 == aki || a + 1 == aki || a - 3 == aki || a + 3 == aki)
        {                                           // もし上下左右のどこかが「空き」なら…
            p[n].transform.position = new Vector3(aki % 3, -(aki / 3), 0);
            aki = a;                                // ピースを移動し、現在位置を「空き」にする
        }
    }
}
```

例題 30

花札を裏返しに４８枚並べて、クリックで絵柄を表示するシーンを作成します。

シーン名　r30hana



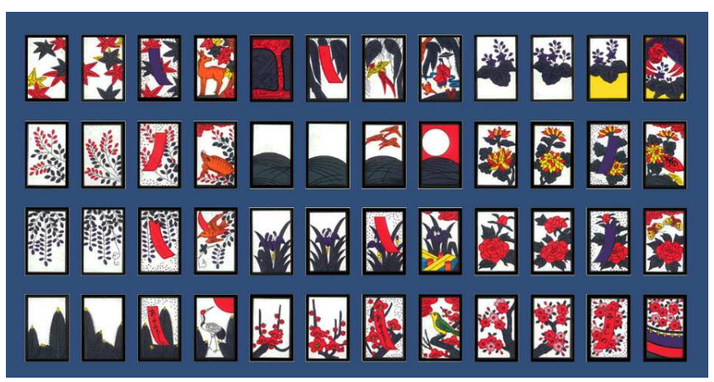
hanafuda.png

①花札の絵柄を１枚に納めた画像「hanafuda.png」をResourcesフォルダへコピー、スプライトエディタで切り分ける。

②切り分けた中の一つ、裏面の画像をシーンに配置しScaleを1.5に設定。プレハブ化し「fuda_p」と名前をつける。

スクリプト	r30_gen.cs
アタッチ先	管理オブジェクト

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

public class r30_gen : MonoBehaviour
{
    public GameObject fuda_p;                       // 札のプレハブ
    GameObject[] hf = new GameObject[48];           // 札の配列
    SpriteRenderer[] sr = new SpriteRenderer[48];   // 札のスプライト（画像）の配列

    Sprite[] image;                                 // 札画像を Resources から読み込むための配列

    void Start()
    {
        image = Resources.LoadAll<Sprite>("hanafuda");    //Resources から札画像を読み込む

        for (int n = 0; n < 4 * 12; n++)                  // ４８枚（４枚×１２か月）
        {
            hf[n] = Instantiate(fuda_p);                          // 札のインスタンス生成
            sr[n] = hf[n].GetComponent<SpriteRenderer>();         // 札の SpriteRendeer を取得
            sr[n].sprite = image[n];                              // 札のスプライト（画像）を設定
            hf[n].transform.position = new Vector3((n%12)*1.3f-7.0f, (n/12)*2-3, 0);
        }                                                         // 札を１２×４に並べる
    }
}
```

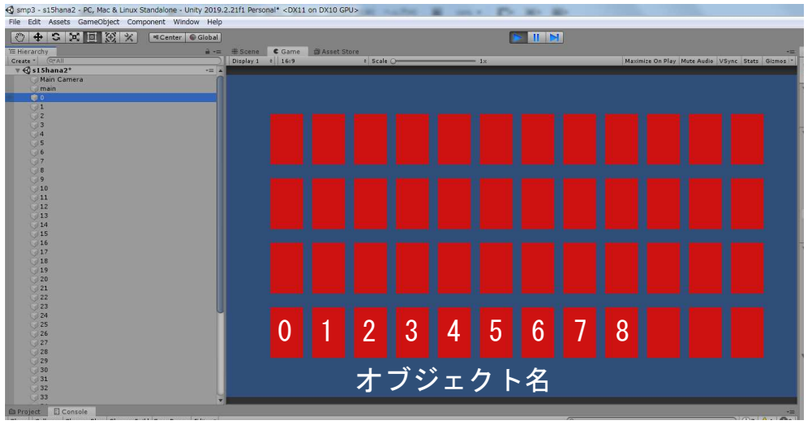


③画像を配列として読み込み例にの様に並べて配置するスクリプトを作成する。

④空のオブジェクトを作り名前を「main」と変更し、スクリプトをアタッチします。

⑤スクリプトの外部変数に札のプレハブを設定。

⑥実行を確認する。



⑦裏返しの状態で札を配置し、クリックで絵柄を表示するようにスクリプト追加、変更する。
クリックされた札の番号を受け渡しするための一つの方法として、札の複製時に配列の「番号」をオブジェクト名としています。
オブジェクトの番号を受け取ったら、その番号の絵柄を表示するメソッドを作成しておきます。

スクリプト	r30_gen.cs
アタッチ先	管理オブジェクト

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

public class r30_gen : MonoBehaviour
{
    public GameObject fuda_p;
    GameObject[] hf = new GameObject[48];
    SpriteRenderer[] sr = new SpriteRenderer[48];

    Sprite[] image;

    void Start()
    {
        image = Resources.LoadAll<Sprite>("hanafuda");

        for (int n = 0; n < 4 * 12; n++)
        {
            hf[n] = Instantiate(fuda_p);
            hf[n].name = n.ToString();        // オブジェクト名を配列番号（文字列）に設定
            sr[n] = hf[n].GetComponent<SpriteRenderer>();
            hf[n].transform.position = new Vector3((n%12)*1.3f-7.0f, (n/12)*2-3, 0);
        }
    }

    public void card(int n)             // 外部からアクセス可能（public）なメソッド
    {
        sr[n].sprite = image[n];        // 受け取った n 番の画像（スプライト）を表示
    }
}
```

スクリプト	r30_click.cs
アタッチ先	花札プレハブ

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

public class r30_click : MonoBehaviour
{
    void OnMouseDown()                  // 札がクリックされたら
    {
        GameObject obj = GameObject.Find("main");
        int n = int.Parse(gameObject.name);      // 札の番号
        obj.GetComponent<r29_gen>().card(n);
    }                                   // 絵柄表示メソッド card() を呼び出す
}
```

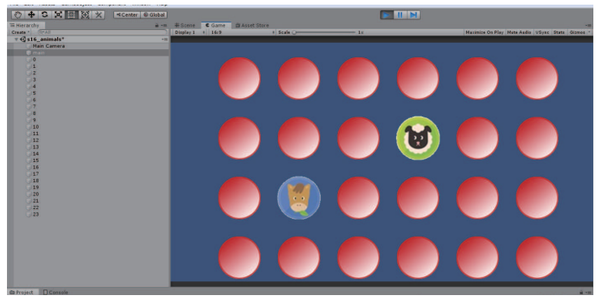
⑧マウスのクリックを取得するため、複製元のプレハブに「BoxCollider2D」をアタッチします。

⑨札がクリックされたらオブジェクト名を数値に変換して、指定番号の画像を表示するメソッドを呼び出すようにスクリプトを作成し、プレハブにアタッチします。

ミニゲーム７ 絵合わせゲーム（神経衰弱）

シーン名　g07card

例題○○をベースに作成していきます。



①動物の絵柄を１枚に納めた画像「animals.png」を「Resources/animals」フォルダへコピーし、スプライトエディタで切り分ける。

②切り分けた中の一つ、裏面の画像をシーンに配置しScaleを「２」に設定。プレハブ化し「card_p」と名前をつける。

③画像をまとめて配列として読み込み例にの様に並べて配置するスクリプトを作成する。この時全ての画像ではなく、０番～１１番までの１２枚を２枚づつ表示するように設定する。

animals.png

④空のオブジェクトを作り名前を「main」と変更し、スクリプトをアタッチします。

⑤スクリプトの外部変数に札のプレハブを設定

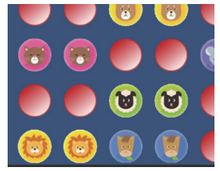
image[] 配列（Sprite 型）

スクリプト：g07_main.cs
アタッチ先：管理オブジェクト

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

public class g07_main : MonoBehaviour
{
    public GameObject card_p;          // カードのプレハブ
    GameObject[] card = new GameObject[24]; // カードの配列

    Sprite[] image;                    // カードの絵柄の配列
    int cnt = 0;                       // クリックしたカードのカウント
    int card1, card2;                  // １,２枚目の可０度を記憶
    SpriteRenderer sr1,sr2;            // カードの SpriteRenderer

    void Start()
    {
        image = Resources.LoadAll<Sprite>("animals");  // 絵柄を Resources/animals から読み込む

        for (int n = 0; n < 6 * 4; n++)      // 横６×縦４　２４回繰り返す
        {
            card[n] = Instantiate(card_p);   // カードを複製しオブジェクト配列に入れる
            card[n].name = n.ToString();     // オブジェクトの名前を番号に変更
            s17_click sp = card[n].GetComponent<s17_click>();  // クリックのスクリプトを取得
            sp.back = image[n / 2];          // スクリプトの外部変数 back にスプライトを設定
            //sr1 = card[n].GetComponent<SpriteRenderer>();
            //sr1.sprite = image[n / 2];     // 絵柄の表示チェック
            card[n].transform.position = new Vector3((n% 6)*2.5f-6.0f,(n/6)*2.5f-4,0);
        }                                    // オブジェクトの位置を設定
    }
}
```

スクリプト：g07_click.cs
アタッチ先：カードプレハブ

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

public class g07_click : MonoBehaviour
{
    public Sprite front;    // 表面のスプライト
    public Sprite back;     // 裏面のスプライト

    void OnMouseDown()
    {
        SpriteRenderer sr = GetComponent<SpriteRenderer>();
        sr.sprite = back;       // スプライトを絵柄（裏面）に変更
    }
}
```



クリックしたカードの番号（オブジェクト名）メインに渡す

スクリプト：g07_click.cs
アタッチ先：カードプレハブ

```
・・・省略

    void OnMouseDown()
    {
        SpriteRenderer sr = GetComponent<SpriteRenderer>();
        if (sr.sprite == front)        // もしスプライトが表面なら
        {
            sr.sprite = back;          // 絵柄に切替表示

            GameObject obj = GameObject.Find("main");
            int n = int.Parse(gameObject.name);
            obj.GetComponent<g07_main>().card_turn(n);
        }         // カード番号をメインの card_turn() メソッドに渡す
    }
}
```

⑥マウスのクリックを取得するため、複製元のプレハブに「BoxCollider2D」をアタッチします。

⑦札がクリックされたら画像を表示するスクリプトを作成し、プレハブにアタッチします。

⑧カードを裏面のまま複製するようスクリプトを変更し、実行を確認します。

マウスをクリックすると最初にカードが並べられた状態で画像が表示されます。

⑨マウスをクリックしたらそのカードの番号（オブジェクト名）をメインに渡して１枚目のカードと２枚目のカードが同じかどうかを判定するスクリプトを追加します。

１枚目、２枚目のカードが一致しているかを判定する。

スクリプト：g07_main.cs
アタッチ先：管理オブジェクト

```
・・・省略
    public void card_turn(int n)     // クリックされたカード番号を受取る
    {
        if (cnt % 2 == 0)            // １枚目のカードなら
            card1 = n;               //   カード番号を記憶
        else                         // ２枚目のカードなら
        {
            card2 = n;
            sr1 = card[card1].GetComponent<SpriteRenderer>();  // １枚目のカードの SpriteRenderer
            sr2 = card[card2].GetComponent<SpriteRenderer>();  // ２枚目　　〃
            if (sr1.sprite == sr2.sprite )          // もし１枚目と２枚目のカードが同じなら
            {
                Destroy(card[card1], 1.0f);         //     １枚目のカードを１秒後に消去する
                Destroy(card[card2], 1.0f);         //     １枚目の　　〃
            }
            else                                    // 同じでなければ
            {
                sr1.sprite = image[23];             //     １枚目のカードを表面に戻す
                sr2.sprite = image[23];             //     ２枚目　　〃
            }
        }
        cnt++;      // カードをクリックされた回数をカウント
    }
```

配列の内容をシャッフルする（ばらばらにする）

⑩順番に並んでいるカードをばらばらにシャッフルする処理を追加します。

シャッフルアルゴリズム

整列している配列データの先頭から、乱数で発生させた番号の配列の内容の交換を繰り返します。

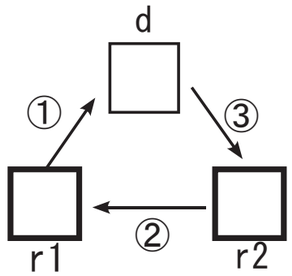
交換するデータはカードの表示位置（transform.position）です。

スクリプト	g07_main.cs
アタッチ先	管理オブジェクト

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

public class g07_main : MonoBehaviour
{
    public GameObject card_p;
    GameObject[] card = new GameObject[24];

    Sprite[] image;
    int cnt = 0;
    int card1, card2;
    SpriteRenderer sr1,sr2;

    void Start()
    {
        image = Resources.LoadAll<Sprite>("animals");

        for (int n = 0; n < 6 * 4; n++)
        {
            card[n] = Instantiate(card_p);
            card[n].name = n.ToString();
            s17_click sp = card[n].GetComponent<s17_click>();
            sp.back = image[n / 2];
            card[n].transform.position = new Vector3((n%6)*2.5f-6.0f,(n/6)*2.5f-4, 0);
        }

        GameObject d = Instantiate(card_p);          //オブジェクトの交換用の変数を用意
        for (int n = 0; n < 24; n++)
        {
            int r1 =Random.Range(n, 24);             //交換する配列を乱数で決める
            d.transform.position = card[r1].transform.position;   //カードのポジションを交換
            card[r1].transform.position = card[n].transform.position;
            card[n].transform.position = d.transform.position;
        }
        Destroy(d);                                  //交換用の変数を消去
    }
```

次ページへ続く

２つ変数の内容を交換

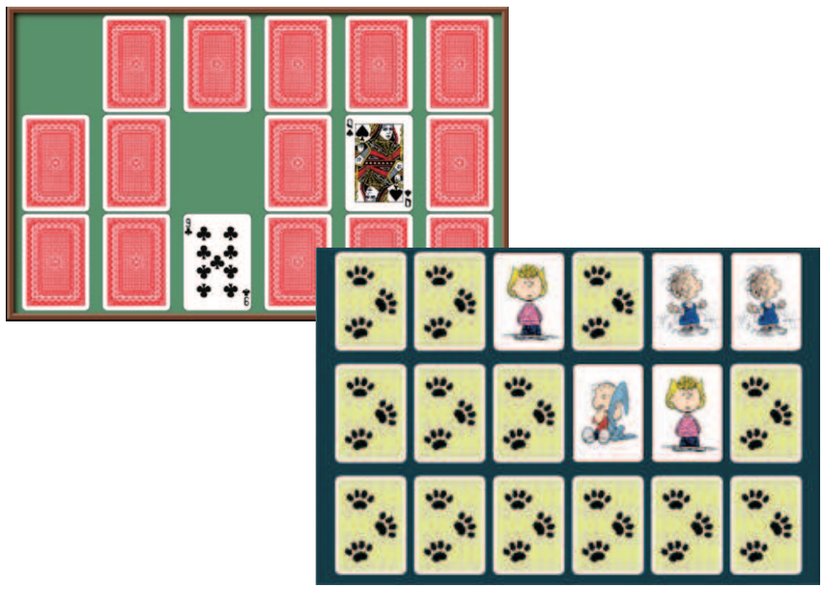



```
    public void card_turn(int n)
    {
        if (cnt % 2 == 0)
            card1 = n;
        else
        {
            card2 = n;
            sr1 = card[card1].GetComponent<SpriteRenderer>();
            sr2 = card[card2].GetComponent<SpriteRenderer>();
            if (sr1.sprite == sr2.sprite )
            {
                Destroy(card[card1], 1.0f);
                Destroy(card[card2], 1.0f);
            }
            else
            {
                Invoke("card_ret", 1);        // １秒後に card_ret() を実行
            }
        }
        cnt++;
    }

    void card_ret()
    {
        sr1.sprite = image[23];               //カードのスプライトを表面に戻す
        sr2.sprite = image[23];
    }
}
```

← image[23]

スクリプト	g06_main.cs
アタッチ先	管理オブジェクト

オブジェクト	スクリプト	役　割
管理	g05_main.cs	花札生成、配置、クリック１枚目、２枚目判定他
花札（プレハブ）	g05_click.cs	半札のクリックを拾って、カード番号を渡す

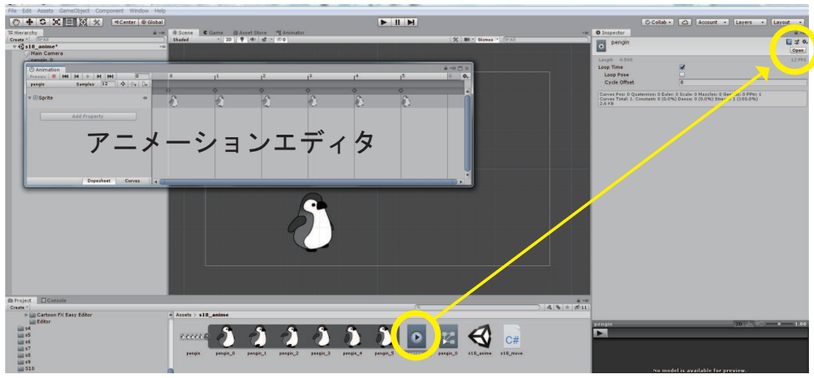


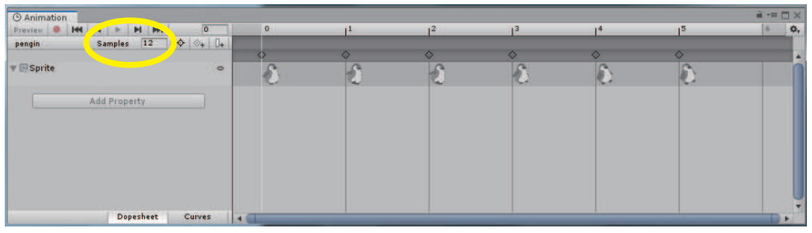
⑪カードが一致した場合には１秒後に消去されますが、一致しなかった場合にはすぐにもとにもどってしまいます。

一致しなかった場合にも絵柄を１秒間、表示させておくように、スクリプトを追加します。

カードの種類、枚数などを変更してオリジナルゲームにしてみましょう！！

例題 31　アニメーション





アニメーション
クリップ

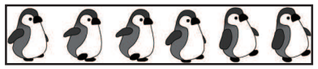
アニメーター
コントローラー

Scale　X = 3

Scale　X = -3

シーン名　r31anime

動きのある画像をいくつか並べてまとめたアトラス画像を、連続的に切り替えてパラパラ漫画の様に表示するアニメーションの方法を学びます。



pengin.png

①ペンギンの画像をインポートしスプライトエディタで切り分けます。

②切り分けた画像を全て選択、まとめてシーンへドラッグ＆ドロップします。

③アニメーションクリップの設定ダイアログが現れます。ファイル名を「pengin.anim」とつけて保存します。

アニメーションクリップとアニメーターコントローラーの２つが作成され、Animator コンポーネントも追加されます。

④スケールを３倍に設定して、実行してみましょう！

⑤アニメーションファイルを選択して「Open」ボタンをクリックすると「アニメーションエデイター」が開きます。

⑥「Samples」の数値を変更すると動きの速さを変更できます。

⑦オブジェクトの「Scale」のＸの値を「－３」に設定すると画像を逆向きにできます。

「←　→」のキー入力でアニメーションの再生スピードを変化させるスクリプトを作成してみます。

スクリプト	r31_back.cs
アタッチ先	背景

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

public class r31_back : MonoBehaviour
{
    float back_sp = -0.1f;          // 背景のスクロールスピード

    void Update()
    {
        transform.Translate(back_sp, 0, 0);  // スクロール
        if (transform.position.x <= -17.0f)
            transform.position = new Vector3(17.0f, 0, 0);
    }                               // 背景が左端まで来たら、右端に戻す
}
```

⑧背景として「back2.png」をシーンに配置し、右から左にスクロールして移動するようスクリプトを作成、アタッチします。

スクリプト	r31_key.cs
アタッチ先	ペンギン

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

public class r31_key : MonoBehaviour
{
    Animator am;            // アニメーション型の変数
    float sp = 1.0f;        // 再生スピード

    void Start()
    {
        am = gameObject.GetComponent<Animator>();
    }                               //Animator コンポーネントを取得

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.RightArrow))  // キーが「←」
        {
            sp += 0.1f;                             // スピードを＋0.1
        }
        if (Input.GetKeyDown(KeyCode.LeftArrow))   // キーが「←」
        {
            sp -= 0.1f;                             // スピードを－0.1
        }
        am.speed = sp;    //Animator コンポーネントにスピードを設定
    }
}
```

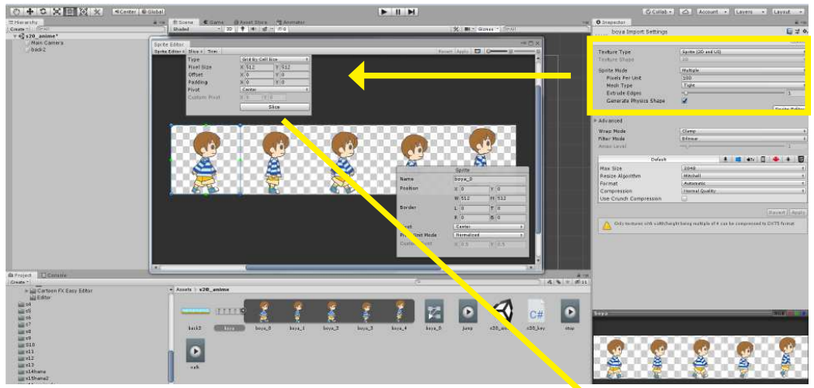
⑨キー入力でアニメーションの再生スピードを変化させるスクリプトを作成し、ペンギンオブジェクトをアタッチします。

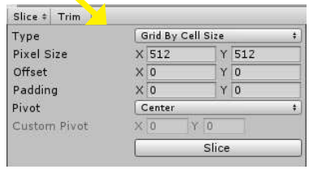
アニメーションスピード

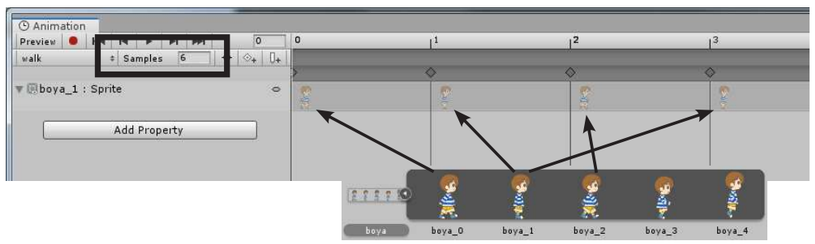
ダウン　　アップ

※アニメーションのスピードと連動して、背景のスクロールスピードも変化させるにはどうすればよいでしょう・・・？

例題 32　アニメーション







シーン名　r32anime

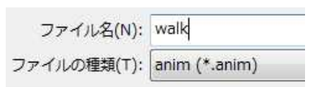
動きのある画像をいくつか並べてまとめたアトラス画像を、連続的に切り替えてパラパラ漫画の様に表示するアニメーションの方法を学びます。

①「boya.png」をインポート、SpriteMode を「Multiple」に設定しスプライトエディタを起動します。

Type を「GridByCellSize」とし PixelSize を XY ともに「512」に設定し Slice します。

②切り分けられた「boy1_1」をシーンへ配置し、選択した状態で「Window」メニューから「Animation」→「Animation」と選択しアニメーションエディタを起動、「Create」ボタンをクリックします。

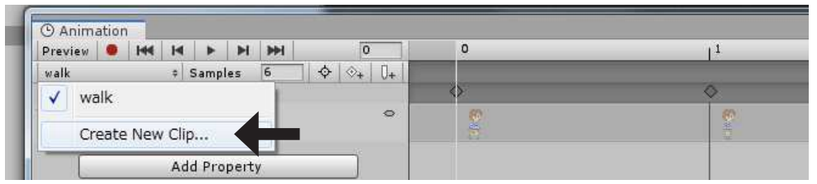
「walk」と名前をつけてアニメーションクリップを保存します。

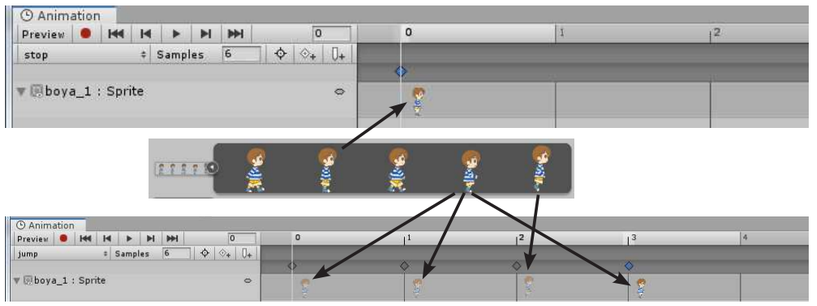


③「Samples」を「６」に設定し、タイムラインに歩行しているスプライトを順番に配置します。

④アニメーションエディタを閉じて実行を確認してみます。

プロジェクトウインドにアニメーションクリップとアニメーションコントローラが作成されます。

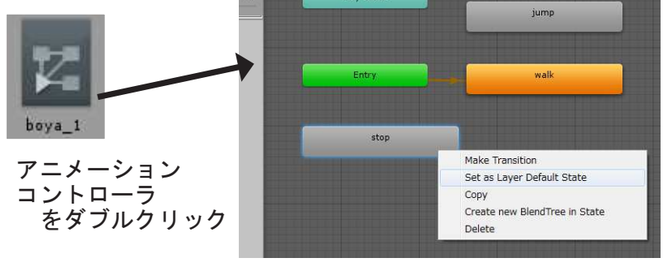


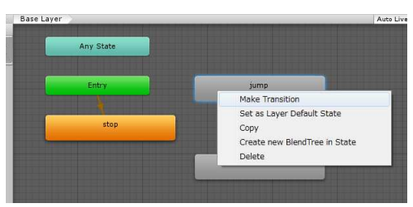


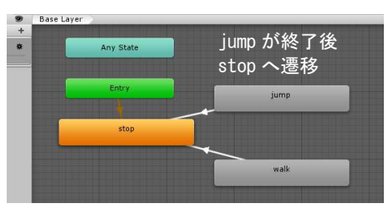
スクリプト	r32_key.cs
アタッチ先	少年

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

public class r32_key : MonoBehaviour
{
    void Update()
    {
        var ac = GetComponent<Animator>();  //Animator コントローラーを取得

        if (Input.GetKey(KeyCode.RightArrow) ){     //→キーなら
            ac.Play("walk");                        //「walk」を再生
        }
        if (Input.GetKey(KeyCode.UpArrow) ){        //↑キーなら
            ac.Play("jump");                        //「jump」を再生
        }
        if (Input.GetKey(KeyCode.DownArrow)){       //↓キーなら
            ac.Play("stop");                        //「stop」を再生
        }
    }
}
```

アニメータ







アニメーションの種類を増やしてみます。

⑤オブジェクトを選択した状態でアニメーションエディタを起動します。

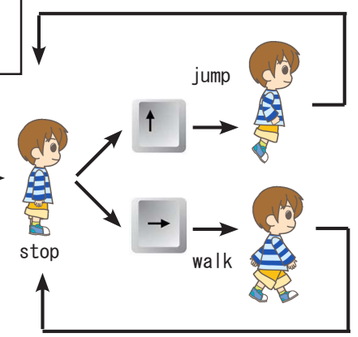
⑥「walk」の文字上でクリックし「Create New Clip」を選択し、「stop」と名前を付けます。

⑦「Samples」を「６」に設定し、立ち止まっているスプライトをタイムラインに配置します。

⑧同じように「jump」クリップも追加します。

⑨キー操作でアニメーションを切り替えるスクリプトを作成してオブジェクトにアタッチ、実行を確認します。

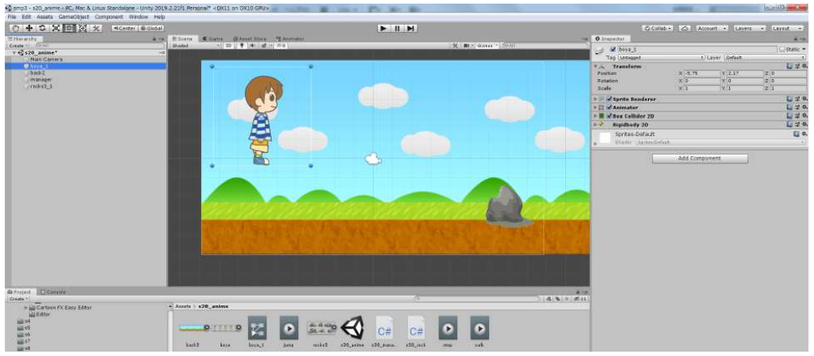
下図の様なアニメーションの遷移（せんい）を設定してみます。



⑩アニメータを開き「stop」上で右クリック「Set …」を選択し初期状態に設定します。

⑪「jump」「walk」から「MakeTransition」を選択して遷移を設定します。

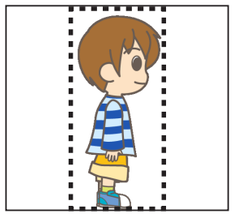
ミニゲーム８ 横スクロールゲーム



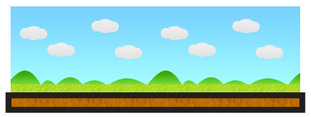
シーン名　g08yoko

「例題３１」をベースに作成していきます。
新たにフォルダを作成して、ファイルメニューから「Save as…」で名前をつけてシーンを保存します。
今回は**１つのスクリプトで全てのオブジェクトを管理**してみます。

①プレイヤーのオブジェクトに「BoxCollider2D」をアタッチし、領域を調整します。「RigidBody2D」もアタッチします。



②背景画像を配置し、地面の部分に「BoxCollider2D」をアタッチします。



BoxCollider2D

③空のオブジェクト（管理オブジェクト）を作成し「main」と名前をつけます。

④例題３１のスクリプトをコピーして「g08_main.cs」を作成し、スペースキーでジャンプする（上方向に力を加える）ように変更し、ストップは削除しておきます。
スクリプトは管理オブジェクト「main」にアタッチします。

⑤キー入力でアニメーションの再生スピードを変えられるようにスクリプトを追加します。

④スペースキー入力でジャンプする

スクリプト	g08_main.cs
アタッチ先	管理オブジェクト

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

public class g08_main : MonoBehaviour
{
    GameObject player;          // プレイヤーオブジェクト
    Animator ac;                // アニメーターコンポーネント

    void Start()
    {
        player = GameObject.Find("boya_1");     // プレイヤーを取得
        ac = player.GetComponent<Animator>();   // アニメーターを取得
    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.RightArrow)){
            ac.Play("walk");            //「→」なら歩くアニメ
        }
        if (Input.GetKeyDown(KeyCode.Space)){
            ac.Play("jump");            //「スペース」ならジャンプアニメ
            Rigidbody2D rb = player.GetComponent<Rigidbody2D>();
            rb.AddForce(new Vector2(0, 400));  // 上方向に力を加える
        }
    }
}
```

⑤キー入力でアニメーションの再生スピードを変化

スクリプト	g08_main.cs
アタッチ先	管理オブジェクト

```
・・・省略
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.RightArrow))
        {
            ac.speed += 0.05f;      //アニメーションスピードを速くする
            ac.Play("walk");
        }
        if (Input.GetKeyDown(KeyCode.LeftArrow))
        {
            ac.speed -= 0.05f;      //アニメーションスピードを遅くする
            ac.Play("walk");
        }
      ・・・省略
    }
```


⑥背景を横にスクロール

スクリプト	g08_main.cs
アタッチ先	管理オブジェクト

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

public class g08_main : MonoBehaviour
{
    GameObject player;
    GameObject back;      // 背景オブジェクト
    Animator ac;
    float bk_speed;

    void Start()
    {
        player = GameObject.Find("boya_1");
        ac = player.GetComponent<Animator>();
        back = GameObject.Find("back2");     //背景オブジェクトを取得
    }

    void Update()
    {
        ・・・省略
        if (Input.GetKeyDown(KeyCode.Space))
        {
            ac.Play("jump");
            Rigidbody2D rb = player.GetComponent<Rigidbody2D>();
            rb.AddForce(new Vector2(0, 400));
        }
        bk_speed = -(ac.speed / 15.0f);     //背景のスクロールスピード
        back.transform.Translate(bk_speed, 0, 0);  //背景をスクロール
        if (back.transform.position.x <= -17.0f)
            back.transform.position = new Vector3(17.0f, 0, 0);
    }
}
```

⑥背景画像を横スクロールするようスクリプトを追加します。このとき、アニメーションの再生スピードと、背景のスクロールスピードが連動するよう、１度の移動量を計算で求めています。

実行して動作を確認しましょう。

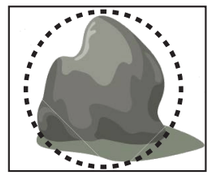
⑦障害物（岩石）を出現

スクリプト	g08_main.cs
アタッチ先	管理オブジェクト

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

public class g08_main : MonoBehaviour
{
    GameObject player;
    GameObject back;
    GameObject rock;      // 岩石オブジェクト
    Animator ac;

    float bk_speed;

    void Start()
    {
        player = GameObject.Find("boya_1");
        ac = player.GetComponent<Animator>();
        back = GameObject.Find("back2");
        rock = GameObject.Find("rocks3_1");  // 岩石オブジェクトを取得
    }

    void Update()
    {
        ・・・省略
        rock.transform.Translate(bk_speed, 0, 0);  // 岩石を移動
        if (rock.transform.position.x < -8.0f)
            rock.transform.position = new Vector3(8.0f, -2.5f, 0);
    }
}
```

⑦障害物として岩石を出現させましょう。

岩石オブジェクトを配置し「CircleCollider2D」をアタッチします。

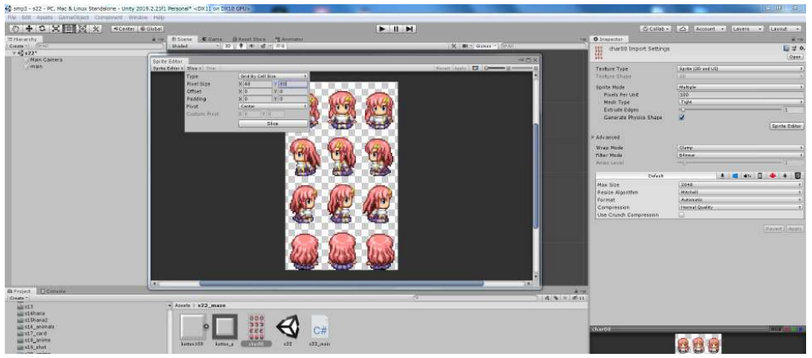


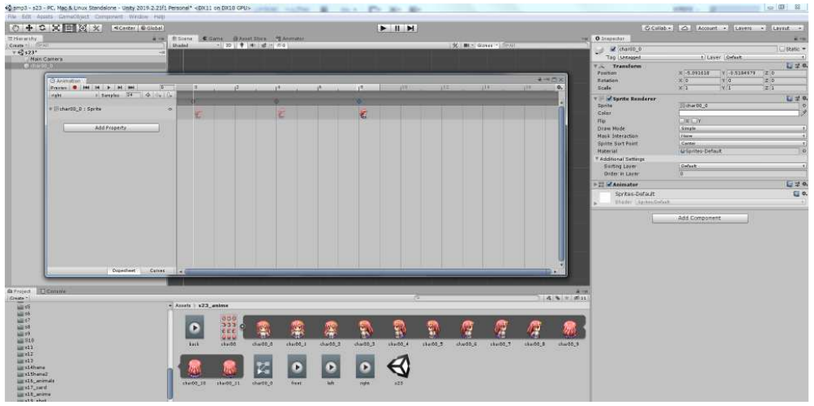
背景のスクロールスピードに合わせて、右から左に移動させます。

プレイヤーと岩石の衝突判定はコライダーに任せ、特に処理は行っていません。

実行して動作を確認しましょう。
今回の例題はゲームのベースとなるものです。これにＢＧＭや得点を付け加えて完成度を高めましょう！

演習 17　キャラクタのアニメーションを切り替えながら４方向へ移動する





```
using System.Collections;                          スクリプト | e17_key.cs
using System.Collections.Generic;                  アタッチ先 | キャラクタ
using UnityEngine;

public class e17_key : MonoBehaviour
{
    void Update()
    {                                  //Animator コントローラーを取得
        var ac = GetComponent<Animator>();

        if (Input.GetKey(KeyCode.UpArrow) ){     // ↑キーなら
            ac.Play("back");                     //「back」を再生
            transform.Translate(0, 0.01f, 0);
        }
        if (Input.GetKey(KeyCode.DownArrow)){    // ↓キーなら
            ac.Play("front");                    //「fornt」を再生
            transform.Translate(0, -0.01f, 0);
        }
        if (Input.GetKey(KeyCode.LeftArrow) ){   //←キーなら
            ac.Play("left");                     //「left」を再生
            transform.Translate(-0.01f,0, 0);
        }
        if                                       // →キーなら
                                                 //「right」を再生
        }
    }
}
```


シーン名　e17anime

①４方向への歩行アニメーションがまとめられたアトラス画像をスライスします。

画像全体のサイズは
　192px × 320px
スライス後のスプライトは
　横　192/3 = 64px
　縦　320/4 = 80 px

②アニメータを起動し、スライスされた画像を配置して前後左右へ歩行するアニメーションを作成します。

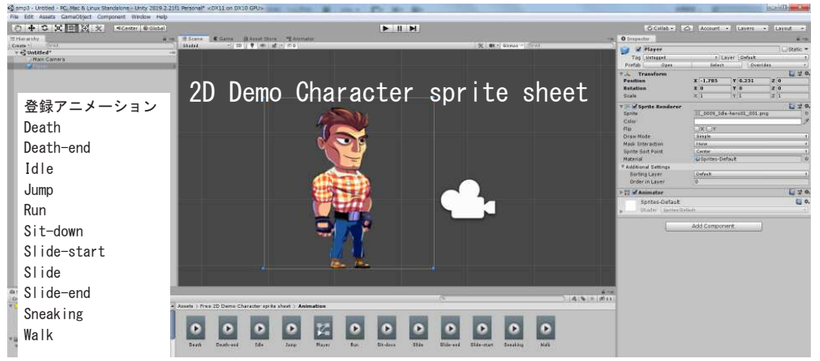
前　front ↑
後　back ↓
左　left ←
右　right→

③カーソルキーで４方向へ移動するスクリプトを記述してアタッチします。

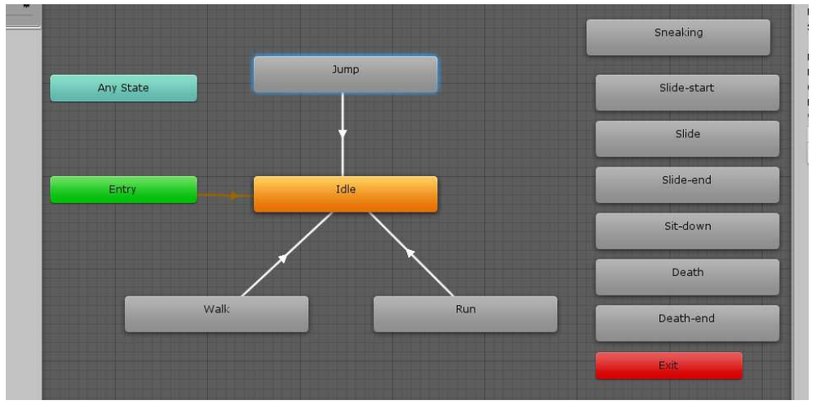
カメラオブジェクトのサイズを小さくして、キャラクタの表示サイズを大きくします。

実行を確認して、スピードや移動量を調整しましょう。

演習 18　アセットストアで入手したアニメーションデータを使用する。

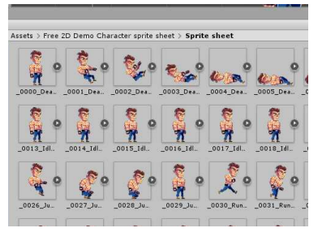



```
using System.Collections;                          スクリプト | e18_key.cs
using System.Collections.Generic;                  アタッチ先 | キャラクタ
using UnityEngine;

public class e18_key : MonoBehaviour
{
    void Update()
    {
        var ac = GetComponent<Animator>(); //Animator コントローラーを取得

        if (Input.GetKey(KeyCode.RightArrow))              // →キーなら
        {
            transform.localScale = new Vector3(1, 1, 1);   // 向きを元に
            if (Input.GetKey(KeyCode.LeftShift))           //SHIFT キーが押下
                ac.Play("Run");                            //「Run」を再生
            else
                ac.Play("Walk");                           //「Walk」を再生
        }
        if (Input.GetKey(KeyCode.LeftArrow))               //←キーなら
        {
            transform.localScale = new Vector3(-1, 1, 1);  // 向きを反転
            if (Input.GetKey(KeyCode.LeftShift))           //SHIFT キーが押下
                ac.Play("Run");                            //「Run」を再生
            else
                ac.Play("Walk");                           //「Walk」を再生
        }
        if                                                 //スペースキーなら
        {

        }
    }
}
```



シーン名　e18anime

アセットストアにはクオリティの高いアニメーションデータが多数あります。これらを使用差売ればより簡単に見栄えのする下０無の作成が出来ます。
今回は「2D Demo Character sprite sheet」を使用してみます。これは 170 未以上のスプライトを組み合わせて歩く、走るなどの８種類のアニメーションが定義されています。



①新シーンを作成し、使用するアセットをプロジェクトにインポートします。

②「Prefab」フォルダにあるプレハブをシーンに配置し実行すると、登録されているアニメーションが順番に再生されます。

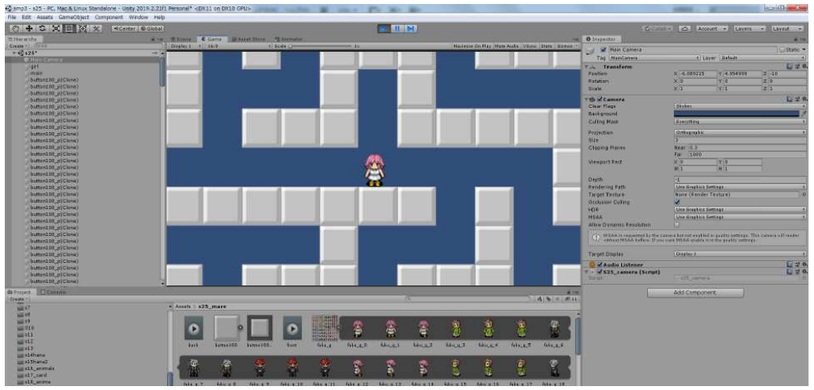
カメラオブジェクトの Size を変更して表示の大きさを調整します。

③Animation フォルダにある「Player」というアニメータを起動し、 アニメーション間に設定されている遷移（矢印）を削除します。

④次のキー入力でそれぞれのアニメーションを動作させるようスクリプトを記述し、player オブジェクトにアタッチします。

⑤それぞれの動作後「Idle」へ戻るよう遷移を設定します。

ミニゲーム９ 迷路ゲーム



シーン名	g09maze

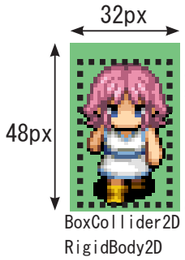
迷路を歩いて移動するゲームベースを作成します。

①４方向の歩行アニメーションの元となるアトラス画像を用意し、同じ大きさにスライスします。

前

左

②スライスしたスプライトのひとつをシーンに配置し、以下を設定します。
・オブジェクト名「girl」
・Scale を 100px 程度
・BoxCollider2D
・RigidBody2D



③演習１４同様に歩行アニメーションしながら、キー入力で４方向へ移動するシーンを作成します。

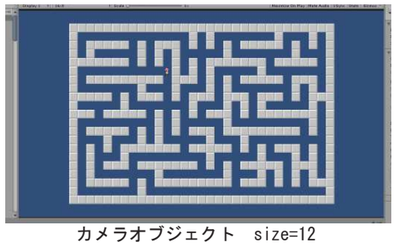
④ブロック画像をシーンに配置し「BoxCollide2D」をアタッチし、プレハブ化します。

⑤管理オブジェクトを作成し、例題３０を参考に迷路を作成するスクリプトを記述しアタッチします。

カメラオブジェクトの「size」を変更しながら実行してみましょう！

キー入力による移動量なども調整しましょう

スクリプト	g09_main.cs
アタッチ先	管理オブジェクト

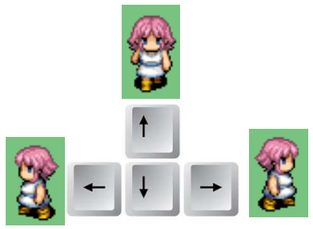


カメラオブジェクト　size=12

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

public class g09_main : MonoBehaviour
{
    public GameObject kabe_p;
    int[,] maze = new int[100,100];
    int W = 31, H = 21;

    void Start()
    {
        int x, y;

        for (y = 0; y < H; y++)
        {
            for (x = 0; x < W; x++)
            {
                maze[y, x] = 0;
                if(y==0 || y==H-1 || x==0 || x==W-1) maze[y,x]=1;
            }
        }

      boutaosi();  //迷路生成関数呼び出し

        for (y = 0; y < H; y++)
        {
            for (x = 0; x < W; x++)
            {
                if (maze[y, x] == 1)
                {
                    GameObject kb = Instantiate(kabe_p);
                    kb.transform.position = new Vector3(x-15, 10-y, 0);
                }
            }
        }
    }

    void boutaosi()     //棒倒し法による迷路生成
    {
        int x, y,r;
        Random.InitState(System.DateTime.Now.Millisecond); //乱数の種の初期化
        for (y = 2; y <= H-3; y += 2)                      //縦方向の繰り返し（棒の場所）
        {
            for (x = 2; x <= W-3; x += 2)                  //横方向の繰り返し（棒の場所）
            {
                maze[y,x] = 1;                             //棒の場所は「１」
                r = Random.Range(0,4);                     //0 ～ 3 の乱数
                switch (r)                            //乱数の値で分岐（switch ～ case 文）
                {
                    case 0: maze[y - 1,x    ] = 1; break;  //もし ０ なら 上に倒す
                    case 1: maze[y + 1,x    ] = 1; break;  //     １      下  〃
                    case 2: maze[y    ,x - 1] = 1; break;  //     ２      左  〃
                    case 3: maze[y    ,x + 1] = 1; break;  //     ３      右  〃
                }
            }
        }
    }
}
```


スクリプト	g09_key.cs
アタッチ先	キャラクタ

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

public class g09_key : MonoBehaviour
{
    void Update()
    {
        var ac = GetComponent<Animator>();     //Animator コントローラーを取得

        if (Input.GetKey(KeyCode.UpArrow) ){     // ↑キーなら
            ac.Play("back");                     //「back」を再生
            transform.Translate(0, 0.05f, 0);
        }
        if (Input.GetKey(KeyCode.DownArrow)){    // ↓キーなら
            ac.Play("front");                    //「fornt」を再生
            transform.Translate(0, -0.05f, 0);
        }
        if (Input.GetKey(KeyCode.LeftArrow) ){   // ←キーなら
            ac.Play("left");                     //   「left」を再生
            transform.Translate(-0.05f,0, 0);
        }
        if (Input.GetKey(KeyCode.RightArrow) ){  //  →キーなら
            ac.Play("right");                    //   「right」を再生
            transform.Translate(0.05f,0, 0);
        }
    }
}
```

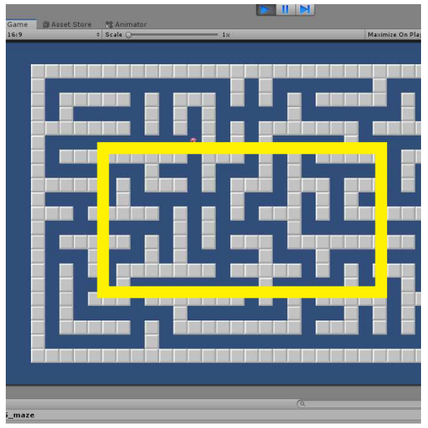


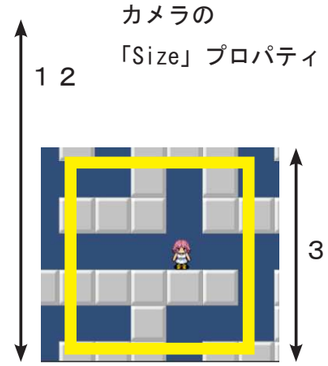
このままだと、キャラクタと壁の衝突時にキャラクタが傾いてしまいます。「RigidBody2D」コンポーネントの「FrezeRotation」の「z」軸にチェックを入れ、不要な回転を防ぎます。

キャラクタの周囲だけ画面に表示し、迷路の部分がスクロールして表示するようにしてみます。

⑥カメラオブジェクトの「size」を「３」程度に小さく変更します。

⑦キャラクタの位置にカメラが追随するようスクリプトを記述し、カメラにアタッチします。



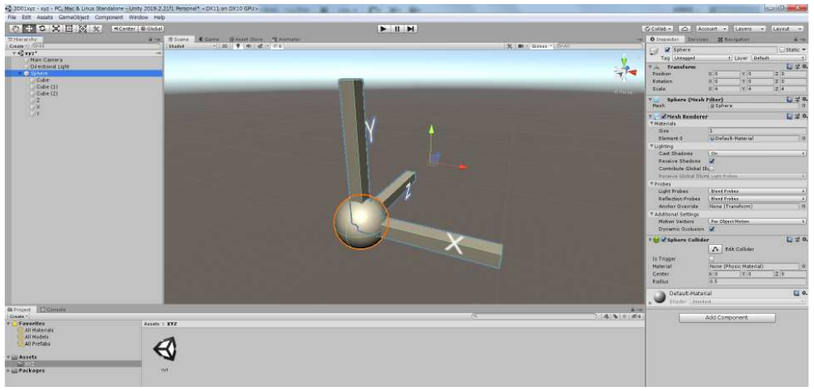


スクリプト	g09_camera.cs
アタッチ先	メインカメラ

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

public class g09_camera : MonoBehaviour
{
    GameObject player;          //キャラクタ（プレイヤー）のゲームオブジェクト

    void Start()
    {
        player = GameObject.Find("girl");   //ゲームオブジェクト名「girl」を探してオブジェクト取得
    }

    void Update()
    {
        Vector3 p_pos = player.transform.position; //キャラクタ（プレイヤー）の位置取得
        transform.position = new Vector3(p_pos.x, p_pos.y, transform.position.z);
    }                                  //カメラの中心位置をキャラクタ（プレイヤー）の位置と合わせる
}
```

例題 33 Unity 3D の基本１

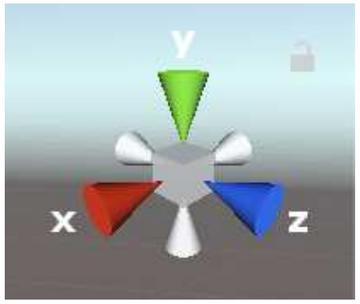
シーン名　r33xyz



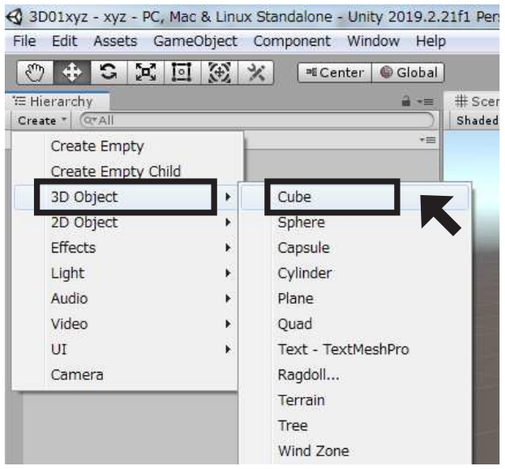
Unityの３Ｄ座標の基本を学びます。

①共通フォルダにある「3D01xyz」をフォルダごと自分のネットワークドライブにコピーする。

②Unityを起動しコピーした「3D01xyz」を読み込み、３Ｄ空間でのマウス操作などを確認する

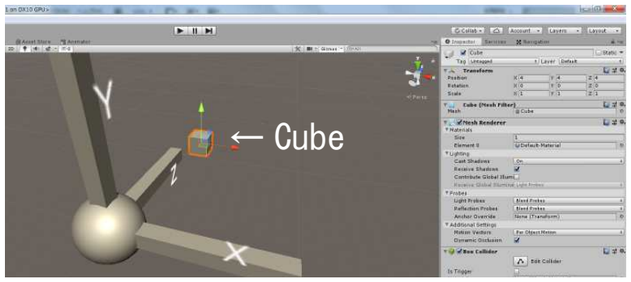


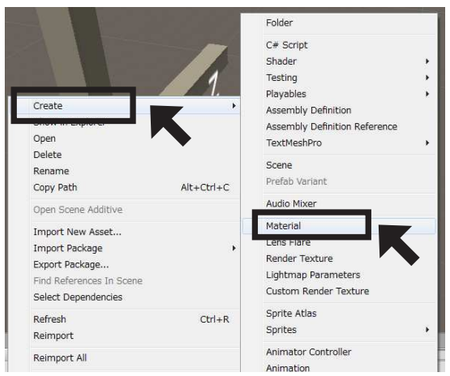
視点の回転	マウス　右ボタン　ドラッグ
視点の平行移動	マウス　ホイール　ドラッグ
視点のズーム	マウス　ホイール　回転
真正面などの視点	シーンギズモの見たい軸をクリック
投影（透視、平行）切替	シーンギズモの中央をクリック
視点の回転をロック／解除	シーンギズモ右上の鍵マーククリック
視点をフィット	ヒエラルキーでオブジェクトをダブルクリック
オブジェクト中心に回転	Alt + 左ドラッグ

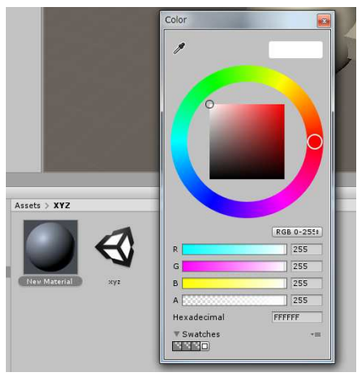


③基本的なオブジェクトを追加し、３Ｄ座標などの理解を深めます。ヒエラルキーの「Create」→「3D Object」→「Cube」と選択します。

④インスペクターの「Transform」でオブジェクトの座標やスケールを変更してみます。

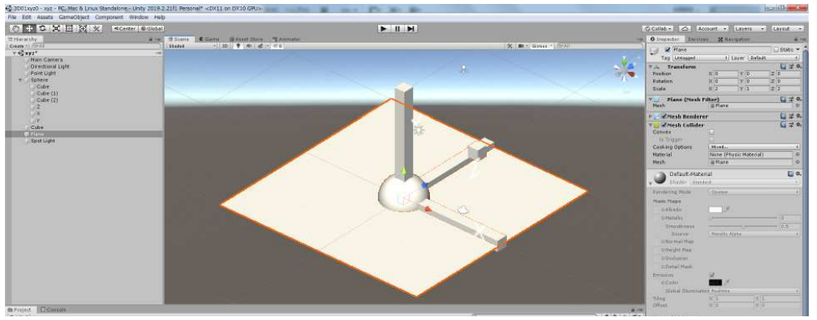


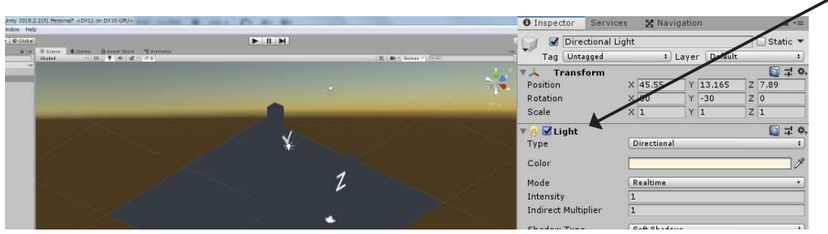


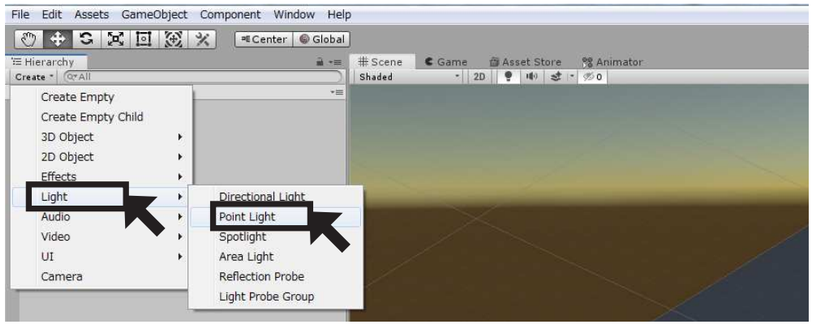


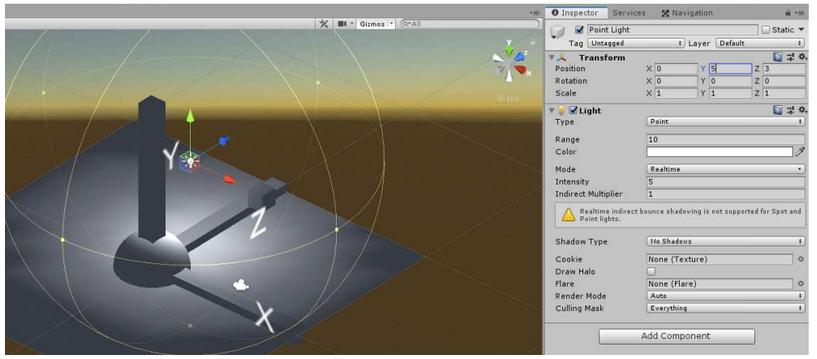
⑤オブジェクトに色を付けるには、プロジェクトウインドで右クッリクメニューから「Create」←「Material」を選択し、色などを設定します。

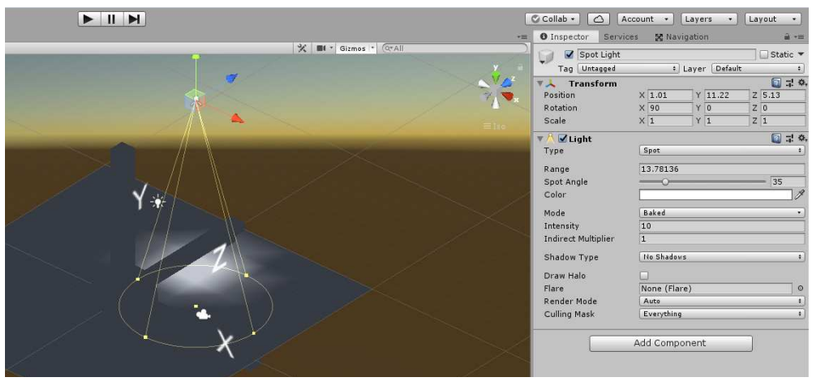
⑥設定したマテリアルをオブジェクトにアタッチします。











⑦平面となる「plane」オブジェクトを追加します。

⑧光源（ライト）の設定を確認してみます。

⑨ヒエラルキーウインドのの「Directional Light」を選択し、インスペクターの「Light」のチェックをON/OFF してみます。

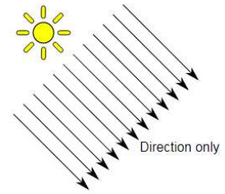
⑩新しいライトを追加してみます。「Create」から「Light」→「Point Light」と選択し、インスペクターの値を変更してみます。

⑪「Spotlight」も追加して設定を変更しましょう。

ライトの種類

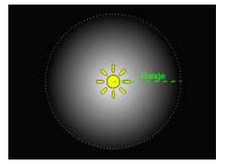
Directional Light

→「太陽」のようにゲーム画面全体を照らします。全てのゲームオブジェクトに対して均一に光を当てます。最も一般的に使用するライトです。



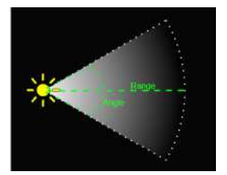
Point Light

→「電球」のようにライトの周囲を照らします。ライトから距離が遠いほど暗くなります。



Spotlight

→「車のヘッドライト」のように円錐状の範囲内を照らします。ライトから一定角度の範囲のみ明るくなります。

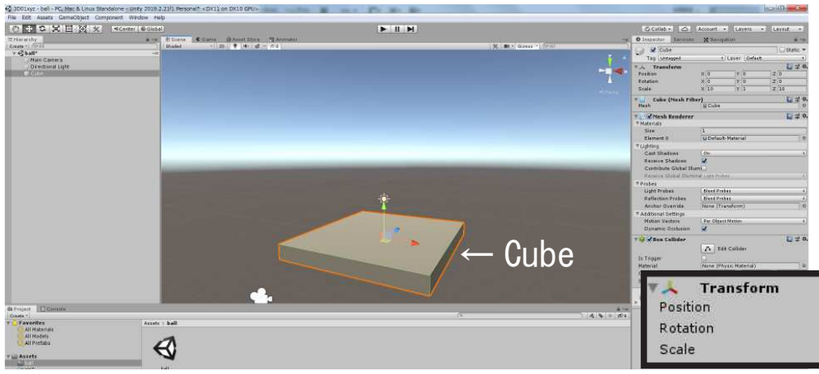


例題 34　３Ｄの基本２

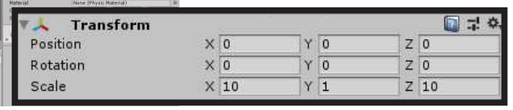
シーン名　r34ball

物理エンジンとカメラの基本について学びます。

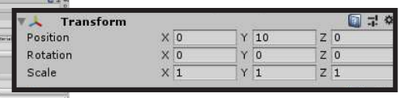
①新しくファルダを作成し、新規シーンを作成します。



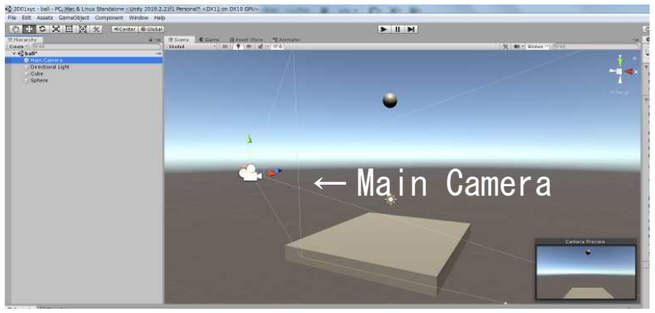
②新たな「Cube」を作り、ポジション、スケールを次のように設定します。



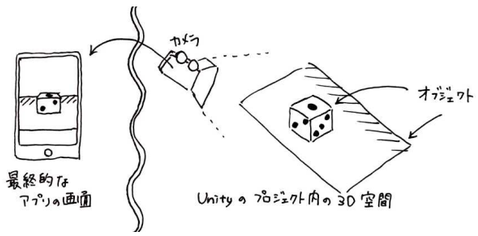
③新たな「Sphere」を作り、ポジション、スケールを次のように設定します。

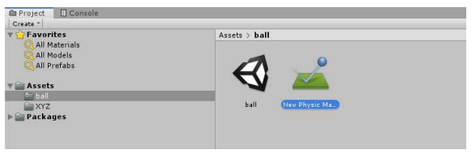


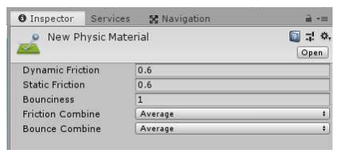
④「Add Component」から「Physics」→「Rigidbody」と選択し物理エンジンを適用します。



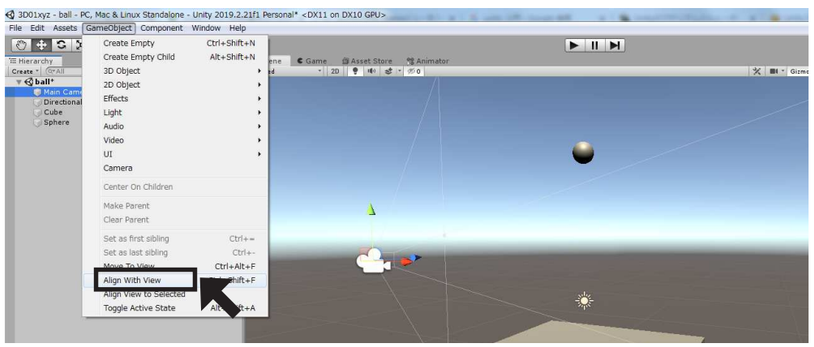
⑤ヒエラルキーの「Main Camera」を選択し位置と方向を設定し、実行してみます。







⑦プロジェクトウインドから「Create」→「Physics Material」を作成し、オブジェクト（Sphere）にアタッチし、実行します。

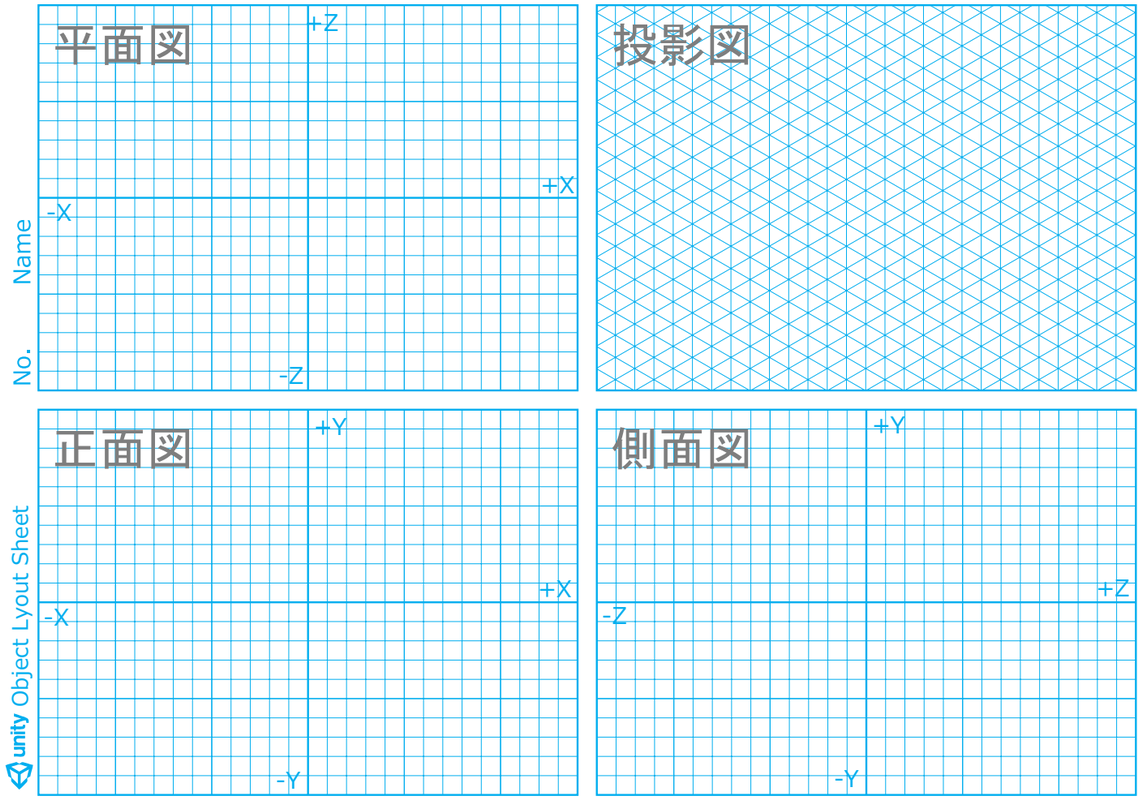


⑧オブジェクトやマテリアル、カメラなどを変更して実行してみましょう。

カメラの視点を現在、編集しているシーンに合わせるには「GameObject」メニューから「Align with View」を選択します。

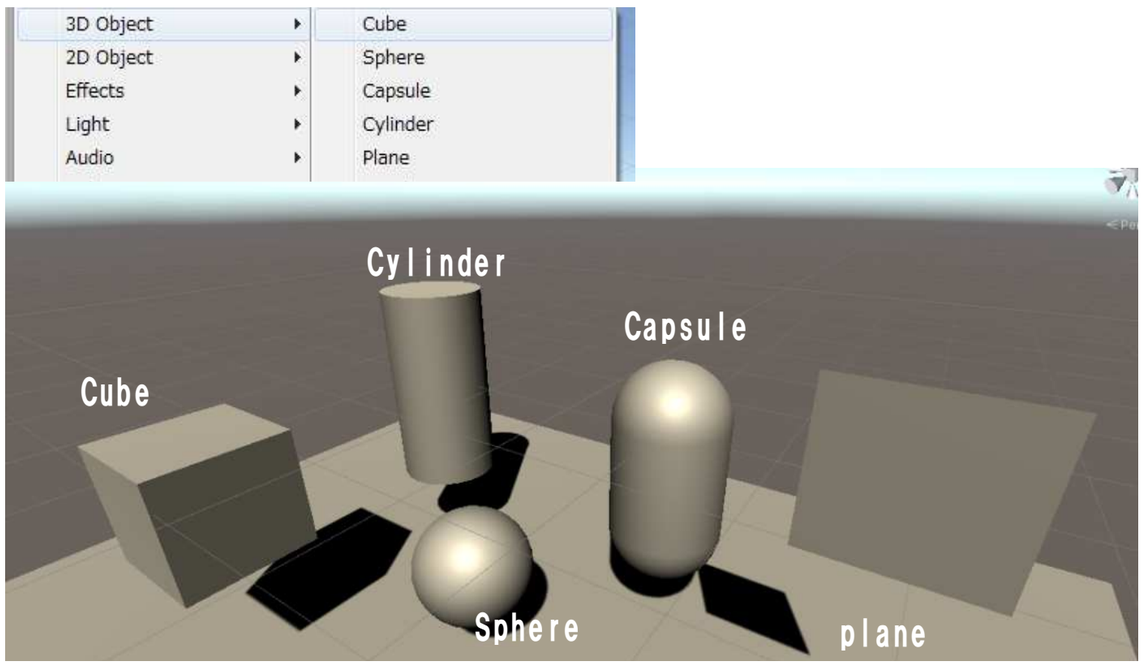
unity Object Lyout Sheet

オブジェクトの位置と大きさの設計に使用



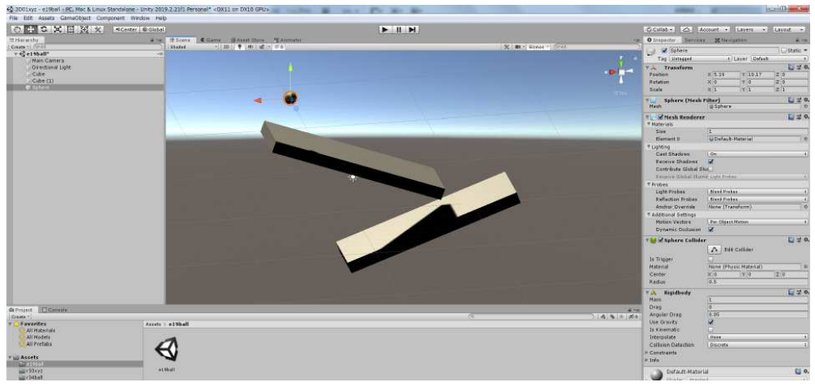
Unity(3D) のプリミティブオブジェクト

Unity（３Ｄ）には標準で何種類かのプリミティブオブジェクトが用意されています。



演習 19　ボールが板を転がるシーンを作成する。

シーン名　e19ball



①板となるオブジェクトを配置し、２０度程度傾けて配置する。Ｚ軸の位置を「０」で揃えるとよい。

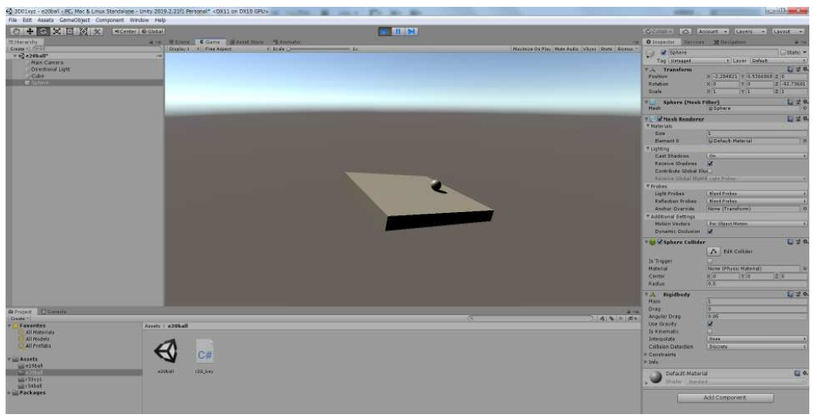
②ボールとなるオブジェクトを上方に配置し「Rigidbody」を設定する。

③カメラの位置などを設定

④実行して動作を確認する

演習 20　ボールを板に落下させ、キー入力で板の傾きをコントロールするシーンを作成する。

シーン名　e20ball



①板オブジェクトを配置、位置を (0,0,0)、スケールを (10,0,10) とする。

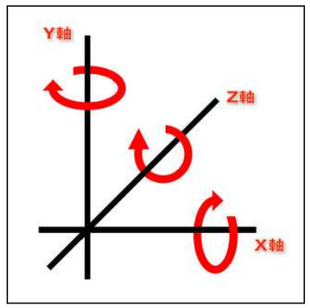
②ボールとなるオブジェクトを上方に配置し「Rigidbody」を設定する。

③スクリプトを作成し板オブジェクトにアタッチ。

④カメラの位置などを設定して実行を確認する

スクリプト：r20_key.cs
アタッチ先：板オブジェクト

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

public class r20_key : MonoBehaviour
{
    void Update()
    {
        if (Input.GetKey(KeyCode.LeftArrow))    //もし「←」が押されたら
        {
            transform.Rotate(0, 0, 1.0f);       //左回りに１度回転
        }
        if (                                    )   //もし「→」が押されたら
        {
            transform.                          );  //右回りに１度回転
        }
    }
}
```



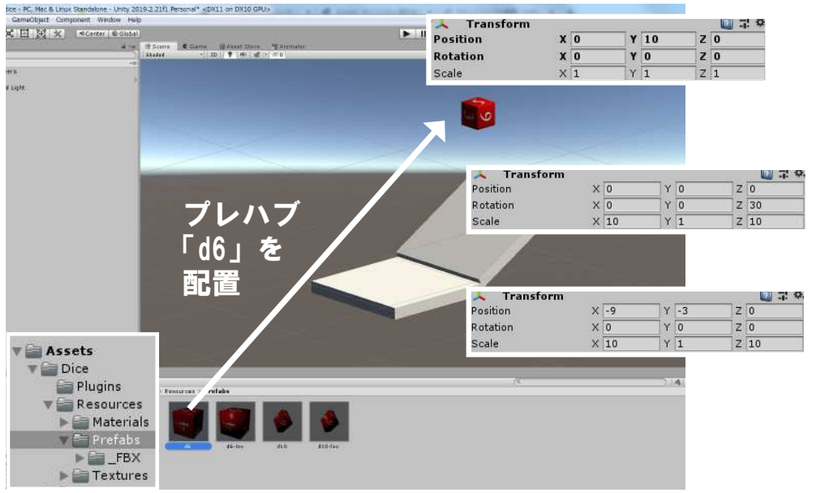
ＸＹＺの回転軸

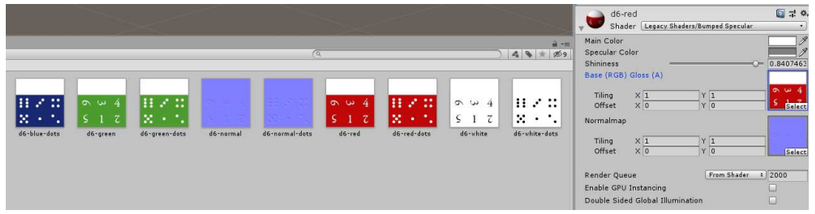
発展１　Ｚ軸回りだけでなく、Ｘ軸回りも「↑」「↓」のキー入力で回転するようにせよ。

発展２　ボールが板から落下 (y<-10) したら上に戻す (y=10) 処理を付け加えよ。

例題 35　３Ｄのアセットを使う

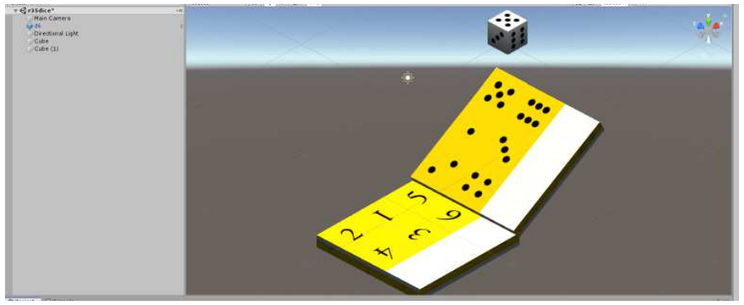
シーン名　r35dice





スクリプト：r35_dice.cs
アタッチ先：ダイスオブジェクト

```
using UnityEngine;
using System.Collections;

public class r35_dice : MonoBehaviour
{
    public int value;
    private Die_d6 die;

    void Start()
    {                                   // 最初にランダムで回転させる
        transform.rotation = Random.rotation;
        die = gameObject.GetComponent<Die_d6>();
    }

    void Update()
    {
        value = die.value;
        Debug.Log(value);       // サイコロの値をコンソールに表示
    }
}
```



サイコロ（ダイス）のアセットを使用して、３Ｄ用のアセットの使い方を学びます。「Cube」で斜面を作ってサイコロを転がしてみます。

①新しくファルダを作成し、新規シーンを作成します。

②新たな「Cube」を２つ作り、ポジションなどを図のように設定します。

③共通フォルダからサイコロの３Ｄアセット「Dice Pack Light.unitypackage」をプロジェクトにインポートします。

④ダイスのプレハブをシーンに配置し、実行します。テクスチャを変更すればダイスの図柄を変える事ができます。

このダイスのプレハブには出た面を返すスクリプト「Die_d6」がアタッチされているので、それを使用することによってサイコロの目を取得することができます。

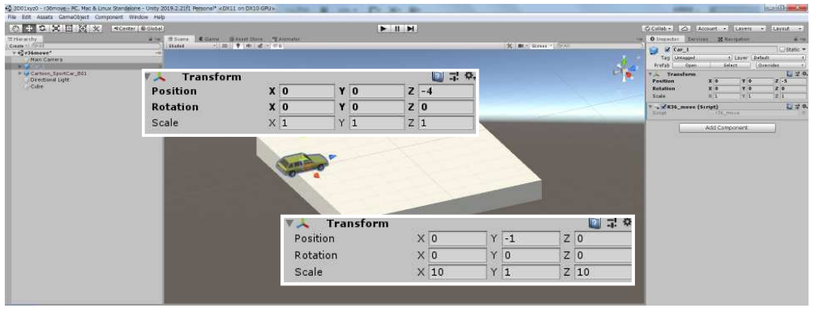
⑤スクリプトを作成しダイスオブジェクトにアタッチします。

⑥コンソールを表示させ、実行してみます。

配置した「Cube」にもテクスチャを設定することができます。アセットストアには無料のテクスチャも多数あります。

例題 36　３Ｄオブジェクトの移動

シーン名　r36move



スクリプト	r36_move.cs
アタッチ先	車オブジェクト

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

public class r36_move : MonoBehaviour
{
    Vector3 pos1, pos2;
    float az = 0.05f;

    void Update()
    {
        pos1 = this.transform.position;
        pos2 = new Vector3(pos1.x, pos1.y, pos1.z + az);
        this.GetComponent<Rigidbody>().MovePosition(pos2);
    }
}
```

スクリプト	r36_move.cs
アタッチ先	車オブジェクト

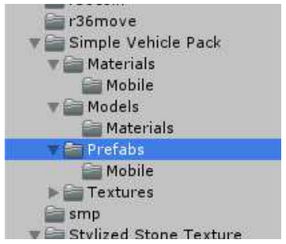
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class r36_move : MonoBehaviour
{
    Vector3 pos1, pos2;
    float az = 0.05f;

    void Update()
    {
        pos1 = this.transform.position;
        if (pos1.z >= 4.0f) az = -0.05f;  // 進む方向を変える
        if (pos1.z <= -4.0f) az = 0.05f;
        pos2 = new Vector3(pos1.x, pos1.y, pos1.z + az);
        this.GetComponent<Rigidbody>().MovePosition(pos2);
    }
}
```

３Ｄオブジェクト（剛体）(Rigidbody）の移動は、２Ｄとは少し違ってきます。

①ベースとなる Cube を配置します。

②自動車の３Ｄアセット「Simple Cars Pack」をインポートします。

③アセットフォルダから車のプレハブをシーンへ配置します。



④車のオブジェクトに「Rigidbody」をアタッチします。

更に車の子オブジェクトをまとめて「Mesh colider」をアタッチします。

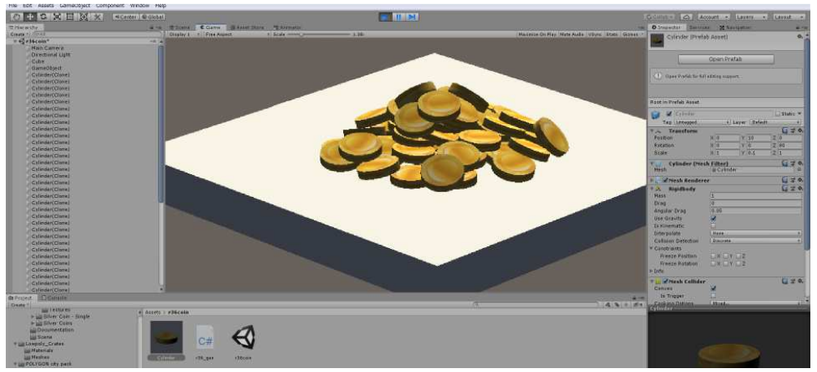
④スクリプトを記述し車のオブジェクトにアタッチ

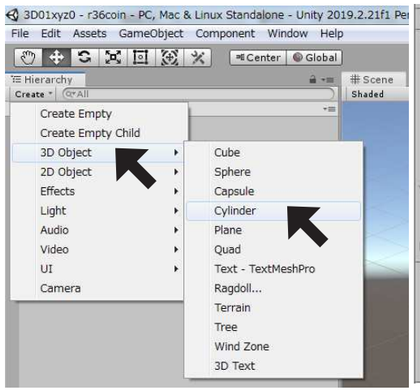
⑤車オブジェクトがＺ方向に往復移動を繰り返すようにスクリプトを変更してみましょう。

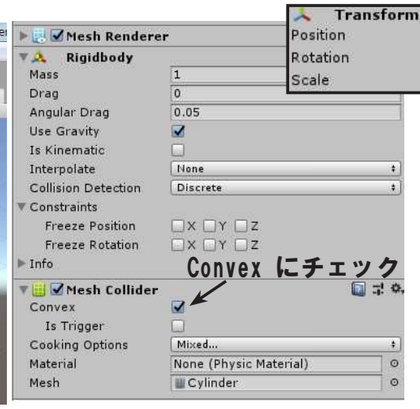
⑥カメラの位置や角度を変えて実行してみましょう。

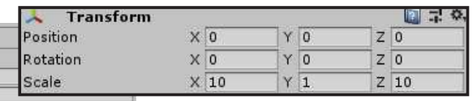
例題 37　プレハブとインスタンス

シーン名　r37coin









スクリプト	r37_coin.cs
アタッチ先	管理オブジェクト

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

public class r37_gen : MonoBehaviour
{
    public GameObject coin_p;

    void Start()
    {                                   //0.5秒ごとに「coin_fuyasu()」を実行
        InvokeRepeating("coin_fuyasu",0.5f, 0.5f);
    }

    public void coin_fuyasu()
    {                                   // プレハブ coin_p からインスタンス c を複製
        GameObject c = Instantiate( coin_p );
        c.transform.position = new Vector3( 0, 5, 0);
    }                                   // インスタンス c 位置指定
}
```

発展１　自動ではなく「スペースキー」のキー入力でコインを増やすようにしてみよう！

３Ｄでもプレハブをコピーしてインスタンスを生成できます。コイン形状のオブジェクトをコピーして増やすシーンを作成します。

①台となる「Cube」オブジェクトを作成し、カメラの位置などを調整します。

②コインとなる「Cylinder」オブジェクトを作成し「Rigidbody」と「Mesh Collider」をアタッチ、プレハブ化します。

Mesh Collider（メッシュコライダー）は メッシュのアセット からそのメッシュにもとづくコライダーを生成します。複雑なメッシュの場合、プリミティブを使用するよりもはるかに正確に衝突検出できます。Convex（凸状）が有効なっているメッシュコライダーは、他のメッシュコライダーと衝突することができます。

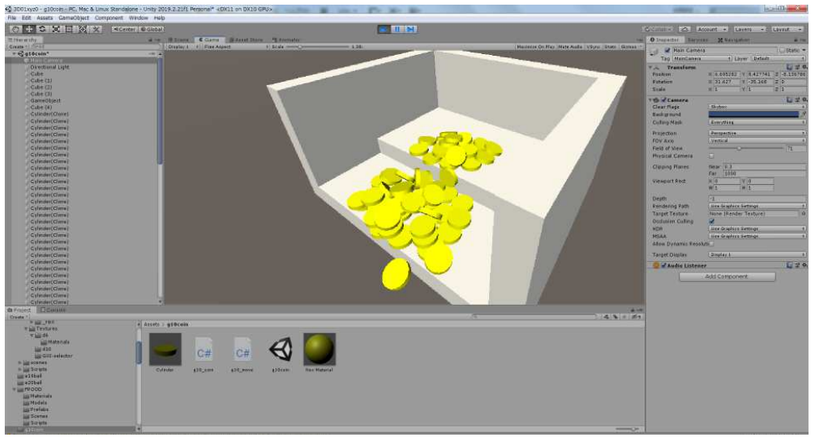
③空の管理オブジェクトを作成します。

④スクリプトを作成し、管理オブジェクトにアタッチ、外部変数にプレハブを設定します。

⑤実行を確認します。

ミニゲーム 10 コインプッシャー

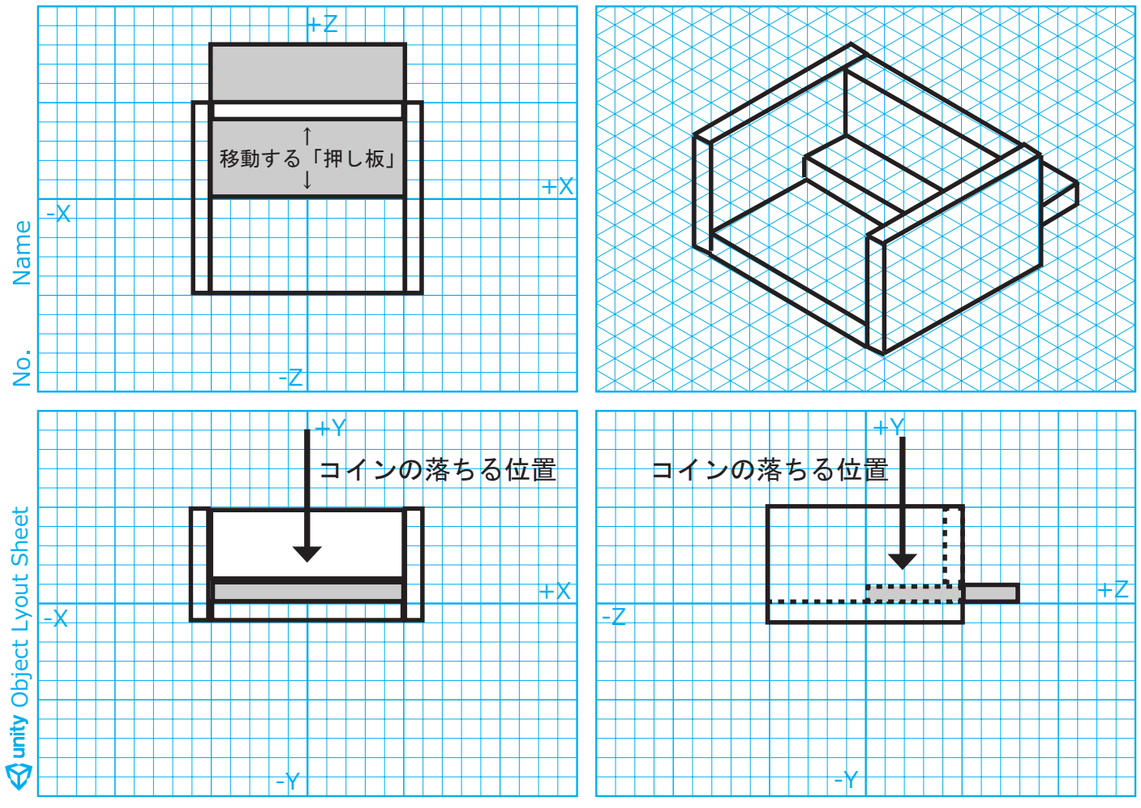
コインプッシャーというメダルゲームを例題 36,37 を参考に作成します。

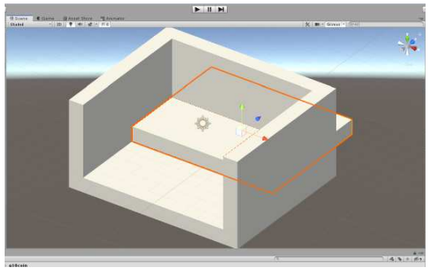


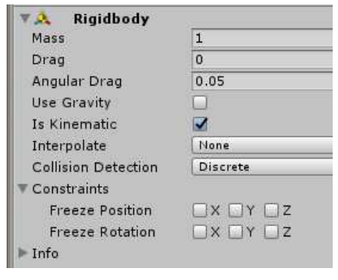
シーン名　g10coin

メダル落とし又はコイン落としとも呼ばれ、メダルゲームの一種として扱われるエレメカである。メダルをマシンの中に投入すると、内部にある押し板によりマシンの中に入っているメダルが押し出されて、当たり口の中に落ちたメダルを獲得する事が出来るものである。

①オブジェクトレイアウトシートで配置と大きさを設計し、Cube を使って配置します。







②「押し板」のオブジェクトに「Rigidbody」をアタッチし「Use Gravity」のチェックを外し、「Is Kinematic」にチェックを入れます。こうすると他のオブジェクトからの衝突や力を受けても影響されません。

スクリプト　g10_move.cs
アタッチ先　押し板オブジェクト

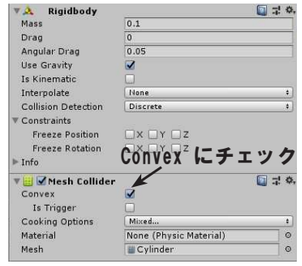
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class g10_move : MonoBehaviour
{
    Vector3 pos1, pos2;
    float az = 0.05f;

    void Update()
    {
        pos1 = this.transform.position;

        if (pos1.z >= 5.0f) az = -0.05f;
        if (pos1.z <= 2.0f) az =  0.05f;

        pos2 = new Vector3 (pos1.x, pos1.y, pos1.z + az);
        this.GetComponent<Rigidbody>().MovePosition(pos2);
    }
}
```

スクリプト　g10_coin.cs
アタッチ先　管理オブジェクト

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

public class g10_coin : MonoBehaviour
{
    public GameObject coin_p;

    public void coin_fuyasu()
    {                       //プレハブ coin_p からインスタンス c を複製
        GameObject c = Instantiate(coin_p);
        c.transform.position = new Vector3(0, 5, 2);
    }                       //インスタンス c 位置指定

    void Update()
    {               //もし「スペースキー」が押されたら
        if (Input.GetKeyDown(KeyCode.Space))
        {
            coin_fuyasu();      //コインを生成
        }
    }
}
```

③「押し板」のオブジェクトにＺ軸方向に往復運動するスクリプトを作成してアタッチします。

④「Cylinder」オブジェクトでコインの形を作成、「Rigidbody」と「Mesh colider」をアタッチし、プレハブ化します。



⑤例題３７を参考にスペースキーが押されたらコインを増やす（落とす）スクリプトを作成、空の管理オブジェクトを作ってアタッチ、実行します。

⑥マテリアルや押し板の移動速度、コインを落とす位置などを調整します。

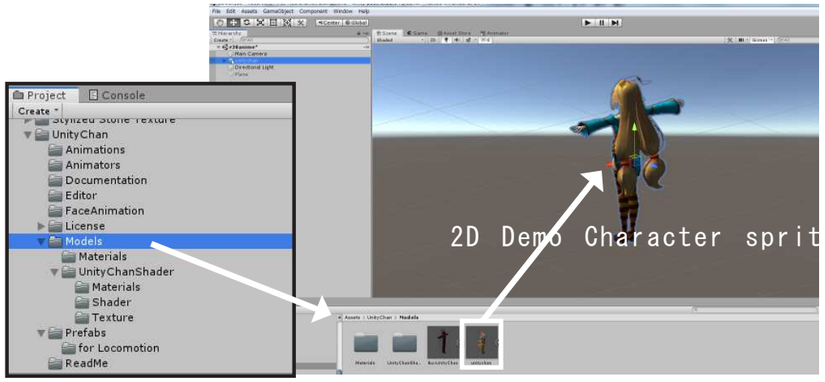
発展１

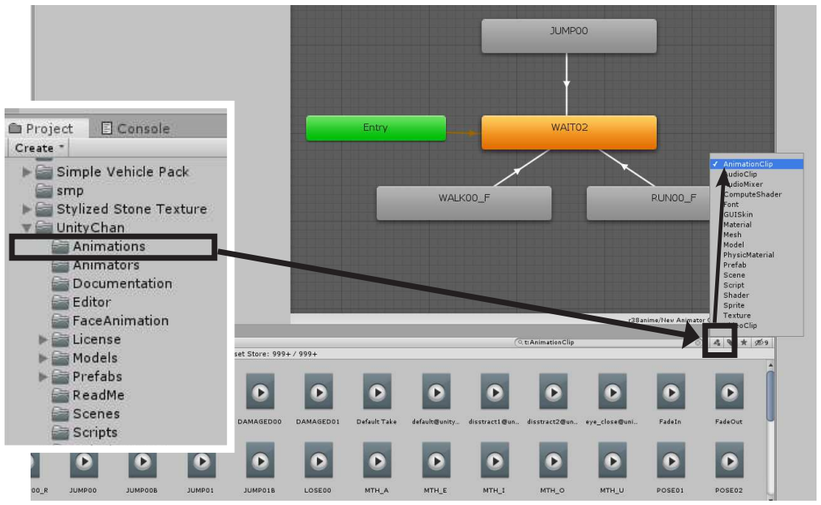
コインを古銭のアセットに変えてみよう！

発展２　「←」「→」キーでコインを落とす位置を移動するスクリプトを追加してみよう！

例題 38 アニメーション

シーン名　r38anime





スクリプト	r38_key.cs
アタッチ先	キャラクタ

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

public class r38_key : MonoBehaviour
{
    void Update()
    {                       //Animator コントローラーを取得
        var ac = GetComponent<Animator>();   /
        if (Input.GetKey(KeyCode.UpArrow))   // ↑キーなら
        {
            if (Input.GetKey(KeyCode.LeftShift)) //左SHIFTキーが押下
                ac.Play("RUN00_F");          //「Run」を再生
            else
                ac.Play("WALK00_F");         //「Walk」を再生
        }
        if (Input.GetKey(KeyCode.Space))     // スペースキーなら
        {
            ac.Play("JUMP00");               //「Jump」を再生
        }
    }
}
```

「ユニティちゃん」は Unity Technologies Japan が提供する開発者のためのオリジナルキャラクターです。
ゲームエンジン「Unity」を使っている開発者の皆様へ、キャラクターを自由に設定できるように利用規約に準じる形でアセット（素材）として無料配布しています。
ゲームで使うことのできるハイクオリティ、かつハイエンドゲームに必要とされる機能を備えつつも、ユーザーに愛される十分なキャラクター性をもったヒロイン、それがユニティちゃんです。

①３Ｄのアニメーションも２Ｄと同様に Animator と Script で制御します。

①アセットにユニティちゃん（UNITY-CHAN!）をインポートします。

②Unity ちゃんのモデルを図のフォルダから配置しカメラの正面を向くように Rotation の Y を 180° に設定します。

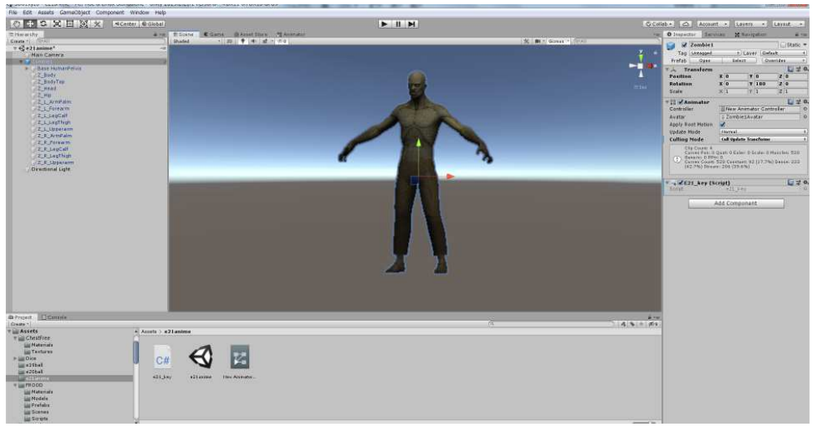
③プロジェクトで Animator を新規作成し、Animation フォルダから図のように配置し遷移を設定します。

④キー入力によってアニメーションを制御するスクリプトを作成し、キャラクタにアタッチします。

この状態ではキャラクタの位置は変わらず、その場でのアニメーションになります。

演習 21 アセットストアで入手した３Ｄアニメーションデータを制御する。

シーン名　e21anime



アセットストアにはクオリティの高い３Ｄアニメーションデータが多数あります。これらを使用すればより簡単に見栄えのするゲームの作成が出来ます。

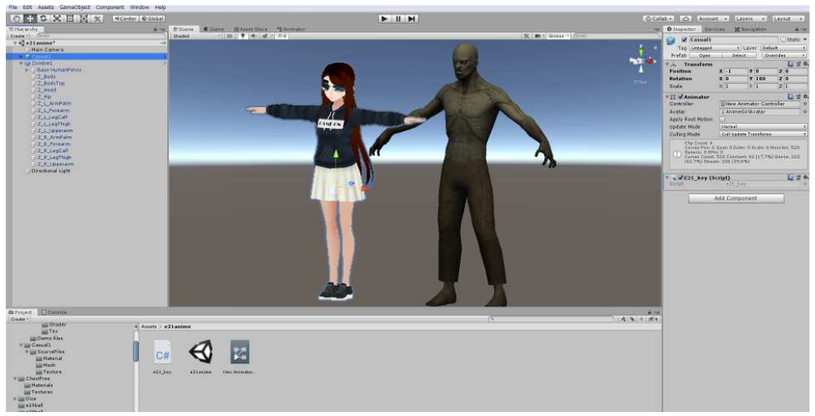
①３Ｄモデルをどれか選択し歩く、走るといくつかのアニメーションを Animator に登録します。

モデルによってアニメーションの名前が異なるので注意します。

②スクリプトを作成しキャラクタにアタッチします。

スクリプト	e21_key.cs
アタッチ先	キャラクタ

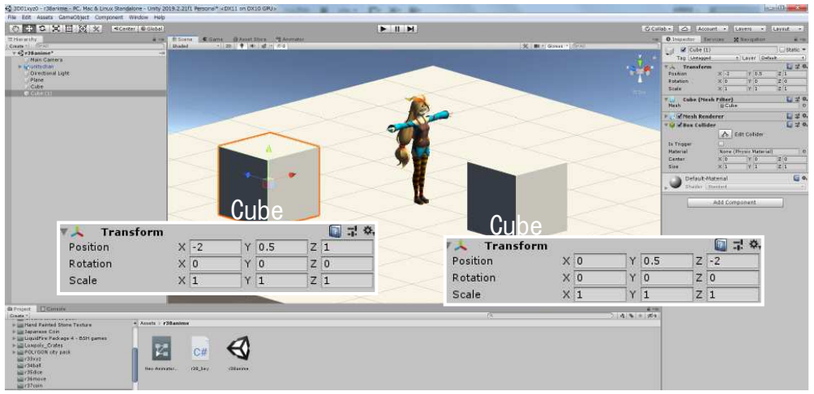
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class e21_key : MonoBehaviour
{
    void Update()
    {
        var ac = GetComponent<Animator>(); // Animator コントローラーを取得

        if (Input.GetKey(KeyCode.UpArrow))          // →キーなら
        {
            if (Input.GetKey(KeyCode.LeftShift))  //SHIFT キーが押下
            {
                ac.Play("Z_Run");                 //「Run」を再生
            }
            else
            {
                ac.Play("Z_Walk");                //「Walk」を再生
            }
        }
        if (Input.GetKey(KeyCode.Space))          // スペースキーなら
        {
            ac.Play("Z_Attack");                  //「Jump」を再生
        }
    }
}
```



いくつかのモデルを並べて、同じアニメーターとスクリプトをアタッチすると、同じ動作をさせることができます。

例題 39 ３Ｄアニメーション ＋ 移動

シーン名　r38anime

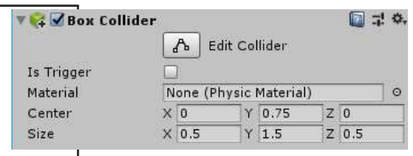


例題３８か演習２１を元に進めます。

①シーンに地面となる「Plane」を配置します。

②シーンに障害物となる「Cubu」をいくつか配置し位置を設定します。

③キャラクタに「Rigidbody」「BoxColider」をアタッチし大きさを設定します。



スクリプト	r38_key.cs
アタッチ先	キャラクタ

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

public class r38_key : MonoBehaviour
{
    void Update()
    {
        var ac = GetComponent<Animator>();  // Animator コントローラーを取得

        if (Input.GetKey(KeyCode.UpArrow))     // ↑キーなら
        {
            if (Input.GetKey(KeyCode.LeftShift))  //SHIFT キーが押下
            {
                ac.Play("RUN00_F");              //「Run」を再生
                transform.Translate(0, 0, 0.05f);  // 0.05f 移動
            }
            else
            {
                ac.Play("WALK00_F");             //「Walk」を再生
                transform.Translate(0, 0, 0.01f);  // 0.01f 移動
            }
        }
        if (Input.GetKey(KeyCode.Space))     // スペースキーなら
        {
            ac.Play("JUMP00");             //「Jump」を再生
        }
        if (Input.GetKey(KeyCode.RightArrow))    // 右キーなら
        {
            transform.Rotate(0, 5.0f, 0);       // 5 度 回転
        }
    }
}
```

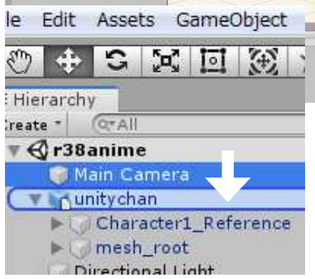
④スクリプトに位置を移動、回転するプログラムを追加し、実行を確認します。

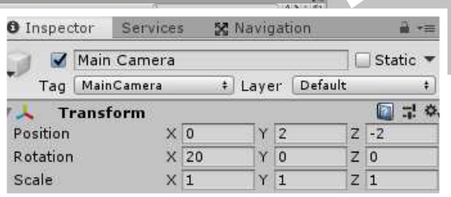
発展１　左カーソルキー「←」で左回転するスクリプトを追加してみよう！

発展２　Cube を増やしたり色を付けたり、テクスチャをはったりしてみよう！

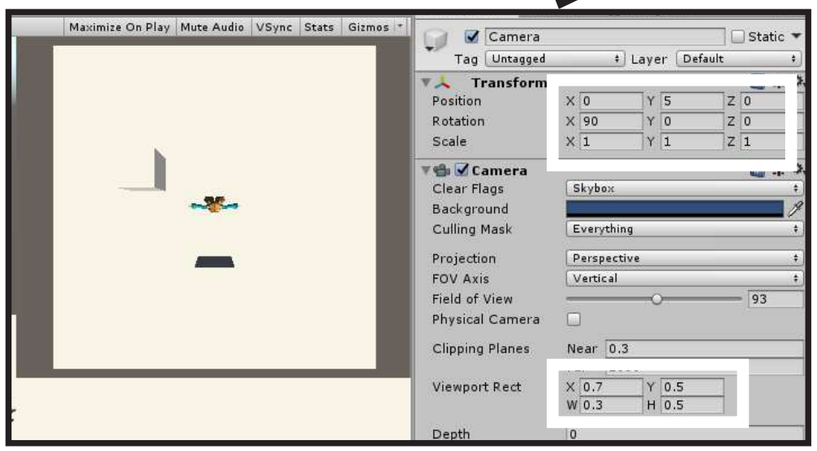
例題 40 カメラワーク

シーン名　r38anime





GameView に切り替えると設定しやすい



例題３９か演習２１を元に進めます。

カメラの位置を変えたり増やしたりすることでゲームの印象は大きく変わって来ます。キャラクタの後方にカメラを設定し、一緒に移動させてみます。

①ヒエラルキーで「MainCamera」をドラッグして「Unitychan」の上でドロップします。

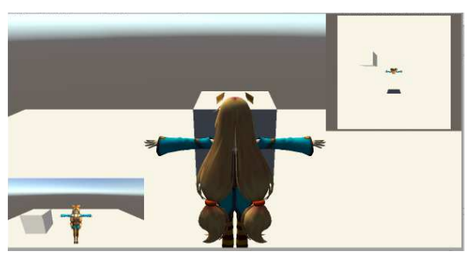
②インスペクターでメインカメラの位置と角度を設定し、実行してみます。

異なる視点からのカメラを増やしてみましょう。

③「Create」から「Camera」を選択します。

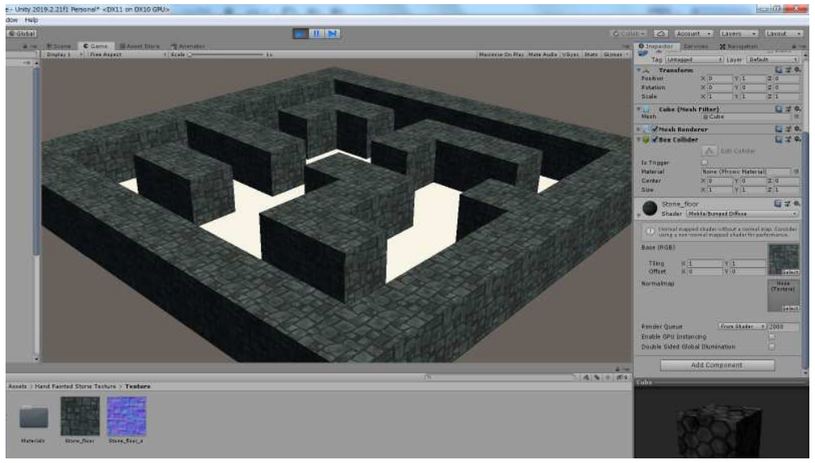
④ヒエラルキーで作成したカメラを選択し、シーンを真上から見下ろした位置に設定し、実行してみます。

発展１　カメラをもう一台追加し、キャラクタの前方から動きに追従して撮影するように設定してみよう！



例題 41　配列データから３Ｄ迷路

２次元配列に設定された迷路データを３Ｄオブジェクトで置き換えます。



シーン名　r41maze

①シーンに「Plane」を配置

②「Cube」をシーンに配置し、プレハブ化します。元の Cube は削除します。

③スクリプト作成し、空の管理オブジェクトにアタッチします。

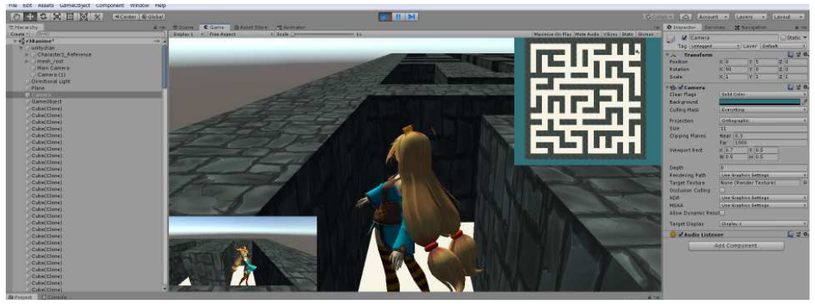
④オブジェクトに色やテクスチャを設定してみます。

スクリプト	r41_maze.cs
アタッチ先	管理オブジェクト

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class main : MonoBehaviour
{
    public GameObject kabe_p;

    int[,] maze = {  // ２次元配列の初期設定
                  {1,1,1,1,1,1,1,1,1,1},
                  {1,0,0,0,1,0,0,0,0,1},
                  {1,0,1,0,0,0,0,1,0,1},
                  {1,0,1,0,0,1,1,1,0,1},
                  {1,0,1,1,0,1,0,0,0,1},
                  {1,0,0,0,0,0,0,0,0,1},
                  {1,0,1,0,0,1,0,1,0,1},
                  {1,0,1,1,0,1,0,1,0,1},
                  {1,0,0,0,0,1,0,0,0,1},
                  {1,1,1,1,1,1,1,1,1,1},
                };       // １が壁、０が通路

    void Start()
    {
        int x, y,z;
        for (z = 0; z < 10; z++)             // 縦方向の繰り返し
        {
            for (x = 0; x < 10; x++)         // 横方向の繰り返し
            {
                if (maze[z, x] == 1)         // もし配列の値が１なら
                {
                    GameObject kb = Instantiate(kabe_p);                // プレハブをコピー
                    kb.transform.position = new Vector3( 4.5f-x , 0.5f, z-4.5f);
                }                                                       // 指定位置に配置

            }
        }
    }
}
```

岩のプレハブを Cube の代りに設定

ミニゲーム 11　３Ｄダンジョンゲーム



シーン名　r38anime

２Ｄの迷路ゲームをベースに、例題３８～４１をまとめて３Ｄ迷路ゲームを作成します。

例題４０、シーン「r38anime」をベースに作成します。

①シーンに配置された「Cube」を削除し、「Plane」のＳｃａｌｅのＸとＺを２にし、テクスチャなどを設定します。

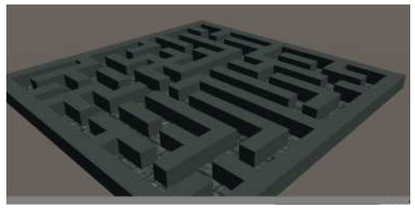
②高さを２にした「Cube」をシーンに配置し、テクスチャなどを設定してプレハブ化します。元の Cube は削除します。

③自動的に迷路を生成するスクリプトを作成して空の管理オブジェクトにアタッチします。

④キャラクタの位置を
Ｘ =8.5　Ｚ =8.5 に設定し実行を確認します。

⑤カメラの位置や角度を調整します。

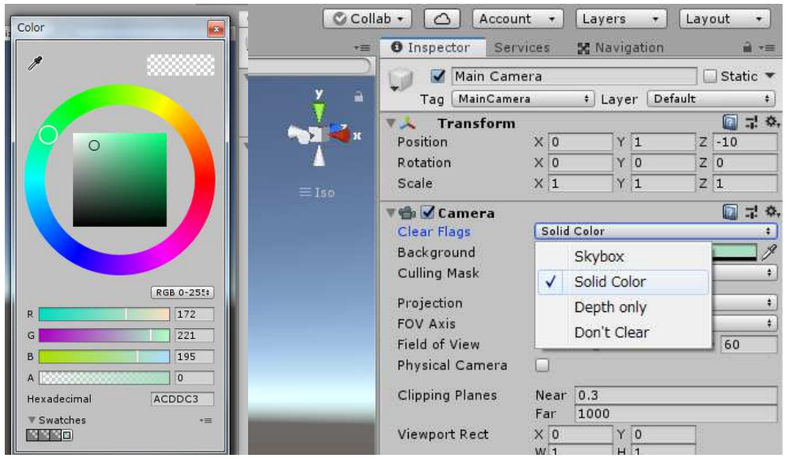
スクリプト	r41_maze.cs
アタッチ先	管理オブジェクト

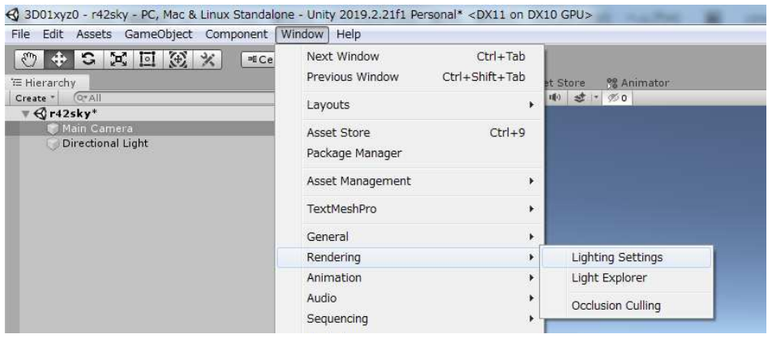


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class g11_maze : MonoBehaviour
{
    public GameObject kabe_p;
    int[,] maze = new int[100, 100];
    int W = 21, H = 21;

    void Start()
    {
        int x, y;

        for (y = 0; y < H; y++)
        {
            for (x = 0; x < W; x++)
            {
                maze[y, x] = 0;
                if (y == 0 || y == H - 1 || x == 0 || x == W - 1) maze[y, x] = 1;
            }
        }

        boutaosi();  // 迷路生成関数呼び出し

        for (y = 0; y < H; y++)
        {
            for (x = 0; x < W; x++)
            {
                if (maze[y, x] == 1)
                {
                    GameObject kb = Instantiate(kabe_p);    // インスタンスを指定位置に配置
                    kb.transform.position = new Vector3(9.5f - x, 0.5f, y - 9.5f);
                }
            }
        }
    }

    void boutaosi()     // 棒倒し法による迷路生成
    {
        int x, y, r;
        Random.InitState(System.DateTime.Now.Millisecond); // 乱数の種の初期化
        for (y = 2; y <= H - 3; y += 2)            // 縦方向の繰り返し（棒の場所）
        {
            for (x = 2; x <= W - 3; x += 2)        // 横方向の繰り返し（棒の場所）
            {
                maze[y, x] = 1;                    // 棒の場所は「１」
                r = Random.Range(0, 4);            //0 ～ 3 の乱数
                switch (r)                         // 乱数の値で分岐（switch ～ case 文）
                {
                    case 0: maze[y - 1, x] = 1; break;     // もし ０ なら 上に倒す
                    case 1: maze[y + 1, x] = 1; break;     //     １     下 〃
                    case 2: maze[y, x - 1] = 1; break;     //     ２     左 〃
                    case 3: maze[y, x + 1] = 1; break;     //     ３     右 〃
                }
            }
        }
    }
}
```

例題 42 背景（Skybox）の変更

シーン名　r42skybox





単色に設定

背景を設定する方法を学びます。

①「Main Cmamera」を選択し、インスペクターの「Clear Flags」を「Solid Color」に設定します。

②「Background」をクリックし、色を設定します。

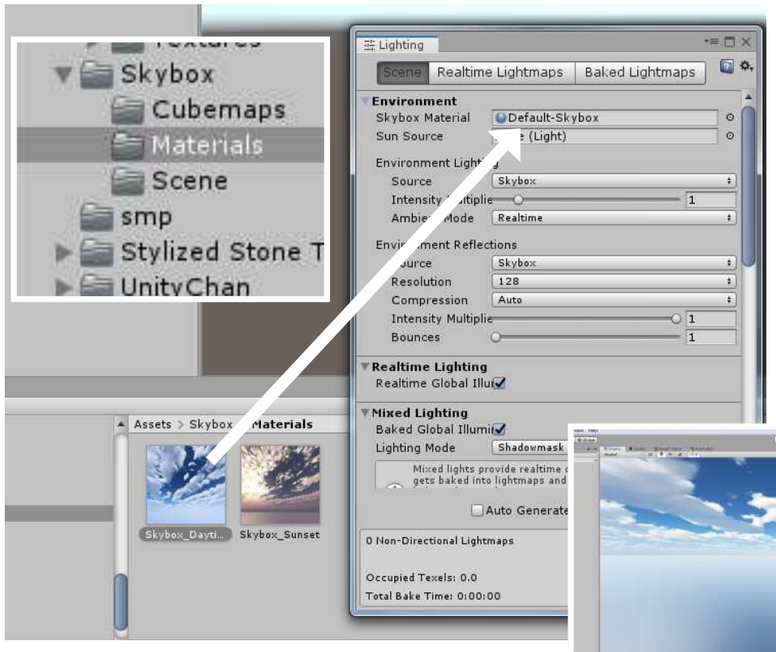
Skybox の設定

① Skybox のアセット「Free HDR Sky」プロジェクトにインポートします。

②「Window」メニューから「Rendering」→「Lighting Setting」を選択します。

③プロジェクトのアセット「Skybox/Materials」からライティングセッテイングの「Skaybox Material」へドラッグします。

④背景が変更されます。



課題 1 オリジナルゲーム１

シーン名　k01game

ミニゲーム１，２を参考に出現するキャラクタをマウスでクリックするオリジナルゲームを作成してみましょう。アイデアシートを記入し、必要なアセットを準備して始めましょう！

unity ミニゲーム　アイデアシート

ゲームタイトル　　作成者

ゲームの概要

シーンレイアウト

必要なアセットとスクリプト

アイデアシート

ゲームタイトル
印象的なタイトルを考えよう

作成者
出席番号と名前

ゲームの概要
ゲームのおおまかな内容

シーンレイアウト
画面（シーン）のデザインやオブジェクトの動きをスケッチします。

アセットとスクリプト
使用する画像やサウンドデータ、それらを動かすために必要なプログラム（スクリプト）があれば内容を書きます。

メモ
ゲーム作成に必要なメモ

※アイデアシートは提出します。

発表

プレゼン資料（PowerPoint スライド、配布資料）を作成し、発表（実演含む）、評価します。
発表時間　４分（入換含む）

※スライド、配布資料は印刷して提出します。

unity Object Lyout Sheet

No. Name