

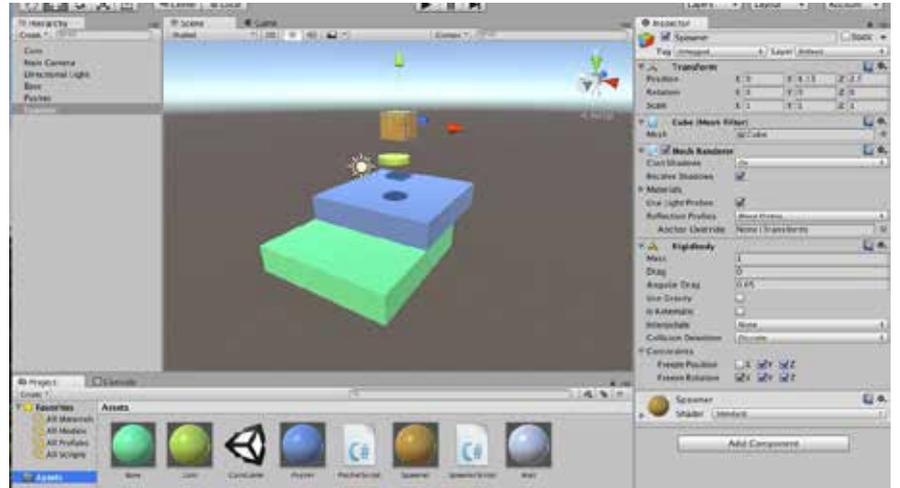


### 1 ページずつ学ぶ



Unity とは、ユニティ・テクノロジー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。

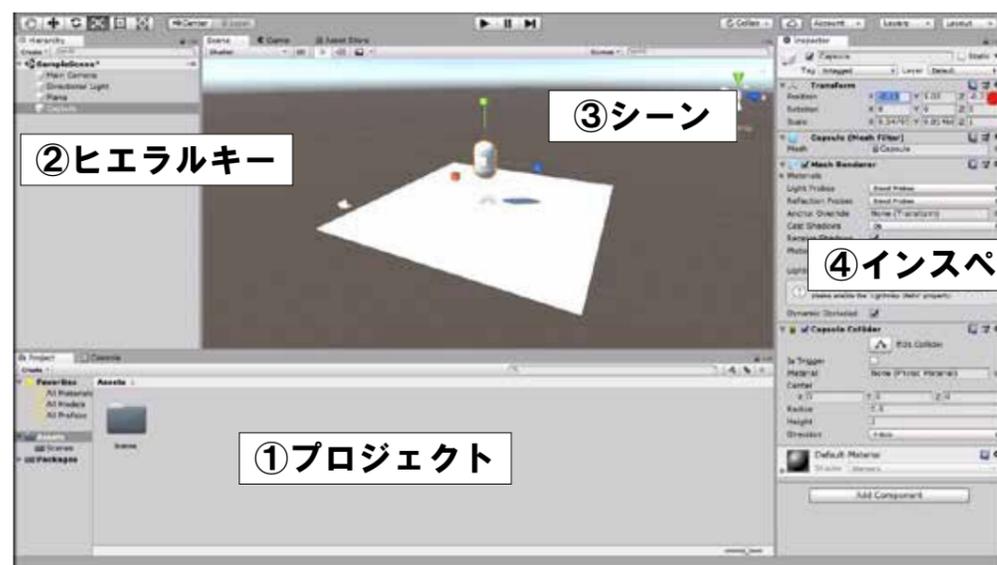
## Unity の画面構成

### ② Hierarchy (ヒエラルキー) ウィンド

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。

### ③ Scene (シーン) ビュー

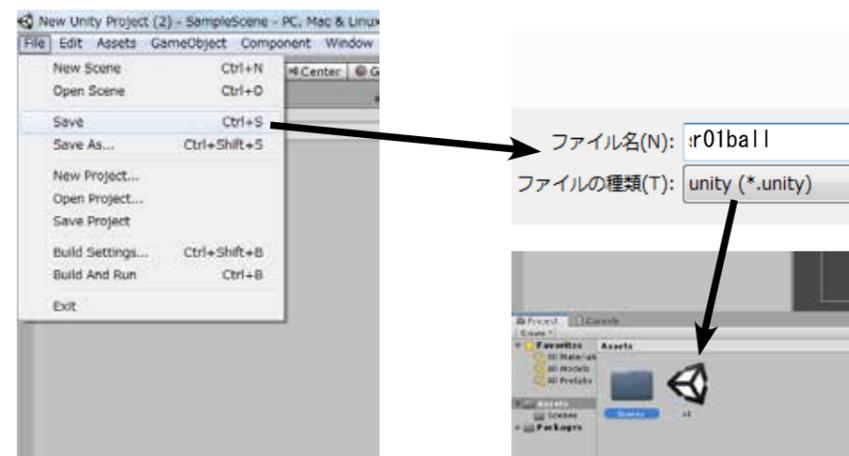
ゲーム画面に表示する画像やボタンや文字などの部品を配置します。実行時にはゲームビュー (Game) として切り替わり、プレイ時に見える画面が表示されます。



### ① Project (プロジェクト) ウィンド

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) ウィ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。

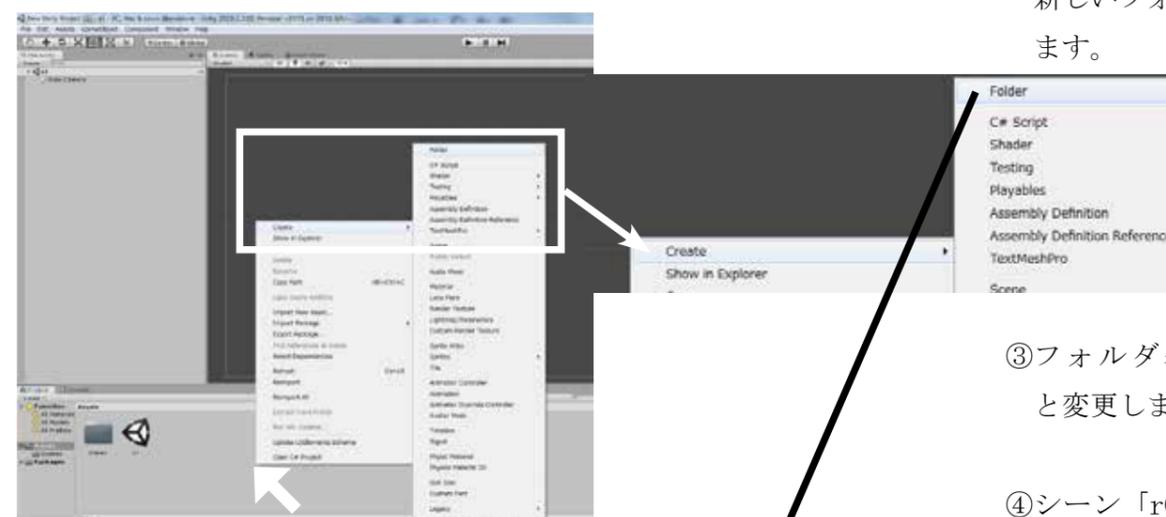
プロジェクトの作成は大掛かりで、最初の設定にも時間がかかります。この冊子の例題、演習はとても小さなプロジェクトなので、そのたびに新たなプロジェクトを作成していると非効率です。そこで今回は、1つ1つの例題、演習をひとつの「シーン」として作成し、学習を進めていきます。それぞれに使用するアセット (素材) はシーンごとにフォルダを作って管理していきます。



シーン名 r01ball

① 「File」→「Save」  
(あるいは「Save As」)  
ここではファイル名を「r01ball」とします。

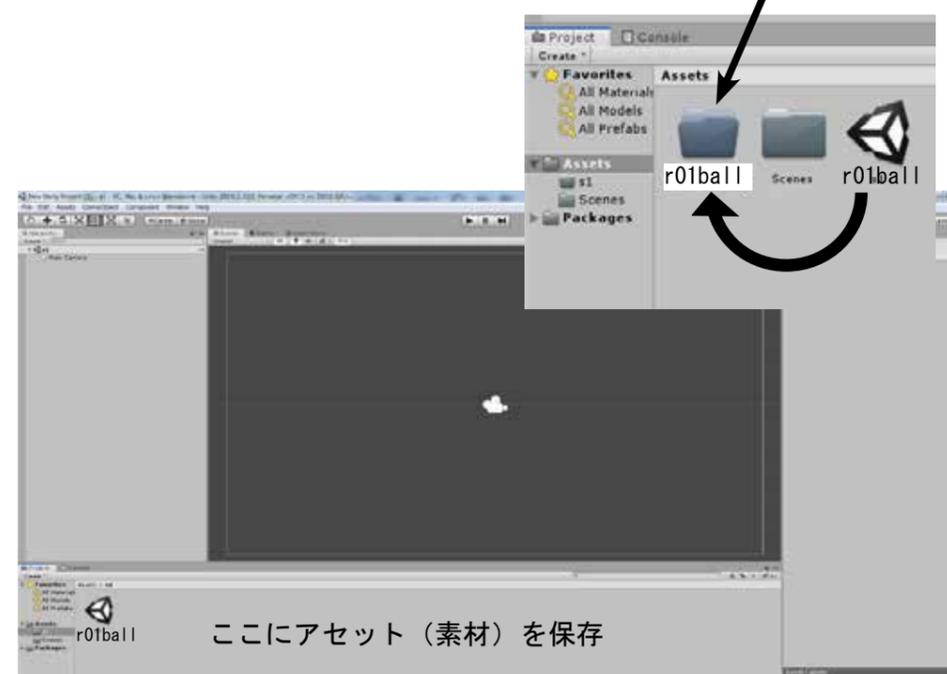
② プロジェクトウインドで右クリックし  
「Create」→「Folder」で新しいフォルダを作成します。



右クリック

③ フォルダ名を「r01ball」と変更します。

④ シーン「r01ball」を作成したフォルダへドラッグ & ドロップして移動します。

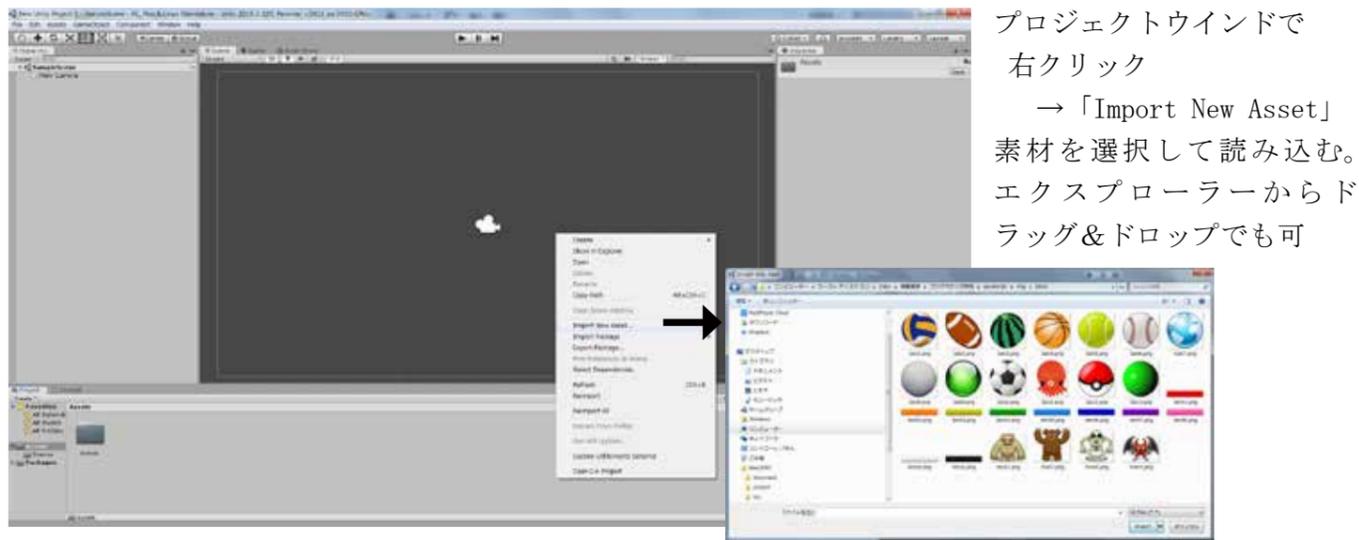


ここにアセット (素材) を保存

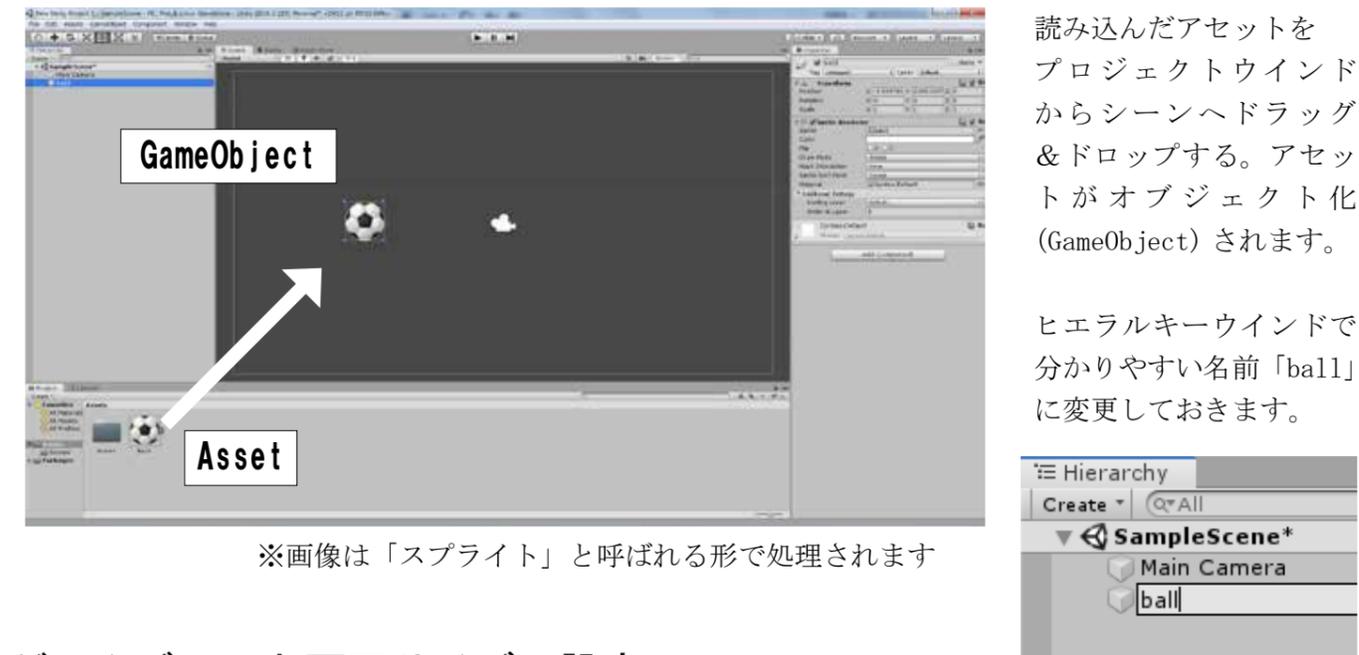
⑤ フォルダ「r01ball」を開きます。使用するアセット (素材) はここへ保存します。

※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。

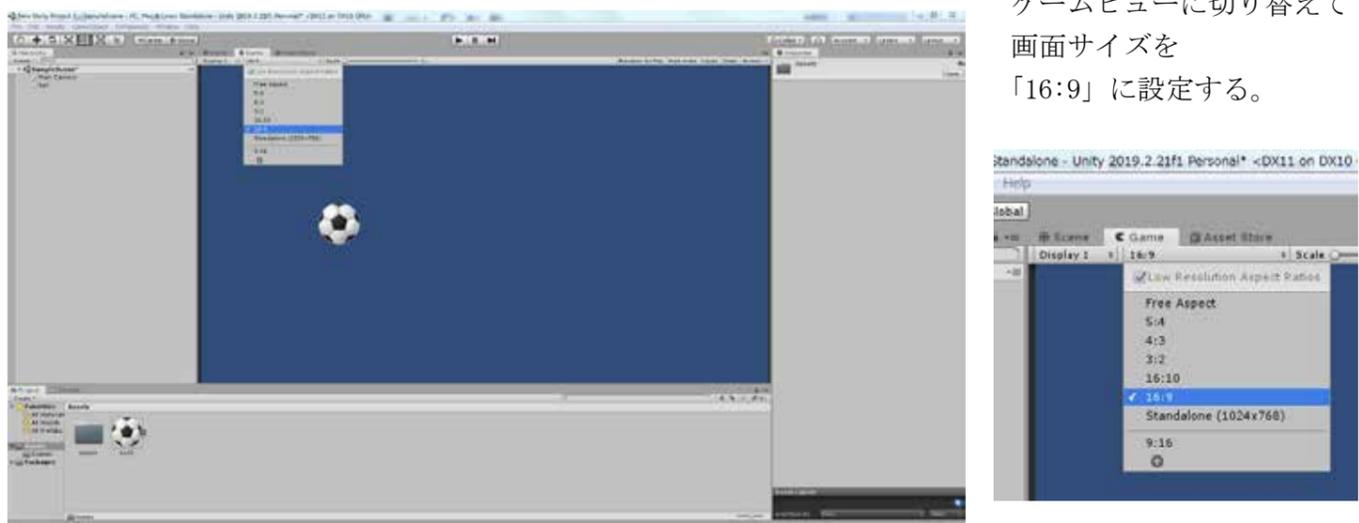
## プロジェクトにアセット（資産）を追加する



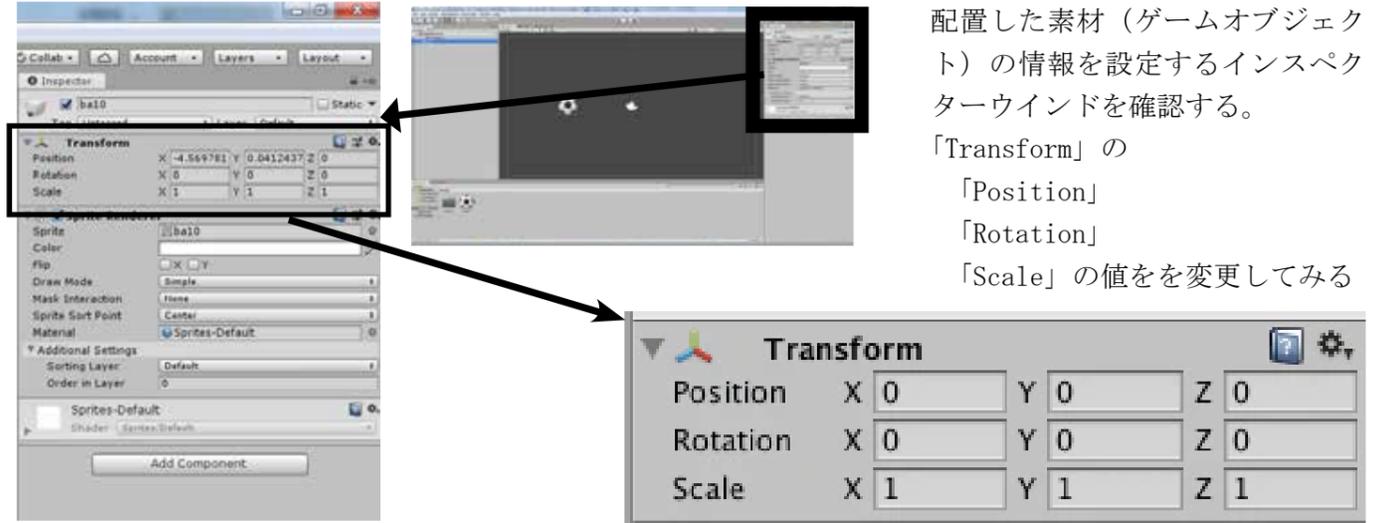
## ヒエラルキーウインド



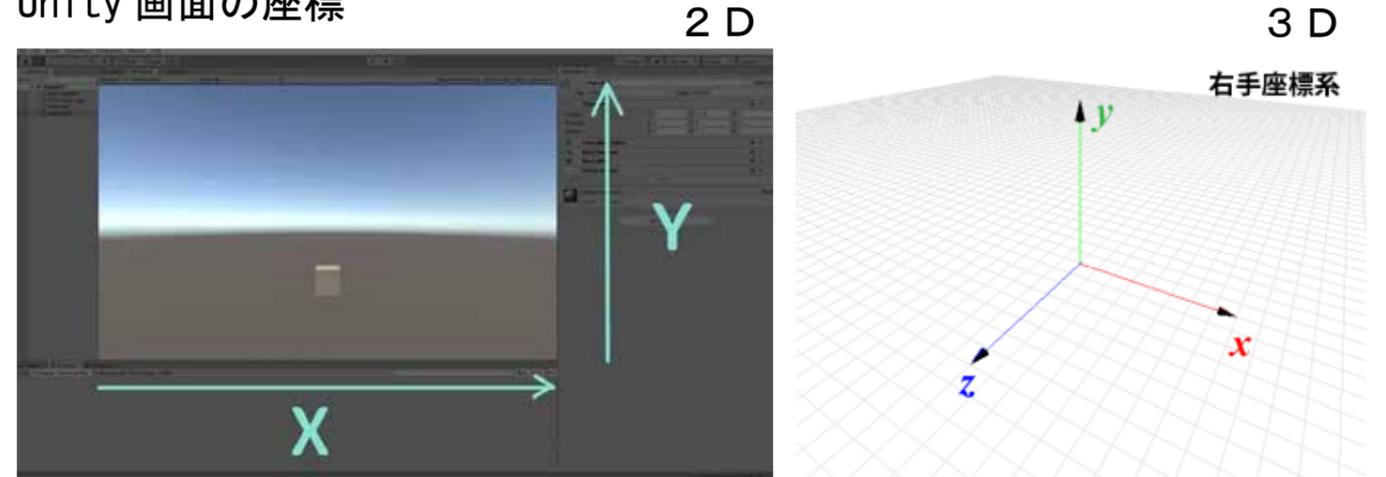
## ゲームビューと画面サイズの設定



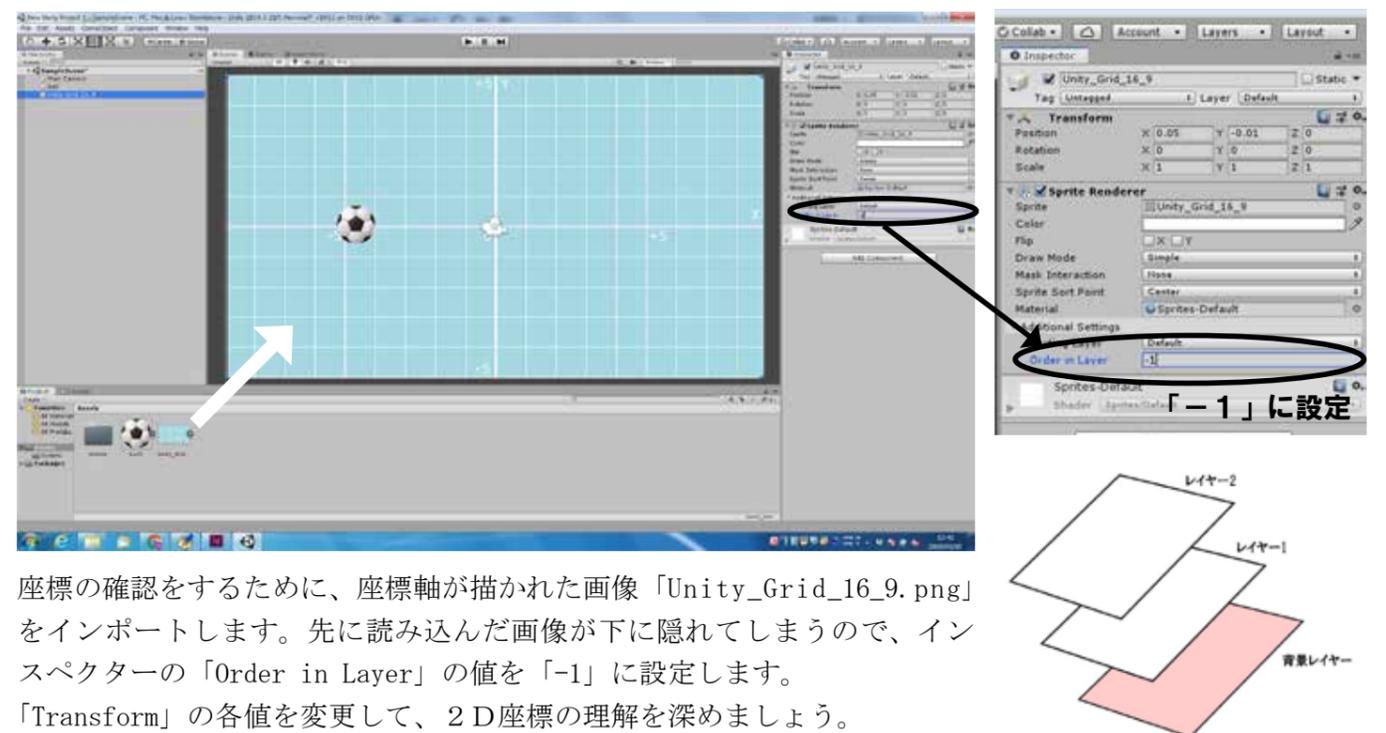
## インスペクターウインド



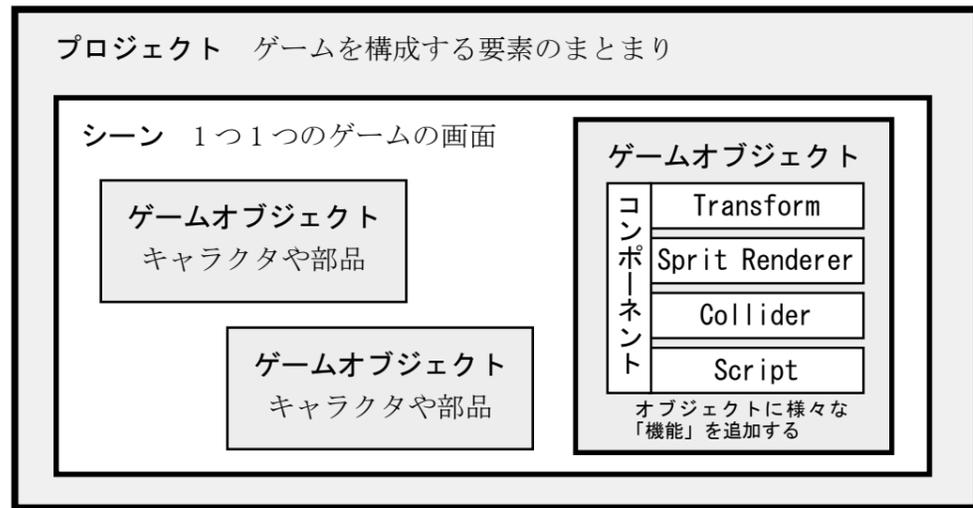
## Unity 画面の座標



## オブジェクト (GameObject) の追加と座標等の確認



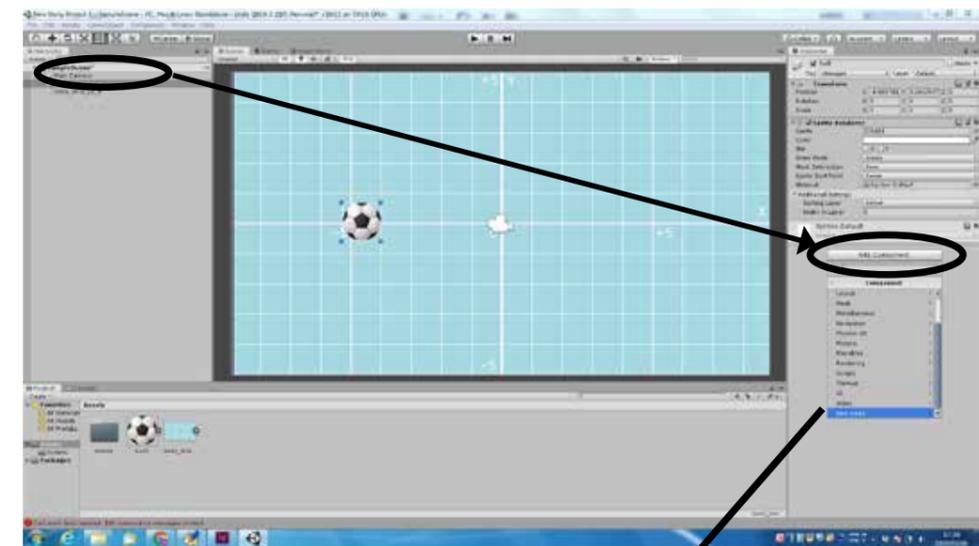
# Unity の構成要素



# 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

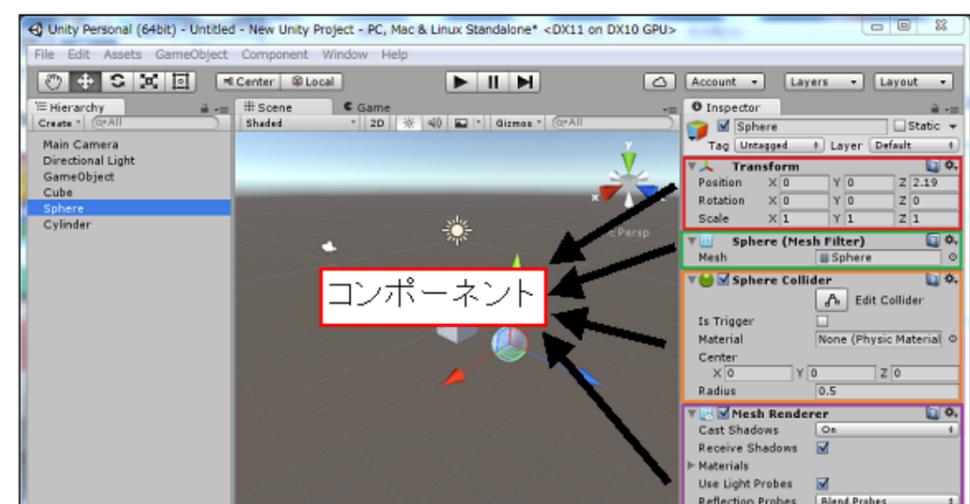
# スクリプトの追加



スクリプトはオブジェクトなどを動作させるためのプログラムです。Unityではオブジェクトに新しい機能を追加する「コンポーネント」のひとつとして設定します。

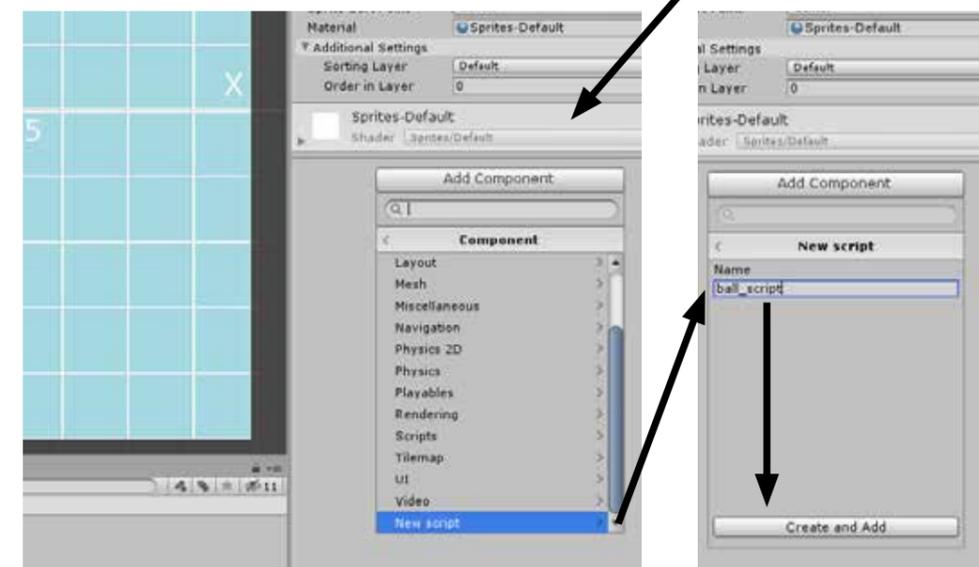
オブジェクト「ball」を選択し、インスペクターウインドの最下部の「Add Component」ボタンをクリックします。

# 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。

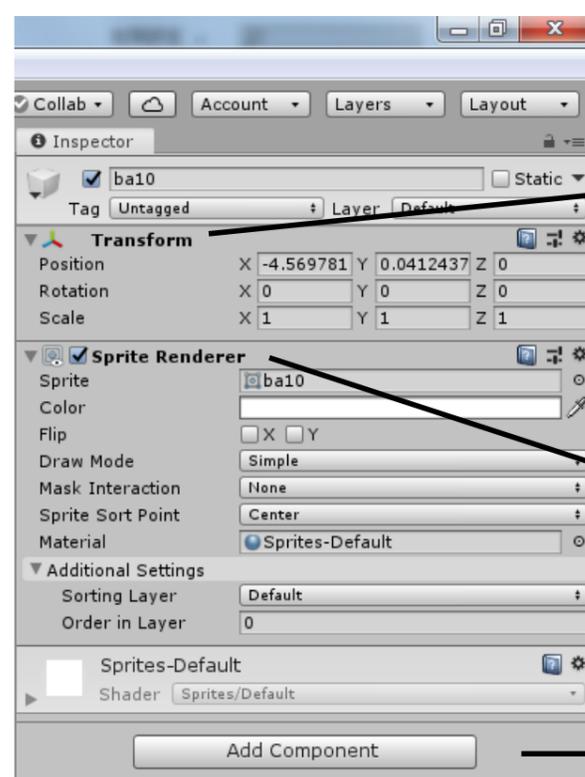


コンポーネント一覧最下部の

「New Script」をクリックし

「r01\_ball」と名前を付け変えて

「Create and Add」ボタンをクリックします。



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

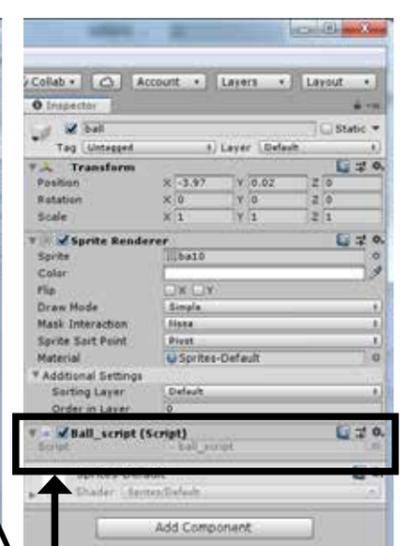
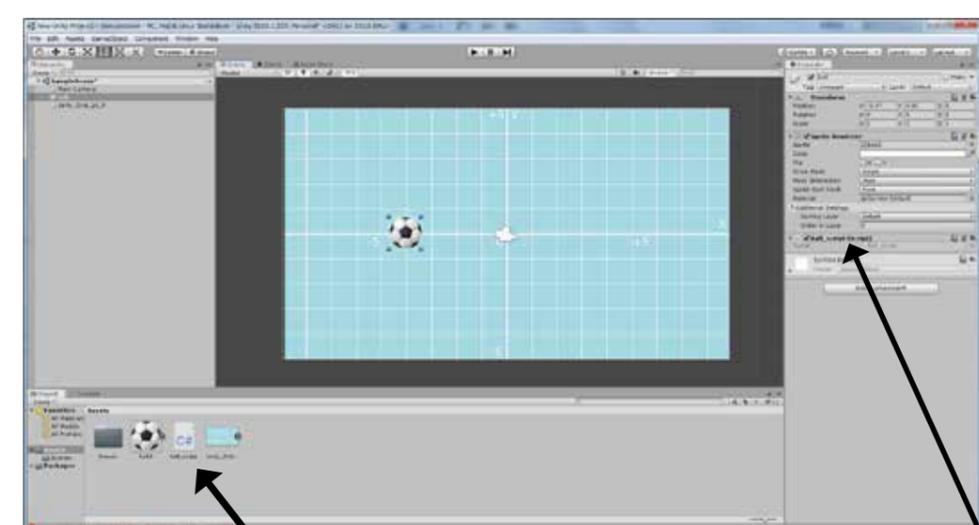
## Transform 位置・回転・スケール

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xy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

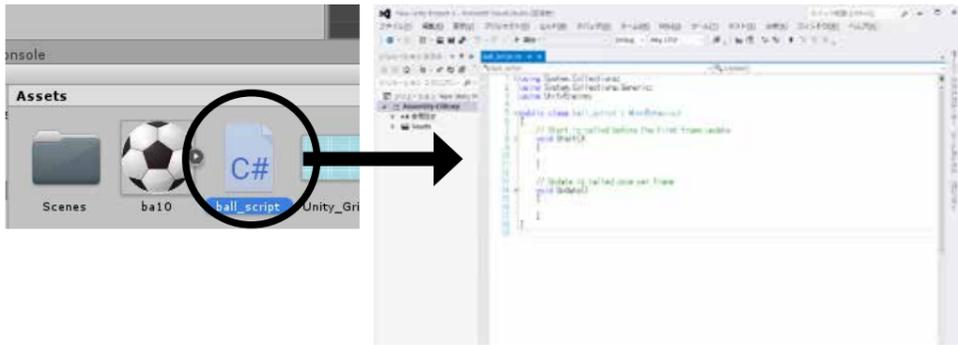
→ 新たなコンポーネントの追加



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



```
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);
    }
}
```

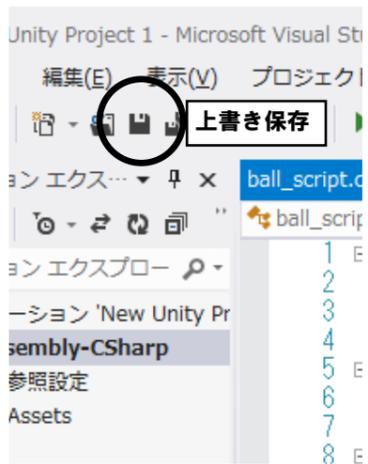
ファイル名と同じクラス名

ゲームスタート時に1回だけ実行

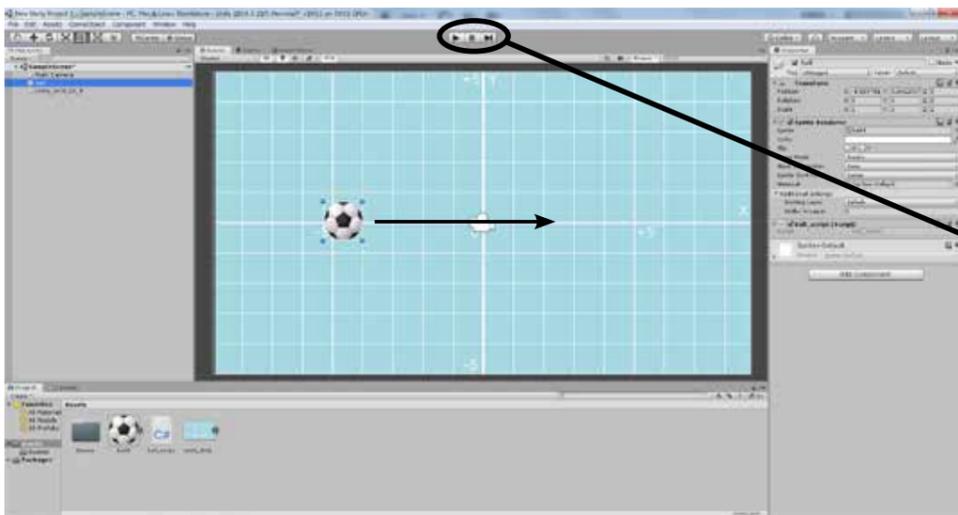
フレーム毎に実行

プロジェクトウインドのスキ  
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。

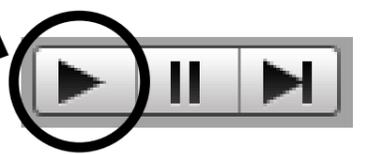
左のようにスクリプトを追  
加入力し、「上書き保存」ボ  
タンをクリックします。



## 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。

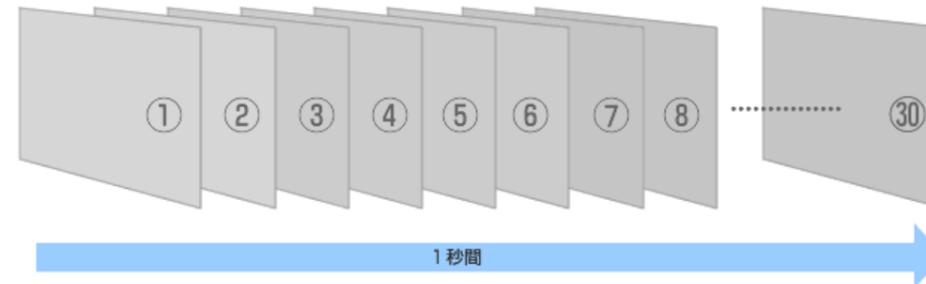
### 重要 動くオブジェクトの作り方

- 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 [30fps]

1秒間に表示できる「コマ」の数がフレームレートとなり、動画ではコマ数が多いほど、動きが滑らかになります。



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);
}
```

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

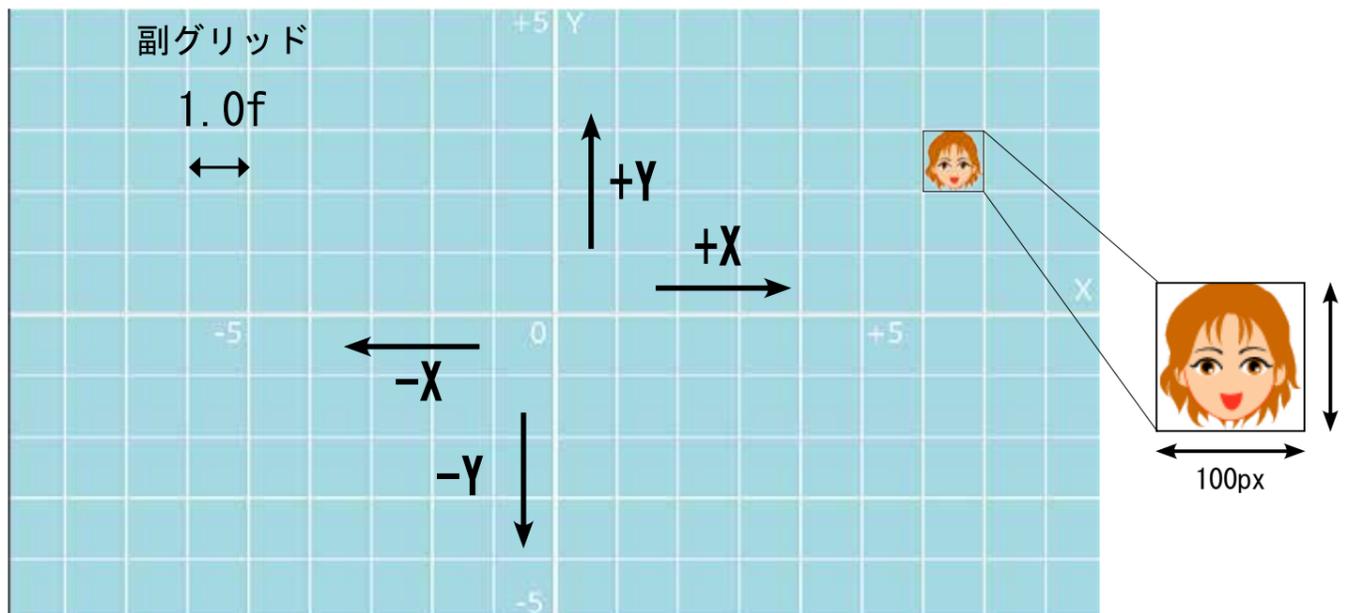
クラスはオブジェクトの設  
計図のようなもので、プロ  
パティはそのオブジェク  
トの様々な性質を表しま  
す。メソッドはそのプロ  
パティを変更したり、オ  
ブジェクトを操作する命  
令です。

## Unityの単位系

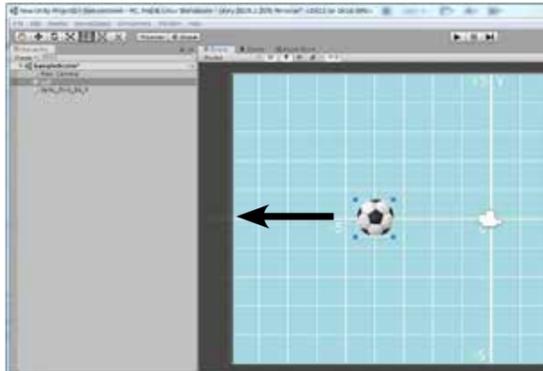
主グリッド	1m × 1m
副グリッド	10cm × 10cm
距離1.0f	10cm
重さ(Mass) 1.0f	1.0kg

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)



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。

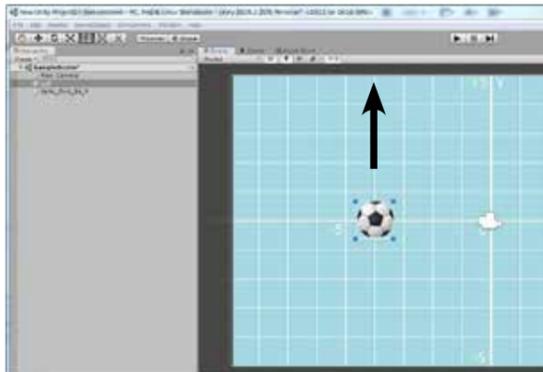


① 「左」へ移動

```

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

```

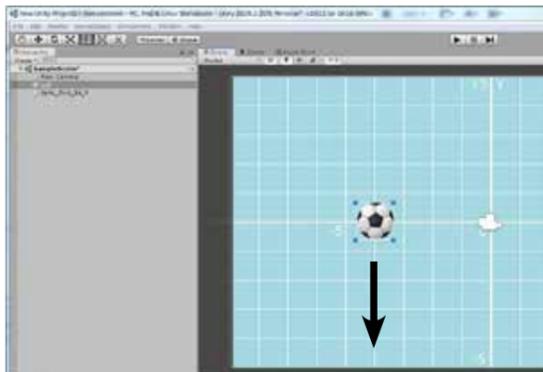


② 「上」へ移動

```

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

```

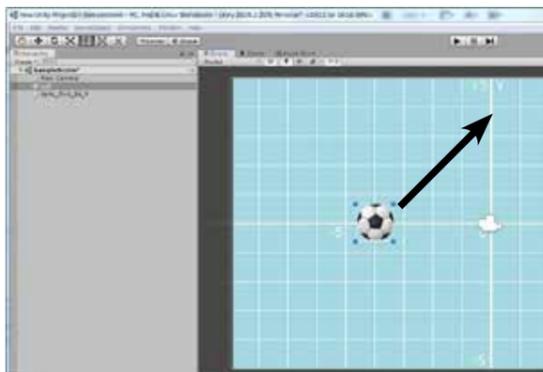


③ 「下」へ移動

```

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

```

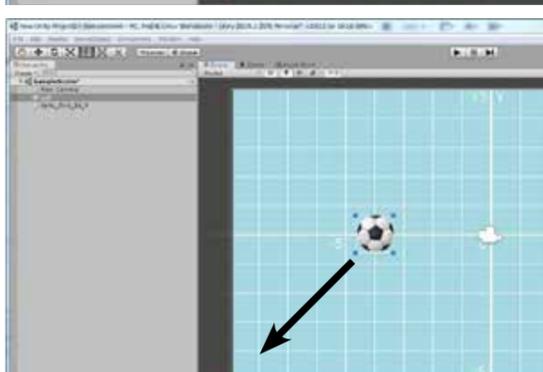


④ 「斜め左上」へ移動

```

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

```



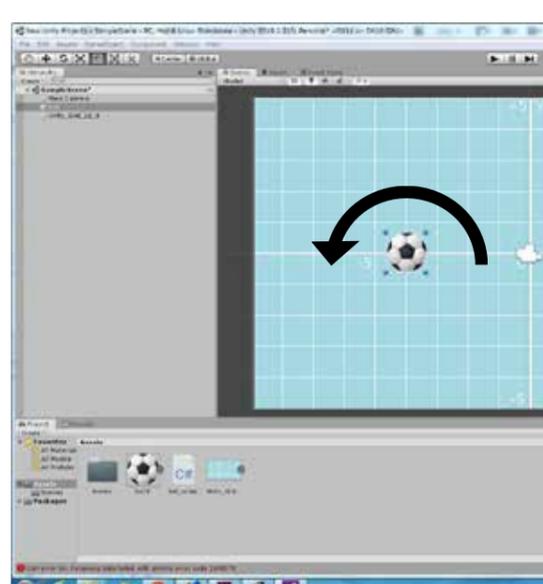
⑤ 「斜め右下」へ移動

```

...
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、逆転)



```

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

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

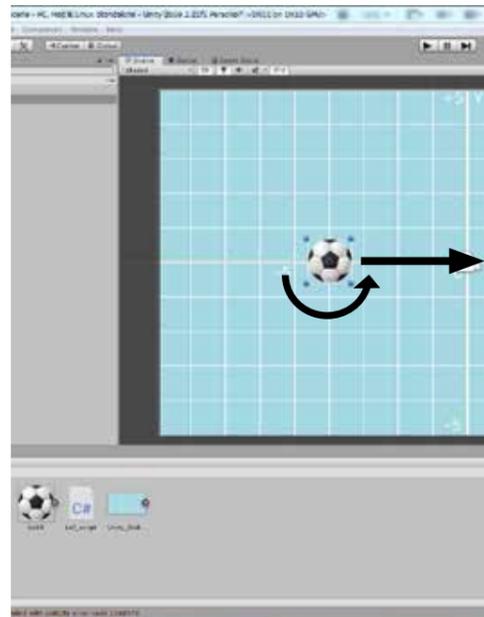
    }

    // 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;

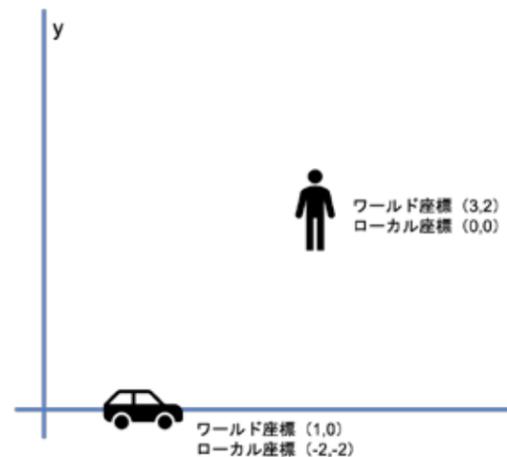
public class r01_ball : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

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

```

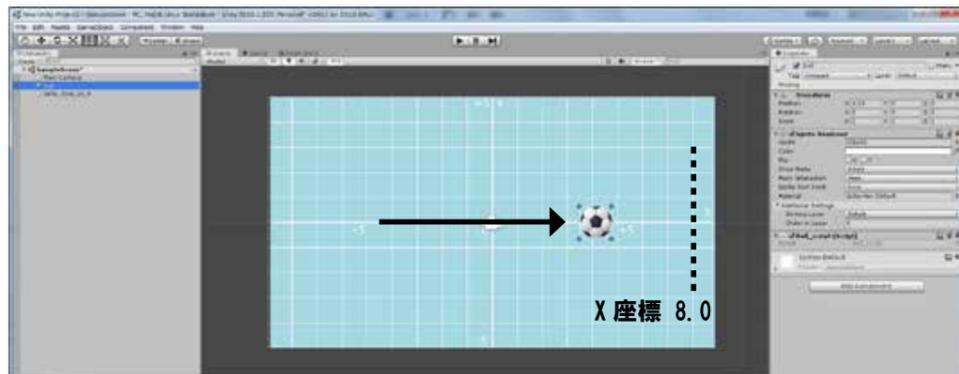
ワールド座標とローカル座標



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 ( i f )



```

...
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
    }
}

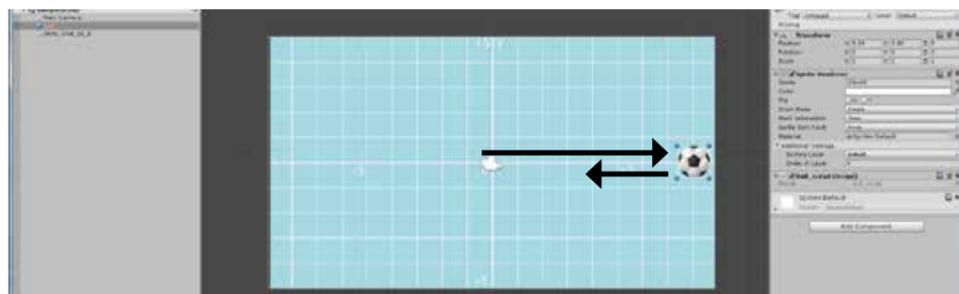
```

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。

**transform.position.x** で現在の x 座標位置を取得し x 座標が「8.0」以上になったら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



```

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
        }
    }
}

```

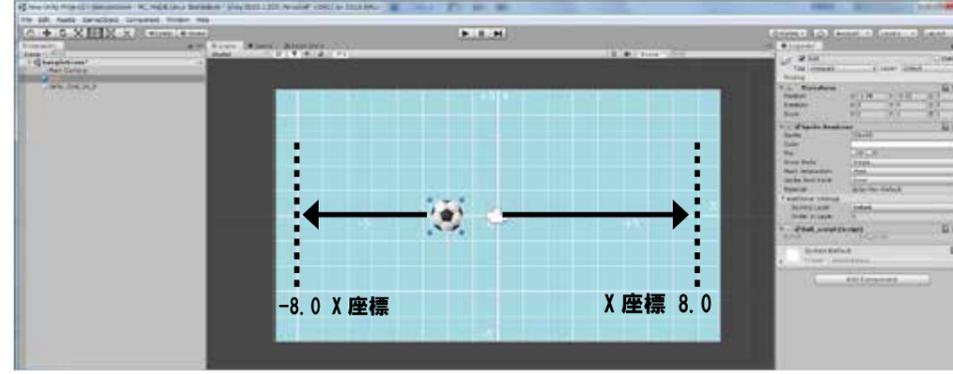
x 座標が「8.0」以上になったらところでスクリプトの実行を移動方向を変えるスクリプトを追加する。

### ○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



```

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

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。

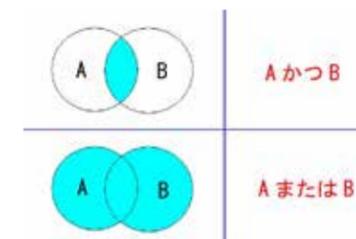
いくつかの条件式の場合  
2つ以上の条件が成り立った時に分岐させたい場合は「論理演算子」を使用して条件式を構成します。

&& かつ  
|| または

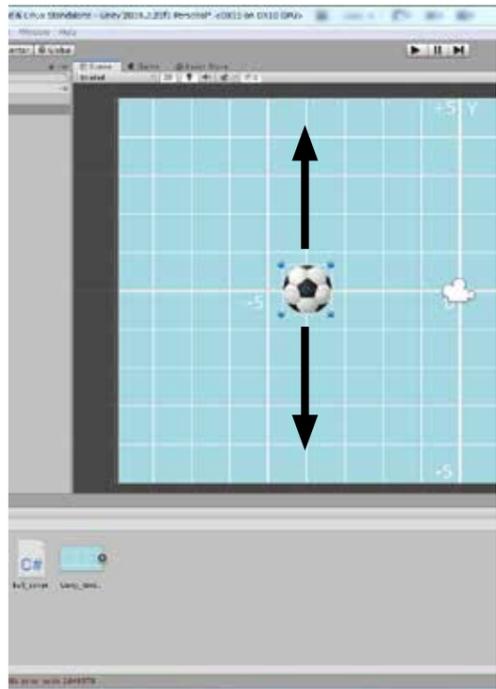
0 <= a && b < 10

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;

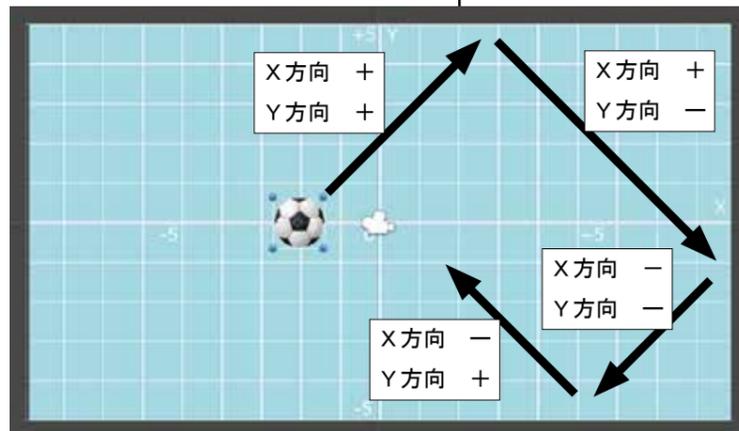
public class r01_ball : MonoBehaviour
{
    float ax = 0.1f;    X方向移動量
    float ay = 0.1f;    Y方向移動量

    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
    }
}
```

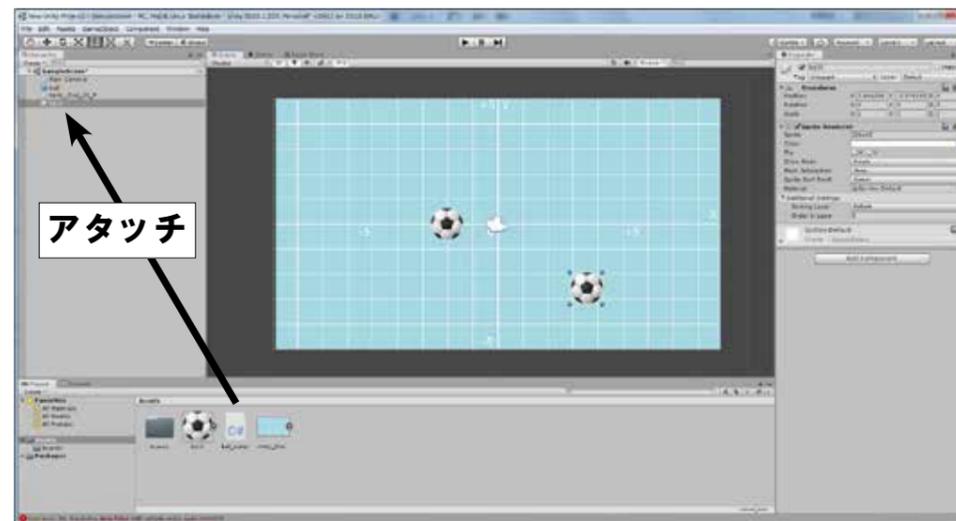
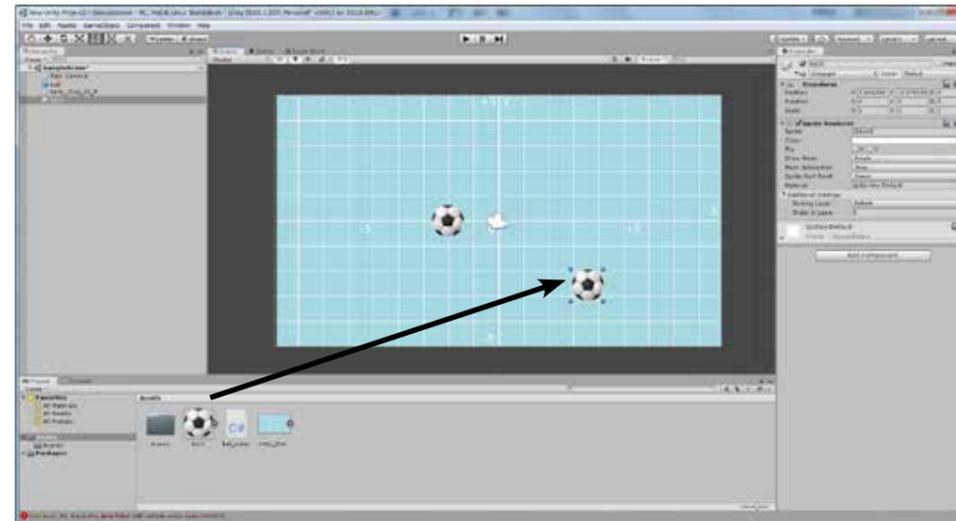
上下左右の壁に当たった時にX, Y方向それぞれの移動量はどうか？



左右の壁に当たった時には ( ) 方向の移動量に変化

上下の壁に当たった時には ( ) 方向の移動量に変化

## スクリプトのアタッチ



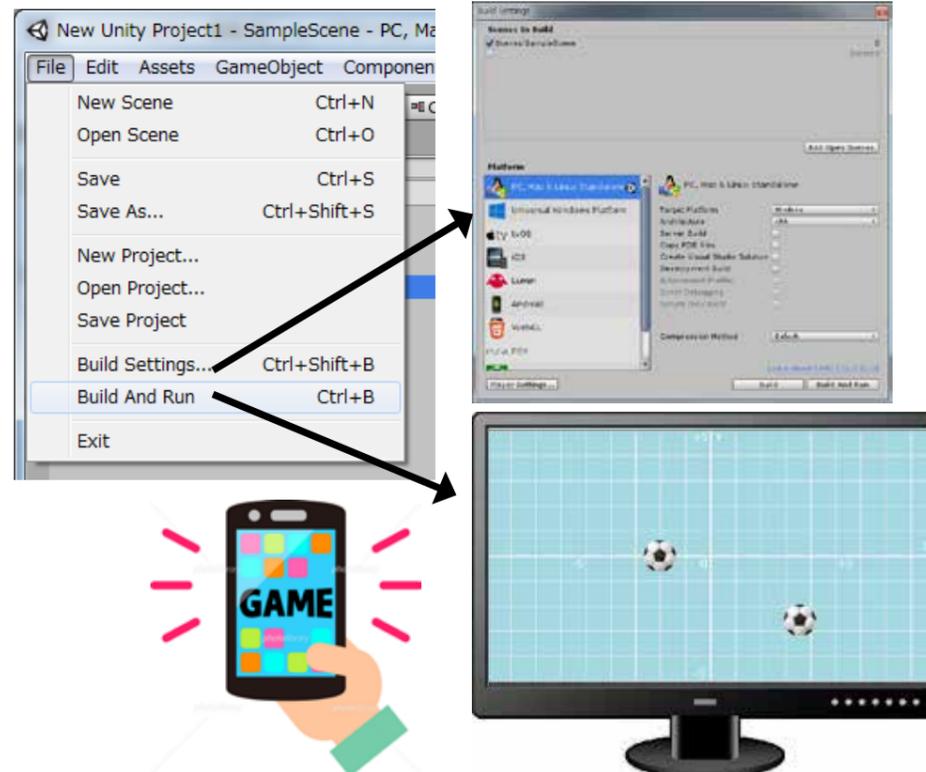
スクリプトは一つのコンポーネントとして、オブジェクトに追加されますが、通常は作成したスクリプトをヒエラルキーウィンドのオブジェクトに「アタッチ」して関連付けます。

プロジェクトウィ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…

プロジェクトウィンドから作成済みのスクリプトをヒエラルキーウィ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。インスペクターウィンドに「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



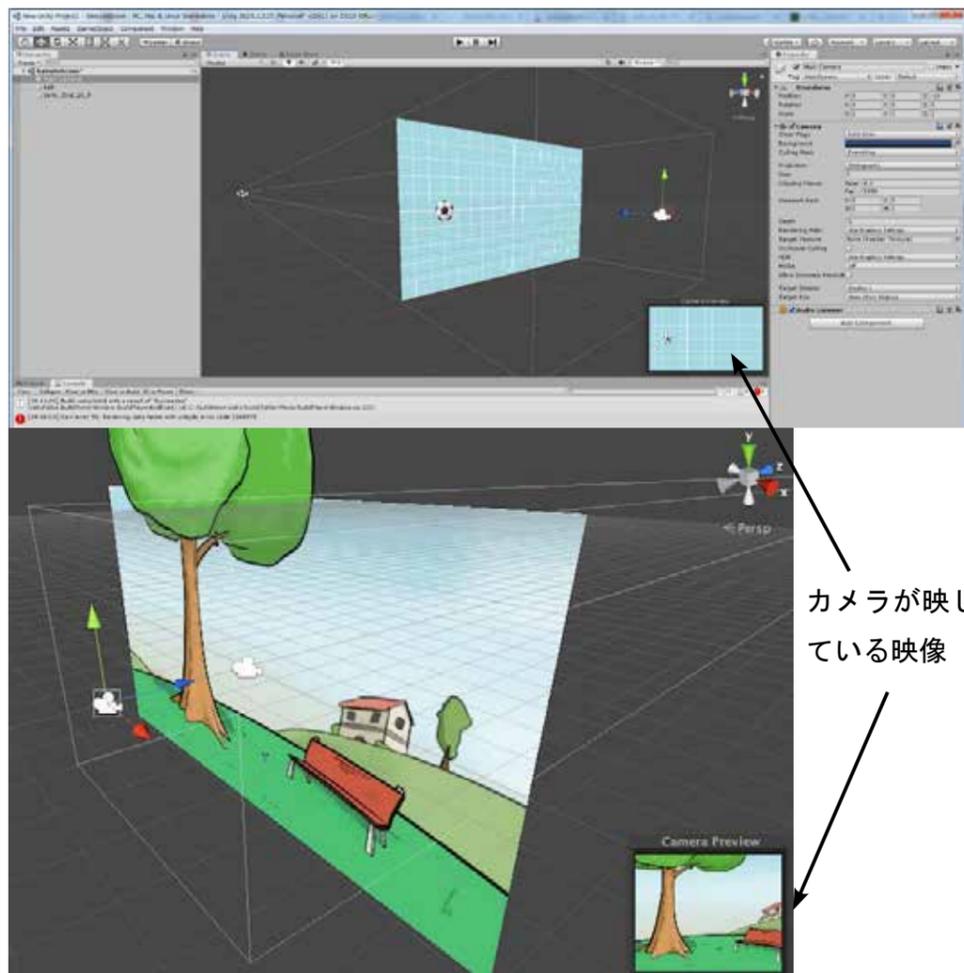
今までの実行は「Unity Editor」上でのものでした。実際にWindowsやスマホで実行するためにはプロジェクトを「ビルド」する必要があります。

「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**

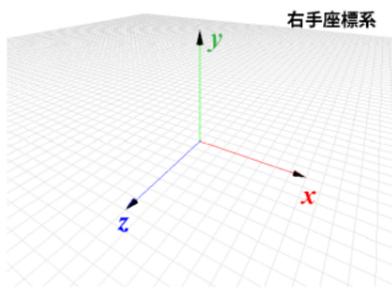
# カメラ



カメラが映している映像

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。

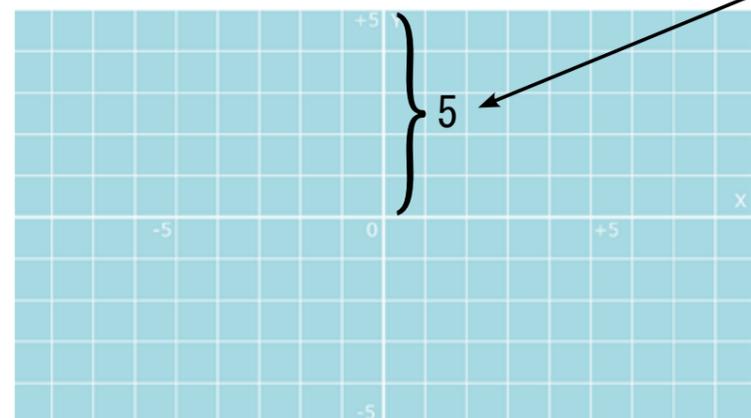
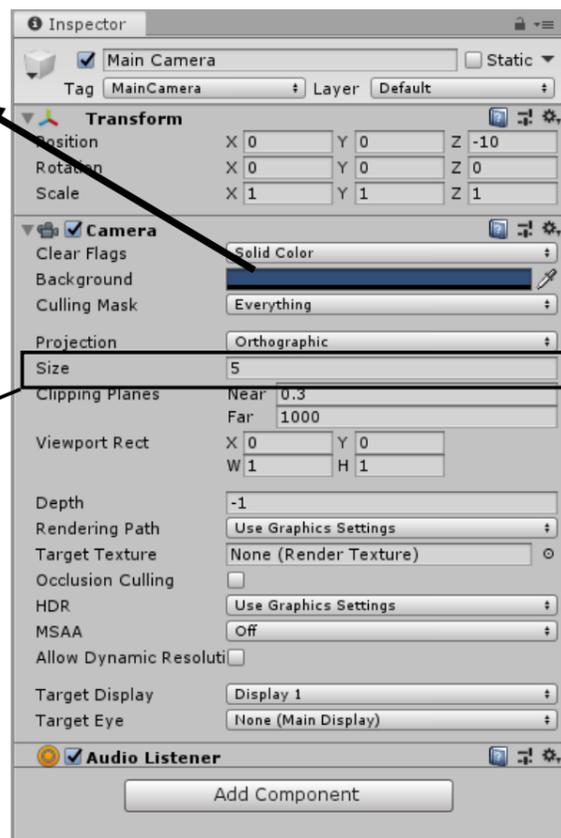
2Dの場合にはXY座標は(0,0)、Z軸は「-10」にセットされています。ヒエラルキーウィンドの「Main Camera」を選択すると、右下にカメラの映像が確認できます。設定はインスペクターウィンドで確認できます。



「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

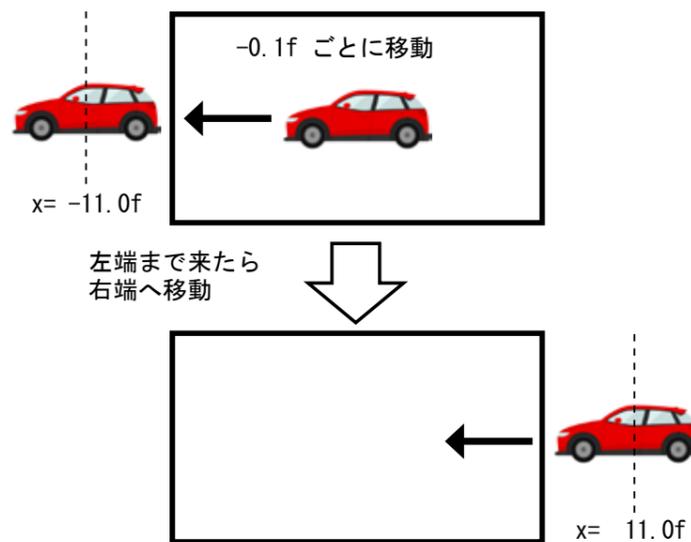
位置を表す変数型「Vector3」の使用方法を学習します。Vector2 もあります。

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
{
    float ax = -0.1f;

    void Update()
    {
        float x;
        transform.Translate( -0.1f , 0, 0);
        x = transform.position.x;
        if (x <= -11.0f )
        {
            transform.position = 
        }
    }
}
```

## 重要 動くオブジェクトの作り方

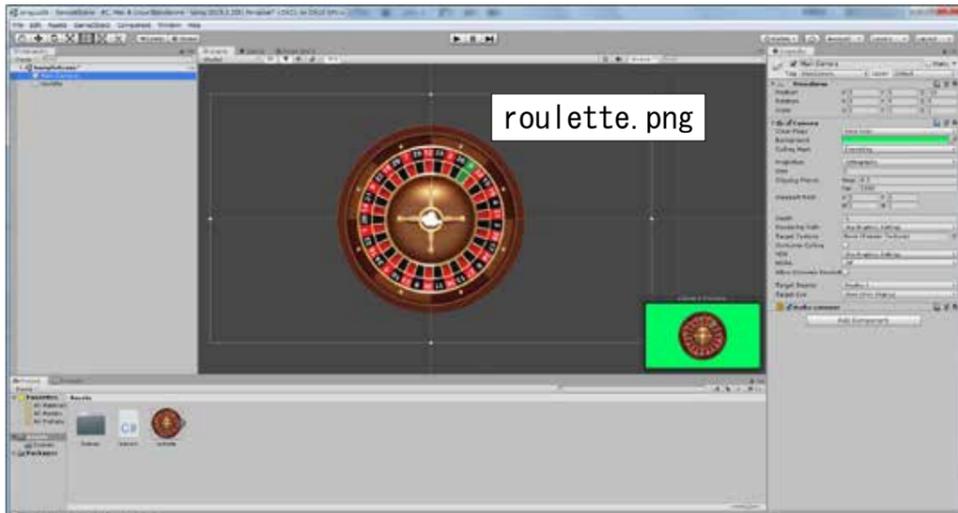
- ①シーンビューにオブジェクトを配置します。
- ②オブジェクトを動作させるスクリプトを作成します。
- ③作成したスクリプトをオブジェクトにアタッチします。

x座標を -0.1 ごとに移動

もし左端 (x <= -11) まで来たら

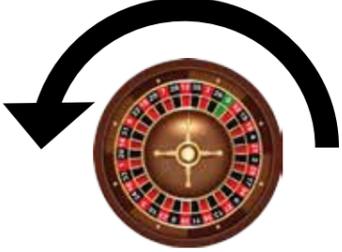
右端 (x =11) に移動する

## 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。



```
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 = 
    }
}
```

speed 分だけ回転  
 回転角度を減らす

## シーン名 e02rou

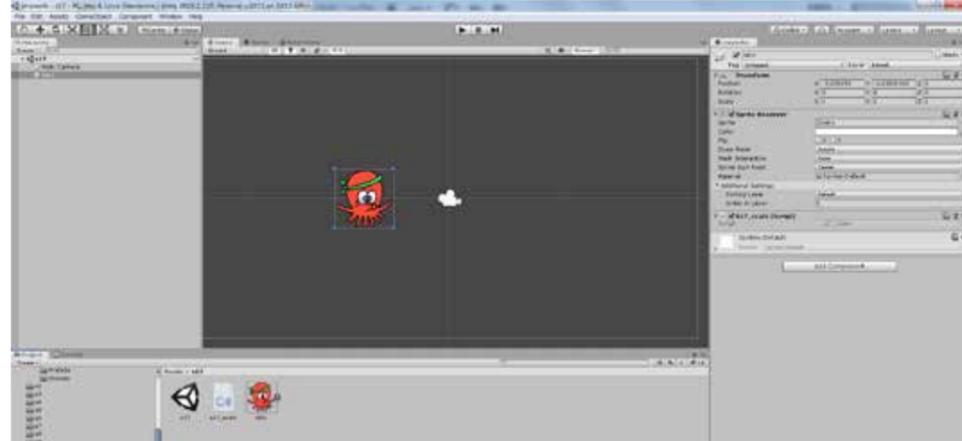
- ①ルーレットの画像「roulette.png」を中心に配置し、大きさを整える。
- ②「Main Camera」を選択し背景色を適当なものに変更する。
- ③オブジェクトを回転させるスクリプトを作成する。
- ④ルーレットのオブジェクトにスクリプトをアタッチする。



## 角度を減らす計算方法例

計算式	1回の回転角度	
	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。



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
```

スクリプト	e03_scale.cs
アタッチ先	タコ

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

public class s17_scale : MonoBehaviour
{
    float s = 1.0f;
    float sx = 0.05f;

    void Update()
    {
        transform.localScale = new Vector3(s, s, 1);
        s = s + sx;
        if (  )
        if (  )
    }
}
```

スケールを表す変数 一度に変化するスケール値  
 Scale 1 ↔ Scale 3  
 x y 方向のスケールを「s」に設定  
 スケールを sx 分だけ増減する。

## 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。



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(
        )
    }
}
```

フレームごとに  
-0.05f 移動

X= +4, Y=0, Z=0

X= -4, Y=0, Z=0

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

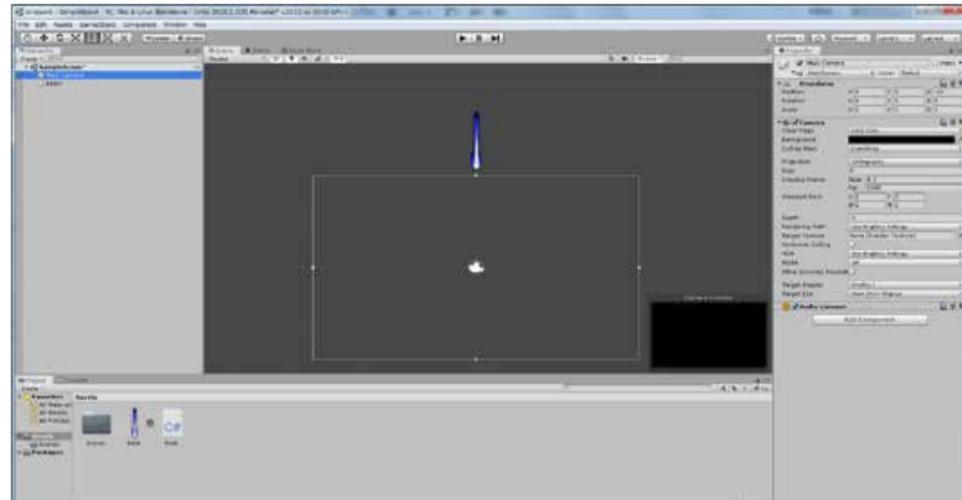
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class e04_tire : MonoBehaviour
{
    void Update()
    {
    }
}
```

タイヤを回転

## 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。

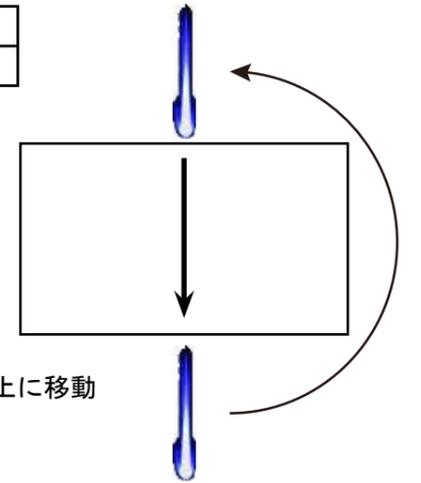
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

スクリプト	r02_beem.cs
アタッチ先	流星

```
public class r02_beem : MonoBehaviour
{
    int x = 0;

    void Update()
    {
        transform.Translate(0, -0.2f, 0);
        if (transform.position.y < -6.0f)
        {
            transform.position = new Vector3(x, 6.0f, 0);
        }
    }
}
```

画面の外に出たら  
画面上に移動

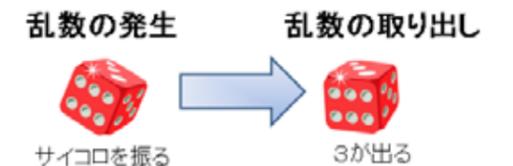


**乱数を得る** 乱数とは、ある範囲の数値から任意に (ランダムに) 取り出した数値のことです。ゲームにおける乱数といえば、敵と遭遇するかどうか、宝箱の出現確率、魔法攻撃の状態異常が発生するか、など様々なシーンで使われています。Unity で乱数を得るためには次のようにします。

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)



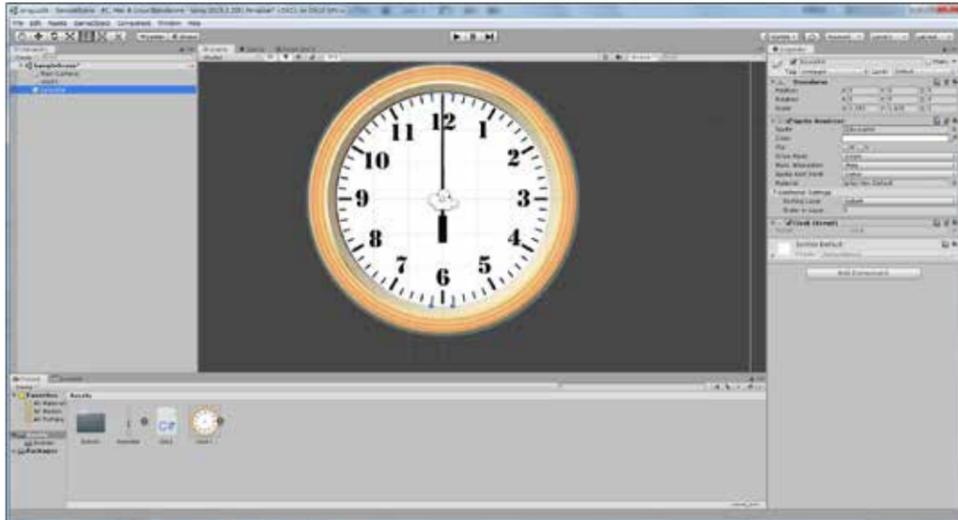
```
... 1行追加する
    if (transform.position.y < -6.0f)
    {
        x = Random.Range (-7, 8); -7 ~ +7 の乱数を発生する
        transform.position = new Vector3 ( x, 6.0f, 0 ); 横位置は乱数
    }
...

```

**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なります。

「move」は自分で作成したメソッドです。

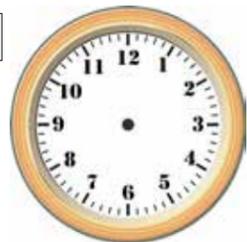
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
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
}
```

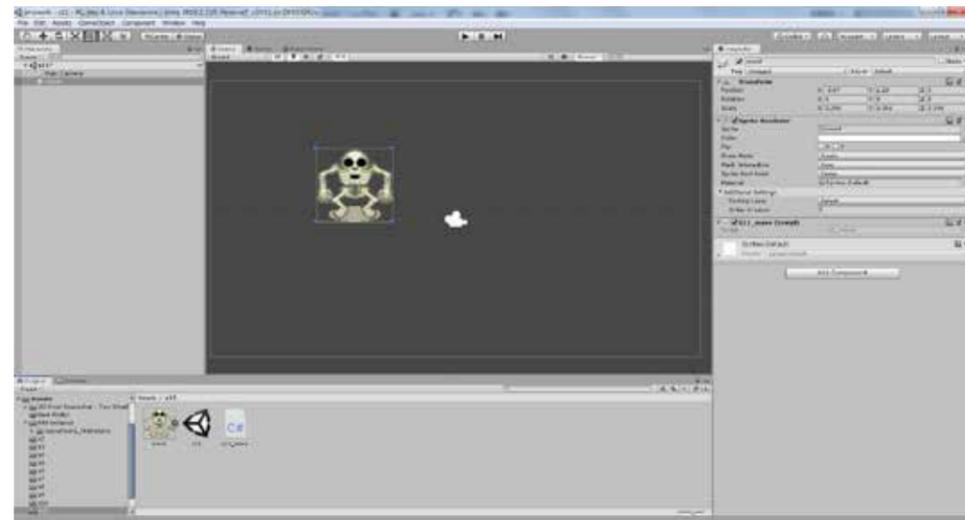
clock1.png



byoushin.png

スクリプト	r03_clock.cs
アタッチ先	秒針

### 演習5 キャラクターが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;
```

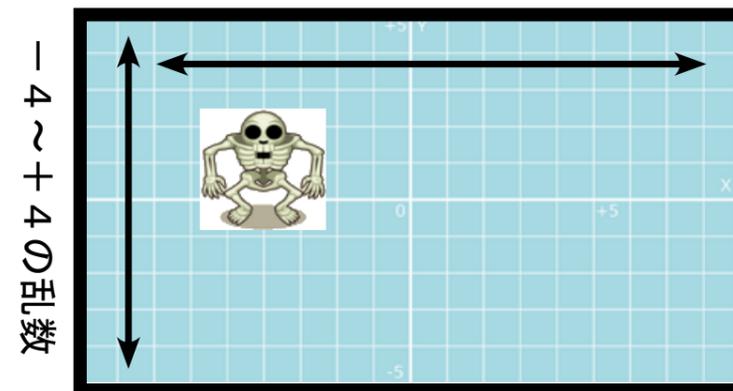
```
public class e05_move : MonoBehaviour
```

```
{
    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

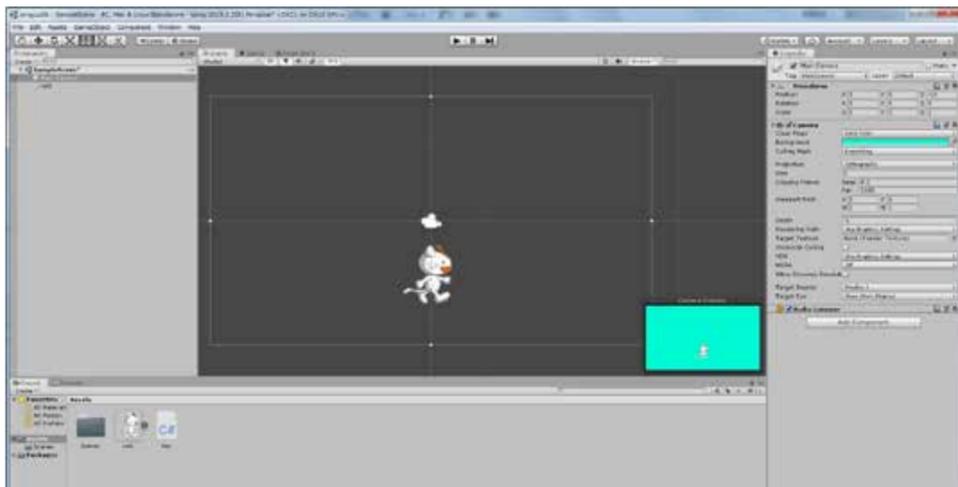


#### 重要 動くオブジェクトの作り方

- 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。



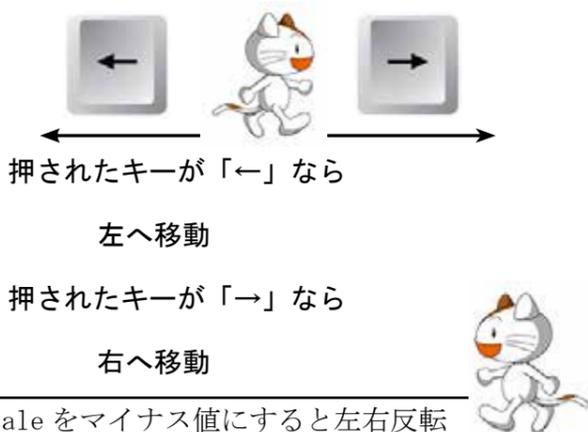
シーン名 r04cat

- 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。

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

```
public class r04_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);
        }
    }
}
```

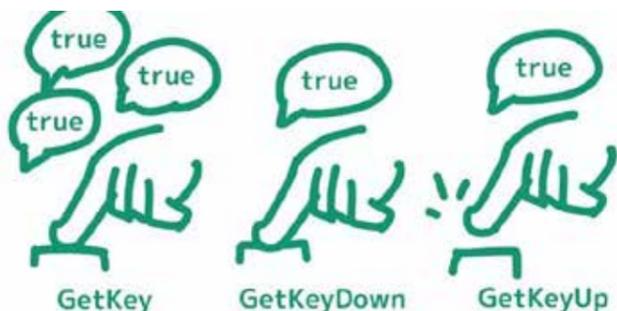
スクリプト r04\_key.cs  
アタッチ先 猫



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
Input.GetKeyUp	キーを押している状態から離れた時

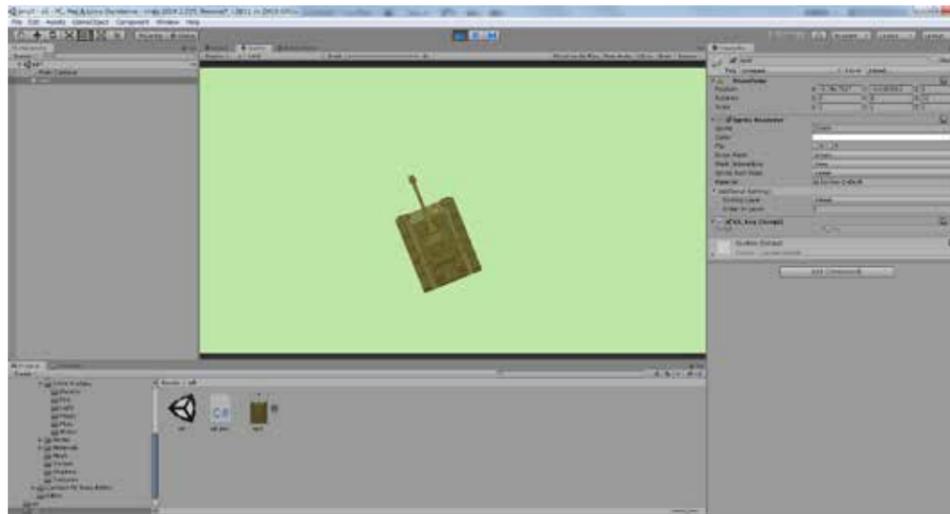


キーボード	値
Return	Return
Space	Space
上矢印キー	UpArrow
下矢印キー	DownArrow
右矢印キー	RightArrow
左矢印キー	LeftArrow
F1	F1
F15	F15
キーボード上の 0	Alpha0
キーボード上の 9	Alpha9
テンキー 0	Keypad0
テンキー 9	Keypad9
a	A
z	Z
Left mouse button	Mouse0
Right mouse button	Mouse1

## 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。

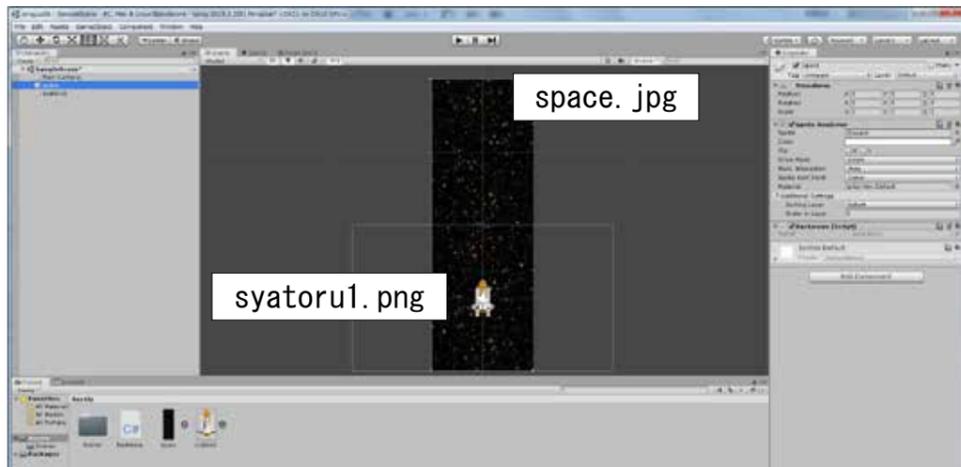


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
public class e06_key : MonoBehaviour
{
    void Update()
    {
        if ( )
        {
            戦車を「左に1度」回転
        }
        if ( )
        {
            戦車を「右に1度」回転
        }
        if ( )
        {
            戦車を「前に0.1f」移動
        }
        if ( )
        {
            戦車を「後に0.1f」移動
        }
    }
}
```

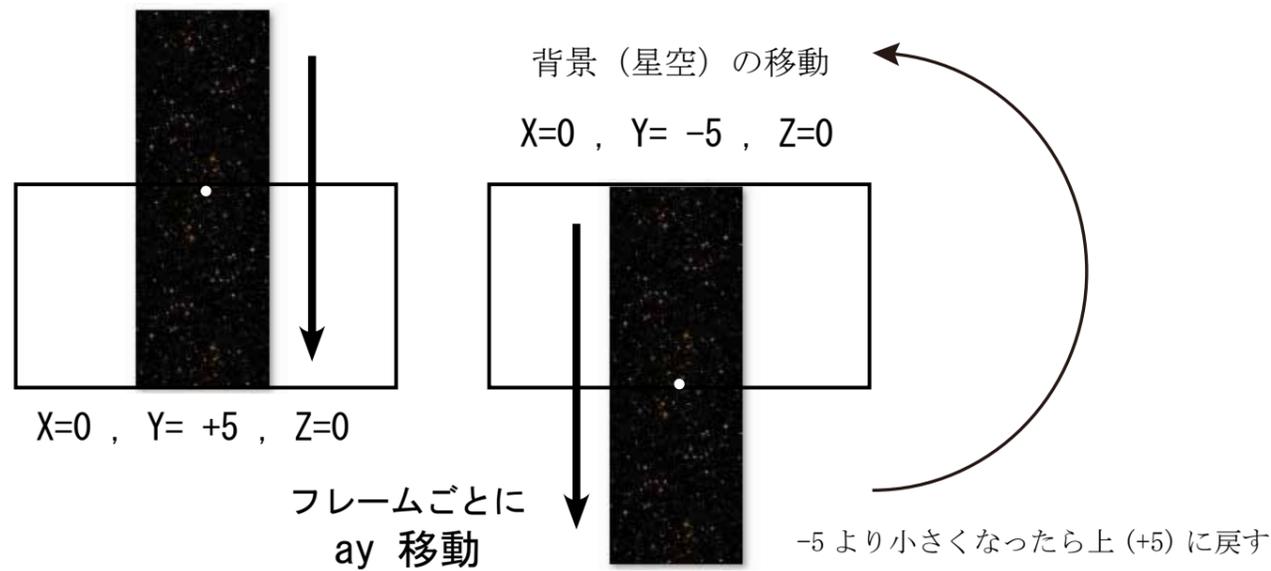
スクリプト e06\_key.cs  
アタッチ先 戦車

**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

背景（星空）のスクロールスピード変化

```

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

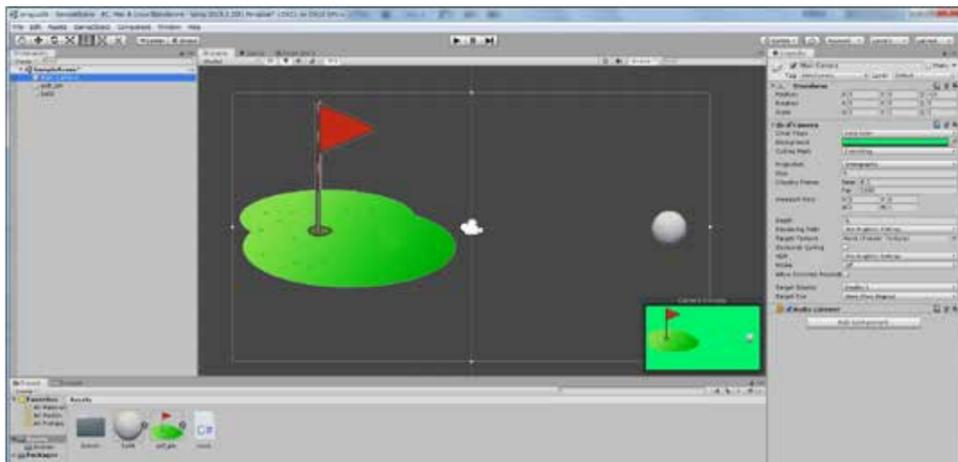
public class e07_move : MonoBehaviour
{
    float ay = -0.02f;

    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
        }
    }
}
    
```

## 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。



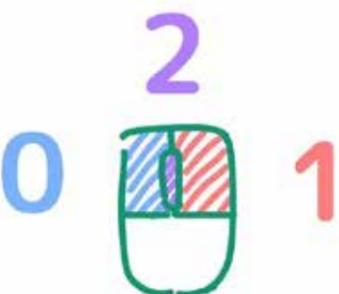
マウスによる入力も Input クラスで実装することが出来ます。スマホやタブレットで実行するときは「タップ」の動作に該当します。ボタンの種類は呼び出すときの引数によって指定します。

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離れた時
Input.mousePosition.x	マウスのX座標
Input.mousePosition.y	マウスのY座標

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。



```

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))
        {
            x1 = Input.mousePosition.x;
            Debug.Log(x1);
        }
        if (Input.GetMouseButtonUp(0))
        {
            x2 = Input.mousePosition.x;
            speed = (x2-x1)/500.0f;
        }
        transform.Translate(speed, 0, 0);
        speed = speed * 0.98f;
    }
}
    
```

左ボタンが押されたら

x1 にマウス X 座標を代入  
コンソールに座標を表示

左ボタンが離されたら

x1 にマウス X 座標を代入  
初速度を計算 (ドラッグ距離を 500 で割る例)

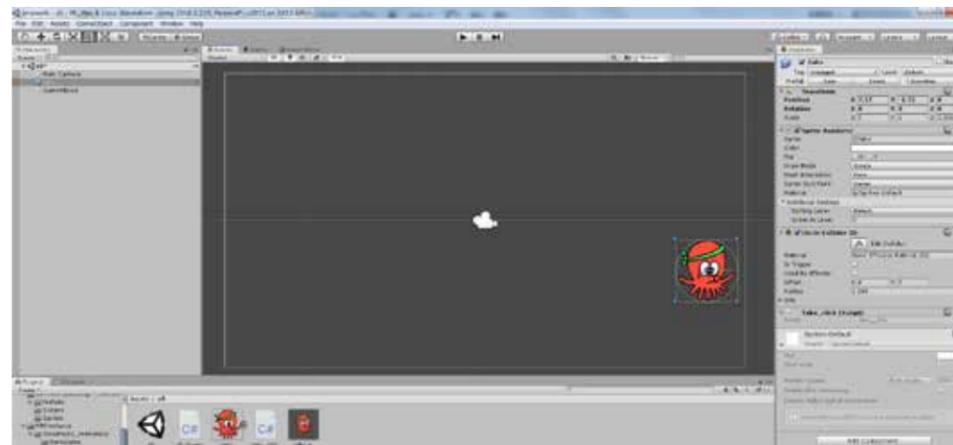
ボール移動  
減速処理 (移動距離をだんだん少なく)

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。

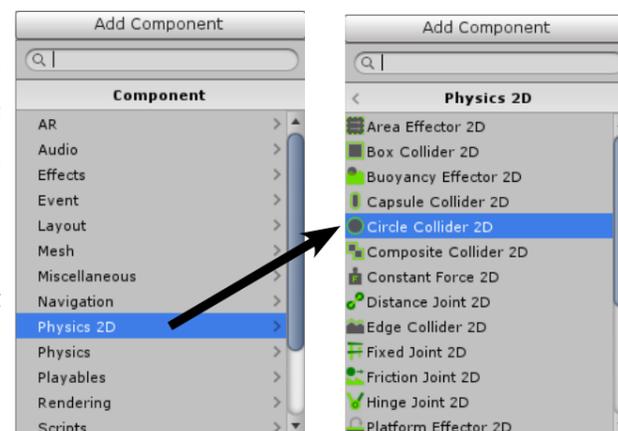
オブジェクトの周りに円形の領域が示されます。

- ③スクリプトを作成しアタッチ、実行します。

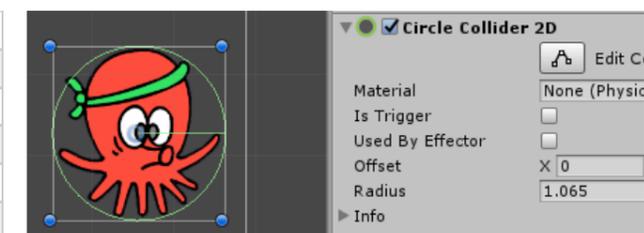


Box Collider 2D

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



```

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

public class r06_click : MonoBehaviour
{
    void OnMouseDown()
    {
        transform.localScale = new Vector3(2, 2, 1);
    }

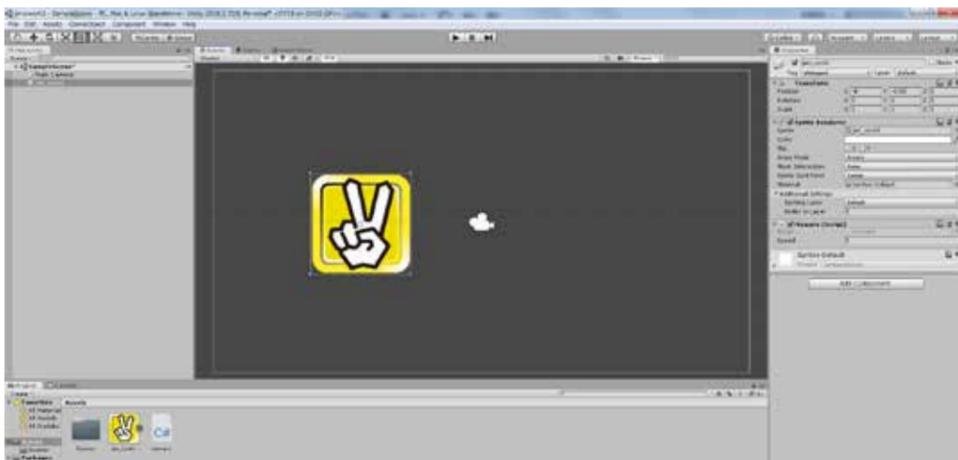
    void OnMouseUp()
    {
        transform.localScale = new Vector3(1, 1, 1);
    }
}
    
```

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
OnMouseUp()	マウスボタンを離れた時にコール
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
OnMouseDown()	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
OnMouseDown()	マウスをドラッグしている間コール



スクリプト	e06_click.cs
アタッチ先	タコ

## 例題7 public 変数

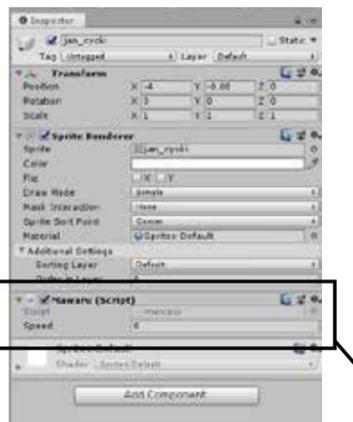


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

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



シーン名 r07pub

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。

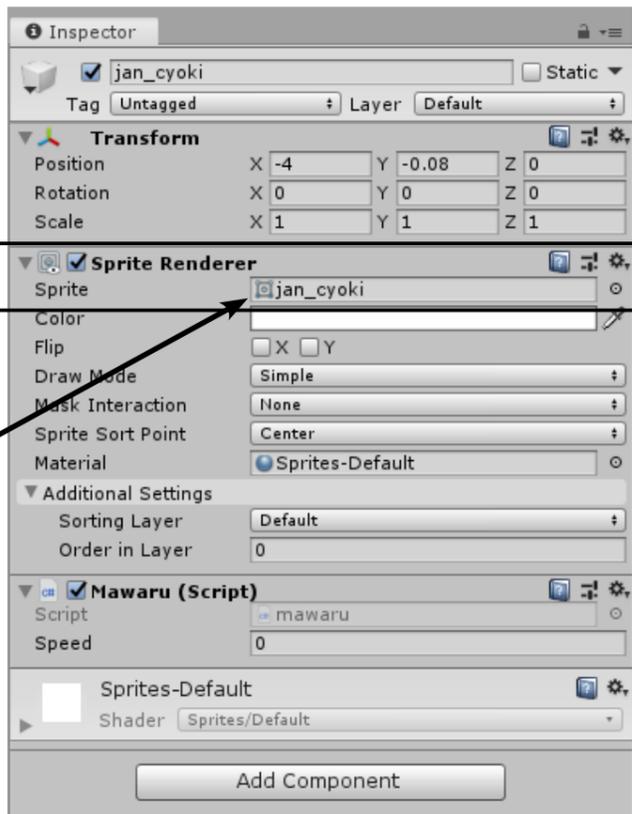
インスペクターの script コンポーネントに public 変数の項目が追加されています。

- 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。

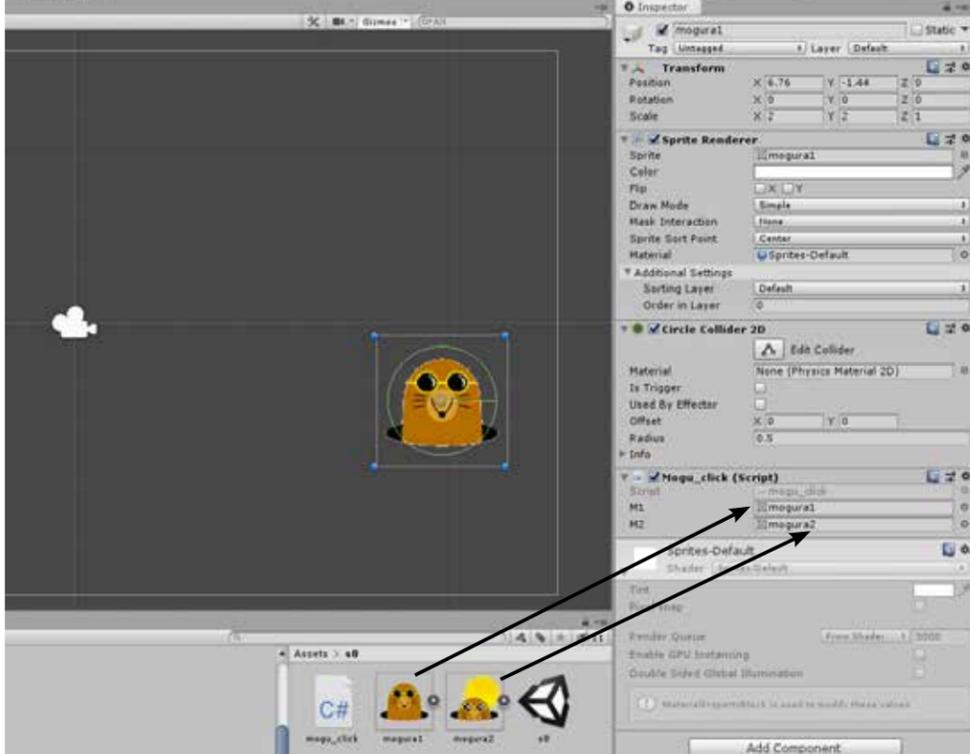


今まで意識してきませんでしたが、これと同じように、使用している画像は「Sprite Renderer」コンポーネントの「Sprite」の項目に設定されています。

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例のよう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Sprite 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;

    public Sprite m1;
    public Sprite m2;

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

    void OnMouseDown()
    {
        spr.sprite = m2;
    }

    void OnMouseUp()
    {
        spr.sprite = m1;
    }
}
```

スクリプト	r08_click.cs
アタッチ先	モグラ

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

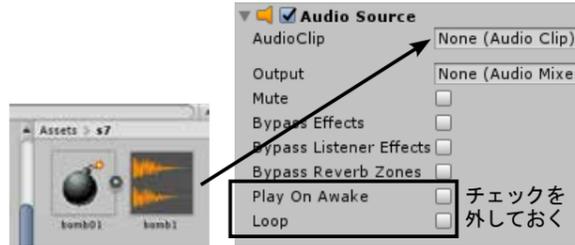
}



## 例題9 効果音・BGMを再生



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。



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

スクリプト	r09_bomb.cs
アタッチ先	爆弾

```
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
    }
}
```

### 複数のサウンドデータを使用したい場合（単独でも可）

```
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>();
    }

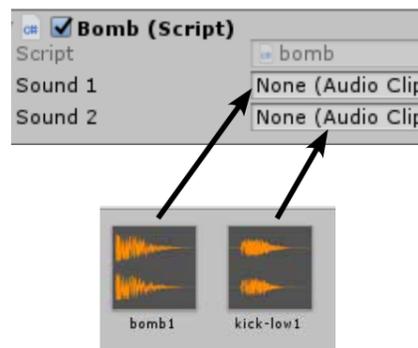
    void OnMouseDown()
    {
        aud.PlayOneShot( sound1 );    変数に格納されているサウンドファイルを再生
    }
}
```

シーン名 r09snd

- 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。

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。

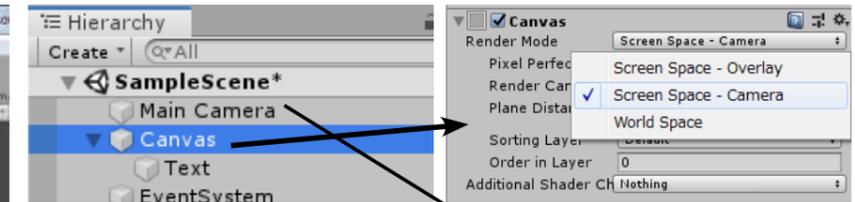
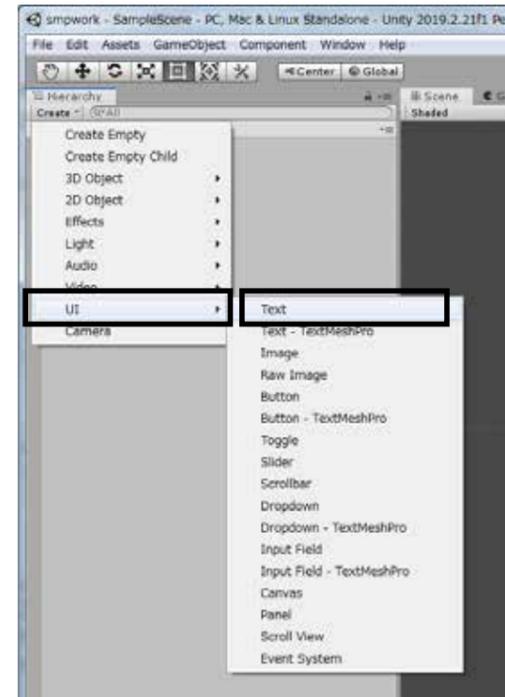
スクリプト	r09_bomb.cs
アタッチ先	爆弾



## 例題10 UI（ユーザーインターフェイス）

シーン名 r10text

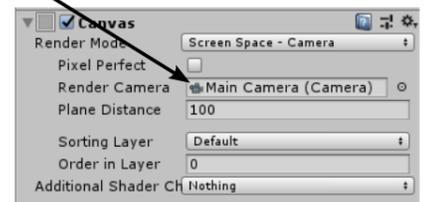
GUI（グラフィカルユーザインターフェース）とは、ユーザが視覚的に操作できる入出力機能です。各種メニューボタンやテキスト入力エリアなどがあります。他の素材と同じようにオブジェクトとして扱えます。例題としてマウスの現在位置を表示する「テキストボックス」を作ります。



①ヒエラルキーウィンドウ上の「Create」ボタンから「UI」→「Text」を選択。

②ヒエラルキーウィンドウに「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。



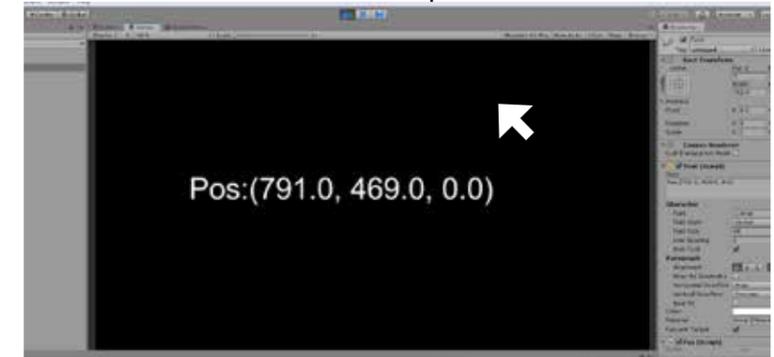
④続いて「RenderCamera」にヒエラルキーの「MainCamera」をドラッグして設定します。シーンの表示とUIの表示位置を合わせる事が出来ます。

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

```
public class r10_pos : MonoBehaviour
{
    GameObject mpos;

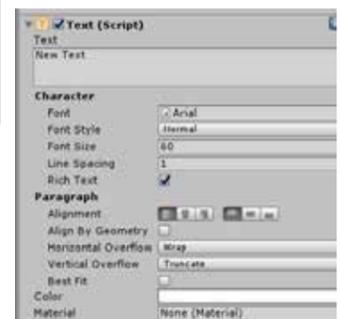
    void Start()
    {
        mpos = GameObject.Find("Text");    Text オブジェクトを探す
    }

    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
    }
}
```



スクリプト	r10_pos.cs
アタッチ先	Text

⑤ヒエラルキーウィンドウの「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。



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 )
```

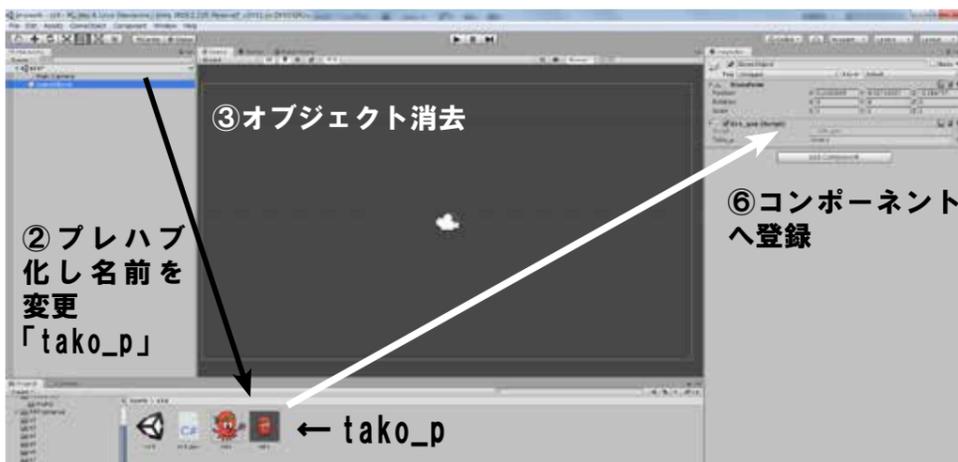
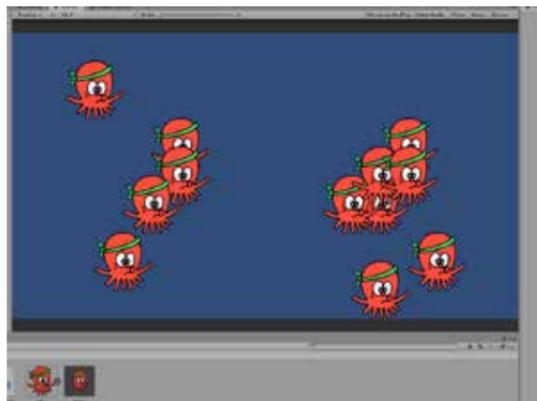
⑥スクリプトを記述しアタッチします。

## 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。



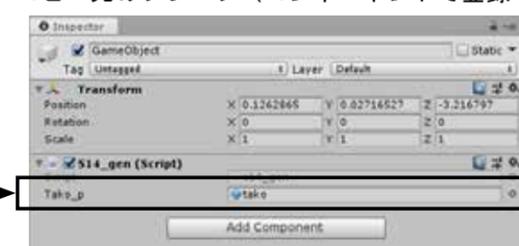
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class r11_gen : MonoBehaviour
{
    public GameObject tako_p;

    void Start()
    {
        InvokeRepeating("tako_fuyasu", 1f, 1f);
    }

    public void tako_fuyasu()
    {
        int x, y;
        GameObject tk = Instantiate( tako_p );
        x = Random.Range(-7, 8);
        y = Random.Range(-4, 5);
        tk.transform.position = new Vector3( x, y, 0);
    }
}
```

コピー元のプレハブ (コンポーネントで登録)



1 秒ごとに「tako\_fuyasu()」を実行

プレハブ tako\_p からインスタンス tk を複製  
X座標位置を -7 - 7 の乱数で取得  
Y " -4 - 4 "  
インスタンス tk 位置指定

## 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。
- ④ 1 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。

スクリプト	r11_gen.cs
アタッチ先	管理オブジェクト

## 演習 6 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 e06beem



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

スクリプト	e06_beem.cs
アタッチ先	ビーム

```
public class e06_beem : MonoBehaviour
{
    void Update()
    {
        [ ]
    }
}
```

ビームを上方に移動

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

スクリプト	e06_gen.cs
アタッチ先	宇宙船

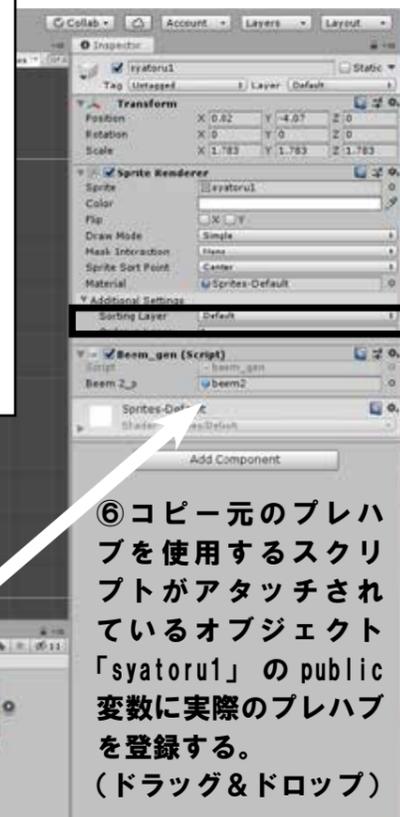
```
public class e06_gen : MonoBehaviour
{
    public GameObject beem2_p; コピー元のプレハブ (コンポーネントで登録)

    void Update()
    {
        if ( [ ] ) もしスペースキーが押されたら
        {
            GameObject bm;
            bm = [ ] 新たなゲームオブジェクト bm を 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。

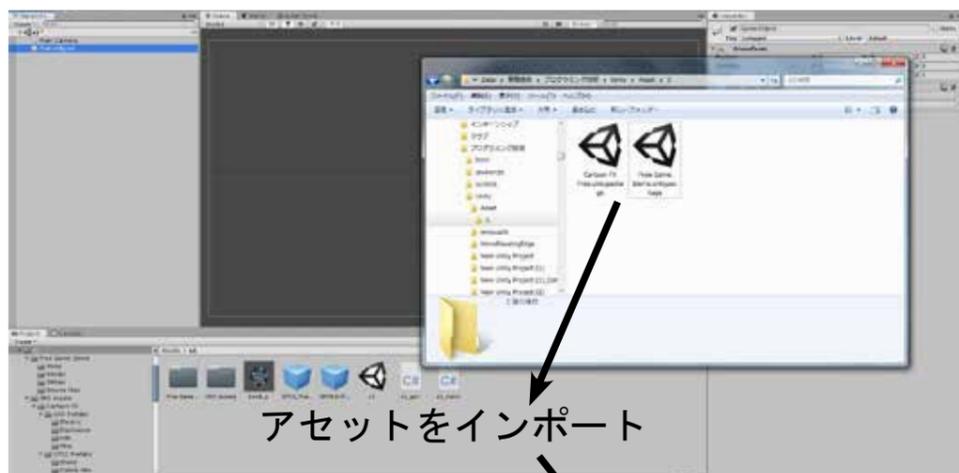
- ① 宇宙船の画像「syatoru1.png」とビームの画像「beem2.png」を大きさと位置を調整して配置する。
- ② ビームのオブジェクトを移動させるスクリプト「e06\_beem.cs」を作成し、実行を確認する。
- ③ オブジェクト「beem2」をヒエラルキーウインドからプロジェクトウインドへドラッグし「プレハブ化」し、名前を「beem2\_p」と変更する。
- ④ プレハブ化したら元のオブジェクトは消去します。
- ⑤ ビームを複製するスクリプトを作成して、宇宙船にアタッチします。
- ⑥ プレハブをコンポーネントに登録します。



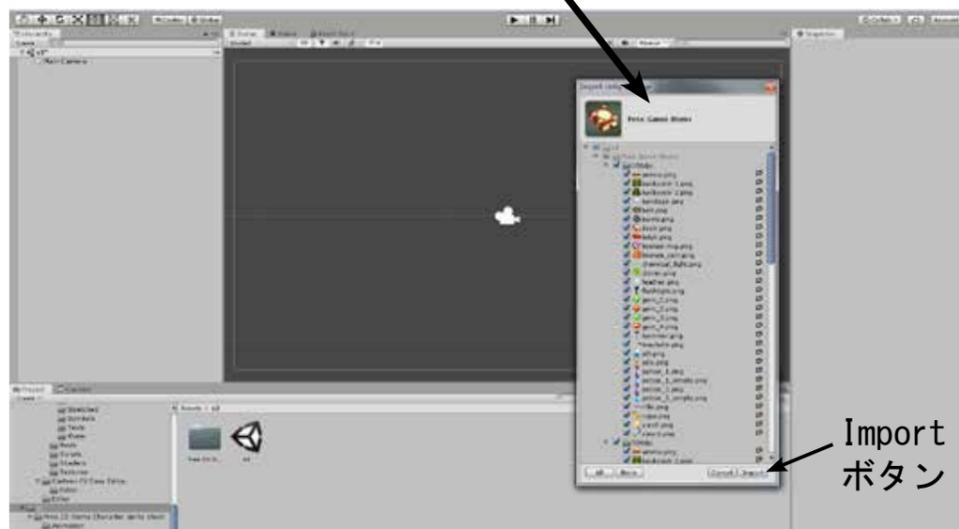
## 例題 12 アセットストア (AssetStore) を使う

シーン名 r12roop

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。



アセットをインポート



Import ボタン



Free Game Items.unitypackage

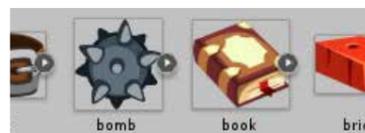
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

スクリプト	r11_move.cs
アタッチ先	爆弾

```
public class r11_move : MonoBehaviour
{
    void Update()
    {
        transform.Translate(0, -0.05f, 0);
    }
}
```

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。

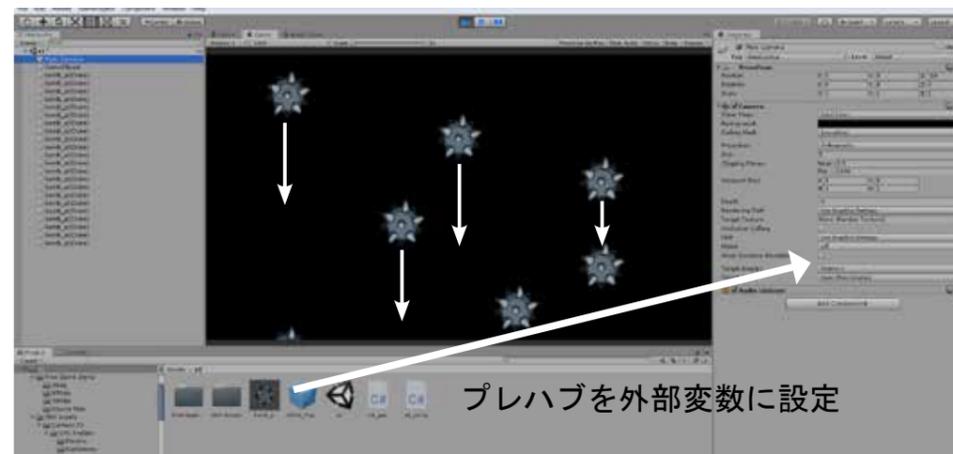
フォルダ Asset 直下にインポートされるので、わかりやすいように、シーンのフォルダへ移動しておきます。「FreeGameItems/150dpi」フォルダの内容を確認してみましょう。



②Asset 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。



プレハブを外部変数に設定

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

スクリプト	r11_gen.cs
アタッチ先	管理オブジェクト

```
public class r11_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);
    }
}
```

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

スクリプト	r11_move.cs
アタッチ先	爆弾

```
public class r11_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。

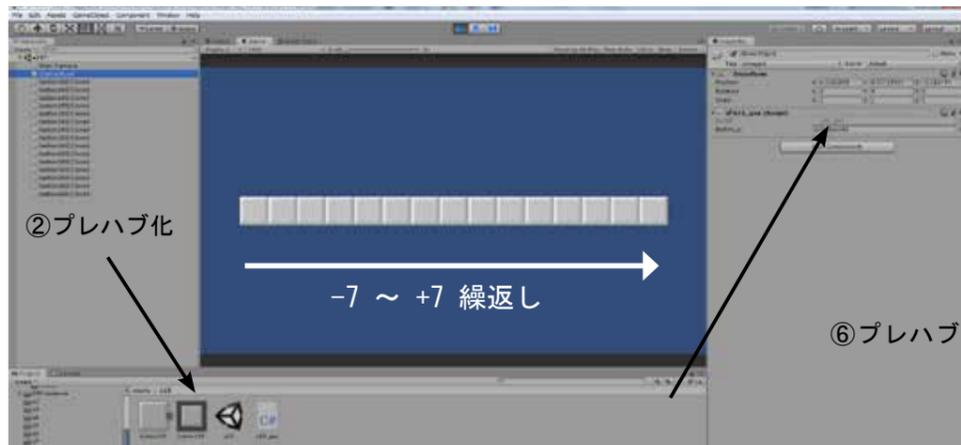


「CFX3\_Fire\_Explosion」エフェクトはプレハブとして提供されています。

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!

## 例題 13 繰り返し処理

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。



### 横一列に配置

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

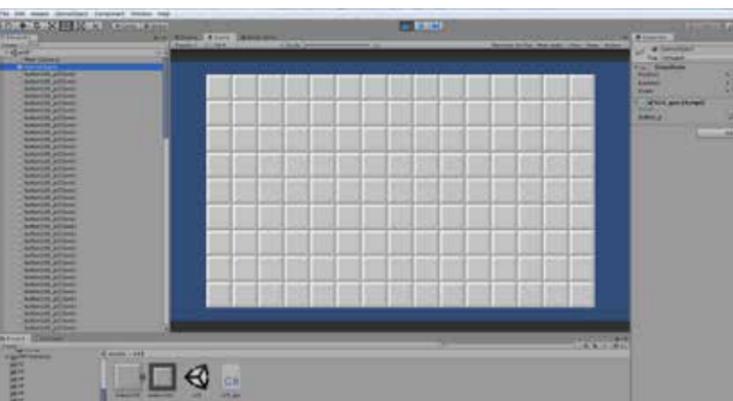
スクリプト	r13_gen.cs
アタッチ先	管理オブジェクト

```
public class r13_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 r13_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	r13_gen.cs
アタッチ先	管理オブジェクト

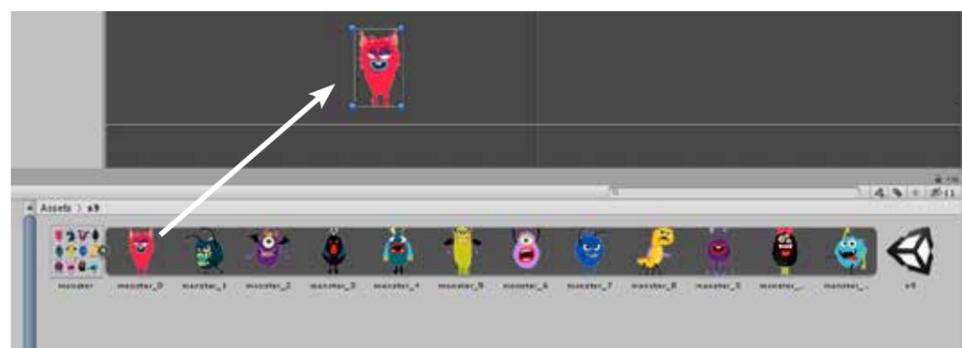
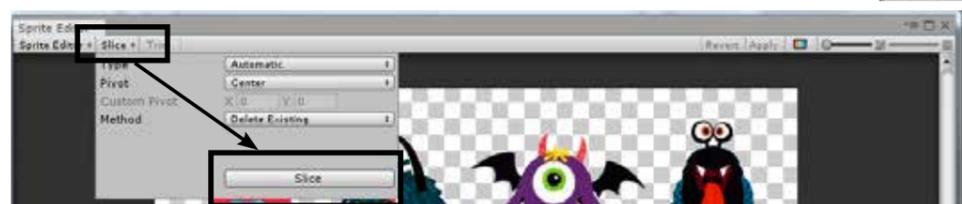
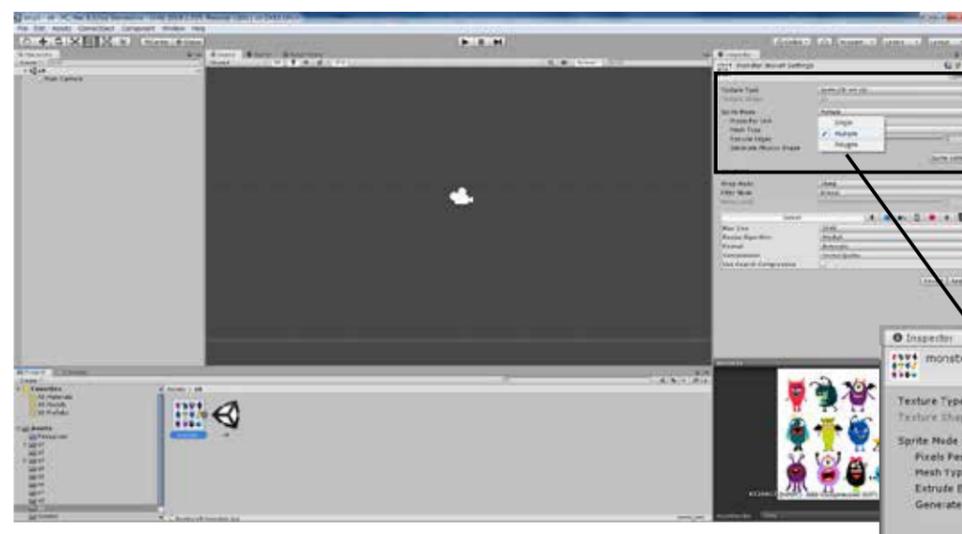
## シーン名 r13roop

- ①ボタン画像「button100.png」を配置する。
- ②オブジェクト「button100」をヒエラルキーウィンドからプロジェクトウィンドへドラッグ「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ヒエラルキーウィ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。

- ⑧縦横に配置するようにスクリプトを変更して(二重ループ)実行を確認する。

## 例題 12 スプライトエディタ (画像の切り分け)

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。



## シーン名 r14roop

- 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。

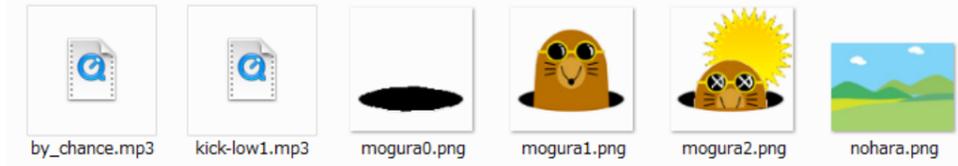
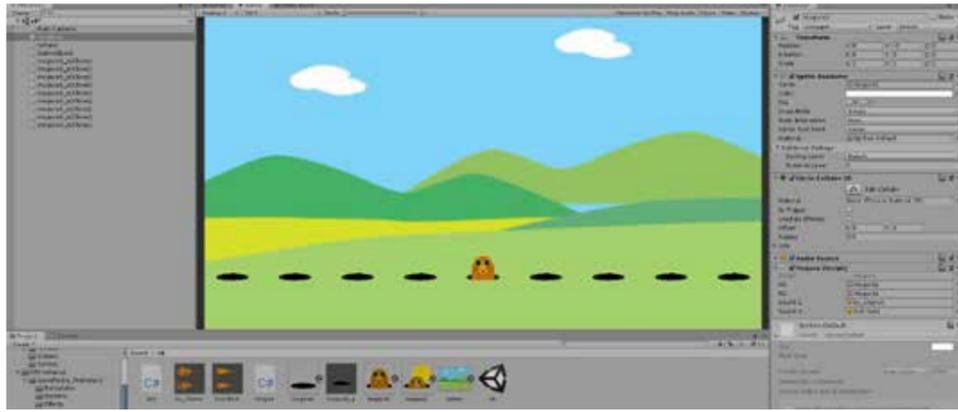
- ④右上の「Apply」ボタンをクリックすると、自動で画像が切り分けられます。それぞれのスプライトを確認してみましょう。

- ⑤スプライトエディタを閉じてプロジェクトウィンドの画像の「▶」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、ひとつ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！



```
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)
        {
            GameObject tk = Instantiate(mogura0_p);
            tk.transform.position = new Vector3(x, -3, 0);
        }
    }
}
```

**穴を並べて生成する**

for で繰り返す

穴を連続して配置

スクリプト	g01_ana.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 g01_mogura : MonoBehaviour
{
    void Start()
    {
        InvokeRepeating(1秒ごとに mogu_move() を呼び出す
    }

    void mogu_move()
    {
        int x;
        x = Random.Range(-4, 4);
        transform.position = new Vector3(x*2, -3, 0);
    }
}
```

**1秒毎ランダムに移動**

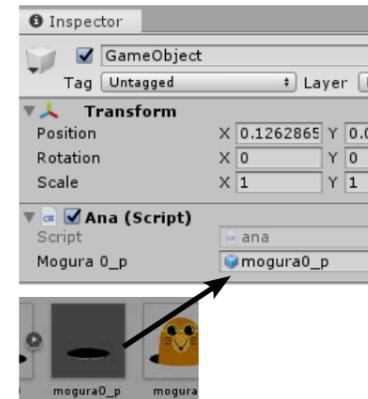
スクリプト	g01_mogura.cs
アタッチ先	モグラ

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。

-4 ~ +4 までの乱数を発生  
モグラのX座標値を -8 ~ +8 へ移動

## 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「mogura0\_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。



```
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 = Random.Range(-4, 4);
        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
    {
        spr.sprite = m2; スプライトを叩かれているものに切替
    }
}
```

**クリックでスプライト切替**

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。



```
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
    }

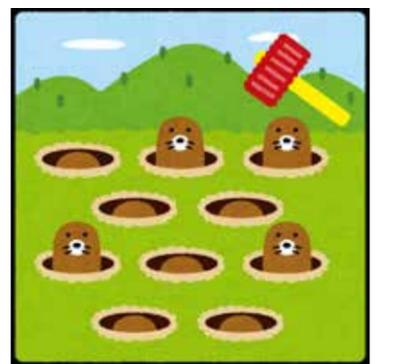
    void mogu_move()
    {
        int x;
        x = Random.Range(-4, 4);
        transform.position = new Vector3(x*2, -3, 0);
        spr.sprite = m1;
        aud.PlayOneShot(sound1); モグラ出現サウンドを再生
    }

    void OnMouseDown()
    {
        spr.sprite = m2;
        aud.PlayOneShot(sound2); 叩くサウンドを再生
    }
}
```

**効果音を再生する**

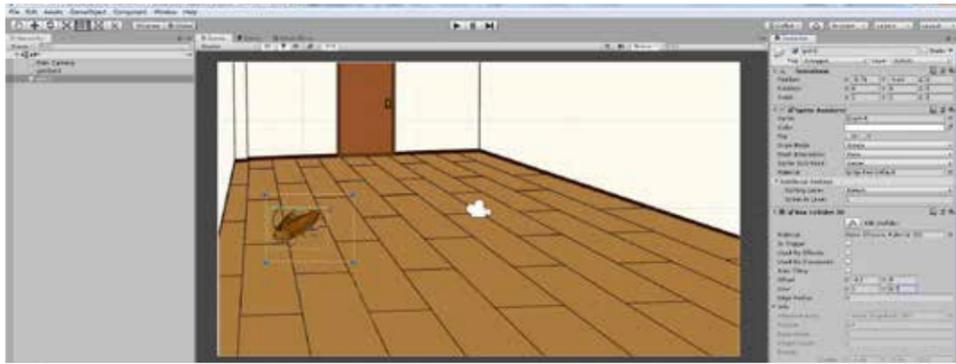
スクリプト	g01_mogura.cs
アタッチ先	モグラ

- 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
- ⑨ AudioClip 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。
  - ⑩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！！



**ゴキブリをランダムに移動**

```
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 = Random.Range(-0.3f, 0.3f);    ±0.3の乱数
        transform.Translate(x, y, 0);      乱数分だけ移動
    }
}
```

スクリプト	g02_goki.cs
アタッチ先	ゴキブリ

**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
    }

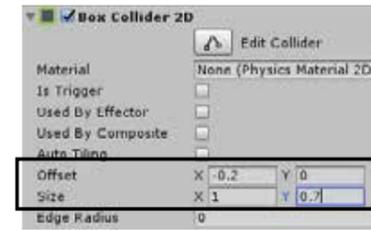
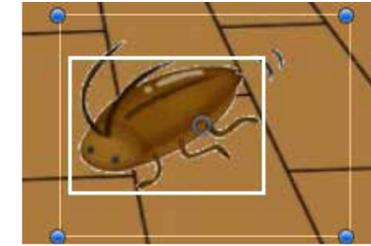
    void OnMouseDown()
    {
        spr.sprite = m2;
        Destroy(gameObject, 0.5f);
    }
}
```

マウスクリックしたら  
スプライトを「叩かれ画像」に  
0.5秒後に消滅

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;

    void Start()
    {
        spr = GetComponent<SpriteRenderer>();
        aud = GetComponent<AudioSource>();
        aud.PlayOneShot(sound1);
    }

    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);
        spr.sprite = m2;
        Destroy(gameObject, 0.5f);
    }
}
```

AudioSourceを取得  
出現した時に再生

クリックされた時に再生

スクリプト	g02_goki.cs
アタッチ先	ゴキブリ

⑦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。



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。

**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()
    {
        InvokeRepeating("goki_gene", 1f, 1f);
    }

    void goki_gene()
    {
        int x, y;
        x = Random.Range(-7, 7);
        y = Random.Range(-7, 7);

        GameObject gk = Instantiate(goki_p);
        gk.transform.position = new Vector3(x, y, 0);
    }
}
```

1秒ごとにゴキブリを増殖

1秒ごとにプレハブからインスタンスを生成

-7 ~ +7 の乱数

スクリプト	g02_gen.cs
アタッチ先	管理オブジェクト

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。

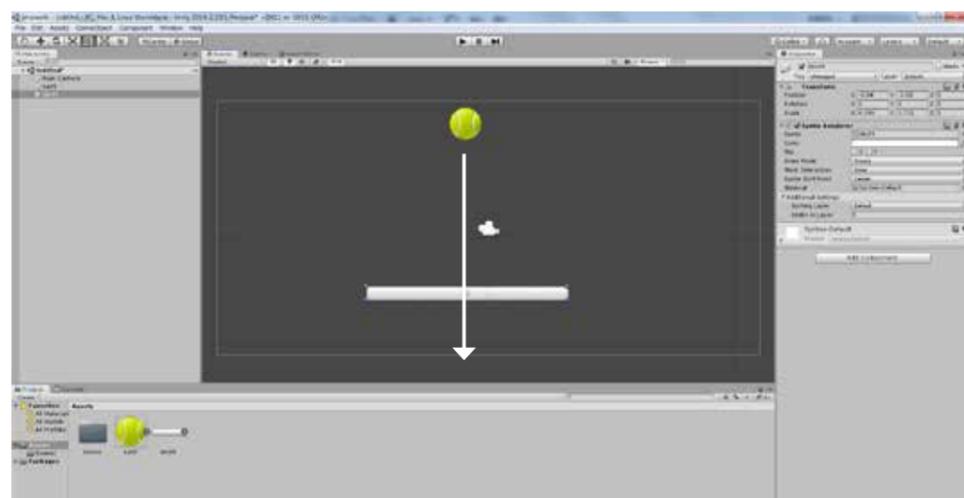
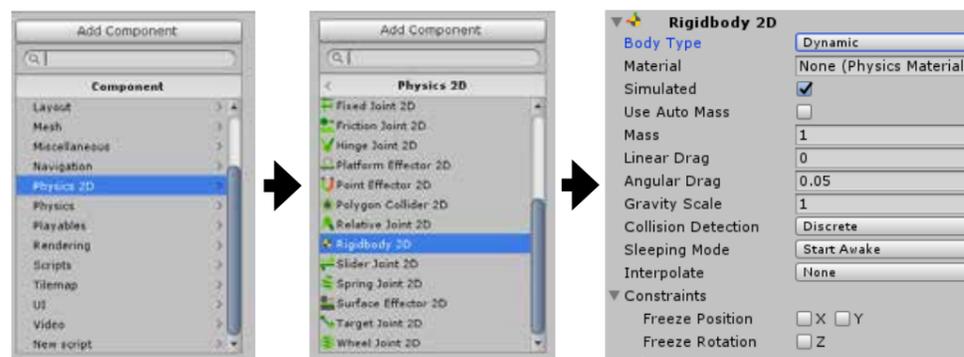
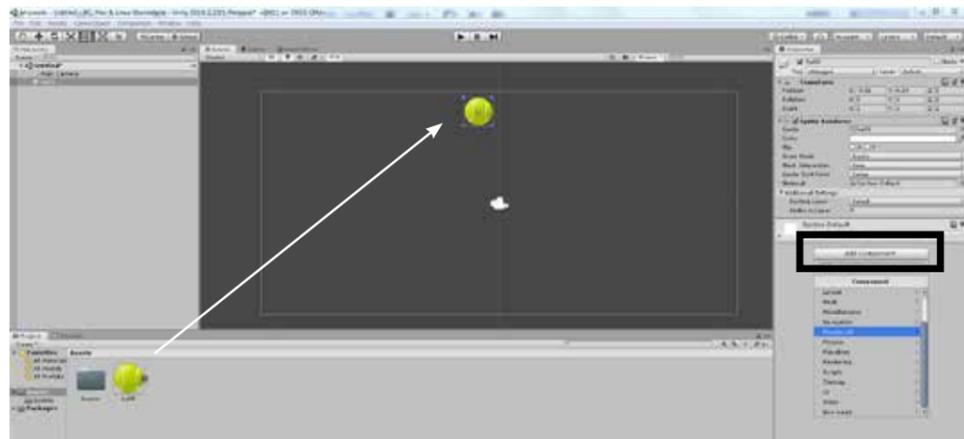
⑩空の「GameObject」のインスペクターの外部変数「goki\_p」にプレハブを設定し、実行を確認します。



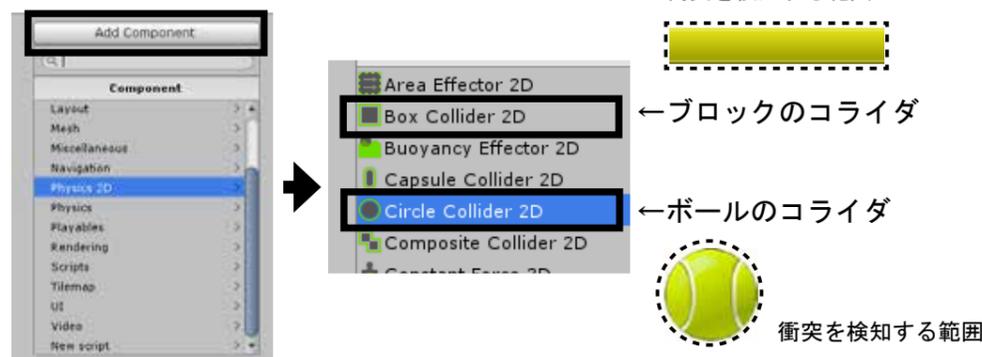
# 例題 15 物理エンジン (Physics)

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。



## 衝突検知 (コライダ) の設定



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「Collider 2D」をアタッチします。

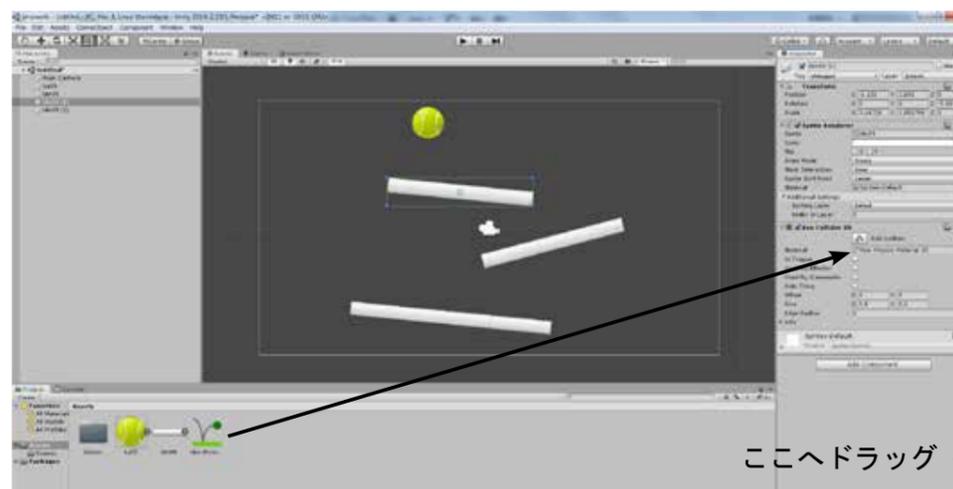
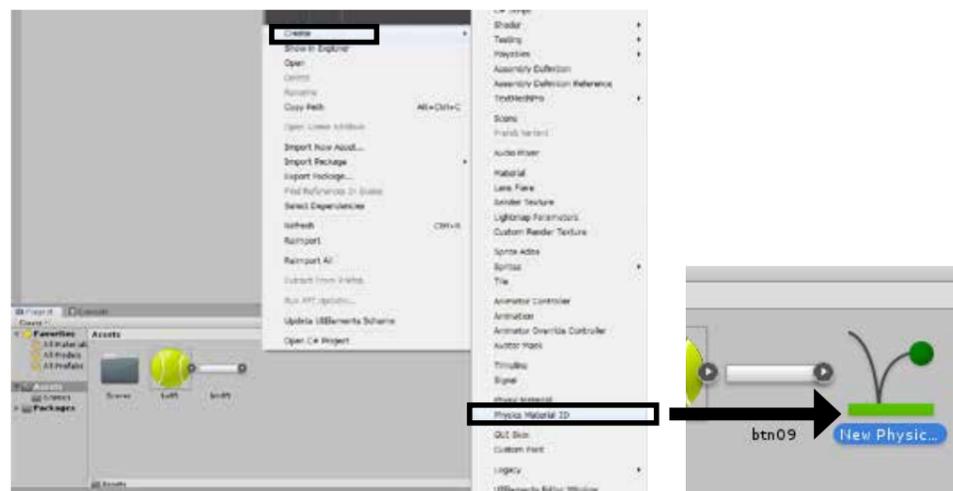
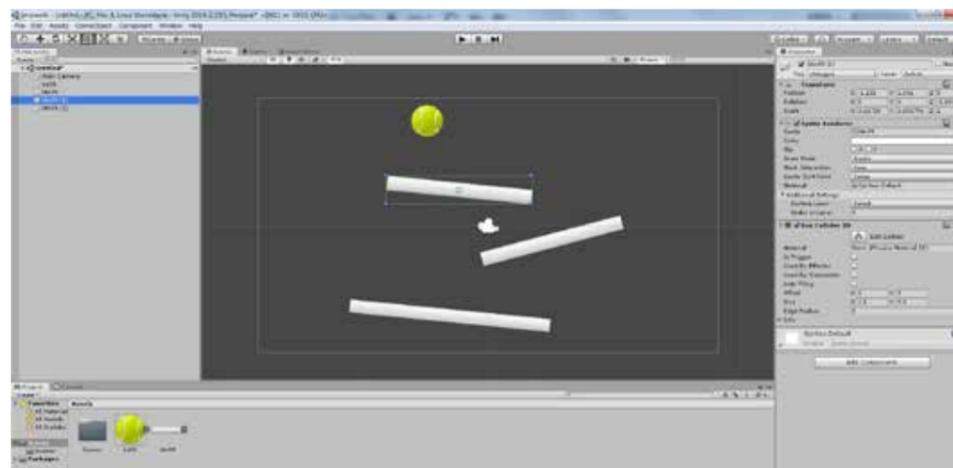
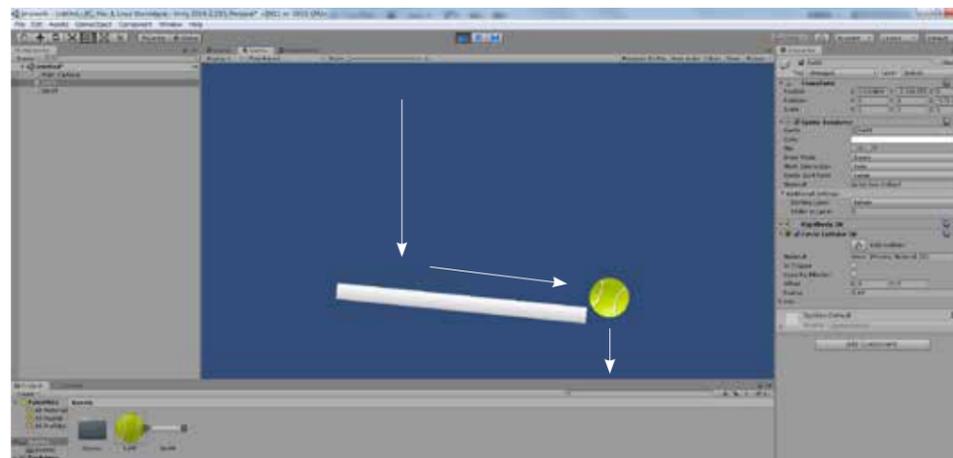
Colliderは「衝突検知」のための「領域指定」です。

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。



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。

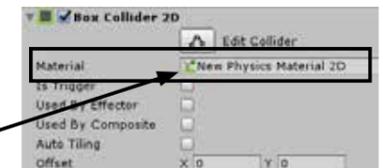
続いてボールを弾ませてみましょう。

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。

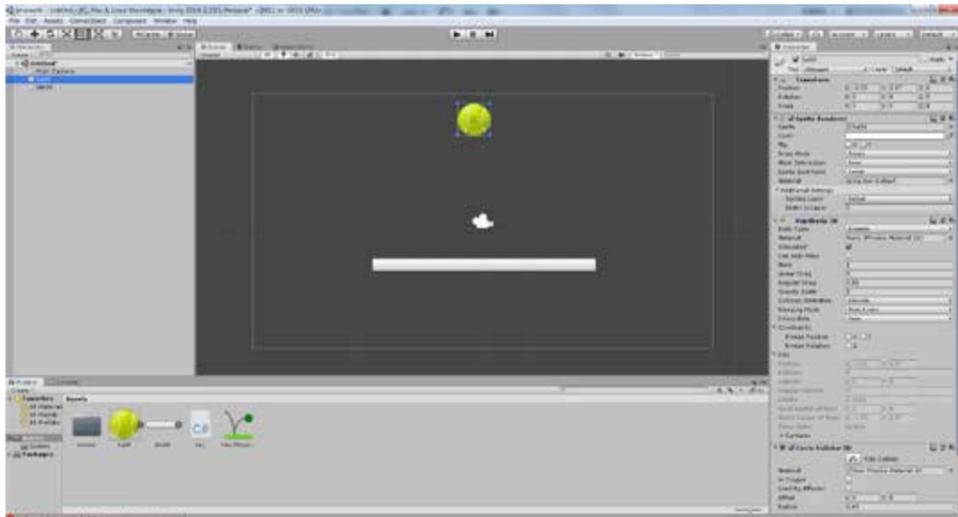


⑩ヒエラルキーウインドでボールオブジェクトを選択し、インスペクターの「Box Collider 2D」コンポーネントの「Material」に、プロジェクトウインドから設定した物理材料をアタッチし、実行してみます。



## 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。



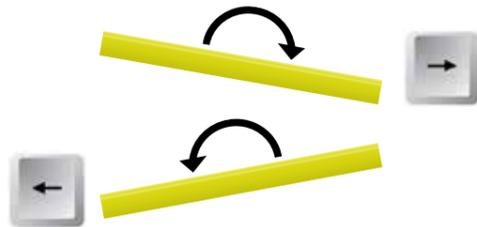
### 重要

Physics 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 ( )
        {
        }
        if ( )
        {
        }
    }
}
```



もし「←」が押されたら

左回りに0.5度回転

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

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。

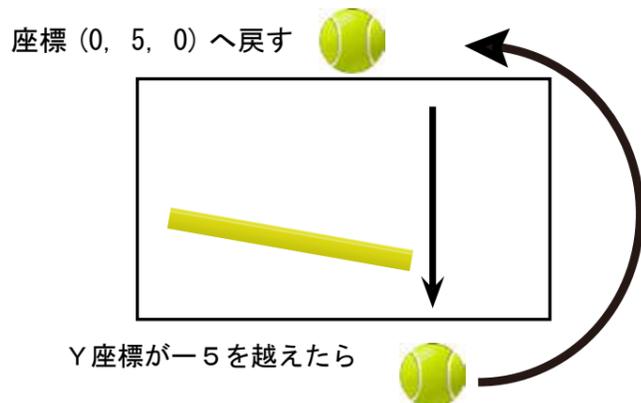
③ブロックオブジェクトに「Box Collider 2D」コンポーネントをアタッチし、実行して動作を確認します。

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。

### 発展

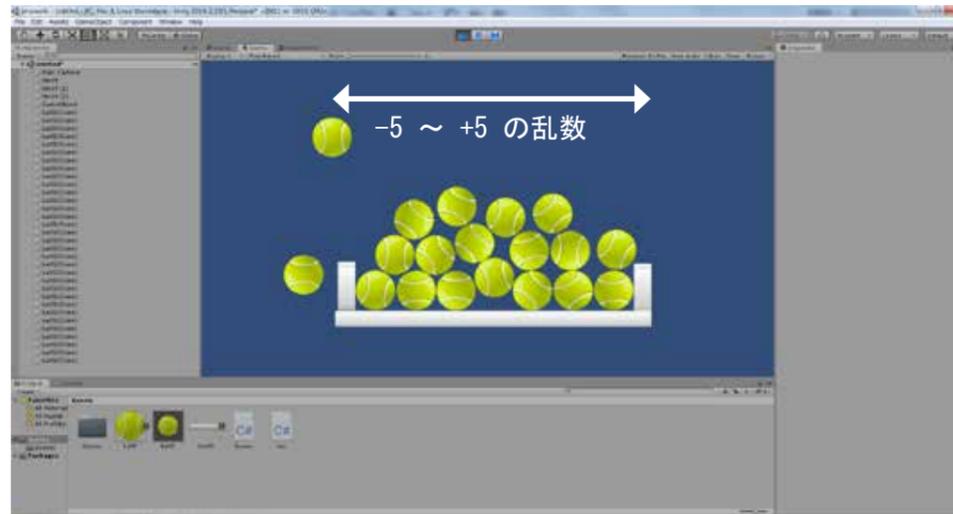
ボールが画面から下へ落ちたら (Y座標が-5より小さくなった) 上へ戻す (Y座標+5) スクリプト (ball.cs) を作成し、ボールオブジェクトにアタッチして実行してみよう。

※ Physics 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でも試してみましょう!

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

スクリプト e10\_fuyasu.cs

アタッチ先 管理オブジェクト

```
public class e10_fuyasu : MonoBehaviour
```

```
{
```

```
    public GameObject ball_p;
```

```
    void Start()
```

```
    {
        InvokeRepeating( 0.5秒ごとに ball_fuyasu を呼び出す )
    }
```

```
    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);
```

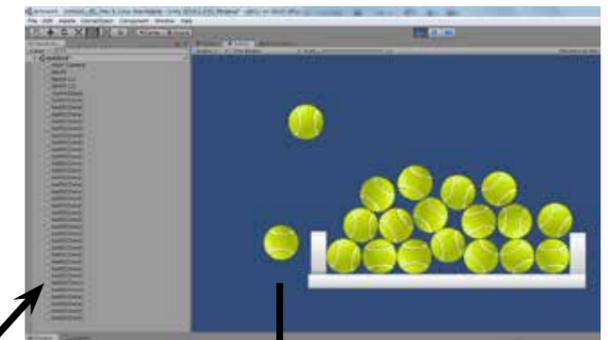
```
        ba.transform.position =
        ポールを上ランダム位置に移動
```

```
    }
```

### 発展

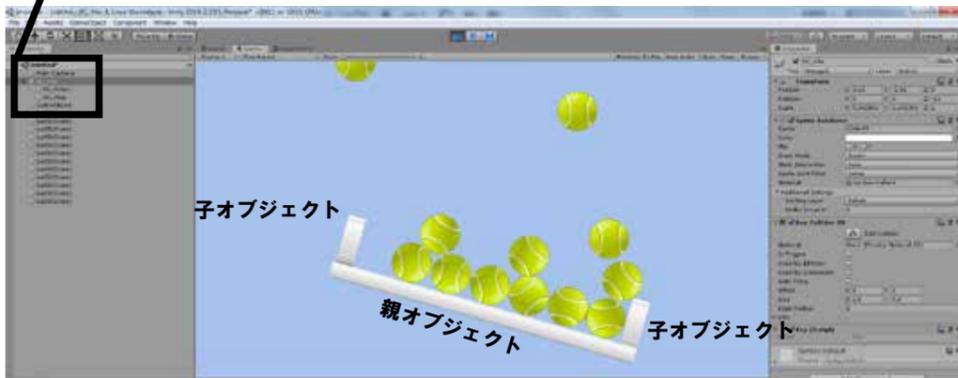
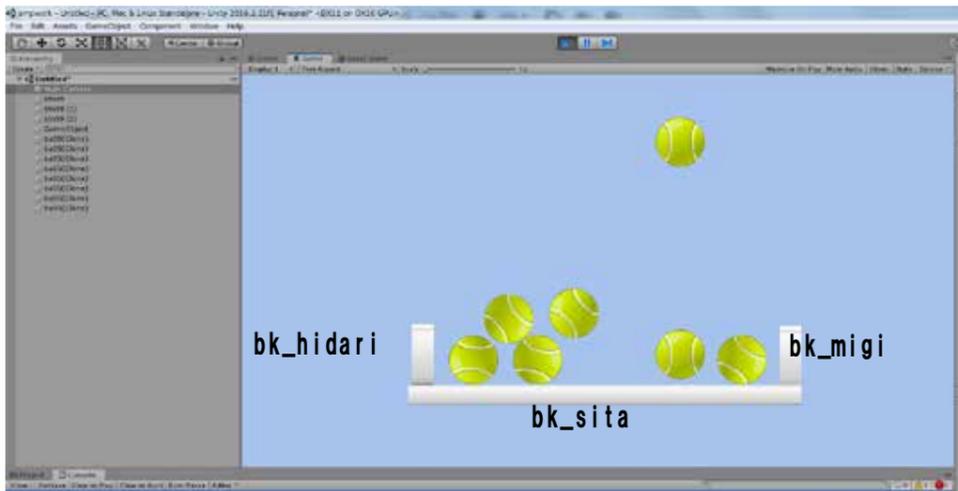
このままでは無限にボールが増え続けてしまうので、ボールが画面外へ落ちたら (Y座標が-5より小さくなった) ボールオブジェクトを削除するスクリプト (ball.cs) を作成し、ボールプレハブ (ball\_p) にアタッチして実行してみよう。

ヒエラルキーウィ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。



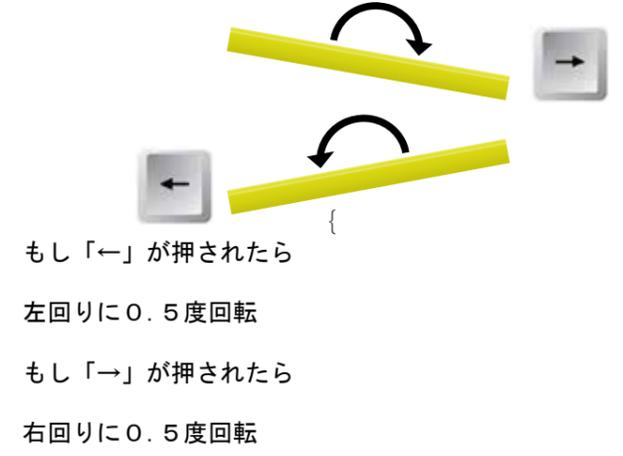
Y座標が-5を越えたら削除 (Destroy) する。

## 例題 16 親子オブジェクト



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

public class e09_key : MonoBehaviour
{
    void Update()
    {
        if (
        {
        }
        if (
        {
        }
    }
}
```



スクリプト	e09_key.cs
アタッチ先	ブロック

## 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。



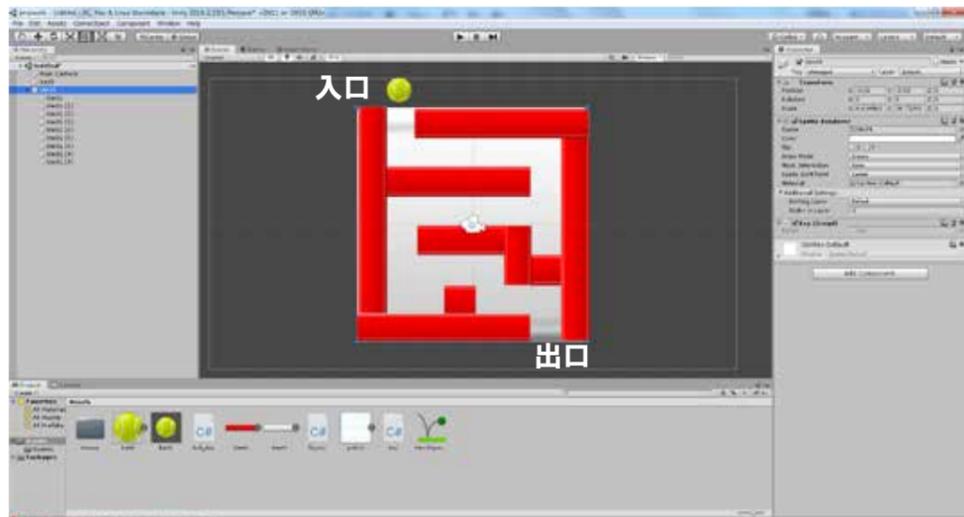
①キー入力で回転させるスクリプトを作成（演習 8と同じ）して「bk\_sita」にアタッチして実行してみます。左の様に bk\_sita だけが回転してしまいます。

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。

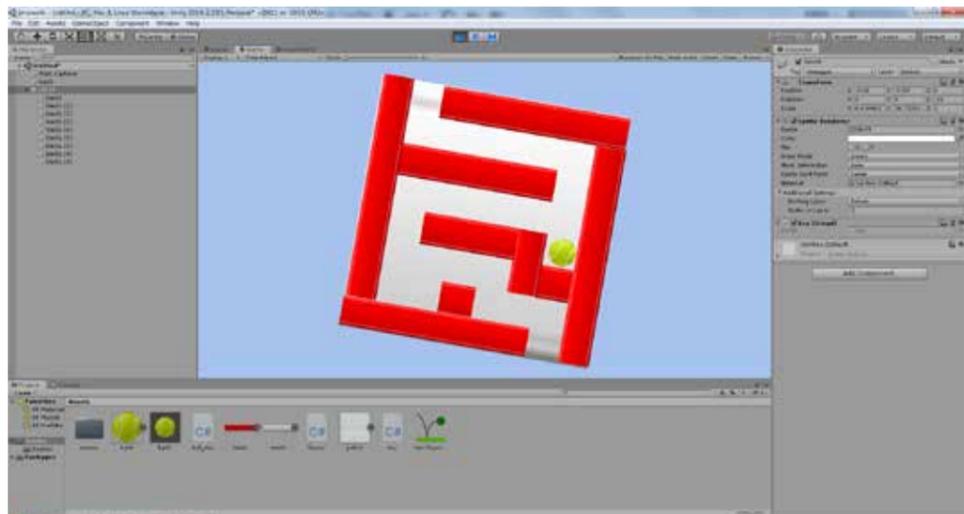
## 演習 11 ブロックを並べて迷路を作り、キー入力で回転させ、転がるボールを出口から出すシーンを作成せよ。



スクリプト	e11_key.cs
アタッチ先	ベースブロック

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

public class e11_key : MonoBehaviour
{
    void Update()
    {
        if (
        {
        }
        if (
        {
        }
    }
}
```

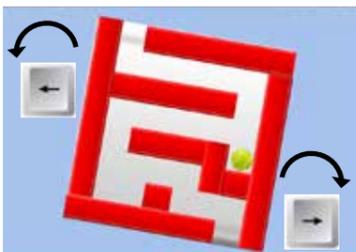


## シーン名 e11maze

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。



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！

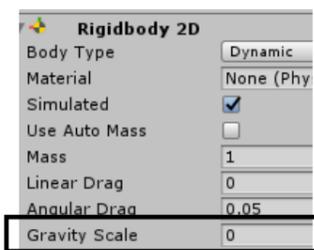
## 例題 17 物理演算によるオブジェクト制御

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。



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。



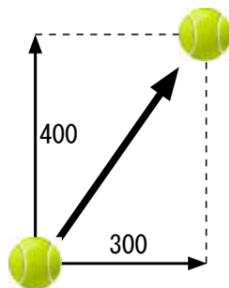
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class r17_force : MonoBehaviour
{
    void Start()
    {
        Rigidbody2D rb = GetComponent<Rigidbody2D>();
        rb.AddForce(new Vector2(300, 400));
    }
}
```

ゲームスタート時に実行

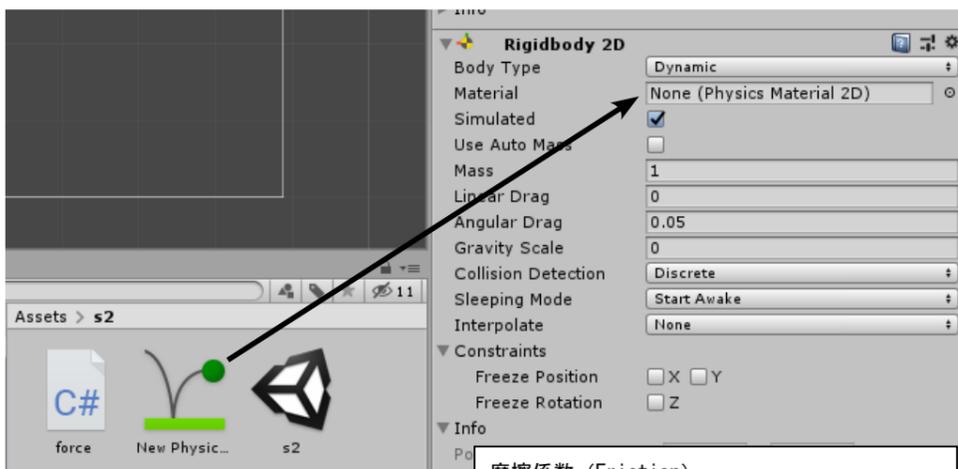
力はX Y軸方向の別々の成分として合成されます。

Vector2(300, 400)



ボールの Rigidbody を取得  
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



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

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いるので、ボールは永久に動き続けます。

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。

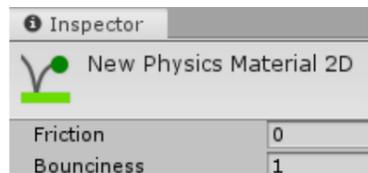
②ボール画像を配置し「Rigidbody2D」と「Circle Collider 2D」コンポーネントをアタッチします。

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！

④「Physics Material」を作成し、下図のように設定します。

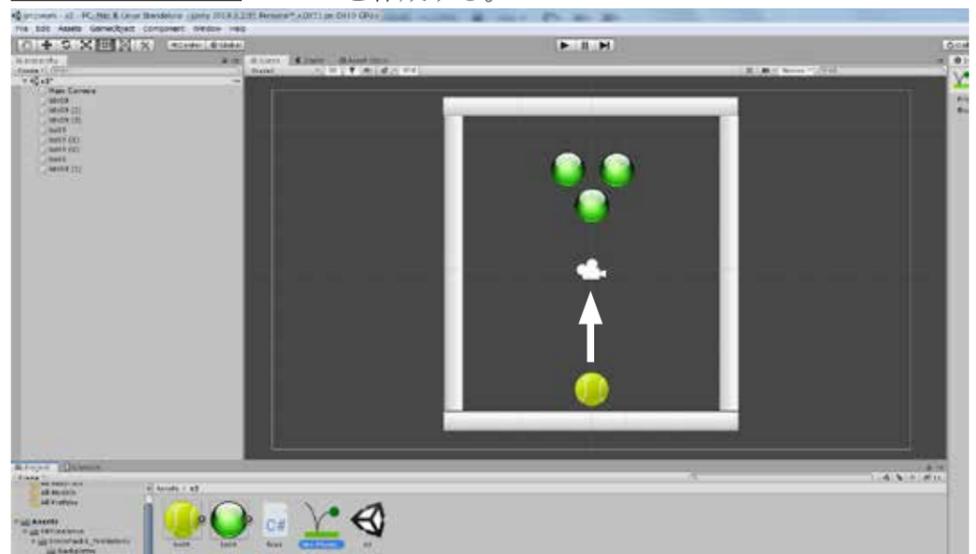
⑤ボールオブジェクトの「Rigidbody 2D」コンポーネントの「Material」にアタッチし、実行します。

このままでは壁に当たっても跳ね返りません…



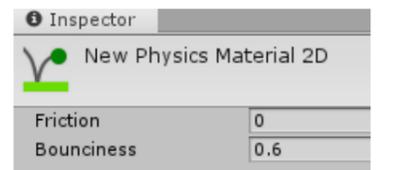
## 演習 12 ビリヤードの玉の動きをシミュレーションするようなシーンを作成する。

シーン名 e12bill



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。

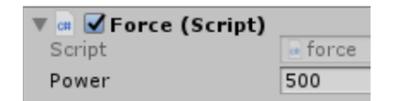


③ボールを1つ配置し「Rigidbody2D」と「Circle Collider2D」をアタッチします。「Rigidbody2D」の「GravityScale」を「0」「Material」に②で作成した「Physics Material」を設定します。

④ヒエラルキーでボールオブジェクトをコピー&ペーストして例のよう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！

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

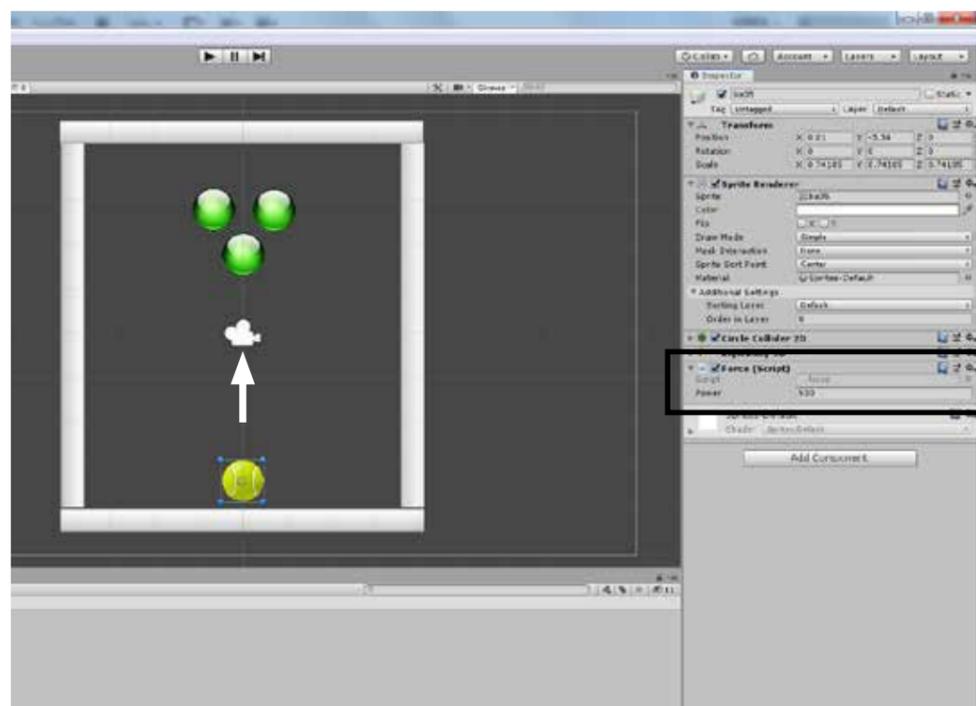
スクリプト	e12_force.cs
アタッチ先	ボール

```
public class e12_force : MonoBehaviour
```

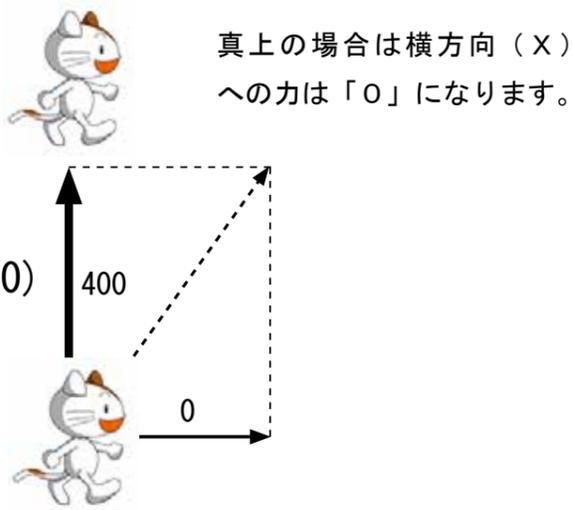
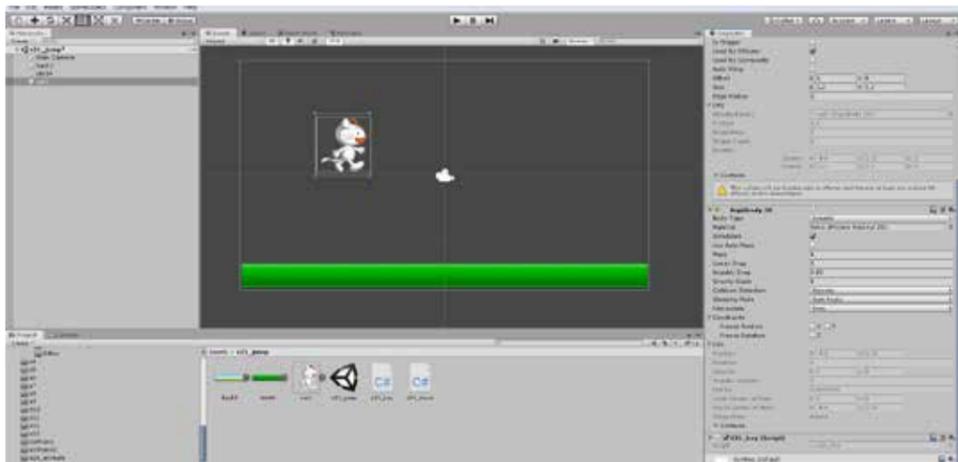
```
{
    public float power; 外部変数 power
```

```
void Start()
```

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



## 演習 13 猫のスプライトをキー入力でジャンプさせる



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

```
public class e13_key : MonoBehaviour
{
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            Rigidbody2D rb = 
            
```

スペースキーが押されたら

Rigidbody2D を取得  
真上方向へ力を加える

## シーン名 e13cat

- 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「Rigidbody2D」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

## 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ことができます。



- ①「例題 17」に例のよう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)
    {
        Destroy(gameObject);
    }
}
```



衝突時に呼ばれる関数（衝突した相手オブジェクト）

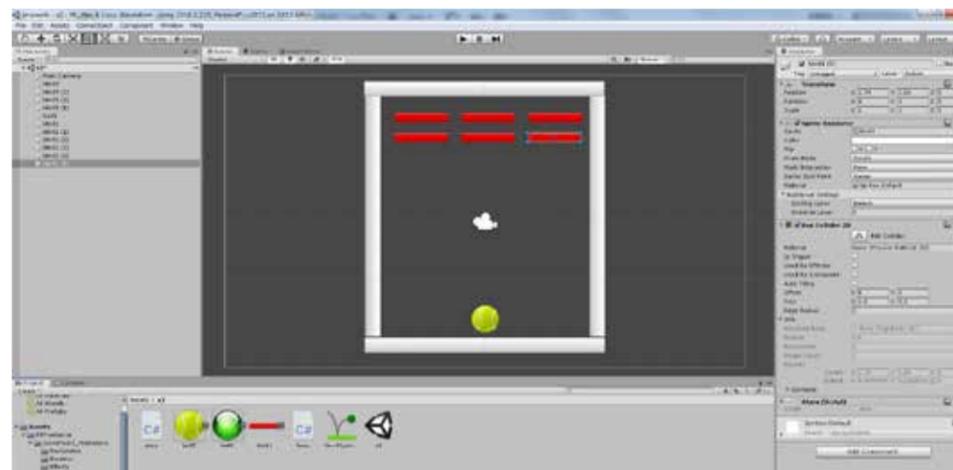
Destroy(gameObject); 自分自身を消す

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。

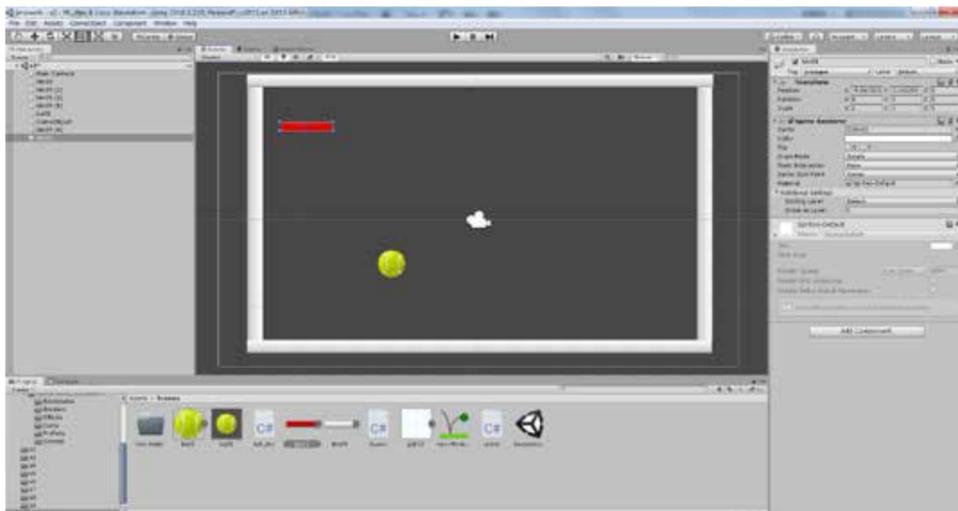
Collision系	跳ね返る
関数	呼び出されるタイミング
OnCollisionEnter	他のcollider/rigidbodyに触れたとき
OnCollisionExit	他のcollider/rigidbodyと触れ合うのをやめたとき
OnCollisionStay	他のrigidbody/colliderに触れている間

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



```
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) {
            for(x=-6.0f; x<=6.0f; x+=2.0f) {
                GameObject bk = Instantiate(block_p);
                bk.transform.position = new Vector3(x, y, 0);
            }
        }
    }
}
```

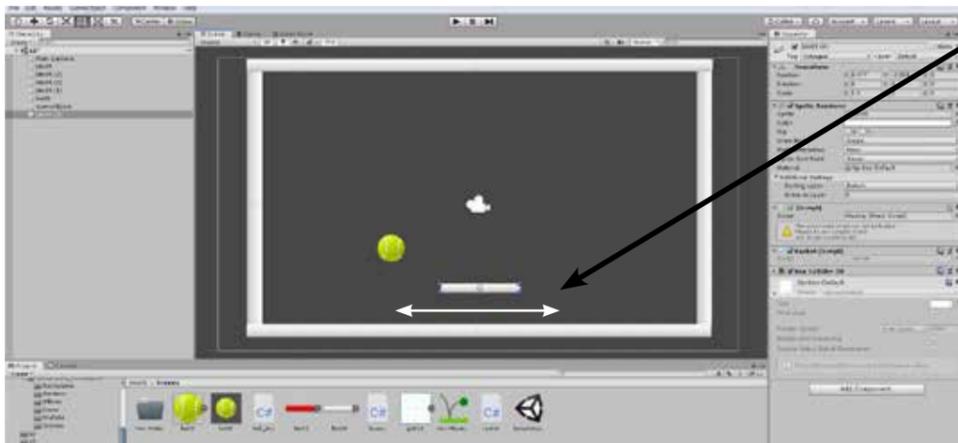
縦方向  
+4 ~ +2  
(5個並べる)

横方向 -6 ~ +6 (7個並べる)

縦 (Y座標) の繰り返し  
横 (X座標) の繰り返し  
ブロックの複製

スクリプト g03\_block.cs  
 アタッチ先 管理オブジェクト

- シーン名 g03blk
- 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「Physics Material」を作成し、「Friction」を「0」、「Bounciness」を「1」に設定します。
  - ⑤1つだけ配置したブロックに「Box Collider 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_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

```
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		BoxCollider2D	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

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！

# 例題 19 Collider による当たり判定



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

public class r19_key : MonoBehaviour
{
    void Update()
    {
        if (Input.GetKey(KeyCode.LeftArrow))
        {
            transform.Rotate(0, 0, 1);
        }
        if (Input.GetKey(KeyCode.RightArrow))
        {
            transform.Rotate(0, 0, -1);
        }
    }
}
```

←キー入力で 左へ1度回転  
→キー入力で 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



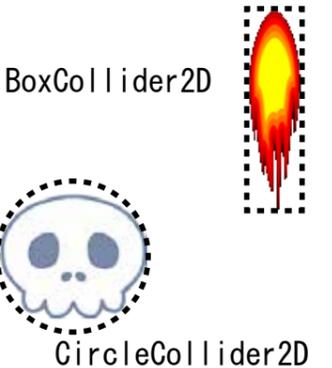
シーン名 r19coll

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。



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。

⑦ヒエラルキーでドクロのオブジェクトをいくつかコピーして配置します。

```
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))
        {
            GameObject be = Instantiate(beem_p, transform.position, transform.rotation);
        }
    }
}
```

スペースキーが押されたら 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

物理挙動 RigidBody2D

Gravity Scale 0

```
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)
    {
        GameObject ex = Instantiate(ex_p, transform.position, Quaternion.identity);
        Destroy(gameObject);
        Destroy(obj.gameObject);
        Destroy(ex.gameObject, 1.0f);
    }
}
```

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。  
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。

オブジェクトの衝突判定 爆発エフェクトの複製  
 ビーム(自分自身)を消す  
 ドクロ(衝突相手)を消す  
 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

# ミニゲーム4 シューティングゲーム



シーン名 g04black

今までの学習を組み合わせ、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ビームと隕石の当たり判定
- 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

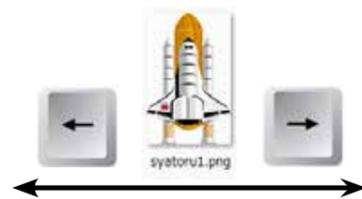
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

スクリプト	g04_key.cs
アタッチ先	宇宙船

```
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。

- 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。

## ②-1 ビームを移動する

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

スクリプト	g04_beem.cs
アタッチ先	ビーム

```
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
    }
}
```

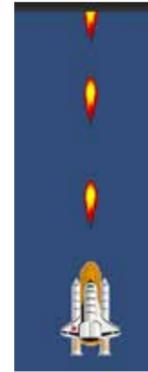


## ②-2 キー入力でビームを発射（連射）する

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

スクリプト	g04_key.cs
アタッチ先	宇宙船

```
public class g04_key : MonoBehaviour
{
    public GameObject beem_p; ビームプレハブの外部変数
    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

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

スクリプト	g04_rock.cs
アタッチ先	隕石（岩石）

```
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。

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

スクリプト	g04_gen.cs
アタッチ先	管理オブジェクト

```
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);
    }
}
```



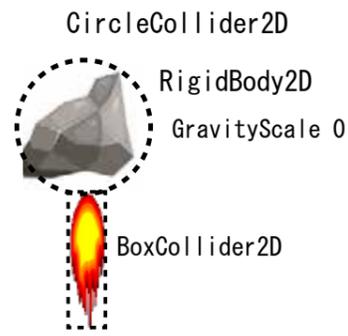
## ⑤ ビームと隕石の当たり判定

```
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);
        Destroy(obj.gameObject);
    }
}
```



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「CircleCollider2D」、隕石のプレハブには「BoxCollider2D」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립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);
        Destroy(obj.gameObject);
        Destroy(ex.gameObject, 1.0f);
    }
}
```

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립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;

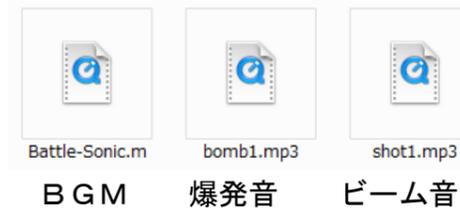
public class g04_back : MonoBehaviour
{
    void Update()
    {
        transform.Translate(0, -0.05f, 0);
        if (transform.position.y < -22.0f)
            transform.position = new Vector3(0, 22.0f, 0);
    }
}
```

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립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립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！

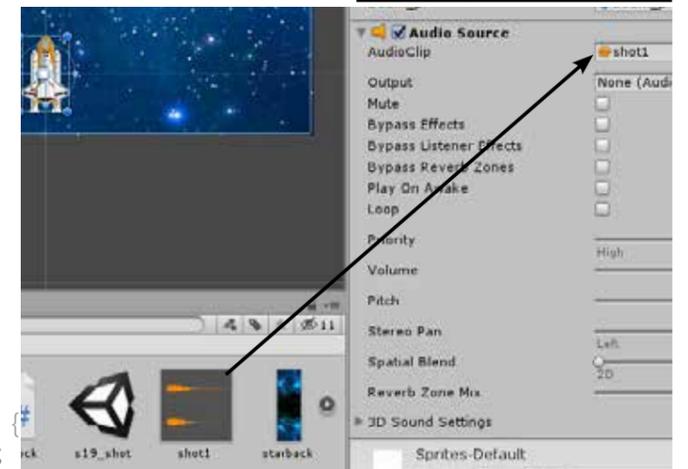
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

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);
        }
    }
}
```

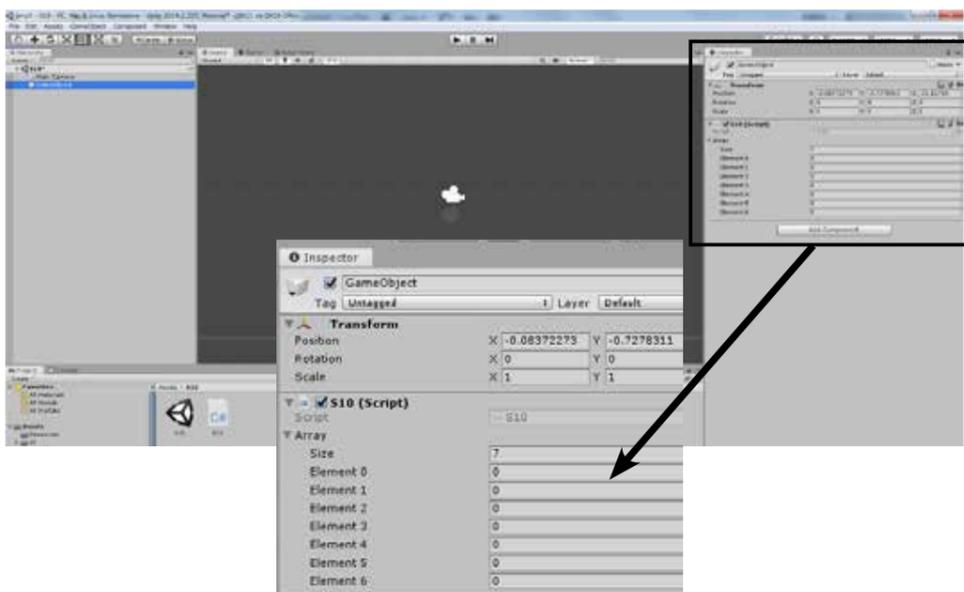
スクリプト	g04_key.cs
アタッチ先	宇宙船



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。

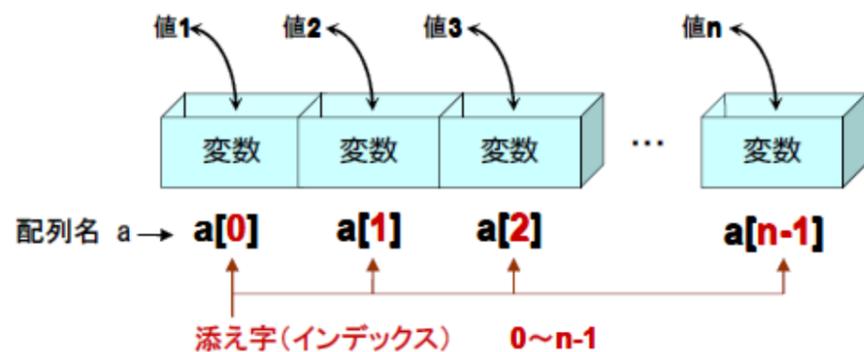


1～6の目をカウントする配列を用意し、全てを「0」に初期設定しておきます。

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できます。

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\_array.cs」をアタッチします。

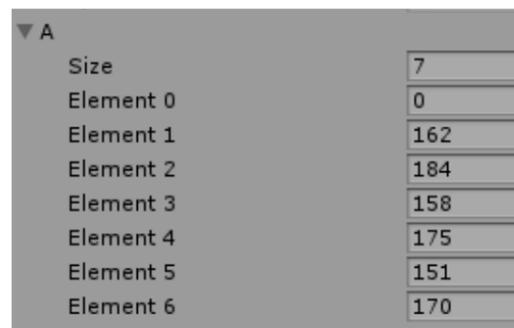
②実行するとインスペクターの外部変数欄にカウントされた数値が表示されます。



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

public class r20_array : MonoBehaviour
{
    public int[] array = { 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++)
        {
            int r = Random.Range(1, 7);
            a[r] = a[r] + 1;
            //Debug.Log(r);
        }
    }
}
```



スクリプト	r20_array.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



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

スクリプト	r21_move.cs
アタッチ先	ボール

```
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);
        }
    }
}
```

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
{
    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

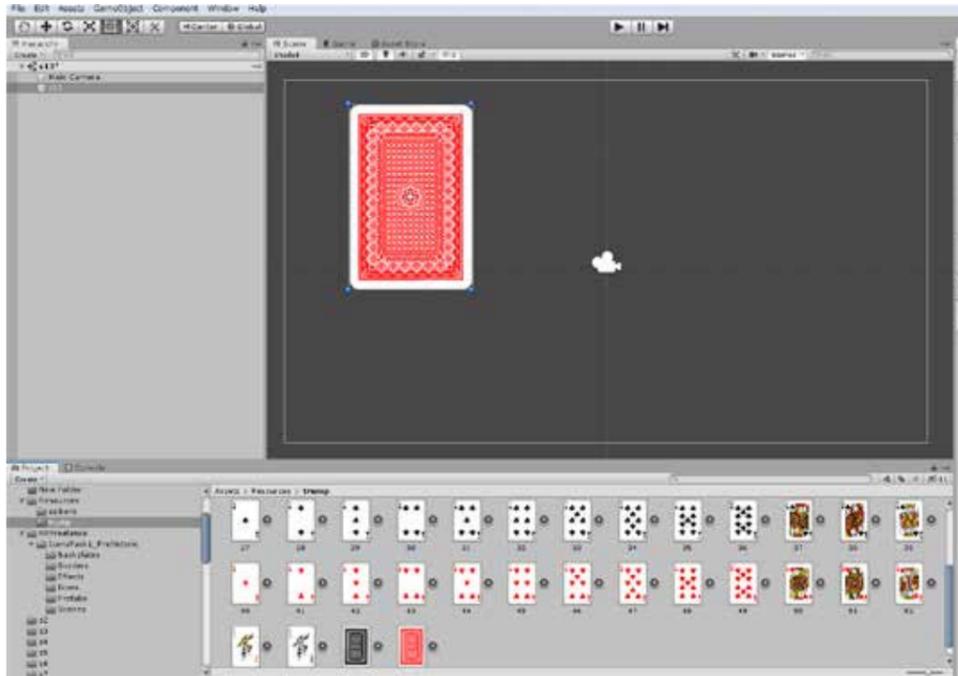
    void Start()
    {
        sr = gameObject.GetComponent<SpriteRenderer>(); // SpriteRenderer コンポーネントを取得
    }

    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
        }
    }
}
```

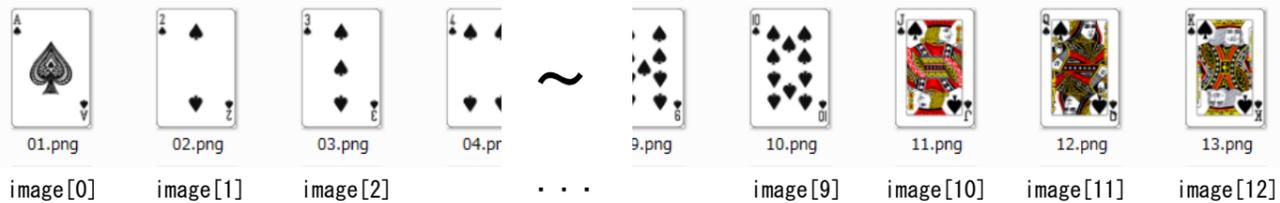
スクリプト	r21_move.cs
アタッチ先	ボール

## 例題 22 多数の画像（スプライト）を配列に読み込む

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。



読み込まれた配列 image の内容



```
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()
```

```
{
    image = Resources.LoadAll<Sprite>("trump");    Assets/Resources/trump フォルダに
    sr = gameObject.GetComponent<SpriteRenderer>();    ある全ての画像を配列に読み込む
    sr.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
    }
}
```

シーン名 r22trump

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 枚を収めた「trump」フォルダをコピーします。

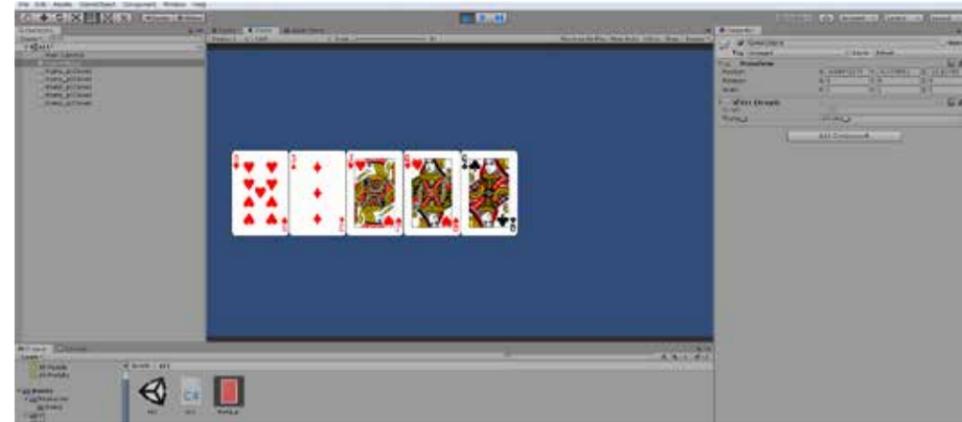
②トランプ裏面の画像を配置し、スケールを調整します。

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。

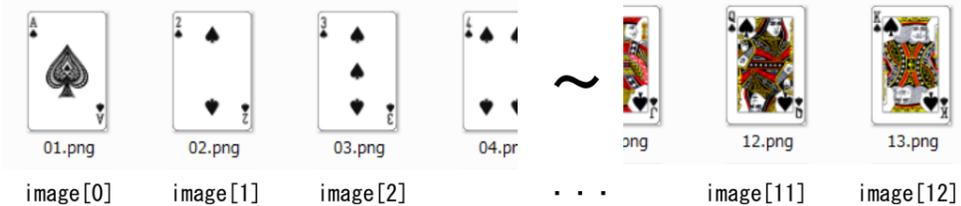
④実行を確認します。

スクリプト	r22_trump.cs
アタッチ先	トランプ

## 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



```
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 (int n = 0; n < 5; n++)    トランプ 5 枚分の繰り返し
        {
            sr = tp[n].GetComponent<SpriteRenderer>();    トランプ配列の SpriteRenderer 取得
            int r = Random.Range(0, image.Length);    トランプ画像をランダムで設定
            sr.sprite = image[r];
        }
    }
}
```



シーン名 e14trump

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 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。

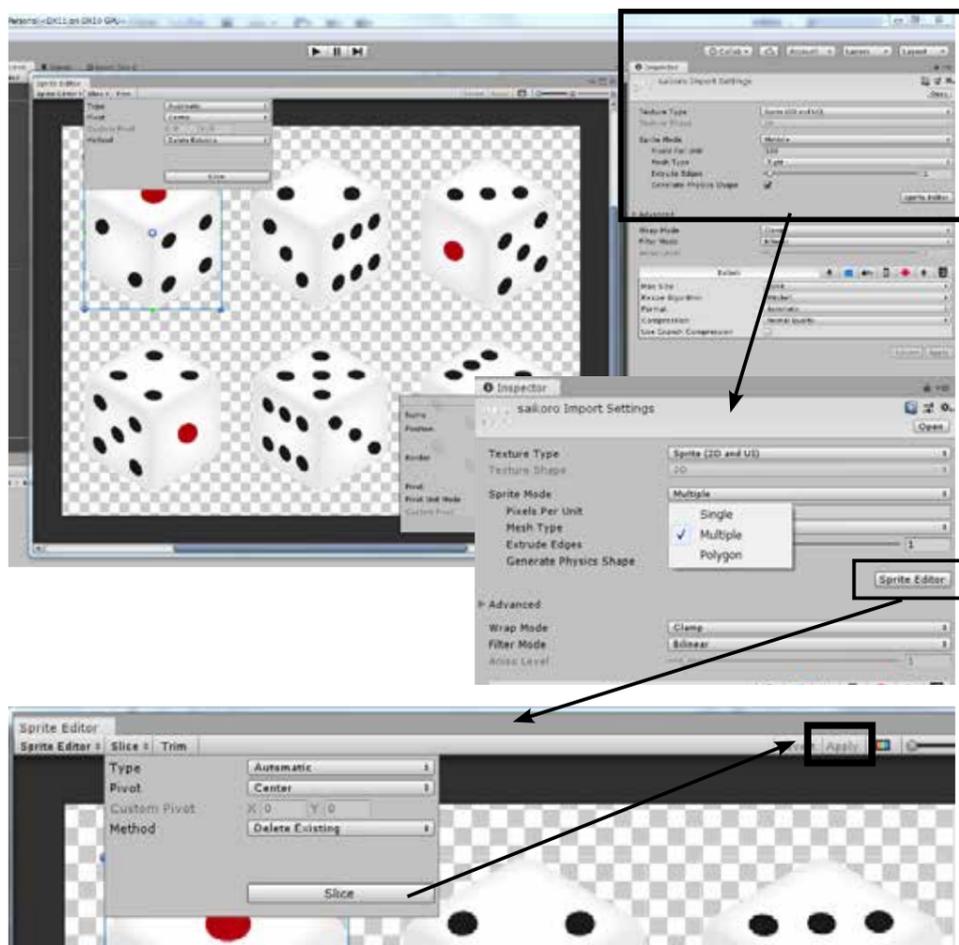
③トランプ 5 枚分のオブジェクトを配列として用意し、読み込んだリソースからランダムでスプライトを設定します。

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

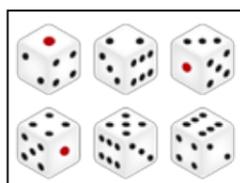
## 例題 23 アトラス画像を切り分け配列に読み込む

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。

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

スクリプト	r23_sai.cs
アタッチ先	サイコロ

```
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()
```

```
{
    image = Resources.LoadAll<Sprite>("saikoro");    Assets/Resources/trump フォルダに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()
```

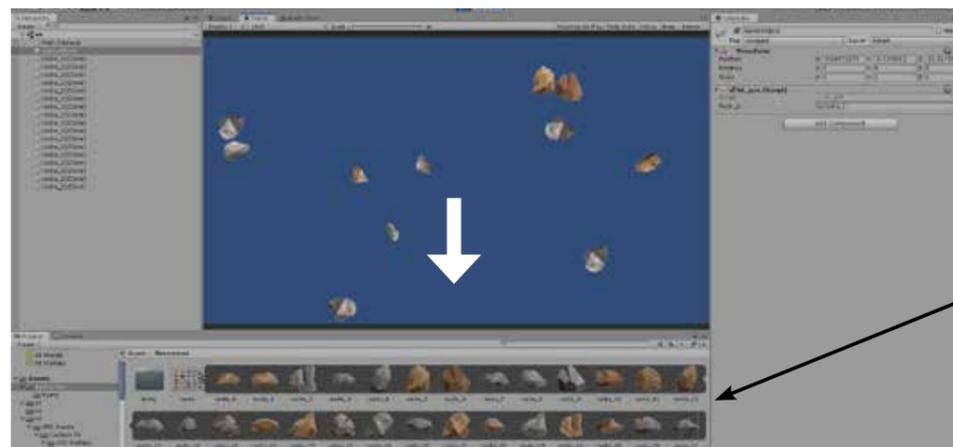
```
{
    if (Input.GetMouseButton(0))    マウスボタンが押されている間...
    {
        int r = Random.Range(0, image.Length);    0~5の乱数を発生
        sr.sprite = image[r];    該当するサイコロ画像をスプライトに設定
    }
}
```

- 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

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

スクリプト	r24_move.cs
アタッチ先	岩石

```
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
}
```

- 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。

```
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);
    float x = Random.Range(-11.0f, 11.0f);
    rk.transform.position = new Vector3(x, 6, 0);
    int r = Random.Range(0, image.Length);
    sr = rk.gameObject.GetComponent<SpriteRenderer>();
    sr.sprite = image[r];
}
```

プレハブからスプライトを複製  
横位置は乱数

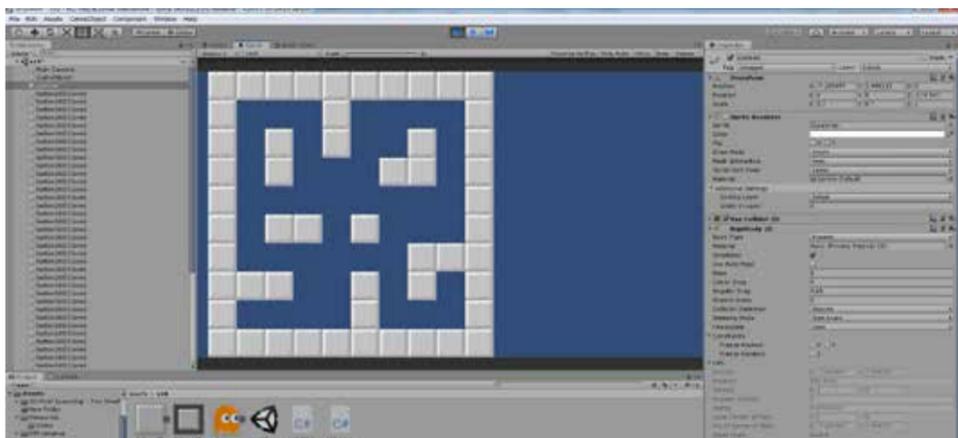
岩の種類（配列番号）をランダムで設定

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

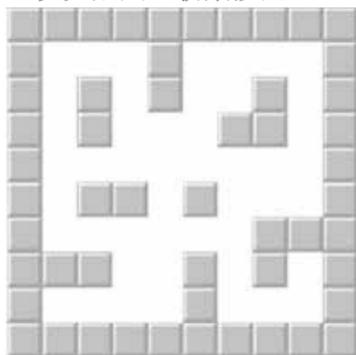
- ①ボタン画像「button100.png」を配置する。
- ②オブジェクト「button100」をヒエラルキーウインドからプロジェクトウインドへドラッグ「プレハブ化」し、名前を「button100\_p」と変更する。
- ③元のオブジェクト「button100」を削除する。

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

```
public class r25_array : 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},
};
```

2次元配列の初期設定



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);
```

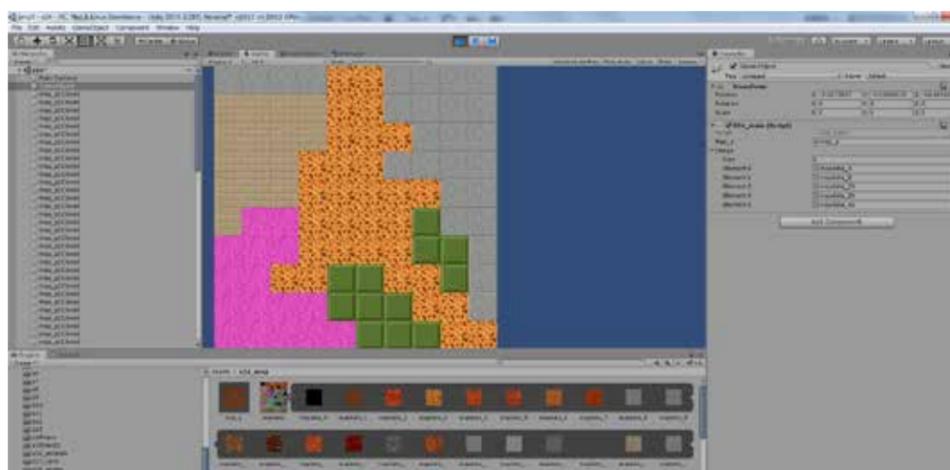
```
                kb.transform.position = new Vector3(x-8, 4.5f-y, 0);
```

プレハブをコピー  
指定位置に配置

スクリプト	r25_array.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。

```
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 = {
    { 3, 3, 3, 28, 28, 28, 3, 3, 3, 3},
    { 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},
};
```

2次元配列の初期設定  
マップのパターン番号

```
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]];
```

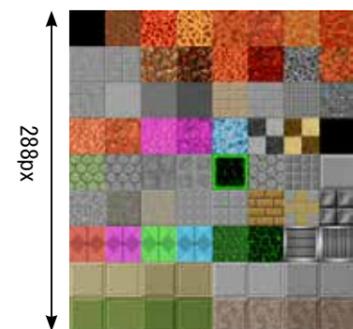
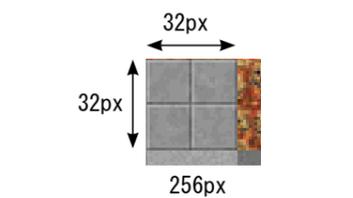
マップのイメージをスプライトに設定

```
            mc.transform.position = new Vector3(x-8, 4.5f-y, 0);
```

指定位置に配置

スクリプト	r26_gen.cs
アタッチ先	管理オブジェクト

スクリプト	r26_gen.cs
アタッチ先	管理オブジェクト



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	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
64	65	66	67	68	69	70	71

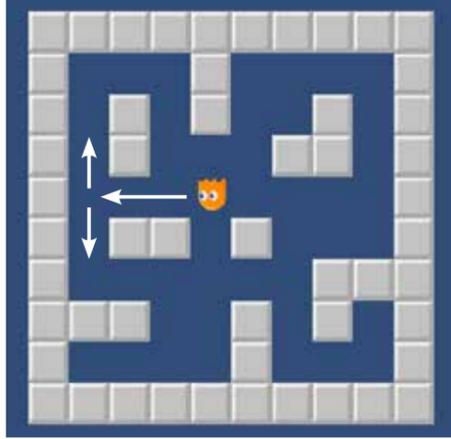
## 演習 15 迷路の中をキャラクターが自動で動き回るシーンを作成してみましょう。

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

スクリプト	e15_main.cs
アタッチ先	管理オブジェクト

```
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);
            }
        }
    }
}
```

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

スクリプト	e15_move.cs
アタッチ先	キャラクター

```
public class e15_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
}
}
```

シーン名 e15maze

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キャラクターをシーンに配置し「BoxCollider2D」と「RigidBody2D」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。



BoxCollider2D  
RigidBody2D

迷路を表示するスクリプトは例題 25 と同じです。

③キャラクターを自身から見て、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。

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

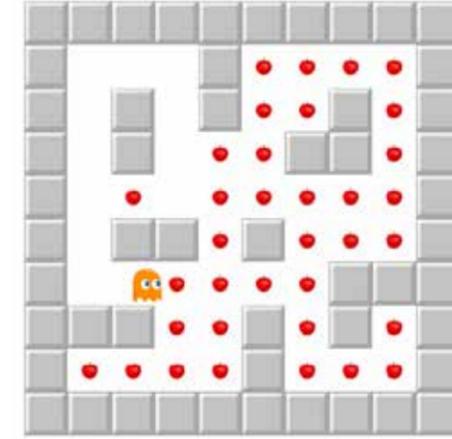
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

スクリプト	r27_main.cs
アタッチ先	管理オブジェクト

```
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;
            }
        }
    }
}
```

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 14 をベースに作成します。

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。

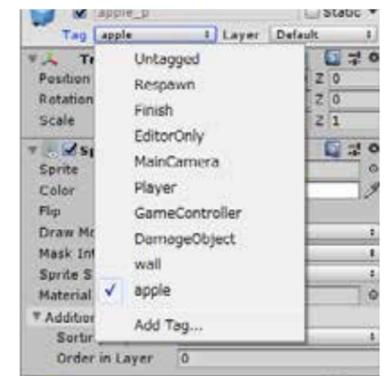
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

スクリプト	r27_move.cs
アタッチ先	キャラクター

```
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{
            transform.Translate(-0.1f, 0, 0);  そうでなければ(壁ならば)
            int r = Random.Range(1, 4);        方向を変える
            transform.Rotate(0, 0, r * 90);
        }
    }
}
```

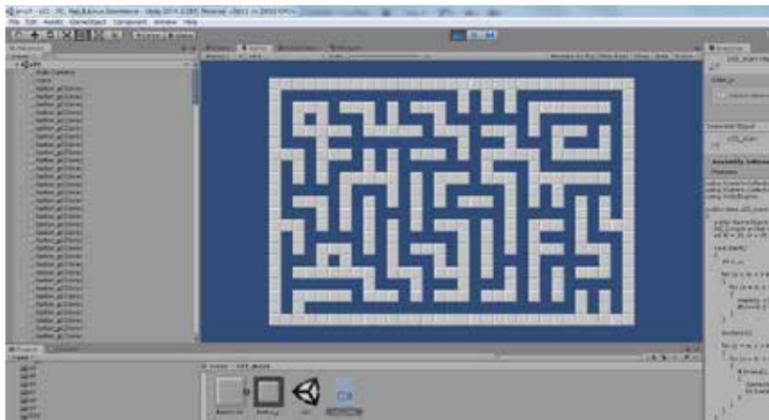


③オブジェクトを識別するためには「タグ」使います。「Add Tag」からタグに「apple」を追加し、設定します。

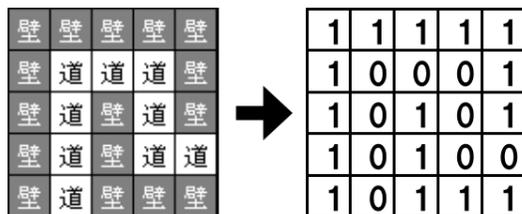
④衝突時のスクリプトでタグを判定して処理します。

# 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。



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。

```
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;

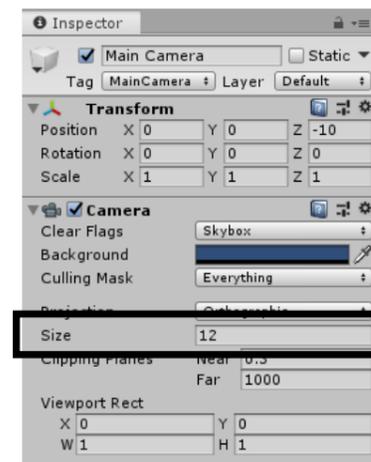
    void Start()
    {
        int x, y;

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

        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);
                }
            }
        }
    }
}
```

スクリプト	r28_main.cs
アタッチ先	管理オブジェクト

- ①ボタン画像「button100.png」を配置する。
- ②オブジェクト「button100」をヒエラルキーウィンドからプロジェクトウィンドへドラッグ「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 || x==W-1)
                    maze[y,x]=1;
            }
        }

        void boutaosi(); // 迷路生成関数呼び出し

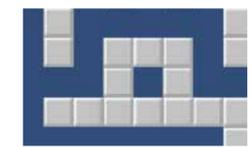
        for (y = 0; y < H; y++)
        {
            for (x = 0; x < W; x++)
            {
                if (maze[y, x] == 1)
                {
                    GameObject kb = Instantiate(kabe_p);
                    kb.transform.position = new Vector3(x-15, 10-y, 0);
                }
            }
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

        void boutaosi() // 棒倒し法による迷路生成
        {
            int x, y, r;
            Random.InitState(System.DateTime.Now.Millisecond);
            for (y = 2; y <= H-3; y += 2)
            {
                for (x = 2; x <= W-3; x += 2)
                {
                    maze[y, x] = 1;
                    r = Random.Range(0, 4);
                    switch (r)
                    {
                        case 0: maze[y - 1, x] = 1; break;
                        case 1: maze[y + 1, x] = 1; break;
                        case 2: maze[y, x - 1] = 1; break;
                        case 3: maze[y, x + 1] = 1; break;
                    }
                }
            }
        }
    }
}
```

**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。

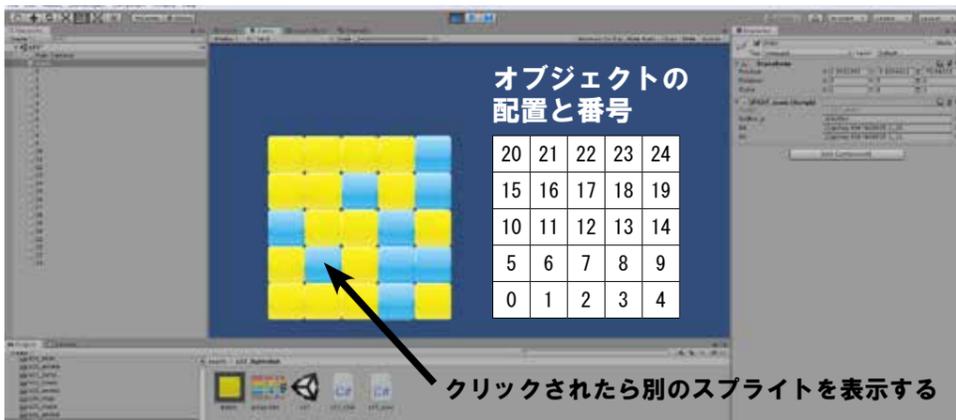
1. まず、外周の壁と1つ飛ばしに棒を倒す対象となる基点の壁があります。(「□」が基点の壁になります)
2. 左上の基点から右の基点に向けて上下左右にランダムに棒(壁)を倒します。
3. 続けて、2行目以降も棒を倒しますが、2行目以降については上方向以外にランダムに棒を倒します。(発展課題の部分)
4. 3行目も手順3と同様に棒を倒します。
5. 一番下の行まで棒を倒し終わったら迷路の完成です。



←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

## 例題 29 オブジェクト配列のクリック取得



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

スクリプト	r29_main.cs
アタッチ先	管理オブジェクト

```
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
        }
    }
}
```

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

スクリプト	r29_click.cs
アタッチ先	ボタン プレハブ

```
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<s27_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。

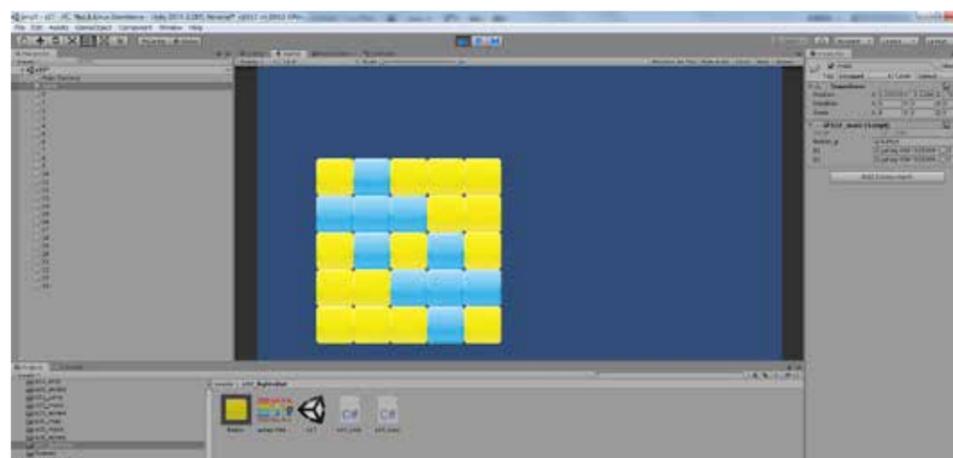
- ⑤メインの外部変数にボタンのスプライトを2つ登録します。



- 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個のライトと一緒に反転する。

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

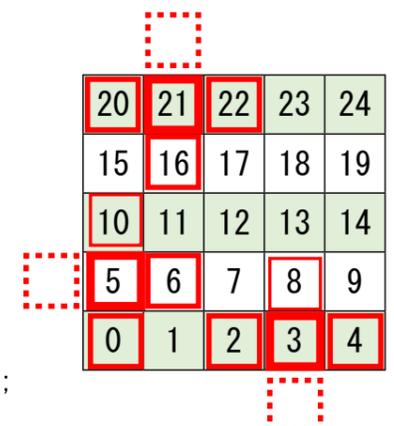
スクリプト	g05_main.cs
アタッチ先	管理オブジェクト

```
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)

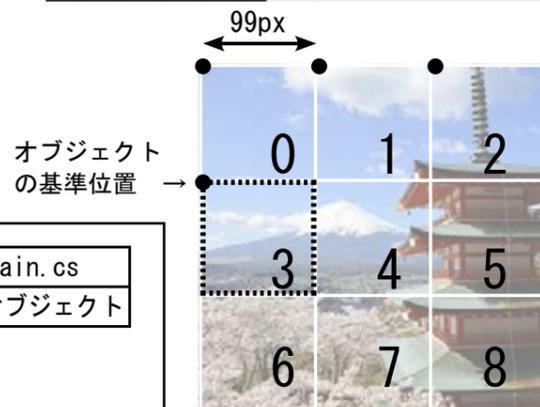
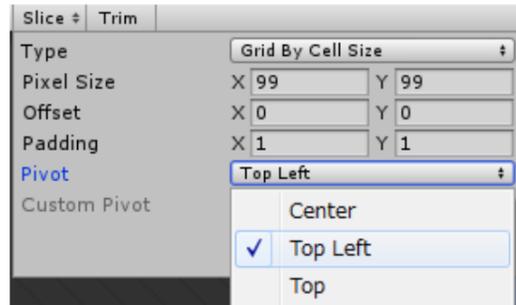
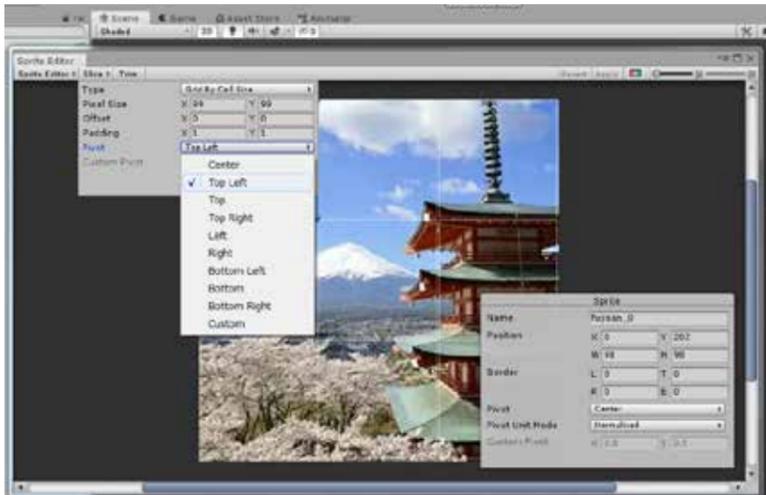
シーン名 g06puzz



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ます (例では 300px × 300px)

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。



```
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
        }
    }
}
```

スクリプト	g06_main.cs
アタッチ先	管理オブジェクト

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。



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。

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。

```
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);
    }
}
```

スクリプト	g06_click.cs
アタッチ先	ピースプレハブ

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。

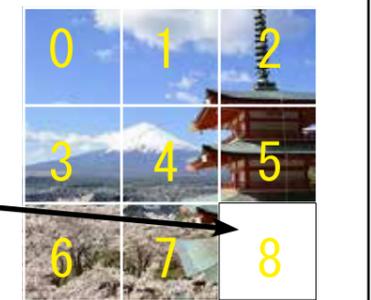
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

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();
        }
    }

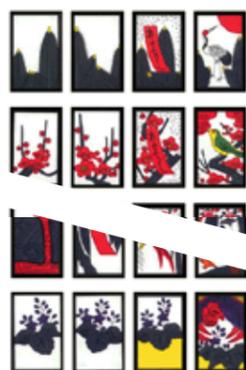
    public void click(int n)
    {
        float x = p[n].transform.position.x;           ピースのX位置 (左上)
        float y = p[n].transform.position.y;           ピースのY位置 (左上)
        int a = (int)(-(y * 3.0f) + x);                 ピースが現在ある場所 (番号) を求める
        if (a - 1 == aki || a + 1 == aki || a - 3 == aki || 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
        }
    }
}
```

スクリプト	g06_main.cs
アタッチ先	管理オブジェクト



### 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。



hanafuda.png

シーン名 r30hana

- 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。

```
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);
    sr[n] = hf[n].GetComponent<SpriteRenderer>();
    sr[n].sprite = image[n];
    hf[n].transform.position = new Vector3((n%12)*1.3f-7.0f, (n/12)*2-3, 0);
}
```

札のプレハブ  
札の配列  
札のスプライト (画像) の配列  
札画像を Resources から読み込むための配列

Resources から札画像を読み込む

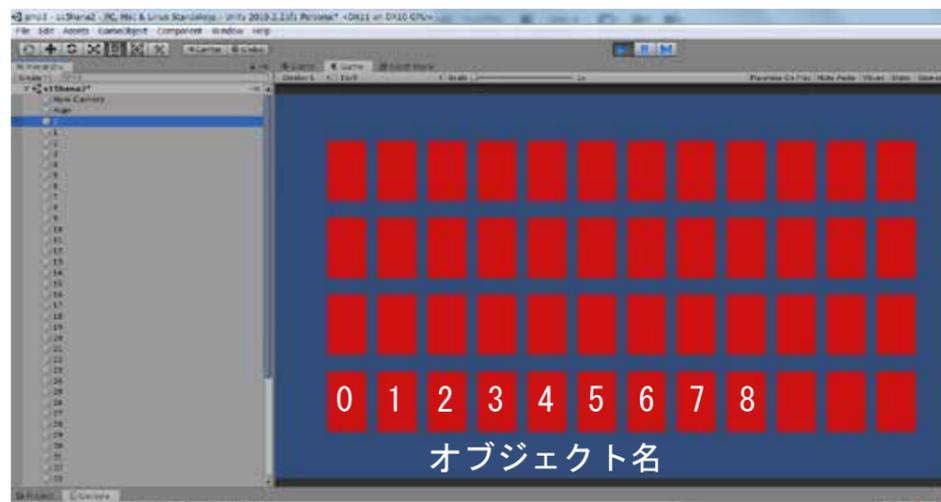
48枚 (4枚×12か月)

札のインスタンス生成  
札の SpriteRendeer を取得  
札のスプライト (画像) を設定  
札を12×4に並べる

スクリプト	r30_gen.cs
アタッチ先	管理オブジェクト



- 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。

```
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();
    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)
```

```
{
    sr[n].sprite = image[n];
}
```

スクリプト	r30_gen.cs
アタッチ先	管理オブジェクト



オブジェクト名を配列番号 (文字列) に設定

外部からアクセス可能 (public) なメソッド

受け取った n 番の画像 (スプライト) を表示

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

```
public class r30_click : MonoBehaviour
```

```
{
    void OnMouseDown()
```

```
{
    GameObject obj = GameObject.Find("main");
    int n = int.Parse(gameObject.name);
    obj.GetComponent<r29_gen>().card(n);
}
```

```
}
    絵柄表示メソッド card() を呼び出す
```

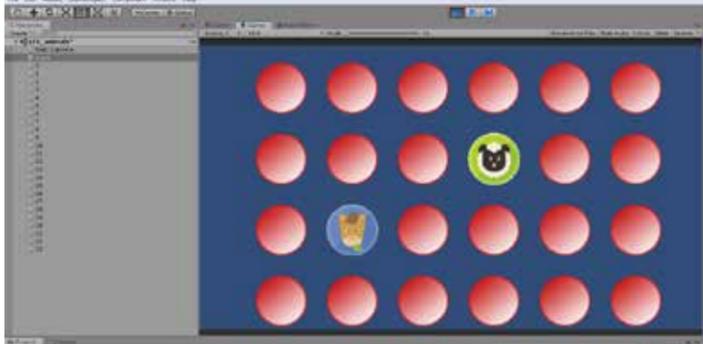
スクリプト	r30_click.cs
アタッチ先	花札プレハブ

- 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

例題〇〇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枚ずつ表示するように設定する。

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

animals.png



## image[] 配列 (Sprite型)

0 1 2 3 4 5 6 7 8 9 10 11



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

スクリプト	g07_main.cs
アタッチ先	管理オブジェクト

```
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
    int card1, card2;                 1, 2枚目の可0度を記憶
    SpriteRenderer sr1, sr2;          カードの SpriteRenderer
```



0 1 2 3 4 5 6  
オブジェクト名

```
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
```

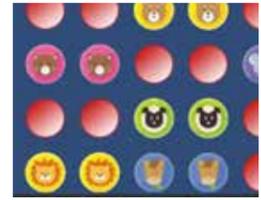
```
{
    card[n] = Instantiate(card_p); カードを複製しオブジェクト配列に入れる
    card[n].name = n.ToString(); オブジェクトの名前を番号に変更
    s17_click sp = card[n].GetComponent<s17_click>(); クリックのスクリプトを取得
    sp.back = image[n / 2]; スクリプトの外部変数 back にスプライトを設定
    //sr1 = card[n].GetComponent<SpriteRenderer>();
    //sr1.sprite = 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
}
```

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

スクリプト	g07_click.cs
アタッチ先	カードプレハブ

```
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
    }
}
```



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。

## 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

...省略

スクリプト	g07_click.cs
アタッチ先	カードプレハブ

```
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(obj.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
    }
}
```

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。

...省略

クリックされたカード番号を受取る

スクリプト	g07_main.cs
アタッチ先	管理オブジェクト

```
public void card_turn(int n)
{
    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 "
        }
        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。

```
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;

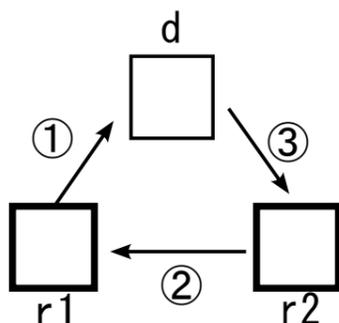
    void Start()
    {
        image = Resources.LoadAll<Sprite>("animals");

        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);
        for (int n = 0; n < 24; n++)
        {
            int r1 =Random.Range(n, 24);
            d.transform.position = card[r1].transform.position;
            card[r1].transform.position = card[n].transform.position;
            card[n].transform.position = d.transform.position;
        }
        Destroy(d);
    }
}
```

スクリプト	g07_main.cs
アタッチ先	管理オブジェクト

2つ変数の内容を交換



GameObject d = Instantiate(card\_p);      オブジェクトの交換用の変数を用意

int r1 =Random.Range(n, 24);      交換する配列を乱数で決める

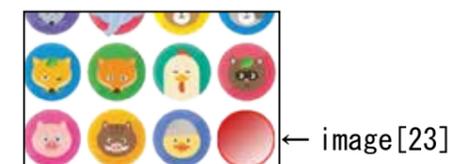
d.transform.position = card[r1].transform.position;      カードのポジションを交換

Destroy(d);      交換用の変数を消去

次ページへ続く

```
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];
    sr2.sprite = image[23];
}
}
```



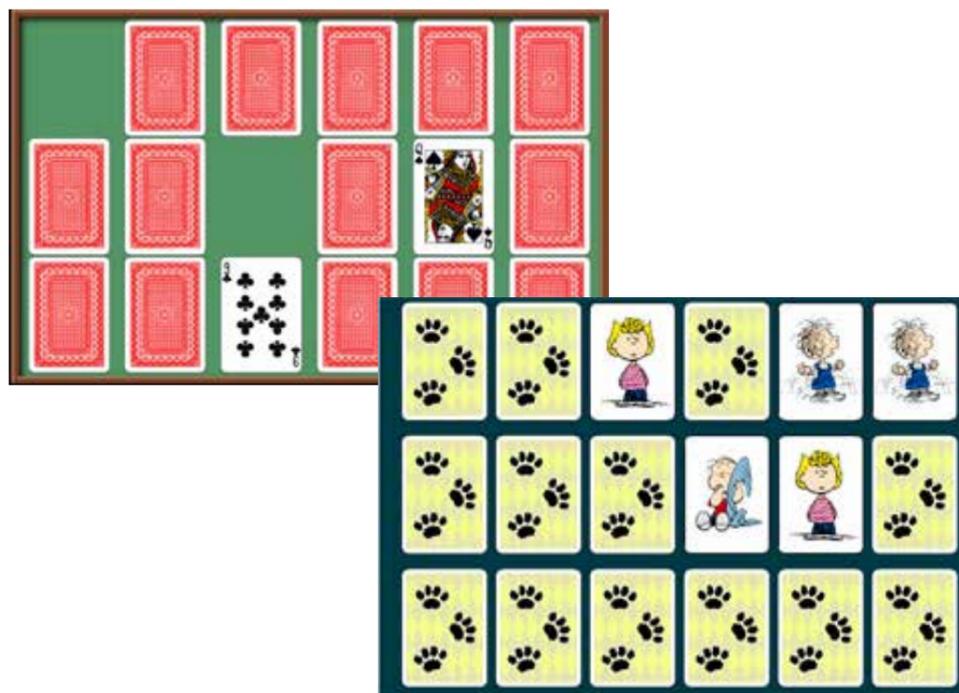
← image[23]      カードのSpriteを表面に戻す

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

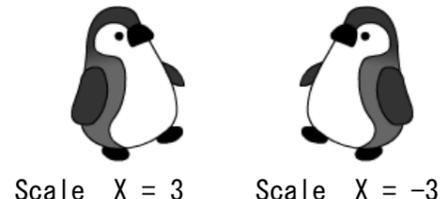
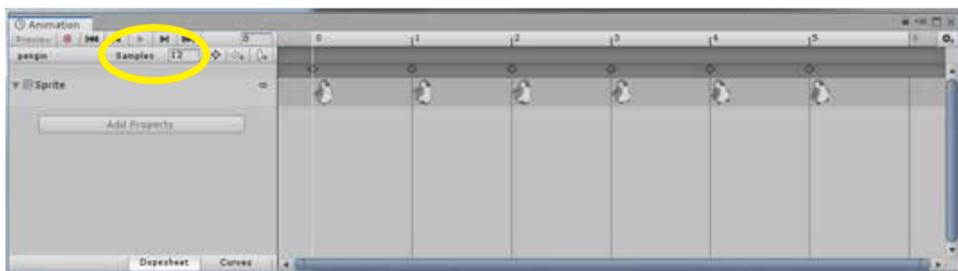
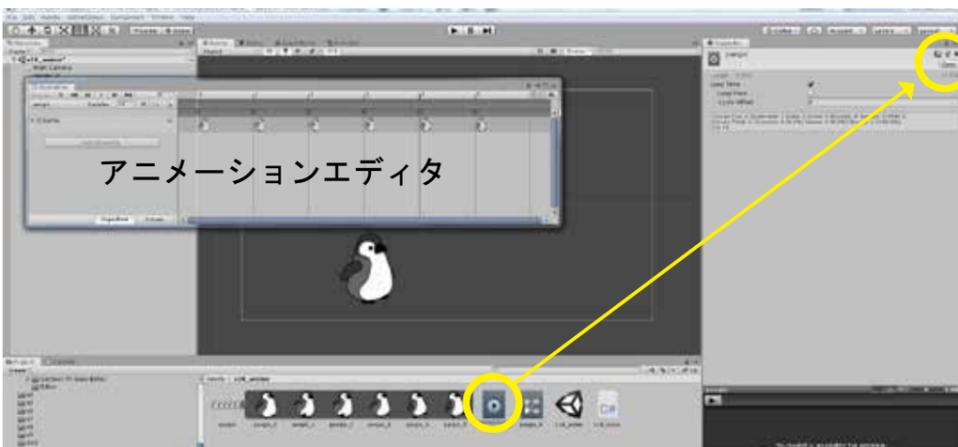
⑪カードが一致した場合には1秒後に消去されますが、一致しなかった場合にはすぐにもとにもどってしまいます。

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



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Sprite エディタで切り分けます。
- ②切り分けた画像を全て選択、まとめてシーンヘッダにドラッグ&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ィター」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。



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

スクリプト	r31_back.cs
アタッチ先	背景

```
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
    }
}
```

「← →」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。

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。

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

スクリプト	r31_key.cs
アタッチ先	ペンギン

```
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
    }
}
```

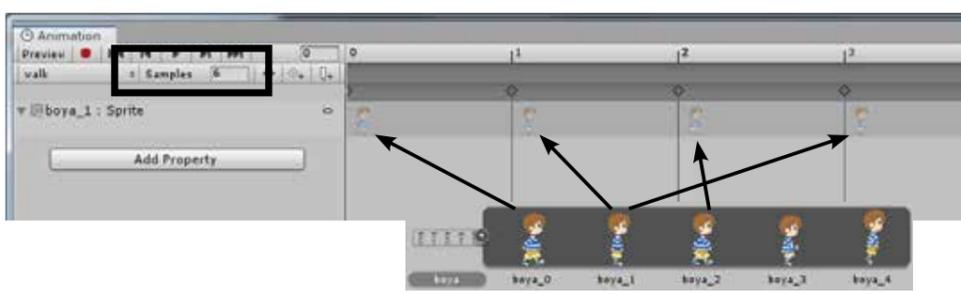
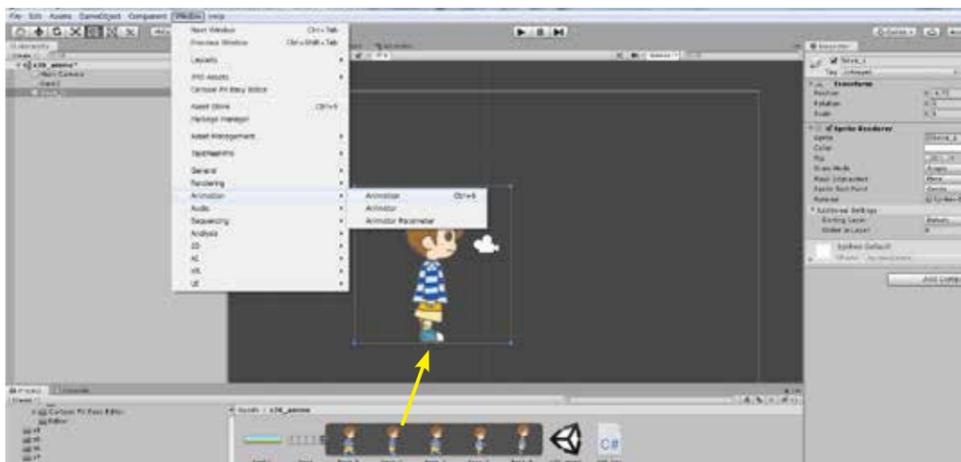
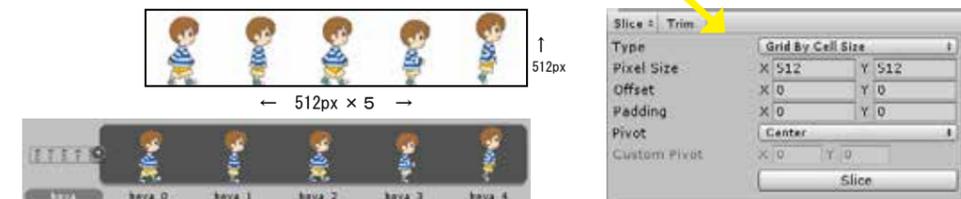
⑨キー入力でアニメーションの再生スピードを変化させるスクリプトを作成し、ペンギンオブジェクトをアタッチします。

アニメーションスピード



※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



4つの画像を順番に連続して表示します

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。

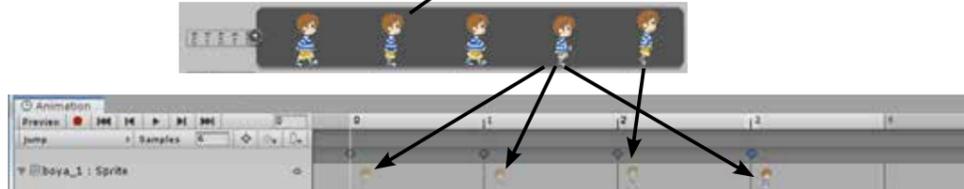
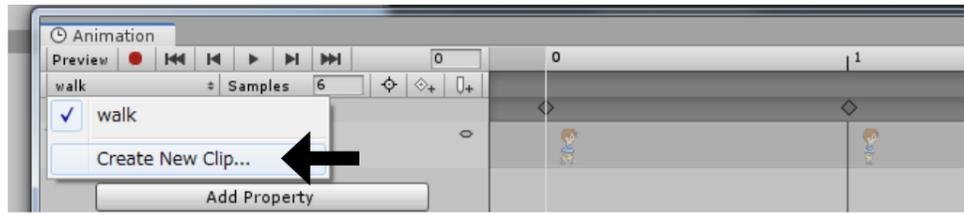
「walk」と名前をつけてアニメーションクリップを保存します。

ファイル名(N): walk  
ファイルの種類(T): anim (\*.anim)

③ 「Samples」を「6」に設定し、タイムラインに歩い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

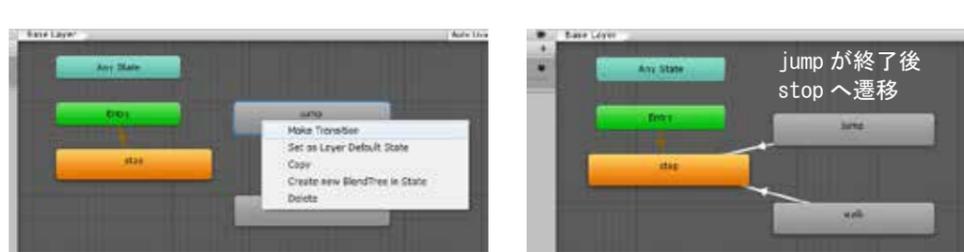
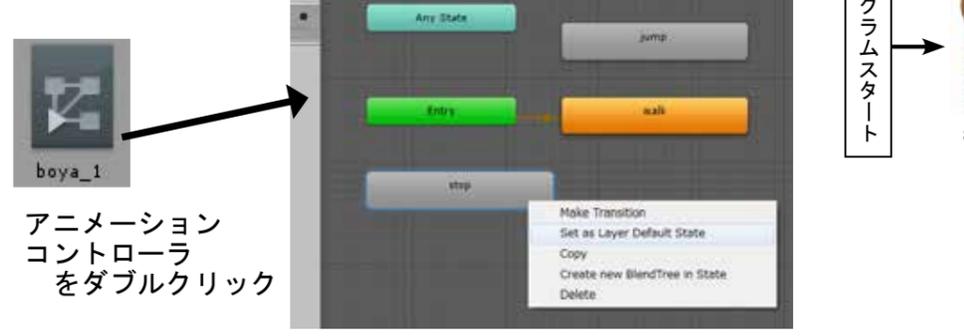
```

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。

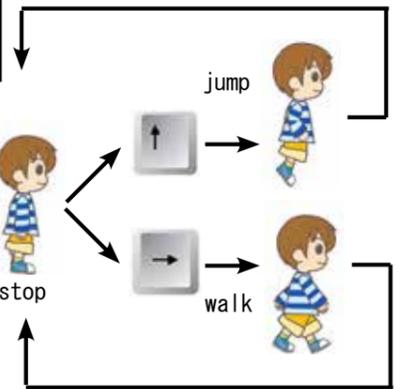
⑥ 「walk」の文字上でクリックし「Create New Clip」を選択し、「stop」と名前を付けます。

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。

下図のよう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



## ④スペースキー入力でジャンプする

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

スクリプト	g08_main.cs
アタッチ先	管理オブジェクト

```
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

```
...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
}
```

スクリプト	g08_main.cs
アタッチ先	管理オブジェクト

シーン名 g08yoko

「例題3 1」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3 1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。

## ⑥背景を横にスクロール

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

スクリプト	g08_main.cs
アタッチ先	管理オブジェクト

```
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に、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

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

スクリプト	g08_main.cs
アタッチ先	管理オブジェクト

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

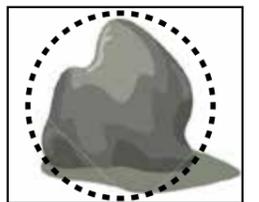
    float bk_speed;

    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。

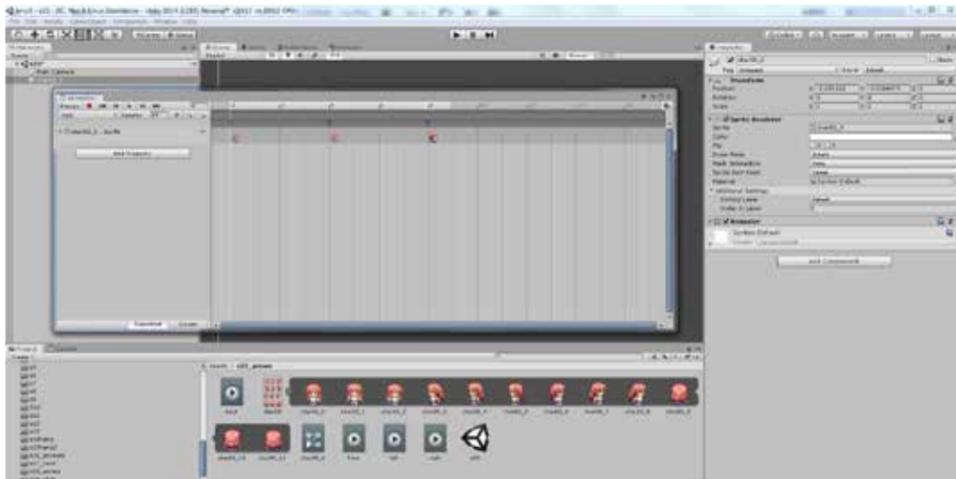


背景のスクロールスピードに合わせて、右から左に移動させます。

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 16 キャラクターのアニメーションを切り替えながら4方向へ移動する



スクリプト	e16_key.cs
アタッチ先	キャラクター

```

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

public class e16_key : MonoBehaviour
{
    void Update()
    {
        var ac = GetComponent<Animator>();

        if (Input.GetKey(KeyCode.UpArrow)) {
            ac.Play("back");
            transform.Translate(0, 0.01f, 0);
        }
        if (Input.GetKey(KeyCode.DownArrow)) {
            ac.Play("front");
            transform.Translate(0, -0.01f, 0);
        }
        if (Input.GetKey(KeyCode.LeftArrow)) {
            ac.Play("left");
            transform.Translate(-0.01f, 0, 0);
        }
        if (
    
```

## シーン名 e16anime

① 4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、キャラクター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 17 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。



スクリプト	e17_key.cs
アタッチ先	キャラクター

```

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

public class e17_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)) 向きを元に
                ac.Play("Run");  ←SHIFT キーが押下
            else 再生
                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)) 向きを反転
                ac.Play("Run");  ←SHIFT キーが押下
            else 再生
                ac.Play("Walk"); 「Walk」を再生
        }
        if (
    
```

## シーン名 e17anime

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。これらを使用す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。

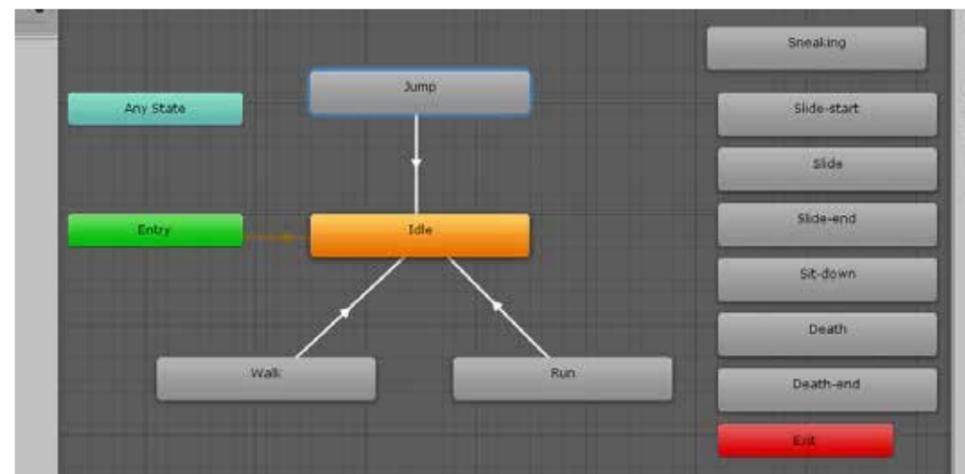
② 「Prefab」フォルダにあるプレハブをシーンに配置し実行すると、登録されているアニメーションが順番に再生されます。

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。

③ Animation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



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

スクリプト	g09_main.cs
アタッチ先	管理オブジェクト

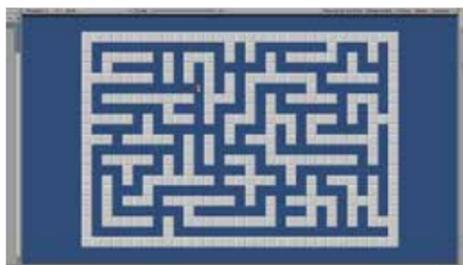
```
public class g09_main : MonoBehaviour
{
    public GameObject kabe_p;
    int[,] maze = new int[100,100];
    int W = 31, H = 21;

    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);
                }
            }
        }
    }
}
```



カメラオブジェクト size=12

```
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
            }
        }
    }
}
```

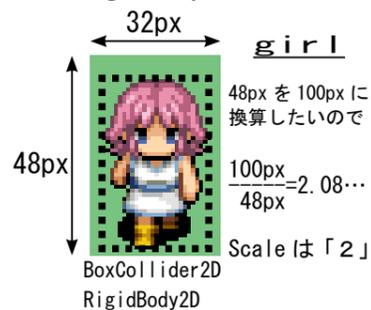
シーン名 g09maze

迷路を歩いて移動するゲームベースを作成します。

- ① 4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。



- 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

```
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)) {
            ac.Play("back");
            transform.Translate(0, 0.05f, 0);
        }

        if (Input.GetKey(KeyCode.DownArrow)) {
            ac.Play("front");
            transform.Translate(0, -0.05f, 0);
        }

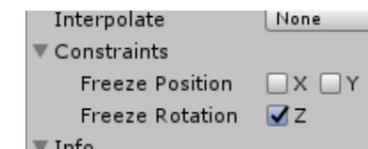
        if (Input.GetKey(KeyCode.LeftArrow)) {
            ac.Play("left");
            transform.Translate(-0.05f, 0, 0);
        }

        if (Input.GetKey(KeyCode.RightArrow)) {
            ac.Play("right");
            transform.Translate(0.05f, 0, 0);
        }
    }
}
```

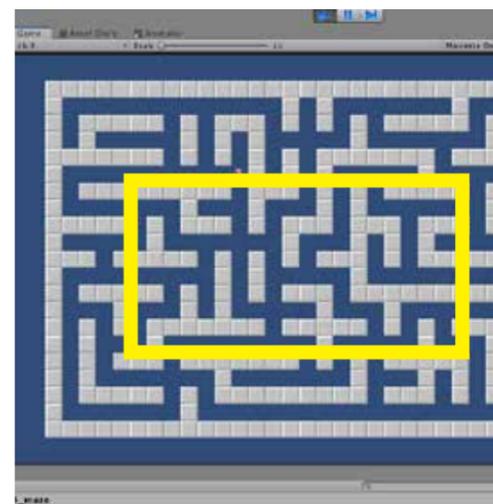
スクリプト	g09_key.cs
アタッチ先	キャラクタ



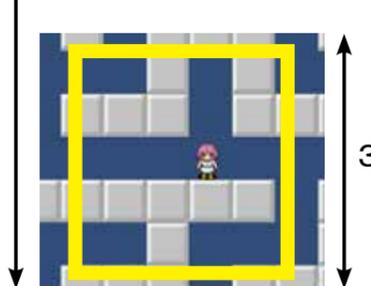
このままだと、キャラクタと壁の衝突時にキャラクタが傾いてしまいます。「Rigidbody2D」コンポーネントの「FreezeRotation」の「z」軸にチェックを入れ、不要な回転を防ぎます。



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。



カメラの「Size」プロパティ



- 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従するようスクリプトを記述し、カメラにアタッチします。

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

スクリプト	g09_camera.cs
アタッチ先	メインカメラ

```
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);
        1
        2
        3
        4
        5
        6
        7
        8
        9
        10
        11
        12
        13
        14
        15
        16
        17
        18
        19
        20
        21
        22
        23
        24
        25
        26
        27
        28
        29
        30
        31
        32
        33
        34
        35
        36
        37
        38
        39
        40
        41
        42
        43
        44
        45
        46
        47
        48
        49
        50
        51
        52
        53
        54
        55
        56
        57
        58
        59
        60
        61
        62
        63
        64
        65
        66
        67
        68
        69
        70
        71
        72
        73
        74
        75
        76
        77
        78
        79
        80
        81
        82
        83
        84
        85
        86
        87
        88
        89
        90
        91
        92
        93
        94
        95
        96
        97
        98
        99
        100
        101
        102
        103
        104
        105
        106
        107
        108
        109
        110
        111
        112
        113
        114
        115
        116
        117
        118
        119
        120
        121
        122
        123
        124
        125
        126
        127
        128
        129
        130
        131
        132
        133
        134
        135
        136
        137
        138
        139
        140
        141
        142
        143
        144
        145
        146
        147
        148
        149
        150
        151
        152
        153
        154
        155
        156
        157
        158
        159
        160
        161
        162
        163
        164
        165
        166
        167
        168
        169
        170
        171
        172
        173
        174
        175
        176
        177
        178
        179
        180
        181
        182
        183
        184
        185
        186
        187
        188
        189
        190
        191
        192
        193
        194
        195
        196
        197
        198
        199
        200
        201
        202
        203
        204
        205
        206
        207
        208
        209
        210
        211
        212
        213
        214
        215
        216
        217
        218
        219
        220
        221
        222
        223
        224
        225
        226
        227
        228
        229
        230
        231
        232
        233
        234
        235
        236
        237
        238
        239
        240
        241
        242
        243
        244
        245
        246
        247
        248
        249
        250
        251
        252
        253
        254
        255
        256
        257
        258
        259
        260
        261
        262
        263
        264
        265
        266
        267
        268
        269
        270
        271
        272
        273
        274
        275
        276
        277
        278
        279
        280
        281
        282
        283
        284
        285
        286
        287
        288
        289
        290
        291
        292
        293
        294
        295
        296
        297
        298
        299
        300
        301
        302
        303
        304
        305
        306
        307
        308
        309
        310
        311
        312
        313
        314
        315
        316
        317
        318
        319
        320
        321
        322
        323
        324
        325
        326
        327
        328
        329
        330
        331
        332
        333
        334
        335
        336
        337
        338
        339
        340
        341
        342
        343
        344
        345
        346
        347
        348
        349
        350
        351
        352
        353
        354
        355
        356
        357
        358
        359
        360
        361
        362
        363
        364
        365
        366
        367
        368
        369
        370
        371
        372
        373
        374
        375
        376
        377
        378
        379
        380
        381
        382
        383
        384
        385
        386
        387
        388
        389
        390
        391
        392
        393
        394
        395
        396
        397
        398
        399
        400
        401
        402
        403
        404
        405
        406
        407
        408
        409
        410
        411
        412
        413
        414
        415
        416
        417
        418
        419
        420
        421
        422
        423
        424
        425
        426
        427
        428
        429
        430
        431
        432
        433
        434
        435
        436
        437
        438
        439
        440
        441
        442
        443
        444
        445
        446
        447
        448
        449
        450
        451
        452
        453
        454
        455
        456
        457
        458
        459
        460
        461
        462
        463
        464
        465
        466
        467
        468
        469
        470
        471
        472
        473
        474
        475
        476
        477
        478
        479
        480
        481
        482
        483
        484
        485
        486
        487
        488
        489
        490
        491
        492
        493
        494
        495
        496
        497
        498
        499
        500
        501
        502
        503
        504
        505
        506
        507
        508
        509
        510
        511
        512
        513
        514
        515
        516
        517
        518
        519
        520
        521
        522
        523
        524
        525
        526
        527
        528
        529
        530
        531
        532
        533
        534
        535
        536
        537
        538
        539
        540
        541
        542
        543
        544
        545
        546
        547
        548
        549
        550
        551
        552
        553
        554
        555
        556
        557
        558
        559
        560
        561
        562
        563
        564
        565
        566
        567
        568
        569
        570
        571
        572
        573
        574
        575
        576
        577
        578
        579
        580
        581
        582
        583
        584
        585
        586
        587
        588
        589
        590
        591
        592
        593
        594
        595
        596
        597
        598
        599
        600
        601
        602
        603
        604
        605
        606
        607
        608
        609
        610
        611
        612
        613
        614
        615
        616
        617
        618
        619
        620
        621
        622
        623
        624
        625
        626
        627
        628
        629
        630
        631
        632
        633
        634
        635
        636
        637
        638
        639
        640
        641
        642
        643
        644
        645
        646
        647
        648
        649
        650
        651
        652
        653
        654
        655
        656
        657
        658
        659
        660
        661
        662
        663
        664
        665
        666
        667
        668
        669
        670
        671
        672
        673
        674
        675
        676
        677
        678
        679
        680
        681
        682
        683
        684
        685
        686
        687
        688
        689
        690
        691
        692
        693
        694
        695
        696
        697
        698
        699
        700
        701
        702
        703
        704
        705
        706
        707
        708
        709
        710
        711
        712
        713
        714
        715
        716
        717
        718
        719
        720
        721
        722
        723
        724
        725
        726
        727
        728
        729
        730
        731
        732
        733
        734
        735
        736
        737
        738
        739
        740
        741
        742
        743
        744
        745
        746
        747
        748
        749
        750
        751
        752
        753
        754
        755
        756
        757
        758
        759
        760
        761
        762
        763
        764
        765
        766
        767
        768
        769
        770
        771
        772
        773
        774
        775
        776
        777
        778
        779
        780
        781
        782
        783
        784
        785
        786
        787
        788
        789
        790
        791
        792
        793
        794
        795
        796
        797
        798
        799
        800
        801
        802
        803
        804
        805
        806
        807
        808
        809
        810
        811
        812
        813
        814
        815
        816
        817
        818
        819
        820
        821
        822
        823
        824
        825
        826
        827
        828
        829
        830
        831
        832
        833
        834
        835
        836
        837
        838
        839
        840
        841
        842
        843
        844
        845
        846
        847
        848
        849
        850
        851
        852
        853
        854
        855
        856
        857
        858
        859
        860
        861
        862
        863
        864
        865
        866
        867
        868
        869
        870
        871
        872
        873
        874
        875
        876
        877
        878
        879
        880
        881
        882
        883
        884
        885
        886
        887
        888
        889
        890
        891
        892
        893
        894
        895
        896
        897
        898
        899
        900
        901
        902
        903
        904
        905
        906
        907
        908
        909
        910
        911
        912
        913
        914
        915
        916
        917
        918
        919
        920
        921
        922
        923
        924
        925
        926
        927
        928
        929
        930
        931
        932
        933
        934
        935
        936
        937
        938
        939
        940
        941
        942
        943
        944
        945
        946
        947
        948
        949
        950
        951
        952
        953
        954
        955
        956
        957
        958
        959
        960
        961
        962
        963
        964
        965
        966
        967
        968
        969
        970
        971
        972
        973
        974
        975
        976
        977
        978
        979
        980
        981
        982
        983
        984
        985
        986
        987
        988
        989
        990
        991
        992
        993
        994
        995
        996
        997
        998
        999
        1000
    }
}
```

No	例題	演習	ミニゲーム	シーンの内容	学習内容	ページ
1	例題1			ボールの移動		2
2		演習1		車の移動	オブジェクトの移動	17
3		演習2		ルーレット回転	オブジェクトの回転	18
4		演習3		タコの拡大縮小	オブジェクトの拡大縮小	19
5		演習4		背景のスクロール	オブジェクトの移動・回転	20
6	例題2			流れ星が乱数で出現	乱数	21
7	例題3			時計の秒針	一定時間ごとの処理	22
8		演習5		キャラクタがランダムに移動	定時処理、乱数	23
9	例題4			キー入力で猫の移動	キー入力処理	24
10		演習6		戦車の移動と回転	キー入力処理	25
11		演習7		縦スクロール移動	キー入力処理	26
12	例題5			ドラッグによるゴルフ	マウス入力	28
13	例題6			タコをクリック	マウスクリックの取得	29
14	例題7			スプライト回転速度	public変数	30
15	例題8			モグラのクリック	スプライトの表示切替	31
16	例題9			クリックで爆発音	効果音の再生	32
17	例題10			マウス位置の表示	ユーザーインターフェイス	33
18	例題11			タコの増殖	プレハブとインスタンス	34
19		演習8		ビームの連続発射	キー入力でインスタンス生成	35
20	例題12			クリックで爆発	アセットの使い方	36
21	例題13			ボタンを並べて配置	繰り返し処理	38
22	例題14			モンスター画像のスライス	スプライトエディタ	39
23			ミニゲーム1	もぐら叩きゲーム		40
24			ミニゲーム2	ゴキブリバスター		42
25			課題1	オリジナルゲーム1	マウスクリックによるゲーム	44
26				透過pngの作り方	Photopeaの使い方	45
27	例題15			ボールの自然落下	物理エンジン(physics)	46
28		演習9		ボールと板	物理エンジン(physics)	48
29		演習10		ボールの増殖	物理エンジン(physics)	49
30	例題16			ボールを貯めるカゴ	親子オブジェクト	50
31		演習11		ボール落下迷路	親子オブジェクト	51
32	例題17			ボール移動	物理演算によるオブジェクト制御	52
33		演習12		ビリヤードシミュレーション	物理演算によるオブジェクト制御	53
34		演習13		キー入力で猫のジャンプ	物理演算によるオブジェクト制御	54
35	例題18			ボールとブロックの衝突	オブジェクトの衝突判定	55
36			ミニゲーム3	ブロック崩しゲーム		56
37	例題19			ビームとドクロ	オブジェクトの衝突判定	58
38			ミニゲーム4	シューティングゲーム		60
39	例題20			サイコロのカウント	配列	64
40	例題21			ランダムなボールの出現	スプライト型の配列	65
41	例題22			トランプの表示	リソースからの一括読み込み	66
42		演習14		トランプの表示(5枚)	配列、乱数	67
43	例題23			サイコロの表示切替	アトラス画像のスライス	68
44	例題24			岩石のランダム落下	アトラス画像の一括読み込み	69
45	例題25			迷路の表示	2次元配列	70
46	例題26			マップデータの表示	2次元配列+アトラス画像	71
47		演習15		迷路内をキャラクタが自動で移動	2次元配列+衝突判定	72
48	例題28			迷路内でドットイート	タグによる衝突オブジェクト判定	73
49	例題27			迷路の自動生成(棒倒し法)	2次元配列 Switch-Case	74
50	例題29			オブジェクト配列のクリックを取得	オブジェクト配列の取得	76
51			ミニゲーム5	ライツアウト(パズル)		77
52			ミニゲーム6	スライドパズル(8パズル)		78
53	例題30			花札の表示	オブジェクト配列の取得	80
54			ミニゲーム7	絵合わせゲーム(神経衰弱)		84
55	例題31			ペンギンのアニメーション	アニメーションクリップ	86
56	例題32			少年のアニメーション	アニメーションクリップとコントローラ	88
57			ミニゲーム8	横スクロールゲーム		90
58		演習16		4方向歩行アニメーション	アニメーションクリップ	92
59		演習17		アセットストアのアニメーションを使う	アニメーター	93
60			ミニゲーム9	迷路ゲーム		94