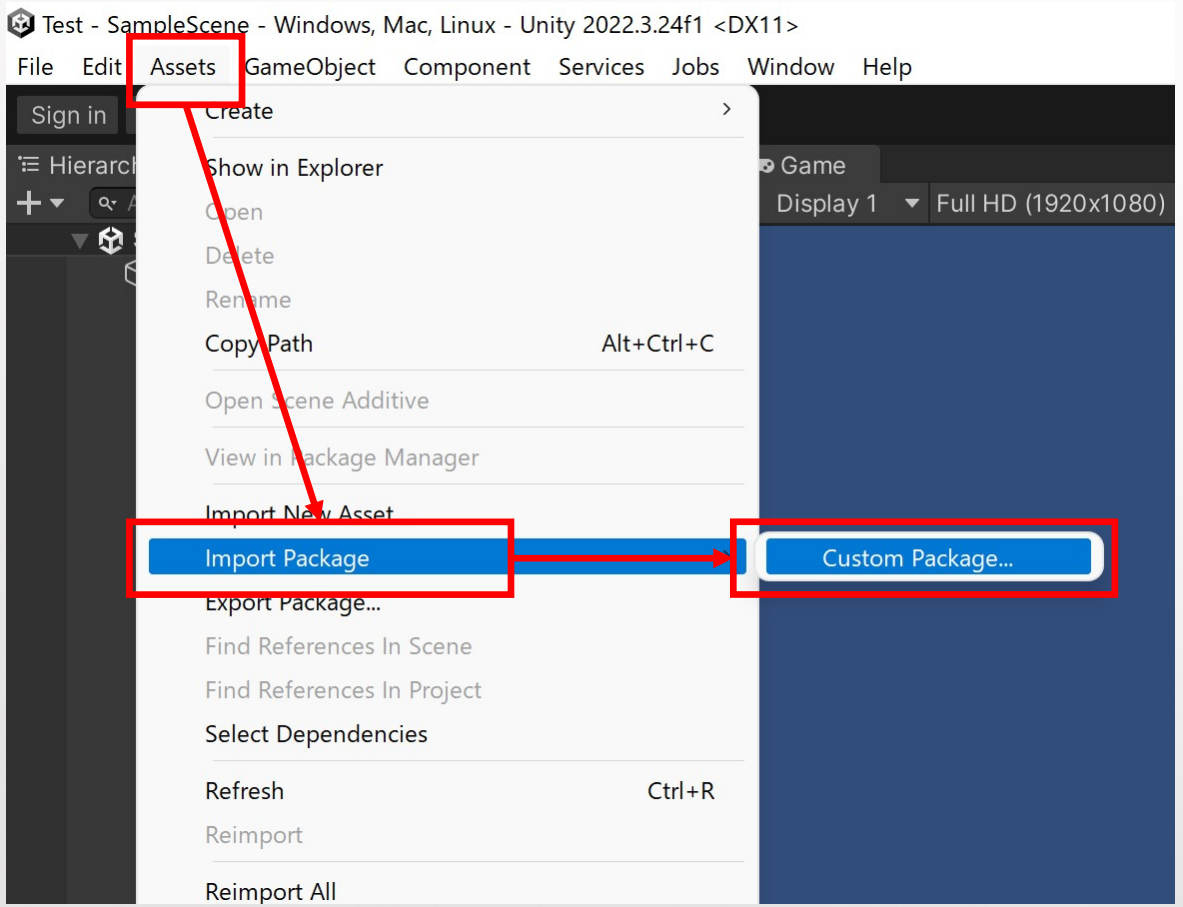
継承とその他機能

目次

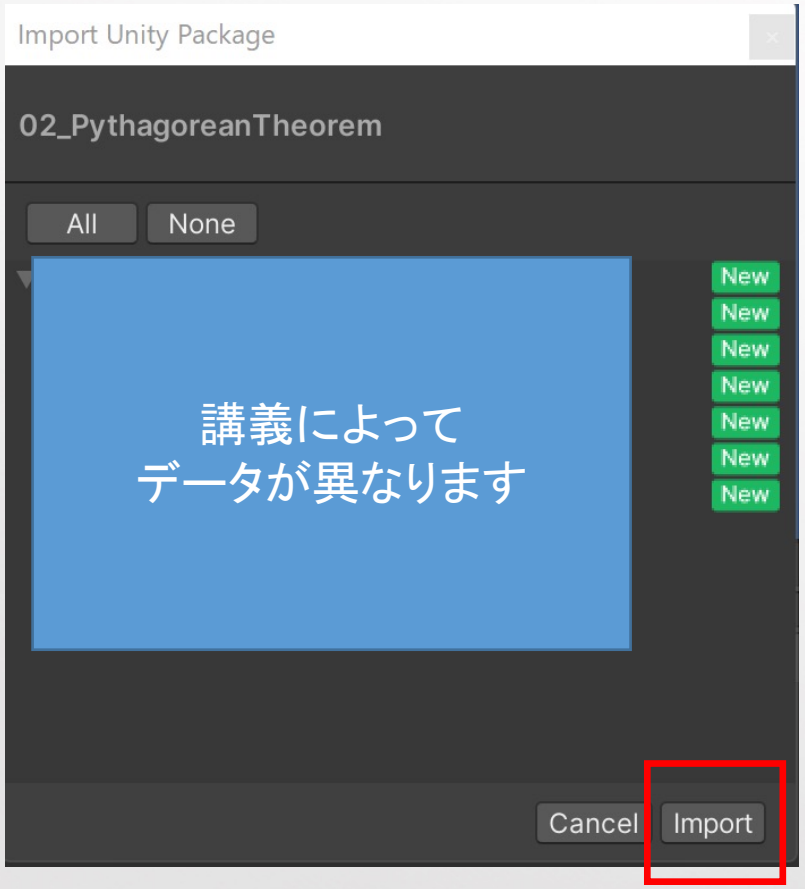
0．UnityPackageを使用しての準備

次に、unitypackageというパッケージファイル(データが詰め込まれたファイル)を読み込みます。



0． UnityPackageを使用しての準備

ファイルを指定し、以下の画面が出たらImportを押すと、ファイルが展開されます。



１．継承とは

＜概念＞

継承とは、あるクラスの機能を受け継いで新しいクラスを作ることを言います。
今回はPlayerControllerクラスを受け継いで新しいクラスを作ります！

PlayerControllerクラスを直接書き換えるのと何が違うの？となるかもしれませんが、例えば他のプロジェクトで動作確認出来ているスクリプトを使って、機能を追加していくことで効率よく開発出来たりします。
(元々の機能は動作が保証されることになる為)

継承する際に、元になるクラスを基底クラス(スーパークラス)、継承したクラスを派生クラス(サブクラス)と言います。

１．継承とは

＜概念＞

実はUnityでスクリプトを作成した場合、既に継承して作られています！

```
public class PlayerController : MonoBehaviour
```

このMonoBehaviourクラスを継承することで、Unityの機能(Start、Update、OnCllisionEnterなど)を使用することが出来るようになります。

なんで、StartとかUpdateの関数が自動的に呼ばれるの？と思ってたかもしれないですが、スーパークラスのMonobehaviourが裏で動かしていたからなんですね！

１．継承とは

＜概念、、ちょっと余談＞

MonoBehaiviourクラスを継承する場合、必ずUnityのヒエラルキー上に
アタッチしないといけない制約があります。
そのため、Unityに直接関わらないような機能を持つクラスの場合は、
無理にMonoBehaviourを継承する必要はないです。
　例：サーバーとの通信を受け持つクラスなど…

この場合はnew演算子でクラスを作成し、使用していきます。
(C++と同様)

１．継承とは

＜サブクラス作成＞

それでは、実際に作成していきましょう！

新しくスクリプトを作成し、PlayerController2.csという名称にしてください。

作成したクラス：PlayerController2.cs

元々作っていたクラス：PlayerController.cs

上記の2つを編集お願いします！

完成したら、Playerにアタッチしているを削除し、代わりに
PlayerController2をアタッチしてください！
同じ動作をするようになったでしょうか？

１．継承とは

＜スクリプト解説＞

(1) PlayerController2.cs

protected override void Start();

protected override void Update();

⇒protectedとはサブクラスのみ公開、という意味を持つ。

(public→protect→private)

overrideとは、スーパークラスのメソッドを引き継いで使用するための宣言。上記の宣言をすることで、EnemyBaseのStartやUpdateメソッドは使用されず、Enemy1のStartやUpdateが使用される。

base.Start(); / base.Update();

⇒スーパークラスのStartやUpdateの呼び出しを実施。

１．継承とは

＜スクリプト解説＞

(2) PlayerController.cs

protected virtual void Start();

protected virtual void Update();

⇒virtualを指定することで、このメソッドはオーバーライドすることが出来ますよ、という目印。

変数には不要なので、注意すること！

１．継承とは

＜問題＞

PlayerControllerを継承した、PlayerController2のクラス内で
以下の機能を実装してください。

(1) 左Shiftを押しながら移動すると、通常より早く動く機能

(2) 右Shiftを押しながらSpaceを押すと大ジャンプする機能

２．雑学的なこと

C++では出来てC＃やJavaでは出来ないこと　⇒　多重継承
この多重継承があるからC++は複雑になりがち。。

多重継承で起きる問題点

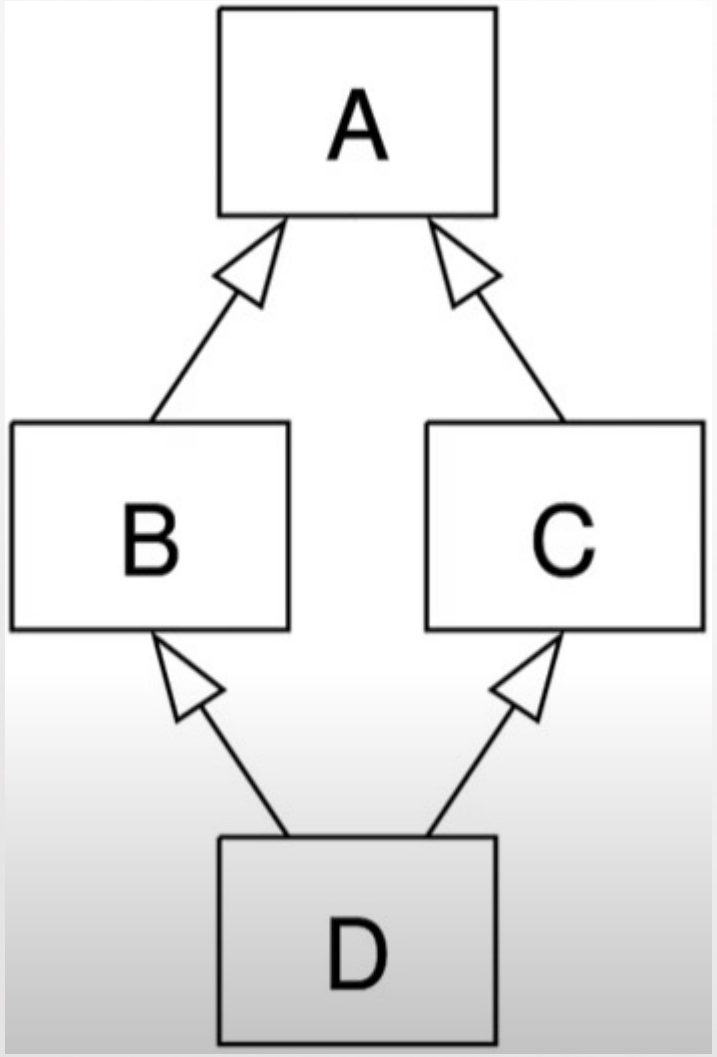
１．名前衝突
２．ダイアモンド継承

ここでは、上記２つの問題がどのようなものか解説する。

２．雑学的なこと(名前衝突)

```
Class A{
Public:
        void Print() { printf(“Aです”); }
};
Class B{
Public:
        void Print() { printf(“Bです”); }
};
Class C : A , B{
        A* a = new A();
        a->Print();
        B* b = new B();
        b->Print();
        C* c = new C();
        c->Print();          ←AかBのどちらかのPrintか分からないからエラー(解決する方法も一応あるが、、、)
};
```

２．雑学的なこと(ダイアモンド継承)



Dで定義しているメソッドを、
BとCでオーバーライドした場合、
Aから見てどちらを使えばよいか分からない。

3．Staticとは

特定のオブジェクトではなく、型自体に属する静的メンバーを宣言することが出来る機能をstaticと言います。(C言語のstaticとは異なるので注意！)

実際に体験した方が早いと思うので、
まずは右記のスクリプトを作成します。

出来た人は、右記のスクリプトを2つ追加してZキーを押した時に、どのような出力がされるのか確認してみてください。

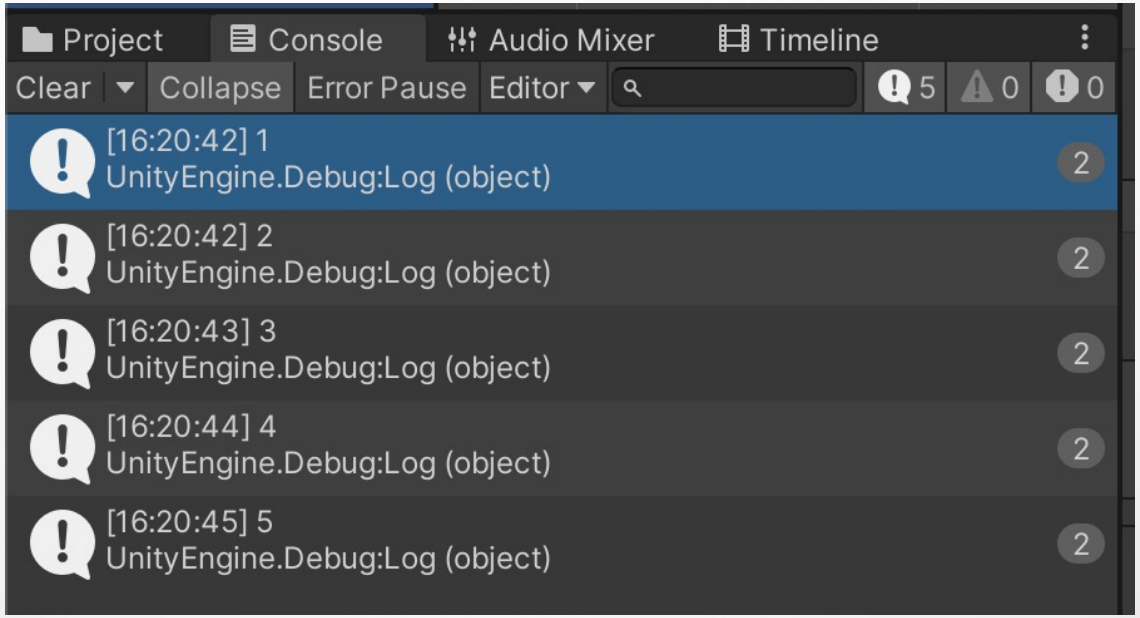
```
public class StaticTest : MonoBehaviour
{
    public int countTest;

    // Start is called before the first frame update
    void Start()
    {
        countTest = 0;
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Z))
        {
            countTest++;
            Debug.Log(countTest);
        }
    }
}
```

3．Staticとは

同じスクリプトを2つつけているので、それぞれのcountTestが+1されて出力しているかと思います。

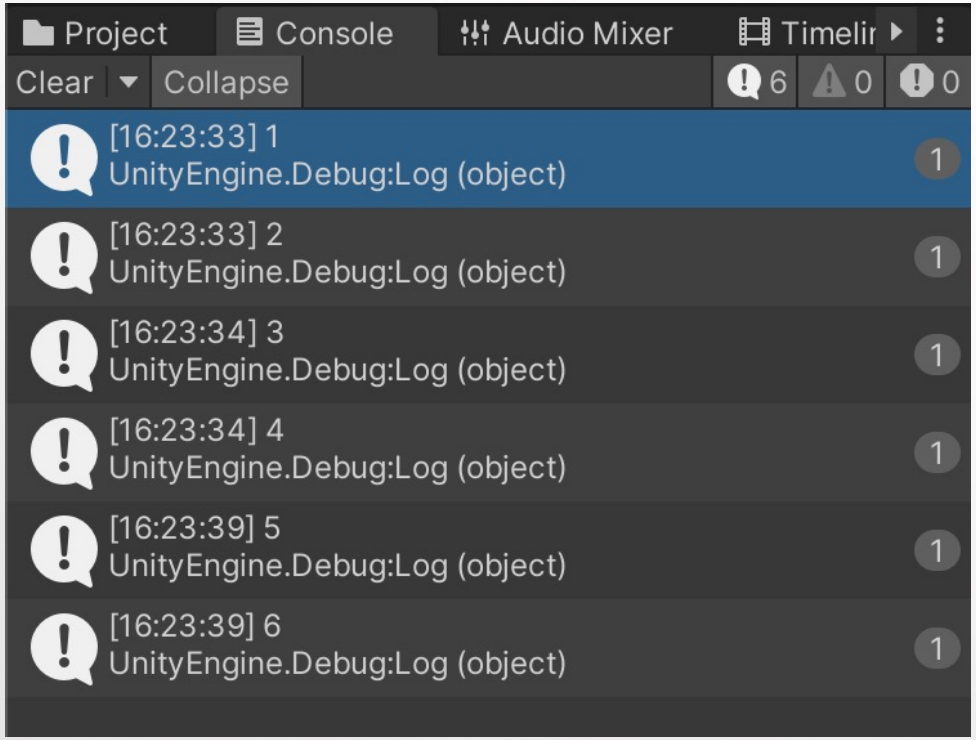


では、以下のように「static」ワードをつけてみてください。

```
public static int countTest;
```

3．Staticとは

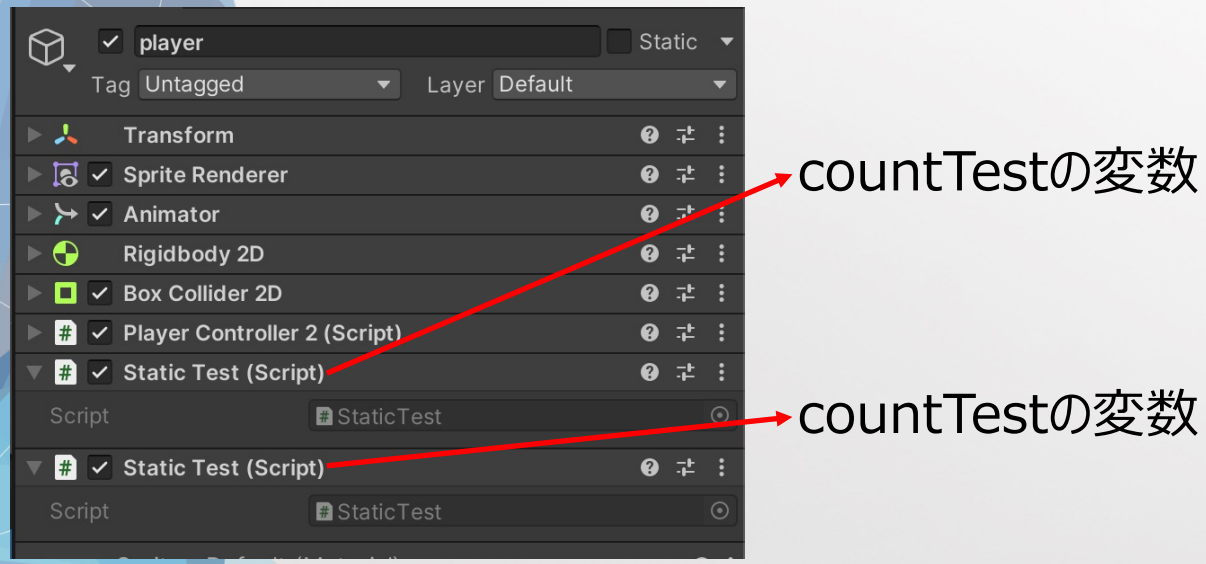
1回Zキーを押すだけで1と2が出力、もう1回押すと3と4が出力されていると思います！

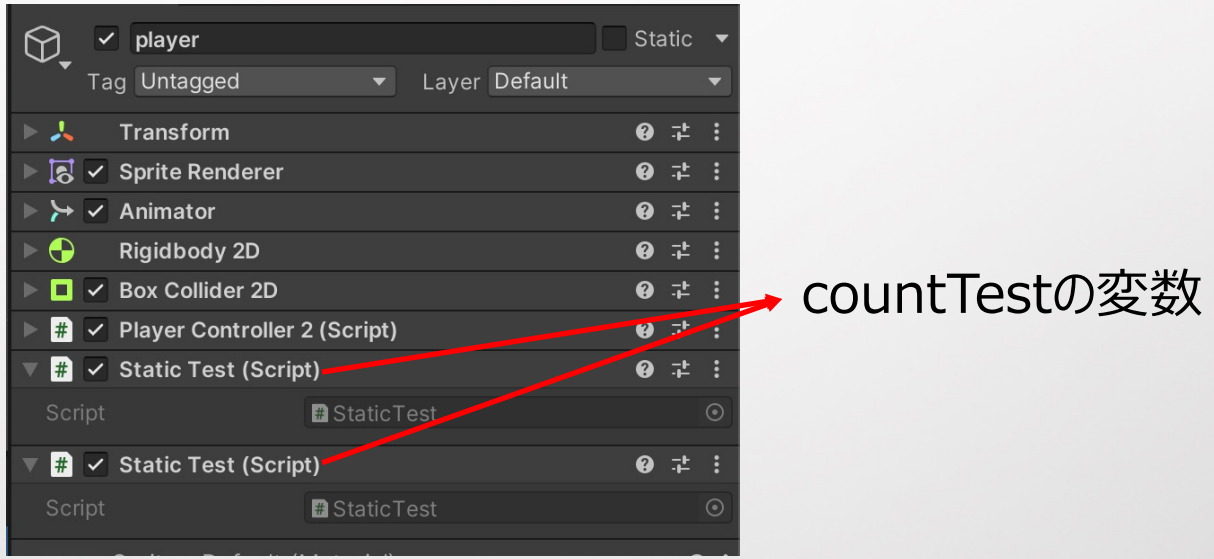


これがstaticの役割となっています！

3．Staticとは

クラスは設計図なので、設計図から出来た実体(インスタンス)ごとに設計図のデータをもっています。ただし、staticというキーワードをつけると、実体ごとにデータを持つのではなく、共通して1つのデータをもつことが出来ます！
(変数だけではなく、メソッドやクラスにも使えます！)





3．Staticとは

Unityで使用する上で、以下のようなメリットとデメリットがあるので状況に応じてぜひ活用してみてください！

＜メリット＞

値の共有が簡単

⇨シーンを跨いで値を使用することが出来る

⇨GetComponentしなくても使用出来る

(PlayerController2の中で以下の1文だけで参照できます)

```
protected override void Update()
{
    Debug.Log(StaticTest.countTest);
    base.Update();
```

＜デメリット＞

メモリから消えることがない

⇨常にメモリに滞留しているので、余計なメモリを取る場合があります。

3．Staticとは

＜問題＞

以下のようなユーティリティクラスの実装をお願いします。

・0から100までのランダムな数字を算出

・数値によって、以下のデータを返す。

80以上：A

60以上：B

40以上：C

それ以下：D

3．Staticとは

＜解答例＞

```
public class RankUtility
{
    public static string GetRank()
    {
        int score = Random.Range(0, 100);
        if (score >= 80) return "A";
        else if (score >= 60) return "B";
        else if (score >= 40) return "C";
        else return "D";
    }
}
```

４．拡張メソッド

拡張メソッドとは、既存のクラスに新しいメソッドを追加する機能のこと。
元の型を継承したり修正することなく元々あるような形で呼び出すことが出来る。
ただし、staticクラスであることが条件となります。

よくあるのは、intやstring、GameObjectやtransformなどのC#やUnity特有の標準の型で拡張メソッドが使われます。
実際に使ってみましょう！(GameObjectExtension.csを作成)

```
public static class GameObjectExtension
{
    public static Rigidbody2D GetR2D(this GameObject go)
    {
        return go.GetComponent<Rigidbody2D>();
    }
}
```

```
public class PlayerController2 : PlayerController
{
    // Start is called before the first frame update
    protected override void Start()
    {
        base.Start();
        gameObject.GetR2D().gravityScale = 2;
    }
```

４．拡張メソッド

＜問題＞

①GameObjectに対して、アクティブ/非アクティブを切り替えする
　ToggleActiveメソッドを作成してください。
　(アクティブだったら非アクティブにする、非アクティブならアクティブにする)

②GameObjectに対して、シーン遷移で破棄されないようにする
　拡張メソッドを作成してください。

③子オブジェクトを全て削除する拡張メソッドを作成してください。

４．拡張メソッド

```
// GameObject のアクティブ状態をトグル（ON/OFF切り替え）する拡張メソッド
public static void ToggleActive(this GameObject obj)
{
    if (obj == null) return; // 念のため null チェック
    obj.SetActive(!obj.activeSelf);
}
```

```
// GameObjectをシーン遷移で破棄されないようにする拡張メソッド
public static void DontDestory(this GameObject obj)
{
    if (obj != null)
    {
        Object.DontDestroyOnLoad(obj);
    }
}
```

```
public static void DestroyAllChildren(this GameObject obj)
{
    foreach (Transform child in obj.transform)
    {
        GameObject.Destroy(child.gameObject);
    }
}
```