Unityで使える設計技術（デザインパターン）

シングルトン（Singleton）

シングルトンとは

- 「シングルトン」（singleton）とは、一人っ子や独身者など、1つだけの人やものを表す

- プログラミングでは、稼働中のアプリの中で1つしかないものを指す。

- 　C#でシングルトンを実現するには、静的クラスを使う方法と、シングルトンパターンを実装したクラスを作る方法がある。メンバーだけ静的でよければ、クラスはpublicでも良い。

静的クラスを
シングルトンとして使う

静的クラスを使うことで、インスタンスを作れない唯一のクラスを作ることはできます。
ただし、静的クラスでは継承が使えない制約があるため、
継承が必要な時は、シングルトンパターンを使う

```
// 静的クラスでも1つしか作れないクラスは作成できますが・・・
public static class StaticClass : MonoBehaviour
{
    // 静的プロパティ：このData1はゲーム中で1つのみのデータ
    public static string Data1 { get; set; }
}
```

❌継承はできません

シングルトンパターンを実装するには

- 静的クラスでないため、インスタンス化が可能。なので外部からインスタンス化できない仕組みが必要
（コンストラクタをプライベートにする）

- インスタンスをプライベートな静的メンバ変数に保持する（インスタンスの生成は静的コンストラクタまたは静的メンバ変数の初期化で行う）

- インスタンスを公開する静的メソッド化プロパティを追加する

シングルトンパターンを実装するには

インスタンスをプライベートな静的メンバ変数に保持
新しくインスタンス化しても変数の中身を保持する

```
class Singleton
{
    static Singleton instance;

    Singleton()
    {
    }

    public static Singleton Instance()
    {
        // 初期化
        if (instance == null)
        {
            instance = new Singleton();
        }

        return instance;
    }
}
```

プライベートコンストラクタ：
privateなので他のクラスからnewできない
publicだと
- Singleton obj1 = new Singleton();
- Singleton obj2 = new Singleton();

など、いくつでもできてしまう

インスタンス取得専用のメソッドを用意
最初にInstance()メソッドが呼び出された時、newを使って自分自身のオブジェクトを生成してinstance変数に保持しておく。
コンストラクタがprivateでもクラス内からアクセス可能。

```
class Singleton
{
    // 初期化（最初はnull状態）
    static Singleton instance;

    Singleton()
    {
    }

    public static Singleton Instance()
    {
        // 初期化（最初はnull状態）
        if (instance == null)
        {
            instance = new Singleton();
        }

        return instance;
    }
}
```

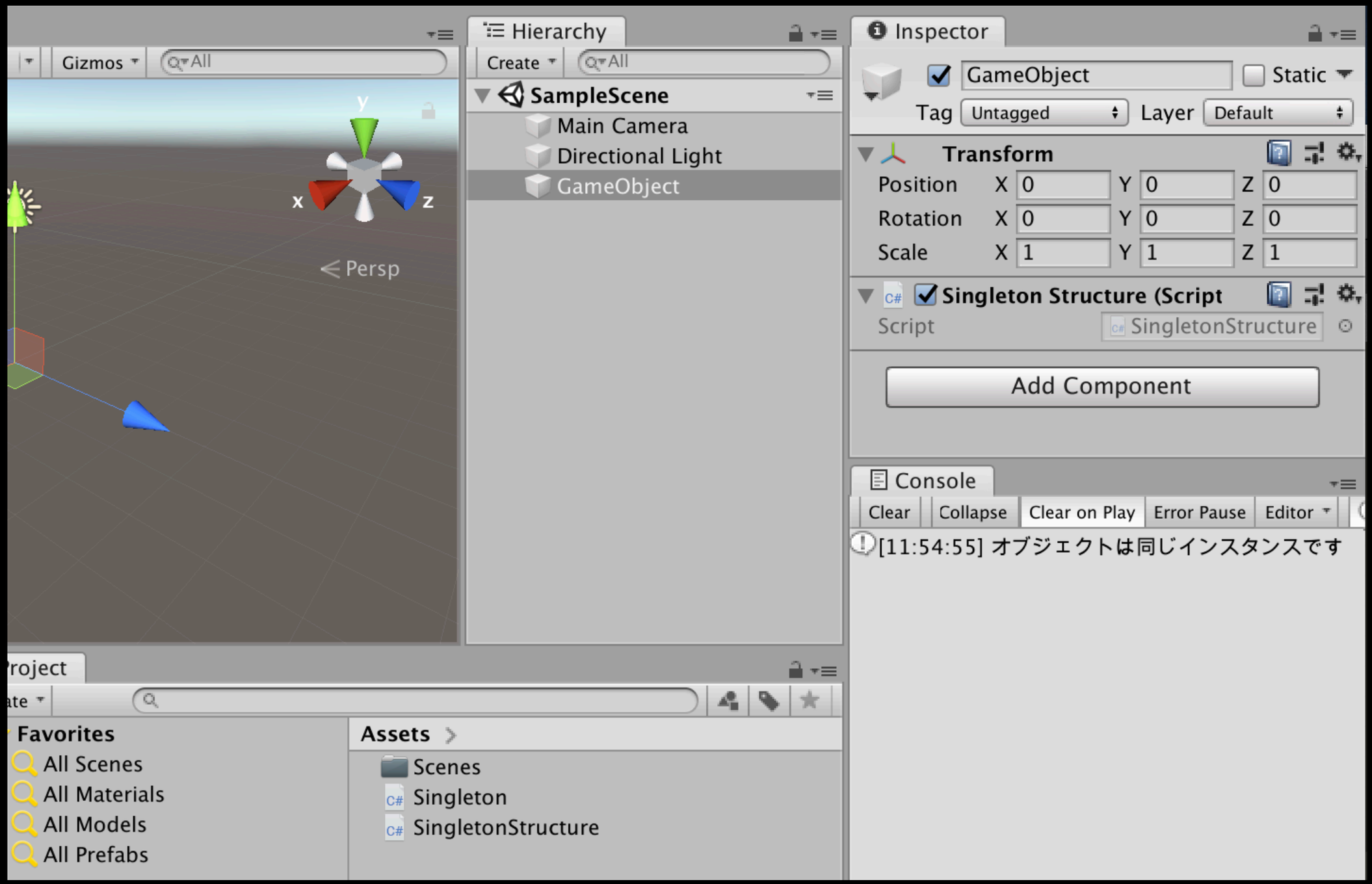
２回目以降は、Instance()メソッドが呼び出された時はすでに生成済みのオブジェクトの参照を返す

呼び出し側スクリプト

```
using UnityEngine;

public class SingletonStructure : MonoBehaviour
{
    void Start()
     {
         // Singletonクラスのコンストラクタは
         // priveteなので、newが使えません
         Singleton s1 = Singleton.Instance();
         Singleton s2 = Singleton.Instance();

         // 同じインスタンスかテストする
         if (s1 == s2)
         {
             Debug.Log("オブジェクトは同じインスタンスです");
         }
     }
}
```

ほんとにインスタンスは、
1つだけしかできない？

プロジェクト
[SingletonSample]



実際の動作を確認

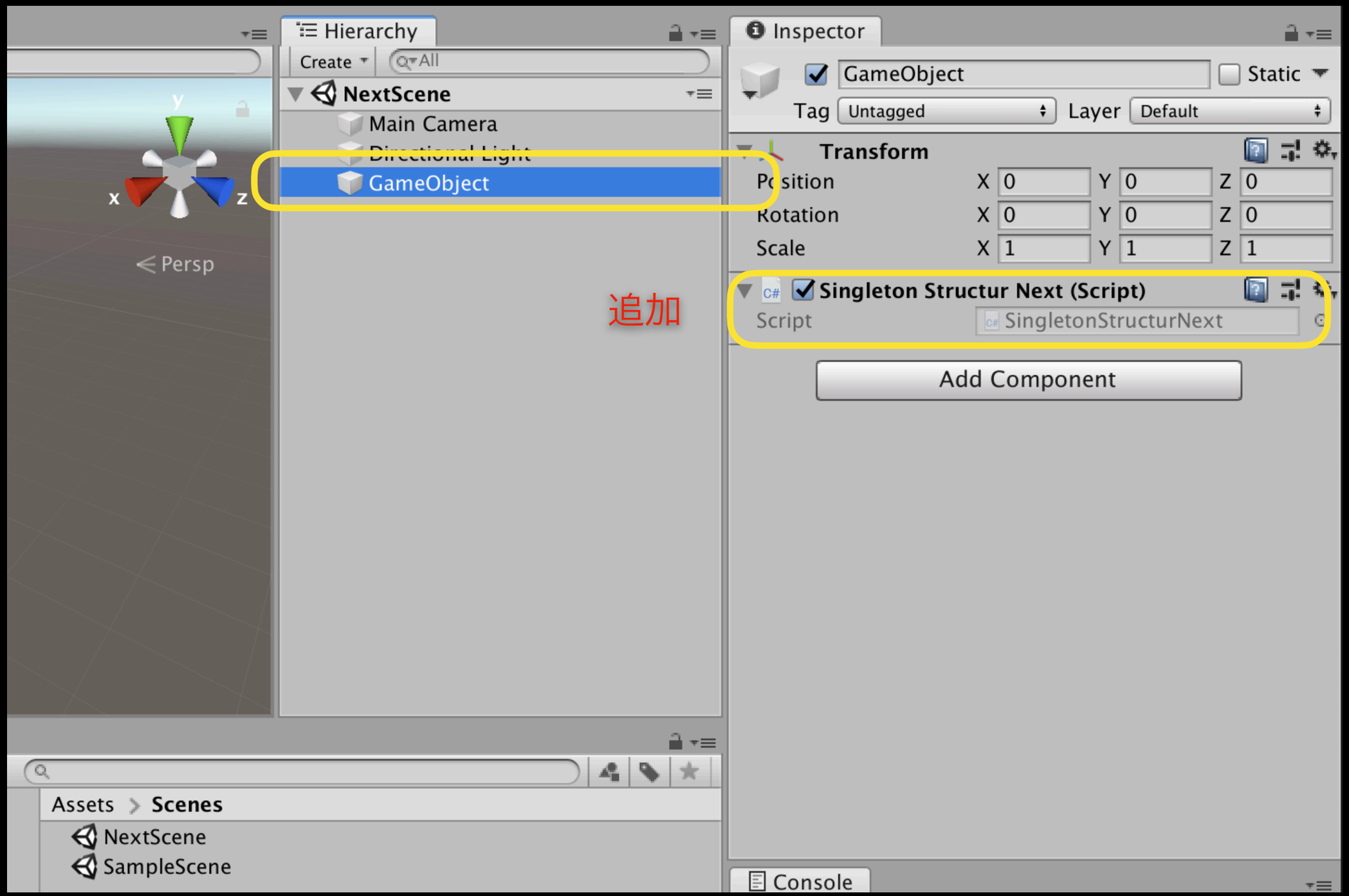
```
class Singleton
{
    static Singleton instance;

    Singleton()
    {
    }

    public static Singleton Instance()
    {
        // 初期化
        if (instance == null)
        {
            instance = new Singleton();
        }

        return instance;
    }
}
```

ここにブレークポイントを設定して
動作確認してみましょう

応用 1

シーン切り替えで情報保持

NextSceneを作成



Singletonを変更

Hiscore保持用のプロパティを追加

```
class Singleton
{
    static Singleton instance;

    public int HiScore { get; set; } = 10;

    Singleton()
    {
    }

    public static Singleton Instance()
    {
        // 初期化
        if (instance == null)
        {
            instance = new Singleton();
        }

        return instance;
    }
}
```

SingletonStructureを変更

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class SingletonStructure : MonoBehaviour
{
    Singleton s1;

    void Start()
    {
        s1 = Singleton.Instance();
    }

    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            Debug.Log($"最初のシーン　{s1.HiScore}");
            SceneManager.LoadScene("NextScene");
        }
    }
}
```

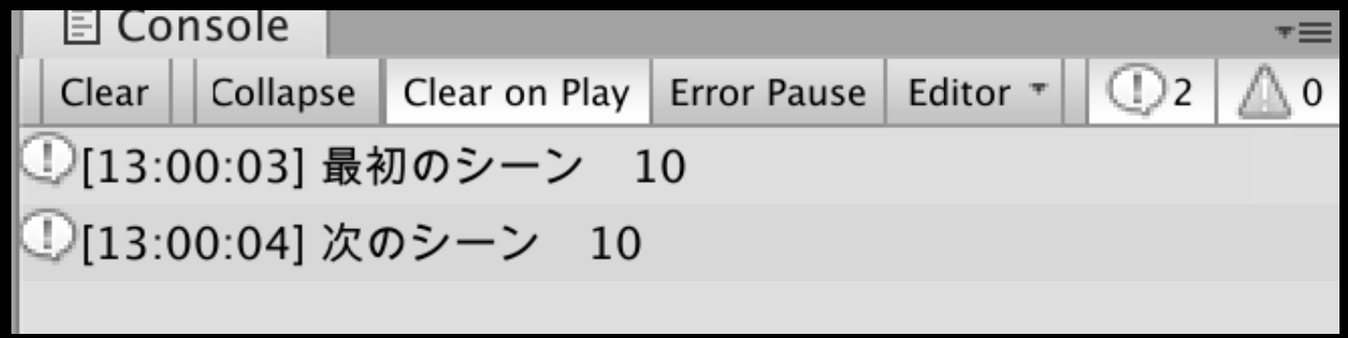
- ハイスコアの表示
- シーンを変更

SingletonStructureNextを追加

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class SingletonStructurNext : MonoBehaviour
{
    Singleton s1;

    void Start()
    {
        s1 = Singleton.Instance();
    }

    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            Debug.Log($"次のシーン  {s1.HiScore}");
        }
    }
}
```

- ハイスコアの表示

実行結果



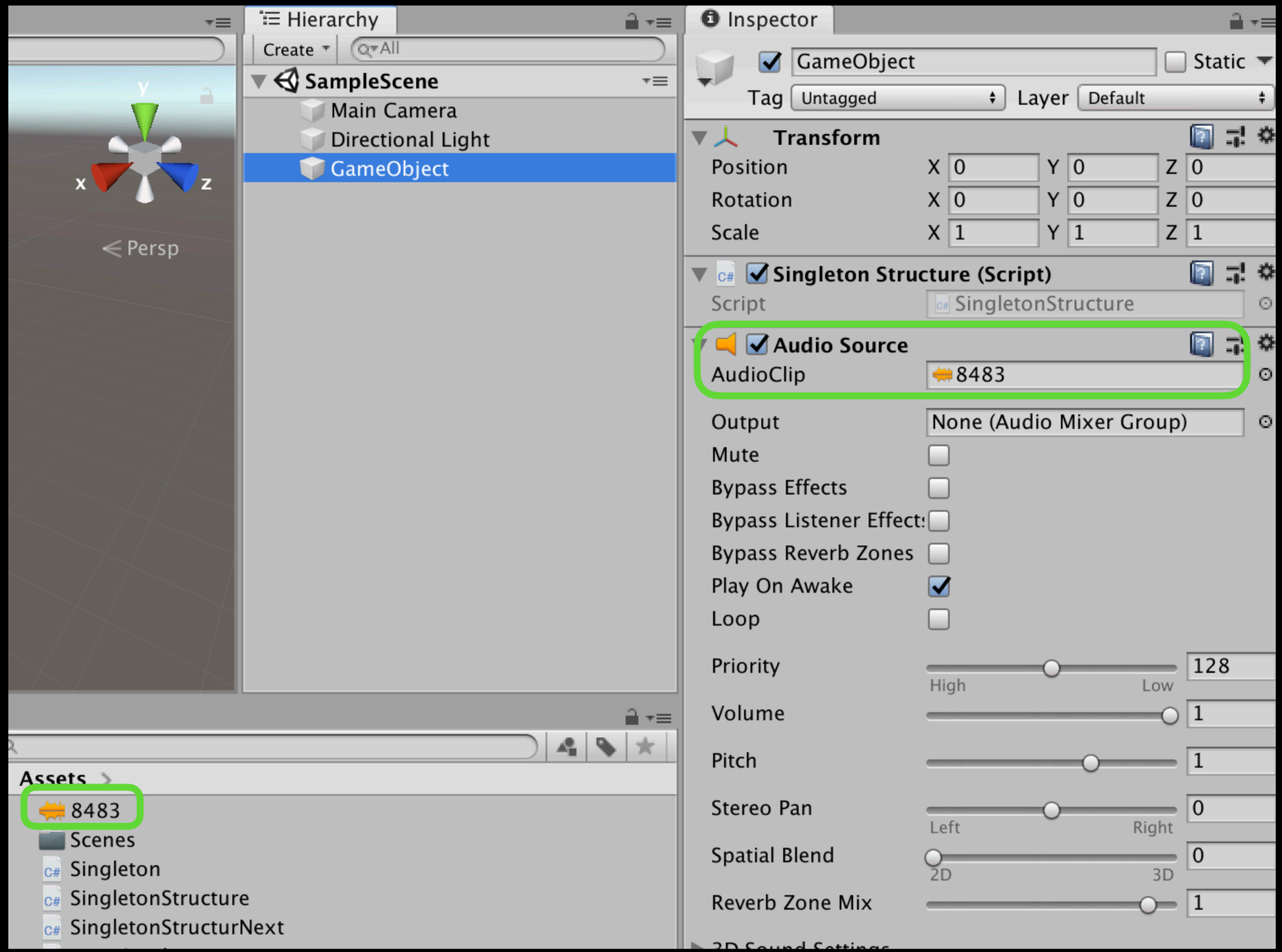
１回目のクリック

２回目のクリック

応用2

シーン切り替えでもサウンドを鳴らし続ける

SampleSceneの変更



呼び出し側スクリプト

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class SingletonStructure : MonoBehaviour
{
    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            SceneManager.LoadScene("NextScene");
        }
    }
}
```

実行結果

- シーン切り替えでサウンドが途切れてしまいますね
- AudioSourceコンポーネントがシーンを切り替えること で破棄されてしまうためです。
- シーンの遷移をしながら、BGMを途切れさせない（継続して鳴らし続ける）ことはできないでしょう？

呼び出し側スクリプト変更

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class SingletonStructure : MonoBehaviour
{
    void Awake()
    {
        DontDestroyOnLoad(gameObject);
    }

    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            SceneManager.LoadScene("NextScene");
        }
    }
}
```

ゲームオブジェクトを保持し続ける（Destroyしないこと実施するメソッド）

実行結果

- シーン切り替えでサウンドが途切れなくなりましたね。
- AudioSourceコンポーネントをアタッチされているオブジェクトがシーンとは切り離されて存在し続けるためです。
- シーンの遷移をしながら、BGMを途切れさせない（継続して鳴らし続ける）ことが実現できました。

でも・・・

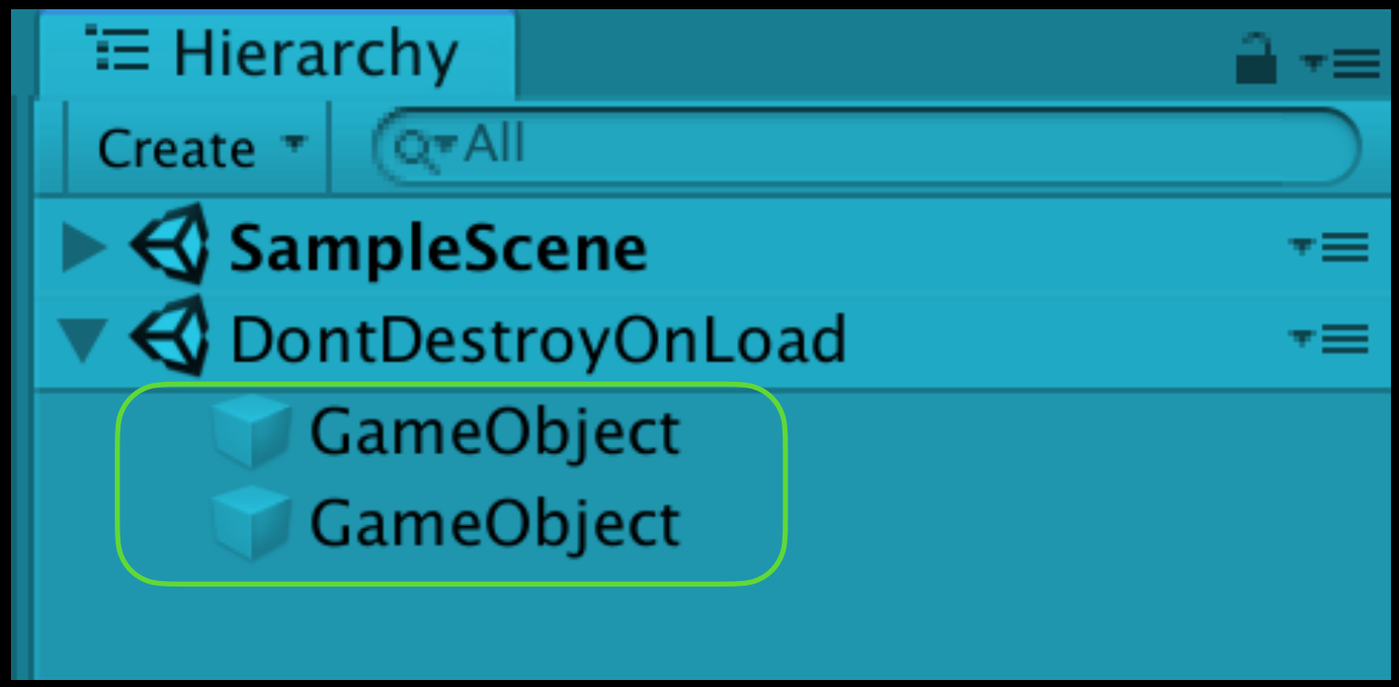
シーンを戻るようにすると

```
using UnityEngine;
using UnityEngine.SceneManagement;
 2つ目のシーンにアタッチされているスクリプト
public class SingletonStructurNext : MonoBehaviour
{
    void Update()
    {                                  1つ目のシーンに戻るメソッドを実行
        if (Input.GetMouseButtonDown(0))
        {
            SceneManager.LoadScene("SampleScene");
        }
    }
}
```

実行結果

- サウンドが重複されました。
- AudioSouceがアタッチされた、2つのGemeObjectが存在してしまうためです。

実行結果



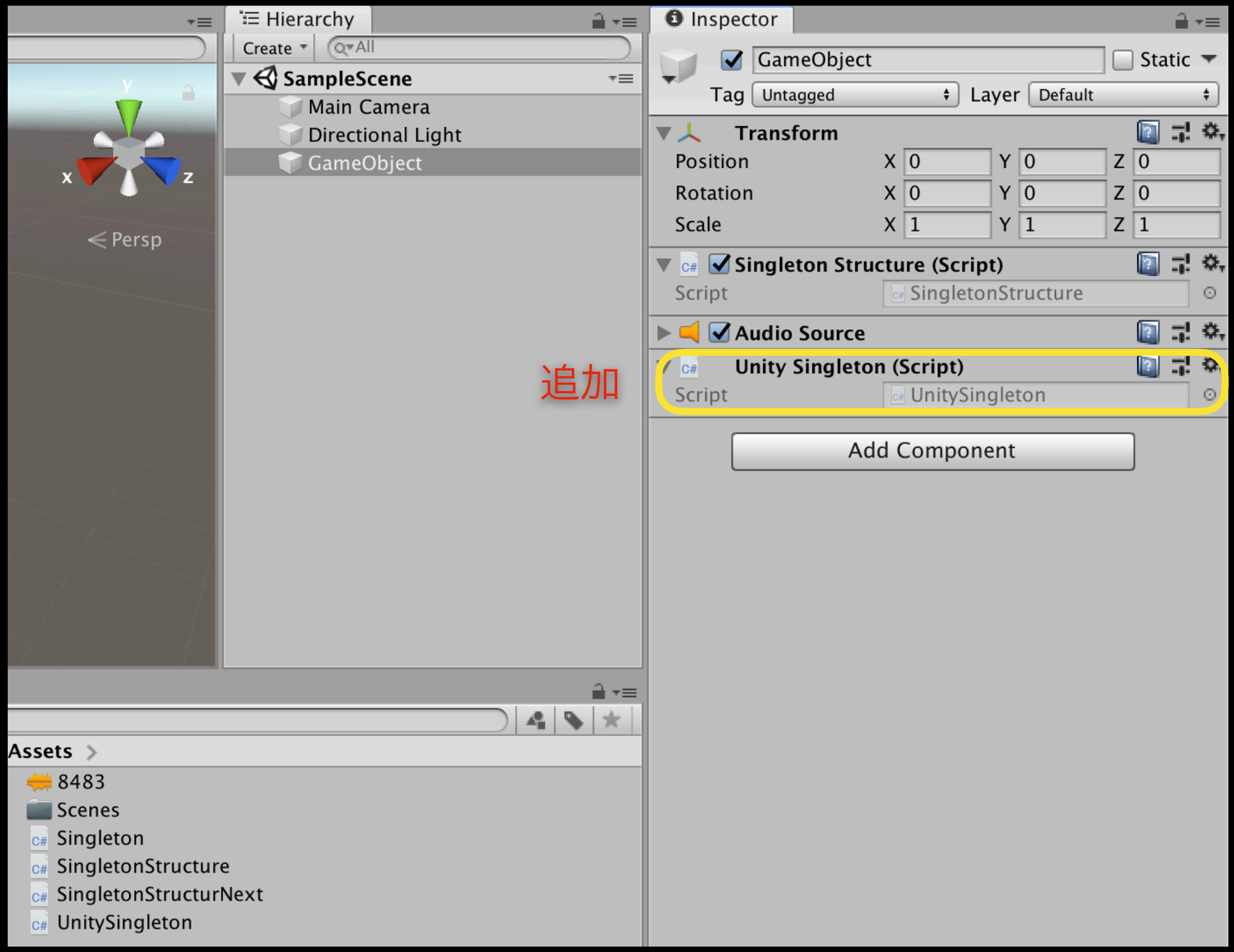
ゲームオブジェクトが２つ

シングルトンパターンを適用

回避するため、新しくスクリプト作成

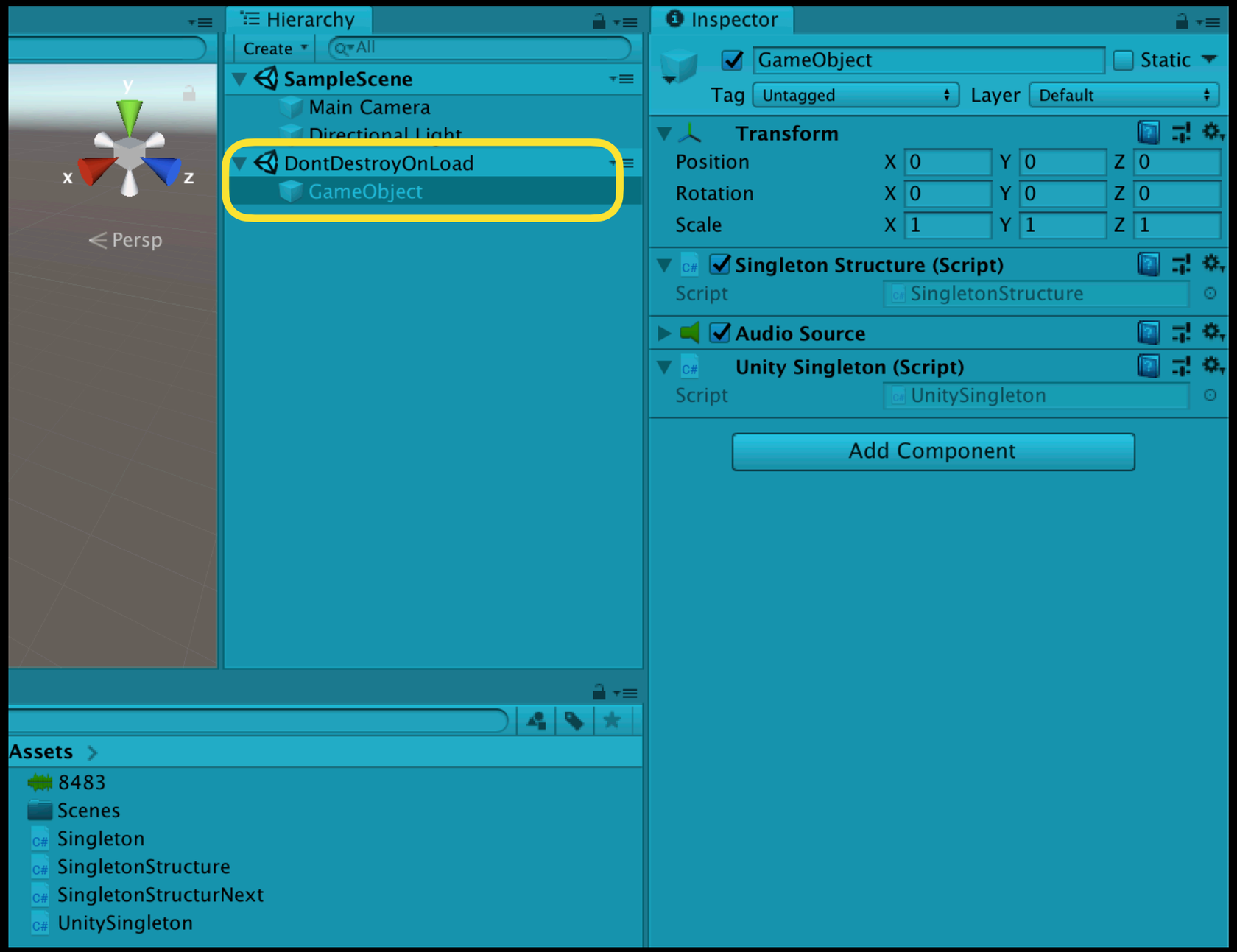
```
using UnityEngine;

public class UnitySingleton : MonoBehaviour
{
    //static: 新しくインスタンス化しても変数の中身を保持する
    public static UnitySingleton instance;

    void Awake()
    {
        //UnitySingletonインスタンスが存在しなければ
        if (instance == null)
        {
            //UnitySingletonインスタンスがなかったら
            //このUnitySingletonをインスタンスとする
            instance = this;
            //シーンを跨いでもUnitySingletonインスタンスを破棄しない
            DontDestroyOnLoad(gameObject);
            return;
        }
        //今回インスタンス化したUnitySingletonを破棄
        Destroy(gameObject);
    }
}
```

シングルトンパターンを適用



実行結果

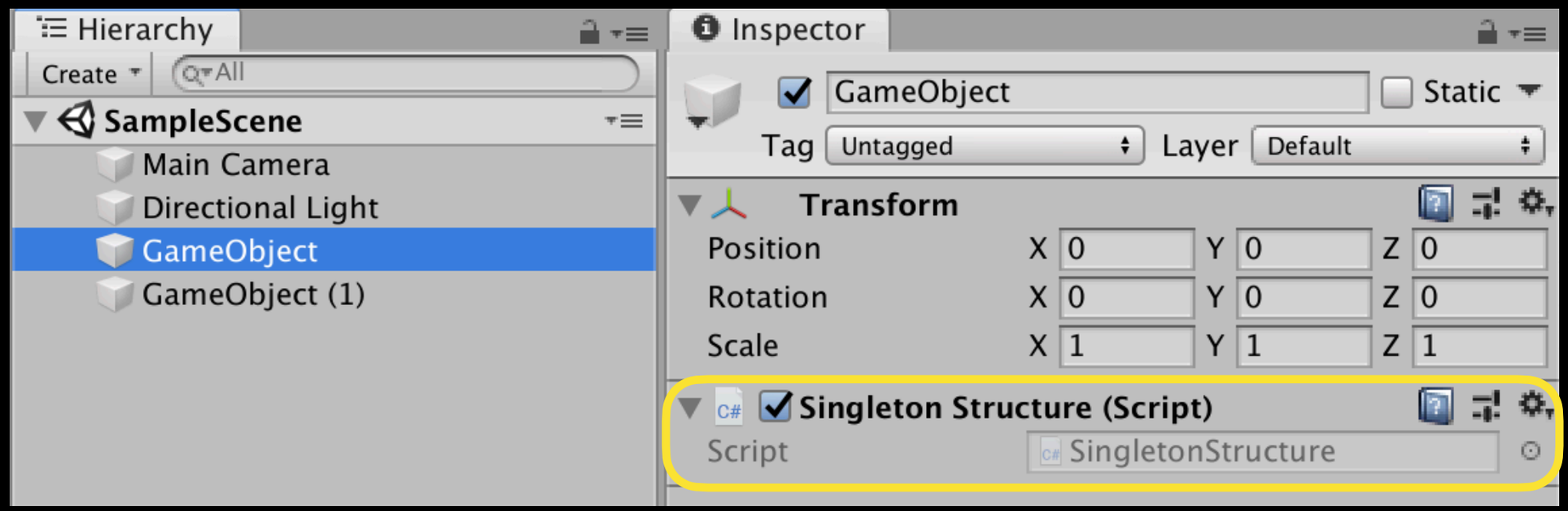
ゲームオブジェクトが１つになりました



おまけ

- シーンの途中からのデバックでは、音が鳴らない
- NextSceneからデバッグすると、音が鳴らない
- 困りました。どのようにすれば、デバッグを効率よくできるでしょう？

おまけ

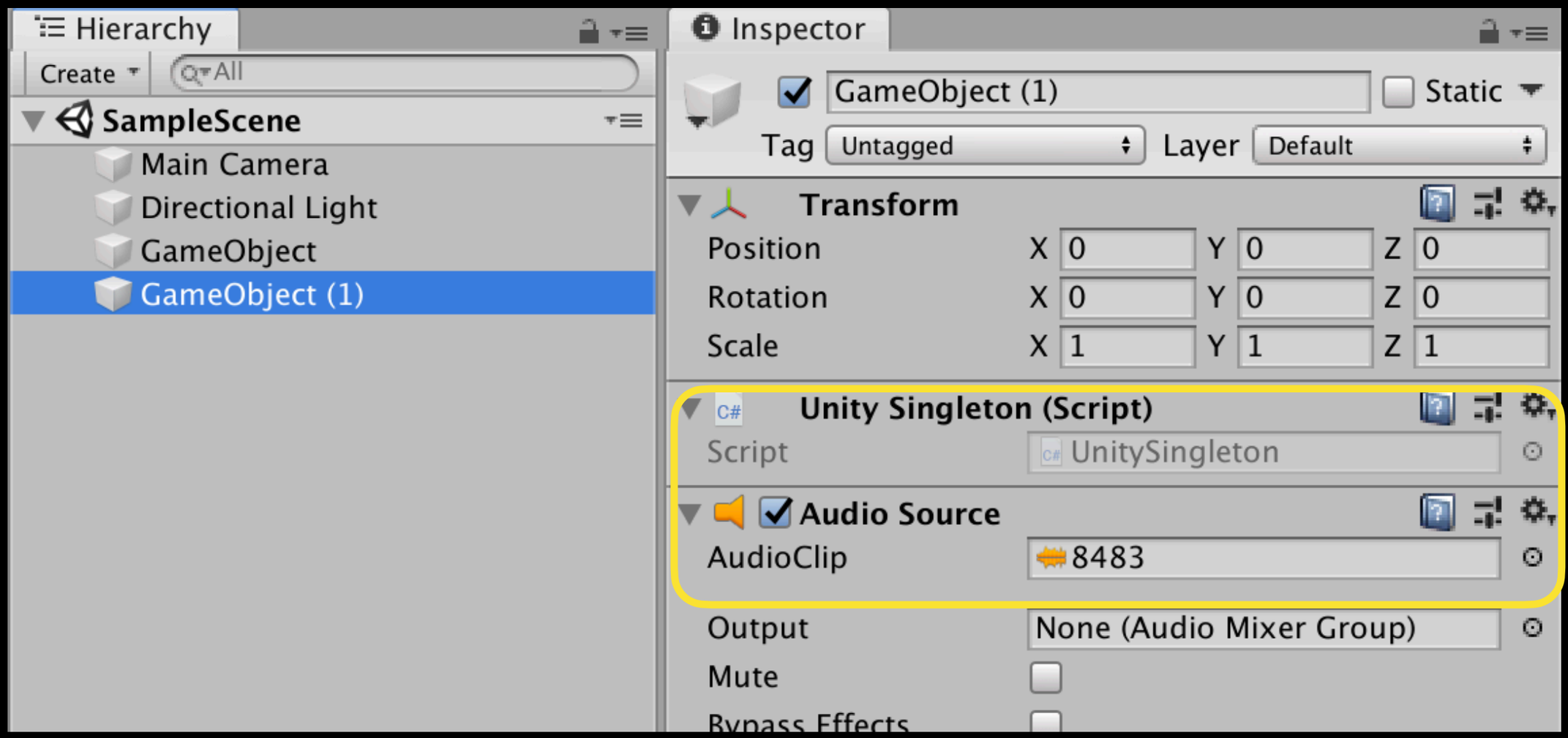


SampleSceneこれだけにする

呼び出し側スクリプト変更

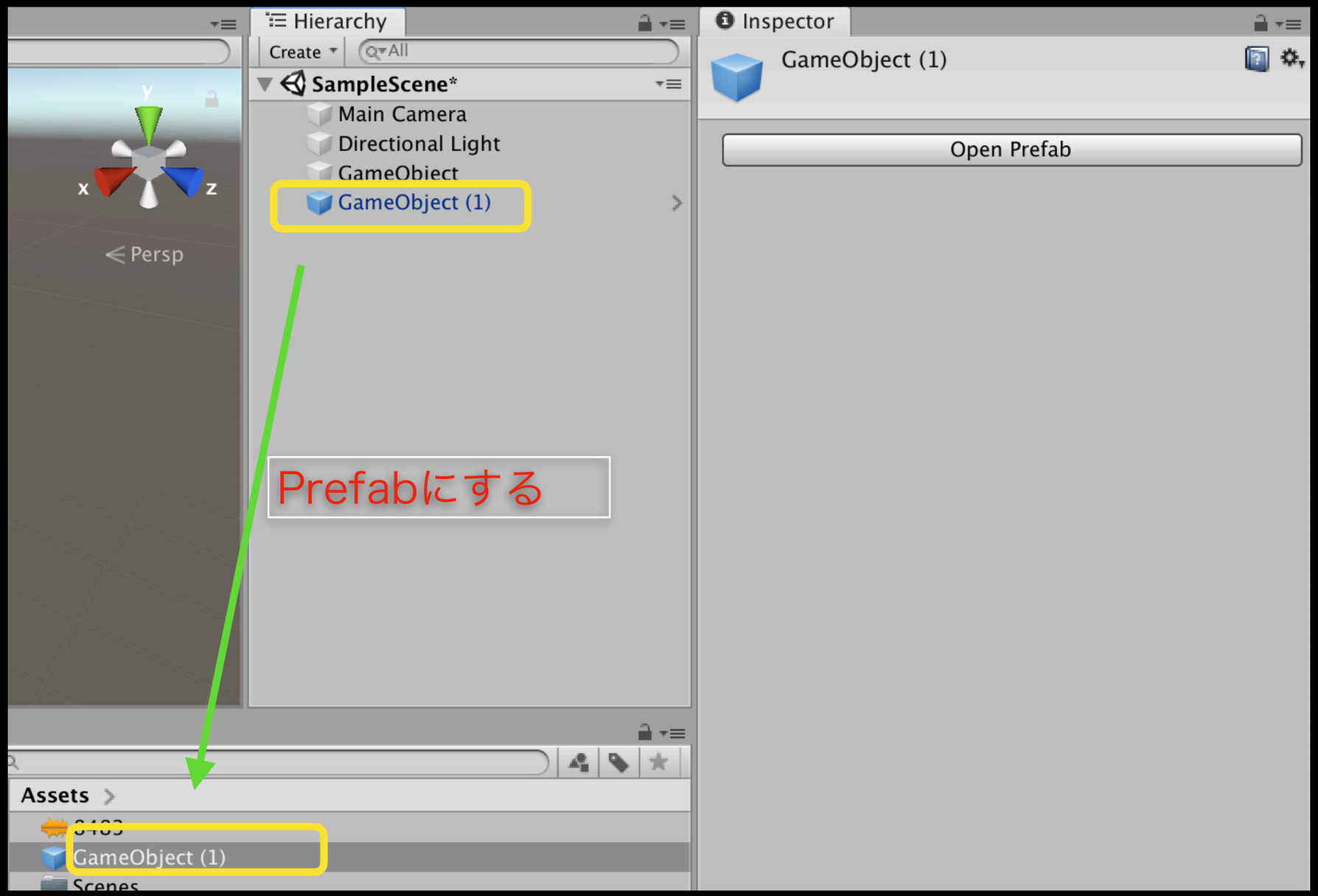
```
using UnityEngine;
using UnityEngine.SceneManagement;

public class SingletonStructure : MonoBehaviour
{
    private void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            SceneManager.LoadScene("NextScene");
        }
    }
}
```

おまけ1

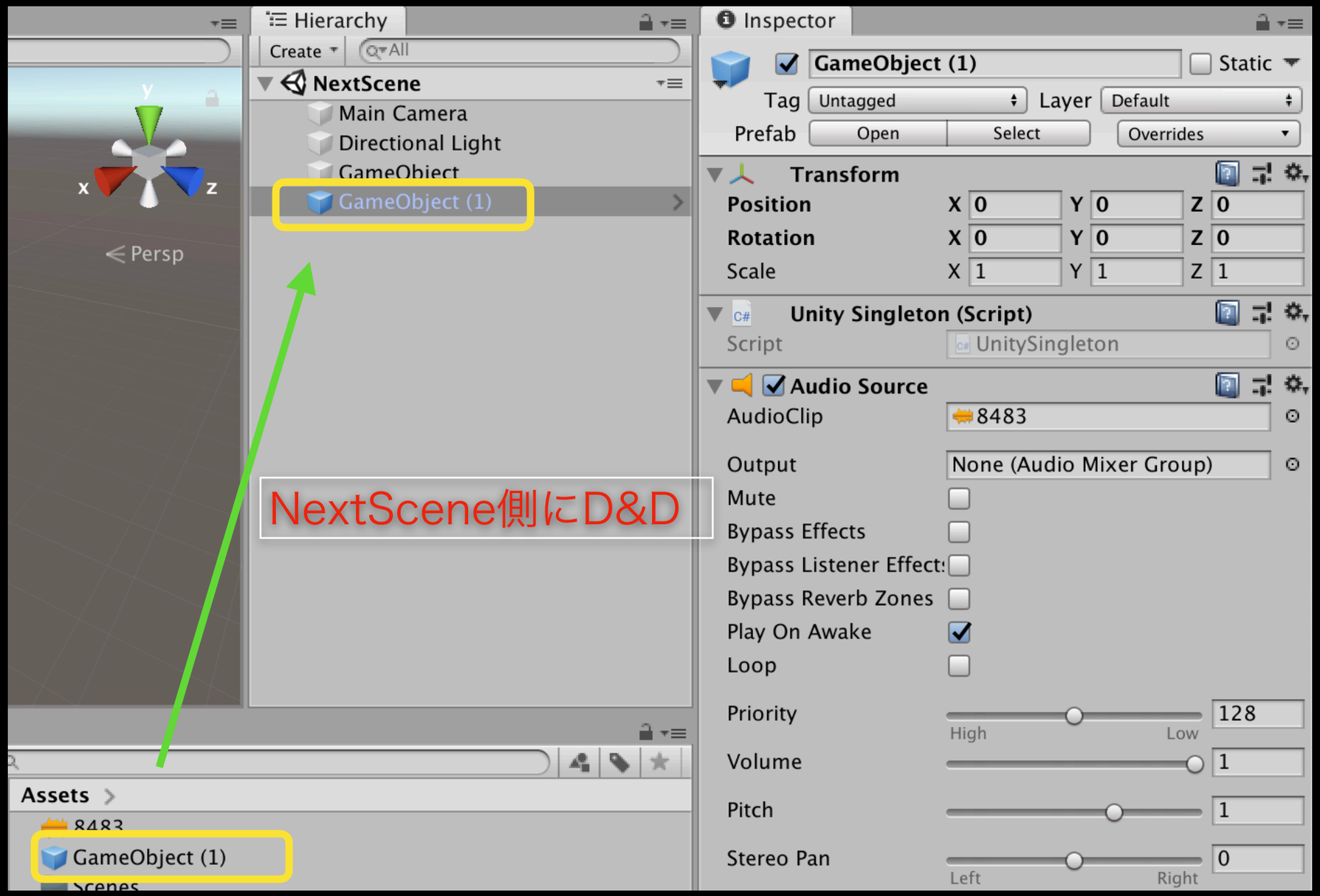


新しく空のオブジェクトを作成
上記２つを設定。
実行して、確認

おまけ1


Hierarchy
Create
All
SampleScene*
Main Camera
Directional Light
GameObject
GameObject (1)
Inspector
GameObject (1)
Open Prefab
Persp
Prefabにする
Assets
GameObject (1)
Scenes

おまけ1



このシーンから始めてもサウンドが再生されるのでデバッグが可能

おまけ２

C#サンプルの変形パターン１

```
class Singleton
{
    private static Singleton instance;

    public int HiScore { get; set; } = 10;

    private Singleton() {}

    public static Singleton Instance
    {
        get
        {
            // 初期化
            if (instance == null)
            {
                instance = new Singleton();
            }

            return instance;
        }
    }
}
```

メソッドをプロパティに変換
呼び出し元の変更が必要

```
var s1 = Singleton.Instance();
↓
var s1 = Singleton.Instance;
```

おまけ２
C#サンプルの変形パターン2

Ver６以降　null 合体演算子（??の記号）

```
class Singleton
{
    private static Singleton instance;

    public int HiScore { get; set; } = 10;

    private Singleton() { }

    public static Singleton Instance
    {
        get
        {
            instance = instance ?? new Singleton();
            return instance;
        }
    }
}
```

instanceには、
instanceがnullでなければ、instance
instanceがnullの場合、new Singleton()
が代入される

おまけ２
C#サンプルの変形パターン3

Ver6以降　プロパティを初期化できる（初期化子）

```
class Singleton
{
    public static Singleton Instance { get; } = new Singleton();

    public int HiScore { get; set; } = 10;

    private Singleton() { }
}
```