(http://clearcutgames.net/home/)

Unity3D Tips #2 – The Singleton Pattern

July 22, 2012 (http://clearcutgames.net/home/?p=437)

/ *admin (http://clearcutgames.net/home/?author=1)*

Today we are going to talk about something truly unique… The singleton pattern!
The singleton pattern lets you write a class, which can only be instantiated once. A singleton object sure is 'one of a kind'.

(http://clearcutgames.net/home/wp-content/uploads/2012/07/OneOfAKind.jpg)

The singleton pattern is a very useful, yet very simple design pattern.
A design pattern is a general solution that you can apply to a problem in software design. In this case our problem is that we want a class, that there should only be one instance of, and we want easy access to that instance.

If you are a relatively new programmer, you might be thinking "What use is this? Why on earth would I write a class that can only be instantiated once?"
Now think of objects like an input manager responsible for reading input from the keyboard and mouse, or an audio handler like the one we described in our last post, responsible for playing audio in your game. Having more than one instance of these, might cause some very incorrect behavior.

Okay, so there's that. But what about writing a static class? Using a static class you can have a class with global access, and it can't be instantiated so you will definitely not have multiple instances.

It seems that a static class could do the job, but it has some downsides:

• You can't extend MonoBehaviour with a static class, and thereby not apply your class as a component on a GameObject in Unity.
• You can't pass around a static class as a parameter
• You can't implement an interface with a static class

Let's try and look at a very simple implementation of the Singleton Pattern in Unity.

```
using UnityEngine;

public class AudioHandler : MonoBehaviour
{
    // Public field, set in the inspector we ca
    // the audio clip through the singleton ins
    public AudioClip explosionClip;

    // Static singleton property
    public static AudioHandler Instance { get;

    void Awake()
    {
        // Save a reference to the AudioHandler
        Instance = this;
    }

    // Instance method, this method can be acce
    public void PlayAudio(AudioClip clip)
    {
        audio.clip = clip;
        audio.Play();
    }
}
```

Now if you want to call the PlaySound() method from another class, you simply do it like this.

```
// Play sound
AudioHandler.Instance.PlayAudio(AudioHandler.Ins
```

A great thing, as you see in the above example, is that you have easy access to the public variables set in the inspector. We can access the sound clips from the inspector through our singleton instance.

Let's try and look at another more complex implementation of the Singleton Pattern

```
using UnityEngine;

public class AudioHandler : MonoBehaviour
{
    // Public field, set in the inspector we ca
    // the audio clip through the singleton ins
    public AudioClip explosionClip;

    // Static singleton property
    public static AudioHandler Instance { get;

    void Awake()
    {
        // First we check if there are any othe
        if(Instance != null && Instance != this
        {
            // If that is the case, we destroy
            Destroy(gameObject);
        }

        // Here we save our singleton instance
        Instance = this;

        // Furthermore we make sure that we don
        DontDestroyOnLoad(gameObject);
    }

    // Instance method, this method can be acce
    public void PlayAudio(AudioClip clip)
    {
        audio.clip = clip;
        audio.Play();
    }
}
```

This implementation ensures that there can only be a single instance of the class, even if you accidently have more than one applied as a component in Unity. It also has the advantage/disadvantage depending on the context, of not getting destroyed between different scenes. Another feature that you might find good or bad is that the static instance variable doesn't get reset between play sessions.

In a true singleton implementation, we have something called **lazy instantiation**. What this means is that the singleton instance itself, will not be created before you try to access it the first time around. This will generally spare you some resources, and is another great advantage of the singleton pattern. Now the reason we didn't have any lazy instantiation in the above example, is because we want to be able to assign our audio clips in the inspector. This means that even before your game starts, the instance already exists, because you applied as a component by dragging and dropping.

I'm going to round up this post with a general implementation of the singleton pattern with lazy instantiation, which you can use if you are not going to assign any inspector fields.

```
using UnityEngine;

public class Singleton : MonoBehaviour
{
    // This field can be accesed through our si
    // but it can't be set in the inspector, be
    public int number;

    // Static singleton instance
    private static Singleton instance;

    // Static singleton property
    public static Singleton Instance
    {
        // Here we use the ?? operator, to retu
        // otherwise we assign instance to a ne
        get { return instance ?? (instance = ne
    }

    // Instance method, this method can be acce
    public void DoSomeAwesomeStuff()
    {
        Debug.Log("I'm doing awesome stuff");
    }
}
```

The use is exactly the same; you call methods through the singleton instance.

```
// Do awesome stuff through our singleton insta
Singleton.Instance.DoSomeAwesomeStuff();
```

Now because of lazy instantiation, the first time you use the singleton instance, it is initialized and a new GameObject appears in the hierarchy.

And we can see that our awesome method actually got called in the console.

I hope you will find the singleton pattern useful, and if you have any trouble, improvements or feedback please leave a comment below 🙂

Comments

11 comments

10 Comments Sort by **Newest**

Add a comment...

Mathias Sjørslev Søholm ·
Aarhus Universitet

This comment box works fine Tor Esa Vestergaard.

Like · Reply · 5y

Tor Esa Vestergaard ·
Co-Founder & Programmer at Sirenix

Haha, yeah... it's because you're using Facebook. DoNotTrack+ kills that kind of stuff 😛

Like · Reply · 5y

Adrianus Kevin Kusuma ·
Swiss German University

nice tutorial there.

Like · Reply · 4y

Brad Hammond ·
Melbourne

Why have I not learned about this until now... Time to rehax some of my code 🙂

Like · Reply · 4y

Megan Kossa ·
Web Application Developer at Viasat Inc.

I don't know if you're still checking comments on these, but I have a question: I have a single game manager object with all my singleton scripts on it, will destroying the instance if another one is created screw everything up, especially with things I assigned via the inspector which may differ in each level? Should I apply each of my singletons to a separate game object in that case?

Like · Reply · 4y

Sindri Jóelsson ·
Oslo

Don't use the null coalescing operator, since it will check the actual object reference and might return an object that has been destroyed.

Like · Reply · 2y

Kostas Tzouvalidis

Hey, a little bit late but I hope someone can answer this question:)
I have a GameManager script and I had some variables such as an integer (Level Pointer), but when I was trying to access it from an instance in the Main Player script, that variable hadn't changed at all! So I decided to make it static and worked just fine(I haven't solved the problem without using the static variable)
Anyway, do you recommend me to make more static variables to my GameManager script in the future? Is it a common game mechanic to "use"?

mechanic to "use"?
Thanks!

Like · Reply · 2y

Xavi Viscious ·
Full Stack developer at Wouldn't you like to know

Had a bit of trouble understanding the difference between static classes and singletons before reading this. Great tutorial!

Like · Reply · 2y

Philippe Bonnet Doring ·
Graphics Software Engineer at Various Companies

Bravo Dude, that's the bomb.

Like · Reply · 1y

Jhoni Soares ·
Petrópolis

Hey, great post. The last two images are broken 😟 could you fix it? Or someone share it?

Like · Reply · 1y

Facebook Comments Plugin

Powered by Facebook Comments (http://peadig.com/wordpress-plugins/facebook-comments/)

⇐ Unity3D Tips #1 – Realistic Repeating SFX (http://clearcutgames.net/home/?p=321)

Learning Games Expo ⇒ (http://clearcutgames.net/home/?p=482)

ARCHIVES

November 2012 (http://clearcutgames.net/home/?m=201211)

July 2012 (http://clearcutgames.net/home/?m=201207)

April 2012 (http://clearcutgames.net/home/?m=201204)

January 2012 (http://clearcutgames.net/home/?m=201201)

November 2011 (http://clearcutgames.net/home/?m=201111)

August 2011 (http://clearcutgames.net/home/?m=201108)

June 2011 (http://clearcutgames.net/home/?m=201106)

May 2011 (http://clearcutgames.net/home/?m=201105)

March 2011 (http://clearcutgames.net/home/?m=201103)

January 2011 (http://clearcutgames.net/home/?m=201101)

December 2010 (http://clearcutgames.net/home/?m=201012)

:)