Instances & Singletons

Documentation

Benoit Poyser

June 19, 2022

Index

Download Package

https://drive.google.com/drive/folders/1CtDAuO-iw2M_kv1-Eo0cF-uFopKaXx-D?usp=sharing

Instance

Description

An instance is a reference to an object, class, script, element, etc.

You can get the object's instance ID with the command: object.GetIntanceID();

```
//Gets this object's intance ID
Debug.Log($"{name}: {GetInstanceID()}");
```

More Resources

lordofduct

Joined: Oct 3, 2011
Posts: 8,032

yes and no... you're answer isn't wrong, but it isn't exactly right either. Sorry I can't give you your preferred answer. I COULD leave it at "NO", but then you wouldn't know.

It's not a script, it's an instance of a script. It's not a class, it's an instance of a class. Each gameobject in your scene is an instance of a GameObject.

It's like you are an instance of human. You are A human... you are not THE human.

That gameobject isn't THE gameobject, it is 1 of many gameobjects.

The code you write is the THE, definite article. Where as you instantiate, make instances of, duplicate, whatever english word you want to say, things that are of the type of that code organized together as a script/class.

https://forum.unity.com/threads/what-is-an-instance.279537/

Singleton

Description

A singleton is a programming pattern used to get a single instance of an object. It ensures that only exists a single instance of the object we are trying to access.

Implementation

1. Create ‘instance” private static variable

```
//This must be static, unchanging
private static T _instance;
```

2. (Optional) Create “onlyDestroyScript”, “createIfNull”, “destroyIfNotThis” and “dontDestroyOnLoad” variables to add more control over the behaviors of the singleton.

```
#region Inspector Variables
[Header("Singleton")]
[Tooltip("Set it true if you want to preserve this object during scene loading")]
[SerializeField] protected bool _dontDestroyOnLoad = false;
#endregion

#region Private Variables
//This must be static, unchanging
private static T _instance;

//If you want to change this values from a child class do it before calling base.Awake();
protected static bool _createIfNull = true;
protected static bool _onlyDestroyScript = false;
protected static bool _destroyIfNotThis = true;
#endregion
```

3. Create a property (get & set) of instance. In the get function validate if there's an existent instance, if not then create it.

```
//Allows to access this class instance from other classes

//Using the get function you can look for this object in the current
//scene or create a new one if it doens't exists
public static T Instance
{
    get
    {
        if (_instance == null)
        {
            //Looks for an object in the scene that contains a script of 'T' type
            _instance = FindObjectOfType<T>();

            //If there isn't a instance in the scene, then create it
            if (_instance == null && _createIfNull)
            {
                GameObject newInstance = new GameObject(typeof(T).Name);
                _instance = newInstance.AddComponent<T>();
            }
        }

        return _instance;
    }
}
```

4. Create a protected virtual Awake() function. Check if there's already an instance of this class. If there's already one then Destroy() this instance. In the same Awake() call DontDestroyOnLoad() if that's something you are looking for.

```
#region Methods
//To create Awake in derived classes, use an override Awake and call base.Awake()
protected virtual void Awake()
{
    //Destroys this instance if there's already one
    if (_instance != null && _instance != this)
    {
        if (_destroyIfNotThis)
        {
            //You can either destory only the script instance
            if (_onlyDestroyScript) Destroy(this);
            //Or the whole game object instance
            else Destroy(gameObject);
        }

        return;
    }

    //If there ins't yet a instance, then assings this object as the current instance
    _instance = this as T;

    //Preserves an Object during scene loading
    if (_dontDestroyOnLoad) DontDestroyOnLoad(gameObject);
}
```

5. Create a new class that inherits SingletonPattern and send the same class as a type

```
public class ExampleObject : SingletonPattern<ExampleObject>
{

}
```

6. Override the Awake() function if you need to use a custom Awake()

```
public class ExampleObject : SingletonPattern<ExampleObject>
{
    protected override void Awake()
    {
        //Modify _createIfNull, _onlyDestroyScript, _destroyIfNotThis and _dontDestroyOnLoad here
        //_createIfNull = false;
        //_onlyDestroyScript = true;
        //_destroyIfNotThis = false;
        //_dontDestroyOnLoad = true;

        //Calls the SingletonPattern virtual Awake()
        base.Awake();
    }
}
```

7. Now you can get the instance of this child class by calling class.Instance from any other class in the project.

```
public class ExampleManager : MonoBehaviour
{
    void Start()
    {
        Debug.Log($"Current Intance of ExampleObject: {ExampleObject.Instance.name}");
    }
}
```

More Resources

https://videlais.com/2021/02/20/singleton-global-instance-pattern-in-unity/

Related Topics

Don’t destroy on load
https://docs.unity3d.com/ScriptReference/Object.DontDestroyOnLoad.html

Find Object of Type
https://docs.unity3d.com/ScriptReference/Object.FindObjectOfType.html

Inheritance
https://learn.unity.com/tutorial/inheritance#