

# UnityNaiveGuide

Alternatively checkout \*.pdf version.

## Preface

Probably the main question that you might have once reaching this page and seeing lots of text which varies in quality: “Is it one more Unity Guide?”

- Yes. There are so many damn good resources on the web that sometimes it is difficult to find them all until you search for something very specific. I will put here all of those as well as some other tests I do in spare time (if I am even capable to compete with those bright minds who help improving my everyday code). I hope that set of practises here can be a good starting point for someone. The text will have many alterations as long as I or other contributors learn new things about Unity.

I’d like to mention that not all the tests and recommendations can fit your project. **I believe that every line is written here makes you more suspicious, resulting in validating the information you are after on here** and then contributing to this guide with your clarifications. Highly appreciate this approach, so let’s learn together!

It is important to mention that the tests that are provided in this guide should be considered quite sceptically. I believe that if you really want to compare two or more operations on which performance is better, then it is fair to compare on the level of instructions. However, I don’t have the access to Unity’s source code as well as I am not a specialist to go that deep to disassemble code. That is why all I can do is running some tests, which, sometimes, can be far from what you might face in production.

*PS, if you want to tell me at some point of reading that some of my examples are far-fetched and/or stupid, that they don’t pass peer review in big teams – I am very proud for your team and still sure that not all the teams are like yours!*

## Table of Contents

- Productivity
  - Project Templates
  - Script Templates
- Side By Side Comparison
  - Animator.StringToHash & Shader.PropertyToID
  - GameObject.SetActive vs Component.enabled
  - Instantiate # Productivity

## Script Templates

It is a very handy to adjust your script templates to make files that already have all the required setup. Simple use case can be legal notice that you can also do with code snippets. . . , but isn't having a script template easier? When the size of your project turns really big, it becomes quite handy. In addition to legal notice you can customize your template to make preset for DOTS or any other custom systems, e.g. NightCache C# Script Template.

### Add Custom templates

There are a few ways to make your script templates. The most convinient, easiest is to create a folder called **ScriptTemplates** under your **Assets**. Inside you should create, for instance, **81-C# Script-NewBehaviourScript.cs.txt**. This template should override your default 'C# Script' in the Right-Click menu in 'Project' Unity Editor window. '81' is the number that is used for ordering in that menu, same as order that you put in **CreateAssetMenu()** when make **ScriptableObject**. If you put 82, 83 or any other, with the same ending, your script will appear above/below as a *duplicate*, if the number is the same, but you, lets say, put "81-B# Script", then it should appear above. This way you don't override the script, that is why you should have the part "81-C# Script" identical. However, you are free to change **NewBehaviourScript**, that is how the default name of the script will look like.

*Note:* these script templates do not override script creation if you create with "Add Component" -> "New Script" via Inspector Window. Unfortunately, if you need this, it is available in the Unity app directory, see below paths corresponding to different platforms.

The number of scripts that you/your studio uses can be different. If it is extensive, you can group your options with "\_\_\_\_", **double** underscore. For example, you can have your scripts under **Fancy Studios Templates** like:

```
81-Fancy Studios Templates__C# Class-NewClass.cs.txt
81-Fancy Studios Templates__C# Interface-NewInterface.cs.txt
81-Fancy Studios Templates__C# Scriptable Object-NewScriptableObjectScript.cs.txt
```

A few words about how your template *can* look. I normally avoid **target** and use only **serializedObject**, but it can be a good illustration. Also, for Editor script it is fine if file name and class names are different. So when you create your Editor script you can name it the same as **MonoBehaviour** class, they are in different assemblies.

```
////////////////////////////////////
///  Not production-ready  ///
////////////////////////////////////
```

```
using UnityEditor;
```

```

namespace NAMESPACE
{
    [CustomEditor(typeof(#SCRIPTNAME#))]
    public class #SCRIPTNAME#Editor : Editor
    {
        private void OnEnable()
        {
            #NOTRIM#
        }

        /// <summary>
        /// <see cref="#SCRIPTNAME#"/>
        /// </summary>
        public override void OnInspectorGUI()
        {
            var #SCRIPTNAME_LOWER# = (#SCRIPTNAME#)target;

            EditorGUI.BeginDisabledGroup(true);
            EditorGUILayout.PropertyField(serializedObject.FindProperty("m_Script"));
            EditorGUI.EndDisabledGroup();

            // var serializedPropertyMyInt = serializedObject.FindProperty("");
            // serializedObject.ApplyModifiedProperties();
        }
    }
}

```

- **#SCRIPTNAME#** – Replaces it with the filename on script creation.
- **#SCRIPTNAME\_LOWER#** – Similar as the one above, but changes the first uppercase letter to lowercase. Before if you name your file with lowercase as first letter, it was removing all the lowercase letters until it reaches uppercase. Now Unity adds ‘my’ prefix. So if filename is **scriptMB** then it swaps **#SCRIPTNAME\_LOWER#** to **myScriptMB**.
- **#NOTRIM#** – Lets parser know not to clean this empty line, e.g. empty function. However, it might not have any effect. It can preserve many tabulations which, I recon, you don’t need.

## Original Templates

Locations of original files: \* MacOS \* Unity: \* UnityHub: \*/UnityInstalls/\*/Unity.app/Contents/Resources  
 \* Windows \* Unity: \* UnityHub: \* Linux \* Unity: \* UnityHub:

Handle Manually

## Side By Side Comparison

### Shader.PropertyToID("") and Animator.StringToHash("")

The official pages for `Animator.StringToHash` and `Shader.PropertyToID`, where it clearly say “IDs are used for optimized setters and getters...” and “using property identifiers is more efficient than passing strings...” respectively. So you can consider avoiding strings as the parameter when you call any of these.

### Probably a Better Approach

IDEs, e.g. as Rider, automatically recognise similar cases in your project and suggest to hash strings into ints. However, over time you might encounter duplicates of the same parameters and it is probably better to have a special place for all of them. The example below also allows you to control all the names in one file. There is also an option to make a config file or `ScriptableObject` file for that, but the simplest example is below:

```
namespace Test.Properties
{
    using UnityEngine;

    public static class AnimatorProperties
    {
        public static readonly int RotateYAnimationClip = Animator.StringToHash("RotateY");
        public static readonly int RotateZAnimationClip = Animator.StringToHash("RotateZ");
    }

    public static class ShaderProperties
    {
        public static readonly int ColorShaderProperty = Shader.PropertyToID("_Color");
        public static readonly int TintShaderProperty = Shader.PropertyToID("_Tint");
    }

    // OR
    [CreateAssetMenu(fileName = "Properties", menuName = "ScriptableObjects/Properties")]
    public class PropertiesSO : ScriptableObject
    {
        public readonly int RotateYAnimationClip = Animator.StringToHash("RotateY");
        public readonly int RotateZAnimationClip = Animator.StringToHash("RotateZ");

        public readonly int ColorShaderProperty = Shader.PropertyToID("_Color");
        public readonly int TintShaderProperty = Shader.PropertyToID("_Tint");
    }
}
```

### Example for Animators:

```
namespace Tests.HashingStrings.Animators
{
    using UnityEngine;

    using HashedProperties;

    public class AnimationChangerMB : MonoBehaviour
    {
        [SerializeField]
        protected GameObject sphere1;
        [SerializeField]
        protected GameObject sphere2;

        protected bool Swapper = false;
        protected Animator Sphere1Animator;
        protected Animator Sphere2Animator;

        protected void Awake()
        {
            Sphere1Animator = sphere1.GetComponent<Animator>();
            Sphere2Animator = sphere2.GetComponent<Animator>();
        }

        protected void Update()
        {
            if (!Input.GetKeyDown(KeyCode.Tab))
                return;

            if (Swapper)
            {
                Sphere1Animator.Play(AnimatorProperties.RotateZAnimationClip);
                Sphere2Animator.Play(AnimatorProperties.RotateYAnimationClip);
            }
            else
            {
                Sphere1Animator.Play(AnimatorProperties.RotateYAnimationClip);
                Sphere2Animator.Play(AnimatorProperties.RotateZAnimationClip);
            }
            Swapper = !Swapper;
        }
    }
}
```

### Performance

- Test #1 – Execute with string e.g. `Sphere1Animator.Play("RotateZ");`
- Test #2 – Execute with locally cached int value;
- Test #3 – Execute with static int from static class;
- Test #4 – Same as above, but value was cached before loop;
- Test #5 – Execute with int from ScriptableObject class;
- Test #6 – Same as above, but value was cached before loop.

## Editor

Times	Test #1	Test #2	Test #3	Test #4	Test #5	Test #6
1	0.00014849	0.00007170	0.00000700	0.00001120	0.00000738	0.00001053
1	0.00000515	0.00000505	0.00000321	0.00000467	0.00000475	0.00000306
10	0.00000666	0.00000404	0.00000669	0.00000592	0.00000386	0.00000384
100	0.00002740	0.00002500	0.00001750	0.00002276	0.00001755	0.00001788
1000	0.00033476	0.00015649	0.00020201	0.00015632	0.00015598	0.00023105
10000	0.00240798	0.00155762	0.00133051	0.00140078	0.00166718	0.00153883
100000	0.01805707	0.01114511	0.01113095	0.01122464	0.01120038	0.01113555
1000000	0.14642538	0.09188678	0.09118677	0.09149084	0.09296595	0.09170776
10000000	1.38090207	0.90337447	0.90877167	0.91157750	0.91057538	0.94460000

## IL2CPP MacOS Intel 64-bit

Times	Test #1	Test #2	Test #3	Test #4	Test #5	Test #6
1	0.00000887	0.00000081	0.00000183	0.00000223	0.00000204	0.00000220
1	0.00000457	0.00000221	0.00000232	0.00000203	0.00000191	0.00000214
10	0.00001018	0.00000407	0.00000394	0.00000421	0.00000356	0.00000364
100	0.00002641	0.00001685	0.00001701	0.00001661	0.00001667	0.00001705
1000	0.00024906	0.00014647	0.00016511	0.00016372	0.00016302	0.00014549
10000	0.00235644	0.00149591	0.00156760	0.00150800	0.00151446	0.00147985
100000	0.01664717	0.01035973	0.00960241	0.00968604	0.00914258	0.01052656
1000000	0.10926833	0.06973652	0.07076260	0.07580751	0.07004646	0.07040648
10000000	1.00485862	0.63289100	0.63487495	0.63104334	0.63236006	0.63263087

## Example for Shaders:

If you make different shaders with the same parameter name, e.g. `"_Color"`, there should be no problems to **reuse** it for different shaders. Unity recalculates value, so there should be no collision. You can try it yourself: just make two spheres with different materials & shaders. Have a property of the same name and type in both shaders, the one you'll access with `.SetColor()`, `.SetFloat()` etc.

```
namespace Tests.HashingStrings.Shaders
{
```

```

using UnityEngine;

using HashedProperties;

public class ColorChangerMB : MonoBehaviour
{
    [SerializeField]
    protected GameObject sphere1;
    [SerializeField]
    protected GameObject sphere2;

    protected Material Sphere1Material;
    protected Material Sphere2Material;

    protected void Awake()
    {
        Sphere1Material = sphere1.GetComponent<Renderer>().sharedMaterial;
        Sphere2Material = sphere2.GetComponent<Renderer>().sharedMaterial;
    }

    protected void Update()
    {
        if (!Input.GetKeyDown(KeyCode.Space))
            return;
        var color1R = Random.Range(0f, 1f);
        var color1G = Random.Range(0f, 1f);
        var color1B = Random.Range(0f, 1f);

        Sphere1Material.SetColor(ShaderProperties.ColorShaderProperty,
            new Color(color1R, color1G, color1B, 1f));

        var color2R = Random.Range(0f, 1f);
        var color2G = Random.Range(0f, 1f);
        var color2B = Random.Range(0f, 1f);

        Sphere2Material.SetColor(ShaderProperties.ColorShaderProperty,
            new Color(color1R, color1G, color1B, 1f));
        // Sphere2Material.SetColor(ShaderProperties.ColorShaderProperty,
        //     new Color(color2R, color2G, color2B, 1f));
    }
}

```

Just make two spheres, two materials and two shaders for them. Put the code below in your shaders for `Sphere_1` and `Sphere_2` respectively.

Shader "Unlit/Sphere\_1" *// Shader "Unlit/Sphere\_2"*

```

{
    Properties { _Color ("Main Color", Color) = (1,1,1,1) }
    SubShader
    {
        Tags { "RenderType"="Opaque" }
        LOD 100

        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            struct appdata
            {
                float4 vertex : POSITION;
            };

            struct v2f
            {
                float4 vertex : SV_POSITION;
            };

            float4 _Color;

            v2f vert (appdata v) {
                v2f o;
                o.vertex = UnityObjectToClipPos(v.vertex);
                return o;
            }

            fixed4 frag (v2f i) : SV_Target
            {
                return _Color;          // Shader "Unlit/Sphere_1"
                return 1 - _Color;      // Shader "Unlit/Sphere_2"
            }
            ENDCG
        }
    }
}

```

## Performance

- Test #1 – Execute with string e.g. `Sphere1Material.SetColor("_Color", SomeColour);`
- Test #2 – Execute with locally cached int value;



- Test #3 – Execute with static int from static class;
- Test #4 – Same as above, but value was cached before loop;
- Test #5 – Execute with int from ScriptableObject class;
- Test #6 – Same as above, but value was cached before loop.

## Editor

Times	Test #1	Test #2	Test #3	Test #4	Test #5	Test #6
1	0.00000831	0.00007536	0.00002213	0.00000792	0.00000742	0.00000848
1 again	0.00000742	0.00000383	0.00000318	0.00000312	0.00000321	0.00000183
10	0.00000503	0.00000458	0.00000430	0.00000523	0.00000454	0.00000515
100	0.00002602	0.00001307	0.00001299	0.00001517	0.00001715	0.00001714
1000	0.00026692	0.00012060	0.00011721	0.00015282	0.00015758	0.00011767
10000	0.00206858	0.00093458	0.00114345	0.00094270	0.00093802	0.00093176
100000	0.01528322	0.00805394	0.00815373	0.00802989	0.00792480	0.00802744
1000000	0.12462365	0.06748622	0.06878386	0.07070566	0.06860883	0.06701352
10000000	1.18080147	0.65498759	0.65808497	0.65136666	0.65934728	0.65718837

## IL2CPP MacOS Intel 64-bit

Times	Test #1	Test #2	Test #3	Test #4	Test #5	Test #6
1	0.00001183	0.00000180	0.00000127	0.00000151	0.00000284	0.00000312
1 again	0.00001055	0.00000188	0.00000180	0.00000172	0.00000167	0.00000217
10	0.00000690	0.00000307	0.00000323	0.00000564	0.00000318	0.00000355
100	0.00002648	0.00001332	0.00006365	0.00001357	0.00001278	0.00001380
1000	0.00023381	0.00011365	0.00011388	0.00019378	0.00013290	0.00011368
10000	0.00233917	0.00120554	0.00119545	0.00116674	0.00118861	0.00117835
100000	0.01656655	0.00874680	0.00864122	0.00908912	0.00848267	0.00857170
1000000	0.11665278	0.05940517	0.05452891	0.05607068	0.05616178	0.05674499
10000000	1.00733784	0.49475148	0.50794711	0.49516667	0.49861623	0.49419712

## Opinion

it is hard to imagine that you have a real reason to update your shader property in the same frame for more than 10-100 materials. If you do so, probably you have to consider how to reuse your materials. If you can reuse your materials Unity could properly use batching. It is clearly 1.5-2x boost if avoiding **strings** as a parameter.

Practically you can see that it is fine to have a separate class to keep your properties. From management perspective it is just easier to change things in one place. Static class or a SO is a preference thing, it only can be quite annoying to assign refereneces if you don't make a singleton or don't automate assigning for your SO. Making a local variable for this case is not useful much.

*Test Setup:* there were 100x100 Objects spawned, each had a sphere mesh, one MeshRenderer and 1-16 BoxColliders. Actually also Sphere collider, forgot to remove it, but doesnt matter, I dont update it.

**Prefab structure** - ROOT\_GO - UnityUpdateComponentsEnableMB(Clone)  
 - Sphere (1 MeshRenderer, 1-16 BoxCollider(s), MeshFilter and SphereCollider) - UnityUpdateComponentsEnableMB(Clone) - Sphere (1 MeshRenderer, 1-16 BoxCollider(s), MeshFilter and SphereCollider) - etc, 100x100 of UnityUpdateComponentsEnableMB.

Below are the code samples I used for testing:

```
// UnityUpdateGOSetActiveEachMB.cs
protected void Update()
{
    // T1: it is fair to make a variable same is for T2 and T3
    // sphere is SphereRenderer.gameObject
    var isEnabled = Time.frameCount % 2 == 0;
    sphere.SetActive(isEnabled);
}

// UnityUpdateComponentsEnableMB.cs
protected void Update()
{
    // T2 and T3 were commented out and were tested separately
    var isEnabled = Time.frameCount % 2 == 0;

    // T2: SphereRenderer is MeshRenderer
    SphereRenderer.enabled = isEnabled;

    // T3: SphereBoxColliders is List<BoxCollider>
    for (int i = 0; i < numberOfComponents; ++i)
    {
        SphereBoxColliders[i].enabled = isEnabled;
    }
}
```

In update I 1. Enable/disable whole game object, 2. Mesh renderer, 3. Forloop every box collider

Test	1, number of BoxColliders is 1	2, Test	FPS	1-16	16-32	32-64	64-128	128-256	256-512	512-1024	1024-2048	2048-4096	4096-8192	8192-16384	16384-32768	32768-65536	65536-131072	131072-262144	262144-524288	524288-1048576	1048576-2097152	2097152-4194304	4194304-8388608	8388608-16777216	16777216-33554432	33554432-67108864	67108864-134217728	134217728-268435456	268435456-536870912	536870912-1073741824	1073741824-2147483648	2147483648-4294967296	4294967296-8589934592	8589934592-17179869184	17179869184-34359738368	34359738368-68719476736	68719476736-137438953472	137438953472-274877906944	274877906944-549755813888	549755813888-1099511627776	1099511627776-2199023255552	2199023255552-4398046511104	4398046511104-8796093022208	8796093022208-17592186044416	17592186044416-35184372088832	35184372088832-70368744177664	70368744177664-140737488355328	140737488355328-281474976710656	281474976710656-562949953421312	562949953421312-1125899906842624	1125899906842624-2251799813685248	2251799813685248-4503599627370496	4503599627370496-9007199254740992	9007199254740992-18014398509481984	18014398509481984-36028797018963968	36028797018963968-72057594037927936	72057594037927936-144115188075855872	144115188075855872-288230376151711744	288230376151711744-576460752303423488	576460752303423488-1152921504606846976	1152921504606846976-2305843009213693952	2305843009213693952-4611686018427387904	4611686018427387904-9223372036854775808	9223372036854775808-18446744073709551616	18446744073709551616-36893488147419103232	36893488147419103232-73786976294838206464	73786976294838206464-147573952589676412928	147573952589676412928-295147905179352825856	295147905179352825856-590295810358705651712	590295810358705651712-1180591620717411303424	1180591620717411303424-2361183241434822606848	2361183241434822606848-4722366482869645213696	4722366482869645213696-9444732965739290427392	9444732965739290427392-18889465931478580854784	18889465931478580854784-37778931862957161709568	37778931862957161709568-75557863725914323419136	75557863725914323419136-151115727451828646838272	151115727451828646838272-302231454903657293676544	302231454903657293676544-604462909807314587353088	604462909807314587353088-1208925819614629174706176	1208925819614629174706176-2417851639229258349412352	2417851639229258349412352-4835703278458516698824704	4835703278458516698824704-9671406556917033397649408	9671406556917033397649408-19342813113834066795298816	19342813113834066795298816-38685626227668133590597632	38685626227668133590597632-77371252455336267181195264	77371252455336267181195264-154742504910672534362390528	154742504910672534362390528-309485009821345068724781056	309485009821345068724781056-618970019642690137449562112	618970019642690137449562112-1237940039285380274899124224	1237940039285380274899124224-2475880078570760549798248448	2475880078570760549798248448-4951760157141521099596496896	4951760157141521099596496896-9903520314283042199192993792	9903520314283042199192993792-19807040628566084398385987584	19807040628566084398385987584-39614081257132168796771975168	39614081257132168796771975168-79228162514264337593543950336	79228162514264337593543950336-158456325028528675187087900672	158456325028528675187087900672-316912650057057350374175801344	316912650057057350374175801344-633825300114114700748351602688	633825300114114700748351602688-1267650600228229401496703205376	1267650600228229401496703205376-2535301200456458802993406410752	2535301200456458802993406410752-5070602400912917605986812821504	5070602400912917605986812821504-10141204801825835211973625643008	10141204801825835211973625643008-20282409603651670423947251286016	20282409603651670423947251286016-40564819207303340847894502572032	40564819207303340847894502572032-81129638414606681695789005144064	81129638414606681695789005144064-162259276829213363391578010288128	162259276829213363391578010288128-324518553658426726783156020576256	324518553658426726783156020576256-649037107316853453566312041152512	649037107316853453566312041152512-1298074214633706907132624082305024	1298074214633706907132624082305024-2596148429267413814265248164610048	2596148429267413814265248164610048-5192296858534827628530496329220096	5192296858534827628530496329220096-10384593717069655257060992658440192	10384593717069655257060992658440192-20769187434139310514121985316880384	20769187434139310514121985316880384-41538374868278621028243970633760768	41538374868278621028243970633760768-83076749736557242056487941267521536	83076749736557242056487941267521536-166153499473114484112975882535043072	166153499473114484112975882535043072-332306998946228968225951765070086144	332306998946228968225951765070086144-664613997892457936451903530140172288	664613997892457936451903530140172288-1329227995784915872903807060280344576	1329227995784915872903807060280344576-2658455991569831745807614120560689152	2658455991569831745807614120560689152-5316911983139663491615228241121378304	5316911983139663491615228241121378304-10633823966279326983230456482242756608	10633823966279326983230456482242756608-21267647932558653966460912964485513216	21267647932558653966460912964485513216-42535295865117307932921825928971026432	42535295865117307932921825928971026432-85070591730234615865843651857942052864	85070591730234615865843651857942052864-170141183460469231731687303715884105728	170141183460469231731687303715884105728-340282366920938463463374607431768211456	340282366920938463463374607431768211456-680564733841876926926749214863536422912	680564733841876926926749214863536422912-1361129467683753853853498429727072845824	1361129467683753853853498429727072845824-2722258935367507707706996859454145691648	2722258935367507707706996859454145691648-5444517870735015415413993718908291383296	5444517870735015415413993718908291383296-10889035741470030830827987437816582766592	10889035741470030830827987437816582766592-21778071482940061661655974875633165533184	21778071482940061661655974875633165533184-43556142965880123323311949751266331066368	43556142965880123323311949751266331066368-87112285931760246646623899502532662132736	87112285931760246646623899502532662132736-174224571863520493293247799005065324265472	174224571863520493293247799005065324265472-348449143727040986586495598010130648530944	348449143727040986586495598010130648530944-696898287454081973172991196020261297061888	696898287454081973172991196020261297061888-1393796574908163946345982392040522594123776	1393796574908163946345982392040522594123776-2787593149816327892691964784081045188247552	2787593149816327892691964784081045188247552-5575186299632655785383929568162090376495104	5575186299632655785383929568162090376495104-11150372599265311570767859136324180752990208	11150372599265311570767859136324180752990208-22300745198530623141535718272648361505980416	22300745198530623141535718272648361505980416-44601490397061246283071436545296723011960832	44601490397061246283071436545296723011960832-89202980794122492566142873090593446023921664	89202980794122492566142873090593446023921664-178405961588244985132285746181186892047843328	178405961588244985132285746181186892047843328-356811923176489970264571492362373784095686656	356811923176489970264571492362373784095686656-713623846352979940529142984724747568191373312	713623846352979940529142984724747568191373312-1427247692705959881058285969449495136382746624	1427247692705959881058285969449495136382746624-2854495385411919762116571938898990272765493248	2854495385411919762116571938898990272765493248-5708990770823839524233143877797980545530986496	5708990770823839524233143877797980545530986496-11417981541647679048466287755595961091061972992	11417981541647679048466287755595961091061972992-22835963083295358096932575511191922182123945984	22835963083295358096932575511191922182123945984-45671926166590716193865151022383844364247891968	45671926166590716193865151022383844364247891968-91343852333181432387730302044767688728495783936	91343852333181432387730302044767688728495783936-182687704666362864775460604089535377456991567872	182687704666362864775460604089535377456991567872-365375409332725729550921208179070754913983135744	365375409332725729550921208179070754913983135744-730750818665451459101842416358141509827966271488	730750818665451459101842416358141509827966271488-1461501637330902918203684832716283019655932542976	1461501637330902918203684832716283019655932542976-2923003274661805836407369665432566039311865085952	2923003274661805836407369665432566039311865085952-5846006549323611672814739330865132078623730171904	5846006549323611672814739330865132078623730171904-11692013098647223345629478661730264157247460343808	11692013098647223345629478661730264157247460343808-23384026197294446691258957323460528314494920687616	23384026197294446691258957323460528314494920687616-46768052394588893382517914646921056628989841375232	46768052394588893382517914646921056628989841375232-93536104789177786765035829293842113257979682750464	93536104789177786765035829293842113257979682750464-187072209578355573530071658587684226515959365500928	187072209578355573530071658587684226515959365500928-374144419156711147060143317175368453031918731001856	374144419156711147060143317175368453031918731001856-748288838313422294120286634350736906063837462003712	748288838313422294120286634350736906063837462003712-1496577676626844588240573268701473812127674924007424	1496577676626844588240573268701473812127674924007424-2993155353253689176481146537402947624255349848014848	2993155353253689176481146537402947624255349848014848-5986310706507378352962293074805895248510699696029696	5986310706507378352962293074805895248510699696029696-11972621413014756705924586149611790497021399392059392	11972621413014756705924586149611790497021399392059392-23945242826029513411849172299223580994042798784118784	23945242826029513411849172299223580994042798784118784-47890485652059026823698344598447161988085597568237568	47890485652059026823698344598447161988085597568237568-95780971304118053647396689196894323976171195136475136	95780971304118053647396689196894323976171195136475136-191561942608236107294793378393788647952342390272950304	191561942608236107294793378393788647952342390272950304-383123885216472214589586756787577295904684780545900608	383123885216472214589586756787577295904684780545900608-766247770432944429179173513575154591809369561091801216	766247770432944429179173513575154591809369561091801216-1532495540865888858358347027150309183618739122183602432	1532495540865888858358347027150309183618739122183602432-3064991081731777716716694054300618367237478244367204864	3064991081731777716716694054300618367237478244367204864-6129982163463555433433388108601236734474956488734409728	6129982163463555433433388108601236734474956488734409728-12259964326927110866866776217202473468949912977468819456	12259964326927110866866776217202473468949912977468819456-24519928653854221733733552434404946937899825954937638912	24519928653854221733733552434404946937899825954937638912-49039857307708443467467104868809893875799651909875277
------	--------------------------------	---------	-----	------	-------	-------	--------	---------	---------	----------	-----------	-----------	-----------	------------	-------------	-------------	--------------	---------------	---------------	----------------	-----------------	-----------------	-----------------	------------------	-------------------	-------------------	--------------------	---------------------	---------------------	----------------------	-----------------------	-----------------------	-----------------------	------------------------	-------------------------	-------------------------	--------------------------	---------------------------	---------------------------	----------------------------	-----------------------------	-----------------------------	-----------------------------	------------------------------	-------------------------------	-------------------------------	--------------------------------	---------------------------------	---------------------------------	----------------------------------	-----------------------------------	-----------------------------------	-----------------------------------	------------------------------------	-------------------------------------	-------------------------------------	--------------------------------------	---------------------------------------	---------------------------------------	--	---	---	---	--	---	---	--	---	---	--	---	---	---	--	---	---	--	---	---	--	---	---	---	--	---	---	--	---	---	--	---	---	---	--	---	---	--	---	---	--	---	---	--	---	---	---	--	---	---	--	---	---	--	---	---	---	--	---	---	--	---	---	--	---	---	---	--	---	---	--	---	---	--	---	---	---	--	---	---	--	---	---	--	---	---	---	--	---	---	--	---	---	--	---	---	---	--	---	---	--	---	---	--	---	---	---	--	---	---	--	---	---	--	---	---	---	--	---	---	--	---	---	--	---	--

`GameObject.SetActive()` | 3.4-3.5 | | | 1 `MeshRenderer` | 31-32 | | | 16  
`BoxCollider` for-loop | 1.9-3.1 |

*Results Interpretation:* when there are just a few components attached to a `GameObject`, it is more costly to disable the whole `GameObject`. Disabling particular components would be better. However, when there are more Components, *roughly 6+* per GO, it seems that iterating them in for-loop seems to be less effective than disabling using `GameObject.SetActive(false)`; because it is done better by Unity on the engine level.

The second test evaluates a scenario when Enabled/Disabled `GameObject` has children in the hierarchy versus Enabling/Disabling particular components. This time it is `MeshRenderer`. Below is the structure of prefab:

**Prefab structure** - `ROOT_GO` - `UnityUpdateGOHasChildren(Clone)` - `Sphere (1 MeshRenderer, MeshFilter and SphereCollider)` - `SphereInstantiated(Clone) (1 MeshRenderer, MeshFilter and SphereCollider)`; - `SphereInstantiated(Clone) (1 MeshRenderer, MeshFilter and SphereCollider)`; - `SphereInstantiated(Clone) (1 MeshRenderer, MeshFilter and SphereCollider)`; - etc, Total 1-16 `SphereInstantiated` based on Test №, see below - `UnityUpdateGOHasChildren(Clone)` - `Sphere (1 MeshRenderer, MeshFilter and SphereCollider)` - `SphereInstantiated(Clone) (1 MeshRenderer, MeshFilter and SphereCollider)`; - `SphereInstantiated(Clone) (1 MeshRenderer, MeshFilter and SphereCollider)`; - `SphereInstantiated(Clone) (1 MeshRenderer, MeshFilter and SphereCollider)`; - etc, Total 1-16 `SphereInstantiated` based on Test №, see below - etc, 100x100 of `UnityUpdateGOHasChildren`.

№	Test	FPS
1: 1 <code>BoxCollider</code>		
	Parent <code>GameObject.SetActive()</code>	8-10
	1 <code>MeshRenderer</code> for-loop	21.7-23.3
2: 2 <code>BoxColliders</code>		
	Parent <code>GameObject.SetActive()</code>	5.9-7.7
	2 <code>MeshRenderer</code> for-loop	13.6-16.5
3: 4 <code>BoxColliders</code>		
	Parent <code>GameObject.SetActive()</code>	4.2-4.3
	4 <code>MeshRenderer</code> for-loop	8.9-9.1
4: 8 <code>BoxColliders</code>		
	Parent <code>GameObject.SetActive()</code>	2.6-2.7
	8 <code>MeshRenderer</code> for-loop	4.8-5.2
5: 16 <code>BoxColliders</code>		
	Parent <code>GameObject.SetActive()</code>	~0.2-1*
	16 <code>MeshRenderer</code> for-loop	2.6-2.7

~0.2-1\* the game just didn't manage at that point and I can conclude that having

abusing all the potential of Hierarchies is dangerous.

### **Instantiate<T>()**

Well, everyone knows that it is a bad idea to abuse that function. Just make sure you don't do as well this. *Yes, I saw that many times, so don't make that expensive function take even more time for no reason, all of those can be other parameters of Instantiate<T>().*

```
protected void Spawner()
{
    for (int i = 0; i < YOUR_LIST.Count; ++i)
    {
        var component = Instantiate<YOUR_TYPE>(prefab);
        component.transform.parent = transform;
        component.transform.position = Vector3.zero;
        // something else, mb rotation. Again this is how NOT to do
    }
}
```