

 無料電子ブック

学習

unity3d

Free unaffiliated eBook created from
Stack Overflow contributors.

#unity3d

.....	1
1: unity3d	2
.....	2
.....	2
Examples	5
.....	5
.....	5
.....	5
Unity	6
.....	6
.....	6
Linux	7
.....	7
.....	8
.....	8
.....	10
2: Android101 -	13
.....	13
.....	13
Android	13
.....	13
.....	14
Examples	14
UnityAndroidPlugin.cs	14
UnityAndroidNative.java	14
UnityAndroidPluginGUI.cs	14
3: CullingGroup API	16
.....	16
Examples	16
.....	16
.....	18

.....	19
.....	19
4: MonoBehaviour	21
Examples.....	21
.....	21
5: ScriptableObject	22
.....	22
AssetBundlesScriptableObjects	22
Examples.....	22
.....	22
ScriptableObject	22
ScriptableObject.....	23
ScriptableObjectsPlayMode.....	23
ScriptableObjects.....	24
6: Unity Profiler	25
.....	25
.....	25
.....	25
.....	25
iOS.....	26
Examples.....	26
.....	26
.....	26
7: UnityGit	28
Examples.....	28
UnityGitLFS.....	28
.....	28
GitGit-LFS	28
1Git GUI.....	28
2GitGit-LFS.....	28
Git	28
UnityGit.....	29

.....	29
Unity	30
.....	30
.....	30
8: Unity	32
.....	32
.....	32
Examples	32
RuntimeInitializeOnLoadMethodAttribute	32
Unity CSingleton MonoBehaviour	33
Unity	34
.....	36
-	38
MonoBehaviourScriptableObject	39
9: Vector3	43
.....	43
.....	43
Examples	43
.....	43
Vector3.zero Vector3.one	43
.....	44
.....	46
Vector3	46
.....	46
Vector2 Vector4	47
.....	47
LerpLerpUnclamped	47
MoveTowards	49
SmoothDamp	50
10:	53
Examples	53

.....	53
.....	53
.....	54
.....	54
1.....	55
11:	56
Examples.....	56
.....	56
.unity.....	56
12:	58
.....	58
.....	58
Examples.....	58
.....	58
.....	60
.....	63
.....	67
1.....	67
2.....	69
.....	70
.....	70
.....	70
.....	71
.....	71
EditorWindow.....	71
.....	71
.....	72
.....	75
SceneView.....	75
13:	79
.....	79
Examples.....	79

-	79
14:	80
Examples	80
.....	80
.....	82
.....	84
15:	86
.....	86
Examples	86
.....	86
.....	86
16:	88
.....	88
.....	88
.....	88
.....	88
Examples	88
.....	88
GameObject	89
.....	89
MonoBehaviour	89
GameObject	89
17:	91
.....	91
.....	91
.....	91
YieldInstructions	91
Examples	91
.....	91
.....	93
.....	93

MonoBehaviour.....	94
.....	95
.....	97
18:	99
Examples.....	99
.....	99
.....	99
.....	100
Zip.....	100
.....	101
19:	103
.....	103
Examples.....	103
.....	103
.....	103
.....	104
.....	104
.....	104
.....	105
GameObject.....	105
GameObjectGameObject.....	105
.....	106
20:	108
Examples.....	108
MVC.....	108
21:	112
.....	112
Unity.....	112
Examples.....	112
.....	113
.....	113
.....	113
.....	113

22: VR	117
Examples.....	117
VR.....	117
SDK.....	117
.....	117
VR.....	117
.....	118
23:	120
.....	120
Examples.....	120
.....	120
.....	120
.....	121
.....	122
.....	122
.....	123
.....	123
24:	127
Examples.....	127
.....	127
.....	127
25:	129
.....	129
Examples.....	129
.....	129
TouchPhase	129
26: UI	131
Examples.....	131
.....	131
.....	131

27:	133
Examples	133
.....	133
.....	134
2D	136
.....	138
28:	141
Examples	141
.....	141
.....	141
.....	141
.....	142
.....	143
.....	144
.....	145
29:	147
Examples	147
.....	147
101	147
.....	147
.....	148
.....	149
30:	150
.....	150
Examples	150
.....	150
Physics2D Raycast2D	150
.....	151
.....	152
31:	153
Examples	153

.....	153
.....	153
32:	155
Examples.....	155
GetKeyGetKeyDownGetKeyUp.....	155
.....	156
Advance.....	156
.....	157
.....	158
33: ImGui	161
.....	161
Examples.....	161
GUILayout.....	161
34:	162
.....	162
Examples.....	162
.....	162
.....	162
35:	165
.....	165
.....	165
SerializeField	165
Examples.....	166
.....	166
.....	168
.....	169
.....	170
.....	172
36:	176
.....	176
.....	176
Examples.....	176

isKinematic	184
.....	185
.....	185
.....	186
39:	187
Examples	187
.....	187
.....	187
.....	187
.....	187
.....	188
.....	188
.....	188
.....	189
.....	189
.....	190
.....	190
.....	190
.....	190
.....	191
.....	191
.....	192
.....	193
.....	193
.....	195
.....	195
Trigger Collider Scripting	195
.....	195
40:	197
.....	197
.....	197

Examples.....	197
.....	197
.....	198
.....	202

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [unity3d](#)

It is an unofficial and free unity3d ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official unity3d.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: unity3dのい

Unityは、にクロスプラットフォームのゲームをします。は、ゲームのプログラミングにCおよび/またはJavaScriptシンタックスをしてUnityScriptをできます。ターゲットプラットフォームは、エディタでにりえることができます。すべてのコアゲームコードは、のプラットフォームにするをいてじです。すべてのバージョンとするダウンロードおよびリリースノートのリストは、 <https://unity3d.com/get-unity/download/archive>にあります。

バージョン

バージョン	
Unity 2017.1.0	2017-07-10
5.6.2	2017-06-21
5.6.1	2017-05-11
5.6.0	2017-03-31
5.5.3	2017-03-31
5.5.2	2017-02-24
5.5.1	2017-01-24
5.5	2016-11-30
5.4.3	2016-11-17
5.4.2	2016-10-21
5.4.1	2016-09-08
5.4.0	2016-07-28
5.3.6	2016-07-20
5.3.5	2016520
5.3.4	2016-03-15
5.3.3	2016-02-23
5.3.2	2016-01-28
5.3.1	2015-12-18

バージョン	
5.3.0	2015-12-08
5.2.5	20160601
5.2.4	2015-12-16
5.2.3	2015-11-19
5.2.2	2015-10-21
5.2.1	2015-09-22
5.2.0	2015-09-08
5.1.5	2015-06-07
5.1.4	2015-10-06
5.1.3	2015-08-24
5.1.2	2015-07-16
5.1.1	2015-06-18
5.1.0	2015-06-09
5.0.4	2015-07-06
5.0.3	2015-06-09
5.0.2	2015-05-13
5.0.1	2015-04-01
5.0.0	2015-03-03
4.7.2	2016-05-31
4.7.1	2016-02-25
4.7.0	2015-12-17
4.6.9	2015-10-15
4.6.8	2015-08-26
4.6.7	2015-07-01
4.6.6	2015-06-08

バージョン	
4.6.5	2015-04-30
4.6.4	2015-03-26
4.6.3	2015-02-19
4.6.2	2015-01-29
4.6.1	2014-12-09
4.6.0	2014-11-25
4.5.5	2014-10-13
4.5.4	2014-09-11
4.5.3	2014-08-12
4.5.2	2014-07-10
4.5.1	2014-06-12
4.5.0	2014-05-27
4.3.4	2014-01-29
4.3.3	2014-01-13
4.3.2	2013-12-18
4.3.1	2013-11-28
4.3.0	2013-11-12
4.2.2	20131010
4.2.1	2013-09-05
4.2.0	2013722
4.1.5	2013-06-08
4.1.4	2013-06-06
4.1.3	2013-05-23
4.1.2	2013-03-26
4.1.0	2013-03-13

バージョン	
4.0.1	2013-01-12
4.0.0	2012-11-13
3.5.7	2012-12-14
3.5.6	2012-09-27
3.5.5	2012-08-08
3.5.4	2012-07-20
3.5.3	2012-06-30
3.5.2	2012-05-15
3.5.1	2012-04-12
3.5.0	2012-02-14
3.4.2	2011-10-26
3.4.1	2011-09-20
3.4.0	2011-07-26

Examples

インストールまたはセットアップ

UnityはWindowsとMacでします。 [Linux alpha](#)もあります。

Unityには4のいがあります

1. -
2. プラス - 1ヶあたり35ドル
3. **Pro** - 1につき125ドル - 24かProプランにした、をし、しているバージョンをするオプションがあります。
4. エンタープライズ - [はUnityにいわせてください](#)

EULAによると、にが10ドルをえたまたはは、 **Unity Plus** またはそのライセンスをするがあります。 200,000ドルをえると、 **Unity Pro** または **Enterprise** をするがあります。

インストール

1. [Unityダウンロードアシスタント](#)をダウンロードしてください。
2. アシスタントをして、ダウンロードしてインストールするモジュールUnityエディタ、MonoDevelop IDE、ドキュメント、およびましいプラットフォームビルドモジュールなどをします。

いバージョンをおいのは、[のにアップデート](#)できます。

UnityダウンロードアシスタントなしでUnityをインストールするは、[Unity 5.5.1リリースノート](#)からコンポーネントインストーラを[できます](#)。

Unityのバージョンのインストール

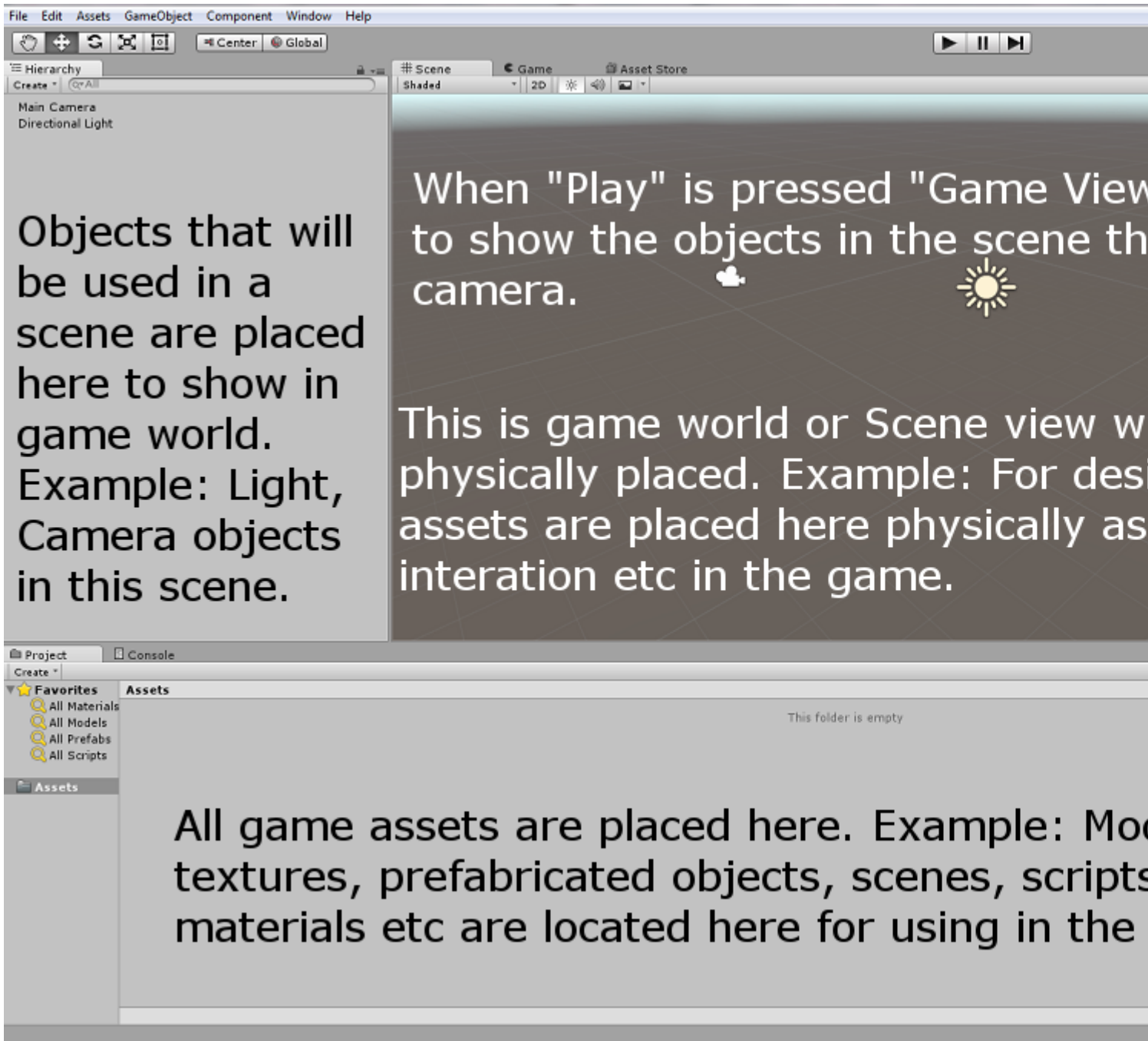
くの、のバージョンのUnityをにインストールするがあります。そうするには

- Windowsでは、のインストールディレクトリを、Unity 5.3.1f1などにしたのフォルダにします。
- Macでは、インストーラはに/Applications/Unityインストールされます。なるバージョンのインストーラをするに、のインストールにこのフォルダのをします
/Applications/Unity5.3.1f1。
- UnityをするときにAltを押しけると、くプロジェクトをできるようになります。そうしないと、ロードされたのプロジェクトがロードされずな。したくないプロジェクトをするようにすプロンプトがされることがあります。

なエディタとコード

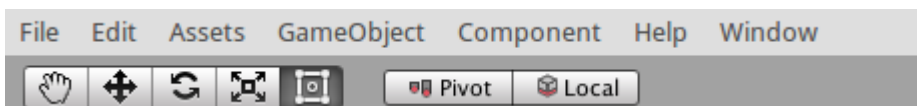
レイアウト

Unityエディタはのようになります。いくつかのデフォルトのウィンドウ/タブのながにされています。



Linuxのレイアウト

のスクリーンショットのように、Linuxバージョンのメニューレイアウトには違いがありますが、



な

ウィンドウをクリックしてのGameObjectし、のをしCreate Empty。プロジェクトウィンドウをクリックし、Create > C# Scriptをしてしいスクリプトをします。にじてをします。

ウィンドウでのGameObjectをしたら、しくしたスクリプトをInspectorウィンドウにドラッグアン

ドロップします。これで、スクリプトはウィンドウのオブジェクトにアタッチされます。デフォルトのMonoDevelop IDEまたはみのスクリプトをききます。

なスクリプト

コードはのようになります `Debug.Log("hello world!!");`。

```
using UnityEngine;
using System.Collections;

public class BasicCode : MonoBehaviour {

    // Use this for initialization
    void Start () {
        Debug.Log("hello world!!");
    }

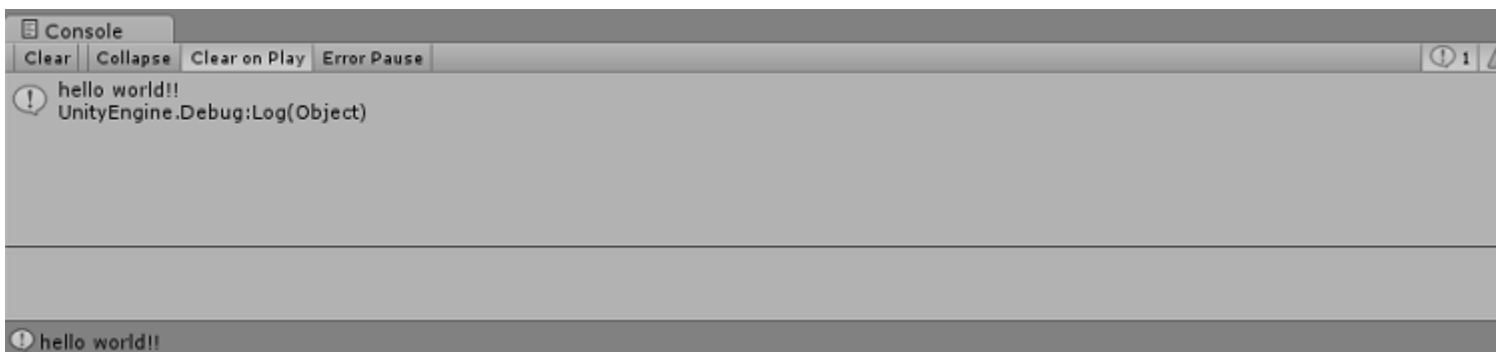
    // Update is called once per frame
    void Update () {

    }

}
```

`Debug.Log("hello world!!");`というをし `Debug.Log("hello world!!"); void Start()`メソッドに `void Start()`。スクリプトをしてエディタにります。エディタのにある「」をしてします。

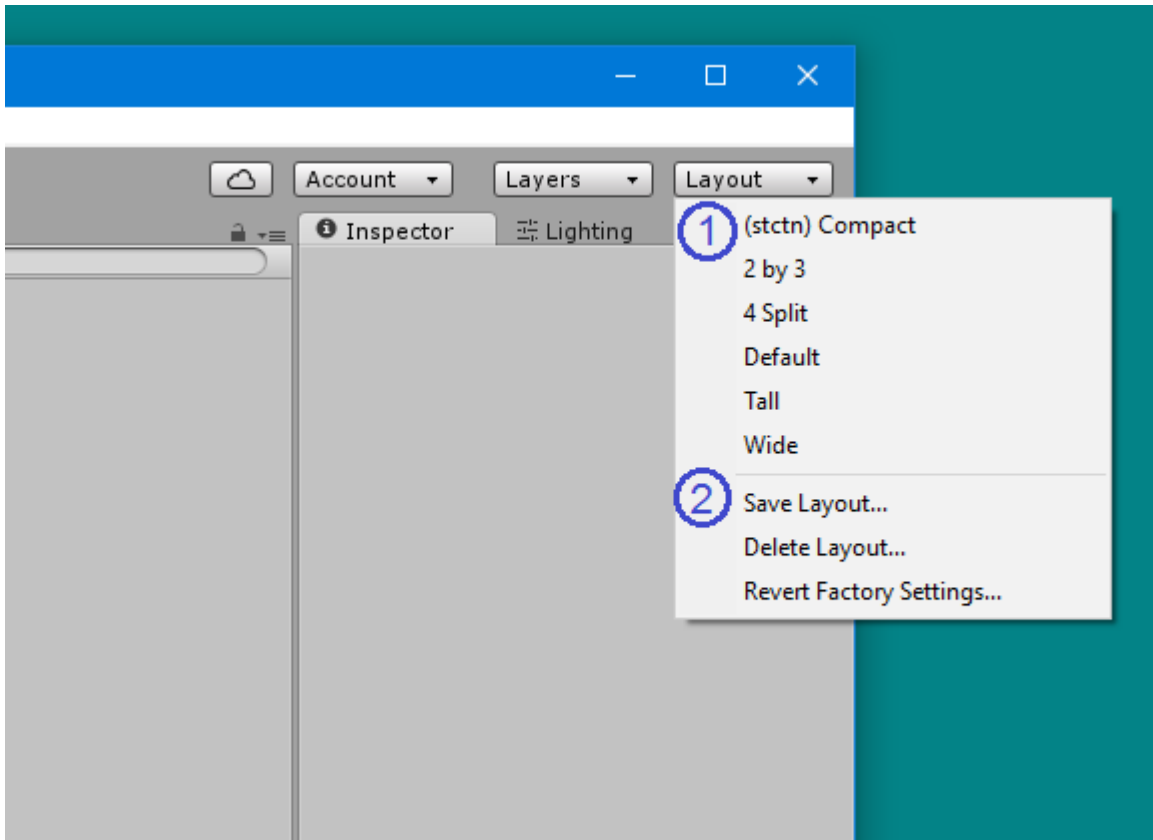
はコンソールウィンドウではのようになります



エディタのレイアウト

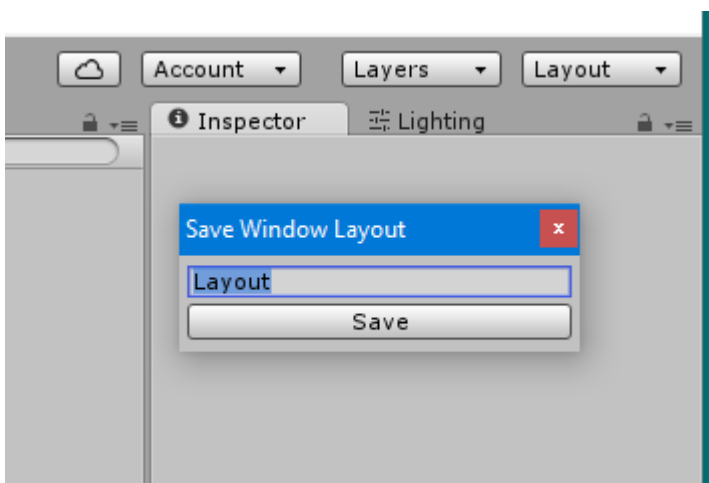
タブとウィンドウのレイアウトをして、をすることができます。

レイアウトメニューはUnity Editorのにあります

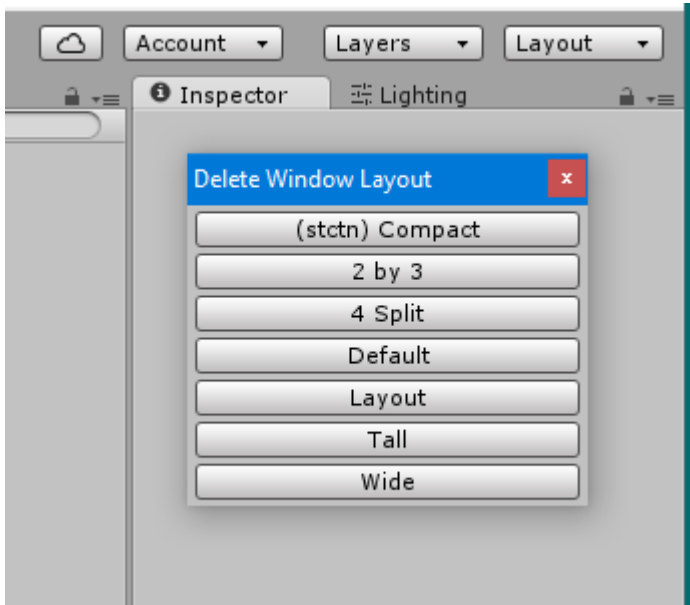


Unityには5つのデフォルトレイアウト2x3、4、デフォルト、さ、ワイド1でマークがしていません。のでは、デフォルトのレイアウトとはに、にカスタムレイアウトもあります。

メニューの「レイアウトの...」ボタンをクリックすると、のレイアウトをできます2でマーク。



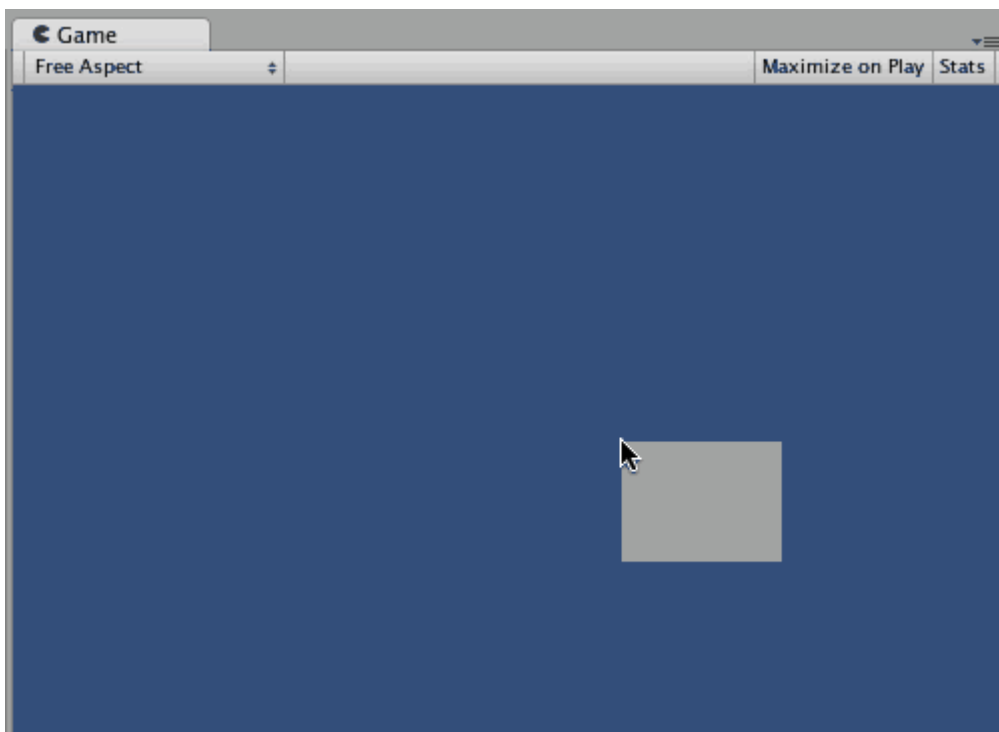
メニューの「レイアウトの...」ボタン2でマークをクリックすると、レイアウトをすることもできます。



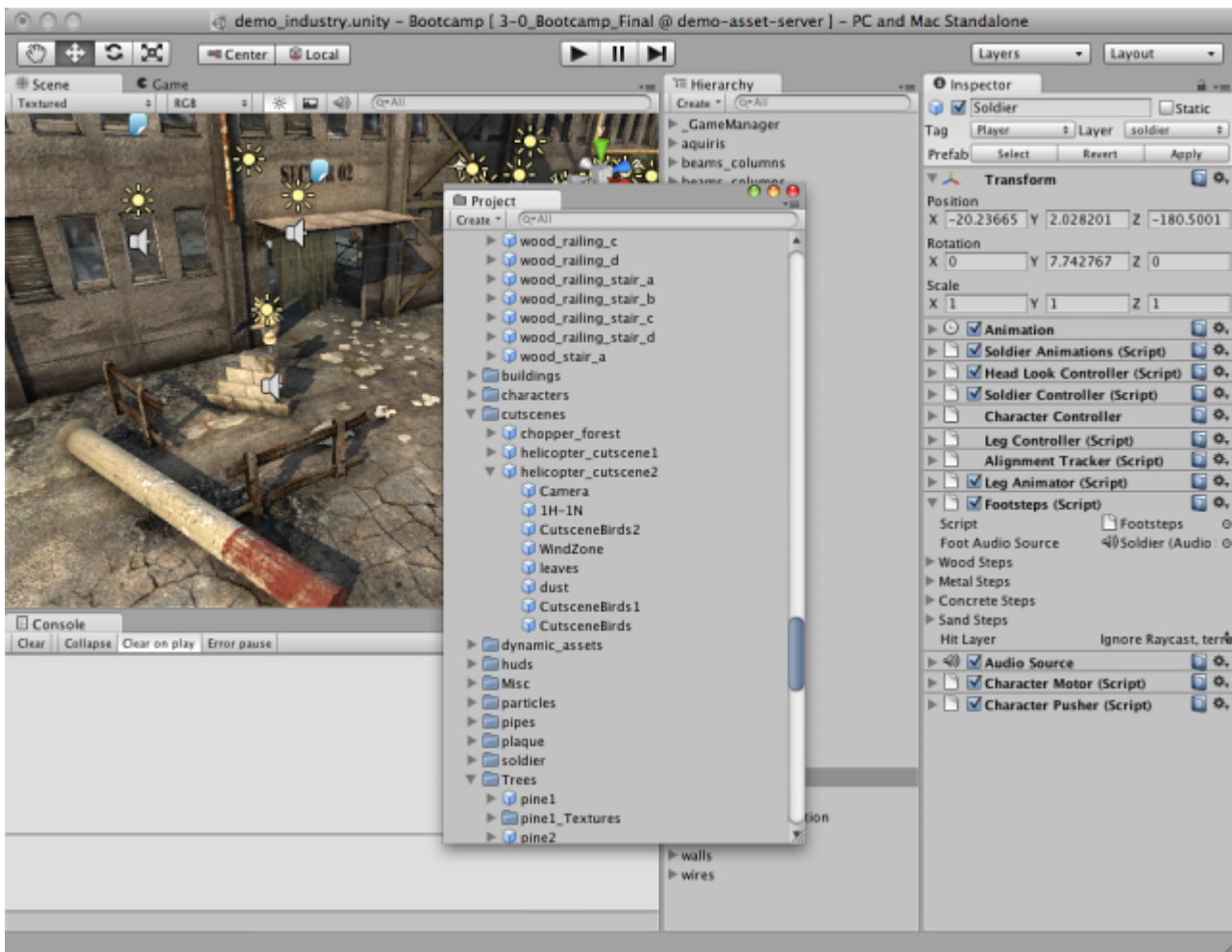
[をにす]ボタンをクリックすると、すべてのカスタムレイアウトがされ、デフォルトのレイアウトがされます2でマークされています。

ワークスペースのカスタマイズ

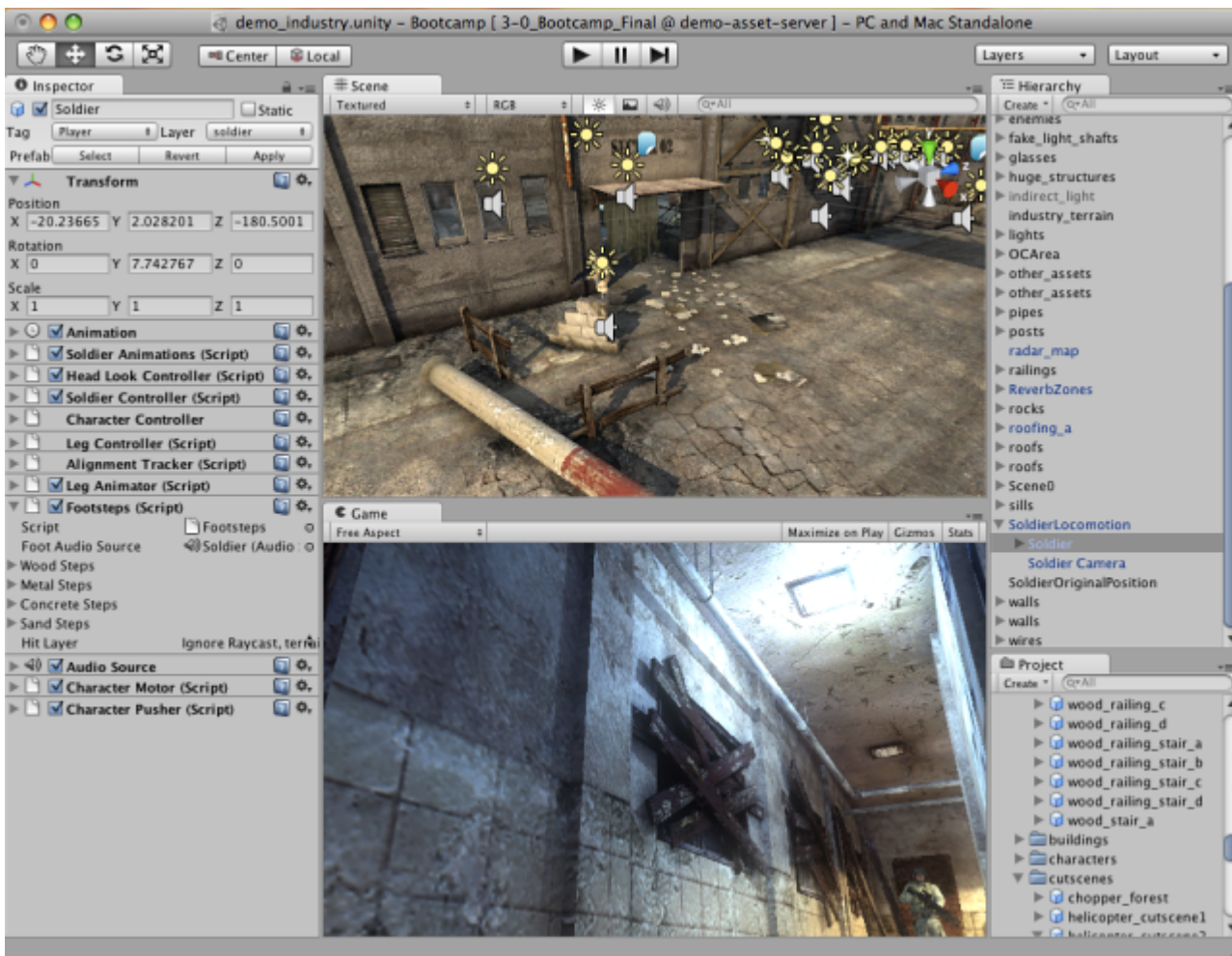
のビューのタブをいくつかのの1つをクリックドラッグして、ビューのレイアウトをカスタマイズできます。のウィンドウのタブエリアにタブをドロップすると、のタブのにタブがされます。または、のDockゾーンにタブをドロップすると、しいウィンドウにビューがされます。



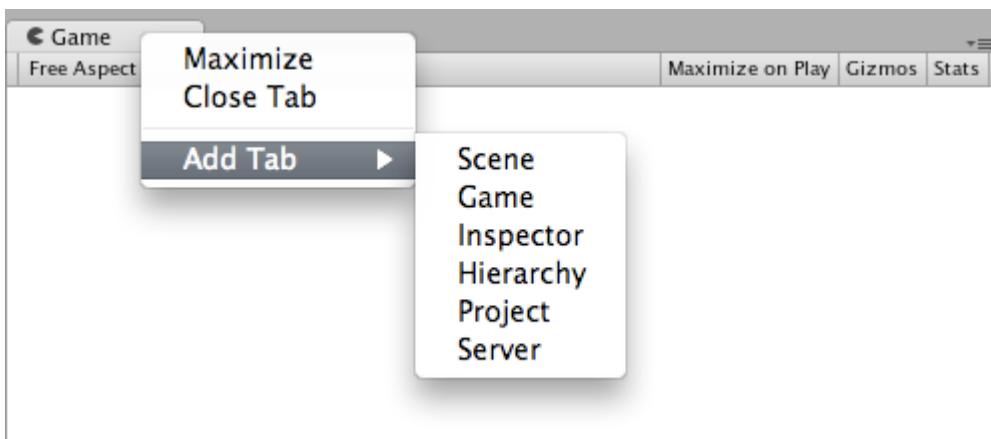
タブは、メインエディタウィンドウからりして、のフローティングエディタウィンドウにすることもできます。フローティングウィンドウには、メインエディタウィンドウとに、ビューとタブのをめることができます。



エディタレイアウトをしたら、レイアウトをしていつでもできます。 [エディタのレイアウトについては、このをしてください。](#)



のビューのタブをクリックすると、いつでもなどのオプションをしたり、じウインドウにしいタブをすることができます。



オンラインでunity3dのいをもむ <https://riptutorial.com/ja/unity3d/topic/846/unity3dのい>

2: Android プラグイン 101 - はじめに

き

このトピックは、UnityのAndroidプラグインをするのシリーズのものです。プラグインやAndroid OSのがほとんどないは、こちらからめてください。

このシリーズをして、はあなたにんでいただきたいリンクをくしています。コンテンツのいえバージョンがここにまれています、のがつがあります。

Androidのプラグインから

、UnityはネイティブAndroidコードをびす2つのをしています。

1. ネイティブAndroidコードをJavaでし、CをしてこれらのJavaをびします。
2. Android OSのであるをびすためのCコードをく

ネイティブコードとするために、Unityはいくつかのクラスとをします。

- [AndroidJavaObject](#) - これは、Unityがネイティブコードとするためにするクラスです。ネイティブコードからされるほとんどすべてのオブジェクトは、[AndroidJavaObject](#)
- [AndroidJavaClass](#) - [AndroidJavaObject](#)からします。これは、ネイティブコードのクラスをするためにされます
- ネイティブオブジェクトのインスタンスの[Get / Set](#)、および[GetStatic / SetStatic](#)バージョン
- ネイティブをびすための[Call / CallStatic](#)

プラグインとのの

1. [Android Studio](#)でネイティブJavaコードをする
2. JAR / AARファイルでコードをエクスポートする [JARファイル](#)と[AARファイル](#)のはこちら
3. JAR / AARファイルを**Assets / Plugins / Android**の Unityプロジェクトにコピーします。
4. プラグインのをびすためにUnityにコードをくCはいつもここにくだった

の3つのは、ネイティブプラグインをするにのみされます。

ここからは、JAR / AARファイルをネイティブプラグインとしてし、CスクリプトをCラッパーとしてします

プラグインのの

プラグインをするのがくかかれていることはすぐにかかります。そのため、ルートをするのはしいようです。ただし、1はカスタムコードをびすためのです。では、どのようにするのですか

にえば、あなたのプラグインです

1. カスタムコードをめる - 1を
2. ネイティブのAndroidのみをびしますか - 2をする

2つのを「ミックス」しないでくださいつまり、1をするプラグインのと2をするプラグインの。まったくですが、することはしばしばです。

Examples

UnityAndroidPlugin.cs

UnityでしいCスクリプトをし、そのをのものにきえます

```
using UnityEngine;
using System.Collections;

public static class UnityAndroidPlugin {

}
```

UnityAndroidNative.java

Android StudioでしいJavaクラスをし、そのをのものにきえます

```
package com.axs.unityandroidplugin;
import android.util.Log;
import android.widget.Toast;
import android.app.ActivityManager;
import android.content.Context;

public class UnityAndroidNative {

}
```

UnityAndroidPluginGUI.cs

UnityでしいCスクリプトをし、これらのをりけます

```
using UnityEngine;
using System.Collections;
```

```
public class UnityAndroidPluginGUI : MonoBehaviour {  
  
    void OnGUI () {  
  
    }  
  
}
```

オンラインでAndroidプラグイン101 - はじめにをむ

<https://riptutorial.com/ja/unity3d/topic/10032/androidプラグイン101----はじめに>

3: CullingGroup API

CullingGroupsのほずしもではないので、マネージャクラスにあるロジックのをカプセルすることです。

そのようなマネージャがどのようにするかについてはのりです。

```
using UnityEngine;
using System;
public interface ICullingGroupManager
{
    int ReserveSphere();
    void ReleaseSphere(int sphereIndex);
    void SetPosition(int sphereIndex, Vector3 position);
    void SetRadius(int sphereIndex, float radius);
    void SetCullingEvent(int sphereIndex, Action<CullingGroupEvent> sphere);
}
```

は、されたのインデックスをすマネージャからのカリングをすることです。その、されたインデックスをしてみのをします。

Examples

オブジェクトのをく

のは、CullingGroupsをしてにってをするをしています。

このスクリプトはにするためにされており、いくつかのパフォーマンスのをしています。

```
using UnityEngine;
using System.Linq;

public class CullingGroupBehaviour : MonoBehaviour
{
    CullingGroup localCullingGroup;

    MeshRenderer[] meshRenderers;
    Transform[] meshTransforms;
    BoundingSphere[] cullingPoints;

    void OnEnable()
    {
        localCullingGroup = new CullingGroup();

        meshRenderers = FindObjectsOfType<MeshRenderer>()
            .Where((MeshRenderer m) => m.gameObject != this.gameObject)
            .ToArray();

        cullingPoints = new BoundingSphere[meshRenderers.Length];
        meshTransforms = new Transform[meshRenderers.Length];
    }
}
```

```

for (var i = 0; i < meshRenderers.Length; i++)
{
    meshTransforms[i] = meshRenderers[i].GetComponent<Transform>();
    cullingPoints[i].position = meshTransforms[i].position;
    cullingPoints[i].radius = 4f;
}

localCullingGroup.onStateChanged = CullingEvent;
localCullingGroup.SetBoundingSpheres(cullingPoints);
localCullingGroup.SetBoundingDistances(new float[] { 0f, 5f });
localCullingGroup.SetDistanceReferencePoint(GetComponent<Transform>().position);
localCullingGroup.targetCamera = Camera.main;
}

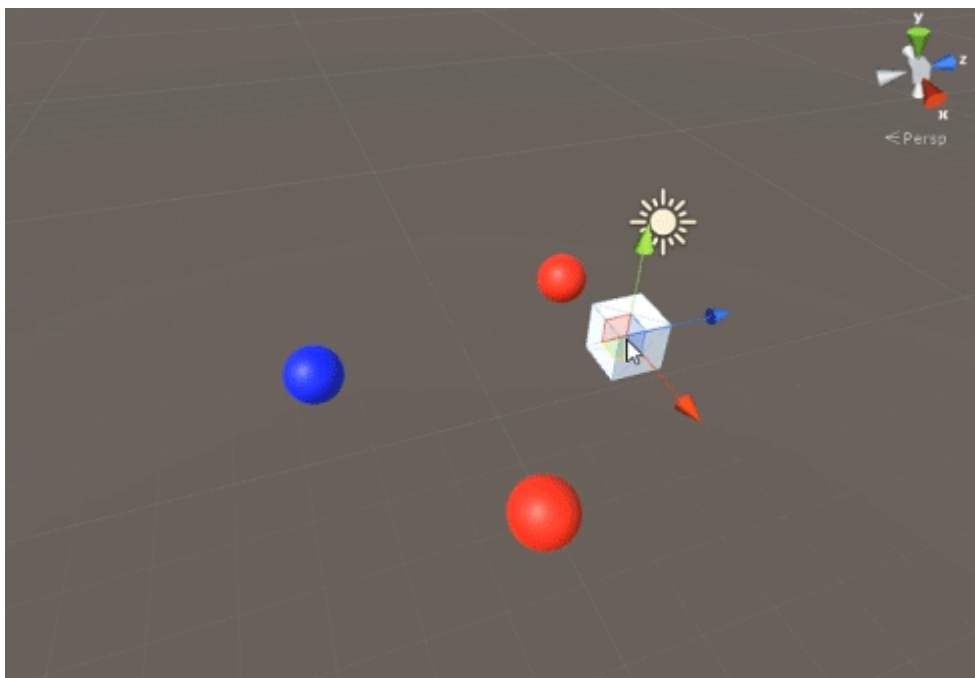
void FixedUpdate()
{
    localCullingGroup.SetDistanceReferencePoint(GetComponent<Transform>().position);
    for (var i = 0; i < meshTransforms.Length; i++)
    {
        cullingPoints[i].position = meshTransforms[i].position;
    }
}

void CullingEvent(CullingGroupEvent sphere)
{
    Color newColor = Color.red;
    if (sphere.currentDistance == 1) newColor = Color.blue;
    if (sphere.currentDistance == 2) newColor = Color.white;
    meshRenderers[sphere.index].material.color = newColor;
}

void OnDisable()
{
    localCullingGroup.Dispose();
}
}

```

スクリプトをGameObjectこのはキューブにし、Playをします。シーンのすべてのゲームオブジェクトは、までのにじてがわかります。



オブジェクトのをす

のスク립トは、されたカメラののにじてイベントをするをしています。

このスク립トでは、にするためにいくつかのパフォーマンスのをしています。

```
using UnityEngine;
using System.Linq;

public class CullingGroupCameraBehaviour : MonoBehaviour
{
    CullingGroup localCullingGroup;

    MeshRenderer[] meshRenderers;

    void OnEnable()
    {
        localCullingGroup = new CullingGroup();

        meshRenderers = FindObjectsOfType<MeshRenderer>()
            .Where((MeshRenderer m) => m.gameObject != this.gameObject)
            .ToArray();

        BoundingSphere[] cullingPoints = new BoundingSphere[meshRenderers.Length];
        Transform[] meshTransforms = new Transform[meshRenderers.Length];

        for (var i = 0; i < meshRenderers.Length; i++)
        {
            meshTransforms[i] = meshRenderers[i].GetComponent<Transform>();
            cullingPoints[i].position = meshTransforms[i].position;
            cullingPoints[i].radius = 4f;
        }

        localCullingGroup.onStateChanged = CullingEvent;
        localCullingGroup.SetBoundingSpheres(cullingPoints);
        localCullingGroup.targetCamera = Camera.main;
    }
}
```

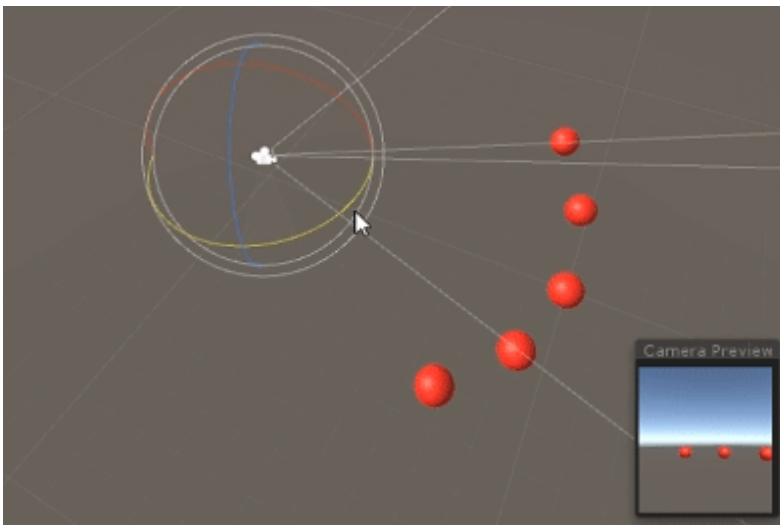
```

void CullingEvent(CullingGroupEvent sphere)
{
    meshRenderers[sphere.index].material.color = sphere.isVisible ? Color.red :
Color.white;
}

void OnDisable()
{
    localCullingGroup.Dispose();
}
}

```

シーンにスクリプトをし、Playをします。シーンのすべてのジオメトリは、についてがわかります。



オブジェクトにMeshRendererコンポーネントがあるは、MonoBehaviour.OnBecameVisible()メソッドをしてのをすることができます。のVector3、Vector3をVector3があるとき、またはオブジェクトのをするされたメソッドをとするときは、Vector3をします。

カリングポイントののにバウンディングをするすることができます。それらは、"い"、"い"、または"にい"のように、カリングポイントのののトリガーになります。

```

cullingGroup.SetBoundingDistances(new float[] { 0f, 10f, 100f});

```

は、でしたにのみをけます。カメラカリングはがありません。

の

、のとなるのは、ののにをします。

まず、カリンググループは、とののをします。2つのがされ、そのかのトリガーになります。このエリアのをして、フィールドをすることができます。


```
float cullingPointArea = Mathf.PI * (cullingPointRadius * cullingPointRadius);  
float boundingArea = Mathf.PI * (boundingDistance * boundingDistance);  
float combinedRadius = Mathf.Sqrt((cullingPointArea + boundingArea) / Mathf.PI);
```

オンラインでCullingGroup APIをむ <https://riptutorial.com/ja/unity3d/topic/4574/cullinggroup-api>

4: MonoBehaviour クラスの

Examples

オーバーライドされないメソッド

Awake、Start、Updateなどのメソッドをオーバーライドするがないのは、クラスでされたメソッドではないためです。

スクリプトがめてアクセスされるとき、スクリプトランタイムはスクリプトをべて、いくつかのメソッドがされているかどうかをべます。これらのがキャッシュされている、そのはキャッシュされ、メソッドはそれぞれのリストにされます。これらのリストは、なるにループされます。

これらのメソッドがでないは、パフォーマンスのためです。すべてのスクリプトにAwake、Start、OnEnable、OnDisable、Update、LateUpdate、FixedUpdateは、これらすべてがリストにされ、これらのメソッドがすべてされることになります。、これはきなではありませんが、これらのメソッドびしはすべてネイティブC++からCにあり、パフォーマンスコストがします。

これをしてみてください。これらのメソッドはすべてリストにあり、ほとんどのメソッドはのメソッドを持っていないかもしれませぬ。これは、もしないメソッドをびすになのパフォーマンスがなることをします。これをぐために、Unityはメソッドをしなようにし、これらのメソッドがにされたときにのみびされるようにするメッセージングシステムをし、なメソッドびしをしました。

ここでUnityブログのについてもっとむことができます [10000 UpdateコールとそのIL2CPP](#)については、[ここをクリックしてください](#) [IL2CPP](#)の

[オンラインでMonoBehaviourクラスのをむ](#)

<https://riptutorial.com/ja/unity3d/topic/2304/monobehaviour> クラスの

5: ScriptableObject

AssetBundles をした ScriptableObjects

スクリプトオブジェクトへのがまれている、プリセットをAssetBundleにするときはしてください。ScriptableObjectsはにアセットなので、UnityはそれらをAssetBundlesにするにをします。これにより、にましくないがするがあります。

このようなGameObjectをAssetBundleからロードするときには、ScriptableObjectアセットをロードされたスクリプトにし、バンドルされたスクリプトをきえるがあります。 [Dependency Injection](#) をしてください。

Examples

き

ScriptableObjectsはMonoBehavioursのようにシーンやゲームオブジェクトにバインドされていないオブジェクトです。には、プロジェクトのアセットファイルにバインドされたデータとメソッドです。これらのScriptableObjectアセットは、メソッドにアクセスできるMonoBehavioursまたはのScriptableObjectsにすことができます。

シリアライズされたアセットとしてののため、れたマネージャクラスとデータソースをします。

ScriptableObject アセットの

は、なScriptableObjectのです。

```
using UnityEngine;

[CreateAssetMenu(menuName = "StackOverflow/Examples/MyScriptableObject")]
public class MyScriptableObject : ScriptableObject
{
    [SerializeField]
    int mySerializedNumber;

    int helloWorldCount = 0;

    public void HelloWorld()
    {
        helloWorldCount++;
        Debug.LogFormat("Hello! My number is {0}.", mySerializedNumber);
        Debug.LogFormat("I have been called {0} times.", helloWorldCount);
    }
}
```

CreateAssetMenu

をクラスにすると、Unityはそれを**Assets / Create**サブメニューにします。これは、**Assets / Create / StackOverflow / Examples**のにあります。

したScriptableObjectインスタンスは、インスペクタをしてのスク립トやScriptableObjectにすることができます。

```
using UnityEngine;

public class SampleScript : MonoBehaviour {

    [SerializeField]
    MyScriptableObject myScriptableObject;

    void OnEnable()
    {
        myScriptableObject.HelloWorld();
    }
}
```

コードをして**ScriptableObject**インスタンスをする

ScriptableObject.CreateInstance<T>()をしてしいScriptableObjectインスタンスをします。

```
T obj = ScriptableObject.CreateInstance<T>();
```

ここで、TはScriptableObjectます。

それらのコンストラクタをびすことによってScriptableObjectをししないでください。

```
new ScriptableObject()。
```

にコードでScriptableObjectをすることは、にデータのシリアルがされるため、めったにびされません。このでクラスをすることもできます。エディタのをスク립ティングするとき、よりです。

ScriptableObjectsは、PlayModeでもエディタでシリアルされます

ScriptableObjectインスタンスのシリアルされたフィールドにアクセスするときは、ながです。

フィールドがpublicとマークされてpublicか、SerializeFieldでシリアルされている、そのをします。MonoBehavioursのようにプレイモードをしてもしリセットされません。これは々にちますが、をくもあります。

このため、シリアルライズされたフィールドをみにし、パブリックフィールドをにけることがベストです。

```
public class MyScriptableObject : ScriptableObject
{
    [SerializeField]
    int mySerializedValue;
```

```
public int MySerializedValue
{
    get { return mySerializedValue; }
}
}
```

セッションでリセットされるScriptableObjectにパブリックをするは、のパターンのをしてください。

```
public class MyScriptableObject : ScriptableObject
{
    // Private fields are not serialized and will reset to default on reset
    private int mySerializedValue;

    public int MySerializedValue
    {
        get { return mySerializedValue; }
        set { mySerializedValue = value; }
    }
}
```

にのScriptableObjectsをつける

にアクティブな ScriptableObjectをつけるには、 `Resources.FindObjectsOfTypeAll()` し `Resources.FindObjectsOfTypeAll()`。

```
T[] instances = Resources.FindObjectsOfTypeAll<T>();
```

ここで、`T`はするScriptableObjectインスタンスのタイプです。アクティブとは、らかのでメモリにロードされたことをします。

このメソッドはにいたので、りをキャッシュしてにびさないようにしてください。スクリプトでScriptableObjectsをすることをめします。

ヒントよりなのために、のインスタンスコレクションをすることができます。

`OnEnable()` にScriptableObjectsをコレクションにさせます。

オンラインでScriptableObjectをむ <https://riptutorial.com/ja/unity3d/topic/3434/scriptableobject>

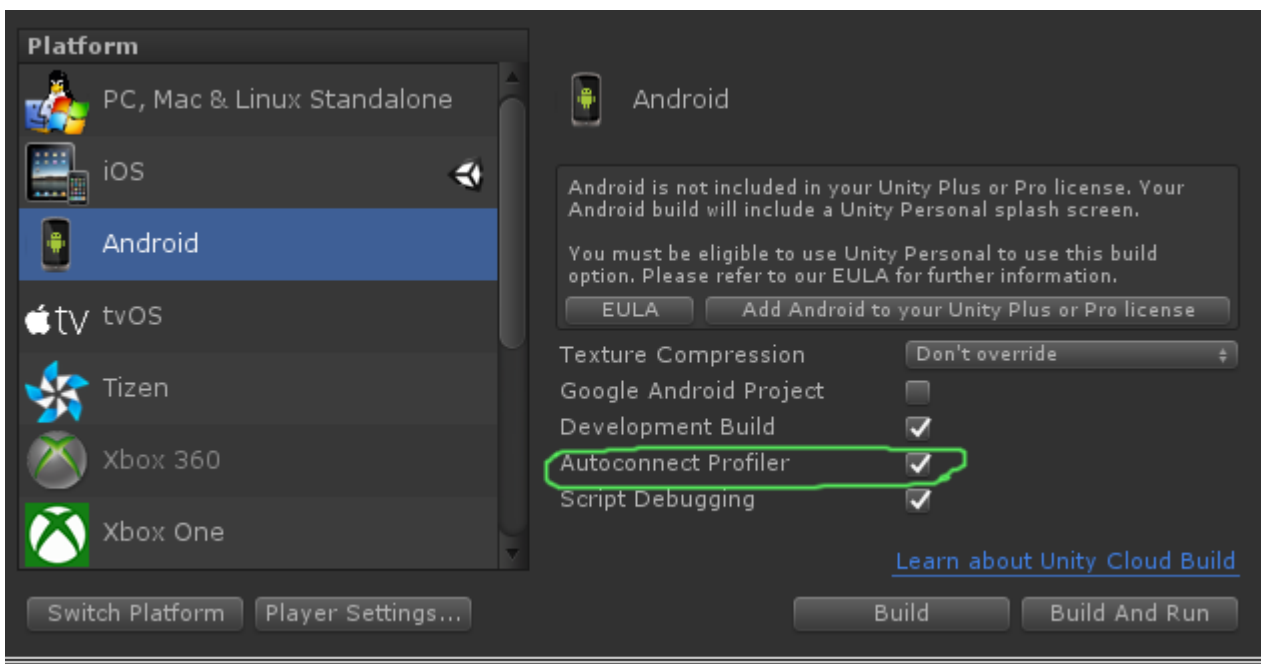
6: Unity Profiler

なるデバイスでのプロファイラの

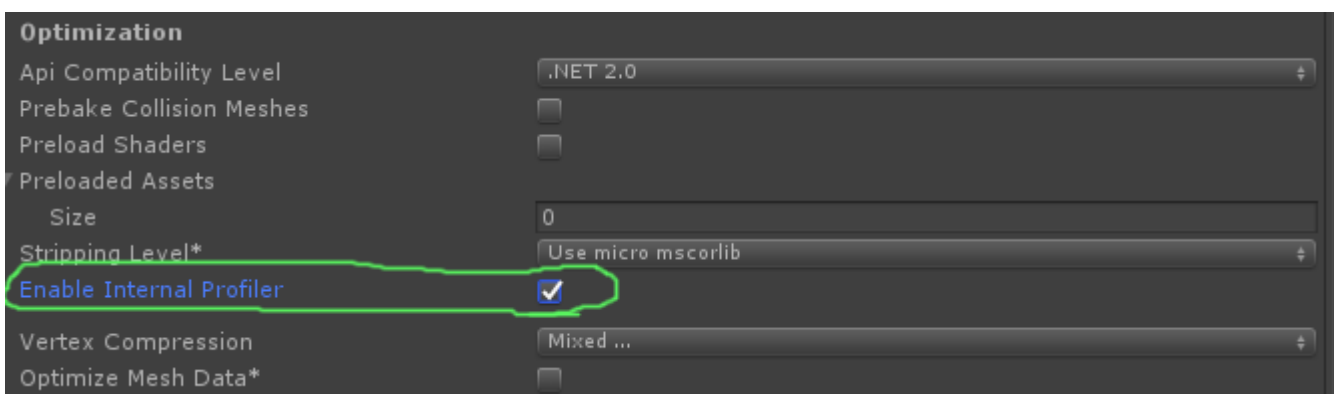
Profilerをなるプラットフォームにするためには、いくつかの必要があります。

アンドロイド

プロファイルをしるするには、[ビルド]ウィンドウの[ビルドと]ボタンをオプションの[オートコネクトプロファイラ]をしてするがあります。



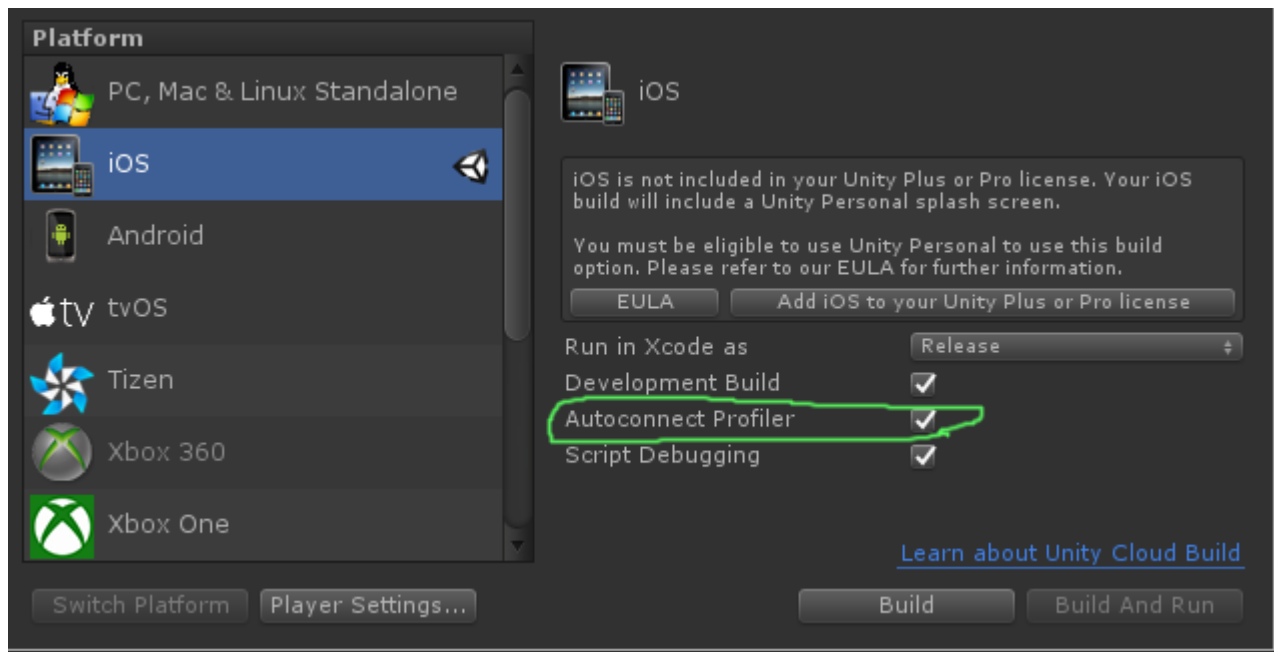
もう一つのオプションは、そののタブのAndroid Playerのインスペクタに、LogCatがプロファイラをするようにチェックするのあるプロファイラをするチェックボックスがあります。



ビルドとはのコマンドラインをしてLogCatでするため、「ビルド」のみをするるとプロファイラはAndroidデバイスにできません。

iOS

プロファイラをしくするには、[ビルド]ウィンドウの[ビルドと]ボタンをオンにして、[プロファイラーの]チェックボックスをオンにするがあります。



iOSでは、プレーヤなので、プロファイラがになるようにするのがあるオプションはありません。それはのするはずです。

Examples

プロファイラマークアップ

プロファイラクラスの

Profilerウィンドウにのエンタリがあるため、Profiler.BeginSampleとProfiler.EndSampleをすることをおめします。

また、これらのタグは、ConditionalAttributeをしてのビルドでされるため、コードからするはありません。

```
public class SomeClass : MonoBehaviour
{
    void SomeFunction()
    {
        Profiler.BeginSample("SomeClass.SomeFunction");
        // Various call made here
        Profiler.EndSample();
    }
}
```

これにより、プロファイラウィンドウに "SomeClass.SomeFunction" というエントリがされ、ボトルネックのデバッグとがになります。

オンラインでUnity Profilerをむ <https://riptutorial.com/ja/unity3d/topic/6974/unity-profiler>

7: UnityでGitソースコントロールをする

Examples

UnityでのGitファイルストレージLFSの

Gitは、そのままのビデオゲームをすることができます。しかし、なは、コミットがするにつれ、きなバージョン> 5 MBのメディアファイルをにすることができることです。Gitはバイナリファイルのバージョンにられたものではありません。

らしいニュースは、2015からGitHubがこのをうGit LFSというプラグインをリリースしたことです。なバイナリファイルをかつにバージョンアップできるようになりました。

に、このドキュメントは、あなたのGitのがビデオゲームのにしていることをにするためのなのとにをてています。このガイドでは、Gitのいについてはれません。

GitGit-LFSのインストール

あなたはとしてできるいくつかのオプションがあります。のは、コアのGitコマンドラインをインストールするか、なGit GUIアプリケーションのどれかをうかどうかです。

オプション1Git GUIアプリケーションをする

これはにはなみです.GitのGUIやGUIをうかどうかというでかなりのオプションがあるからです。あなたはいくつかのアプリケーションをできますが、ここでは3つのアプリケーションがあります

- [Sourcetree](#)
- [Githubデスクトップ](#)
- [SmartGitCommerical](#)

したアプリケーションをインストールしたら、googleをし、Git-LFSのをしてください。このガイドでは、アプリケーションごとにこのステップをします。

オプション2GitGit-LFSをインストールする

これはかなりです - [Gitをインストールしてください](#)。その、[Git LFSをインストールします](#)。

プロジェクトにGitファイルストレージをする

Git LFSプラグインをしてバイナリファイルをよりサポートしているは、Git LFSによってされるいくつかのファイルタイプをするがあります。リポジトリのルートにある.gitattributesファイルにをして、Unityプロジェクトでされるなバイナリファイルをサポートします。

```
# Image formats:
*.tga filter=lfs diff=lfs merge=lfs -text
*.png filter=lfs diff=lfs merge=lfs -text
*.tif filter=lfs diff=lfs merge=lfs -text
*.jpg filter=lfs diff=lfs merge=lfs -text
*.gif filter=lfs diff=lfs merge=lfs -text
*.psd filter=lfs diff=lfs merge=lfs -text

# Audio formats:
*.mp3 filter=lfs diff=lfs merge=lfs -text
*.wav filter=lfs diff=lfs merge=lfs -text
*.aiff filter=lfs diff=lfs merge=lfs -text

# 3D model formats:
*.fbx filter=lfs diff=lfs merge=lfs -text
*.obj filter=lfs diff=lfs merge=lfs -text

# Unity formats:
*.sbsar filter=lfs diff=lfs merge=lfs -text
*.unity filter=lfs diff=lfs merge=lfs -text

# Other binary formats
*.dll filter=lfs diff=lfs merge=lfs -text
```

UnityのGitリポジトリをする

UnityのGitリポジトリをするときには、するがあるいくつかのことがあります。

ユニティフォルダをする

すべてがリポジトリでバージョンされるべきではありません。のテンプレートをリポジトリのルートにある.gitignoreファイルにすることができます。あるいは、[GitHub](#)のオープンソースの[Unity.Gitignore](#)をチェックして、[gitignore.io](#)をつてUnityをすることもできます。

```
# Unity Generated
[Tt]emp/
[Ll]ibrary/
[Oo]bj/

# Unity3D Generated File On Crash Reports
sysinfo.txt

# Visual Studio / MonoDevelop Generated
ExportedObj/
obj/
*.csproj
*.unityproj
*.sln
*.suo
```

```
*.tmp
*.user
*.userprefs
*.pidb
*.booproj
*.svd

# OS Generated
desktop.ini
.DS_Store
.DS_Store?
.Spotlight-V100
.Trashes
ehthumbs.db
Thumbs.db
```

.gitignoreファイルののについては、[こちらをご覧ください](#)。

Unityプロジェクトの

デフォルトでは、Unityプロジェクトはバージョンをしくサポートするようにされていません。

1. v4.5のをしてください Unity → Preferences → Packages → Repository External オプションをにしてください。
2. Edit → Project Settings → Editor → Version Control Mode で、Visible Meta Files にりえ Edit → Project Settings → Editor → Version Control Mode 。
3. Edit → Project Settings → Editor → Asset Serialization Mode Force Text にりえ Edit → Project Settings → Editor → Asset Serialization Mode 。
4. File メニューからシーンとプロジェクトをします。

UnityプロジェクトでGitをするのいくつかのなみのつは、Gitはディレクトリをにせず、ファイルをしてのディレクトリをしておくことです。Unityはこれらのディレクトリに*.meta ファイルをし、Gitがこれらのメタファイルをししたりすると、チームメンバーでしをきこすがあります。

このGitのpost-mergeフックを、Unityプロジェクトがあるリポジトリの/.git/hooks/フォルダにします。Gitをpull / mergeした、どのファイルがされたかをべ、していたディレクトリがであるかどうかをし、するはします。

マージするシーンとプレハブ

Unityをつてするときのは、2のがUnityシーンまたはプリファブ*.unityファイルをしていることです。Gitはそれらをボックスからしくマージするをらない。ありがたいことに、UnityチームはSmartMergeというツールをしました。これはなマージをにいます。にうことは、.gitまたは.gitconfigファイルにのをすることですWindows %USERPROFILE%\gitconfig、Linux / Mac OS X ~/.gitconfig

```
[merge]
tool = unityyamlmerge

[mergetool "unityyamlmerge"]
trustExitCode = false
cmd = '<path to UnityYAMLMerge>' merge -p "$BASE" "$REMOTE" "$LOCAL" "$MERGED"
```

Windowsでは、UnityYAMLMergeのパスはのとおりです。

```
C:\Program Files\Unity\Editor\Data\Tools\UnityYAMLMerge.exe
```

または

```
C:\Program Files (x86)\Unity\Editor\Data\Tools\UnityYAMLMerge.exe
```

MacOSXでは

```
/Applications/Unity/Unity.app/Contents/Tools/UnityYAMLMerge
```

これがすると、マージ/リベースにがしたに、マージツールがになります。 `git mergetool` として UnityYAMLMerge をすることをしないでください。

オンラインでUnityでGitソースコントロールをするをむ

<https://riptutorial.com/ja/unity3d/topic/2195/unityでgitソースコントロールをする>

8: Unityのシングルトン

シングルトンのをけていないがいえであるというのあるがあるのはいありませんが、[gameprogrammingpatterns.com](http://gameprogrammingpatterns.com/Singleton/)のSingletonのように、シームレスなバックグラウンドミュージックのようにのシーンでGameObjectをUnityにさせたいがあります。インスタンスをさせることはできません。シングルトンのな

このスクリプトをGameObjectにすることにより、インスタンスされるとSceneののにするなどして、シーンでアクティブなままになり、インスタンスは1つしかしません。

[ScriptableObject](#) [UnityDoc](#) インスタンスは、ユースケースによってはシングルトンにするなをします。それらはのうちにのインスタンスルールをしません、シーンでをし、Unityのシリアルプロセスをうまくいます。がエディタをしてされると、[Inversion of Control](#)もされます。

```
// MyAudioManager.cs
using UnityEngine;

[CreateAssetMenu] // Remember to create the instance in editor
public class MyAudioManager : ScriptableObject {
    public void PlaySound() {}
}
```

```
// MyGameObject.cs
using UnityEngine;

public class MyGameObject : MonoBehaviour
{
    [SerializeField]
    MyAudioManager audioManager; //Insert through Inspector

    void OnEnable()
    {
        audioManager.PlaySound();
    }
}
```

- [C#でのシングルトン](#)

Examples

[RuntimeInitializeOnLoadMethodAttribute](#)をした

Unity 5.2.5、[RuntimeInitializeOnLoadMethodAttribute](#)をしてMonoBehaviourのをバイパスしてロジックをすることができます。よりクリーンでなをするをします。

```

using UnityEngine;

sealed class GameDirector : MonoBehaviour
{
    // Because of using RuntimeInitializeOnLoadMethod attribute to find/create and
    // initialize the instance, this property is accessible and
    // usable even in Awake() methods.
    public static GameDirector Instance
    {
        get; private set;
    }

    // Thanks to the attribute, this method is executed before any other MonoBehaviour
    // logic in the game.
    [RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.BeforeSceneLoad)]
    static void OnRuntimeMethodLoad()
    {
        var instance = FindObjectOfType<GameDirector>();

        if (instance == null)
            instance = new GameObject("Game Director").AddComponent<GameDirector>();

        DontDestroyOnLoad(instance);

        Instance = instance;
    }

    // This Awake() will be called immediately after AddComponent() execution
    // in the OnRuntimeMethodLoad(). In other words, before any other MonoBehaviour's
    // in the scene will begin to initialize.
    private void Awake()
    {
        // Initialize non-Monobehaviour logic, etc.
        Debug.Log("GameDirector.Awake()", this);
    }
}

```

の

1. GameDirector.OnRuntimeMethodLoad() がしました...
2. GameDirector.Awake()
3. GameDirector.OnRuntimeMethodLoad() しました。
4. OtherMonoBehaviour1.Awake()
5. OtherMonoBehaviour2.Awake() など

Unity C#のシンプルなSingleton MonoBehaviour

ここでは、クラスのプライベートなインスタンスがにされています。

フィールドのはインスタンスでされるため、このクラスのしいインスタンスがされると、`if`はしいインスタンスまたはそのゲームオブジェクトをしてのSingletonオブジェクトへのをつけます。

```

using UnityEngine;

public class SingletonExample : MonoBehaviour {

```

```

private static SingletonExample _instance;

void Awake(){

    if (_instance == null){

        _instance = this;
        DontDestroyOnLoad(this.gameObject);

        //Rest of your Awake code

    } else {
        Destroy(this);
    }
}

//Rest of your class code
}

```

なUnityシングルトン

ここでは、インターネットにあるMonoBehaviourシングルトンのバリエーションを1つにまとめ、グローバルなフィールドにできるようなります。

これは、Unity 5をってテストされました。このシングルトンをするには、のようになるだけです。public class MySingleton : Singleton<MySingleton> {}のAwakeの代わりにAwakeSingletonをするには、AwakeSingletonをオーバーライドするがあります。さらにするには、のフィールドのデフォルトをします。

1. このでは、 **DisallowMultipleComponent** をして、 **GameObject** ごとに1つのインスタンスをします。
2. このクラスはクラスであり、はです。また、のAwakeをするの代わりに、オーバーライドするがあるメソッドAwakeSingletonもまれています。
3. このはスレッドセーフです。
4. このシングルトンはされています。インスタンスのヌルチェックの代わりにインスタンスinstantiatedフラグをinstantiatedことにより、Unityの==のオーバーヘッドをします。
[きをむ](#)
5. このは、Unityによってされようとしているときにシングルトンインスタンスへのびしをしません。
6. このシングルトンには、のオプションがあります。

- FindInactive アクティブなFindInactiveされたじタイプのコンポーネントののインスタンスをすかどうか。
- Persist シーンきているコンポーネントをするかどうか。
- DestroyOthers じタイプののコンポーネントをし、1つだけするかどうか。
- Lazy シングルトンインスタンスを「オンザフライで」Awakeするのか、「オンデマンドでのみ」ゲッターがびされるかのみします。

```

using UnityEngine;

[DisallowMultipleComponent]
public abstract class Singleton<T> : MonoBehaviour where T : Singleton<T>
{
    private static volatile T instance;
    // thread safety
    private static object _lock = new object();
    public static bool FindInactive = true;
    // Whether or not this object should persist when loading new scenes. Should be set in
    Init().
    public static bool Persist;
    // Whether or not destroy other singleton instances if any. Should be set in Init().
    public static bool DestroyOthers = true;
    // instead of heavy comparision (instance != null)
    // http://blogs.unity3d.com/2014/05/16/custom-operator-should-we-keep-it/
    private static bool instantiated;

    private static bool applicationIsQuitting;

    public static bool Lazy;

    public static T Instance
    {
        get
        {
            if (applicationIsQuitting)
            {
                Debug.LogWarningFormat("[Singleton] Instance '{0}' already destroyed on
application quit. Won't create again - returning null.", typeof(T));
                return null;
            }
            lock (_lock)
            {
                if (!instantiated)
                {
                    Object[] objects;
                    if (FindInactive) { objects = Resources.FindObjectsOfTypeAll(typeof(T)); }
                    else { objects = FindObjectsOfType(typeof(T)); }
                    if (objects == null || objects.Length < 1)
                    {
                        GameObject singleton = new GameObject();
                        singleton.name = string.Format("{0} [Singleton]", typeof(T));
                        Instance = singleton.AddComponent<T>();
                        Debug.LogWarningFormat("[Singleton] An Instance of '{0}' is needed in
the scene, so '{1}' was created{2}", typeof(T), singleton.name, Persist ? " with
DontDestroyOnLoad." : ".");
                    }
                    else if (objects.Length >= 1)
                    {
                        Instance = objects[0] as T;
                        if (objects.Length > 1)
                        {
                            Debug.LogWarningFormat("[Singleton] {0} instances of '{1}'!",
objects.Length, typeof(T));
                            if (DestroyOthers)
                            {
                                for (int i = 1; i < objects.Length; i++)
                                {
                                    Debug.LogWarningFormat("[Singleton] Deleting extra '{0}'
instance attached to '{1}'", typeof(T), objects[i].name);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```



```

        Destroy(objects[i]);
    }
}
return instance;
}
return instance;
}
protected set
{
    instance = value;
    instantiated = true;
    instance.AwakeSingleton();
    if (Persist) { DontDestroyOnLoad(instance.gameObject); }
}
}

// if Lazy = false and gameObject is active this will set instance
// unless instance was called by another Awake method
private void Awake()
{
    if (Lazy) { return; }
    lock (_lock)
    {
        if (!instantiated)
        {
            Instance = this as T;
        }
        else if (DestroyOthers && Instance.GetInstanceID() != GetInstanceID())
        {
            Debug.LogWarningFormat("[Singleton] Deleting extra '{0}' instance attached to
'{1}'", typeof(T), name);
            Destroy(this);
        }
    }
}

// this might be called for inactive singletons before Awake if FindInactive = true
protected virtual void AwakeSingleton() {}

protected virtual void OnDestroy()
{
    applicationIsQuitting = true;
    instantiated = false;
}
}
}

```

ベースクラスによるシングルトン

いくつかのシングルトンクラスをとするプロジェクトではたいていの、シングルトンのるいをクラスにすることはきれいでです。

```

using UnityEngine;
using System.Collections.Generic;
using System;

```

```

public abstract class MonoBehaviourSingleton<T> : MonoBehaviour {

    private static Dictionary<Type, object> _singletons
        = new Dictionary<Type, object>();

    public static T Instance {
        get {
            return (T)_singletons[typeof(T)];
        }
    }

    void OnEnable() {
        if (_singletons.ContainsKey(GetType())) {
            Destroy(this);
        } else {
            _singletons.Add(GetType(), this);
            DontDestroyOnLoad(this);
        }
    }
}

```

に、MonoBehaviourは、MonoBehaviourSingletonをすることによってシングルトンパターンをすることができます。このアプローチにより、シングルトンのパターンをにえてパターンをすることができます。

```

using UnityEngine;
using System.Collections;

public class SingletonImplementation : MonoBehaviourSingleton<SingletonImplementation> {

    public string Text= "String Instance";

    // Use this for initialisation
    IEnumerator Start () {
        var demonstration = "SingletonImplementation.Start()\n" +
            "Note that the this text logs only once and\n"
            "only one class instance is allowed to exist.";
        Debug.Log(demonstration);
        yield return new WaitForSeconds(2f);
        var secondInstance = new GameObject();
        secondInstance.AddComponent<SingletonImplementation>();
    }
}

```

シングルトンパターンのの1つは、インスタスへののにアクセスできることです。

```

// Logs: String Instance
Debug.Log(SingletonImplementation.Instance.Text);

```

しかし、カップリングをらすためには、このをにえるがあります。このアプローチには、Dictionaryによるわずかなパフォーマンスコストもありますが、このコレクションにはシングルトンクラスのインスタスが1つしかまかれていないため、DRYをりしてはいけません、はいです

。

ユニティエントティをしたシングルトンパターン - コンポーネントシステム

コアアイデアは、**GameObject**をしてシングルトンをすることです。シングルトンにはのがあります。

- さをにえながら、などのをサポート
- シングルトンは、Entity-ComponentシステムとしてのUnityライフサイクルをちます
- シングルトンは、にとされるえば、ループで、ロードされ、
- フィールドはありません
- のMonoBehaviours /コンポーネントをシングルトンとしてするようにするはありません
- にリセットすることができますただSingletons GameObjectをします、のにびにみまれます
- にモックをするモックでしてからする
- のUnityエディタをしたインスペクションとコンフィグレーション。エディタににするがあります [UnityエディタでアクセスなSingletonのスクリーンショット](#)

Test.csシングルトンのをしています

```
using UnityEngine;
using UnityEngine.Assertions;

public class Test : MonoBehaviour {
    void Start() {
        ExampleSingleton singleton = ExampleSingleton.instance;
        Assert.IsNotNull(singleton); // automatic initialization on first usage
        Assert.AreEqual("abc", singleton.myVar1);
        singleton.myVar1 = "123";
        // multiple calls to instance() return the same object:
        Assert.AreEqual(singleton, ExampleSingleton.instance);
        Assert.AreEqual("123", ExampleSingleton.instance.myVar1);
    }
}
```

ExampleSingleton.csサンプルとのシングルトンクラスがまれています

```
using UnityEngine;
using UnityEngine.Assertions;

public class ExampleSingleton : MonoBehaviour {
    public static ExampleSingleton instance { get { return Singleton.get<ExampleSingleton>(); } }
    public string myVar1 = "abc";
    public void Start() { Assert.AreEqual(this, instance, "Singleton more than once in scene"); }
}

/// <summary> Helper that turns any MonoBehaviour or other Component into a Singleton
</summary>
public static class Singleton {
    public static T get<T>() where T : Component {
        return GetOrAddGo("Singletons").GetOrAddChild("" + typeof(T)).GetOrAddComponent<T>();
    }
    private static GameObject GetOrAddGo(string goName) {
        var go = GameObject.Find(goName);
    }
}
```

```

        if (go == null) { return new GameObject(goName); }
        return go;
    }
}

public static class GameObjectExtensionMethods {
    public static GameObject GetOrAddChild(this GameObject parentGo, string childName) {
        var childGo = parentGo.transform.FindChild(childName);
        if (childGo != null) { return childGo.gameObject; } // child found, return it
        var newChild = new GameObject(childName);          // no child found, create it
        newChild.transform.SetParent(parentGo.transform, false); // add it to parent
        return newChild;
    }

    public static T GetOrAddComponent<T>(this GameObject parentGo) where T : Component {
        var comp = parentGo.GetComponent<T>();
        if (comp == null) { return parentGo.AddComponent<T>(); }
        return comp;
    }
}
}

```

GameObjectの2つのメソッドは、シングルトンクラスでそれらをしてにするがないにも、のでにちます。

MonoBehaviourScriptableObject ベースのシングルトンクラス

ほとんどのシングルトンでは、MonoBehaviourをクラスとしてしています。なは、このSingletonクラスがにのみすることです。これにはいくつかのがあります。

- コードのシングルトンフィールドをすることはありません。
- シングルトンにのアセットへのをすることはありません。
- シングルトンをUnity UIイベントのとしてすることはありません。は "Proxy Component" とばれるものをしてします。そののは、 "GameManager.Instance.SomeGlobalMethod" をびす1のメソッドをつことです。

にされているように、クラスとしてScriptableObjectsをしてこれをしようとするがありますが、MonoBehaviourのランタイムをいます。このでは、にScriptableObjectをクラスおよびするMonoBehaviorとしてすることで、このをしています。

- これはアセットであるため、のUnityアセットのようにそのプロパティをエディタですることが出来ます。
- Unityのシリアライゼーションプロセスでうまくいきます。
- シングルトンのリファレンスをエディタからのアセットにりてることが出来ますはエディタをしてされます。
- Unityイベントは、シングルトンのメソッドをびすことができます。
- "SingletonClassName.Instance" をしてコードベースのどこからでもびすことができます
- Update、Awake、Start、FixedUpdate、StartCoroutineなどのMonoBehaviourイベントとメソッドにアクセスできます。

```

/*****

```

```

* Better Singleton by David Darias
* Use as you like - credit where due would be appreciated :D
* Licence: WTFPL V2, Dec 2014
* Tested on Unity v5.6.0 (should work on earlier versions)
* 03/02/2017 - v1.1
* *****/

using System;
using UnityEngine;
using SingletonScriptableObjectNamespace;

public class SingletonScriptableObject<T> :
SingletonScriptableObjectNamespace.BehaviourScriptableObject where T :
SingletonScriptableObjectNamespace.BehaviourScriptableObject
{
    //Private reference to the scriptable object
    private static T _instance;
    private static bool _instantiated;
    public static T Instance
    {
        get
        {
            if (_instantiated) return _instance;
            var singletonName = typeof(T).Name;
            //Look for the singleton on the resources folder
            var assets = Resources.LoadAll<T>("");
            if (assets.Length > 1) Debug.LogError("Found multiple " + singletonName + "s on
the resources folder. It is a Singleton ScriptableObject, there should only be one.");
            if (assets.Length == 0)
            {
                _instance = CreateInstance<T>();
                Debug.LogError("Could not find a " + singletonName + " on the resources
folder. It was created at runtime, therefore it will not be visible on the assets folder and
it will not persist.");
            }
            else _instance = assets[0];
            _instantiated = true;
            //Create a new game object to use as proxy for all the MonoBehaviour methods
            var baseObject = new GameObject(singletonName);
            //Deactivate it before adding the proxy component. This avoids the execution of
the Awake method when the the proxy component is added.
            baseObject.SetActive(false);
            //Add the proxy, set the instance as the parent and move to DontDestroyOnLoad
scene
            SingletonScriptableObjectNamespace.BehaviourProxy proxy =
baseObject.AddComponent<SingletonScriptableObjectNamespace.BehaviourProxy>();
            proxy.Parent = _instance;
            Behaviour = proxy;
            DontDestroyOnLoad(Behaviour.gameObject);
            //Activate the proxy. This will trigger the MonoBehaviourAwake.
            proxy.gameObject.SetActive(true);
            return _instance;
        }
    }
    //Use this reference to call MonoBehaviour specific methods (for example StartCoroutine)
    protected static MonoBehaviour Behaviour;
    public static void BuildSingletonInstance() {
SingletonScriptableObjectNamespace.BehaviourScriptableObject i = Instance; }
    private void OnDestroy(){ _instantiated = false; }
}

```

```

// Helper classes for the SingletonScriptableObject
namespace SingletonScriptableObjectNamespace
{
    #if UNITY_EDITOR
    //Empty custom editor to have cleaner UI on the editor.
    using UnityEditor;
    [CustomEditor(typeof(BehaviourProxy))]
    public class BehaviourProxyEditor : Editor
    {
        public override void OnInspectorGUI(){}
    }

    #endif

    public class BehaviourProxy : MonoBehaviour
    {
        public IBehaviour Parent;

        public void Awake() { if (Parent != null) Parent.MonoBehaviourAwake(); }
        public void Start() { if (Parent != null) Parent.Start(); }
        public void Update() { if (Parent != null) Parent.Update(); }
        public void FixedUpdate() { if (Parent != null) Parent.FixedUpdate(); }
    }

    public interface IBehaviour
    {
        void MonoBehaviourAwake();
        void Start();
        void Update();
        void FixedUpdate();
    }

    public class BehaviourScriptableObject : ScriptableObject, IBehaviour
    {
        public void Awake() { ScriptableObjectAwake(); }
        public virtual void ScriptableObjectAwake() { }
        public virtual void MonoBehaviourAwake() { }
        public virtual void Start() { }
        public virtual void Update() { }
        public virtual void FixedUpdate() { }
    }
}

```

ここには、たくさんのコメントがあるSingletonScriptableObjectをつたGameManagerシングルトンクラスがあります

```

using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

//this attribute is optional but recommended. It will allow the creation of the singleton via
the asset menu.
//the singleton asset should be on the Resources folder.
[CreateAssetMenu(fileName = "GameManager", menuName = "Game Manager", order = 0)]
public class GameManager : SingletonScriptableObject<GameManager> {

    //any properties as usual
    public int Lives;
}

```

```

public int Points;

//optional (but recommended)
//this method will run before the first scene is loaded. Initializing the singleton here
//will allow it to be ready before any other GameObjects on every scene and will
//will prevent the "initialization on first usage".
[RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.BeforeSceneLoad)]
public static void BeforeSceneLoad() { BuildSingletonInstance(); }

//optional,
//will run when the Singleton Scriptable Object is first created on the assets.
//Usually this happens on edit mode, not runtime. (the override keyword is mandatory for
this to work)
public override void ScriptableObjectAwake(){
    Debug.Log(GetType().Name + " created." );
}

//optional,
//will run when the associated MonoBehaviour awakes. (the override keyword is mandatory
for this to work)
public override void MonoBehaviourAwake(){
    Debug.Log(GetType().Name + " behaviour awake." );

    //A coroutine example:
    //Singleton Objects do not have coroutines.
    //if you need to use coroutines use the attached MonoBehaviour
    Behaviour.StartCoroutine(SimpleCoroutine());
}

//any methods as usual
private IEnumerator SimpleCoroutine(){
    while(true){
        Debug.Log(GetType().Name + " coroutine step." );
        yield return new WaitForSeconds(3);
    }
}

//optional,
//Classic runtime Update method (the override keyword is mandatory for this to work).
public override void Update(){

}

//optional,
//Classic runtime FixedUpdate method (the override keyword is mandatory for this to work).
public override void FixedUpdate(){

}
}

/*
* Notes:
* - Remember that you have to create the singleton asset on edit mode before using it. You
have to put it on the Resources folder and of course it should be only one.
* - Like other Unity Singleton this one is accessible anywhere in your code using the
"Instance" property i.e: GameManager.Instance
*/

```

オンラインでUnityのシングルトンをむ <https://riptutorial.com/ja/unity3d/topic/2137/unityのシングルトン>

9: Vector3

き

Vector3は3Dをし、UnityEngineライブラリのバックボーンの1つです。Vector3は、ほとんどのゲームオブジェクトのTransformコンポーネントでもにされ、やスケールをするためにされます。Vector3は、のベクトルをするためのれたをします。Vector3 APIのVector3についてしくむことができます。

- public Vector3;
- public Vector3 float x、 float y;
- public Vector3 x、 y、 z。
- Vector3.LerpVector3 startPosition、 Vector3 targetPosition、 float movementFraction;
- Vector3.LerpUnclampedVector3 startPosition、 Vector3 targetPosition、 float movementFraction;
- Vector3.MoveTowardsVector3 startPosition、 Vector3 targetPosition、 。

Examples

Vector3には、よくされるVector3をするいくつかのがまれています。ほとんどがをしています、をするためににできます。

Vector3.zero および Vector3.one

Vector3.zeroとVector3.oneは、されたVector3にしてされます。つまり、Vector3 x、 y、 zのがどのように1のきさをするが、Vector3.zero、もいをしVector3.oneをします。

Vector3.zeroは、オブジェクトのデフォルトのをするためによくされます。

のクラスでは、Vector3.zeroとVector3.oneをしてをさせてさせます。

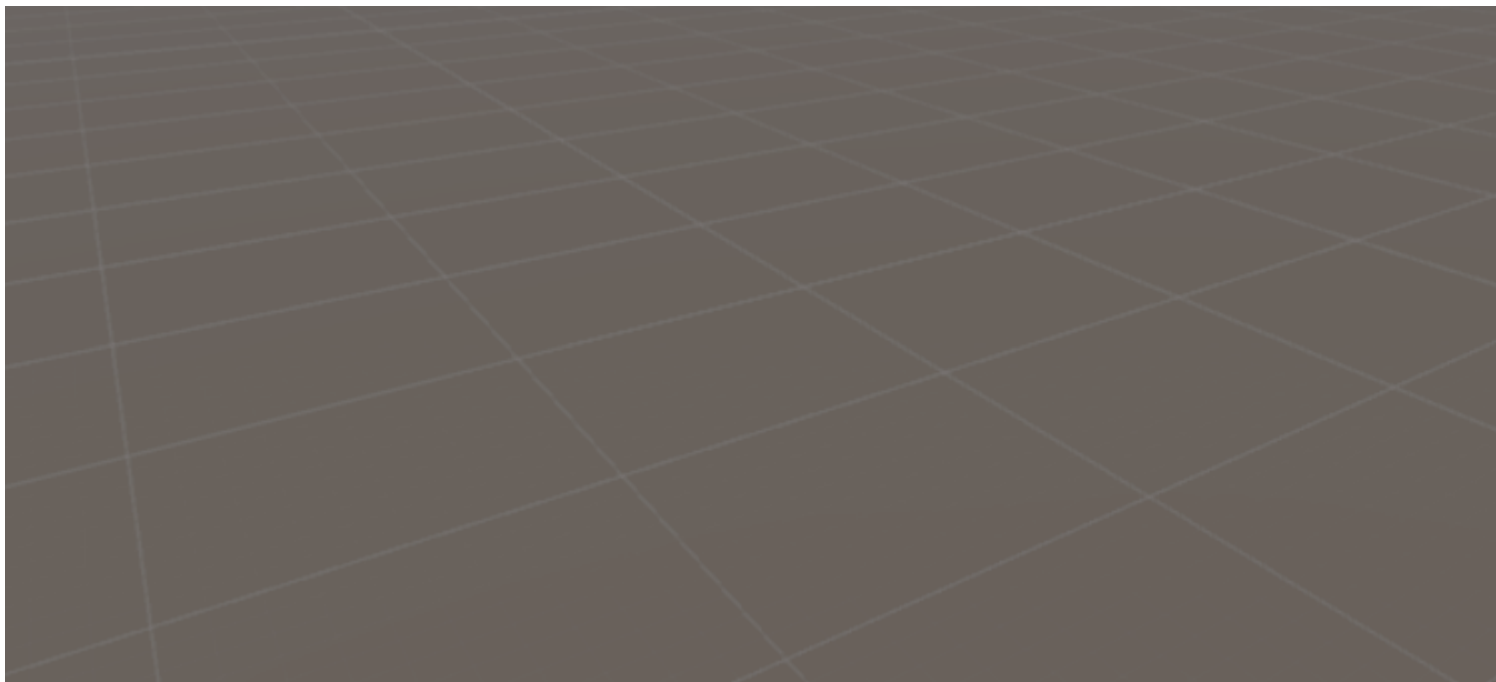
```
using UnityEngine;

public class Inflater : MonoBehaviour
{
    <summary>A sphere set up to inflate and deflate between two values.</summary>
    public ScaleBetween sphere;

    ///<summary>On start, set the sphere GameObject up to inflate
    /// and deflate to the corresponding values.</summary>
    void Start()
    {
        // Vector3.zero = Vector3(0, 0, 0); Vector3.one = Vector3(1, 1, 1);
    }
}
```

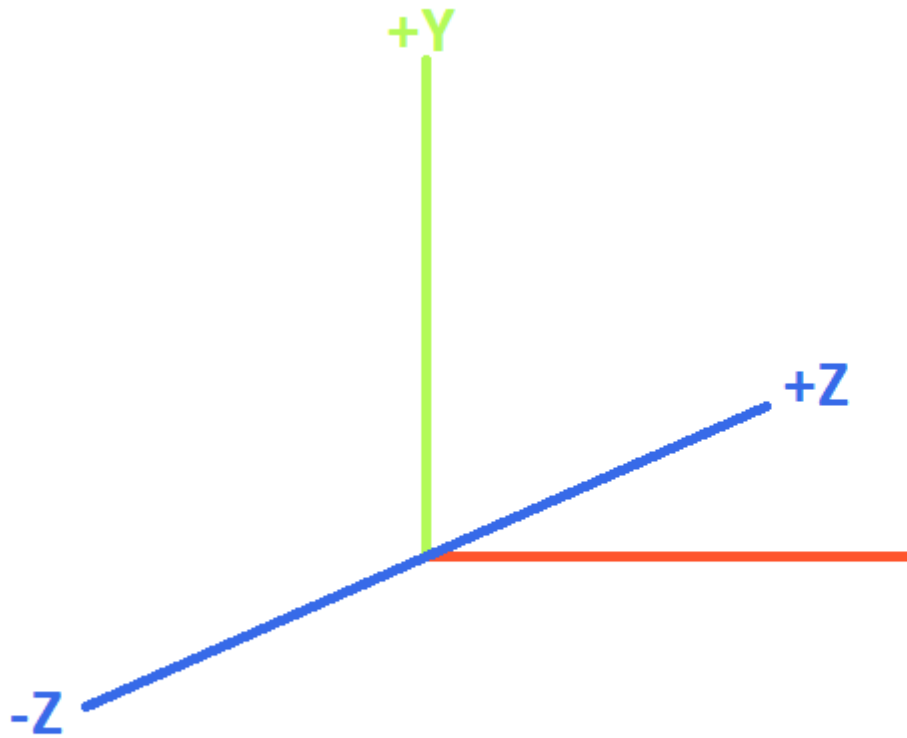


```
sphere.SetScale(Vector3.zero, Vector3.one);  
}  
}
```



な

なは、3つのすべてのおよびにたをうくにおいてである。Unityはにするをすることにする
ことができます。



LEFT-HANDED COORDINATE SYSTEM

のクラスでは、`Vector3` をして3つの方向によってオブジェクトをします。

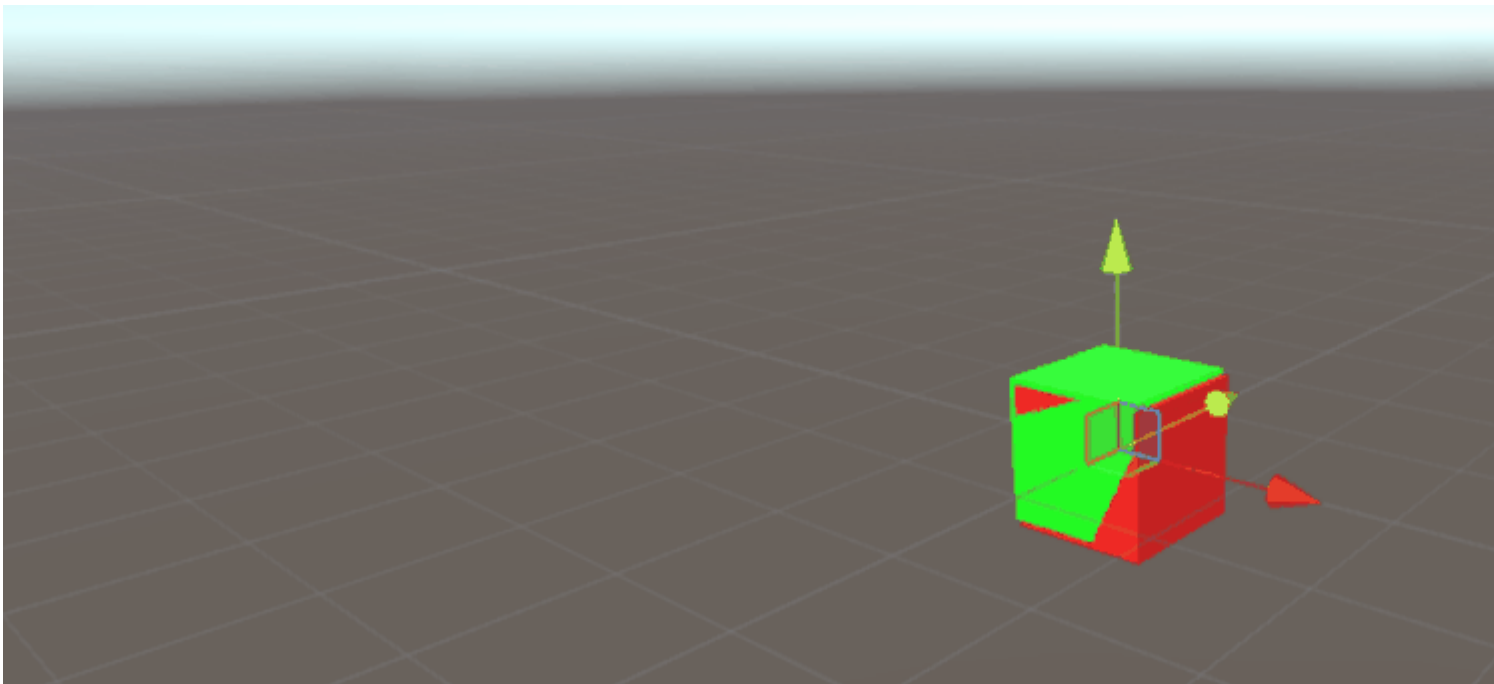
```
using UnityEngine;

public class StaticMover : MonoBehaviour
{
    <summary>GameObjects set up to move back and forth between two directions.</summary>
    public MoveBetween xMovement, yMovement, zMovement;

    ///<summary>On start, set each MoveBetween GameObject up to move
    /// in the corresponding direction(s).</summary>
    void Start()
    {
        // Vector3.left = Vector3(-1, 0, 0); Vector3.right = Vector3(1, 0, 0);
        xMovement.SetDirections(Vector3.left, Vector3.right);

        // Vector3.down = Vector3(0, -1, 0); Vector3.up = Vector3(0, 0, 1);
        yMovement.SetDirections(Vector3.down, Vector3.up);

        // Vector3.back = Vector3(0, 0, -1); Vector3.forward = Vector3(0, 0, 1);
        zMovement.SetDirections(Vector3.back, Vector3.forward);
    }
}
```



インデックス

	バツ	y	z	のnew Vector3() メソッド
Vector3.zero	0	0	0	new Vector3(0, 0, 0)
Vector3.one	1	1	1	new Vector3(1, 1, 1)
Vector3.left	-1	0	0	new Vector3(-1, 0, 0)
Vector3.right	1	0	0	new Vector3(1, 0, 0)
Vector3.down	0	-1	0	new Vector3(0, -1, 0)
Vector3.up	0	1	0	new Vector3(0, 1, 0)
Vector3.back	0	0	-1	new Vector3(0, 0, -1)
Vector3.forward	0	0	1	new Vector3(0, 0, 1)

Vector3の

Vector3はいくつかのことができます。Vector3はなので、はにインスタンスするがあります。

コンストラクタ

Vector3をインスタンスするためのコンストラクタが3つみまれています。

コンストラクタ	
<code>new Vector3()</code>	0、0、0のをつVector3をします。
<code>new Vector3(float x, float y)</code>	えられたxとyをつVector3をします。zは0にされます。
<code>new Vector3(float x, float y, float z)</code>	されたx、y、zをつVector3をします。

Vector2 または Vector4 からの

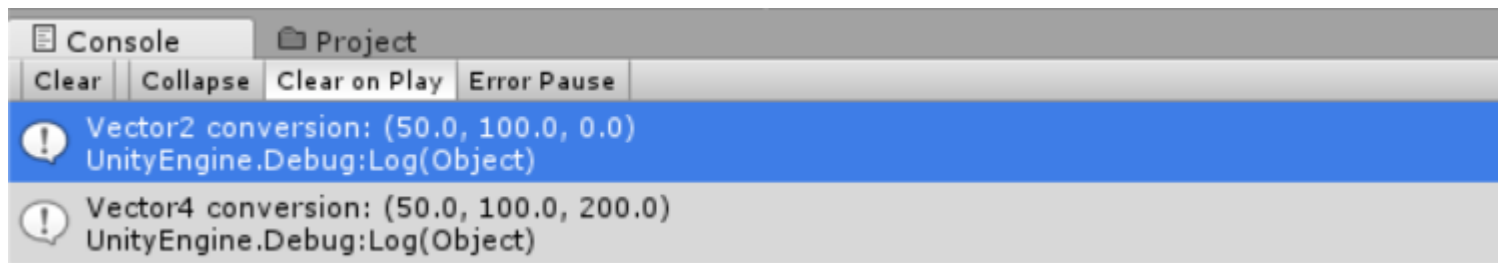
まれている、あなたはのするがあるだろうなにするかもしれVector2またはVector4ようなVector3。このような、あなたはにすことができますVector2またはVector4にVector3にそれをインスタンスせずに、。すると、Vector2はxとyだけをし、Vector4クラスはそのwをします。

のスク립トでをることができます。

```
void VectorConversionTest ()
{
    Vector2 vector2 = new Vector2(50, 100);
    Vector4 vector4 = new Vector4(50, 100, 200, 400);

    Vector3 fromVector2 = vector2;
    Vector3 fromVector4 = vector4;

    Debug.Log("Vector2 conversion: " + fromVector2);
    Debug.Log("Vector4 conversion: " + fromVector4);
}
```



を

Vector3には、Vector3きをするときにながまれています。

Lerp と LerpUnclamped

lerpは、されたについて2つののをします。Lerpが2つののみをする、LerpUnclampedは2つののにするをLerpUnclampedます。

たちは、のをfloatとしてします。が0.5、2つのVector3ののがつかります。が0または1はりますVector3これらののはって1ないきにのいずれかとして、respectivelyをVector3、またはきこれは2Vector3。どちらのものにしないことにすることがです。これはでするがあります。

Lerpでは、すべてのが0と1でクランプされ0。これは、にあってきをしたいにです。また、をオーバーシュートしたくないにもです。LerpUnclampedはのをとることができ、かられるか、をえてするためにできます。

のスク립トは、LerpとLerpUnclampedをしてオブジェクトをのペースでします。

```
using UnityEngine;

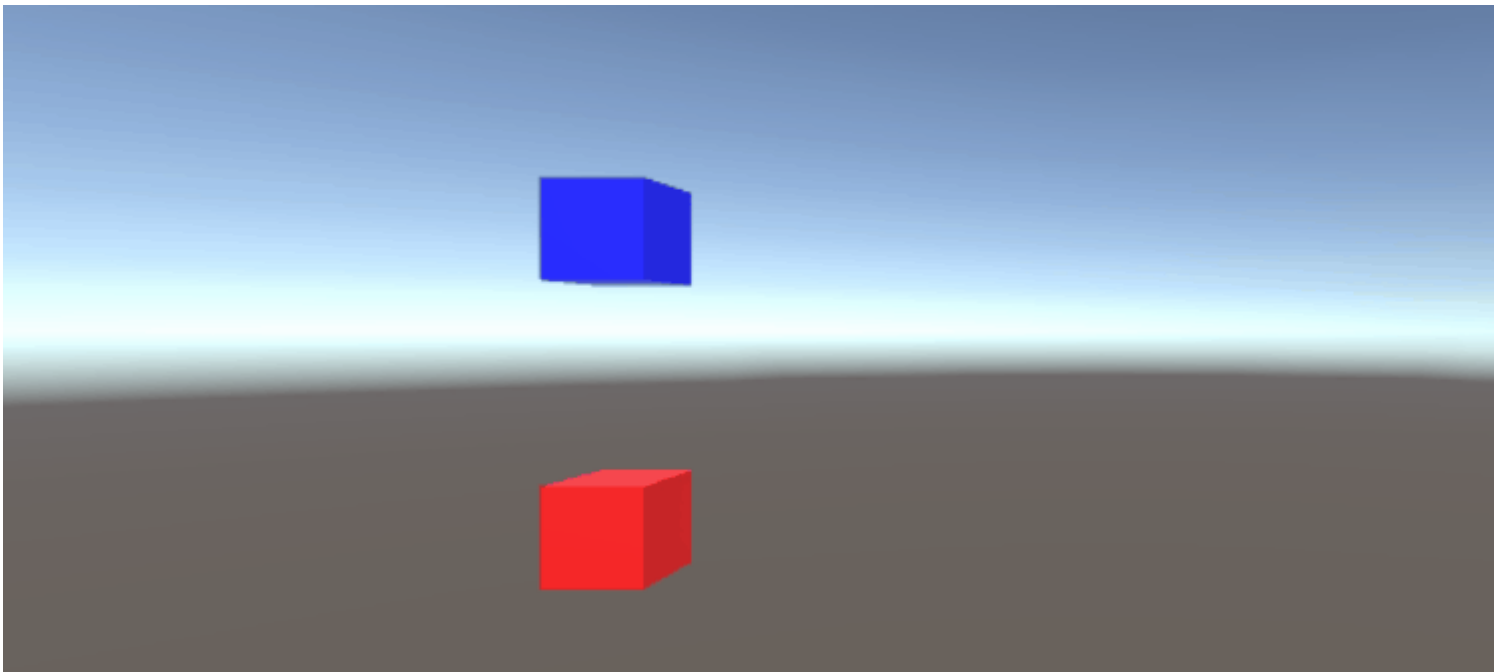
public class Lerp ping : MonoBehaviour
{
    /// <summary>The red box will use Lerp to move. We will link
    /// this object in via the inspector.</summary>
    public GameObject lerpObject;
    /// <summary>The starting position for our red box.</summary>
    public Vector3 lerpStart = new Vector3(0, 0, 0);
    /// <summary>The end position for our red box.</summary>
    public Vector3 lerpTarget = new Vector3(5, 0, 0);

    /// <summary>The blue box will use LerpUnclamped to move. We will
    /// link this object in via the inspector.</summary>
    public GameObject lerpUnclampedObject;
    /// <summary>The starting position for our blue box.</summary>
    public Vector3 lerpUnclampedStart = new Vector3(0, 3, 0);
    /// <summary>The end position for our blue box.</summary>
    public Vector3 lerpUnclampedTarget = new Vector3(5, 3, 0);

    /// <summary>The current fraction to increment our lerp functions by.</summary>
    public float lerpFraction = 0;

    private void Update()
    {
        // First, I increment the lerp fraction.
        // deltaTime * 0.25 should give me a value of +1 every second.
        lerpFraction += (Time.deltaTime * 0.25f);

        // Next, we apply the new lerp values to the target transform position.
        lerpObject.transform.position
            = Vector3.Lerp(lerpStart, lerpTarget, lerpFraction);
        lerpUnclampedObject.transform.position
            = Vector3.LerpUnclamped(lerpUnclampedStart, lerpUnclampedTarget, lerpFraction);
    }
}
```



MoveTowards

MoveTowardsはLerpによく似ています。コアの違いは、2つのポイントののではなく、するのをするということです。 MoveTowardsはターゲットVector3されないことにすることがVector3。

LerpUnclampedとに、ターゲットVector3からざかるようにのすることができます。このような、ターゲットVector3をしてしないので、きはです。このような、ターゲットVector3を「」としてすることができます。 Vector3がVector3にしてじをしているVector3、のきはのようにするはずはです。

のスク립トは、 MoveTowardsをして、スムーズなをしてオブジェクトのグループをのにします。

```
using UnityEngine;

public class MoveTowardsExample : MonoBehaviour
{
    /// <summary>The red cube will move up, the blue cube will move down,
    /// the green cube will move left and the yellow cube will move right.
    /// These objects will be linked via the inspector.</summary>
    public GameObject upCube, downCube, leftCube, rightCube;
    /// <summary>The cubes should move at 1 unit per second.</summary>
    float speed = 1f;

    void Update()
    {
        // We determine our distance by applying a deltaTime scale to our speed.
        float distance = speed * Time.deltaTime;

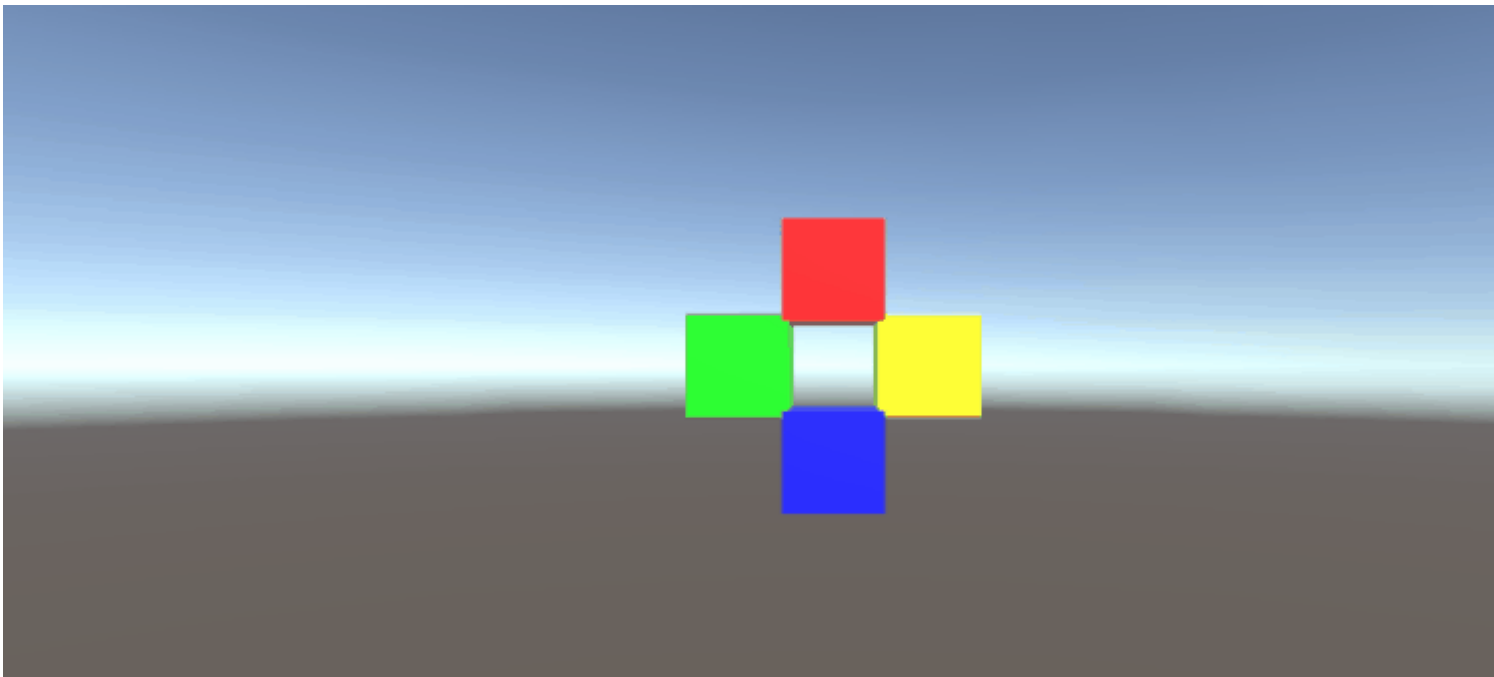
        // The up cube will move upwards, until it reaches the
        //position of (Vector3.up * 2), or (0, 2, 0).
        upCube.transform.position
            = Vector3.MoveTowards(upCube.transform.position, (Vector3.up * 2f), distance);

        // The down cube will move downwards, as it enforces a negative distance..
```

```
downCube.transform.position
    = Vector3.MoveTowards(downCube.transform.position, Vector3.up * 2f, -distance);

// The right cube will move to the right, indefinitely, as it is constantly updating
// its target position with a direction based off the current position.
rightCube.transform.position = Vector3.MoveTowards(rightCube.transform.position,
    rightCube.transform.position + Vector3.right, distance);

// The left cube does not need to account for updating its target position,
// as it is moving away from the target position, and will never reach it.
leftCube.transform.position
    = Vector3.MoveTowards(leftCube.transform.position, Vector3.right, -distance);
}
}
```



SmoothDamp

SmoothDamp はスムージングを MoveTowards だ MoveTowards とえてください。のによると、これは、スムーズなカメラフォローをうためにもにされます。

およびターゲットの Vector3 に Vector3 で、ベロシティをす Vector3 と、をするのになおよそのをす float もするがあります。のとはなり、々はにインクリメントされるべきとしてをする。これをメモすることがです。をしているのにのをすると、がましくないになるがあります。

なにえて、オブジェクトのをす float と、オブジェクトへのの SmoothDamp びしのギャップをす float をすることもできます。これらのをするはありません。デフォルトでははなく、は Time.deltaTime とされ Time.deltaTime 。さらになことに、 MonoBehaviour.Update () のでオブジェクトごとにをびす、のギャップをするはありません。

```
using UnityEngine;
```

```

public class SmoothDampMovement : MonoBehaviour
{
    /// <summary>The red cube will imitate the default SmoothDamp function.
    /// The blue cube will move faster by manipulating the "time gap", while
    /// the green cube will have an enforced maximum speed. Note that these
    /// objects have been linked via the inspector.</summary>
    public GameObject smoothObject, fastSmoothObject, cappedSmoothObject;

    /// <summary>We must instantiate the velocities, externally, so they may
    /// be manipulated from within the function. Note that by making these
    /// vectors public, they will be automatically instantiated as Vector3.Zero
    /// through the inspector. This also allows us to view the velocities,
    /// from the inspector, to observe how they change.</summary>
    public Vector3 regularVelocity, fastVelocity, cappedVelocity;

    /// <summary>Each object should move 10 units along the X-axis.</summary>
    Vector3 regularTarget = new Vector3(10f, 0f);
    Vector3 fastTarget = new Vector3(10f, 1.5f);
    Vector3 cappedTarget = new Vector3(10f, 3f);

    /// <summary>We will give a target time of 5 seconds.</summary>
    float targetTime = 5f;

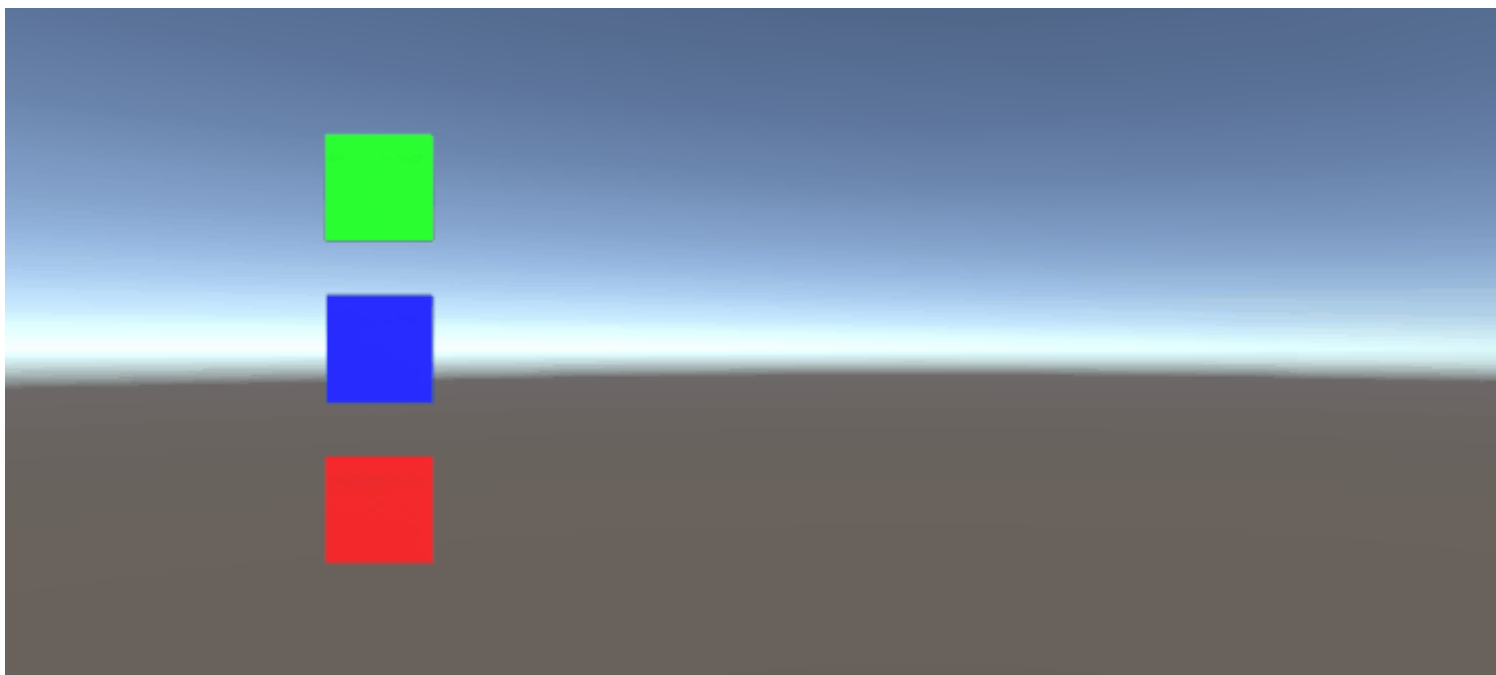
    void Update()
    {
        // The default SmoothDamp function will give us a general smooth movement.
        smoothObject.transform.position = Vector3.SmoothDamp(smoothObject.transform.position,
            regularTarget, ref regularVelocity, targetTime);

        // Note that a "maxSpeed" outside of reasonable limitations should not have any
        // effect, while providing a "deltaTime" of 0 tells the function that no time has
        // passed since the last SmoothDamp call, resulting in no movement, the second time.
        smoothObject.transform.position = Vector3.SmoothDamp(smoothObject.transform.position,
            regularTarget, ref regularVelocity, targetTime, 10f, 0f);

        // Note that "deltaTime" defaults to Time.deltaTime due to an assumption that this
        // function will be called once per update function. We can call the function
        // multiple times during an update function, but the function will assume that enough
        // time has passed to continue the same approximate movement. As a result,
        // this object should reach the target, quicker.
        fastSmoothObject.transform.position = Vector3.SmoothDamp(
            fastSmoothObject.transform.position, fastTarget, ref fastVelocity, targetTime);
        fastSmoothObject.transform.position = Vector3.SmoothDamp(
            fastSmoothObject.transform.position, fastTarget, ref fastVelocity, targetTime);

        // Lastly, note that a "maxSpeed" becomes irrelevant, if the object does not
        // realistically reach such speeds. Linear speed can be determined as
        // (Distance / Time), but given the simple fact that we start and end slow, we can
        // infer that speed will actually be higher, during the middle. As such, we can
        // infer that a value of (Distance / Time) or (10/5) will affect the
        // function. We will half the "maxSpeed", again, to make it more noticeable.
        cappedSmoothObject.transform.position = Vector3.SmoothDamp(
            cappedSmoothObject.transform.position,
            cappedTarget, ref cappedVelocity, targetTime, 1f);
    }
}

```

オンラインでVector3をむ <https://riptutorial.com/ja/unity3d/topic/7827/vector3>

10: アセットストア

Examples

ストアへのアクセス

Unity Asset Storeにアクセスするには3つのがあります

- Unityのメインメニューから「ウィンドウ」→「アセットストア」をしてAsset Storeウィンドウをきます。
- ショートカットキーをしますWindowsではCtrl + 9、Mac OSでは9
- Webインターフェイスをしてください <https://www.assetstore.unity3d.com/>

Unity Asset Storeにめてアクセスするは、のユーザアカウントをするか、サインインするようめられます。

の

アセットストアにアクセスし、ダウンロードしたいアセットをしたら、[ダウンロード]ボタンをクリックします。アセットにするコストがある、ボタンテキストは[すぐ]になることがあります。

The image shows a screenshot of the Unity Asset Store interface. On the left, there is a dark sidebar with the following text: 'Unity Ads', 'Services/In-Game Advertising', 'Unity Technologies', '★★★★ (1569)', 'Free', a blue 'Download' button with a heart icon, and social media icons for Twitter, Facebook, and Google+. Below these is a testimonial: 'Crossy Road earned over \$1 million in 45 days on iOS alone just with Unity Ads.' At the bottom of the sidebar, it says: 'Unity Ads is the top performing video ad network, used by King and other developers in hit games like Hill Climb Racing, Crossy Road, and many more.' The main area has a light blue background with the Unity logo and 'unity ADS' text. Below this, it says: 'Crossy Road earned over \$1 million on iOS in 45 days with delightful ads: watch a video, earn a virtual item'. An arrow points from this text to the 'CROSSY ROAD' logo and game characters (a chicken, a penguin, and a frog).

WebインターフェイスからUnity Asset Storeをしている、[ダウンロード]ボタンのテキストは[Unityでく]としてされます。このボタンをすると、Unityのインスタンスがし、アセットストアウィンドウにアセットがされます。

Unity Asset Storeからめてするは、のユーザーアカウントをするかサインインするようにめられます。

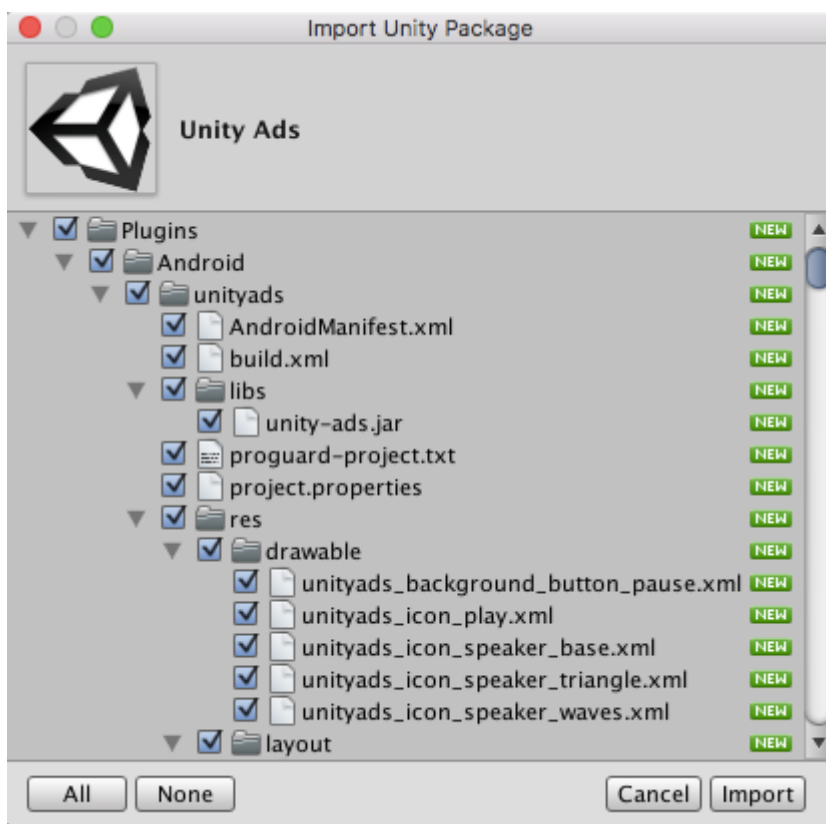
する、Unityはいをけることをします。

アセットのインポート

アセットがUnityでダウンロードされた、ダウンロードまたはボタンがインポートに変わります。

このオプションをすると、*Import Unity Package*ウィンドウがされます。ここで、ユーザーはプロジェクトでインポートするアセットファイルができます。

[インポート]をしてプロセスをし、したアセットファイルを[プロジェクトビュー]ウィンドウにされている[アセット]フォルダにします。



アセットの

1. サイトのアカウントをる
2. サイトアカウントにアセットをする
3. アセットストアからアセットストアツールをダウンロードし、
4. [アセットストアツール]> [パッケージのアップロード]にします。
5. ツールのウィンドウでしいパッケージとプロジェクトフォルダをしてください
6. クリックアップロード
7. アセットをオンラインでする

TODO - を、

1つののをする

は、サイトののをするためにされます。アセットまたはプラグインのくのは、サポートのにじてをします。は、またはプラグインをアクティブするためのライセンスキーとしてもされます。

は2かにあります。

1. アセットをすると、が「Unity Asset Store...」であるメールがされます。はこのメールのPDFファイルにされています。



UNITY3D.COM

Unity Technologies ApS

Vendersgade 28
1363 København K
Danmark

INVOICE

Invoice No.	[REDACTED]
Date	[REDACTED]
Due Date	[REDACTED]
Order No.	[REDACTED]

2. <https://www.assetstore.unity3d.com/#!/account/transactions>をくと、にがされます。

Credit Card / PayPal		
Date	Action	Description
[REDACTED]	CREDIT CARD / PAYPAL	# [REDACTED] 30 Mesh Terrain Editor Pro

オンラインでアセットストアをむ <https://riptutorial.com/ja/unity3d/topic/5705/アセットストア>

11: アセットパッケージの

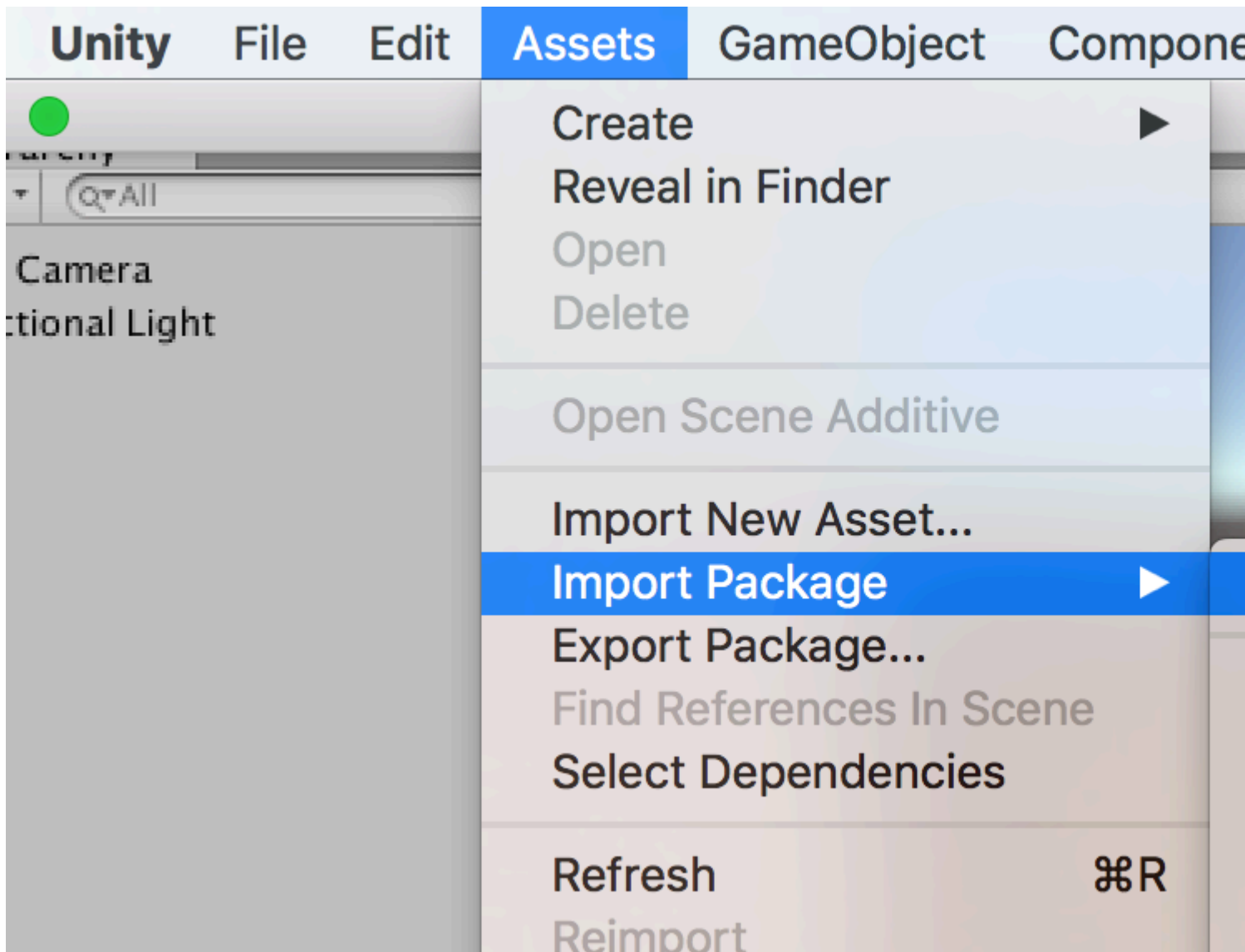
Examples

パッケージ

Asset Packages `.unitypackage`のファイルはUnityプロジェクトをのユーザにするためによくわれるです。のSDKをつ **Oculus**などをう、これらのパッケージの1つをダウンロードしてインポートするようにめられることがあります。

`.unity`パッケージのインポート

パッケージをインポートするには、Unityメニューバーで `Assets > Import Package > Custom Package...` をクリックし、されるファイルブラウザで `.unitypackage` ファイルにします。



オンラインでアセットパッケージのをむ <https://riptutorial.com/ja/unity3d/topic/4491/アセットパッ>

ページの

12: エディタの

- [MenuItemstring itemName]
- [MenuItemstring itemName、 bool isValidFunction]
- [MenuItemstring itemName、 bool isValidFunction、 int priority]
- [ContextMenu]
- [ContextMenuitem、]
- [DrawGizmo GizmoタイプGizmo]
- [DrawGizmoGizmoType Gizmo、 タイプdrawnGizmoType]

パラメーター

パラメータ	
MenuCommand	MenuCommandは、MenuItemのコンテキストをするためにされます。
MenuCommand.context	メニューコマンドのとなるオブジェクト
MenuCommand.userData	カスタムをメニューにすためのint

Examples

カスタムインスペクタ

カスタムインスペクタをすると、インスペクタでスクリプトをするをできます。によっては、カスタムプロパティドローではできないのをインスペクターでスクリプトにすることもできます。

に、カスタムインスペクタをしてよりなをできるカスタムオブジェクトのなをします。

```
using UnityEngine;
#if UNITY_EDITOR
using UnityEditor;
#endif

public class InspectorExample : MonoBehaviour {

    public int Level;
    public float BaseDamage;

    public float DamageBonus {
        get {
            return Level / 100f * 50;
        }
    }
}
```

```

    public float ActualDamage {
        get {
            return BaseDamage + DamageBonus;
        }
    }
}

#if UNITY_EDITOR
[CustomEditor( typeof( InspectorExample ) )]
public class CustomInspector : Editor {

    public override void OnInspectorGUI() {
        base.OnInspectorGUI();

        var ie = (InspectorExample)target;

        EditorGUILayout.LabelField( "Damage Bonus", ie.DamageBonus.ToString() );
        EditorGUILayout.LabelField( "Actual Damage", ie.ActualDamage.ToString() );
    }
}
#endif

```

まず、いくつかのフィールドでカスタムをします

```

public class InspectorExample : MonoBehaviour {
    public int Level;
    public float BaseDamage;
}

```

「インスペクタ」ウィンドウでスクリプトをしているときは、のフィールドがにカスタムインスペクタなしでされます。

```

public float DamageBonus {
    get {
        return Level / 100f * 50;
    }
}

public float ActualDamage {
    get {
        return BaseDamage + DamageBonus;
    }
}

```

これらのプロパティはUnityによってにされません。インスペクタビューでこれらのプロパティをするには、カスタムインスペクタをするがあります。

まず、このようなカスタムインスペクタをするがあります

```

[CustomEditor( typeof( InspectorExample ) )]
public class CustomInspector : Editor {

```

カスタムインスペクタはエディタからし、 *CustomEditor*をとします。のパラメータは、このカス

タムインスペクタをするオブジェクトのタイプです。

は、`OnInspectorGUI`メソッドです。このメソッドは、スクリプトがインスペクタウィンドウにされるたびにひされます。

```
public override void OnInspectorGUI() {
    base.OnInspectorGUI();
}
```

`base.OnInspectorGUI`をびして、Unityがスクリプトののフィールドをできるようにします。たちがこれをぶつもりならば、たちでもっとをしなければならないでしょう。

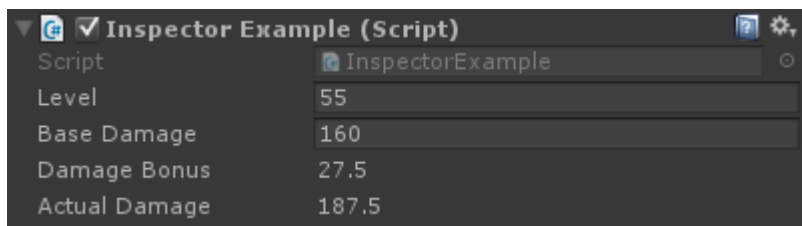
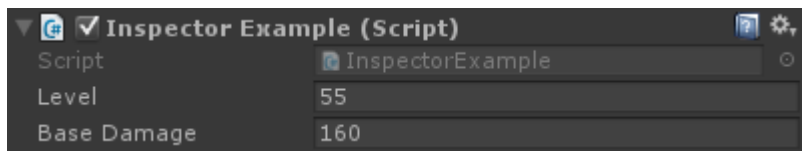
は、たちがせたいカスタムプロパティです

```
var ie = (InspectorExample)target;

EditorGUILayout.LabelField( "Damage Bonus", ie.DamageBonus.ToString() );
EditorGUILayout.LabelField( "Actual Damage", ie.ActualDamage.ToString() );
```

カスタムにキャストされたターゲットをするをするがありますターゲットはエディタからしたものです。

に、プロパティをするをできます。この、をしてすることができないので、2つのラベルがあればです。



カスタムプロパティドロー

によっては、データをむカスタムオブジェクトがありますが、`MonoBehaviour`からしません。これらのオブジェクトを`MonoBehaviour`クラスのフィールドとしてすると、オブジェクトののにのカスタムプロパティードロワーをしないうり、なはありません。

は、`MonoBehaviour`にされたカスタムオブジェクトのなと、カスタムオブジェクトのカスタムプロパティードロワーです。

```
public enum Gender {
    Male,
    Female,
    Other
}
```

```

}

// Needs the Serializable attribute otherwise the CustomPropertyDrawer wont be used
[Serializable]
public class UserInfo {
    public string Name;
    public int Age;
    public Gender Gender;
}

// The class that you can attach to a GameObject
public class PropertyDrawerExample : MonoBehaviour {
    public UserInfo UInfo;
}

[CustomPropertyDrawer( typeof( UserInfo ) )]
public class UserInfoDrawer : PropertyDrawer {

    public override float GetPropertyHeight( SerializedProperty property, GUIContent label ) {
        // The 6 comes from extra spacing between the fields (2px each)
        return EditorGUIUtility.singleLineHeight * 4 + 6;
    }

    public override void OnGUI( Rect position, SerializedProperty property, GUIContent label )
    {
        EditorGUI.BeginProperty( position, label, property );

        EditorGUI.LabelField( position, label );

        var nameRect = new Rect( position.x, position.y + 18, position.width, 16 );
        var ageRect = new Rect( position.x, position.y + 36, position.width, 16 );
        var genderRect = new Rect( position.x, position.y + 54, position.width, 16 );

        EditorGUI.indentLevel++;

        EditorGUI.PropertyField( nameRect, property.FindPropertyRelative( "Name" ) );
        EditorGUI.PropertyField( ageRect, property.FindPropertyRelative( "Age" ) );
        EditorGUI.PropertyField( genderRect, property.FindPropertyRelative( "Gender" ) );

        EditorGUI.indentLevel--;

        EditorGUI.EndProperty();
    }
}

```

まず、すべてのをつカスタムオブジェクトをします。ユーザーをするなクラスです。このクラスはPropertyDrawerExampleクラスでされ、GameObjectにできます。

```

public enum Gender {
    Male,
    Female,
    Other
}

[Serializable]
public class UserInfo {
    public string Name;
    public int Age;
    public Gender Gender;
}

```

```

}

public class PropertyDrawerExample : MonoBehaviour {
    public UserInfo UInfo;
}

```

カスタムクラスにはSerializableがです。そうでなければCustomPropertyDrawerはされません

はCustomPropertyDrawerです

まず、PropertyDrawerからしたクラスをすることがあります。クラスには、CustomPropertyDrawerもです。されるパラメーターは、このドロワーにするオブジェクトのタイプです。

```

[CustomPropertyDrawer( typeof( UserInfo ) )]
public class UserInfoDrawer : PropertyDrawer {

```

に、GetPropertyHeightをオーバーライドします。これにより、プロパティのカスタムをすることができます。この、たちのにはラベル、、、の4つがあることがわかります。このため、EditorGUIUtility.singleLineHeight * 4をして、フィールドに2ピクセルのをけたいので、の6ピクセルをします。

```

public override float GetPropertyHeight( SerializedProperty property, GUIContent label ) {
    return EditorGUIUtility.singleLineHeight * 4 + 6;
}

```

はこのOnGUIメソッドです。々は[..]EditorGUI.BeginPropertyでそれをめるとEditorGUI.EndPropertyでをします。このプロパティがプレハブのである、のプレハブオーバーライドロジックは、これらの2つのメソッドののすべてにしてするように、これをいます。

```

public override void OnGUI( Rect position, SerializedProperty property, GUIContent label ) {
    EditorGUI.BeginProperty( position, label, property );

```

その、フィールドのをむラベルをし、フィールドのをします。

```

EditorGUI.LabelField( position, label );

var nameRect = new Rect( position.x, position.y + 18, position.width, 16 );
var ageRect = new Rect( position.x, position.y + 36, position.width, 16 );
var genderRect = new Rect( position.x, position.y + 54, position.width, 16 );

```

すべてのフィールドは16 + 2ピクセルのをち、さは16ですこれはEditorGUIUtility.singleLineHeightとじです

に、1つのタブでUIをインデントし、しいレイアウトをし、プロパティをし、GUIをインデントしないで、EditorGUI.EndPropertyでします。

```

EditorGUI.indentLevel++;

```

```

EditorGUI.PropertyField( nameRect, property.FindPropertyRelative( "Name" ) );
EditorGUI.PropertyField( ageRect, property.FindPropertyRelative( "Age" ) );
EditorGUI.PropertyField( genderRect, property.FindPropertyRelative( "Gender" ) );

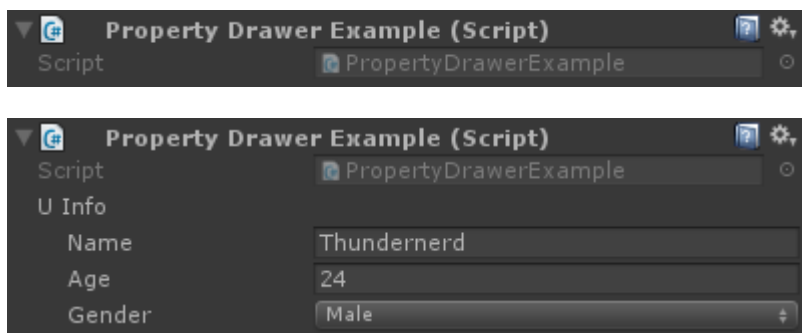
EditorGUI.indentLevel--;

EditorGUI.EndProperty();

```

フィールドのには、のをとる `EditorGUI.PropertyField` と、プロパティをする `SerializedProperty` をします。 `OnGUI` でされたプロパティで `FindPropertyRelative` "... " をびすことによってプロパティをします。これらはとをし、のプロパティが見つからないことにしてください。

このでは、 `property.FindPropertyRelative` "... " からのりをしていません。なびしをぐには、これらのクラスをプライベートフィールドにするがあります



メニュー

メニューは、カスタムアクションをエディタにするれたです。メニューバーにメニューをしたり、のコンポーネントのコンテキストをクリックしたり、スクリプトのフィールドをコンテキストをクリックしたりすることもできます。

は、メニューをするのです。

```

public class MenuItemsExample : MonoBehaviour {

    [MenuItem( "Example/DoSomething %#&d" )]
    private static void DoSomething() {
        // Execute some code
    }

    [MenuItem( "Example/DoAnotherThing", true )]
    private static bool DoAnotherThingValidator() {
        return Selection.gameObjects.Length > 0;
    }

    [MenuItem( "Example/DoAnotherThing _PGUP", false )]
    private static void DoAnotherThing() {
        // Execute some code
    }

    [MenuItem( "Example/DoOne %a", false, 1 )]
    private static void DoOne() {
        // Execute some code
    }
}

```

```

[MenuItem( "Example/DoTwo #b", false, 2 )]
private static void DoTwo() {
    // Execute some code
}

[MenuItem( "Example/DoFurther &c", false, 13 )]
private static void DoFurther() {
    // Execute some code
}

[MenuItem( "CONTEXT/Camera/DoCameraThing" )]
private static void DoCameraThing( MenuCommand cmd ) {
    // Execute some code
}

[ContextMenu( "ContextSomething" )]
private void ContentSomething() {
    // Execute some code
}

[ContextMenu( "Reset", "ResetDate" )]
[ContextMenu( "Set to Now", "SetDateToNow" )]
public string Date = "";

public void ResetDate() {
    Date = "";
}

public void SetDateToNow() {
    Date = DateTime.Now.ToString();
}
}

```

それはこのようにえる

Example	Window	Help
DoOne		Ctrl+A
DoTwo		Shift+B
DoFurther		Alt+C
DoSomething	Ctrl+Shift+Alt+D	
DoAnotherThing		PgUp

なメニューをてみましょう。にすように、メニューのタイトルとしてをす *MenuItem* をつをするがあります。に/をすることで、レベルのメニューをくすることができます。

```

[MenuItem( "Example/DoSomething %#&d" )]
private static void DoSomething() {
    // Execute some code
}

```

トップレベルにメニューアイテムをくことはできません。あなたのメニューはサブメニューにあるがあります

MenuItemののにあるはショートカットキーのためのものですが、これらはではありません。

ショートカットキーにできるはのとおりです。

- - WindowsではCtrl、OS XではCmd
- - シフト
- - Alt

つまり、ショートカットdは、Windowsではctrl + shift + alt + Dをし、OS Xではcmd + shift + alt + Dをします。

なキーなしでショートカットをしたいは、例えば 'D'キーだけをするは、したいショートカットキーのに_アンダースコアをけることができます。

サポートされているのいくつかのなキーがあります

- 、 、 、 - キー
- F1 ... F12 - ファンクションキー
- HOME、END、PGUP、PGDN - ナビゲーションキー

ショートカットキーは、のテキストとスペースでるがあります

に、バリデータメニューです。バリデータメニューでは、がたされていないときにメニューをにすることができますグレー、クリック。あなたのメニューはのGameObjectsのにし、Validatorメニューでできます。

```
[MenuItem( "Example/DoAnotherThing", true )]
private static bool DoAnotherThingValidator() {
    return Selection.gameObjects.Length > 0;
}

[MenuItem( "Example/DoAnotherThing _PGUP", false )]
private static void DoAnotherThing() {
    // Execute some code
}
```

バリデータメニューをさせるには、MenuItemとしショートカットキーはありませんので、2つのをするがあります。それらのいは、ブールパラメータをすことによって、バリデータとしてマークすることです。

また、をしてメニューのをすることもできます。は、3のパラメータとしてすによってされます。リストのがさいほど、リストのがきくなります。2つのメニューのにセパレータをするには、メニューののに10あることをします。

```
[MenuItem( "Example/DoOne %a", false, 1 )]
private static void DoOne() {
    // Execute some code
}

[MenuItem( "Example/DoTwo #b", false, 2 )]
```

```
private static void DoTwo() {
    // Execute some code
}

[MenuItem( "Example/DoFurther &c", false, 13 )]
private static void DoFurther() {
    // Execute some code
}
```

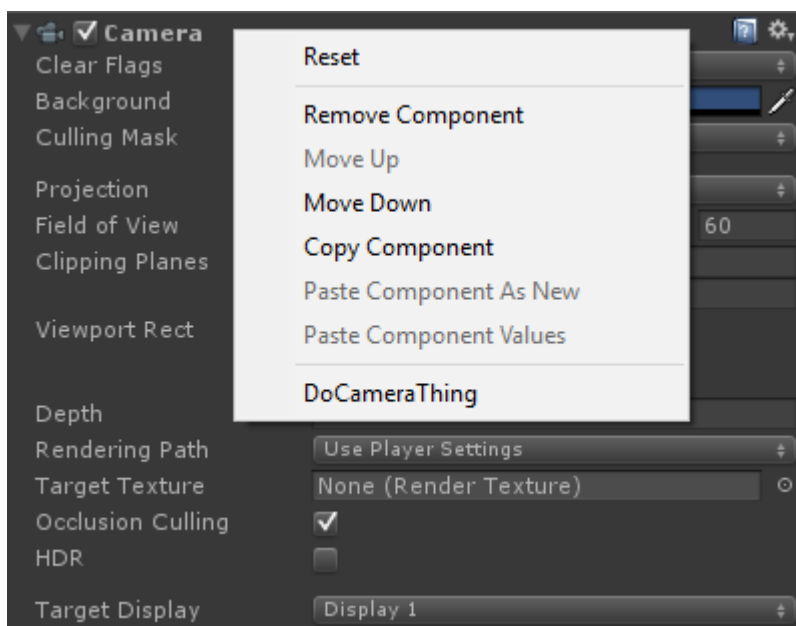
けされたとのないがみわされたメニューリストがある、のないはとりされます。

に、のコンポーネントのコンテキストメニューにメニューをします。CONTEXTをすることでMenuItemのをし、にMenuCommandパラメータをりませるがあります。

のスニペットは、カメラコンポーネントにコンテキストメニューをします。

```
[MenuItem( "CONTEXT/Camera/DoCameraThing" )]
private static void DoCameraThing( MenuCommand cmd ) {
    // Execute some code
}
```

それはこのようにえる

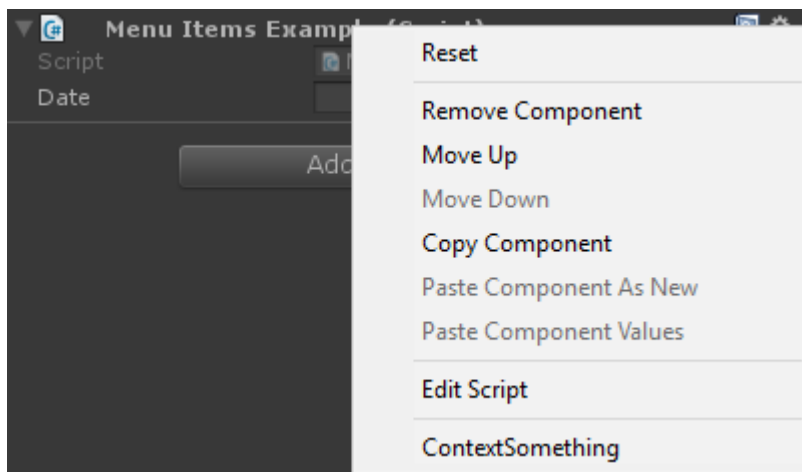


MenuCommandパラメータをすると、コンポーネントのと、それとともにされるすべてのユーザーデータにアクセスできます。

また、ContextMenuをして、のコンポーネントにコンテキストメニューをすることもできます。これは、またはやがなく、メソッドのでなければなりません。

```
[ContextMenu( "ContextSomething" )]
private void ContentSomething() {
    // Execute some code
}
```

それはこのようにえる



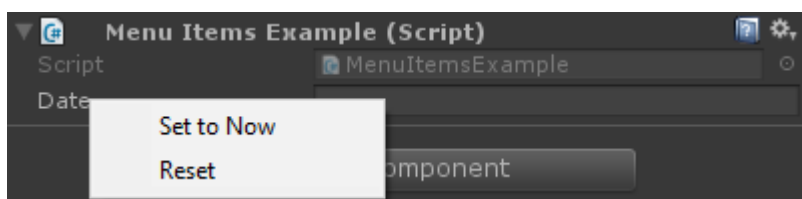
コンテキストメニューは、のコンポーネントのフィールドにすることもできます。これらのメニューは、それらがするをコンテキストクリックするとされ、そのでしたメソッドをできます。このようにして、にすように、えばデフォルトまたはのをすることができます。

```
[ContextMenuItem( "Reset", "ResetDate" )]
[ContextMenuItem( "Set to Now", "SetDateToNow" )]
public string Date = "";

public void ResetDate() {
    Date = "";
}

public void SetDateToNow() {
    Date = DateTime.Now.ToString();
}
```

それはこのようにえる



Gizmo

Gizmoは、シーンビューでをするためにされます。これらのをして、ゲームオブジェクトにするのえば、らがつまたはをくことができます。

は、これをうの2つのです

1

このでは、 *OnDrawGizmos*メソッドと *OnDrawGizmosSelected* magicメソッドをしています。


```

public class GizmoExample : MonoBehaviour {

    public float GetDetectionRadius() {
        return 12.5f;
    }

    public float GetFOV() {
        return 25f;
    }

    public float GetMaxRange() {
        return 6.5f;
    }

    public float GetMinRange() {
        return 0;
    }

    public float GetAspect() {
        return 2.5f;
    }

    public void OnDrawGizmos() {
        var gizmoMatrix = Gizmos.matrix;
        var gizmoColor = Gizmos.color;

        Gizmos.matrix = Matrix4x4.TRS( transform.position, transform.rotation,
transform.lossyScale );
        Gizmos.color = Color.red;
        Gizmos.DrawFrustum( Vector3.zero, GetFOV(), GetMaxRange(), GetMinRange(), GetAspect()
);

        Gizmos.matrix = gizmoMatrix;
        Gizmos.color = gizmoColor;
    }

    public void OnDrawGizmosSelected() {
        Handles.DrawWireDisc( transform.position, Vector3.up, GetDetectionRadius() );
    }
}

```

ここでは、オブジェクトがアクティブなときにく OnDrawGizmos と、オブジェクトがでされているとき OnDrawGizmosSelected の2つのをして Gizmo をします。

```

public void OnDrawGizmos() {
    var gizmoMatrix = Gizmos.matrix;
    var gizmoColor = Gizmos.color;

    Gizmos.matrix = Matrix4x4.TRS( transform.position, transform.rotation,
transform.lossyScale );
    Gizmos.color = Color.red;
    Gizmos.DrawFrustum( Vector3.zero, GetFOV(), GetMaxRange(), GetMinRange(), GetAspect() );

    Gizmos.matrix = gizmoMatrix;
    Gizmos.color = gizmoColor;
}

```

に、Gizmo とをします。これはするであり、の Gizmo にをえないようにするためににしたいから

です。

にオブジェクトにあるをきたいが、Gizmosのを、スケールにわせてするがある。々はまた、Gizmosのをにして、をしました。これがすると、`Gizmos.DrawFrustum`をびしてシーンビューにをすることができます。

したいものしたら、ギズモのをリセットします。

```
public void OnDrawGizmosSelected() {
    Handles.DrawWireDisc( transform.position, Vector3.up, GetDetectionRadius() );
}
```

また、GameObjectをするときをきたい。Gizmosクラスにはディスクのメソッドがないため、これはHandlesクラスをしてわれます。

こののギズモをすると、のようなられます。

2

このでは、`DrawGizmo`をしています。

```
public class GizmoDrawerExample {

    [DrawGizmo( GizmoType.Selected | GizmoType.NonSelected, typeof( GizmoExample ) )]
    public static void DrawGizmo( GizmoExample obj, GizmoType type ) {
        var gizmoMatrix = Gizmos.matrix;
        var gizmoColor = Gizmos.color;

        Gizmos.matrix = Matrix4x4.TRS( obj.transform.position, obj.transform.rotation,
obj.transform.lossyScale );
        Gizmos.color = Color.red;
        Gizmos.DrawFrustum( Vector3.zero, obj.GetFOV(), obj.GetMaxRange(), obj.GetMinRange(),
obj.GetAspect() );

        Gizmos.matrix = gizmoMatrix;
        Gizmos.color = gizmoColor;

        if ( ( type & GizmoType.Selected ) == GizmoType.Selected ) {
            Handles.DrawWireDisc( obj.transform.position, Vector3.up, obj.GetDetectionRadius()
);
        }
    }
}
```

こので、スクリプトからギズモびしをすることができます。これは、2つのことをいて、のとじコードをします。

```
[DrawGizmo( GizmoType.Selected | GizmoType.NonSelected, typeof( GizmoExample ) )]
public static void DrawGizmo( GizmoExample obj, GizmoType type ) {
```

のパラメーターとしてEnum GizmoTypeをとり、2のパラメーターとしてTypeをとるDrawGizmo

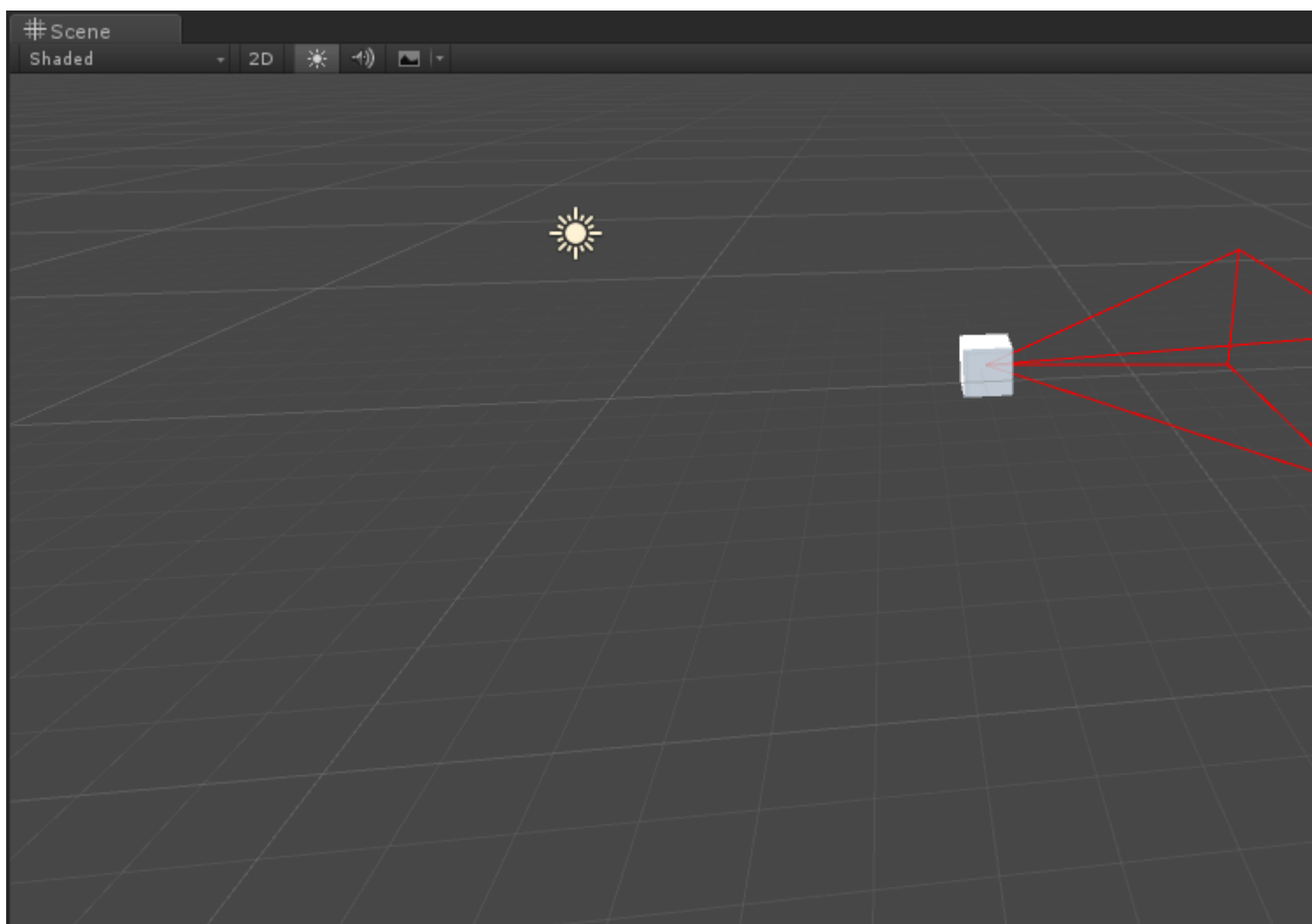
をするがあります。タイプは、ギズモをするためにするタイプでなければなりません。

ギズモをするは、パブリックまたはパブリックであるがあり、のをけることができます。のパラメータは、の2のパラメータとしてされるとするがあります。2のパラメータは、オブジェクトののをすenum GizmoTypeです。

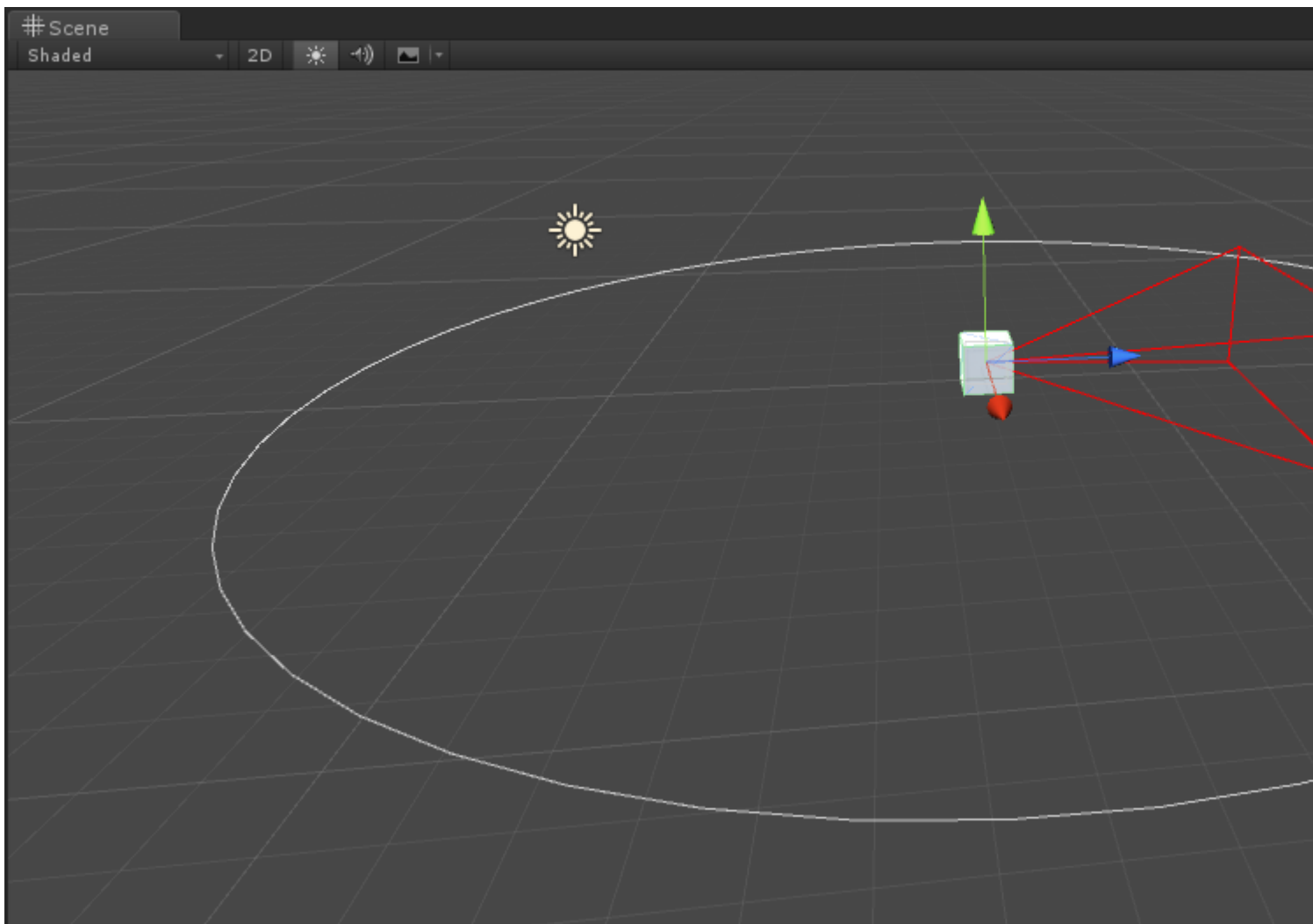
```
if ( ( type & GizmoType.Selected ) == GizmoType.Selected ) {  
    Handles.DrawWireDisc( obj.transform.position, Vector3.up, obj.GetDetectionRadius() );  
}
```

もう1つのいは、オブジェクトのGizmoTypeがであるかをべるために、なパラメータとタイプをANDでチェックするがあることです。

されていません



された



エディタウィンドウ

なぜエディタウィンドウ

このように、カスタムインスペクタでこのことをうことができますカスタムインスペクタのかわからないは、こちらのを[してください](http://www.riptutorial.com/unity3d/topic/2506)。http://www.riptutorial.com/unity3d/topic/2506エディタをすることができますが、あるでは、パネルやカスタマイズされたアセットパレットをすることもできます。その、[EditorWindow](#)をします。Unity UIはエディタウィンドウでされ、はトッパバーをして、タブなど

なEditorWindowをする

な

カスタムエディタウィンドウのはかなりです。EditorWindowクラスをし、InitメソッドとOnGUIメソッドをすだけです。ここにながあります

```
using UnityEngine;  
using UnityEditor;
```

```

public class CustomWindow : EditorWindow
{
    // Add menu named "Custom Window" to the Window menu
    [MenuItem("Window/Custom Window")]
    static void Init()
    {
        // Get existing open window or if none, make a new one:
        CustomWindow window = (CustomWindow) EditorWindow.GetWindow(typeof(CustomWindow));
        window.Show();
    }

    void OnGUI()
    {
        GUILayout.Label("This is a custom Editor Window", EditorStyles.boldLabel);
    }
}

```

3つのポイントはのとおりです。

1. EditorWindowをすることをわれないでください
2. このにすようにInitをしてください。 [EditorWindow.GetWindow](#)は、CustomWindowがすでにされているかどうかをしています。そうでないは、しいインスタンスがされます。これをすると、にのウィンドウインスタンスをたないようにすることができます
3. いつものようにOnGUIをとってウィンドウにをする

なはのようになります。



CustomWindow
This is a custom Editor Window

よりくむ

もちろん、このEditorWindowをしていくつかのアセットをまたはしたいとうでしょう。に、[Selection](#)クラスをしてアクティブなSelectionをする、されたアセットプロパティを

SerializedObjectおよびSerializedPropertyでするをします。

```
using System.Linq;
using UnityEngine;
using UnityEditor;

public class CustomWindow : EditorWindow
{
    private AnimationClip _animationClip;
    private SerializedObject _serializedClip;
    private SerializedProperty _events;

    private string _text = "Hello World";

    // Add menu named "Custom Window" to the Window menu
    [MenuItem("Window/Custom Window")]
    static void Init()
    {
        // Get existing open window or if none, make a new one:
        CustomWindow window = (CustomWindow) EditorWindow.GetWindow(typeof(CustomWindow));
        window.Show();
    }

    void OnGUI()
    {
        GUILayout.Label("This is a custom Editor Window", EditorStyles.boldLabel);

        // You can use EditorGUI, EditorGUILayout and GUILayout classes to display
        anything you want
        // A TextField example
        _text = EditorGUILayout.TextField("Text Field", _text);

        // Note that you can modify an asset or a gameobject using an EditorWindow. Here
        is a quick example with an AnimationClip asset
        // The _animationClip, _serializedClip and _events are set in OnSelectionChange()

        if (_animationClip == null || _serializedClip == null || _events == null) return;

        // We can modify our serializedClip like we would do in a Custom Inspector. For
        example we can grab its events and display their information

        GUILayout.Label(_animationClip.name, EditorStyles.boldLabel);

        for (var i = 0; i < _events.arraySize; i++)
        {
            EditorGUILayout.BeginVertical();

            EditorGUILayout.LabelField(
                "Event : " +
                _events.GetArrayElementAtIndex(i).FindPropertyRelative("functionName").stringValue,
                EditorStyles.boldLabel);

            EditorGUILayout.PropertyField(_events.GetArrayElementAtIndex(i).FindPropertyRelative("time"),
                true,
                GUILayout.ExpandWidth(true));

            EditorGUILayout.PropertyField(_events.GetArrayElementAtIndex(i).FindPropertyRelative("functionName"),
                true, GUILayout.ExpandWidth(true));

            EditorGUILayout.PropertyField(_events.GetArrayElementAtIndex(i).FindPropertyRelative("floatParameter"),
```

```

        true, GUILayout.ExpandWidth(true));
EditorGUILayout.PropertyField(_events.GetArrayElementAtIndex(i).FindPropertyRelative("intParameter"),
        true, GUILayout.ExpandWidth(true));
EditorGUILayout.PropertyField(
_events.GetArrayElementAtIndex(i).FindPropertyRelative("objectReferenceParameter"), true,
        GUILayout.ExpandWidth(true));

EditorGUILayout.Separator();
EditorGUILayout.EndVertical();
}

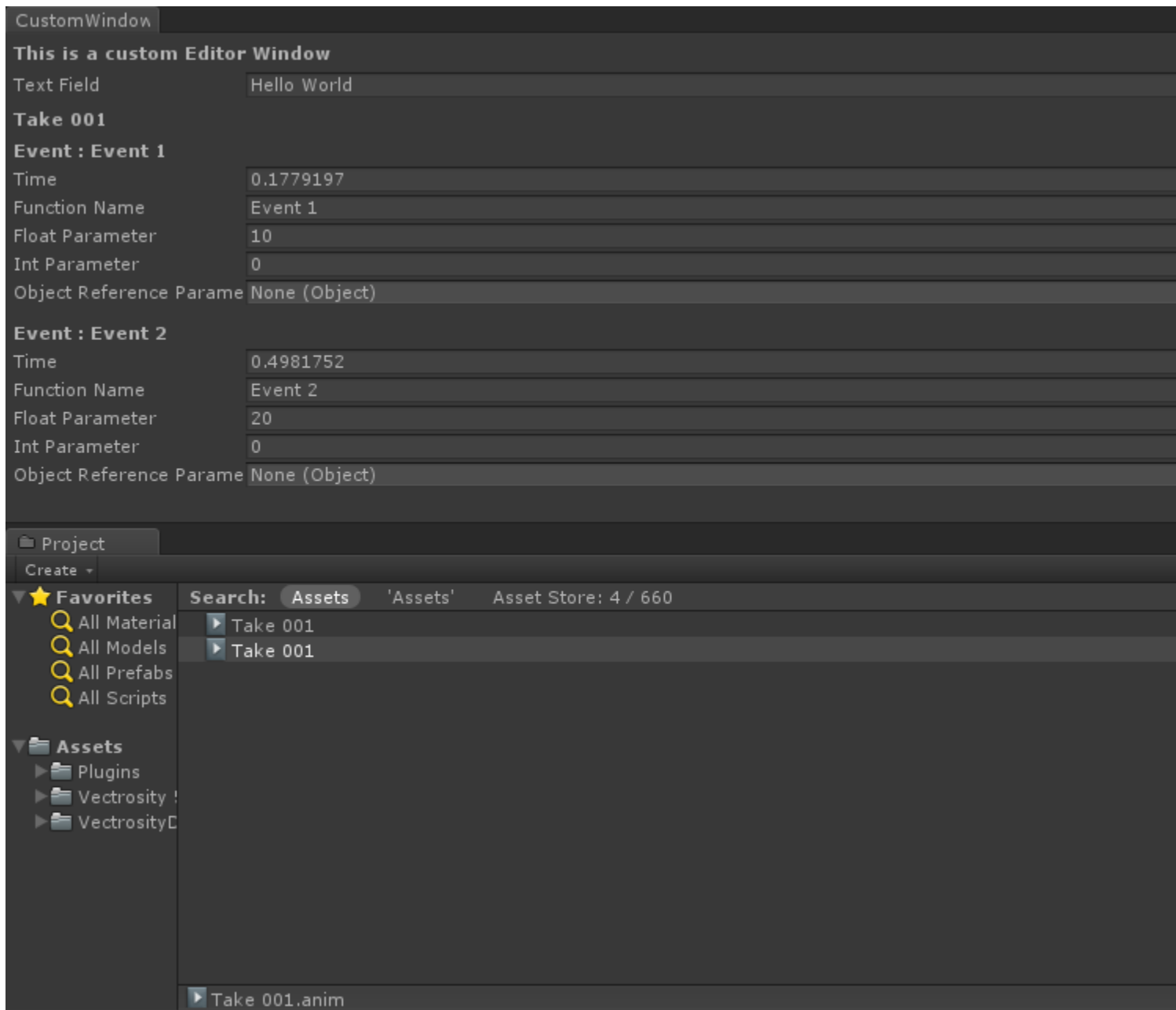
// Of course we need to Apply the modified properties. We don't our changes won't
be saved
_serializedClip.ApplyModifiedProperties();
}

/// This Message is triggered when the user selection in the editor changes. That's
when we should tell our Window to Repaint() if the user selected another AnimationClip
private void OnSelectionChange()
{
    _animationClip =
        Selection.GetFiltered(typeof(AnimationClip),
SelectionMode.Assets).FirstOrDefault() as AnimationClip;
    if (_animationClip == null) return;

    _serializedClip = new SerializedObject(_animationClip);
    _events = _serializedClip.FindProperty("m_Events");
    Repaint();
}
}

```

はのとおりで。



なトピック

あなたはエディタでいくつかのなにをすることができ、EditorWindowクラスはのをするのにです。Unity Asset StoreNodeCanvasやPlayMakerなどのほとんどのなアセットは、カスタムビューをするためにEditorWindowをします。

SceneViewでの

EditorWindowでういことの1つは、SceneViewにをすることです。これで、にカスタマイズされたマップ/ワールドエディタをできます。たとえば、カスタムEditorWindowをアセットパレットとしてし、SceneViewのクリックをリスンしてしいオブジェクトをインスタンスします。にをします。

```
using UnityEngine;
```



```

using System;
using UnityEditor;

public class CustomWindow : EditorWindow {

    private enum Mode {
        View = 0,
        Paint = 1,
        Erase = 2
    }

    private Mode CurrentMode = Mode.View;

    [MenuItem ("Window/Custom Window")]
    static void Init () {
        // Get existing open window or if none, make a new one:
        CustomWindow window = (CustomWindow)EditorWindow.GetWindow (typeof (CustomWindow));
        window.Show();
    }

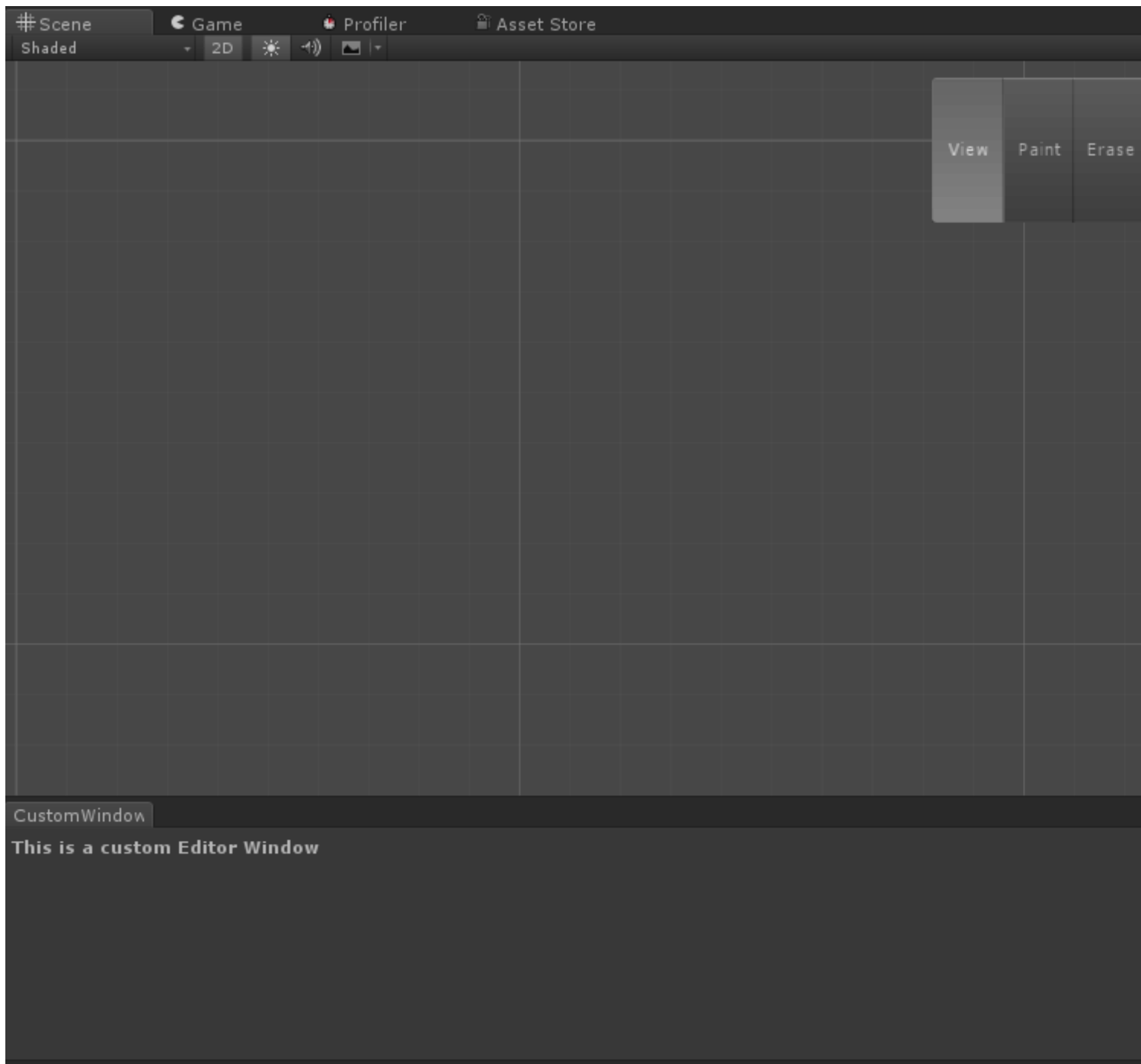
    void OnGUI () {
        GUILayout.Label ("This is a custom Editor Window", EditorStyles.boldLabel);
    }

    void OnEnable() {
        SceneView.onSceneGUIDelegate = SceneViewGUI;
        if (SceneView.lastActiveSceneView) SceneView.lastActiveSceneView.Repaint();
    }

    void SceneViewGUI(SceneView sceneView) {
        Handles.BeginGUI();
        // We define the toolbars' rects here
        var ToolBarRect = new Rect((SceneView.lastActiveSceneView.camera.pixelRect.width / 6),
10, (SceneView.lastActiveSceneView.camera.pixelRect.width * 4 / 6) ,
SceneView.lastActiveSceneView.camera.pixelRect.height / 5);
        GUILayout.BeginArea(ToolBarRect);
        GUILayout.BeginHorizontal();
        GUILayout.FlexibleSpace();
        CurrentMode = (Mode) GUILayout.Toolbar(
            (int) CurrentMode,
            Enum.GetNames (typeof (Mode)),
            GUILayout.Height (ToolBarRect.height));
        GUILayout.FlexibleSpace();
        GUILayout.EndHorizontal();
        GUILayout.EndArea();
        Handles.EndGUI();
    }
}

```

これにより、ツールバーがSceneViewにされます



あなたがどこまでくことができるかをにてみましょう

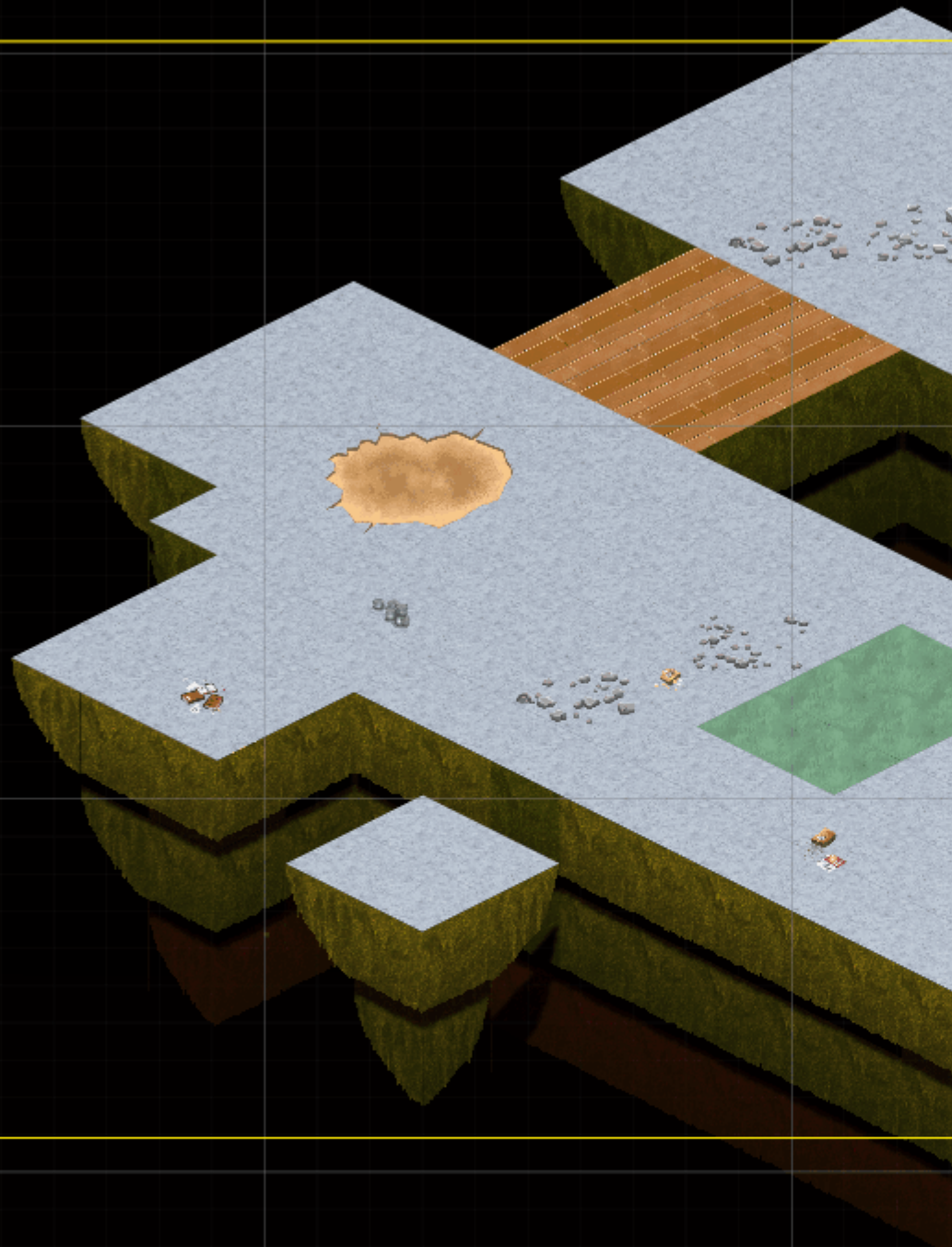
Position sceneView camera

- Camera Lock
- Draw Walkable Gizmo
- Draw Cover Gizmo
- Hide Map Hierarchy
- Show grid
- Draw GridCell Neighbors

Dimensions:

+ - + -

+ - + -

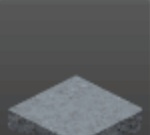





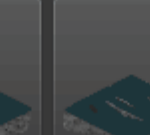

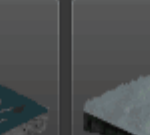



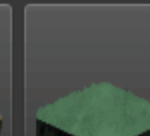



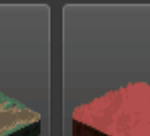



Map Editor Project Console Pro 3

Palette

Search Term :

[Grid]

 Aeroport_01	 Aeroport_02	 Aeroport_03	 Aeroport_04	 Aeroport_05	 petAeroport	 petAeroport	 petAeroport	 arpetBlue_
								

13: オーディオシステム

き

これは、Unity3Dでオーディオをすることにするドキュメントです。

Examples

オーディオクラス - オーディオをする

```
using UnityEngine;

public class Audio : MonoBehaviour {
    AudioSource audioSource;
    AudioClip audioClip;

    void Start() {
        audioClip = (AudioClip)Resources.Load("Audio/Soundtrack");
        audioSource.clip = audioClip;
        if (!audioSource.isPlaying) audioSource.Play();
    }
}
```

オンラインでオーディオシステムをむ <https://riptutorial.com/ja/unity3d/topic/8064/オーディオシステム>

14: オブジェクトプーリング

Examples

オブジェクトプール

ゲームをするときに、じタイプのオブジェクトをりししてするがあります。プレハブをし、にじてこれをインスタンス/することでにうことができますが、これをうとがくなり、ゲームがくなるがあります。

このをする1つのは、オブジェクトプーリングです。には、なインスタンスやをぐためにいつでもするオブジェクトのプールにがあるかどうかにかかわらずがあることをします。

は、なオブジェクトプールのです

```
public class ObjectPool : MonoBehaviour
{
    public GameObject prefab;
    public int amount = 0;
    public bool populateOnStart = true;
    public bool growOverAmount = true;

    private List<GameObject> pool = new List<GameObject>();

    void Start()
    {
        if (populateOnStart && prefab != null && amount > 0)
        {
            for (int i = 0; i < amount; i++)
            {
                var instance = Instantiate(Prefab);
                instance.SetActive(false);
                pool.Add(instance);
            }
        }
    }

    public GameObject Instantiate (Vector3 position, Quaternion rotation)
    {
        foreach (var item in pool)
        {
            if (!item.activeInHierarchy)
            {
                item.transform.position = position;
                item.transform.rotation = rotation;
                item.SetActive( true );
                return item;
            }
        }

        if (growOverAmount)
        {
            var instance = (GameObject)Instantiate(prefab, position, rotation);
            pool.Add(instance);
        }
    }
}
```

```

        return instance;
    }

    return null;
}
}

```

にをべましょう

```

public GameObject prefab;
public int amount = 0;
public bool populateOnStart = true;
public bool growOverAmount = true;

private List<GameObject> pool = new List<GameObject>();

```

- `GameObject prefab` これは、オブジェクトプールがしいオブジェクトをプールにインスタンスするためにするプレハブです。
- `int amount` これは、プールにれることができるアイテムのです。のアイテムをインスタンスしたいが、そのプールがににしている、プールののアイテムがされます。
- `bool populateOnStart` にプールをするかどうかをできます。そうすることで、プレハブのインスタンスでプールがいっぱいになるので、めて `Instantiate` をびすときには、のオブジェクト
- `bool growOverAmount` これを `true` にすると、のののクエストがあったにプールがします。プールにれるアイテムのをににできるわけではありませんので、にじてプールにします。
- `List<GameObject> pool` これはプールで、インスタンスされた/されたすべてのオブジェクトがされるです。

では、`Start` をてみましょう

```

void Start()
{
    if (populateOnStart && prefab != null && amount > 0)
    {
        for (int i = 0; i < amount; i++)
        {
            var instance = Instantiate(Prefab);
            instance.SetActive(false);
            pool.Add(instance);
        }
    }
}

```

では、にリストにをするかどうかをチェックし、`prefab` がされ、が0よりきいはそうでなければにする

これは、しいオブジェクトをインスタンスし、それらをプールにれるなループです。すべきの1つは、すべてのインスタンスをアクティブにすることです。このでは、まだゲームではされません

に、`Instantiate` があります。これは、ほとんどののがこるです

```

public GameObject Instantiate (Vector3 position, Quaternion rotation)
{
    foreach (var item in pool)
    {
        if (!item.activeInHierarchy)
        {
            item.transform.position = position;
            item.transform.rotation = rotation;
            item.SetActive(true);
            return item;
        }
    }

    if (growOverAmount)
    {
        var instance = (GameObject)Instantiate(prefab, position, rotation);
        pool.Add(instance);
        return instance;
    }

    return null;
}

```

`Instantiate`はUnityの`Instantiate`のようにえますが、プレハブはすでにクラスメンバーとしてされています。

`Instantiate`のステップは、プールにアクティブなオブジェクトがあるかどうかをすることです。つまり、そのオブジェクトをしてリクエストにすることができます。プールにアクティブなオブジェクトがあるは、とをし、アクティブにしますそうしないと、アクティブにするのをれたはにされるがあります。

2のステップは、プールにアクティブがなく、プールがのをえてするにのみします。がこるかはですプレハブののインスタンスがされ、プールにされます。プールのをすると、プールになのオブジェクトをできます。

3の「ステップ」は、プールにアクティブがなく、プールができないにのみします。これがきると、リクエストはの`GameObject`をけります。これはもできなかったことをし、`NullReferenceExceptions`をぐためににするがあります。

あなたのアイテムがプールにるようにするには、ゲームオブジェクトをしてはいけません。あなたがするがあるのは、それらをアクティブにすることだけで、プールをしてできるようになります。

オブジェクトプール

は、のオブジェクトタイプのレンタルとをにするオブジェクトプールのです。オブジェクトプールをするには、`create`の`Func`とオブジェクトをする`Action`がです。プールがのときにオブジェクトをすると、しいオブジェクトがされ、プールにオブジェクトがあるときにオブジェクトがプールからされてされます。

オブジェクトプール

```

public class ResourcePool<T> where T : class
{
    private readonly List<T> objectPool = new List<T>();
    private readonly Action<T> cleanUpAction;
    private readonly Func<T> createAction;

    public ResourcePool(Action<T> cleanUpAction, Func<T> createAction)
    {
        this.cleanUpAction = cleanUpAction;
        this.createAction = createAction;
    }

    public void Return(T resource)
    {
        this.objectPool.Add(resource);
    }

    private void PurgeSingleResource()
    {
        var resource = this.Rent();
        this.cleanUpAction(resource);
    }

    public void TrimResourcesBy(int count)
    {
        count = Math.Min(count, this.objectPool.Count);
        for (int i = 0; i < count; i++)
        {
            this.PurgeSingleResource();
        }
    }

    public T Rent()
    {
        int count = this.objectPool.Count;
        if (count == 0)
        {
            Debug.Log("Creating new object.");
            return this.createAction();
        }
        else
        {
            Debug.Log("Retrieving existing object.");
            T resource = this.objectPool[count-1];
            this.objectPool.RemoveAt(count-1);
            return resource;
        }
    }
}

```

```

public class Test : MonoBehaviour
{
    private ResourcePool<GameObject> objectPool;

    [SerializeField]
    private GameObject enemyPrefab;

    void Start()
    {
        this.objectPool = new ResourcePool<GameObject>(Destroy, () =>

```



```

Instantiate(this.enemyPrefab) );
    }

    void Update()
    {
        // To get existing object or create new from pool
        var newEnemy = this.objectPool.Rent();
        // To return object to pool
        this.objectPool.Return(newEnemy);
        // In this example the message 'Creating new object' should only be seen on the frame
call
        // after that the same object in the pool will be returned.
    }
}

```

のなオブジェクトプール

のをつ。

Weaponは、するBulletsのオブジェクトプールとしてします。

```

public class Weapon : MonoBehaviour {

    // The Bullet prefab that the Weapon will create
    public Bullet bulletPrefab;

    // This List is our object pool, which starts out empty
    private List<Bullet> availableBullets = new List<Bullet>();

    // The Transform that will act as the Bullet starting position
    public Transform bulletInstantiationPoint;

    // To spawn a new Bullet, this method either grabs an available Bullet from the pool,
    // otherwise Instantiates a new Bullet
    public Bullet CreateBullet () {
        Bullet newBullet = null;

        // If a Bullet is available in the pool, take the first one and make it active
        if (availableBullets.Count > 0) {
            newBullet = availableBullets[availableBullets.Count - 1];

            // Remove the Bullet from the pool
            availableBullets.RemoveAt(availableBullets.Count - 1);

            // Set the Bullet's position and make its GameObject active
            newBullet.transform.position = bulletInstantiationPoint.position;
            newBullet.gameObject.SetActive(true);
        }
        // If no Bullets are available in the pool, Instantiate a new Bullet
        else {
            newBullet newObject = Instantiate(bulletPrefab, bulletInstantiationPoint.position,
Quaternion.identity);

            // Set the Bullet's Weapon so we know which pool to return to later on
            newBullet.weapon = this;
        }

        return newBullet;
    }
}

```

```
    }  
  
}  
  
public class Bullet : MonoBehaviour {  
  
    public Weapon weapon;  
  
    // When Bullet collides with something, rather than Destroying it, we return it to the  
pool  
    public void ReturnToPool () {  
        // Add Bullet to the pool  
        weapon.availableBullets.Add(this);  
  
        // Disable the Bullet's GameObject so it's hidden from view  
        gameObject.SetActive(false);  
    }  
  
}
```

オンラインでオブジェクトプーリングをむ <https://riptutorial.com/ja/unity3d/topic/2276/オブジェクトプーリング>

15: クォータニオン

- Quaternion.LookRotation(Vector3 forward [, Vector3 up]);
- Quaternion.AngleAxis(float angle, Vector3 axisOfRotation);
- float angleBetween = Quaternion.Angle(クォータニオン1, クォータニオン2);

Examples

クォータニオンとオイラーの

オイラーは90、180、45、30のような「」です。クォータニオンはオイラーとなり、のをしますは1です。このは、でぶの3Dバージョンとえることができます。クォータニオンはオイラーとなり、3Dをするためにをします。

これはになるかもしれませんがおそらくそれはいないかもしれませんが、ユニティは、オイラーとクォータニオンののりえをにするれたみみと、クォータニオンをするをえています。

オイラーとクォータニオンの

```
// Create a quaternion that represents 30 degrees about X, 10 degrees about Y
Quaternion rotation = Quaternion.Euler(30, 10, 0);

// Using a Vector
Vector3 EulerRotation = new Vector3(30, 10, 0);
Quaternion rotation = Quaternion.Euler(EulerRotation);

// Convert a transform's Quaternion angles to Euler angles
Quaternion quaternionAngles = transform.rotation;
Vector3 eulerAngles = quaternionAngles.eulerAngles;
```

クォータニオンをう

クォータニオンは、ジンバルロッキングとばれるをします。これは、のが3とになるときにします。[なは @ 2:09](#)です

クォータニオンルックローテーション

Quaternion.LookRotation(Vector3 forward [, Vector3 up])は、ベクトルをにて、Yを 'up'ベクトルにえたクォータニオンをします。アップベクトルがされていない、Vector3.upがされます。

このゲームオブジェクトをさせて、ターゲットゲームオブジェクトをる

```
// Find a game object in the scene named Target
public Transform target = GameObject.Find("Target").GetComponent<Transform>();

// We subtract our position from the target position to create a
// Vector that points from our position to the target position
```

```
// If we reverse the order, our rotation would be 180 degrees off.  
Vector3 lookVector = target.position - transform.position;  
Quaternion rotation = Quaternion.LookRotation(lookVector);  
transform.rotation = rotation;
```

オンラインでクォータニオンをむ <https://riptutorial.com/ja/unity3d/topic/1782/クォータニオン>

16: ゲームオブジェクトのと

- `public static GameObject Find;`
- パブリック `GameObject FindGameObjectWithTag` タグ;
- パブリック `GameObject [] FindGameObjectsWithTag` タグ;
- `public static Object FindObjectOfType` タイプタイプ;
- `public static Object [] FindObjectsOfType` タイプタイプ;

する

に `GameObjects` をしているときは、リソースをすることがあるのでしてください。に、`FindObjectOfType` や `Find in Update`、`FixedUpdate` をしないでください。よりには、フレームあたり1つのとばれるメソッドでしてください。

- メソッド `FindObjectOfType` をびし、なとときにのみ `Find`
- `FindGameObjectWithTag` はのベースのメソッドとしてにれたパフォーマンスをとっています。Unityは、タグきオブジェクトに々のタブをち、シーンではなくします。
- エディタでされた "な" `GameObjectsUI` や `プレハブ` などは、エディタでシリアルライズな `GameObject` [リファレンス](#) をします
- であるリストまたはに `GameObject` のリストを
- に、じタイプの `GameObject` をくインスタンスすると、[Object Pooling](#)
- なをりしすることをけるために、をキャッシュします。

よりくむ

Unityにのメソッドにえて、のおよびメソッドをするのはです。

- `FindObjectsOfType()` は、スクリプトで `static` コレクションのリストをすることができます。シーンのオブジェクトをしてするよりも、オブジェクトのリストをすることはかにはです。
- または、インスタンスをベースの `Dictionary` にするスクリプトをし、できるシンプルなタグけシステムをすることができます。

Examples

ゲームオブジェクトの

```
var go = GameObject.Find("NameOfTheObject");
```

いやすい シーンのゲームオブジェクトのにじてパフォーマンスがします

	はであり、ユーザーエラーがわれる
--	------------------

GameObjectのタグでする

```
var go = GameObject.FindGameObjectWithTag("Player");
```

のオブジェクトとグループ のが	はであり、ユーザーエラーのいがあります。
--------------------	----------------------

かつ	タグはスクリプトでハードコードされているため、コードはできません。
----	-----------------------------------

モードでスクリプトに

```
[SerializeField]  
GameObject[] gameObjects;
```

らしいパフォーマンス	オブジェクトコレクションはです
------------	-----------------

ポータブルコード	じのGameObjectsのみをできる
----------	---------------------

MonoBehaviourスクリプトによるゲームオブジェクトの

```
ExampleScript script = GameObject.FindObjectOfType<ExampleScript>();  
GameObject go = script.gameObject;
```

FindObjectOfType() は、もつからない場合はnullしnull。

くけされた	になのゲームオブジェクトによってパフォーマンスがする
-------	----------------------------

のオブジェクトとグループのが	
----------------	--

オブジェクトからでGameObjectをす

```
Transform tr = GetComponent<Transform>().Find("NameOfTheObject");  
GameObject go = tr.gameObject;
```

つからない場合はnull Findしnull

られた、にされた	はです
----------	-----

オンラインでゲームオブジェクトのとをむ <https://riptutorial.com/ja/unity3d/topic/3793/ゲームオブジェクトのと>

17: コルーチン

- パブリック Coroutine StartCoroutineIEnumeratorルーチン;
- public Coroutine StartCoroutinemethodName、オブジェクト= null;
- public void StopCoroutinestring methodName;
- public void StopCoroutineIEnumeratorルーチン;
- public void StopAllCoroutines;

パフォーマンスの

にはパフォーマンスコストがうため、コルーチンをにすることがです。

- Coroutinesは、なUpdateメソッドよりもCPUからのがい。
- ユニティボックスのMoveNextりのためにコルーチンがサイクルでゴミをずるUnityのいくつかのバージョンではあります。これは5.4.0b13でにされた。 [バグレポート](#)

YieldInstructionsをキャッシュしてゴミをらす

コルーチンでされるゴミをらすためのなトリックは、YieldInstructionをキャッシュすることYieldInstruction。

```
IEnumerator TickEverySecond()
{
    var wait = new WaitForSeconds(1f); // Cache
    while(true)
    {
        yield return wait; // Reuse
    }
}
```

nullをすと、なガページがしません。

Examples

コルーチン

まず、ゲームエンジンUnityなどが「フレームベース」のパラダイムであることをすることがです。

コードはすべてのフレームでされます。

これにはUnityのコードとコードがまれます。

フレームについてえるとき、フレームがいつするかはにされないことをすることがです。らはな

ビートでこることはありません。フレームのギャップは、例えば、0.02632、0.021167、0.029778などとすることができる。ここでは、すべて「」1/50ですが、それらはすべてなります。そして、いつでも、あなたはよりい、またはよりいがかかるフレームをるかもしれません。フレームでいつでもコードをすることができます。

それをにいて、あなたはするかもしれませんどのようにしてこれらのフレームにあなたのコード、Unityでアクセスしますか

にうと、Updateコールをするか、コルーチンをしします。まったくじことです。つまり、フレームごとにコードをすることができます。

コルーチンのはのとおりです。

いくつかのコードをして、のフレームまで「してつ」ことができます。

のフレームまでつことができます。また、いくつかのフレームをつこともできますし、にはおおよそのをつこともできます。

たとえば、1つことをする「1」をつことができます。そして、おおよそ1にコードをいくつかのフレームにれます。、そのフレームでは、いつでもコードをすることができます。りすには、ちょうど1ではありません。なタイミングは、ゲームエンジンではがありません。

コルーチン

1つのフレームをつには

```
// do something
yield return null; // wait until next frame
// do something
```

3つのフレームをつには

```
// do something
yield return null; // wait until three frames from now
yield return null;
yield return null;
// do something
```

0.5つ

```
// do something
yield return new WaitForSeconds (0.5f); // wait for a frame in about .5 seconds
// do something
```

1つのフレームごとにかする

```
while (true)
{
    // do something
}
```

```
yield return null; // wait until the next frame
}
```

これは、Unityの「Update」コールのにかをれることとまったく同じです。「かをする」というコードは、すべてのフレームでされます。

TickerをGameObjectとしてGameObject。そのゲームオブジェクトがアクティブな、ティックがされます。ゲームオブジェクトがアクティブになると、スクリプトはくルーチンをするにしてください。これは、ルーチンのをしくするなです。

```
using UnityEngine;
using System.Collections;

public class Ticker:MonoBehaviour {

    void OnEnable()
    {
        StartCoroutine(TickEverySecond());
    }

    void OnDisable()
    {
        StopAllCoroutines();
    }

    IEnumerator TickEverySecond()
    {
        var wait = new WaitForSeconds(1f); // REMEMBER: IT IS ONLY APPROXIMATE
        while(true)
        {
            Debug.Log("Tick");
            yield return wait; // wait for a frame, about 1 second from now
        }
    }
}
```

ルーチンをする

あるがされたら、にするようにルーチンをすることがよくあります。

```
IEnumerator TickFiveSeconds()
{
    var wait = new WaitForSeconds(1f);
    int counter = 1;
    while(counter < 5)
    {
        Debug.Log("Tick");
        counter++;
        yield return wait;
    }
    Debug.Log("I am done ticking");
}
```

コルーチンをコルーチンの ""からするには、のからにれるようにに "る"ことはできません。わりに、 `yield break` をします。

```
IEnumerator ShowExplosions()
{
    ... show basic explosions
    if(player.xp < 100) yield break;
    ... show fancy explosions
}
```

するに、スクリプトによってされたすべてのコルーチンをにすることもできます。

```
void OnDisable()
{
    // Stops all running coroutines
    StopAllCoroutines();
}
```

びしからのコルーチンをするは、によってなります。

コルーチンをでした

```
StartCoroutine("YourAnimation");
```

[StopCoroutine](#) をじでびすことでできます。

```
StopCoroutine("YourAnimation");
```

または、コルーチンメソッドによってされた `IEnumerator` または `StartCoroutine` によってされる `Coroutine` オブジェクトのいずれかへのをし、それらのいずれかで `StopCoroutine` をびす `StopCoroutine` ができます。

```
public class SomeComponent : MonoBehaviour
{
    Coroutine routine;

    void Start () {
        routine = StartCoroutine(YourAnimation());
    }

    void Update () {
        // later, in response to some input...
        StopCoroutine(routine);
    }

    IEnumerator YourAnimation () { /* ... */ }
}
```

コルーチンができる `MonoBehaviour` メソッド

コルーチンにすることができる3つの `MonoBehaviour` メソッドがあります。

- 1.
2. OnBecameVisible
3. OnLevelWasLoaded

これは、たとえばオブジェクトがカメラにえるときにのみされるスクリプトをするためにできます。

```
using UnityEngine;
using System.Collections;

public class RotateObject : MonoBehaviour
{
    IEnumerator OnBecameVisible()
    {
        var tr = GetComponent<Transform>();
        while (true)
        {
            tr.Rotate(new Vector3(0, 180f * Time.deltaTime));
            yield return null;
        }
    }

    void OnBecameInvisible()
    {
        StopAllCoroutines();
    }
}
```

コルーチンの

コルーチンはでし、のコルーチンをつことができます。

だから、あなたは "もうつの"をさせることができます。

これはにで、Unityのとなるです。

ゲームでは、のことが「どおりに」こらなければならぬのはのことです。ゲームのほほすべての「ラウンド」は、のとに、らかのでこっているのイベントからまります。カーレースのゲームをめるはのとおりです。

```
IEnumerator BeginRace()
{
    yield return StartCoroutine(PrepareRace());
    yield return StartCoroutine(Countdown());
    yield return StartCoroutine(StartRace());
}
```

だから、BeginRaceをびすと...

```
StartCoroutine(BeginRace());
```

それはあなたの "レース"ルーチンをします。おそらく、いくつかのライトをさせ、いくつかの

をらし、スコアをリセットするなど。これがすると、おそらくUIのカウントダウンをアニメーションする「カウントダウン」シーケンスがされます。それがわると、レーススタートコードがされ、サウンドエフェクトをしたり、AIドライバをしたり、カメラをのさせたりします。

わかりやすくするために、3つのびし

```
yield return StartCoroutine(PrepareRace());
yield return StartCoroutine(Countdown());
yield return StartCoroutine(StartRace());
```

コルーチンのにいます。つまり、`IEnumerator`のなければなりません。したがって、このでは`IEnumerator BeginRace`です。したがって、「の」コードから、`StartCoroutine`コールでそのコルーチンをしします。

```
StartCoroutine(BeginRace());
```

をさらにするために、コルーチンをさせるがあります。コルーチンのをしします。このは、に、にくのコルーチンをしします。

```
// run various routines, one after the other
IEnumerator OneAfterTheOther( params IEnumerator[] routines )
{
    foreach ( var item in routines )
    {
        while ( item.MoveNext() ) yield return item.Current;
    }

    yield break;
}
```

ここでそれをどのようにびすかをしします。3つのがあるとしまししょう。らはすべて`IEnumerator`なければならないことをいしてください

```
IEnumerator PrepareRace()
{
    // codesay, crowd cheering and camera pan around the stadium
    yield break;
}

IEnumerator Countdown()
{
    // codesay, animate your countdown on UI
    yield break;
}

IEnumerator StartRace()
{
    // codesay, camera moves and light changes and launch the AIs
    yield break;
}
```

あなたはこれをこのようにびます

```
StartCoroutine( MultipleRoutines( PrepareRace(), Countdown(), StartRace() ) );
```

またはこれとじように

```
IEnumerator[] routines = new IEnumerator[] {  
    PrepareRace(),  
    Countdown(),  
    StartRace() };  
StartCoroutine( MultipleRoutines( routines ) );
```

りすには、ゲームのもの1つは、のとともにあるものが々と「シーケンスで」こるということ
です。あなたはUnityでそれをににします。

```
yield return StartCoroutine(PrepareRace());  
yield return StartCoroutine(Countdown());  
yield return StartCoroutine(StartRace());
```

する

のフレームまでつことができます。

```
yield return null; // wait until sometime in the next frame
```

これらのびしをしてして、なだけくのフレームをさせることができます。

```
//wait for a few frames  
yield return null;  
yield return null;
```

nちます。これはにおおよそのであることをすることはにです。

```
yield return new WaitForSeconds(n);
```

なタイミングののフォームにして「WaitForSeconds」コールをすることはにです。

くの、アクションをさせたいとっています。それで、かをして、それがわったらかをしてくださ
い、そして、それがわったらかのことをしてください。それをするには、のルーチンをつ

```
yield return StartCoroutine(coroutine);
```

あなたはルーチンからしかそれをびすことができないことをする。そう

```
StartCoroutine(Test());
```

これは、あなたが"の"コードからルーチンをめるです。

に、のルーチンで

```
Debug.Log("A");
StartCoroutine(LongProcess());
Debug.Log("B");
```

それはAをし、いプロセスをし、すぐにBをします。いプロセスがするのを待つことはありません。

```
Debug.Log("A");
yield return StartCoroutine(LongProcess());
Debug.Log("B");
```

それはAをし、いプロセスをし、するまでってから、Bをします。

コルーチンはスレッドすることがにのもないことをれないでください。このコードでは

```
Debug.Log("A");
StartCoroutine(LongProcess());
Debug.Log("B");
```

バックグラウンドでのスレッドでLongProcessをするような"もの"とえるのはです。しかし、それはにっています。それはなるコルーチンです。ゲームエンジンはフレームベースであり、Unityの「コルーチン」はにフレームにアクセスすることができます。

Webリクエストがするのをつのはとてもです。

```
void Start() {
    string url = "http://google.com";
    WWW www = new WWW(url);
    StartCoroutine(WaitForRequest(www));
}

IEnumerator WaitForRequest(WWW www) {
    yield return www;

    if (www.error == null) {
        //use www.data);
    }
    else {
        //use www.error);
    }
}
```

にまれなケースでは、Unityでのをします。はされないWaitForFixedUpdate()びしがあります。にスクリーンキャプチャをすることにしてのでされるのびしのバージョンのUnityではWaitForEndOfFrame()があります。Unityがするにつれて、なみがしわるので、があるならば、のをgoogleにってください。

オンラインでコルーチンをもむ <https://riptutorial.com/ja/unity3d/topic/3415/コルーチン>

18: サーバーとの

Examples

する

Webサーバーからデータをしています。 `new WWW("https://urlexample.com");` URLをして2のパラメータがないは**Get**をしています。

すなわち、

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour
{
    public string url = "http://google.com";

    IEnumerator Start()
    {
        WWW www = new WWW(url); // One get.
        yield return www;
        Debug.Log(www.text); // The data of the url.
    }
}
```

フィールド

2のパラメータをつ**WWW**のすべてのインスタンスはポストです。

に、ユーザーIDとパスワードをサーバーにするをします。

```
void Login(string id, string pwd)
{
    WWWForm dataParameters = new WWWForm(); // Create a new form.
    dataParameters.AddField("username", id);
    dataParameters.AddField("password", pwd); // Add fields.
    WWW www = new WWW(url+"/account/login", dataParameters);
    StartCoroutine("PostdataEnumerator", www);
}

IEnumerator PostdataEnumerator(WWW www)
{
    yield return www;
    if (!string.IsNullOrEmpty(www.error))
    {
        Debug.Log(www.error);
    }
    else
    {
        Debug.Log("Data Submitted");
    }
}
```



```
}
```

ファイルをアップロードする

サーバーにファイルをアップロードすることもポストです。のように、**WWW**からにファイルをアップロードすることができます

Zip ファイルをサーバーにアップロードする

```
string mainUrl = "http://server/upload/";
string saveLocation;

void Start()
{
    saveLocation = "ftp:///home/xxx/x.zip"; // The file path.
    StartCoroutine(PrepareFile());
}

// Prepare The File.
IEnumerator PrepareFile()
{
    Debug.Log("saveLoacation = " + saveLocation);

    // Read the zip file.
    WWW loadTheZip = new WWW(saveLocation);

    yield return loadTheZip;

    PrepareStepTwo(loadTheZip);
}

void PrepareStepTwo(WWW post)
{
    StartCoroutine(UploadTheZip(post));
}

// Upload.
IEnumerator UploadTheZip(WWW post)
{
    // Create a form.
    WWWForm form = new WWWForm();

    // Add the file.
    form.AddBinaryData("myTestFile.zip", post.bytes, "myFile.zip", "application/zip");

    // Send POST request.
    string url = mainUrl;
    WWW POSTZIP = new WWW(url, form);

    Debug.Log("Sending zip...");
    yield return POSTZIP;
    Debug.Log("Zip sent!");
}
}
```

あなたはユニティコルーチンについてののをおりになりたいは、このでは、それは、してファイルをアップロードするためにコルーチンをして、ごください [コルーチン](#) を。

サーバーへのリクエストの

Unityをクライアントとしてしてサーバとするはたくさんありますにじていくつかのがのよりれています。まず、サーバーとのでをにできるようにするために、サーバーのをするがあります。このでは、するためにいくつかのデータをサーバーにします。

ほとんどの、プログラマは、イベントをしてクライアントにじてするために、サーバにらかのハンドラをしますが、それはこののです。

C

```
using System.Net;
using System.Text;

public class TestCommunicationWithServer
{
    public string SendDataToServer(string url, string username, string password)
    {
        WebClient client = new WebClient();

        // This specialized key-value pair will store the form data we're sending to the
server
        var loginData = new System.Collections.Specialized.NameValueCollection();
        loginData.Add("Username", username);
        loginData.Add("Password", password);

        // Upload client data and receive a response
        byte[] opBytes = client.UploadValues(ServerIpAddress, "POST", loginData);

        // Encode the response bytes into a proper string
        string opResponse = Encoding.UTF8.GetString(opBytes);

        return opResponse;
    }
}
```

まず、WebClientクラスとNameValueCollectionクラスをできるusingステートメントをします。

このでは、SendDataToServerは3つのオプションのパラメータをります。

1. たちがしているサーバーのURL
2. のデータ
3. サーバーにするデータの2の

ユーザーとパスワードは、サーバーにするオプションのデータです。このでは、データベースやそののストレージからさらにするためにしています。

をしたので、にデータをするためにするしいWebClientをインスタンスします。これで、NameValueCollectionにデータをロードし、サーバーにデータをアップロードするがあります。

UploadValuesは3つのなパラメータもります

1. サーバのIPアドレス

2. HTTPメソッド

3. しているデータこのユーザーとパスワード

これは、サーバーからのバイトをします。されたバイトをのにエンコードして、レスポンスを
してできるようにするがあります。

のようなことができます

```
if (opResponse.Equals (ReturnMessage.Success))  
{  
    Debug.Log("Unity client has successfully sent and validated data on server.");  
}
```

でもあなたはしているかもしれないので、は、サーバーをうについてにします。

ここでは、PHPをしてクライアントからのをします。はPHPをバックエンドスクリプトとしてす
ることをおめします。これはでいやすく、ほとんどののはです。サーバでをするのはありません
が、のでは、PHPはUnityへのもでなです。

PHP

```
// Check to see if the unity client send the form data  
if (!isset($_REQUEST['Username']) || !isset($_REQUEST['Password']))  
{  
    echo "Empty";  
}  
else  
{  
    // Unity sent us the data - its here so do whatever you want  
  
    echo "Success";  
}
```

これはもなです - エコーです。クライアントがデータをサーバーにアップロードすると、クライ
アントはそのバイトにまたはリソースをします。クライアントがをけると、データがされたこと
をっているので、そのイベントがするとクライアントでできます。また、しているデータのある
とにしているをにえるについてえるがあります。

これは、Unityからデータをするので。プロジェクトによっては、よりながいくつかあります。

オンラインでサーバーとのをむ <https://riptutorial.com/ja/unity3d/topic/5578/サーバーとの>

19: タグ

き

タグは `GameObject` をマークするためにできます。このようにして、の `GameObject` オブジェクトをコードですることがになります。

1つまたはのゲームオブジェクトにタグをできますが、ゲームオブジェクトにはに1つのタグしかありません。デフォルトでは、タグ「Untagged」は、にタグけされていない `GameObject` をすためにされます。

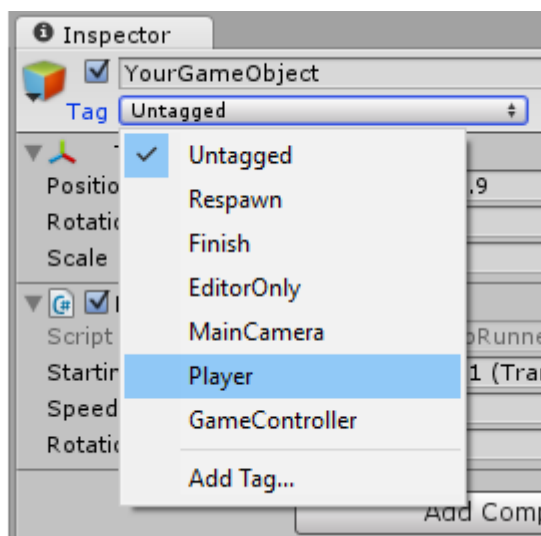
Examples

タグのと

タグは、エディタをしてされます。ただし、スクリプトをしてタグをすることもできます。すべてのカスタムタグは、ゲームオブジェクトにされるに[タグとレイヤー]ウィンドウでするがあります。

エディタでタグをする

1つまたはのゲームオブジェクトをしたで、インスペクタからタグをできます。ゲームオブジェクトにはに1つのタグをちます。デフォルトでは、ゲームオブジェクトは「タグなし」としてタグけされます。[タグとレイヤー]ウィンドウにするには、[タグを ...]をします。ただし、これは、[タグとレイヤー]ウィンドウにするだけであることにすることがです。したタグはゲームオブジェクトにはにされません。



スクリプトによるタグの

コードをしてゲームオブジェクトタグをすることができます。のタグのリストからタグをするがあることにすることができます。まだされていないタグをすると、エラーになります。

のでされているように、タグをできむのではなく、の `static string` をすると、とをできます。

のスクリプトは、 `static string` をしてをするためにのゲームオブジェクトタグをするをしています。 `static string` は、[タグとレイヤー]ウィンドウですすでにされているタグをしていることにしてください。

```
using UnityEngine;

public class Tagging : MonoBehaviour
{
    static string tagUntagged = "Untagged";
    static string tagPlayer = "Player";
    static string tagEnemy = "Enemy";

    /// <summary>Represents the player character. This game object should
    /// be linked up via the inspector.</summary>
    public GameObject player;
    /// <summary>Represents all the enemy characters. All enemies should
    /// be added to the array via the inspector.</summary>
    public GameObject[] enemy;

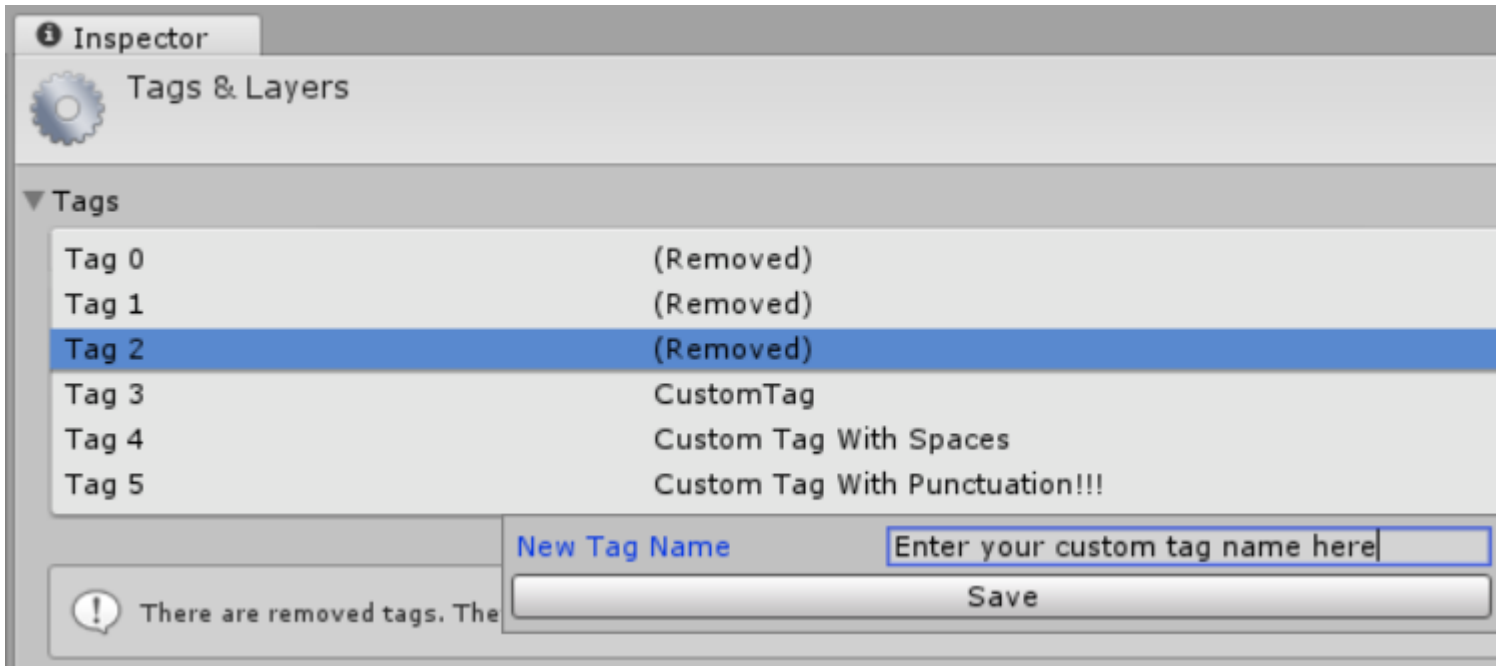
    void Start ()
    {
        // We ensure that the game object this script is attached to
        // is left untagged by using the default "Untagged" tag.
        gameObject.tag = tagUntagged;

        // We ensure the player has the player tag.
        player.tag = tagUntagged;

        // We loop through the enemy array to ensure they are all tagged.
        for(int i = 0; i < enemy.Length; i++)
        {
            enemy[i].tag = tagEnemy;
        }
    }
}
```

カスタムタグの

インスペクタをしてタグをするか、スクリプトをしてタグをするかにかかわらず、タグはに[タグとレイヤー]ウィンドウでするがあります。このウィンドウにアクセスするには、ゲームオブジェクトのタグドロップダウンメニューから[タグを...]をします。または、**[> [プロジェクト]> [タグとレイヤー]**のにしてください。



に+ボタンをし、のをし、[]をしてタグをします。-ボタンをすると、されているタグがされます。ここでは、タグはすぐに「された」とされ、プロジェクトがにみまれるとにされます。

ウィンドウのにある/をすると、すべてのカスタムオプションをリセットできます。これにより、「レイヤーのべえ」および「レイヤー」のにあるカスタムレイヤーとともに、すべてのカスタムタグがにされます。

タグによるゲームオブジェクトの

タグをすると、のゲームオブジェクトのがにになります。1つのゲームオブジェクトをしたり、のゲームオブジェクトをすことができます。

の `GameObject` つける

`GameObject.FindGameObjectWithTag(string tag)` をして々のゲームオブジェクトをすことができます。このように、ゲームオブジェクトはのでされないことにすることがです。シーンのゲームオブジェクトでされているタグをす、このはされるゲームオブジェクトをすることはできません。したがって、このようなタグをすゲームオブジェクトが1つしかないことがわかっている、またはされる `GameObject` のなインスタンスがされていないは、よりです。

```
///
```

の `GameObject` インスタンスの `GameObject`

`GameObject.FindGameObjectsWithTag(string tag)`をして、のタグをするすべてのゲームオブジェクトをできます。これは、のゲームオブジェクトのグループをしたいときにです。これは、のゲームオブジェクトをしたいが、じタグをってのゲームオブジェクトをつにもです。々によってはされたなインスタンスをすることはできませんのよう `GameObject.FindGameObjectWithTag(string tag)`、々ではなく、すべてののをしなければならぬ `GameObject` つインスタンス `GameObject.FindGameObjectsWithTag(string tag)`、さらに々は、インスタンスをつけるために、のをしますしている。

```
///
```

タグの

タグで2つの `GameObject` をすると、がされるため、のことがガベージコレクタのオーバーヘッドをきこすことにしてください。

```
if (go.Tag == "myTag")
{
    //Stuff
}
```

`Update`やののUnityのコールバックまたはループでこれらのをするは、このヒープリてフリーメソッドをするがあります。

```
if (go.CompareTag("myTag")
{
    //Stuff
}
```

さらに、タグをなクラスにするがです。

```
public static class Tags
{
    public const string Player = "Player";
    public const string MyCustomTag = "MyCustomTag";
}
```

に、あなたはにすることができます

```
if (go.CompareTag(Tags.MyCustomTag)
```

```
{
    //Stuff
}
```

ここでは、コンパイルにタグがされ、スペルミスのがされます。

タグをなクラスにするのと同じように、にタグをすることもできます。

```
public enum Tags
{
    Player, Enemies, MyCustomTag;
}
```

`enum toString()` メソッドをすることができます。

```
if (go.CompareTag (Tags.MyCustomTag.toString())
{
    //Stuff
}
```

オンラインでタグをむ <https://riptutorial.com/ja/unity3d/topic/5534/タグ>

20: デザインパターン

Examples

モデルビューコントローラMVCデザインパターン

モデルビューコントローラは、MVCデザインパターンであり、かなりいしてきました。このパターンは、クラスをにけることによってスパゲッティコードをらすことにをてています。、はUnityでこのデザインパターンをしていて、なをしたいといします。

MVCデザインは、Model、View、Controllerの3つのコアでされています。

モデルモデルは、オブジェクトのデータをすクラスです。これはプレイヤー、、またはレベルであるがあります。しくプログラムされていれば、このスクリプトをしてUnityのですることができます。

モデルについていくつかしてください。

- MonoBehaviourからすべきではありません
- それはのためのUnityのコードをんではいけません
- Unity APIびしをけているので、これはModelクラスののコンバータのようなものをけるがあります

Player.cs

```
using System;

public class Player
{
    public delegate void PositionEvent(Vector3 position);
    public event PositionEvent OnPositionChanged;

    public Vector3 position
    {
        get
        {
            return _position;
        }
        set
        {
            if (_position != value) {
                _position = value;
                if (OnPositionChanged != null) {
                    OnPositionChanged(value);
                }
            }
        }
    }
    private Vector3 _position;
}
```

```
}
```

Vector3.cs

データモデルであるカスタムVector3クラス。

```
using System;

public class Vector3
{
    public float x;
    public float y;
    public float z;

    public Vector3(float x, float y, float z)
    {
        this.x = x;
        this.y = y;
        this.z = z;
    }
}
```

ビュービューは、モデルにけられたをすクラスです。これはMonobehaviourからするのにしたクラスです。これには、 OnCollisinEnter、 Start、 UpdateなどのUnityのAPIとするコードがまれているがあります。

- にMonobehaviourからします
- ユニティのコードがまれています

PlayerView.cs

```
using UnityEngine;

public class PlayerView : MonoBehaviour
{
    public void SetPosition(Vector3 position)
    {
        transform.position = position;
    }
}
```

コントローラコントローラは、モデルとビューのをバインドするクラスです。コントローラは、モデルとビューのをさせ、ドライブのやりとりをします。コントローラは、いずれかのパートナーからのイベントをリッスンし、それにじてすることができます。

- でモデルとビューのをバインドする
- パートナーのをできます
- コントローラはポータブルでも、そうでなくてもかまいませんUnityコードをここでするがあります
- コントローラをにしないは、エディタのにつMonobehaviourをすることをしてください

PlayerController.cs

```
using System;

public class PlayerController
{
    public Player model { get; private set; }
    public PlayerView view { get; private set; }

    public PlayerController(Player model, PlayerView view)
    {
        this.model = model;
        this.view = view;

        this.model.OnPositionChanged += OnPositionChanged;
    }

    private void OnPositionChanged(Vector3 position)
    {
        // Sync
        Vector3 pos = this.model.position;

        // Unity call required here! (we lost portability)
        this.view.SetPosition(new UnityEngine.Vector3(pos.x, pos.y, pos.z));
    }

    // Calling this will fire the OnPositionChanged event
    private void SetPosition(Vector3 position)
    {
        this.model.position = position;
    }
}
```

ながすべてしたので、3つのすべてをするファクトリをできます。

PlayerFactory.cs

```
using System;

public class PlayerFactory
{
    public PlayerController controller { get; private set; }
    public Player model { get; private set; }
    public PlayerView view { get; private set; }

    public void Load()
    {
        // Put the Player prefab inside the 'Resources' folder
        // Make sure it has the 'PlayerView' Component attached
        GameObject prefab = Resources.Load<GameObject>("Player");
        GameObject instance = GameObject.Instantiate<GameObject>(prefab);
        this.model = new Player();
        this.view = instance.GetComponent<PlayerView>();
        this.controller = new PlayerController(model, view);
    }
}
```

そしてに、マネージャーからにすることができます...

Manager.cs

```
using UnityEngine;

public class Manager : MonoBehaviour
{
    [ContextMenu("Load Player")]
    private void LoadPlayer()
    {
        new PlayerFactory().Load();
    }
}
```

シーンのGameObjectにManagerスクリプトをし、コンポーネントをクリックして "Load Player"をします。

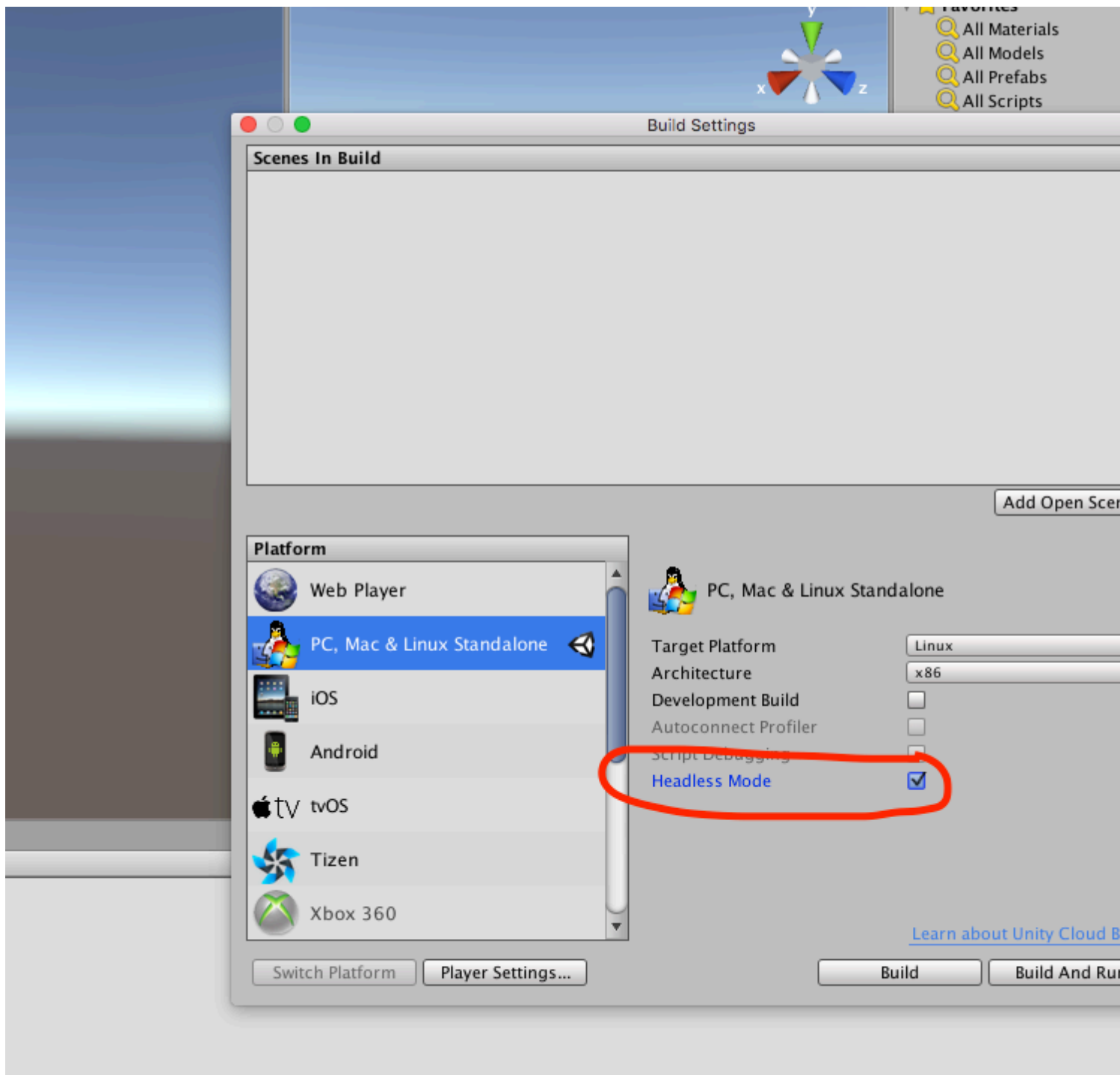
よりなロジックのは、クラスとインタフェースをして、されたアーキテクチャをすることができます。

オンラインでデザインパターンをむ <https://riptutorial.com/ja/unity3d/topic/10842/デザインパターン>

21: ネットワーキング

Unityのヘッドレスモード

Linuxでするサーバーをする、ビルドには「ヘッドレスモード」オプションがあります。このオプションをしてアプリケーションをビルドすると、もされず、ユーザーのがみられません。これは、サーバーにとってなものです。



Examples

サーバー、クライアントをし、メッセージをする。

Unityネットワークは、レベルのからされたネットワークをするためのAPIHLAをします。

このでは、1つまたはのクライアントとできるサーバーをするをします。

HLAでは、クラスをシリアライズして、このクラスのオブジェクトをネットワークでにすることができます。

シリアライズするためにしているクラス

このクラスはMessageBaseからするがあります。このでは、このクラスにをします。

```
using System;
using UnityEngine.Networking;

public class MyNetworkMessage : MessageBase
{
    public string message;
}
```

サーバーの

ポート9999をリッスンし、10のをし、カスタムクラスのネットワークからオブジェクトをみむサーバーをします。

HLAはなるタイプのメッセージをIDにける。Unity NetworkingのMsgTypeクラスでされているデフォルトのメッセージタイプがあります。たとえば、タイプはID 32をち、クライアントがするときにはサーバでびされます。サーバにするときにはクライアントでびされます。ハンドラをして、さまざまなメッセージをできます。

たちのようにカスタムクラスをするときには、ネットワークでするクラスにけられたしいIDをつハンドラをします。

```
using UnityEngine;
using System.Collections;
using UnityEngine.Networking;

public class Server : MonoBehaviour {

    int port = 9999;
    int maxConnections = 10;

    // The id we use to identify our messages and register the handler
    short messageID = 1000;

    // Use this for initialization
    void Start () {
```

```

    // Usually the server doesn't need to draw anything on the screen
    Application.runInBackground = true;
    CreateServer();
}

void CreateServer() {
    // Register handlers for the types of messages we can receive
    RegisterHandlers ();

    var config = new ConnectionConfig ();
    // There are different types of channels you can use, check the official documentation
    config.AddChannel (QosType.ReliableFragmented);
    config.AddChannel (QosType.UnreliableFragmented);

    var ht = new HostTopology (config, maxConnections);

    if (!NetworkServer.Configure (ht)) {
        Debug.Log ("No server created, error on the configuration definition");
        return;
    } else {
        // Start listening on the defined port
        if(NetworkServer.Listen (port))
            Debug.Log ("Server created, listening on port: " + port);
        else
            Debug.Log ("No server created, could not listen to the port: " + port);
    }
}

void OnApplicationQuit() {
    NetworkServer.Shutdown ();
}

private void RegisterHandlers () {
    // Unity have different Messages types defined in MessageType
    NetworkServer.RegisterHandler (MessageType.Connect, OnClientConnected);
    NetworkServer.RegisterHandler (MessageType.Disconnect, OnClientDisconnected);

    // Our message use his own message type.
    NetworkServer.RegisterHandler (messageID, OnMessageReceived);
}

private void RegisterHandler(short t, NetworkMessageDelegate handler) {
    NetworkServer.RegisterHandler (t, handler);
}

void OnClientConnected(NetworkMessage netMessage)
{
    // Do stuff when a client connects to this server

    // Send a thank you message to the client that just connected
    MyNetworkMessage messageContainer = new MyNetworkMessage();
    messageContainer.message = "Thanks for joining!";

    // This sends a message to a specific client, using the connectionId
    NetworkServer.SendToClient(netMessage.conn.connectionId,messageID,messageContainer);

    // Send a message to all the clients connected
    messageContainer = new MyNetworkMessage();
    messageContainer.message = "A new player has conected to the server";

    // Broadcast a message a to everyone connected

```

```

        NetworkServer.SendToAll (messageID,messageContainer);
    }

    void OnClientDisconnected(NetworkMessage netMessage)
    {
        // Do stuff when a client disssconnects
    }

    void OnMessageReceived(NetworkMessage netMessage)
    {
        // You can send any object that inherence from MessageBase
        // The client and server can be on different projects, as long as the MyNetworkMessage
or the class you are using have the same implementation on both projects
        // The first thing we do is deserialize the message to our custom type
        var objectMessage = netMessage.ReadMessage<MyNetworkMessage>();
        Debug.Log("Message received: " + objectMessage.message);
    }
}

```

クライアント

はクライアントをします

```

using System;
using UnityEngine;
using UnityEngine.Networking;

public class Client : MonoBehaviour
{
    int port = 9999;
    string ip = "localhost";

    // The id we use to identify our messages and register the handler
    short messageID = 1000;

    // The network client
    NetworkClient client;

    public Client ()
    {
        CreateClient();
    }

    void CreateClient()
    {
        var config = new ConnectionConfig ();

        // Config the Channels we will use
        config.AddChannel (QosType.ReliableFragmented);
        config.AddChannel (QosType.UnreliableFragmented);

        // Create the client ant attach the configuration
        client = new NetworkClient ();
        client.Configure (config,1);
    }
}

```



```

    // Register the handlers for the different network messages
    RegisterHandlers();

    // Connect to the server
    client.Connect (ip, port);
}

// Register the handlers for the different message types
void RegisterHandlers () {

    // Unity have different Messages types defined in MsgType
    client.RegisterHandler (messageID, OnMessageReceived);
    client.RegisterHandler(MsgType.Connect, OnConnected);
    client.RegisterHandler(MsgType.Disconnect, OnDisconnected);
}

void OnConnected(NetworkMessage message) {
    // Do stuff when connected to the server

    MyNetworkMessage messageContainer = new MyNetworkMessage();
    messageContainer.message = "Hello server!";

    // Say hi to the server when connected
    client.Send(messageID,messageContainer);
}

void OnDisconnected(NetworkMessage message) {
    // Do stuff when disconnected to the server
}

// Message received from the server
void OnMessageReceived(NetworkMessage netMessage)
{
    // You can send any object that inherence from MessageBase
    // The client and server can be on different projects, as long as the MyNetworkMessage
or the class you are using have the same implementation on both projects
    // The first thing we do is deserialize the message to our custom type
    var objectMessage = netMessage.ReadMessage<MyNetworkMessage>();

    Debug.Log("Message received: " + objectMessage.message);
}
}

```

オンラインでネットワーキングをむ <https://riptutorial.com/ja/unity3d/topic/5671/ネットワーキング>

22: バーチャルリアリティ VR

Examples

VRプラットフォーム

VRには、**Google Cardboard**、**Samsung GearVR**、**HTC Vive**、Oculus、PS VRなどのPCプラットフォームのようなモバイルプラットフォームが2つあります。

Unityは、**Oculus Rift**、**Google Carboard**、**Steam VR**、**Playstation VR**、**Gear VR**、**Microsoft Hololens**をにサポートしています。

ほとんどのプラットフォームはのサポートとSDKをとっています。は、まずにのためにsdkをダウンロードするがあります。

SDK

- [Google Cardboard](#)
- [デイドリームプラットフォーム](#)
- [Samsung GearVR](#) Unity 5.3にされています
- [オクルスリフト](#)
- [HTC Vive / Open VR](#)
- [Microsoft Hololens](#)

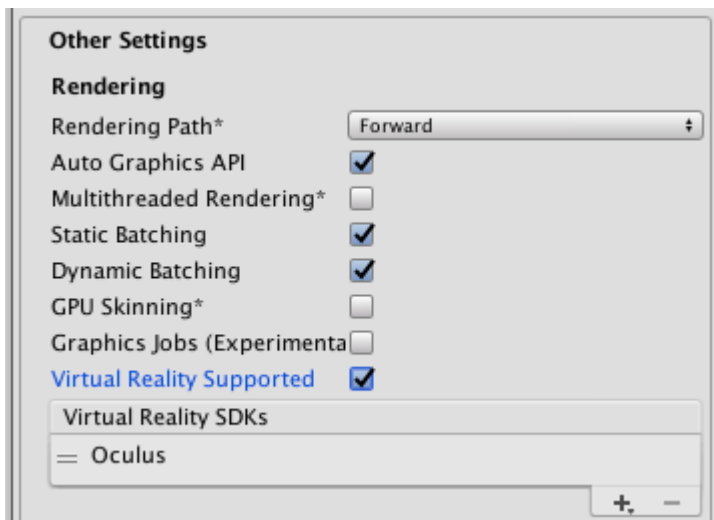
ドキュメンテーション

- [Google Cardboard / Daydream](#)
- [Samsung GearVR](#)
- [オクルスリフト](#)
- [HTC Vive](#)
- [Microsoft Hololens](#)

VRサポートをにする

Unity Editorで、「プレイヤー」をきます「」>「プロジェクト」>「プレーヤー」。

[そのの]で、[サポート]をオンにします。



チェックボックスの *Virtual Reality SDKs* リストで、ビルドターゲットのVRデバイスをまたはします。

ハードウェア

VRアプリケーションにはなハードウェアのがあります。これは、しているプラットフォームにします。ハードウェアデバイスには、モーションについて2つのいカテゴリがあります。

1. 3
2. 6

3 DOFは、ヘッドマウントディスプレイHMDのきが、HMDのをとする3つの、、およびをにする3できるようにされていることをします。のきをロールといい、のきをピッチといい、をとしたきをヨーといい、やのようなくのきをするのであり、であなたのHMDのきですべてのX、Y、Zをすることができますが、かをかすこともタッチすることもできませんのBluetoothコントローラによるきはじではありません。

しかし、6 DOFでは、ルームスケールでのエクスペリエンスがです.X、Y、Zのりを、、つまり6にするロール、ピッチ、ヨーかられてすることもできます。

のところ、6のスケールのVRは、ハイエンドのグラフィックカードとRAMをした、いをとします。なノートパソコンではられないでしようにし、なをえ、なくとも6ftx6ft 3のは、のジャイロのスマートフォンには200ドルのコストがかかりますをえたのスマートフォンだけでできます。

されているなデバイスには、のものがありません。

- [Oculus Rift 6](#)
- [HTC Vive 6](#)
- [デイドリーム 3 DOF](#)
- [ギアVR Powered by Oculus 3 DOF](#)
- [Google Cardboard 3](#)

オンラインでバーチャルリアリティVRをむ <https://riptutorial.com/ja/unity3d/topic/5787/バーチャ>

23: プレハブ

- `public static Object PrefabUtility.InstantiatePrefab`オブジェクトターゲット。
- `public static Object AssetDatabase.LoadAssetAtPathAssetPath`、タイプのタイプ。
- `public static Object Object.Instantiate`オブジェクトオリジナル。
- `public static Object Resources.Load`パス;

Examples

き

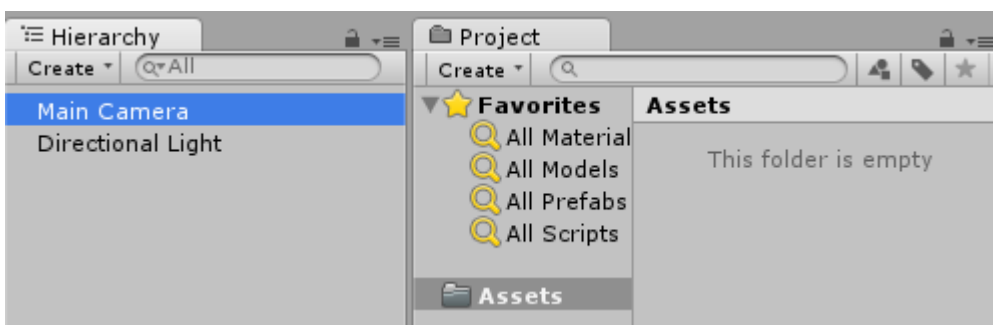
プレハブは、コンポーネント、プロパティ、アタッチされたコンポーネント、およびシリアルされたプロパティとともにな `GameObject` のをにするアセットタイプです。これがいつなシナリオはたくさんあります。

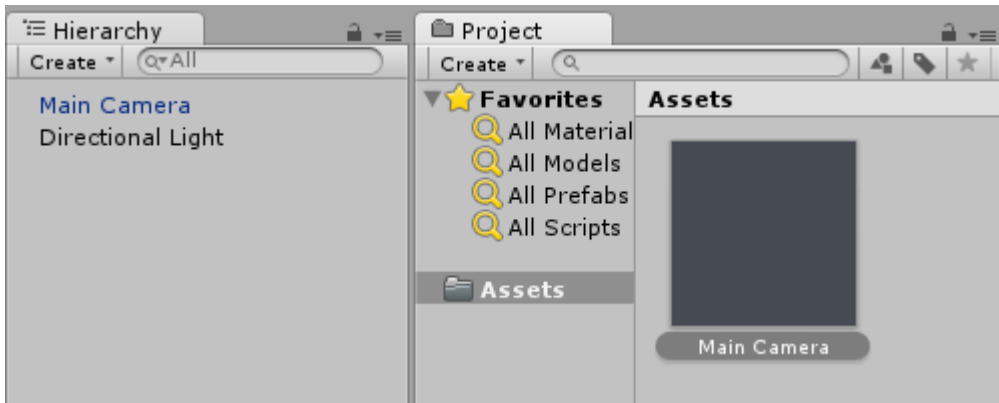
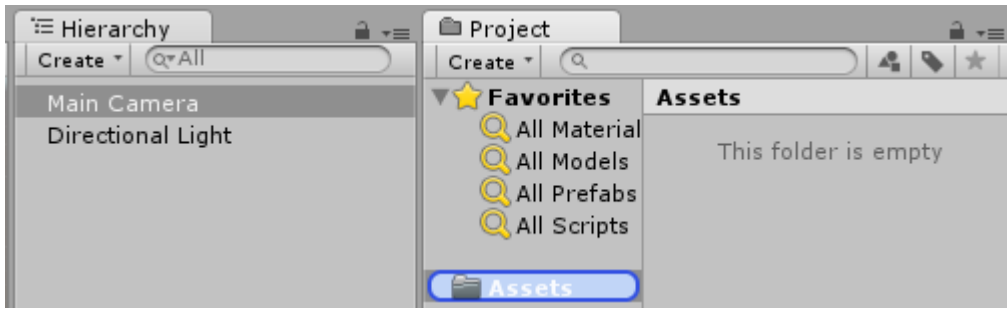
- シーンオブジェクトをする
- のシーンでのオブジェクトをする
- プレハブをし、そののをオブジェクト/シーンにできること
- のを1つのプレハブからにしながら、わずかなでオブジェクトをする
- にゲームオブジェクトをインスタンスする

Unityには、「すべてがPrefabsでなければならない」というのがあります。これはおそらくではありますが、コードのと `GameObjects` のをにします。

プレハブの

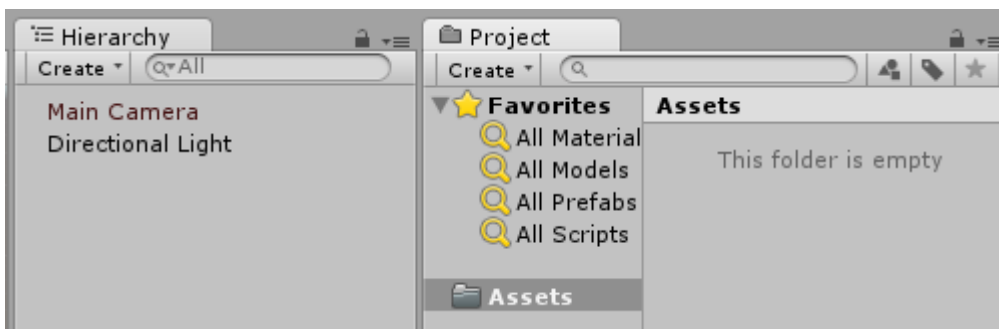
プレハブをするには、シーンから **Assets** フォルダまたはサブフォルダにゲームオブジェクトをドラッグします。





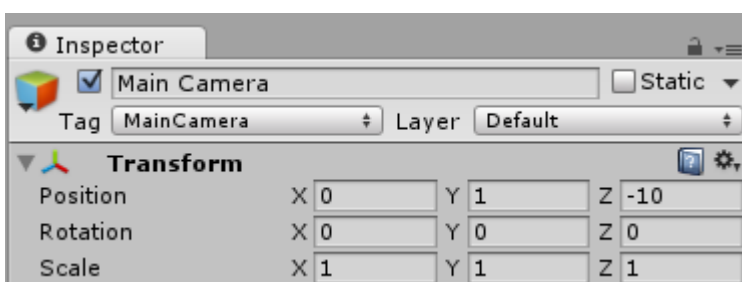
ゲームオブジェクトはにわり、プレハブにされていることをします。
このオブジェクトは、クラスのオブジェクトインスタンスとに、プレハブインスタンスです。

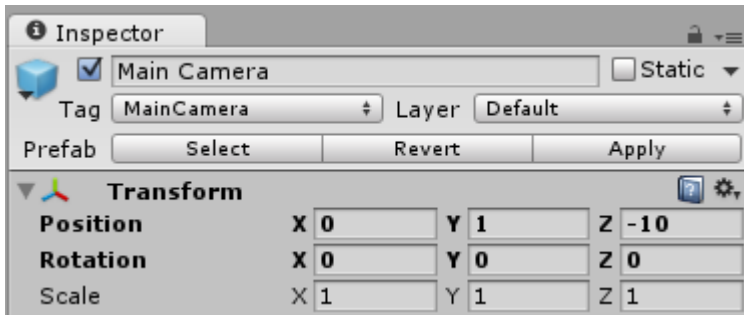
プレハブは、インスタンスにすることができます。その、にされたゲームオブジェクトのがにわります。



プレハブ

ビューでプレハブをすると、そのインスペクタがのゲームオブジェクトとしようにづくでしよう





このプロパティは、オブジェクトがプレハブとなることを示します。オブジェクトがプレハブに設定されず、インスタンスされたプレハブのプロパティを変更することができます。プレハブのインスタンスで設定されると、元のプレハブの値は、そのインスタンスには変更されません。

[Apply] ボタンをクリックすると、オブジェクトがプレハブになります。また、インスタンスされたオブジェクトがプレハブになります。さらに、元のオブジェクトには、そのオブジェクトをクリックして、[Prefab] ボタンをクリックして、「Prefab」を選択します。コンポーネントをプレハブにするには、コンポーネントをクリックし、[Prefab] ボタンをクリックします。

[Apply] ボタンをクリックすると、プレハブのプロパティがゲームオブジェクトのプロパティで置き換わります。「Prefab」ボタンやダイアログはありませんので、このボタンはご注意ください。

Prefab ボタンは、プロジェクトのフォルダからプレハブを選択します。

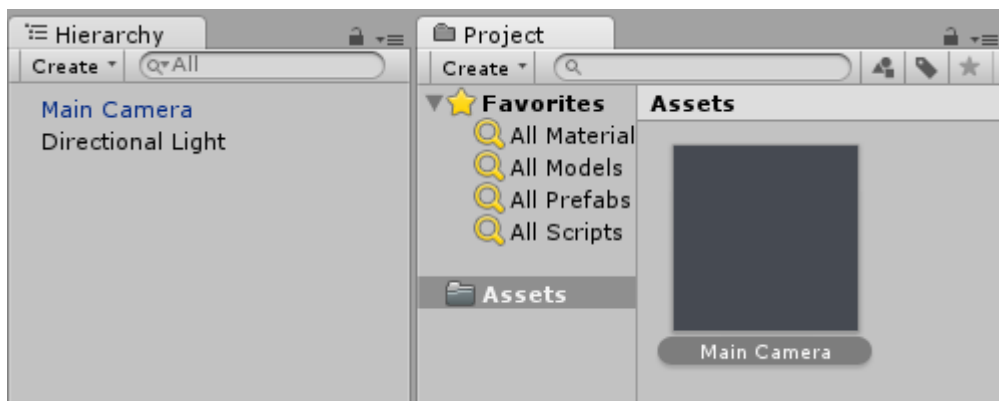
プレハブのインスタンス

プレハブをインスタンスするには、以下の2つの方法があります。

プレハブのインスタンス

デザインビューでプレハブをインスタンスすると、ゲームオブジェクトのプレハブのインスタンスを作成することができます。たとえば、ゲームのレベルを作成するときツリーを作成するなど。

- プレハブをプレハブビューからインスタンスするには、プロジェクトビューからシーンにドラッグします。



- エディタでプレハブをプレハブビューからインスタンスするには、PrefabUtility.InstantiatePrefab() メソッドをプログラムで呼び出すプレハブをインスタンスすることもできます。

```
GameObject gameObject =
    (GameObject)PrefabUtility.InstantiatePrefab(AssetDatabase.LoadAssetAtPath("Assets/MainCamera.prefab",
    typeof(GameObject)));
```

ランタイムインスタンス

にプレハブをインスタンス化すると、ロジックにじてオブジェクトのインスタンスをするの
にですたとえば、5ごとにをするなど。

プレハブをインスタンスするには、プレハブオブジェクトへののがです。これはっていることによ
ってうことができるpublic GameObject あなたのフィールドをMonoBehaviour スクリプトユニティ・エ
ディタでインスペクタをしてそのをします

```
public class SomeScript : MonoBehaviour {
    public GameObject prefab;
}
```

またはでプレハブをくことによって、リソースのフォルダやResources.Load

```
GameObject prefab = Resources.Load("Assets/Resources/MainCamera");
```

プレハブオブジェクトへののをしたら、コードののにInstantiate をしてInstantiate することができ
ますたとえば、ループでのオブジェクトをする。

```
GameObject gameObject = Instantiate<GameObject>(prefab, new Vector3(0,0,0),
Quaternion.identity);
```

プレハブはにしません。

ネストされたプレハブ

れになったプレハブは、でUnityではできません。あるプレハブをのプレハブにドラッグしてする
ことはできますが、そのプレハブのはネストされたプレハブにはされません。

しかし、ながあります - あなたはのプレハブになスクリプトをしなければなりません。それはの
インスタンスをします。

```
using UnityEngine;

public class ParentPrefab : MonoBehaviour {

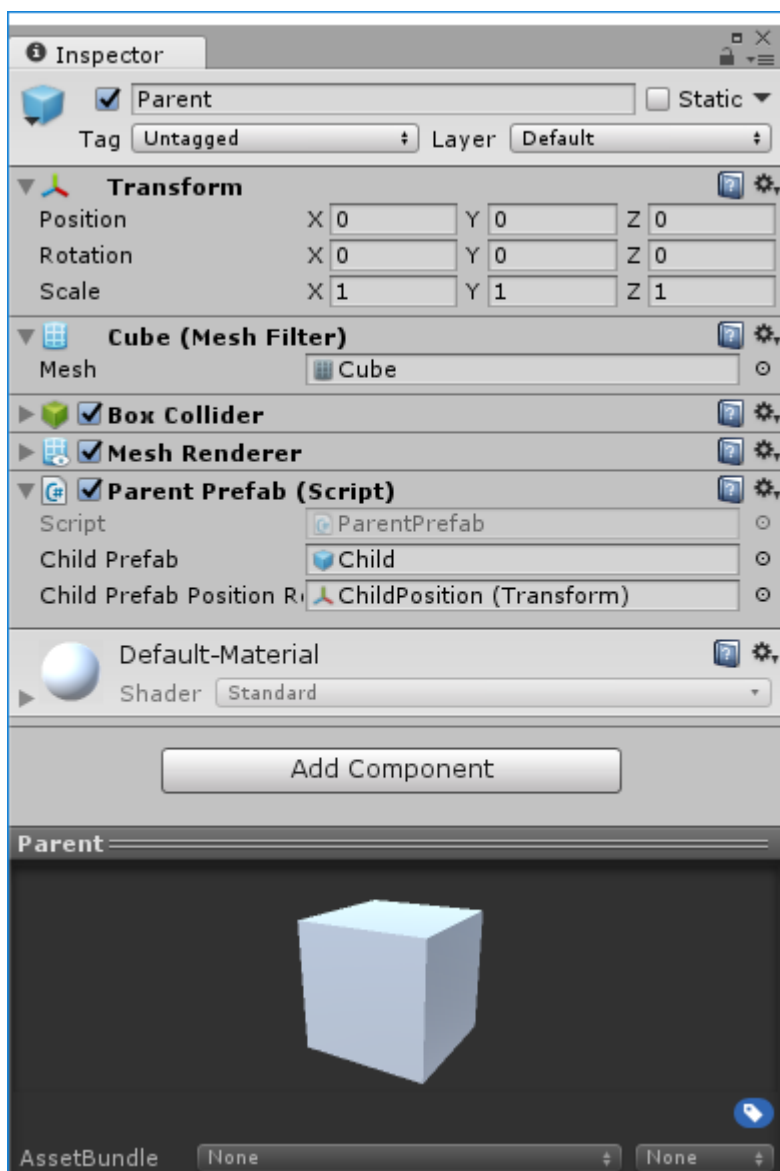
    [SerializeField] GameObject childPrefab;
    [SerializeField] Transform childPrefabPositionReference;

    // Use this for initialization
    void Start () {
        print("Hello, I'm a parent prefab!");
        Instantiate(
```

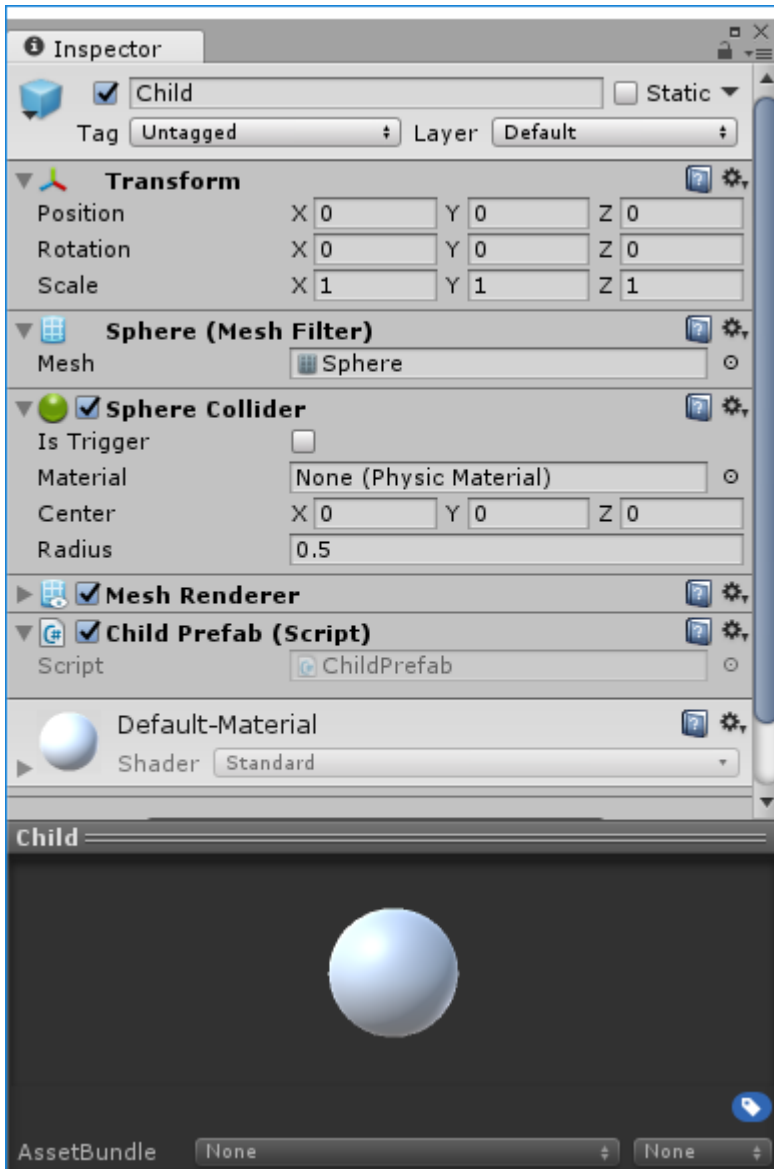


```
childPrefab,  
childPrefabPositionReference.position,  
childPrefabPositionReference.rotation,  
gameObject.transform  
);  
}  
}
```

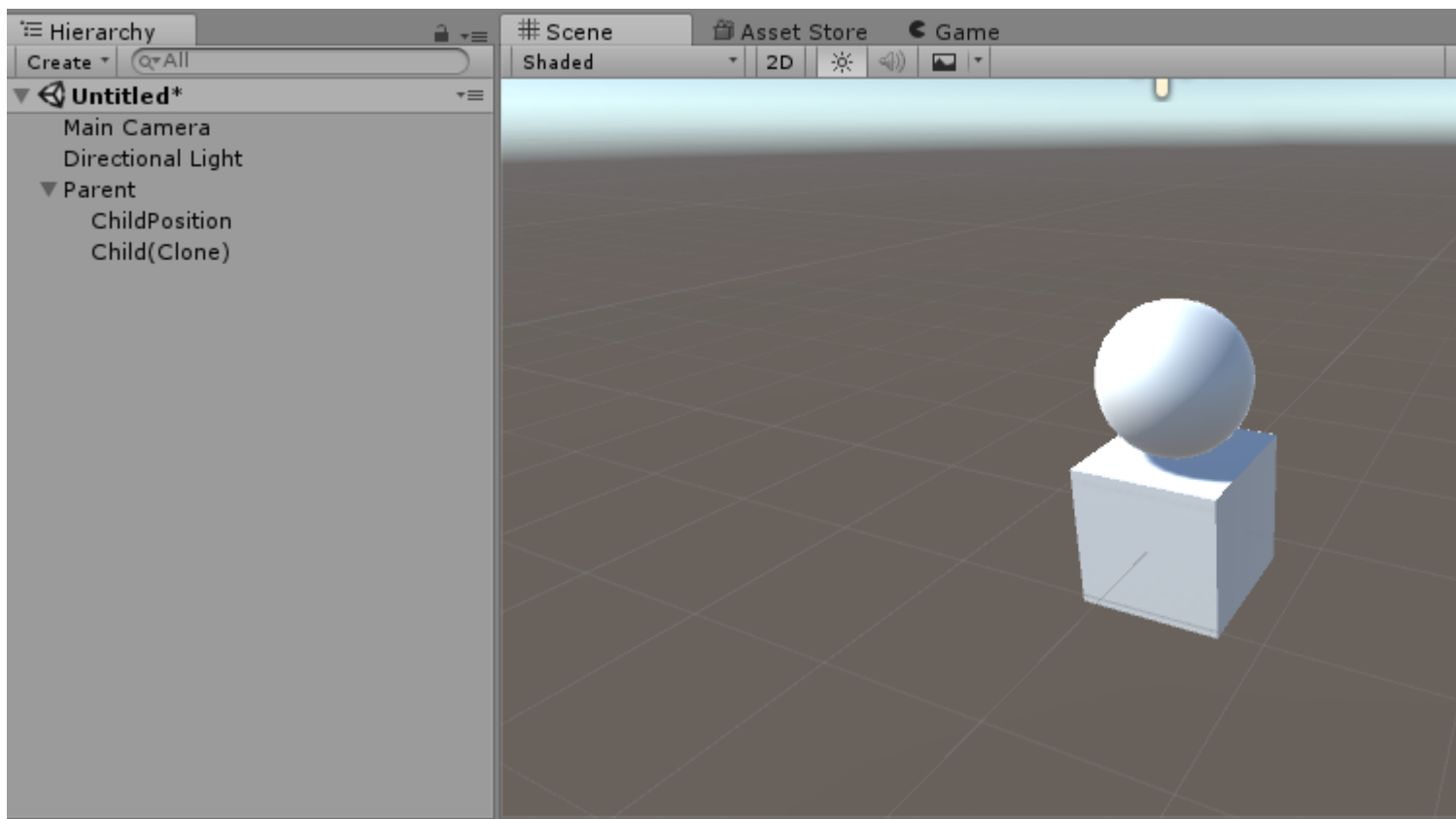
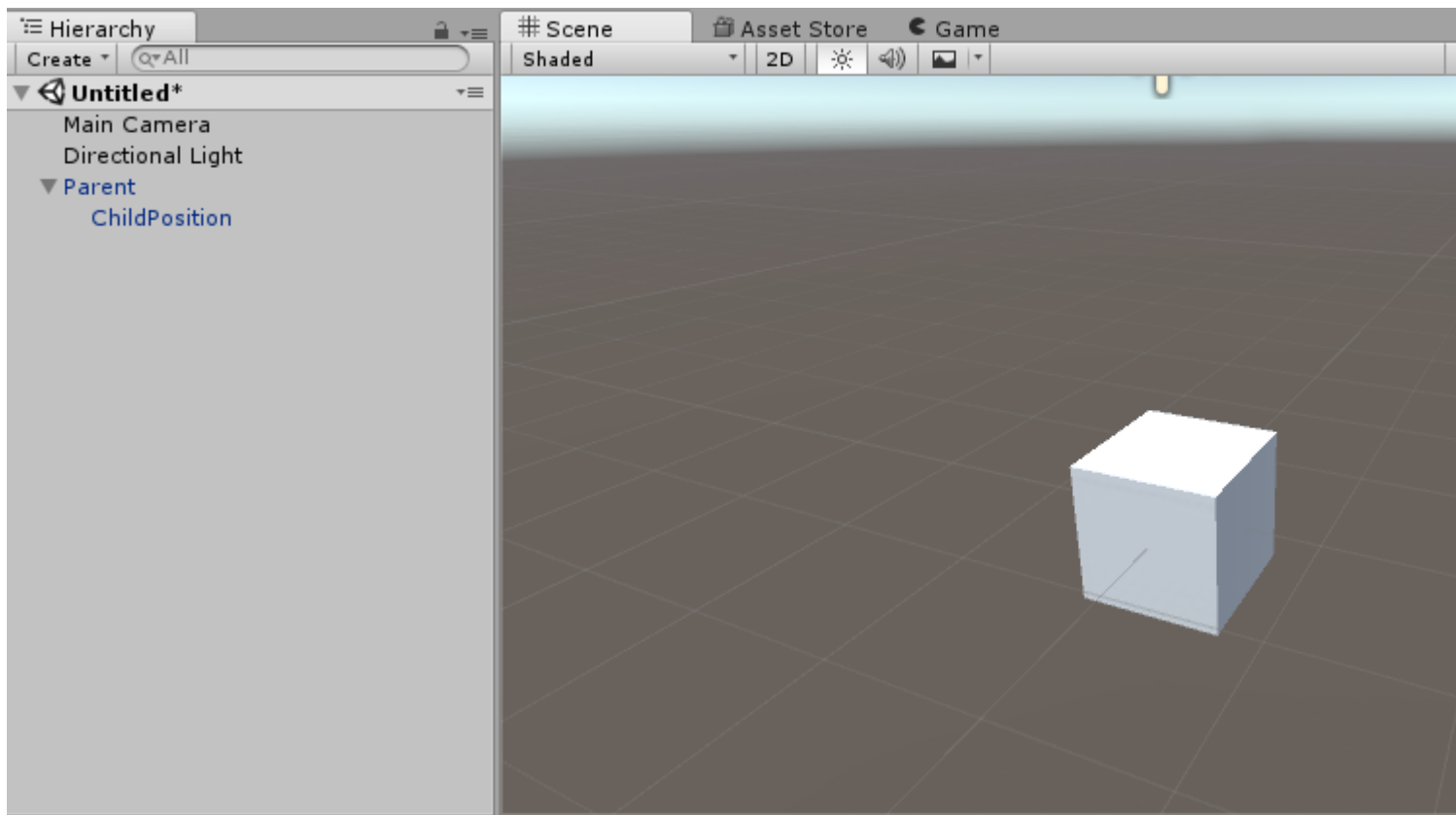
プレハブ



のプレハブ



のシーン



オンラインでプレハブをむ <https://riptutorial.com/ja/unity3d/topic/2133/プレハブ>

24: マルチプラットフォーム

Examples

コンパイラの

コンパイラはプラットフォームのコードをします。それらをする、さまざまなプラットフォームでできないことをすることができます。

- リンゴデバイスのゲームセンターのとAndroidデバイスのGoogleのをトリガーします。
- メニューのアイコンをするWindowsではウィンドウロゴ、LinuxではLinuxペンギン。
- おそらく、プラットフォームにしてプラットフォームのメカニックがあるかもしれません。
- さらにくの...

```
void Update(){  
  
#if UNITY_IPHONE  
    //code here is only called when running on iPhone  
#endif  
  
#if UNITY_STANDALONE_WIN && !UNITY_EDITOR  
    //code here is only ran in a unity game running on windows outside of the editor  
#endif  
  
//other code that will be ran regardless of platform  
  
}
```

Unityコンパイラの名リストは、[ここでつけることができます](#)

プラットフォームのメソッドをクラスにする

クラスは、スクリプトのコアロジックをプラットフォームのメソッドからするきれいなをします。

なクラスとメソッドには、キーワード `partial` がいています。これはコンパイラにクラスを "オープン"のままにし、のファイルをりののためにべるようにします。

```
// ExampleClass.cs  
using UnityEngine;  
  
public partial class ExampleClass : MonoBehaviour  
{  
    partial void PlatformSpecificMethod();  
  
    void OnEnable()  
    {  
        PlatformSpecificMethod();  
    }  
}
```

```
}
```

これでメソッドをするプラットフォームのスキプトのファイルができます。メソッドは、パラメータをつことができますまた、 `ref` が、 `void` があります。

```
// ExampleClass.Iphone.cs

#if UNITY_IPHONE
using UnityEngine;

public partial class ExampleClass
{
    partial void PlatformSpecificMethod()
    {
        Debug.Log("I am an iPhone");
    }
}
#endif
```

```
// ExampleClass.Android.cs

#if UNITY_ANDROID
using UnityEngine;

public partial class ExampleClass
{
    partial void PlatformSpecificMethod()
    {
        Debug.Log("I am an Android");
    }
}
#endif
```

なメソッドがされていない、コンパイラはびしをします。

ヒントこのパターンは、エディターのメソッドをするにもです。

オンラインでマルチプラットフォームをむ <https://riptutorial.com/ja/unity3d/topic/4816/マルチプラットフォーム>

25: モバイルプラットフォーム

- `public static int Input.touchCount`
- パブリックTouch `Input.GetTouch(int index)`

Examples

タッチの

Unityのタッチをするには、`Input.GetTouch()`をしてインデックスをすだけです。

```
using UnityEngine;
using System.Collections;

public class TouchExample : MonoBehaviour {
    void Update() {
        if (Input.touchCount > 0 && Input.GetTouch(0).phase == TouchPhase.Began)
        {
            //Do Stuff
        }
    }
}
```

または

```
using UnityEngine;
using System.Collections;

public class TouchExample : MonoBehaviour {
    void Update() {
        for(int i = 0; i < Input.touchCount; i++)
        {
            if (Input.GetTouch(i).phase == TouchPhase.Began)
            {
                //Do Stuff
            }
        }
    }
}
```

これらのは、のゲームフレームのタッチをします。

TouchPhase

TouchPhaseのには、5のTouchPhase

- まった - がにれた
- - でがいた
-

- はにありますが、いていません
- - をからちげた
- キャンセル - システムがタッチのトラッキングをキャンセルしました

たとえば、オブジェクトをするには、このスクリプトはタッチについてにりけられます。

```
public class TouchMoveExample : MonoBehaviour
{
    public float speed = 0.1f;

    void Update () {
        if(Input.touchCount > 0 && Input.GetTouch(0).phase == TouchPhase.Moved)
        {
            Vector2 touchDeltaPosition = Input.GetTouch(0).deltaPosition;
            transform.Translate(-touchDeltaPosition.x * speed, -touchDeltaPosition.y * speed,
0);
        }
    }
}
```

オンラインでモバイルプラットフォームをむ <https://riptutorial.com/ja/unity3d/topic/6285/モバイルプラットフォーム>

26: ユーザーインターフェイスシステムUI

Examples

コードでイベントをする

デフォルトでは、インスペクタをしてイベントをするがありますが、コードでうがよいもあります。このでは、それをするためにボタンのclickイベントにします。

```
using UnityEngine;
using UnityEngine.UI;

[RequireComponent(typeof(Button))]
public class AutomaticClickHandler : MonoBehaviour
{
    private void Awake()
    {
        var button = this.GetComponent<Button>();
        button.onClick.AddListener(HandleClick);
    }

    private void HandleClick()
    {
        Debug.Log("AutomaticClickHandler.HandleClick()", this);
    }
}
```

UIコンポーネントは、メインのリスナーをにします。

- Button [onClick](#)
- ドロップダウン [onValueChanged](#)
- InputField [onEndEdit](#)、[onValidateInput](#)、[onValueChanged](#)
- スクロールバー [onValueChanged](#)
- ScrollRect [onValueChanged](#)
- スライダ [onValueChanged](#)
- トグル [onValueChanged](#)

マウスリスナーの

によっては、コンポーネントによってネイティブにされていないのイベント、にマウスイベントにリスナーをすることがあります。これをうには、 `EventTrigger` コンポーネントをしてでするがあります

```
using UnityEngine;
using UnityEngine.EventSystems;

[RequireComponent(typeof(EventTrigger))]
public class CustomListenersExample : MonoBehaviour
{
```



```

void Start( )
{
    EventTrigger eventTrigger = GetComponent<EventTrigger>( );
    EventTrigger.Entry entry = new EventTrigger.Entry( );
    entry.eventID = EventTriggerType.PointerDown;
    entry.callback.AddListener( ( data ) => { OnPointerDownDelegate(
(PointerEventData)data ); } );
    eventTrigger.triggers.Add( entry );
}

public void OnPointerDownDelegate( PointerEventData data )
{
    Debug.Log( "OnPointerDownDelegate called." );
}
}

```

さまざまなイベントIDがです

- PointerEnter
- PointerExit
- PointerDown
- PointerUp
- PointerClick
- ドラッグ
- ドロップ
- スクロール
- UpdateSelected
-
- する
-
- InitializePotentialDrag
- BeginDrag
- EndDrag
- する
- キャンセル

オンラインでユーザーインターフェイスシステムUIをむ

<https://riptutorial.com/ja/unity3d/topic/2296/ユーザーインターフェイスシステム-ui->

27: ユニティアニメーション

Examples

のためのアニメーション

このコードは、Unityのアニメーションのなをしています。

ここでは、2つのアニメーションクリップがです。ランとアイドル。これらのアニメーションは、Stand-In-Placeモーションでなければなりません。アニメーションクリップをしたら、Animator Controllerをします。このコントローラーを、アニメートするプレイヤーまたはゲームオブジェクトにします。

WindowsオプションからAnimatorウィンドウをきます。2つのアニメーションクリップをAnimatorウィンドウにドラッグすると、2つのがされます。したら、のパラメータタブをして、2つのパラメータをとともboolとしてします。「PerformRun」と「PerformIdle」とをけてください。「PerformIdle」をtrueにします。

アイドルからとにしてアイドルにします。Idle-> Run transitionをクリックし、InspectorウィンドウでHasExitのをします。のにもじことをします。アイドル→ランの、をしますPerformIdle。Run-> Idleの、PerformRunをします。のCスクリプトをゲームオブジェクトにします。のボタンをってアニメーションをし、とのボタンでさせるがあります。

```
using UnityEngine;
using System.Collections;

public class RootMotion : MonoBehaviour {

    //Public Variables
    [Header("Transform Variables")]
    public float RunSpeed = 0.1f;
    public float TurnSpeed = 6.0f;

    Animator animator;

    void Start ()
    {
        /**
         * Initialize the animator that is attached on the current game object i.e. on which you
         will attach this script.
         */
        animator = GetComponent<Animator>();
    }

    void Update()
    {
        /**
         * The Update() function will get the bool parameters from the animator state machine and
         set the values provided by the user.
        */
    }
}
```

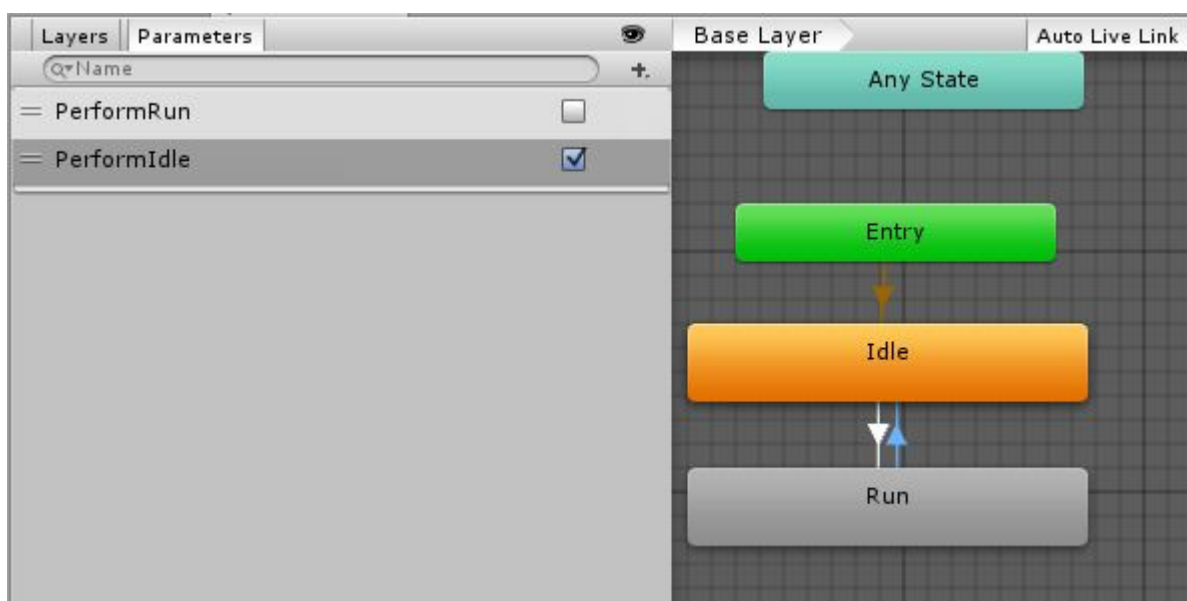
```

    * Here, I have only added animation for Run and Idle. When the Up key is pressed, Run
    animation is played. When we let go, Idle is played.
    */

    if (Input.GetKey (KeyCode.UpArrow)) {
        animator.SetBool ("PerformRun", true);
        animator.SetBool ("PerformIdle", false);
    } else {
        animator.SetBool ("PerformRun", false);
        animator.SetBool ("PerformIdle", true);
    }
}

void OnAnimatorMove()
{
    /**
    * OnAnimatorMove() function will shadow the "Apply Root Motion" on the animator. Your
    game objects position will now be determined
    * using this fuction.
    */
    if (Input.GetKey (KeyCode.UpArrow)){
        transform.Translate (Vector3.forward * RunSpeed);
        if (Input.GetKey (KeyCode.RightArrow)) {
            transform.Rotate (Vector3.up * Time.deltaTime * TurnSpeed);
        }
        else if (Input.GetKey (KeyCode.LeftArrow)) {
            transform.Rotate (-Vector3.up * Time.deltaTime * TurnSpeed);
        }
    }
}
}
}
}

```

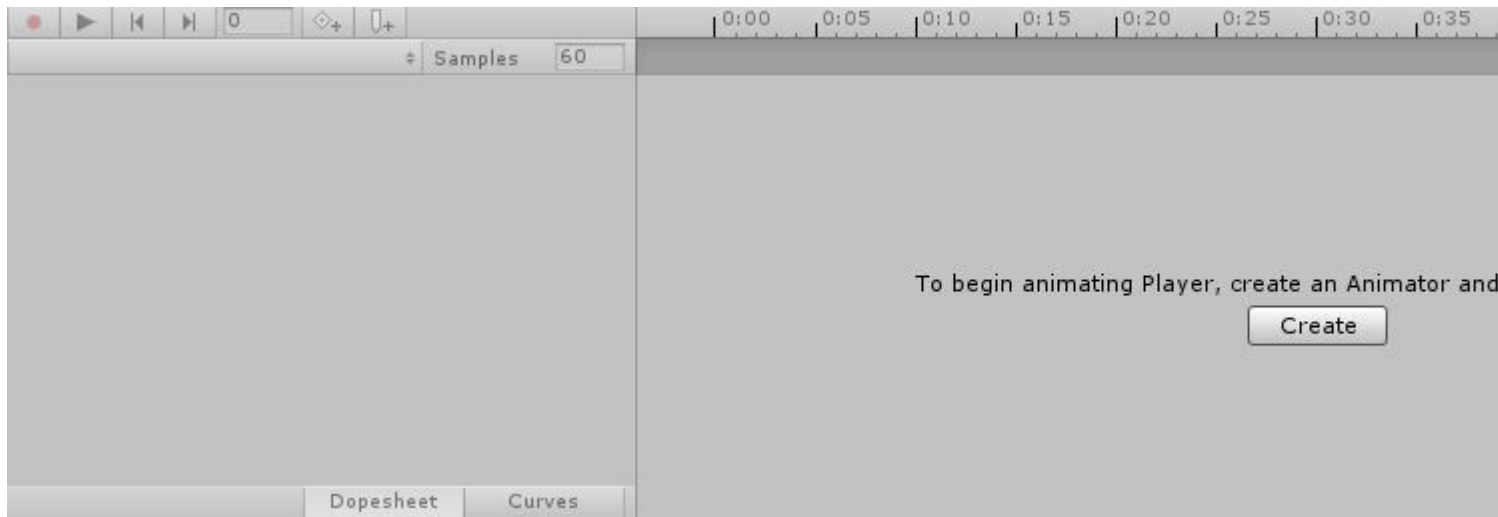


アニメーションクリップのと

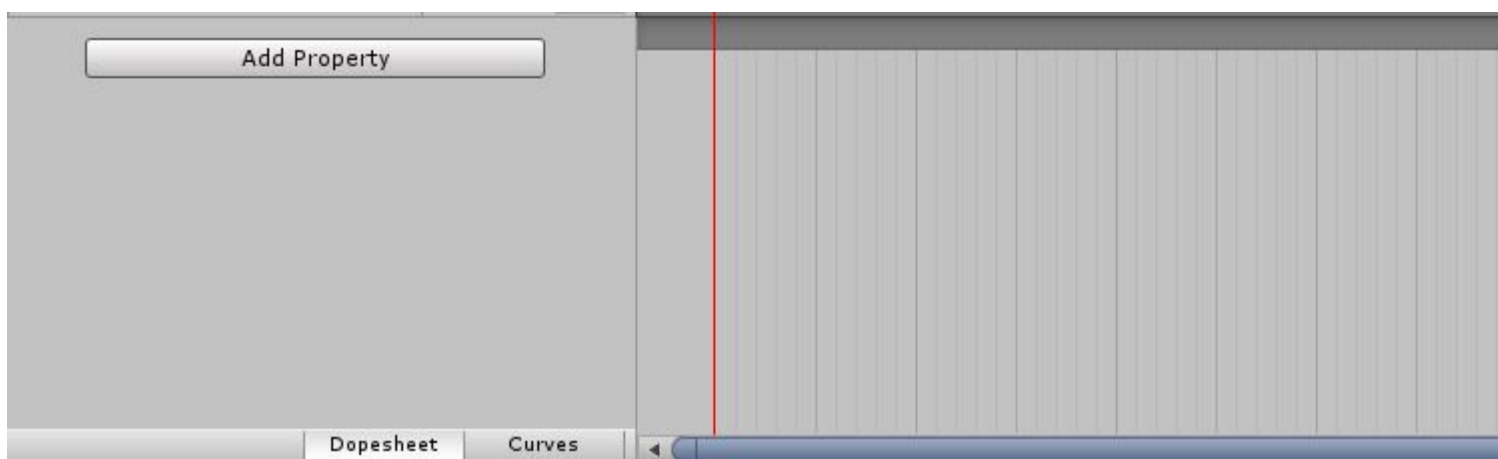
ここでは、ゲームオブジェクトまたはプレーヤのアニメーションクリップをしてするをします。

このでされるモデルは、Unity Asset Storeからダウンロードされます。プレイヤーはこのリンクからダウンロードされました <https://www.assetstore.unity3d.com/en/#!/content/21874>

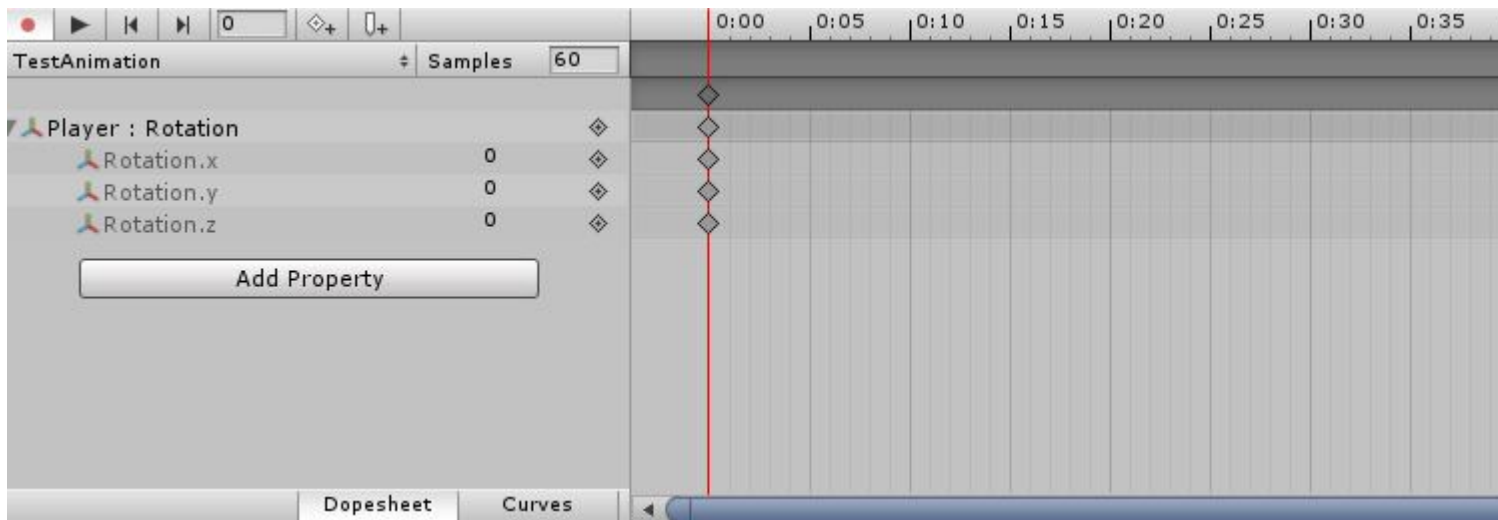
アニメーションをするには、まずアニメーションウィンドウを開きます。ウィンドウをクリックしてアニメーションをするか、Ctrl + 6をしてることができます。ウィンドウから、アニメーションクリップをするゲームオブジェクトをし、アニメーションウィンドウの「」ボタンをクリックします。



アニメーションにつけIdlePlayer、SprintPlayer、DyingPlayerなど、します。アニメーションウィンドウから、プロパティのボタンをクリックします。これにより、にしてゲームオブジェクトまたはプレイヤーのプロパティをすることができます。これには、、、スケールなどのトランジション、ゲームオブジェクトにするanyotherプロパティCollider、Mesh Rendererなどがまれます。



ゲームオブジェクトのアニメーションをするには、ヒューマノイド3Dモデルがです。このリンクからモデルをダウンロードできます。のにつて、しいアニメーションをします。 Transformプロパティをし、のいずれかのRotationをします。



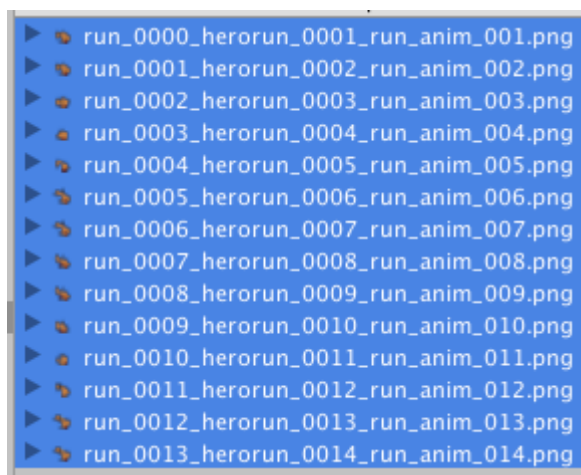
ここで、ゲームオブジェクトプロパティの[]ボタンと[]のはにわっていました。ドロップダウンをクリックすると、X、Y、Zのがされます。デフォルトのアニメーションは1にされています。アニメーションはキーフレームをしてをします。アニメートするには、なるでキーをし、インスペクタウィンドウからをします。たとえば、0.0sでののは0.0とすることができます。0.5sでは、Xのは20.0になります。1.0sでは、は0.0になります。1.0でアニメーションをできます。

アニメーションのさは、アニメーションににしたキーによってなります。をスムーズにするために、さらにキーをすることができます。

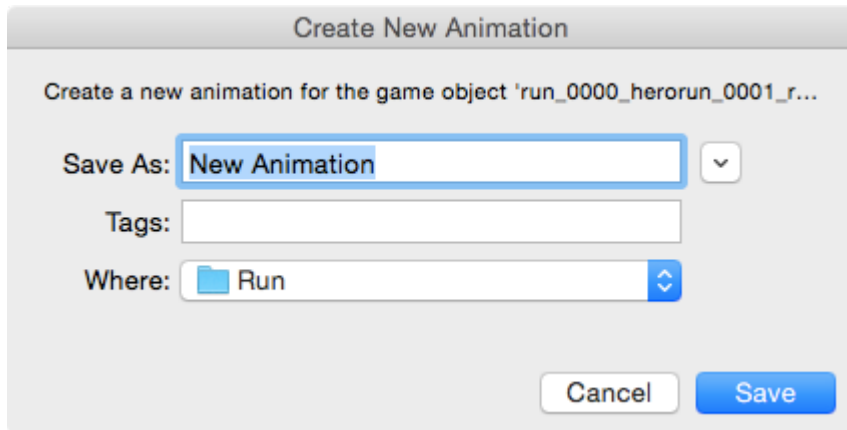
2D スプライトアニメーション

スプライトアニメーションは、イメージまたはフレームののシーケンスをすることからります。

にのをアセットフォルダにインポートします。いくつかのをからするか、Asset Storeからをダウンロードしてください。ここでは、[このフリーアセット](#)をしています。

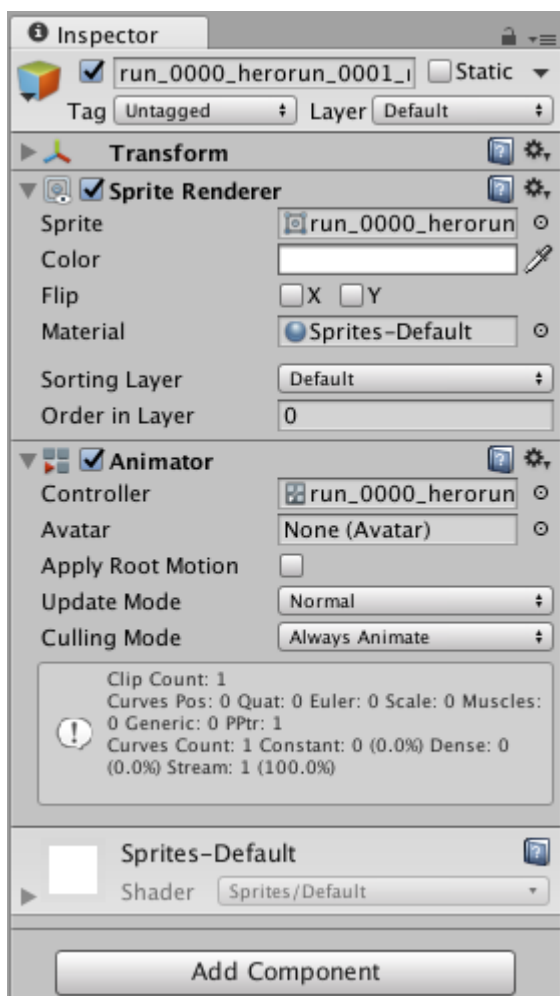


1つのアニメーションのすべてののをアセットフォルダからシーンビューにドラッグします。しいアニメーションクリップにをけるダイアログがされます。

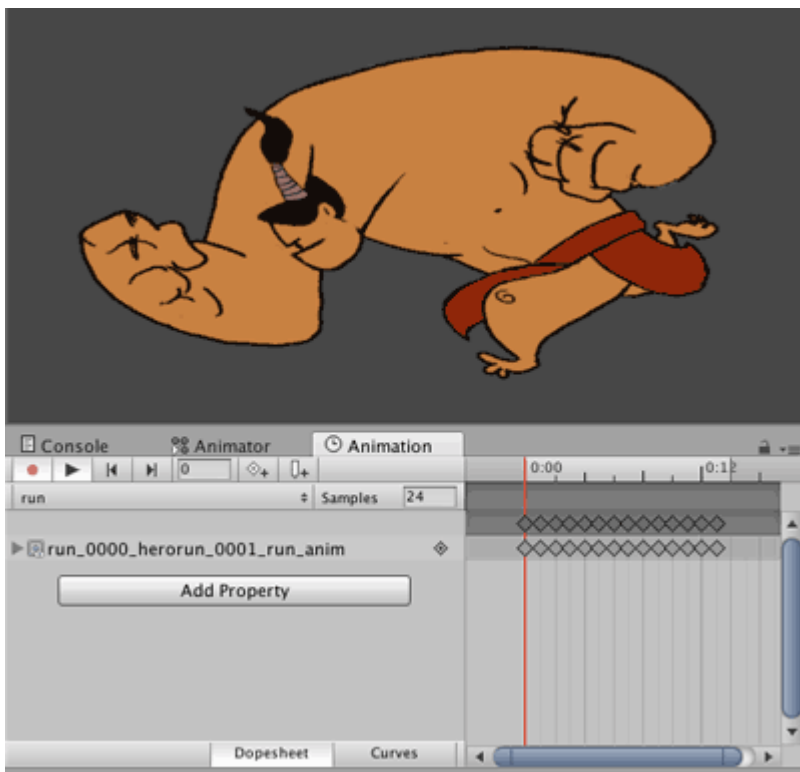


これはなショートカットです

- 新しいゲームオブジェクトを作る
- 2つのコンポーネントスプライトレンダラーとアニメーターをりて、
- アニメーションコントローラを作るそして新しいAnimatorコンポーネントをそれらにリンクする
- したフレームでアニメーションクリップを作る



をクリックして、アニメーションタブでをプレビューします。



じメソッドをして、じゲームオブジェクトのしいアニメーションをし、しいゲームオブジェクトとアニメーションコントローラをすることができます。しいアニメーションクリップを3Dアニメーションとじてそのオブジェクトのアニメーションコントローラにします。

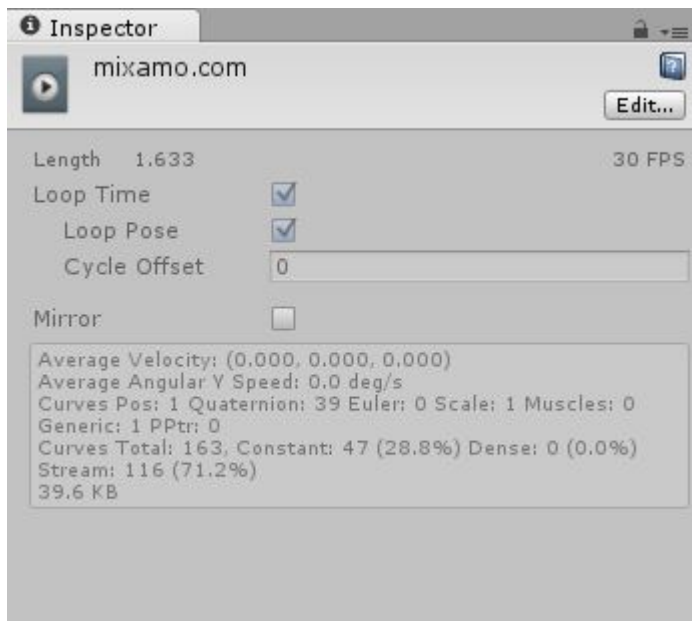
アニメーション

アニメーションカーブでは、アニメーションのにパラメータをできます。たとえば、さ60のアニメーションがあり、パラメータがなは、アニメーション= 0.0;アニメーション= 30.0; X = 1.0、アニメーション= 60.0; X = 0.0。

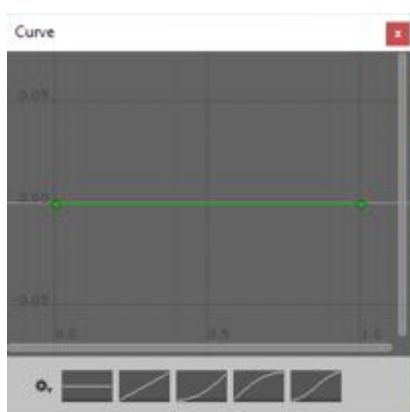
をしたら、のでそれを、、、またはすることができます。

のでは、プレーヤーのゲームオブジェクトがであることをします。のためのアニメーションがされるとき、アニメーションがむにつれてプレーヤーのがするはずでず。アニメーションがすると、がするはずでず。

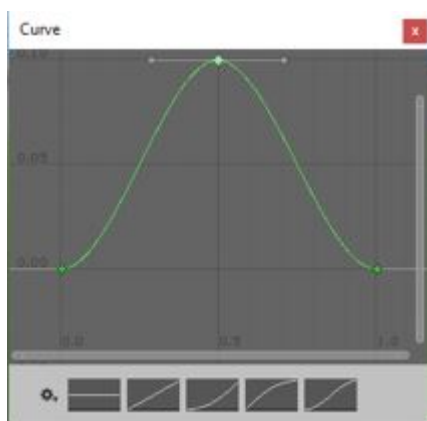
はこのアニメーションクリップをしました。クリップをし、インスペクタウィンドウで[]をクリックします。



そこまでスクロールしてにします。カーブをするには、+をクリックします。のを ForwardRunCurveとします。のミニチュアカーブをクリックします。デフォルトのカーブをつさなウィンドウがきます。



たちは、それがきてからちるのカーブがです。デフォルトでは、に2つのがあります。カーブをダブルクリックすると、ポイントをできます。をドラッグして、のようなをします。



Animatorウィンドウで、のクリップをします。また、とじのfloatパラメータ、つまり ForwardRunCurveをします。

アニメーションがされると、はカーブによってします。のコードは、floatのをします。

```
using UnityEngine;
using System.Collections;

public class RunAnimation : MonoBehaviour {

    Animator animator;
    float curveValue;

    void Start ()
    {
        animator = GetComponent<Animator> ();
    }

    void Update ()
    {
        curveValue = animator.GetFloat ("ForwardRunCurve");

        transform.Translate (Vector3.forward * curveValue);
    }

}
```

`curveValue`は、のにおけるForwardRunCurveのをします。たちはそのをってのスピードをえています。このスクリプトをプレイヤーのゲームオブジェクトにすることができます。

オンラインでユニティアニメーションをむ <https://riptutorial.com/ja/unity3d/topic/5448/ユニティアニメーション>

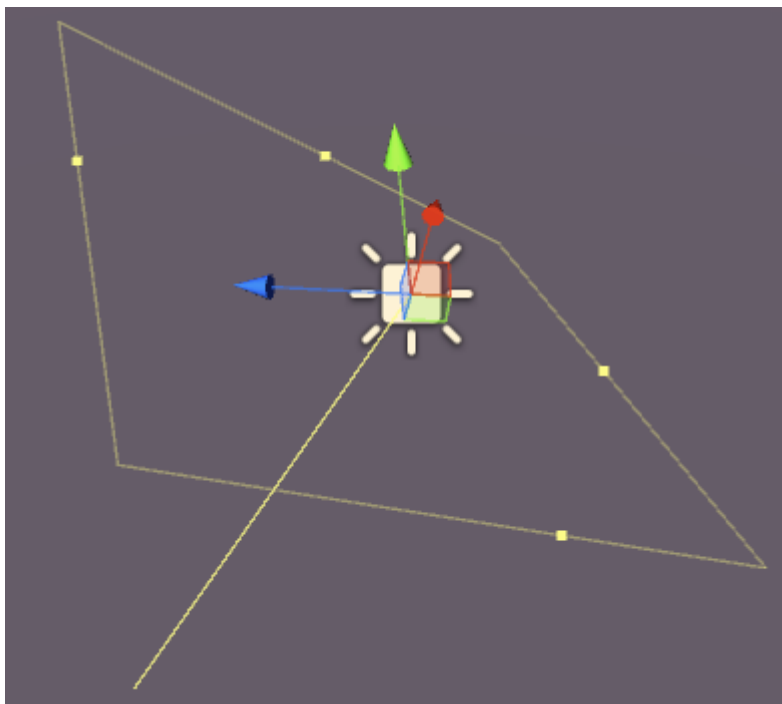
28: ユニティライティング

Examples

の

エリアライト

のをってがされる。らはあなたがシーンをくまでをることができないことをするだけかされている。

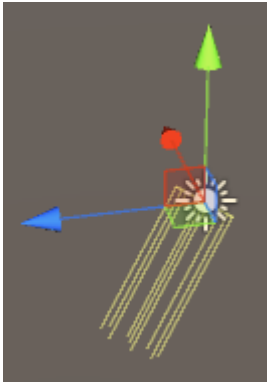


エリアライトには、のプロパティがあります。

- - ライトの。
- さ - ライトエリアのさ。
- - ライトのをりてます。
- **Intensity** - のさは**08**です。
- バウンス - なのさは**08**です。
- ドローハロー - のりにハローをく。
- **Flare** - ライトにフレアをりてることができます。
- レンダリングモード - 、 、 ではありません。
- **Culling Mask** - シーンのをにすることができます。

ライト

ライトは、ののようによをします。のGameObjectがどこにかれていても、が「どこでも」どこにあるかはあります。のさは、のののよようにしません。

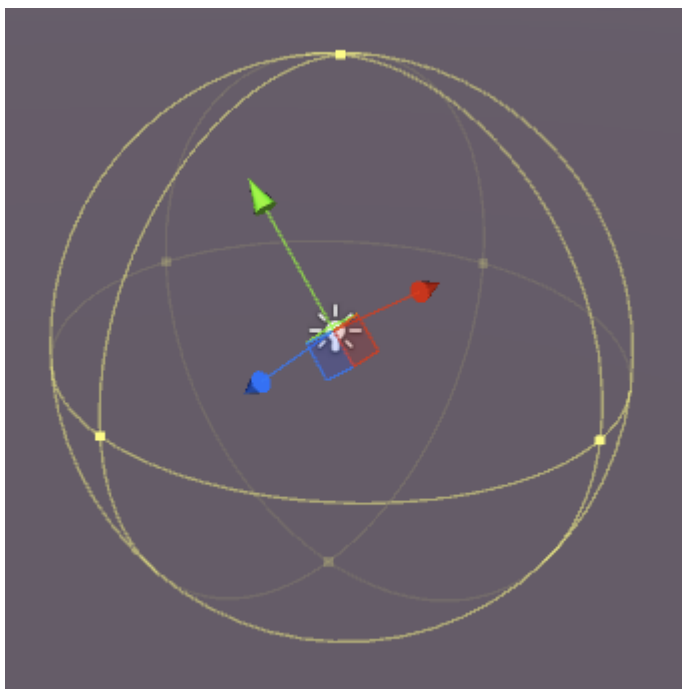


Directional Lightには、のプロパティがあります。

- ベーキング - リアルタイム、ベーキングまたは。
- - ライトのをりてます。
- **Intensity** - のさは**08**です。
- バウンス - なのさは**08**です。
- シャドウタイプ - シャドウ、ハードシャドウまたはソフトシャドウなし。
- クッキー - ライトにクッキーをりてることができます。
- **Cookie Size** - りてられたCookieのサイズ。
- ドローハロー - のりにハローをく。
- **Flare** - ライトにフレアをりてることができます。
- レンダリングモード - 、 、 ではありません。
- **Culling Mask** - シーンのをにすることができます。

ポイントライト

ポイントライトは、あらゆるののからをします。かられるほど、のはくなります。

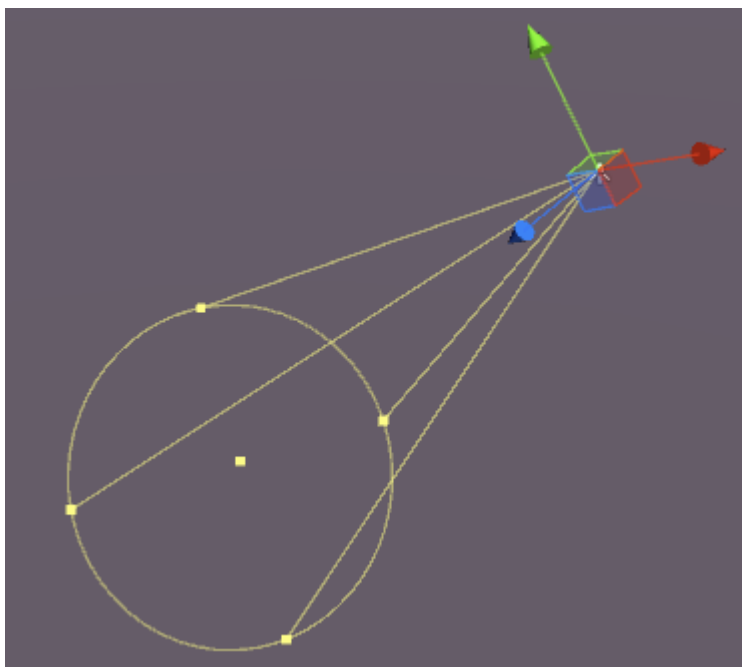


ポイントライトにはのがあります

- ベーキング - リアルタイム、ベーキングまたは。
- - がもはやしないからの。
- - ライトのをりてます。
- **Intensity** - のさは**08**です。
- バウンス - なのさは**08**です。
- シャドウタイプ - シャドウ、ハードシャドウまたはソフトシャドウなし。
- クッキー - ライトにクッキーをりてることができます。
- ドローハロー - のりにハローをく。
- **Flare** - ライトにフレアをりてることができます。
- レンダリングモード - 、 、 ではありません。
- **Culling Mask** - シーンのをにすることができます。

スポットライト

スポットライトはポイントライトによくていますが、はにされています。は、の"コーン"で、のヘッドライトやサーチライトにです。



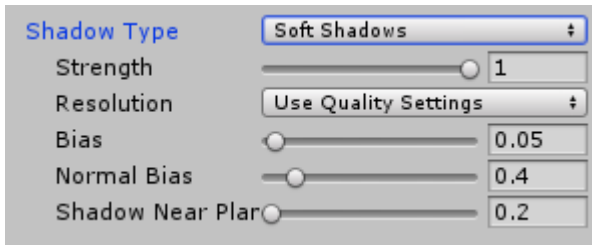
スポットライトにはのがあります

- ベーキング - リアルタイム、ベーキングまたは。
- - がもはやしないからの。
- **Spot Angle** - の。
- - ライトのをりてます。
- **Intensity** - のさは**08**です。
- バウンス - なのさは**08**です。
- シャドウタイプ - シャドウ、ハードシャドウまたはソフトシャドウなし。
- クッキー - ライトにクッキーをりてることができます。
- ドローハロー - のりにハローをく。
- **Flare** - ライトにフレアをりてることができます。
- レンダリングモード - 、 、 ではありません。
- **Culling Mask** - シーンのをにすることができます。

シャドウにする

ハードまたはソフトシャドウをすると、インスペクタでのオプションがになります。

- さ - が0から1までどのくらいですか
- - シャドウの。
- バイアス - シャドウキャストリングのがからざかるい。
- ノーマルバイアス - シャドウキャストサーフェスがノーマルにってにしまれるい。
- のくの - 0.1 - 10。



は、またはむしろがをするときである。シェーダをするオブジェクトのマテリアルのインスペクタパネルには、プロパティがあります。

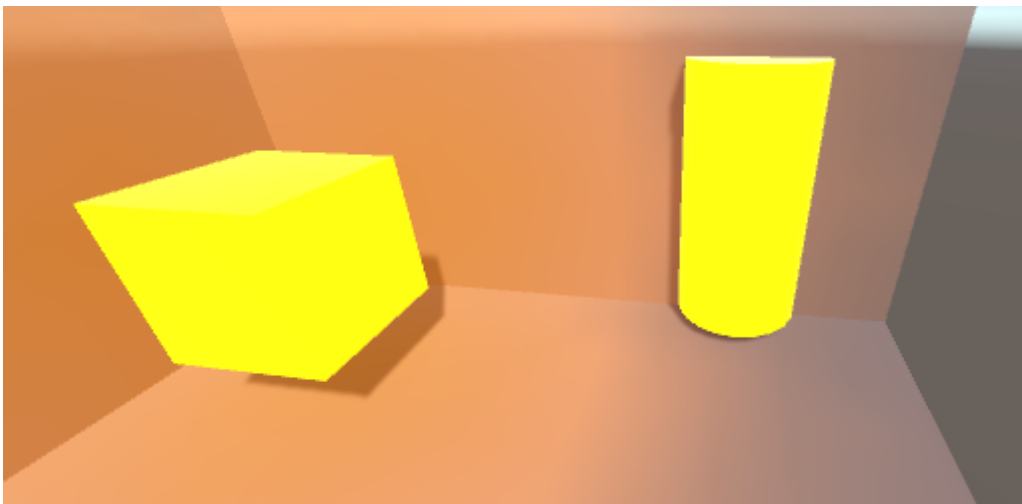


このプロパティをデフォルトの0よりきいにすると、をしたり、マップをそのにりてることができます。このスロットにりてられたテクスチャは、エミッションがのをできるようにします。

グローバルイルミネーションオプションもあります。このオプションをすると、エミッションがくのオブジェクトにベイクされるかどうかをできます。

- ベークド - エミッションはシーンにきけられます
- リアルタイム - エミッションはオブジェクトにします
- なし - エミッションはくのオブジェクトにはしません

オブジェクトがにされていない、エフェクトはオブジェクトを「グロー」にせかけますが、はされません。ここのはですが、はそうではありません



あなたはのようなコードでをすることができます

```
Renderer renderer = GetComponent<Renderer>();  
Material mat = renderer.material;  
mat.SetColor("_EmissionColor", Color.yellow);
```

されるはなでち、シーンのなにしてのみされます。

オンラインでユニティライティングをむ <https://riptutorial.com/ja/unity3d/topic/7884/ユニティライ>

ティング

29: リソース

Examples

ま

`Resources`クラスをすると、シーンのではないアセットをロードすることができます。オンデマンドアセットをするがある、たとえばのオーディオ、テキストなどをローカライズするにはにです。

アセットは、**Resources**というフォルダにするがあります。のリソースフォルダをプロジェクトのにさせることはです。`Resources`クラスはあなたがっているがあるすべての`Resources`フォルダをします。

リソースにされたすべてのアセットは、コードでされていなくてもビルドにまれます。したがって、リソースにアセットをしないでください。

```
//Example of how to load language specific audio from Resources

[RequireComponent(typeof(AudioSource))]
public class loadIntroAudio : MonoBehaviour {
    void Start () {
        string language = Application.systemLanguage.ToString();
        AudioClip ac = Resources.Load(language + "/intro") as AudioClip; //loading intro.mp3
        specific for user's language (note the file file extension should not be used)
        if (ac==null)
        {
            ac = Resources.Load("English/intro") as AudioClip; //fallback to the english
            version for any unsupported language
        }
        transform.GetComponent<AudioSource>().clip = ac;
        transform.GetComponent<AudioSource>().Play();
    }
}
```

リソース101

ま

Unityには、さまざまなにできるいくつかの「なの」フォルダがあります。これらのフォルダの1つは「リソース」とばね、

'Resources'フォルダは、Unityでにをみむ2つのうちの1つですもう1つは[AssetBundlesUnity Docs](#)

'Resources'フォルダはAssetsフォルダのどこにあってもかまいません。また、Resourcesという

のフォルダをつことができます。すべての 'Resources' フォルダのは、コンパイルにマージされます。

Resources フォルダからアセットをロードするなは、 **Resources.Load** をすることです。これはパラメータをります。このパラメータをすると、Resources フォルダにするファイルのパスをできます。アセットをロードするにファイルをするはありません

```
public class ResourcesSample : MonoBehaviour {

    void Start () {
        //The following line will load a TextAsset named 'foobar' which was previously place
        under 'Assets/Resources/Stackoverflow/foobar.txt'
        //Note the absence of the '.txt' extension! This is important!

        var text = Resources.Load<TextAsset>("Stackoverflow/foobar").text;
        Debug.Log(string.Format("The text file had this in it :: {0}", text));
    }
}
```

のオブジェクトでされるオブジェクトも、リソースからロードできます。そのようなオブジェクトは、テクスチャがきけられた3Dモデル、またはのスプライトです。

```
//This example will load a multiple sprite texture from Resources named "A_Multiple_Sprite"
var sprites = Resources.LoadAll("A_Multiple_Sprite") as Sprite[];
```

すべてをにれて

ここでは、すべてのサウンドをすべてのゲームにロードするためにするヘルパークラスをします。これをシーンのGameObjectにすることができ、されたオーディオファイルを 'Resources / Sounds' フォルダからロードします

```
public class SoundManager : MonoBehaviour {

    void Start () {

        //An array of all sounds you want to load
        var filesToLoad = new string[] { "Foo", "Bar" };

        //Loop over the array, attach an Audio source for each sound clip and assign the
        //clip property.
        foreach(var file in filesToLoad) {
            var soundClip = Resources.Load<AudioClip>("Sounds/" + file);
            var audioSource = gameObject.AddComponent<AudioSource>();
            audioSource.clip = soundClip;
        }
    }
}
```

ファイナルノート

1. Unityは、アセットをビルドにみむにはスマートです。シリアルされていないつまり、ビルドにまれているシーンでされているは、ビルドからされます。ただし、これはResourcesフォルダのにはされません。したがって、このフォルダにアセットをするにはにさないください
2. Resources.LoadまたはResources.LoadAllをしてロードされるアセットは、
[Resources.UnloadUnusedAssets](#)または[Resources.UnloadAsset](#)をしてアンロードできます

オンラインでリソースをむ <https://riptutorial.com/ja/unity3d/topic/4070/リソース>

30: レイキャスト

パラメーター

パラメータ	
	におけるの
	の
maxDistance	レイがするかどうかをチェックする
レイヤーマスク	rayをキャストするときにColliderをにするためにされるLayerマスク。
queryTriggerInteraction	このクエリがトリガーにたるをします。

Examples

レイキャスト

これは、シーンのすべてのコライダーにして、さ `maxDistance` `direction` に `origin` からのをキャストします。

これは、 `maxDistance` の `origin` `direction` `maxDistance` 、 `maxDistance` にコライダーがあるかどうかをします。

```
Physics.Raycast(origin, direction, maxDistance);
```

例えば、ゲーム `GameObject` 10ユニットにかある、 `Hello World` がコンソールにされます

```
using UnityEngine;

public class TestPhysicsRaycast: MonoBehaviour
{
    void FixedUpdate()
    {
        Vector3 fwd = transform.TransformDirection(Vector3.forward);

        if (Physics.Raycast(transform.position, fwd, 10))
            print("Hello World");
    }
}
```

Physics2D Raycast2D

レイキャストをして、aiがレベルのからちることなくくことができるかどうかをできます。

```
using UnityEngine;

public class Physics2dRaycast: MonoBehaviour
{
    public LayerMask LineOfSightMask;
    void FixedUpdate()
    {
        RaycastHit2D hit = Physics2D.Raycast(raycastRightPart, Vector2.down, 0.6f *
heightCharacter, LineOfSightMask);
        if(hit.collider != null)
        {
            //code when the ai can walk
        }
        else
        {
            //code when the ai cannot walk
        }
    }
}
```

このでは、はしいです。raycastRightPartはのであるため、レイキャストはのでわれます。はキャラクターのさの0.6であるので、レイキャストはにたったときにヒットしません。でっているよりもいです。Layermaskがのみにされていることをします。そうでないは、ののオブジェクトもします。

RaycastHit2Dはであり、クラスではないため、ヒットはnullにはなりません。つまり、RaycastHit2Dのコライダーをするがあります。

レイキャストコールのカプセル

あなたのスクリプトがRaycastびすと、のコリジョンをするがある、にするためにすべてのLayerMaskフィールドをするがあるため、につながるがあります。あなたのプロジェクトのによつては、これはきなになるかもしれません。

Raycastコールをカプセルすることで、あなたのがになるかもしれません。

SoCのからると、ゲームオブジェクトにはレイヤマスクをらない、またはにすべきではありません。をスキャンするだけがです。レイキャストのがこれをすのか、それともゲームオブジェクトにとってでないのか。それはそれがけるにのみし、それがするについてをらない。

これにアプローチする1つのは、LayerMaskをScriptableObjectインスタンスにし、それらをスクリプトにするレイキャストサービスのフォームとしてすることです。

```
// RaycastService.cs
using UnityEngine;

[CreateAssetMenu(menuName = "StackOverflow")]
public class RaycastService : ScriptableObject
{
    [SerializeField]
```

```
LayerMask layerMask;

public RaycastHit2D Raycast2D(Vector2 origin, Vector2 direction, float distance)
{
    return Physics2D.Raycast(origin, direction, distance, layerMask.value);
}

// Add more methods as needed
}
```

```
// MyScript.cs
using UnityEngine;

public class MyScript : MonoBehaviour
{
    [SerializeField]
    RaycastService raycastService;

    void FixedUpdate()
    {
        RaycastHit2D hit = raycastService.Raycast2D(Vector2.zero, Vector2.down, 1f);
    }
}
```

これにより、すべてのレイアキャストサービスができます。レイアキャストサービスはすべて、さまざまにじてレイヤーマスクのみわせがなります。あなたはののにするものと、のとのプラットフォームにするものをつことができます。

LayerMaskをにするがあるは、これらのRaycastServiceアセットをするだけでみます。

- の
-

オンラインでレイキャストをむ <https://riptutorial.com/ja/unity3d/topic/2826/レイキャスト>

31: レイヤー

Examples

レイヤー

ユニティレイヤーはタグとていますが、がなオブジェクトや、のであるオブジェクトをするのにできますが、レイヤーはにPhysicsクラスのでされます。 [Unity Documentation - Physics](#)

レイヤーはでされ、のようにしてにすことができます。

```
using UnityEngine;
class LayerExample {

    public int layer;

    void Start()
    {
        Collider[] colliders = Physics.OverlapSphere(transform.position, 5f, layer);
    }
}
```

このでレイヤーをすると、されたでされたレイヤーをつゲームオブジェクトをつColliderだけがまれます。これにより、ロジックをさらにし、パフォーマンスをさせます。

レイヤーマスク

LayerMaskは、のにをすのとほぼじようにするインタフェースです。ただし、のメリットは、インスペクタのドロップダウンメニューからのレイヤーをできるようにすることです。

```
using UnityEngine;
class LayerMaskExample{

    public LayerMask mask;
    public Vector3 direction;

    void Start()
    {
        if(Physics.Raycast(transform.position, direction, 35f, mask))
        {
            Debug.Log("Raycast hit");
        }
    }
}
```

また、レイヤをインデックスまたはインデックスにするためののもされています。

```
using UnityEngine;
class NameToLayerExample{
```

```
void Start()
{
    int layerindex = LayerMask.NameToLayer("Obstacle");
    {
    }
```

レイヤーチェックをにするために、このメソッドをします。

```
public static bool IsInLayerMask(this GameObject @object, LayerMask layerMask)
{
    bool result = (1 << @object.layer & layerMask) == 0;

    return result;
}
```

このメソッドは、ゲームオブジェクトがレイアースマスクエディタでにあるかどうかをチェックすることをします。

オンラインでレイヤーをむ <https://riptutorial.com/ja/unity3d/topic/4762/レイヤー>

32: システム

Examples

GetKeyとGetKeyDownとGetKeyUpのい

は、からみるがあります。

なすべてのKeyCode enumの。

1. Input.GetKey キープレス を Input.GetKey

Input.GetKeyは、ユーザーがされたキーをしている、りし true をします。これはされたキーをしたままをりしするのにできます。はSpaceキーをしたときのののです。プレイヤーはキーをももしてすはありません。

```
public GameObject bulletPrefab;
public float shootForce = 50f;

void Update()
{
    if (Input.GetKey(KeyCode.Space))
    {
        Debug.Log("Shooting a bullet while SpaceBar is held down");

        //Instantiate bullet
        GameObject bullet = Instantiate(bulletPrefab, transform.position, transform.rotation)
as GameObject;

        //Get the Rigidbody from the bullet then add a force to the bullet
        bullet.GetComponent<Rigidbody>().AddForce(bullet.transform.forward * shootForce);
    }
}
```

2でキーをして.Reading Input.GetKeyDown

Input.GetKeyDownは、されたキーがされたときに1だけtrueになります。これは、 Input.GetKeyと Input.GetKeyDown ない Input.GetKeyDown 。 そのの1つのは、 UIまたはまたはアイテムのオン/オフをりえることである。

```
public Light flashLight;
bool enableFlashLight = false;

void Update()
{
    if (Input.GetKeyDown(KeyCode.Space))
    {
        //Toggle Light
        enableFlashLight = !enableFlashLight;
        if (enableFlashLight)
        {

```



```

        flashLight.enabled = true;
        Debug.Log("Light Enabled!");
    }
    else
    {
        flashLight.enabled = false;
        Debug.Log("Light Disabled!");
    }
}
}

```

3でキーをして、Reading Input.GetKeyUp

これは Input.GetKeyDown と Input.GetKeyUp です。キーが/されたときをするためにされます。

Input.GetKeyDown とに、 true 1 だけし true。たとえば、 Input.GetKeyDown キーが Input.GetKeyDown れているとき enable ライトを enable し、 Input.GetKeyDown キーを Input.GetKeyUp ときにライトをにすることが enable ます。

```

public Light flashLight;
void Update()
{
    //Disable Light when Space Key is pressed
    if (Input.GetKeyDown(KeyCode.Space))
    {
        flashLight.enabled = true;
        Debug.Log("Light Enabled!");
    }

    //Disable Light when Space Key is released
    if (Input.GetKeyUp(KeyCode.Space))
    {
        flashLight.enabled = false;
        Debug.Log("Light Disabled!");
    }
}

```

センサーのみり

Input.acceleration は、センサーをみるためにされます。これは、3Dに x、y、z をむとして Vector3 をします。

```

void Update()
{
    Vector3 acclerometerValue = rawAccelValue();
    Debug.Log("X: " + acclerometerValue.x + " Y: " + acclerometerValue.y + " Z: " +
acclerometerValue.z);
}

Vector3 rawAccelValue()
{
    return Input.acceleration;
}

```

センサのみり Advance

センサからのをしてGameObjectをまたはさせると、きやなどがするがあります。をしてからすることをおめします。には、センサからの、するににするがあります。これは、ローパスフィルタでできます。これは、Vector3.Lerpがされています。

```
//The lower this value, the less smooth the value is and faster Accel is updated. 30 seems fine for this
const float updateSpeed = 30.0f;

float AccelerometerUpdateInterval = 1.0f / updateSpeed;
float LowPassKernelWidthInSeconds = 1.0f;
float LowPassFilterFactor = 0;
Vector3 lowPassValue = Vector3.zero;

void Start()
{
    //Filter Accelerometer
    LowPassFilterFactor = AccelerometerUpdateInterval / LowPassKernelWidthInSeconds;
    lowPassValue = Input.acceleration;
}

void Update()
{
    //Get Raw Accelerometer values (pass in false to get raw Accelerometer values)
    Vector3 rawAccelValue = filterAccelValue(false);
    Debug.Log("RAW X: " + rawAccelValue.x + " Y: " + rawAccelValue.y + " Z: " + rawAccelValue.z);

    //Get smoothed Accelerometer values (pass in true to get Filtered Accelerometer values)
    Vector3 filteredAccelValue = filterAccelValue(true);
    Debug.Log("FILTERED X: " + filteredAccelValue.x + " Y: " + filteredAccelValue.y + " Z: " + filteredAccelValue.z);
}

//Filter Accelerometer
Vector3 filterAccelValue(bool smooth)
{
    if (smooth)
        lowPassValue = Vector3.Lerp(lowPassValue, Input.acceleration, LowPassFilterFactor);
    else
        lowPassValue = Input.acceleration;

    return lowPassValue;
}
```

センサのみり

センサーをにみります。

このでは、メモリをりてます。

```
void Update()
{
    //Get Precise Accelerometer values
    Vector3 accelValue = preciseAccelValue();
    Debug.Log("PRECISE X: " + accelValue.x + " Y: " + accelValue.y + " Z: " + accelValue.z);
}
```

```

}

Vector3 preciseAccelValue()
{
    Vector3 accelResult = Vector3.zero;
    foreach (AccelerationEvent tempAccelEvent in Input.accelerationEvents)
    {
        accelResult = accelResult + (tempAccelEvent.acceleration * tempAccelEvent.deltaTime);
    }
    return accelResult;
}

```

このではメモリをりてません。

```

void Update()
{
    //Get Precise Accelerometer values
    Vector3 accelValue = preciseAccelValue();
    Debug.Log("PRECISE X: " + accelValue.x + " Y: " + accelValue.y + " Z: " + accelValue.z);
}

Vector3 preciseAccelValue()
{
    Vector3 accelResult = Vector3.zero;
    for (int i = 0; i < Input.accelerationEventCount; ++i)
    {
        AccelerationEvent tempAccelEvent = Input.GetAccelerationEvent(i);
        accelResult = accelResult + (tempAccelEvent.acceleration * tempAccelEvent.deltaTime);
    }
    return accelResult;
}

```

これはフィルタリングされないことにしてください。 [ここで](#)、ノイズをするためにのをするをて
ください。

マウスポタンのみみ、、クリック

これらのは、マウスポタンクリックのチェックにされます。

- Input.GetMouseButton(int button);
- Input.GetMouseButtonDown(int button);
- Input.GetMouseButtonUp(int button);

それらはすべてじパラメータをる。

- 0 = マウスクリック。
- 1 = マウスクリック。
- 2 = マウスクリック。

GetMouseButton は、マウスポタンがしてされたことをするためにされます。されたマウスポタンが
されているは true をし true 。

```

void Update()
{
    if (Input.GetMouseButton(0))
    {
        Debug.Log("Left Mouse Button Down");
    }

    if (Input.GetMouseButton(1))
    {
        Debug.Log("Right Mouse Button Down");
    }

    if (Input.GetMouseButton(2))
    {
        Debug.Log("Middle Mouse Button Down");
    }
}

```

`GetMouseButtonDown`は、マウスクリックがあるときをするためにされます。すと`true`し`true`。マウスボタンがされてびされるまで、び`true`りません。

```

void Update()
{
    if (Input.GetMouseButtonDown(0))
    {
        Debug.Log("Left Mouse Button Clicked");
    }

    if (Input.GetMouseButtonDown(1))
    {
        Debug.Log("Right Mouse Button Clicked");
    }

    if (Input.GetMouseButtonDown(2))
    {
        Debug.Log("Middle Mouse Button Clicked");
    }
}

```

`GetMouseButtonUp`は、されたマウスボタンがいつ`GetMouseButtonUp`されるかをするためにされます。これは、されたマウスボタンがされると`true`し`true`。もうをすには、それをびさなければならない。

```

void Update()
{
    if (Input.GetMouseButtonUp(0))
    {
        Debug.Log("Left Mouse Button Released");
    }

    if (Input.GetMouseButtonUp(1))
    {
        Debug.Log("Right Mouse Button Released");
    }

    if (Input.GetMouseButtonUp(2))
    {

```

```
        Debug.Log("Middle Mouse Button Released");  
    }  
}
```

オンラインでシステムをむ <https://riptutorial.com/ja/unity3d/topic/3413/システム>

33: モードグラフィカルユーザーインターフェイスシステム

- public static void GUILayout.Labelテキスト、パラメータGUILayoutOption [] options
- public static bool GUILayout.Buttonテキスト、パラメータGUILayoutOption [] options
- public static string GUILayout.TextAreaテキスト、params GUILayoutOption [] options

Examples

GUILayout

いUIシステムツール。ゲームでのくなくプロトタイプやデバッグにされます。

```
void OnGUI ()
{
    GUILayout.Label ("I'm a simple label text displayed in game.");

    if ( GUILayout.Button("CLICK ME") )
    {
        GUILayout.TextArea ("This is a \n
                             multiline comment.")
    }
}
```

GUILayoutは**OnGUI**でします。

オンラインでモードグラフィカルユーザーインターフェイスシステムをむ

<https://riptutorial.com/ja/unity3d/topic/6947/モードグラフィカルユーザーインターフェイスシステム-ImGui->

34:

- void Transform.Translate(Vector3 translation, Space relativeTo = Space.Self)
- void transform.Translate(float x, float y, float z, Space relativeTo = Space.Self)
- void Transform.Rotate(Vector3 eulerAngles, Space relativeTo = Space.Self)
- void Transform.Rotate(float xAngle, float yAngle, float zAngle, Space relativeTo = Space.Self)
- void Transform.Rotate(Vector3, Space relativeTo = Space.Self)
- void Transform.RotateAround(ベクトル3, ベクトル3, き)
- void Transform.LookAt(Transform target, Vector3 worldUp = Vector3.up)
- void Transform.LookAt(Vector3 worldPosition, Vector3 worldUp = Vector3.up)

Examples

トランスフォームは、オブジェクト、、、りなどのオブジェクトのをします。また、これらのプロパティのそれぞれをするもえています。すべてのGameObjectにはTransformがあります。

オブジェクトの

```
// Move an object 10 units in the positive x direction
transform.Translate(10, 0, 0);

// translating with a vector3
vector3 distanceToMove = new Vector3(5, 2, 0);
transform.Translate(distanceToMove);
```

オブジェクトの

```
// Rotate an object 45 degrees about the Y axis
transform.Rotate(0, 45, 0);

// Rotates an object about the axis passing through point (in world coordinates) by angle in degrees
transform.RotateAround(point, axis, angle);
// Rotates on it's place, on the Y axis, with 90 degrees per second
transform.RotateAround(Vector3.zero, Vector3.up, 90 * Time.deltaTime);

// Rotates an object to make it's forward vector point towards the other object
transform.LookAt(otherTransform);
// Rotates an object to make it's forward vector point towards the given position (in world coordinates)
transform.LookAt(new Vector3(10, 5, 0));
```

よりくのはUnityドキュメンテーションでることができます。

また、ゲームでをしているは、にisKinematic == trueがisKinematic == trueでないり、をししないでください。そのようなは、にするAddForceやののメソッドをします。

てと

Unityは、プロジェクトをしておくために、とします。エディタをしてオブジェクトをのりにりてすることはできますが、コードでうこともできます。

て

オブジェクトののをメソッドですることができます

```
var other = GetOtherGameObject();
other.transform.SetParent( transform );
other.transform.SetParent( transform, worldPositionStays );
```

トランスフォームのをすると、オブジェクトのをワールドにちます。 *worldPositionStays*パラメーターに *false*をして、こののをにすることができます。

のメソッドをして、オブジェクトがのトランスフォームのであるかどうかをすることもできます

```
other.transform.IsChildOf( transform );
```

をる

オブジェクトはおいをとしてうことができるので、のもつけることができます。これをうもなは、のをすることです

```
transform.Find( "other" );
transform.FindChild( "other" );
```

*FindChild*びしは、フードのでします。

さらにのにあるをすることもできます。これをうには、「/」をしてレベルをくします。

```
transform.Find( "other/another" );
transform.FindChild( "other/another" );
```

をフェッチするもう1つのは、*GetChild*

```
transform.GetChild( index );
```

*GetChild*はインデックスとしてをとします。これはカウントよりもさくなければなりません

```
int count = transform.childCount;
```

インデックスの

*GameObject*ののをすることができます。これをうと、のをできますじZレベルとじソートである

とします。

```
other.transform.SetSiblingIndex( index );
```

のメソッドをして、インデックスをまたはにくすることもできます

```
other.transform.SetAsFirstSibling();  
other.transform.SetAsLastSibling();
```

すべてのをける

トランスフォームのすべてのをしたい、これをうことができます

```
foreach(Transform child in transform)  
{  
    child.parent = null;  
}
```

また、Unityはこののためのをします

```
transform.DetachChildren();
```

に、`DetachChildren()` と `DetachChildren()` は、のさののをnullにします。つまり、をたないこととなります。

のさののである

オンラインでをむ <https://riptutorial.com/ja/unity3d/topic/2190/>

35:

- [AddComponentMenustring menuName]
- [AddComponentMenustring menuName、 int order]
- [CanEditMultipleObjects]
- [ContextMenuitem、]
- [ContextMenu]
- [CustomEditorタイプinspectedType]
- [CustomEditorされた、 bool editorForChildClasses]
- [CustomPropertyDrawerの]
- [CustomPropertyDrawerの、 bool useForChildren]
- [DisallowMultipleComponent]
- [DrawGizmo GizmoタイプGizmo]
- [DrawGizmoGizmoType Gizmo、 タイプdrawnGizmoType]
- [ExecuteInEditMode]
- [ヘッダーヘッダー]
- [HideInInspector]
- [InitializeOnLoad]
- [InitializeOnLoadMethod]
- [MenuItemstring itemName]
- [MenuItemstring itemName、 bool isValidFunction]
- [MenuItemstring itemName、 bool isValidFunction、 int priority]
- [int]
- [PreferenceItem]
- [float min、 float max]
- [RequireComponentの]
- [RuntimeInitializeOnLoadMethod]
- [RuntimeInitializeOnLoadMethodRuntimeInitializeLoadType loadType]
- [SerializeField]
- [スペースきのさ]
- [TextAreaint minLines、 int maxLines]
- [ツールチップツールチップ]

SerializeField

Unityのシリアライゼーションシステムをして、をすることができます

- シリアライズの中のフィールドをシリアルすることができます
- [SerializeField]でマークされたのフィールドをシリアルできます
- フィールドをシリアルできません
- プロパティをシリアルできません

フィールドは、たとえ `SerializeField` でマークされていても、Unity がシリアルできるタイプであるにのみけされます。

- `UnityEngine.Object` からしたすべてのクラス `GameObject`、`Component`、`MonoBehaviour`、`Texture2D`
- `int`、`string`、`float`、`bool` のようなすべてのデータ
- `Vector2 / 3 / 4`、`Quaternion`、`Matrix4x4`、`Color`、`Rect`、`LayerMask` などのビルトインタイプ
- シリアライズなの
- なのリスト
-
-

Examples

のインスペクタ

```
[Header( "My variables" )]
public string MyString;

[HideInInspector]
public string MyHiddenString;

[Multiline( 5 )]
public string MyMultilineString;

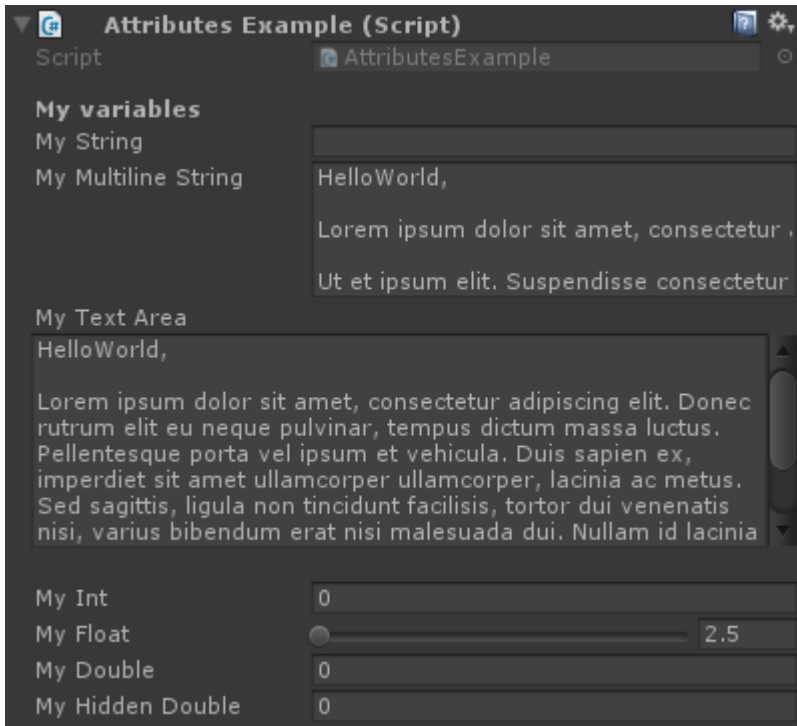
[TextArea( 2, 8 )]
public string MyTextArea;

[Space( 15 )]
public int MyInt;

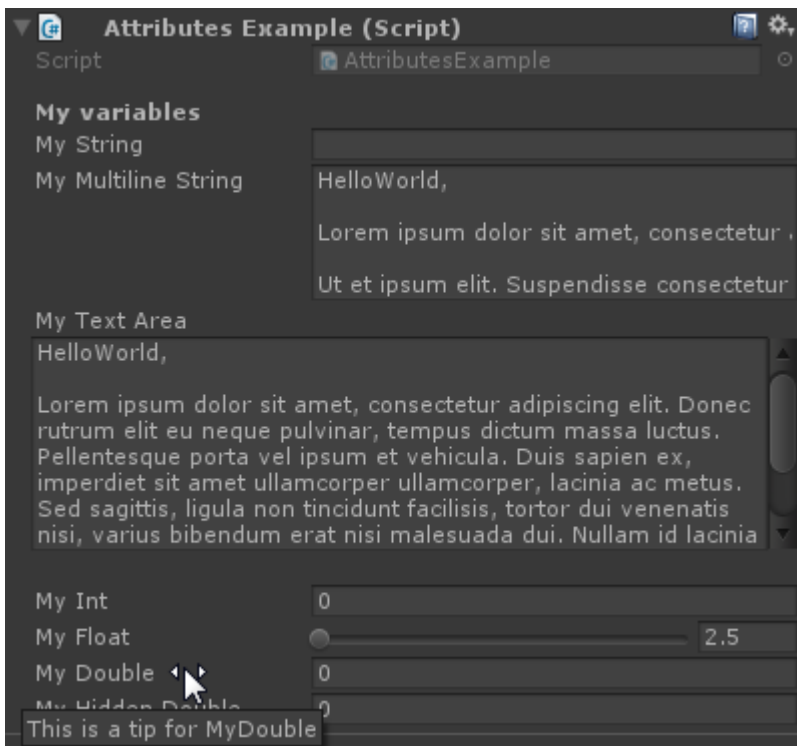
[Range( 2.5f, 12.5f )]
public float MyFloat;

[Tooltip( "This is a tip for MyDouble" )]
public double MyDouble;

[SerializeField]
private double myHiddenDouble;
```



フィールドのラベルにカーソルをわせると、のようになります。



```
[Header( "My variables" )]
public string MyString;
```

ヘッダーは、フィールドのラベルをします。これは、グループをのラベルにしてたせるためにグループにラベルをけるによくされます。

```
[HideInInspector]
public string MyHiddenString;
```

HideInInspectorは、パブリックフィールドがインスペクタにされないようにします。これは、されないかでないコードののからフィールドにアクセスするにです。

```
[Multiline( 5 )]
public string MyMultilineString;
```

Multilineは、されたのテキストボックスをします。このをえると、ボックスがされず、テキストもりされません。

```
[TextArea( 2, 8 )]
public string MyTextArea;
```

TextAreaは、りしをむスタイルのテキストと、テキストがりてられたをえるはスクロールバーをします。

```
[Space( 15 )]
public int MyInt;
```

スペースは、のアイテムとのアイテムのになスペースをするようにインスペクタにします。これは、グループのとにちます。

```
[Range( 2.5f, 12.5f )]
public float MyFloat;
```

Rangeは、とののをします。minとmaxがとしてされていて、このはとでもします。

```
[Tooltip( "This is a tip for MyDouble" )]
public double MyDouble;
```

ツールチップには、フィールドのラベルをにいたときにのがされます。

```
[SerializeField]
private double myHiddenDouble;
```

SerializeFieldはUnityにフィールドのシリアルをします。プライベートフィールドにです。

コンポーネントの

```
[DisallowMultipleComponent]
[RequireComponent( typeof( Rigidbody ) )]
public class AttributesExample : MonoBehaviour
{
    [...]
}
```

```
[DisallowMultipleComponent]
```

`DisallowMultipleComponent`は、ユーザーがこのコンポーネントののインスタンスを1つの `GameObject`にするのをぎます。

```
[RequireComponent ( typeof( Rigidbody ) ) ]
```

`RequireComponent`をすると、このコンポーネントを `GameObject`にするときのとしてのコンポーネントまたはそれをできません。このコンポーネントを `GameObject`にすると、なコンポーネントがにされますししない。それらのコンポーネントは、なコンポーネントがされるまでできません。

```
[ExecuteInEditMode]
public class AttributesExample : MonoBehaviour
{
    [RuntimeInitializeOnLoadMethod]
    private static void FooBar()
    {
        [...]
    }

    [RuntimeInitializeOnLoadMethod( RuntimeInitializeLoadType.BeforeSceneLoad )]
    private static void Foo()
    {
        [...]
    }

    [RuntimeInitializeOnLoadMethod( RuntimeInitializeLoadType.AfterSceneLoad )]
    private static void Bar()
    {
        [...]
    }

    void Update()
    {
        if ( Application.isEditor )
        {
            [...]
        }
        else
        {
            [...]
        }
    }
}
```

```
[ExecuteInEditMode]
public class AttributesExample : MonoBehaviour
```

`ExecuteInEditMode`は、ゲームがプレイされていないでも、このスクリプトのマジックメソッドをUnityににさせます。

これは、モードのようににびされるわけではありません

- は、シーンのかがされたときにのみびされます。
- `OnGUI`は、ゲームビューがイベントをけるとびされます。

- **OnRenderObject**とのレンダリングコールバックは、シーンビューまたはゲームビューのすべてのでびされます。

```
[RuntimeInitializeOnLoadMethod]
private static void FooBar()

[RuntimeInitializeOnLoadMethod( RuntimeInitializeLoadType.BeforeSceneLoad )]
private static void Foo()

[RuntimeInitializeOnLoadMethod( RuntimeInitializeLoadType.AfterSceneLoad )]
private static void Bar()
```

RuntimeInitializeOnLoadMethodをすると、ゲームがランタイムをロードしたときに、ユーザーからのなしでランタイムクラスメソッドをびすことができます。

シーンロードのまたはにメソッドをびすかどうかをできますデフォルトは**after**です。このをするメソッドのはされていません。

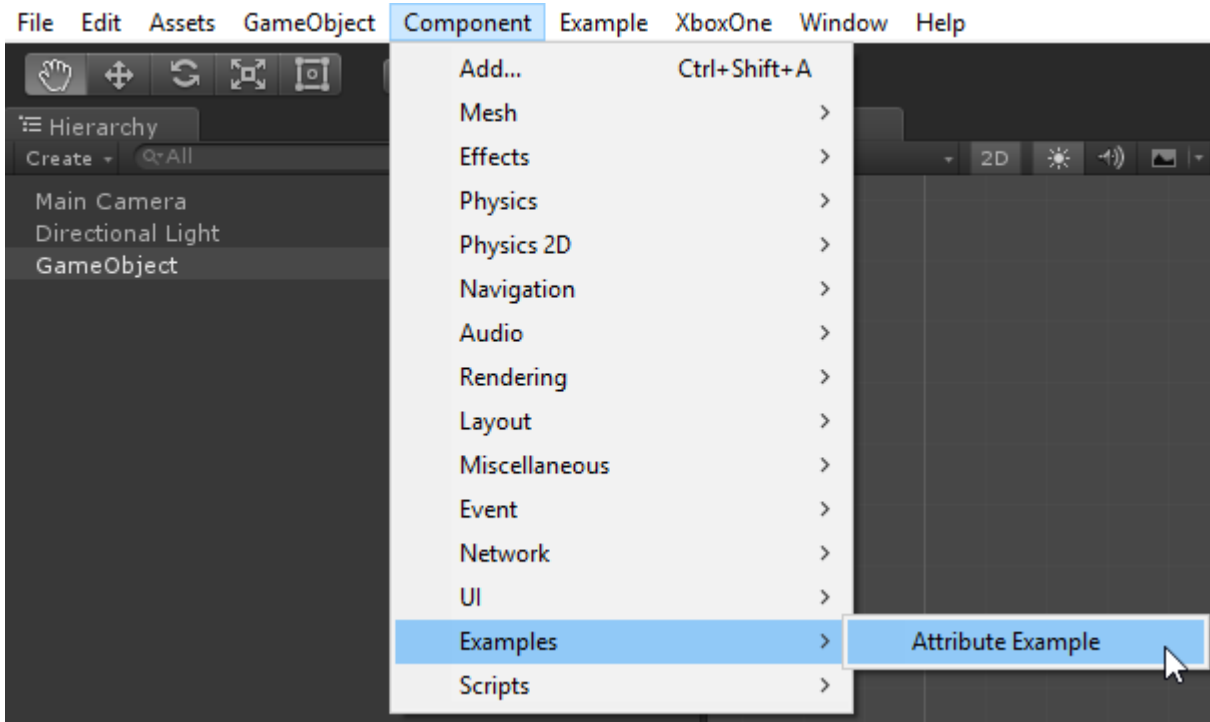
メニュー

```
[AddComponentMenu( "Examples/Attribute Example" )]
public class AttributesExample : MonoBehaviour
{
    [ContextMenu( "My Field Action", "MyFieldContextAction" )]
    public string MyString;

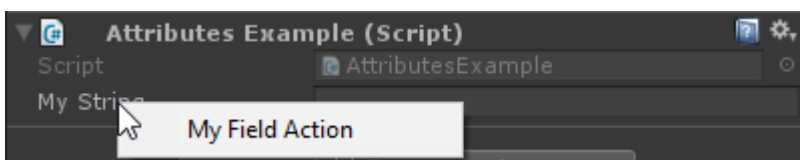
    private void MyFieldContextAction()
    {
        [...]
    }

    [ContextMenu( "My Action" )]
    private void MyContextMenuAction()
    {
        [...]
    }
}
```

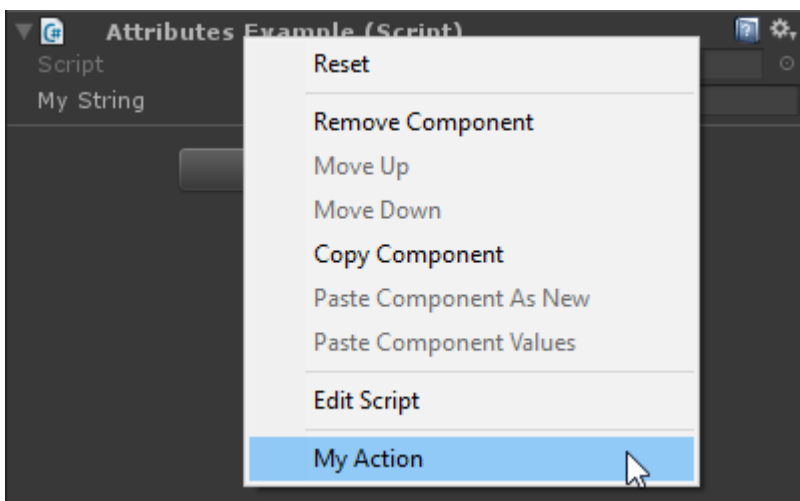
[AddComponentMenu]の



[ContextMenuItem]の



[ContextMenu]の



```
[AddComponentMenu( "Examples/Attribute Example" )]
public class AttributesExample : MonoBehaviour
```

AddComponentMenuをすると、コンポーネント ->スクリプトメニューではなく、コンポーネントメニューののにコンポーネントをできます。

```
[ContextMenuItem( "My Field Action", "MyFieldContextAction" )]
```



```
public string MyString;

private void MyFieldContextAction()
{
    [...]
}
```

ContextMenuItemをすると、フィールドのコンテキストメニューにできるをできます。これらのはにされます。

```
[ContextMenu( "My Action" )]
private void MyContextMenuAction()
{
    [...]
}
```

ContextMenuをすると、コンポーネントのコンテキストメニューにできるをできます。

エディタの

```
[InitializeOnLoad]
public class AttributesExample : MonoBehaviour
{

    static AttributesExample()
    {
        [...]
    }

    [InitializeOnLoadMethod]
    private static void Foo()
    {
        [...]
    }
}
```

```
[InitializeOnLoad]
public class AttributesExample : MonoBehaviour
{

    static AttributesExample()
    {
        [...]
    }
}
```

InitializeOnLoadをすると、ユーザーはユーザーとのやりとりなしにクラスをできます。これは、エディタがするかコンパイルされるたびにします。コンストラクタは、これがののにびされることをします。

```
[InitializeOnLoadMethod]
private static void Foo()
{
    [...]
}
```

```
}
```

InitializeOnLoadをすると、ユーザーはユーザーとのやりとりなしにクラスをできます。これは、エディタがするかコンパイルされるたびにします。このをするメソッドのはされていません。

```
[CanEditMultipleObjects]
public class AttributesExample : MonoBehaviour
{
    public int MyInt;

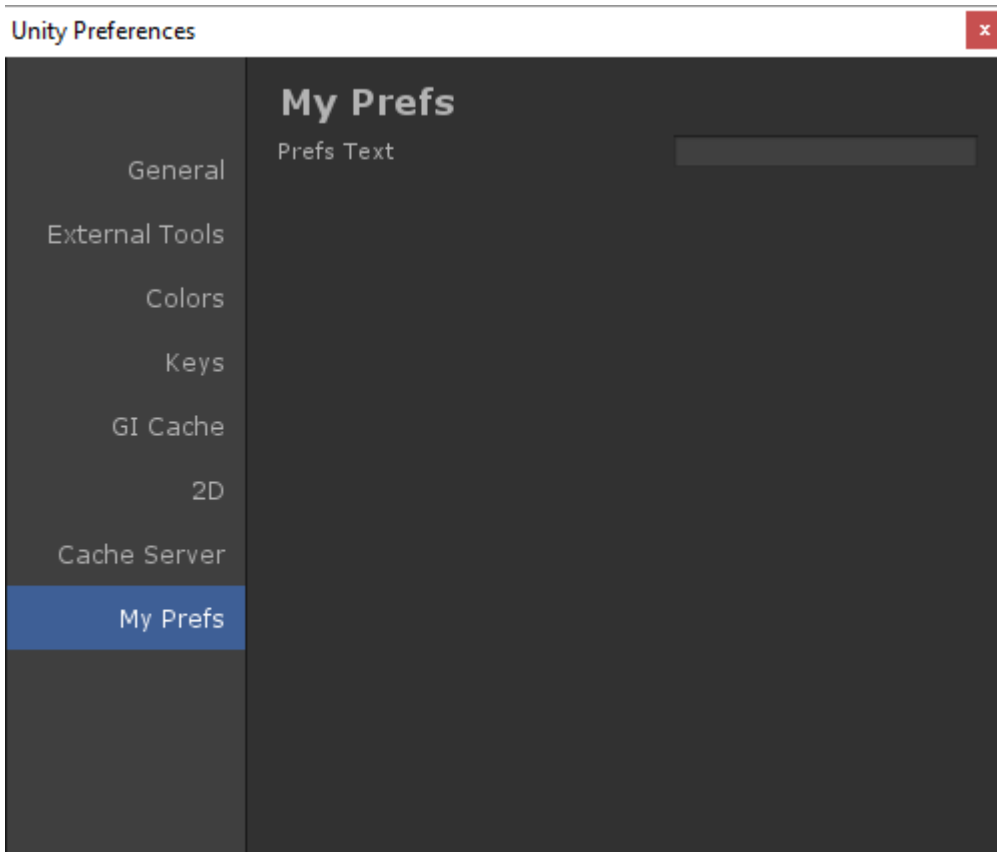
    private static string prefsText = "";

    [PreferenceItem( "My Prefs" )]
    public static void PreferencesGUI()
    {
        prefsText = EditorGUILayout.TextField( "Prefs Text", prefsText );
    }

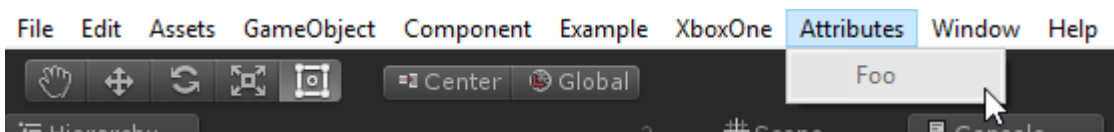
    [MenuItem( "Attributes/Foo" )]
    private static void Foo()
    {
        [...]
    }

    [MenuItem( "Attributes/Foo", true )]
    private static bool FooValidate()
    {
        return false;
    }
}
```

[PreferenceItem]の



[MenuItem]の



```
[CanEditMultipleObjects]  
public class AttributesExample : MonoBehaviour
```

CanEditMultipleObjectsをすると、コンポーネントからのGameObjectにアクセスをすることができます。このコンポーネントがなければ、GameObjectをしたときにコンポーネントがのようにされることはありませんが、代わりに「マルチオブジェクトのはサポートされていません」というメッセージがされます。

これは、カスタムがマルチをサポートするためのものです。カスタムエディタはにマルチをサポートします。

```
[PreferenceItem( "My Prefs" )]  
public static void PreferencesGUI()
```

PreferenceItemをすると、Unityのメニューでアクセスをすることができます。は、するにはであるがあります。

```
[MenuItem( "Attributes/Foo" )]  
private static void Foo()  
{  
    [...]  
}
```

```
}

[MenuItem( "Attributes/Foo", true )]
private static bool FooValidate()
{
    return false;
}
```

MenuItemをすると、カスタムメニューをしてをできます。このでは、のをぐためにバリデータもしますに**false**をします。

```
[CustomEditor( typeof( MyComponent ) )]
public class AttributesExample : Editor
{
    [...]
}
```

CustomEditorをすると、コンポーネントのカスタムエディタをできます。これらのエディタはインスペクタでコンポーネントをするためにされ、**Editor**クラスからするがあります。

```
[CustomPropertyDrawer( typeof( MyClass ) )]
public class AttributesExample : PropertyDrawer
{
    [...]
}
```

CustomPropertyDrawerをすると、インスペクタのカスタムプロパティドロワをできます。これらのきしをカスタムデータにすると、インスペクタですることができます。

```
[DrawGizmo( GizmoType.Selected )]
private static void DoGizmo( AttributesExample obj, GizmoType type )
{
    [...]
}
```

DrawGizmoをすると、コンポーネントのカスタムギズモをできます。これらのギズモはシーンビューでされます。 **DrawGizmo**の**GizmoType**パラメーターをして、ギズモをするタイミングをできます。

メソッドは2つのパラメータをとします。1つは、ギズモをするコンポーネントで、2つは、されたギズモとするオブジェクトがっています。

オンラインでもむ <https://riptutorial.com/ja/unity3d/topic/5535/>

36: の

き

このトピックでは、ユニティやGoogle AdMobなどのサードパーティサービスをUnityプロジェクトにするについてします。

これはUnity Adsにされます。

にUnity Adsのテストモードがになっていることをする

は、のゲームでをクリックしてインプレッションやインストールをすることはできません。そうすることで、Unity Adsにし、Unity Adsネットワークからをされます。

については、ユニティのをご覧ください。

Examples

Cのユニティの

```
using UnityEngine;
using UnityEngine.Advertisements;

public class Example : MonoBehaviour
{
    #if !UNITY_ADS // If the Ads service is not enabled
    public string gameId; // Set this value from the inspector
    public bool enableTestMode = true; // Enable this during development
    #endif

    void InitializeAds () // Example of how to initialize the Unity Ads service
    {
        #if !UNITY_ADS // If the Ads service is not enabled
        if (Advertisement.isSupported) { // If runtime platform is supported
            Advertisement.Initialize(gameId, enableTestMode); // Initialize
        }
        #endif
    }

    void ShowAd () // Example of how to show an ad
    {
        if (Advertisement.isInitialized || Advertisement.IsReady()) { // If the ads are ready
        to be shown
            Advertisement.Show(); // Show the default ad placement
        }
    }
}
```

JavaScriptのユニティの

```
#pragma strict
import UnityEngine.Advertisements;

#if !UNITY_ADS // If the Ads service is not enabled
public var gameId : String; // Set this value from the inspector
public var enableTestMode : boolean = true; // Enable this during development
#endif

function InitializeAds () // Example of how to initialize the Unity Ads service
{
    #if !UNITY_ADS // If the Ads service is not enabled
    if (Advertisement.isSupported) { // If runtime platform is supported
        Advertisement.Initialize(gameId, enableTestMode); // Initialize
    }
    #endif
}

function ShowAd () // Example of how to show an ad
{
    if (Advertisement.isInitialized && Advertisement.IsReady()) { // If the ads are ready to
be shown
        Advertisement.Show(); // Show the default ad placement
    }
}
```

オンラインでのをむ <https://riptutorial.com/ja/unity3d/topic/9796/>の

37:

1. であれば、でないオブジェクトのスク립トをにします。えは、オブジェクトにスク립トをっていて、プレイヤーをしたりしたりする、がプレイヤーからすぎるにこのスク립トをにすることをしてください。

Examples

かつなチェック

なり、にUpdateなどのももびされるメソッドでは、なやメソッドびしをけてください。

/チェック

をするときは、magnitudeわりにsqrMagnitudeをします。これにより、なsqrtがされます。sqrMagnitudeをするsqrMagnitudeは、もするがあります。

```
if ((target.position - transform.position).sqrMagnitude < minDistance * minDistance))
```

チェック

オブジェクトのは、そのCollider / Rendererがしているかどうかをチェックすることによって、きちんとチェックすることができます。Boundsには、2つのがするかどうかをするのにつ、なIntersectsメソッドもあります。

Boundsまた、オブジェクトののにまでののいをするためにたちをけるBounds.SqrDistance。

チェックは、オブジェクトのはにうまくしますが、オブジェクトのチェックは、オブジェクトののよってはるかにいをもたらずがあります。

Mesh.boundsをすることは、ローカルのをすのでされません。わりにMeshRenderer.boundsをしてください。

コルーチンの

スレッドセーフではないUnity APIにするのをしているは、Coroutinesをしてのフレームにし、アプリケーションをにします。

コルーチンはまた、フレームでそのアクションをするわりに、なアクションをnのフレームごとに

するのにちます。

のフレームにわたるルーチンの

Coroutinesは、されるのをフレームにして、アプリケーションのフレームレートをするのにちます。

きにをペイントまたはするルーチンやノイズをするルーチンは、コルーチンのがなです。

```
for (int y = 0; y < heightmap.Height; y++)
{
    for (int x = 0; x < heightmap.Width; x++)
    {
        // Generate pixel at (x, y)
        // Assign pixel at (x, y)

        // Process only 32768 pixels each frame
        if ((y * heightmap.Height + x) % 32 * 1024 == 0)
            yield return null; // Wait for next frame
    }
}
```

のコードはかりやすいです。プロダクションコードでは、`yield return`いつ`yield return`おそらく23ごとに、`for`ループをにすることをチェックするピクセルのチェックをけるほうがよいです。

なアクションをあまりにしない

コルーチンは、なアクションのをらすことができます。そのため、フレームごとにされるとじょうにパフォーマンスがすることはありません。

マニュアルからのをえてみましょう

```
private void ProximityCheck()
{
    for (int i = 0; i < enemies.Length; i++)
    {
        if (Vector3.Distance(transform.position, enemies[i].transform.position) <
            dangerDistance)
            return true;
    }
    return false;
}

private IEnumerator ProximityCheckCoroutine()
{
    while(true)
    {
        ProximityCheck();
        yield return new WaitForSeconds(.1f);
    }
}
```



```
}  
}
```

テストは、[CullingGroup API](#)をしてさらにすることができます。

とし

がよくういは、コルーチンのコルーチンのやにアクセスすることです。コルーチンは、`yield return`にし、そのまたはがまだされないとすぐに、びしにをします。コルーチンで/をするがあるをするには、[このを](#)してください。

なよりもUnityにきなリソースがあるとするかもしれませんが、いであるがです。

でゴミがされる

ほとんどののはごくわずかなのゴミをしますが、それらのが1のでかびされると、スタックされます。がつとにガベージコレクションがトリガーされ、CPUスパイクがつことがあります。

をキャッシュする

のをえてみましょう。

```
string[] StringKeys = new string[] {  
    "Key0",  
    "Key1",  
    "Key2"  
};  
  
void Update()  
{  
    for (var i = 0; i < 3; i++)  
    {  
        // Cached, no garbage generated  
        Debug.Log(StringKeys[i]);  
    }  
  
    for (var i = 0; i < 3; i++)  
    {  
        // Not cached, garbage every cycle  
        Debug.Log("Key" + i);  
    }  
  
    // The most memory-efficient way is to not create a cache at all and use literals or  
    constants.  
    // However, it is not necessarily the most readable or beautiful way.  
    Debug.Log("Key0");  
    Debug.Log("Key1");  
    Debug.Log("Key2");  
}
```

それはかでにえるかもしれませんが、Shadersでしているは、これらのにするかもしれません。キーをキャッシュすることでいがまれます。

リテラルとは、プログラムスタックににされるため、ガベージをしないことにしてください。あなたはにをしていると、ののようにじをすることがされているは、キャッシュはいなくちます。

されるがじでないのには、それらのをすることはありません。そのため、でをするメモリスパイクは、にものがされないうり、はできます。

ほとんどののはデバッグメッセージです

デバッグメッセージの、つまり`Debug.Log("Object Name: " + obj.name)`はなく、にけることはできません。ただし、リリースされたにのデバッグメッセージがらないようにすることがです。

1つのは、デバッグびして`Conditional`をすることです。これはメソッドびしだけでなく、すべてのもりきます。

```
using UnityEngine;
using System.Collections;

public class ConditionalDebugExample: MonoBehaviour
{
    IEnumerator Start()
    {
        while(true)
        {
            // This message will pop up in Editor but not in builds
            Log("Elapsed: " + Time.timeSinceLevelLoad);
            yield return new WaitForSeconds(1f);
        }
    }

    [System.Diagnostics.Conditional("UNITY_EDITOR")]
    void Log(string Message)
    {
        Debug.Log(Message);
    }
}
```

これはなです。になロギングルーチンをするには、しばらくをやすことをおめします

。



これはマイナーなですが、するがあります。のは、えられるよりもです。システムは、デフォルトでないをにれようとします。わりになバイナリをすることができます。これはです。

```
// Faster string comparison
if (strA.Equals(strB, System.StringComparison.Ordinal)) {...}
// Compared to
```

```

if (strA == strB) {...}

// Less overhead
if (!string.IsNullOrEmpty(strA)) {...}
// Compared to
if (strA == "") {...}

// Faster lookups
Dictionary<string, int> myDic = new Dictionary<string, int>(System.StringComparer.Ordinal);
// Compared to
Dictionary<string, int> myDictionary = new Dictionary<string, int>();

```

キャッシュ

にでなびしをけるためにをキャッシュします。これは、なはにこれらのをキャッシュするか、またはなにはnull / boolフラットをチェックしてをしないようにすることによってできます。

コンポーネントをキャッシュする

する

```

void Update()
{
    var renderer = GetComponent<Renderer>();
    renderer.material.SetColor("_Color", Color.green);
}

```

に

```

private Renderer myRenderer;
void Start()
{
    myRenderer = GetComponent<Renderer>();
}

void Update()
{
    myRenderer.material.SetColor("_Color", Color.green);
}

```

オブジェクトをキャッシュする

する

```

void Update()
{
    var enemy = GameObject.Find("enemy");
    enemy.transform.LookAt(new Vector3(0,0,0));
}

```

に

```

private Transform enemy;

```

```
void Start()
{
    this.enemy = GameObject.Find("enemy").transform;
}

void Update()
{
    enemy.LookAt(new Vector3(0, 0, 0));
}
```

さらに、なりMathfへのびしのようななびしをキャッシュします。

をしたメソッドのびしをける

メソッドをけることができるをってメソッドをびさないようにしてください。このアプローチでは、リフレクションをして、にでするときにゲームをさせることができます。

```
//Avoid StartCoroutine with method name
this.StartCoroutine("SampleCoroutine");

//Instead use the method directly
this.StartCoroutine(this.SampleCoroutine());

//Avoid send message
var enemy = GameObject.Find("enemy");
enemy.SendMessage("Die");

//Instead make direct call
var enemy = GameObject.Find("enemy") as Enemy;
enemy.Die();
```

のをける

のをけてください。プログラミングスタイルがいただけでなく、ランタイムスクリプトにうオーバーヘッドもなくなります。くの、これはビルドアップしてパフォーマンスにをえます。

```
void Update
{
}

void FixedUpdate
{
}
```

オンラインでをむ <https://riptutorial.com/ja/unity3d/topic/3433/>

38:

Examples

リジッドボディー

Rigidbodyコンポーネントは、GameObjectがににすることをにできるようにします。あなたはGameObjectにをえることも、やのがそれにたるようにさせることもできます。

リジッドボディーコンポーネントの

コンポーネント>>リジッドボディーをクリックしてリジッドボディーをできます

オブジェクトの

RigidbodyをGameObjectにするは、Transformををするのではなく、かすためにまたはトルクをすることをめします。AddForce()またはAddTorque()この

```
// Add a force to the order of myForce in the forward direction of the Transform.
GetComponent<Rigidbody>().AddForce(transform.forward * myForce);

// Add torque about the Y axis to the order of myTurn.
GetComponent<Rigidbody>().AddTorque(transform.up * torque * myTurn);
```

リジッドボディーゲームオブジェクトのをして、のリジッドボディーやとどのようにするかにをえます。よりいは、GameObjectがのベースのGameObjectにもっとをえ、よりきなをとすることをします。なるのは、じををする、じでする。コードのをするには

```
GetComponent<Rigidbody>().mass = 1000;
```

ドラッグ

ドラッグがいほど、オブジェクトはにします。ののようにえる。コードのドラッグをするには

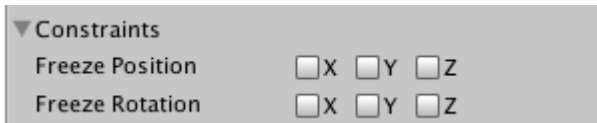
```
GetComponent<Rigidbody>().drag = 10;
```

isKinematic

あなたがRigidbodyをKinematicとしてマークした、それはののをけることはできませんが、のGameObjectにはまだをぼします。コードをするには

```
GetComponent<Rigidbody>().isKinematic = true;
```

リジッドボディのやをするために、にをえることもできます。デフォルトはRigidbodyConstraints.Noneです。



コードのの

```
// Freeze rotation on all axes.
GetComponent<Rigidbody>().constraints = RigidbodyConstraints.FreezeRotation

// Freeze position on all axes.
GetComponent<Rigidbody>().constraints = RigidbodyConstraints.FreezePosition

// Freeze rotation and motion an all axes.
GetComponent<Rigidbody>().constraints = RigidbodyConstraints.FreezeAll
```

ビットのORをできます、のようにのをみわせる

```
// Allow rotation on X and Y axes and motion on Y and Z axes.
GetComponent<Rigidbody>().constraints = RigidbodyConstraints.FreezePositionZ |
    RigidbodyConstraints.FreezeRotationX;
```

RigidbodyをしたGameObjectでにさせたいは、Colliderをするがあります。コライダーのタイプはのとおりです。

- ボックスコライダー
-
- カプセルコライダー
- ホイール
- メッシュコライダー

GameObjectにのコライダーをするは、コライダーとびます。

OnTriggerEnter()、OnTriggerStay()およびOnTriggerExit()メソッドをするには、トリガーにコライ

データをします。トリガーコライダーはににせず、のGameObjectはにそれをします。それらは、アイテムをするときなど、のGameObjectがのにあるかどうかをします。

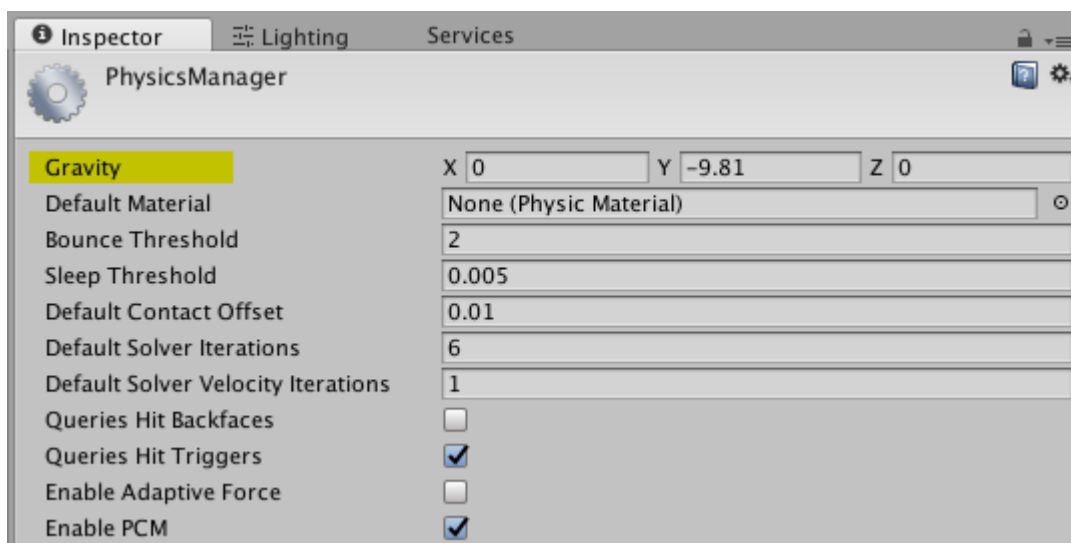
の

useGravityのプロパティRigidBodyがそれにをえるかどうかをします。falseするfalse、RigidBodyはあるにのをえずにのようになります。

```
GetComponent<RigidBody>().useGravity = false;
```

これは、RigidBodyのすべてののがでされたモーシヨンのにはにです。

にすると、RigidBodyはPhysics Settingsでされたのをけます



Gravityは、1あたりのでされ、3ベクトルとしてここにされます。ののでは、useGravityプロパティがTrueされたRigidBodiesは、useGravity 9.81のきにユニティののにおいてのyとしてき。

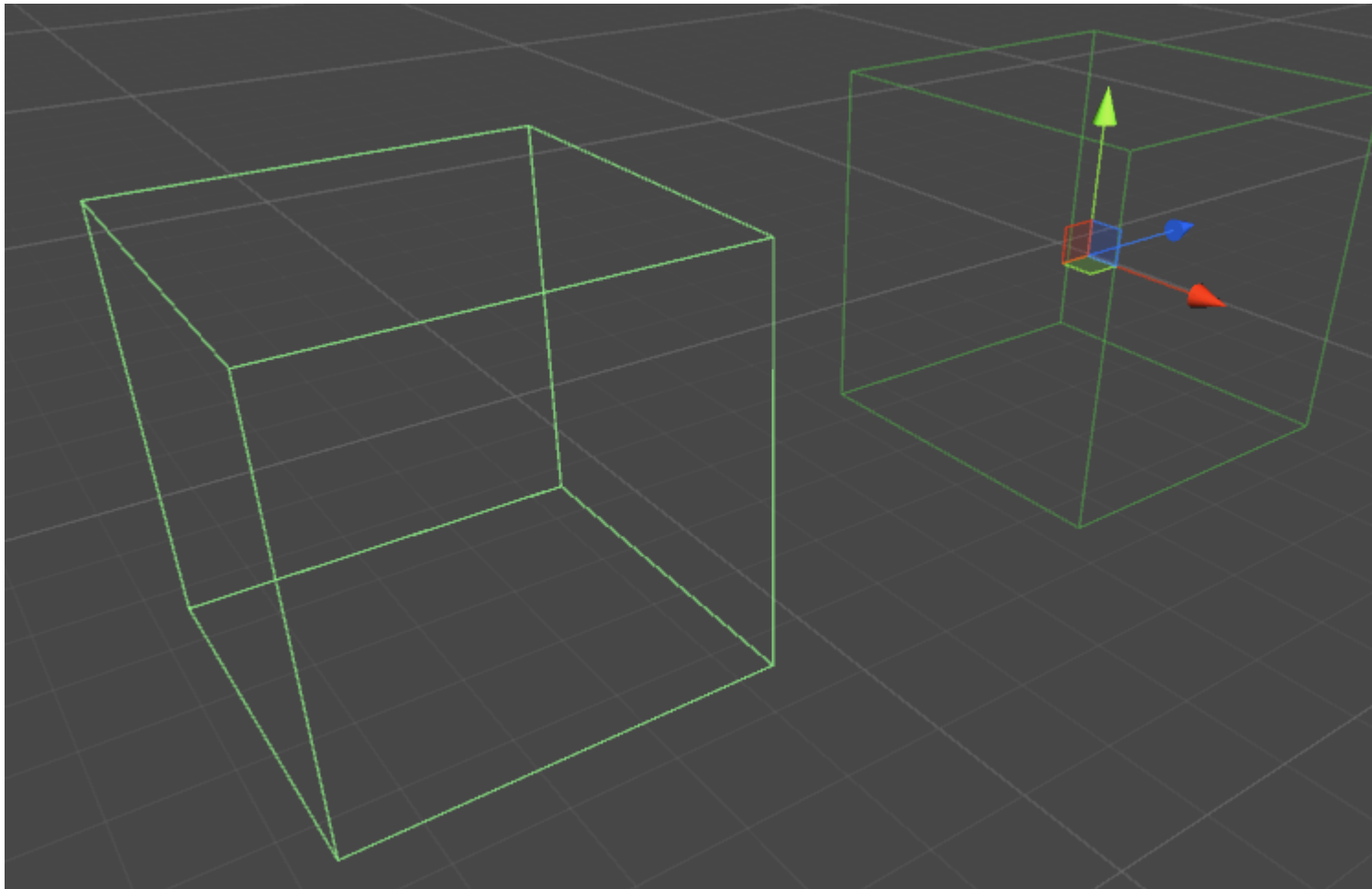
オンラインでをむ <https://riptutorial.com/ja/unity3d/topic/3680/>

39:

Examples

ボックスコライダー

のようなをしたなコライダー。



プロパティ

- **Is Trigger** - チェックをれると、Box Colliderはをしてトリガーコライダーになります
- **マテリアル** - されているは、Box Colliderのマテリアルへの
- **センター** - ローカルスペースにおけるBox Colliderの
- **Size** - ローカルスペースでされたBox Colliderのサイズ

```
// Add a Box Collider to the current GameObject.  
BoxCollider myBC = BoxCollider(myGameObject.gameObject.AddComponent(typeof(BoxCollider)));
```

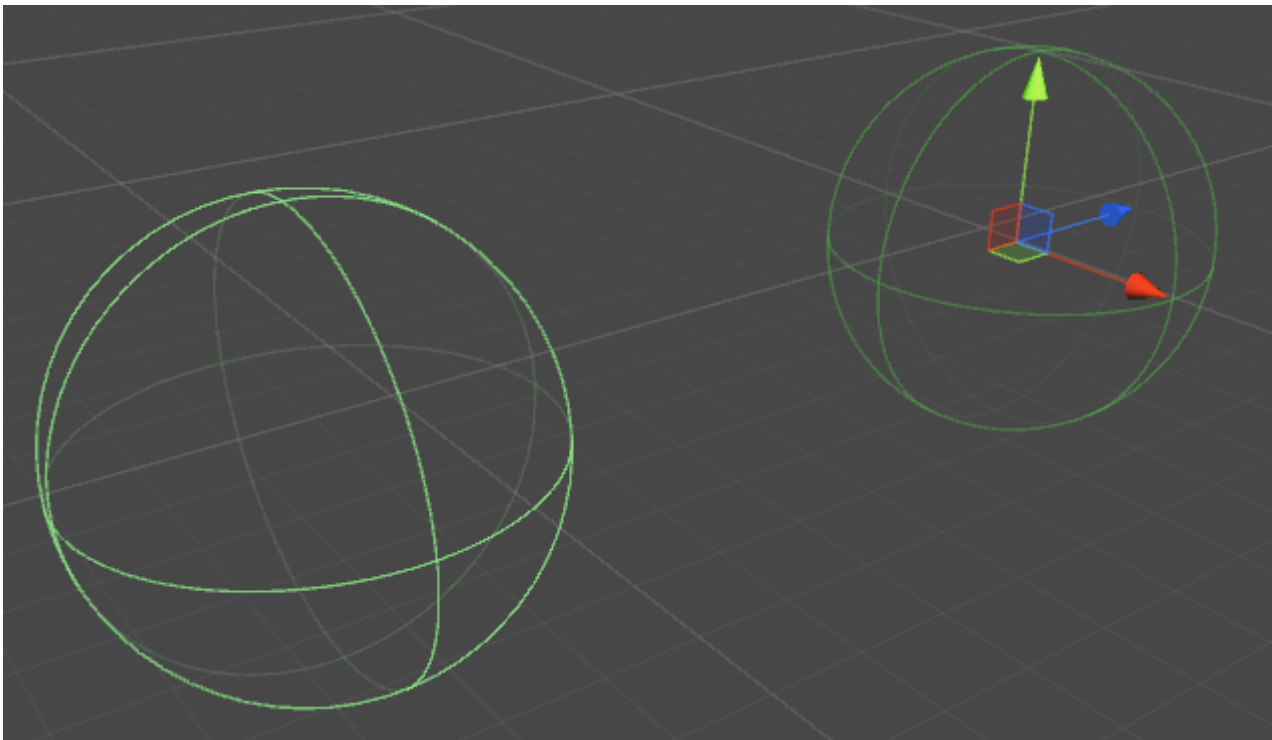


```
// Make the Box Collider into a Trigger Collider.
myBC.isTrigger= true;

// Set the center of the Box Collider to the center of the GameObject.
myBC.center = Vector3.zero;

// Make the Box Collider twice as large.
myBC.size = 2;
```

のようなをしたコライダー。



プロパティ

- トリガー**Trigger** - チェックをれると、Sphere Colliderはをしてトリガー・コライダー
- マテリアル - されている、Sphere Colliderのマテリアルへの
- **Center** - Sphere Colliderのローカルスペースにおける
- **Radius** - Colliderの

```
// Add a Sphere Collider to the current GameObject.
SphereCollider mySC =
SphereCollider)myGameObject.gameObject.AddComponent (typeof (SphereCollider));

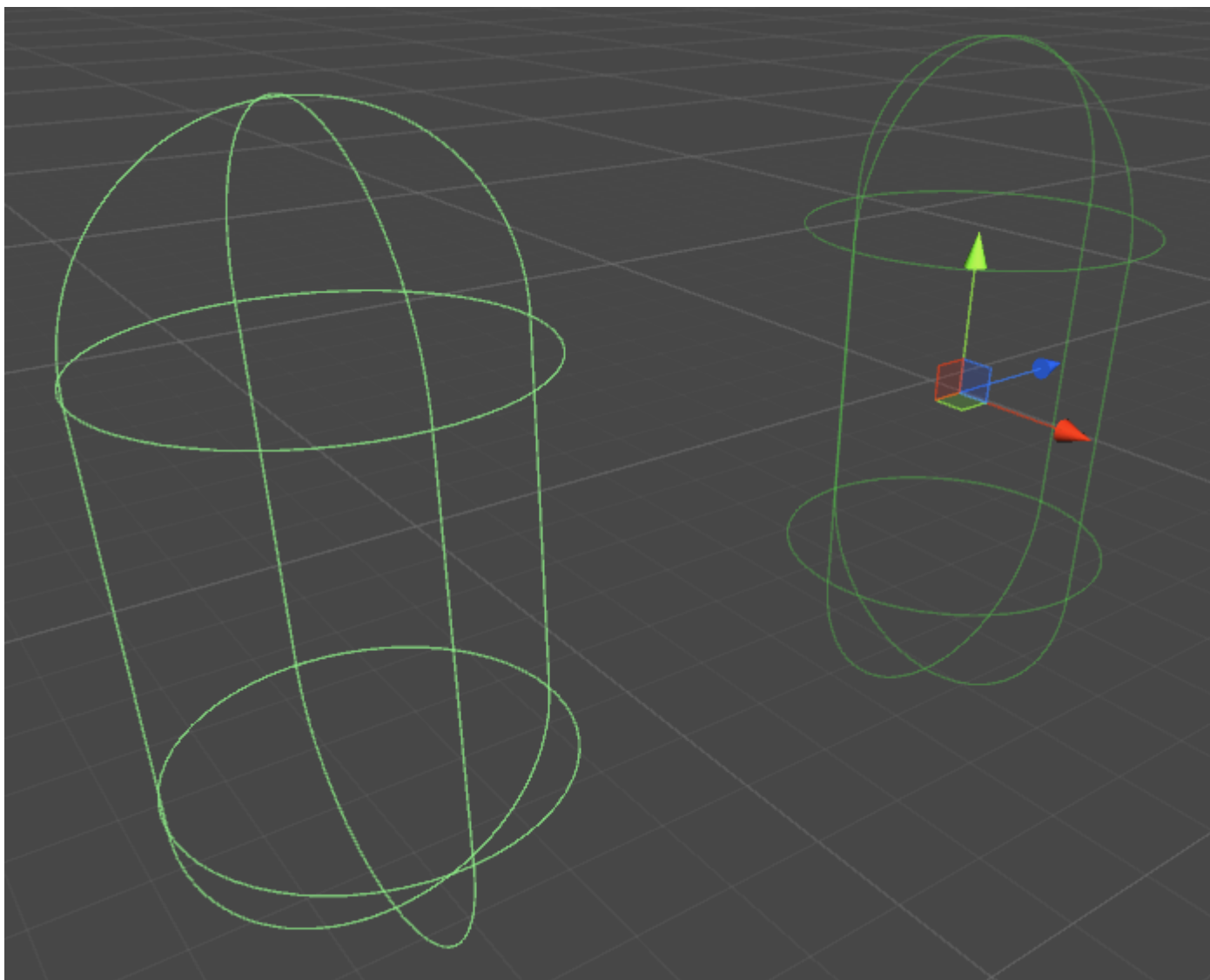
// Make the Sphere Collider into a Trigger Collider.
mySC.isTrigger= true;

// Set the center of the Sphere Collider to the center of the GameObject.
```

```
mySC.center = Vector3.zero;  
  
// Make the Sphere Collider twice as large.  
mySC.radius = 2;
```

カプセルコライダー

シリンダーでされた2つの。



プロパティ

- **Is Trigger** - チェックをされると、Capsule Colliderはをしてトリガーコライダーになります
- マテリアル **Material** - されている、カプセルコライダーのマテリアルへの
- センター - ローカルスペースでのカプセルコライダーの
- **Radius** - ローカルの

- さ - コライダーの
- **Direction** - ローカルにおけるの

```
// Add a Capsule Collider to the current GameObject.
CapsuleCollider myCC =
CapsuleCollider)myGameObject.gameObject.AddComponent (typeof (CapsuleCollider));

// Make the Capsule Collider into a Trigger Collider.
myCC.isTrigger= true;

// Set the center of the Capsule Collider to the center of the GameObject.
myCC.center = Vector3.zero;

// Make the Sphere Collider twice as tall.
myCC.height= 2;

// Make the Sphere Collider twice as wide.
myCC.radius= 2;

// Set the axis of lengthwise orientation to the X axis.
myCC.direction = 0;

// Set the axis of lengthwise orientation to the Y axis.
myCC.direction = 1;

// Set the axis of lengthwise orientation to the Y axis.
myCC.direction = 2;
```

ホイールコライダー

プロパティ

- - ホイールコライダーの
- **Radius** - ローカルの
- - の
- サスペンション - ローカルでのYにつた
- アプリポイントのをする - がされるポイント、
- センター - ローカルスペースのホイールコライダーの

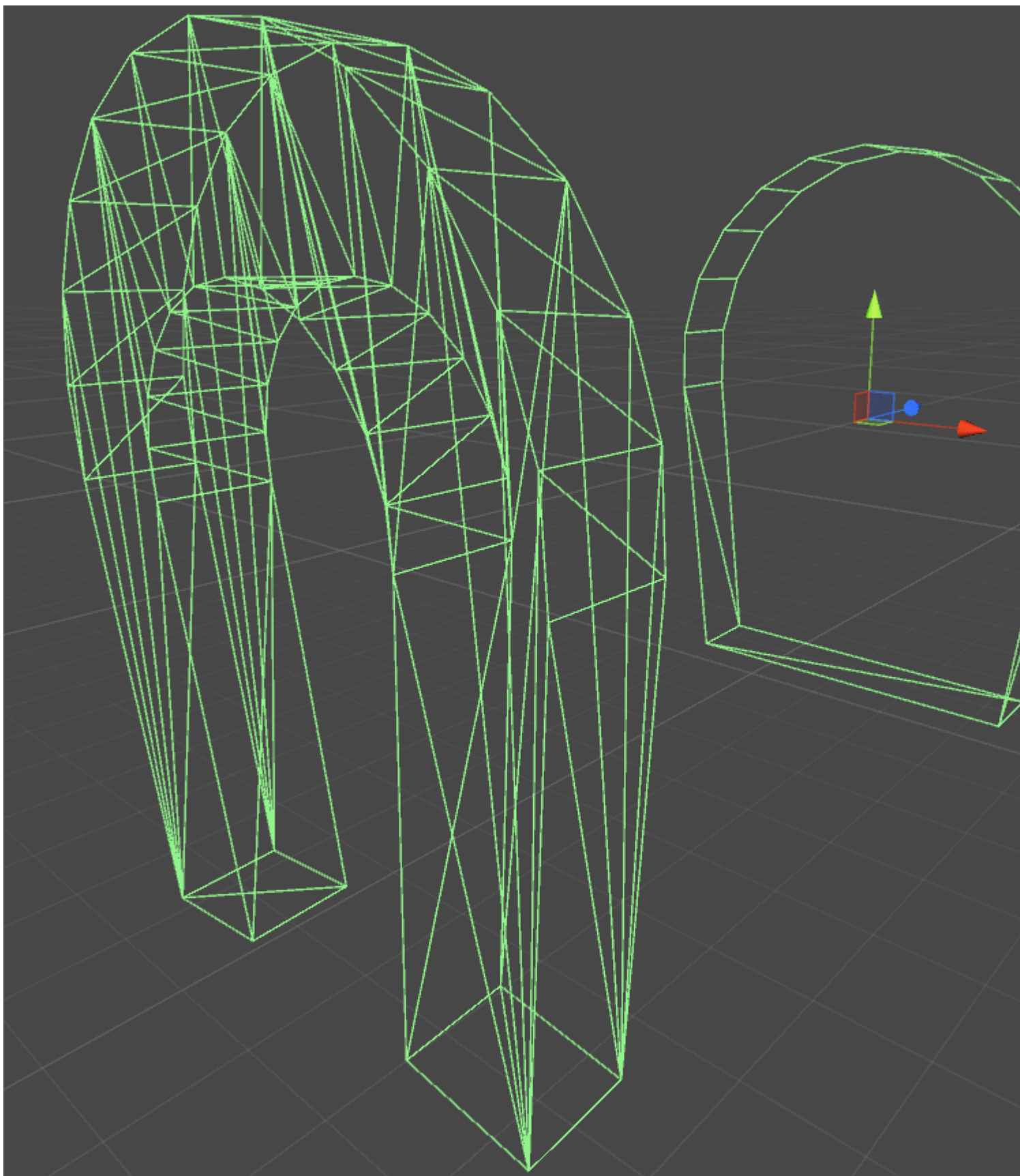
サスペンションスプリング

- **Spring** - Wheelがにろうとする

- ダンパー - がきいほどがし、サスペンションがくなります
- - デフォルトは0.5、0はサスペンションがボトムアウト、1が
- 1の - またはにがるときのタイヤの

メッシュコライダー

メッシュにづくコライダー。

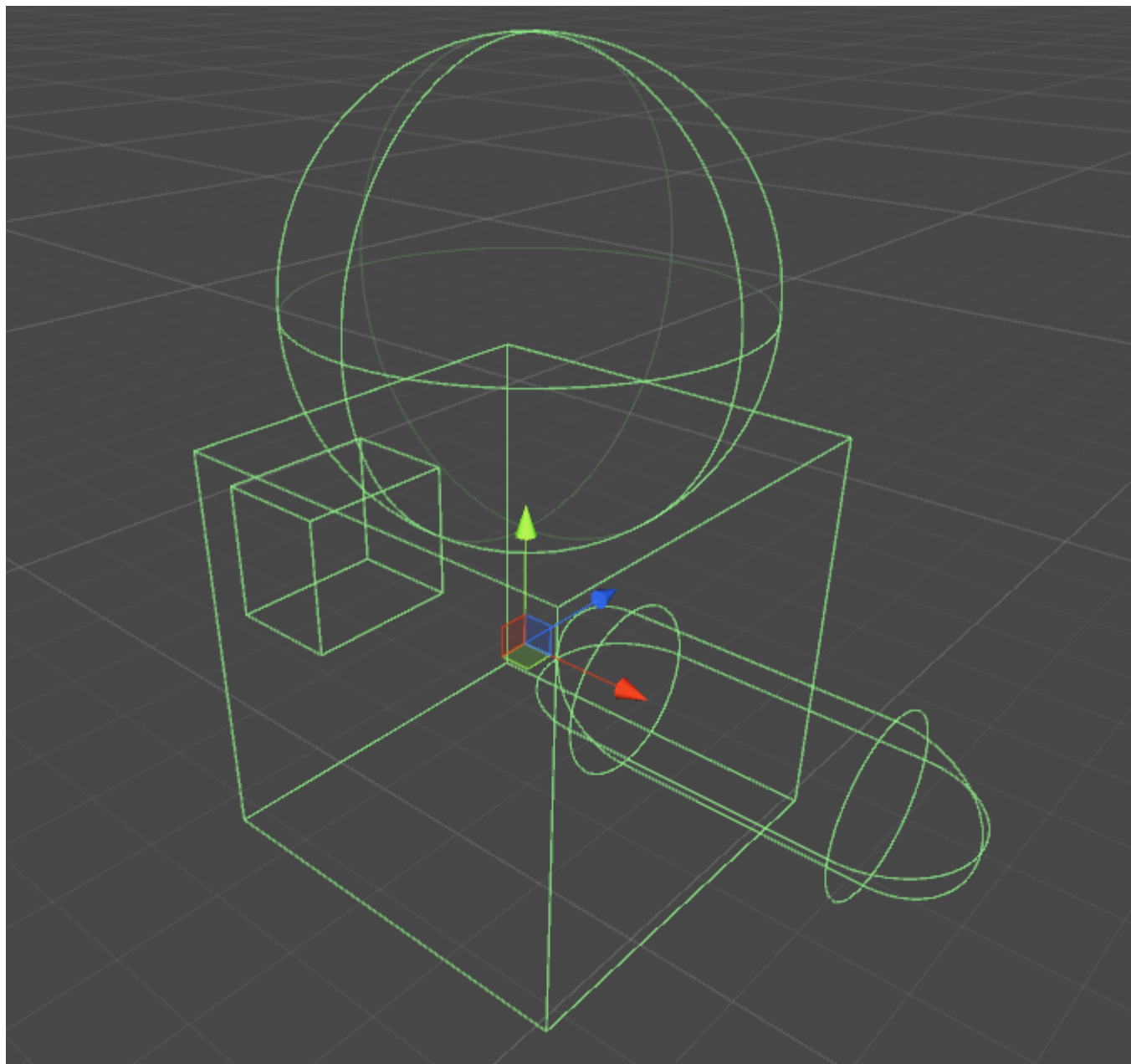


プロパティ

- **Is Trigger** - チェックをされると、Box Colliderはをしてトリガーコライダーになります

- マテリアル - されているは、Box Colliderのマテリアルへの
- **Mesh** - Colliderがついているメッシュへの
- **Convex** - Convex Meshのコライダーは255ポリゴンにされています。にすると、このColliderはのメッシュコライダーとするがあります

GameObjectにのColliderをするは、それをCompound Colliderとびます。



ホイールコライダー

ホイールコライダーのには、NvidiaのPhysXホイールコライダーがみまれているため、くののがあります。には「のない」プログラムですが、すべてをするために、いくつかのながです。

プロパティ

-

- ホイールのキログラム。これはホイールモーメントとのインターアモーメントにされます

。

- - メートルで、の。
- Wheel Damping Rate - がトルクにどれくらいするかをします。
- サスペンション - ができる
- Force App Point Distance - のにえられたサスペンションからの
- Center - ホイールの

サスペンションの

- - これは、バネKニュートン/メートルをでしたものです。

=ばね*

こののいは、あなたののを、のでり、50から100までのでしたてなければなりません。えは、が4つの2,000kgのがあれば、500kgをサポートします。これに75をけ、あなたのばねは37,500ニュートン/メートルでなければなりません。

- ダンパー - のショックアブソーバーにします。サスペンションがいほどレートがくなり、レートをくするほど「らかく」なり、するがくなります。はこれのやをらない、はそれがのとしているとう。

サイドウェイ

1つのは、からのまでののりm/sによってされるスリップをする。

- すべり - これは、をうにがることができるm/sです
- Extremum Value - これは、ホイールにされるべきのです。

Extremum Slipのは、ほとんどのなの、0.2mから2m/sのでなければなりません。2m/sは6フィート、または5mphです。これはくのスリップです。スリップのために2m/sのをつがあるとじるは、をやすことをするがあります。

Max Fraction Extremum Valueは、のです。

ニュートン=*きニュートン

これは、1ので、スリップの+サスペンションのをしていることをします。のアプリケーションでは、1よりきいはまれですが、ではありません。したアスファルトのタイヤの、0.7と.9ののはななので、デフォルトの1.0がましい。

このは、ながしめるため、には2.5をえてはなりません。えは、あなたはにがりめますが、このがにいので、あなたのとはにきなえられ、あなたはれているわりにターンにりめます。

のをにしたは、とをげめるがあります。りは、0.5m/sから2m/sのでなければならず、すべりをえるすべりにするをする。トラクションがれるまでがうまくすることがわかった、はにあるよう

にしますが、をけるがあります。あなたのがドリフトすることができないとかったら、をけるべきです。

のはのとじですが、これはがどれほどののをにつかをしています。がすぎると、はえきてタイヤをさせてから、にゆっくりとします。もしそれがすぎると、あなたのは、ひどい、またはいことをやってみるがあります。

そのの

これらのをするだけで、GTAクローンやのレースクローンをすることはできません。ほとんどのゲームでは、、、およびがなるため、これらのがスクリプトでにされています。さらに、キーがされているときにホイールコライダーにのトルクをえるだけでは、ゲームはにしません。のでは、はにえられるトルクをさせるトルクとトランスミッションをとっています。

のをるには、のがになるまでこれらのをしてから、ホイールトルク、、、およびスクリプトのをするがあります。

ホイールのにするは、Nvidiaのドキュメント

<http://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/Vehicles.html>をしてください。

トリガー・コライダー

メソッド

- OnTriggerEnter()
- OnTriggerStay()
- OnTriggerExit()

OnTriggerEnter()、 OnTriggerStay()およびOnTriggerExit()メソッドをするには、Colliderをトリガーにすることができます。トリガーコライダーはににせず、のゲームオブジェクトはにそれをします。それらは、アイテムをするときなど、のGameObjectがのにあるかどうかをするのにです

。

Trigger Collider Scripting

のメソッドは、のコライダーがGameObjectのコライダープレイヤーなどになるタイミングをするトリガーリスナーのです。トリガーメソッドは、GameObjectにりてられているのスクリプトにできます。

```
void OnTriggerEnter(Collider other)
{
    //Check collider for specific properties (Such as tag=item or has component=item)
}
```


オンラインでもむ <https://riptutorial.com/ja/unity3d/topic/4405/>

40: およびポストプロセッサ

- AssetPostprocessor.OnPreprocessTexture

String.Contains() をして、されたをアセット・パスにつアセットのみをします。

```
if (assetPath.Contains("ProcessThisFolder"))
{
    // Process asset
}
```

Examples

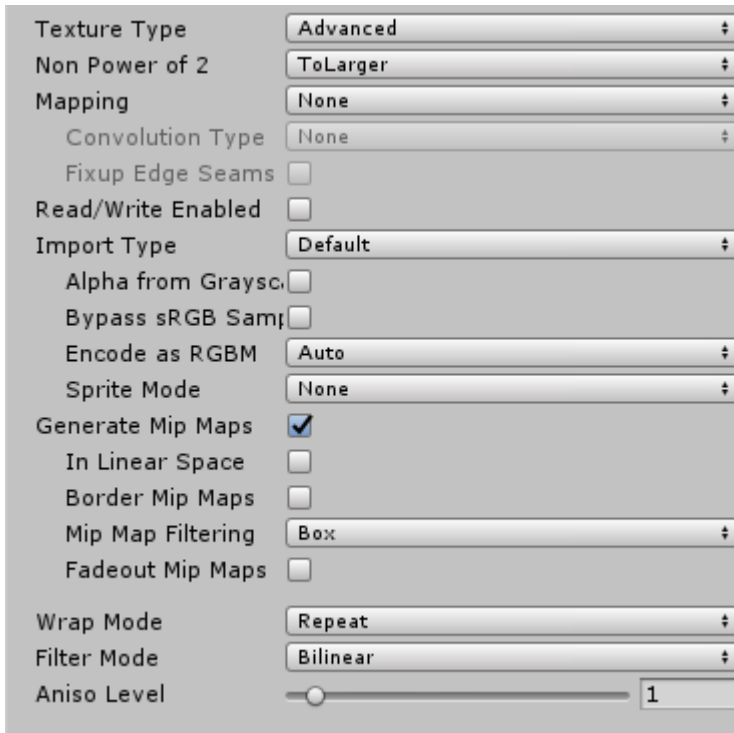
テクスチャポストプロセッサ

Assets フォルダののに TexturePostProcessor.cs ファイルをします。

```
using UnityEngine;
using UnityEditor;

public class TexturePostProcessor : AssetPostprocessor
{
    void OnPostprocessTexture(Texture2D texture)
    {
        TextureImporter importer = assetImporter as TextureImporter;
        importer.anisoLevel = 1;
        importer.filterMode = FilterMode.Bilinear;
        importer.mipmapEnabled = true;
        importer.npotScale = TextureImporterNPOTScale.ToLarger;
        importer.textureType = TextureImporterType.Advanced;
    }
}
```

、Unityがテクスチャをインポートするたびに、のパラメータがあります



ポストプロセッサをする、エディターでのインポートをしてテクスチャーパラメータをすることはできません。

[]ボタンをすとテクスチャがインポートされ、ポストプロセッサコードがひされます。

な

インポータをするカスタムファイルがあるとします。それは.xlsファイルなんでもかまいません。これはJSONファイルをししますが、これはですが、カスタムをして、どのファイルがたちのものであるかをにできるようにします。

JSONファイルのが

```
{
  "someValue": 123,
  "someOtherValue": 456.297,
  "someBoolValue": true,
  "someStringValue": "this is a string",
}
```

Example.testとしてののどこかにしましょう。

に、データの-customクラスをつMonoBehaviourをします。Customクラスは、JSONのシリアルをにするためのものです。Customクラスをするはありませんが、このはくなります。これを

TestData.cs し TestData.cs

```
using UnityEngine;
using System.Collections;

public class TestData : MonoBehaviour {
```

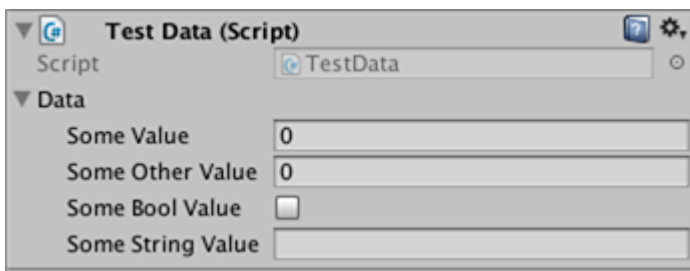
```

[System.Serializable]
public class Data {
    public int someValue = 0;
    public float someOtherValue = 0.0f;
    public bool someBoolValue = false;
    public string someStringValue = "";
}

public Data data = new Data();
}

```

そのスクリプトをGameObjectにするのは、



に、AssetsにEditorフォルダをします。はどんなレベルでもよい。エディターフォルダーにTestDataAssetPostprocessor.cs ファイルをし、これをします。

```

using UnityEditor;
using UnityEngine;
using System.Collections;

public class TestDataAssetPostprocessor : AssetPostprocessor
{
    const string s_extension = ".test";

    // NOTE: Paths start with "Assets/"
    static bool IsFileWeCareAbout(string path)
    {
        return System.IO.Path.GetExtension(path).Equals(
            s_extension,
            System.StringComparison.Ordinal);
    }

    static void HandleAddedOrChangedFile(string path)
    {
        string text = System.IO.File.ReadAllText(path);
        // should we check for error if the file can't be parsed?
        TestData.Data newData = JsonUtility.FromJson<TestData.Data>(text);

        string prefabPath = path + ".prefab";
        // Get the existing prefab
        GameObject existingPrefab =
            AssetDatabase.LoadAssetAtPath(prefabPath, typeof(Object)) as GameObject;
        if (!existingPrefab)
        {
            // If no prefab exists make one
            GameObject newGameObject = new GameObject();
            newGameObject.AddComponent<TestData>();
            PrefabUtility.CreatePrefab(prefabPath,
                newGameObject,

```

```

        ReplacePrefabOptions.Default);
    GameObject.DestroyImmediate(newGameObject);
    existingPrefab =
        AssetDatabase.LoadAssetAtPath(prefabPath, typeof(Object)) as GameObject;
}

TestData testData = existingPrefab.GetComponent<TestData>();
if (testData != null)
{
    testData.data = newData;
    EditorUtility.SetDirty(existingPrefab);
}
}

static void HandleRemovedFile(string path)
{
    // Decide what you want to do here. If the source file is removed
    // do you want to delete the prefab? Maybe ask if you'd like to
    // remove the prefab?
    // NOTE: Because you might get many calls (like you deleted a
    // subfolder full of .test files you might want to get all the
    // filenames and ask all at once ("delete all these prefabs?").
}

static void OnPostprocessAllAssets (string[] importedAssets, string[] deletedAssets,
string[] movedAssets, string[] movedFromAssetPaths)
{
    foreach (var path in importedAssets)
    {
        if (IsFileWeCareAbout(path))
        {
            HandleAddedOrChangedFile(path);
        }
    }

    foreach (var path in deletedAssets)
    {
        if (IsFileWeCareAbout(path))
        {
            HandleRemovedFile(path);
        }
    }

    for (var ii = 0; ii < movedAssets.Length; ++ii)
    {
        string srcStr = movedFromAssetPaths[ii];
        string dstStr = movedAssets[ii];

        // the source was moved, let's move the corresponding prefab
        // NOTE: We don't handle the case if there already being
        // a prefab of the same name at the destination
        string srcPrefabPath = srcStr + ".prefab";
        string dstPrefabPath = dstStr + ".prefab";

        AssetDatabase.MoveAsset(srcPrefabPath, dstPrefabPath);
    }
}
}
}

```

これをすると、でした `Example.test` ファイルを Unity Assets フォルダにドラッグアンドドロップで

き、するプレハブがされているはずでず。 `Example.test` をすると、プレハブのデータがすぐにされることわかります。プレハブをシーンにドラッグすると、それがされ、 `Example.test` がされます。 `Example.test` をのフォルダにすると、するプレハブもにします。インスタンスのフィールドをした、 `Example.test` ファイルをすると、インスタンスでしなかつたフィールドのみがされま

す。
のでは、 `Example.test` を `Assets` フォルダにドラッグすると、 `Example.test.prefab` と `Example.test` のがされます。モデルのインポーターがうまくするようになっておくといいでしよう。のように `Example.test` しかされず、 `AssetBundle` やそのようなものです。あなたがそのをどのようにしているかかっているなら

オンラインでおよびポストプロセッサをむ <https://riptutorial.com/ja/unity3d/topic/5279/および-ポスト-プロセッサ>

クレジット

S. No		Contributors
1	unity3dのい	Alexey Shimansky , Chris McFarland , Community , Desutoroiya , driconmax , F̃l̃ámínġ óm̃bíé , James Radvan , josephsw , Linus Juhlin , Luís Fonseca , Maarten Bicknese , martinhodler , matiaslauriti , Mike B , Minzkraut , PlanetVaster , R.K123 , S. Tarik Çetin , Skyblade , SourabhV , SP. , tenpn , tim , user3071284
2	Androidプラグイン 101 - はじめに	Venkat at Axiom Studios
3	CullingGroup API	volvis
4	MonoBehaviourクラ スの	matiaslauriti , Skyblade , Thundernerd , user3797758
5	ScriptableObject	volvis
6	Unity Profiler	Amitayu Chakraborty , ForceMagic , RamenChef , Skyblade
7	UnityでGitソースコ ントロールをする	Commodore Yournero , Hacky , James Radvan , matiaslauriti , Max Yankov , Maxim Kamalov , Pierrick Bignet , Ricardo Amores , S. Tarik Çetin , S.Richmond , Skyblade , Thulani Chivandikwa , YsenGrimm , yummypasta
8	Unityのシングルトン	David Darias , Fehr , James Radvan , JohnTube , matiaslauriti , Maxim Kamalov , Simon Heinen , SP. , Tiziano Coroneo , Umair M , volvis , Zze ,
9	Vector3	driconmax , F̃l̃ámínġ óm̃bíé , Gnemlock
10	アセットストア	JakeD , Trent , zwcloud
11	アセットパッケージ の	F̃l̃ámínġ óm̃bíé
12	エディタの	Pierrick Bignet , Skyblade , Thundernerd , lolæz əɥl qoq , volvis
13	オーディオシステム	R4mbi , lolæz əɥl qoq
14	オブジェクトプーリ ング	Chris McFarland , Ed Marty , lase , matiaslauriti , S. Tarik Çetin , Thulani Chivandikwa , Thundernerd , lolæz əɥl qoq , volvis

15	クオータニオン	matiaslauriti , Tiziano Coroneo , Xander Luciano , yummypasta
16	ゲームオブジェクトのと	Pierrick Bignet , S. Tarık Çetin , volvis
17	コルーチン	agiro , Fattie , Fehr , Giuseppe De Francesco , Problematic , Skyblade , Thulani Chivandikwa , Thundernerd , łolæz əuʔ qoq , volvis
18	サーバーとの	David Martinez , devon t , ĔĴámínġ ómċbíé , Maxim Kamalov , tim
19	タグ	Arijoon , Augure , glaubergft , Gnemlock , MadJlzz , Skyblade , Trent
20	デザインパターン	Ian Newland
21	ネットワーキング	David Martinez , driconmax , Rafiwui , RamenChef
22	バーチャルリアリティVR	4444 , Airwarfare , Guglie , pew. , Pratham Sehgal , tim
23	プレハブ	Brandon Mintern , Dávid Florek , ĔĴámínġ ómċbíé , gman , Gnemlock , Guglie , James Radvan , Jean Vitor , josephsw , Lich , matiaslauriti , Skyblade , Thulani Chivandikwa , łolæz əuʔ qoq , Woltus , yummypasta
24	マルチプラットフォーム	user3797758 , volvis
25	モバイルプラットフォーム	Airwarfare , Skyblade
26	ユーザーインターフェイスシステムUI	Helium , matiaslauriti , Maxim Kamalov , Programmer , RamenChef , Skyblade , Umair M
27	ユニティアニメーション	4444 , Fiery Raccoon , Guglie
28	ユニティライティング	ĔĴámínġ ómċbíé
29	リソース	glaubergft , MadJlzz , Skyblade , Venkat at Axiom Studios
30	レイキャスト	driconmax , Meinkraft , Skyblade , user3570542 , volvis , wouterrobot
31	レイヤー	Arijoon , dreadnought , Light Drake , RamenChef , Skyblade

32	システム	Programmer , Skyblade , lolæz əʊʌ qoq
33	モードグラフィカル ユーザーインターフ ェイスシステム IMGUI	Skyblade , Soaring Code
34		ADB , Jean Vitor , matiaslauriti , S. Tarık Çetin , Skyblade , Thundernerd , Xander Luciano
35		4444 , Thundernerd
36	の	lolæz əʊʌ qoq
37		Ed Marty , EvilTak , F̃l̃ámínġ óm̃bíé , Grigory , JohnTube , Skyblade , Thulani Chivandikwa , volvis
38		eunoia , F̃l̃ámínġ óm̃bíé , jack jay
39		F̃l̃ámínġ óm̃bíé , jjhavokk , Xander Luciano
40	およびポストプロセ ッサ	gman , Skyblade , volvis