

Unity Script の基本と Unity4.x アセット活用術

1 : Unity Script の構文とクラスと各関数

●スクリプトを書くエディター（MonoDevelop）の起動方法

まず、基本的な考えとして、「Unity」ではプログラミングをしないと思っておいってください。Unity が持っているのは、「C#や Javascript のファイルを作成する機能」と、作成したプログラムをコンポーネントとしてゲームオブジェクトに組み込んだりする機能だけしかありません。

実際のプログラミングは何で行うかと言うと、Unity と一緒にインストールされる「MonoDevelop」、というアプリケーションで行います。しかし、必ずしも、「MonoDevelop」を使う必要はなく、「Visual Studio 2013」でもプログラミングできますし、普通にエディターでソースを開いて編集しても構いません。

ただし、今回の書籍では、Unity に専用で付属している「MonoDevelop」アプリケーションを使用します。

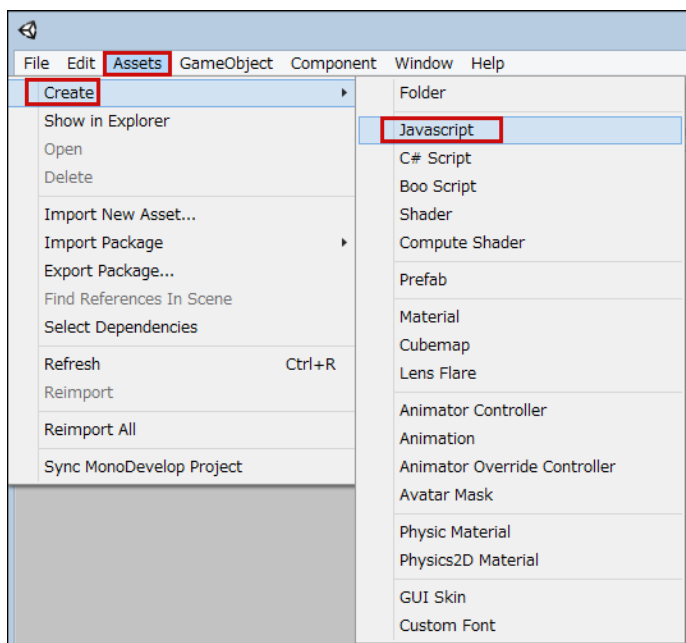
では、「MonoDevelop」の起動方法を紹介します。

●MonoDevelop の起動方法

起動方法はいろいろあるのですが、まだ Unity の画面上に、何もオブジェクトが配置されていない状態で、「MonoDevelop」を起動します。

Unity メニューの [Assets] – [Create] – [Javascript] と選択します（図 1-1）。

図 1-1 「Javascript」ファイルを作成する手順



すると、Unity の画面下にある「Assets」フォルダー内に、「NewBehaviourScript.js」という「Javascript」ファイルが作成されます。このファイル名をクリックすると編集状態になり、ファイル名を変更することができます。今回はデフォルトのファイル名のままで進

めます。

この「NewBehaviourScript.js」をダブルクリックします。すると「MonoDevelop」が起動します（図 1-2）。

図 1-2 「MonoDevelop」が起動した

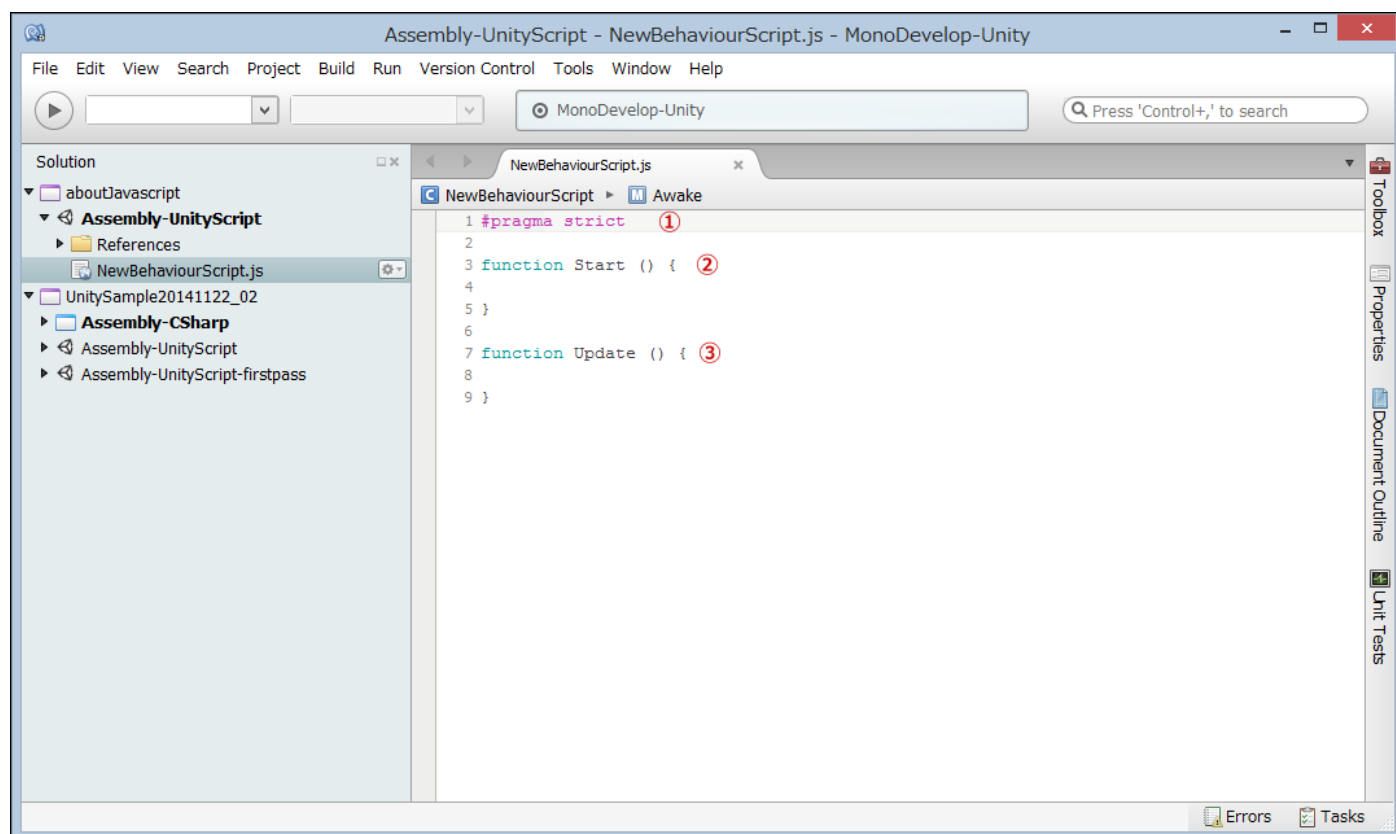


図 1-2 の内容を見ていきましょう。

① #pragma strict

これは、Unity Javascript 特有の文で、「厳格モード」でコーディングすることを示すプラグマディレクティブ（コンパイル時に実行する命令）です。「厳格モード」とは「変数などの曖昧な書き方をチェックして禁止する」という意味です。「Unity Script」には先頭にこれが必要なんだと覚えておくだけでいいでしょう。

② Start 関数

Start 関数は、スクリプトを組み込んだゲームオブジェクトのあるシーンが、表示される際に発生するイベントです。この Start 関数に処理を書いておけば、シーンが開始された場合に、それを実行します。

③ Update 関数

この関数が、Unity のイベントの中でも、もっとも重要な役割を果たします。Update 関数は、Unity のフレームが切り替わるごとに発生するイベントです。ここに処理を記述することで、画面が更新される毎に処理を実行させることができます。

Unity は、「フレーム」と呼ばれる表示を高速で切り替えて動いています。このフレームが切り替わる際に実行されるのが Update 関数です。

Unity の Script では、とりあえずは、この Start 関数と Update 関数内に、処理を書く！と覚えておくといいでしょう。

では試しに、「Start 関数」と「Update 関数」内に、何か文章を表示させるコードを記述して、「Console」に表示させてみましょう。

●コードで試してみよう！

リスト 1 のようなコードを書いてみました。

リスト 1 (NewBehaviourScript.js)

```
function Start () {  
    Debug.Log("はじめての Unity");  
}
```

```
function Update () {  
    Debug.Log("こんにちは Unity");  
}
```

`Debug.Log("表示させる文字");`の構文でコンソールに指定した文字を表示します。

「MonoDevelop」メニューの [Build] – [Build All] と実行してください。エラーが表示された場合は修正します。

では、これを Unity に組み込んで表示させてみましょう。

Unity メニューから [GameObject] – [Create Empty] と選択し、「空の GameObject」を「Hierarchy」内に作成します。「Hierarchy」から「GameObject」を選択して、「Inspector」を表示し、「Add Component」ボタンから [Scripts] – [New Behaviour Script] を選択します。

すると、「GameObject」に「New Behaviour Script (Script)」が追加されます。

これで実行して、「Console」画面でみてみましょう。図 1-3 のように表示されます。

図 1-3 「Start 関数」と「Update 関数」内の文字が表示された

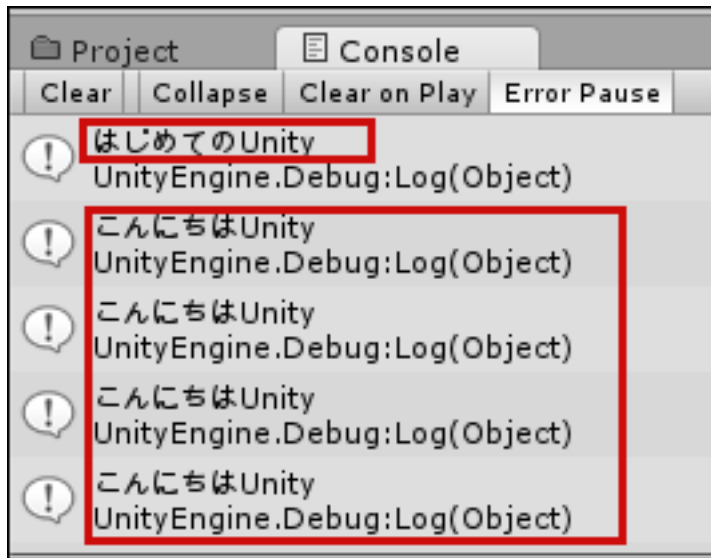


図 1-3 を見るとわかるように、「Start 関数」内に記述した内容は、最初の 1 回だけ呼び出されています。「Update」関数内に記述した内容は、ずっと下まで同じ文章が表示され続けています。

Update 関数は、Unity のフレームが切り替わるごとに発生するイベントですので、このように常に呼び出されています。

「Start 関数」と「Update 関数」の違いがお分かりいただけたと思います。

●変数の宣言

Unity Script の変数宣言は、普通の Javascript とは、少し異なります。

```
Private var no:int=10;
```

通常の Javascript では変数の型宣言は行わないのですが、Unity Script では：（コロン）の後に型を宣言します。

```
var 変数名:型名=初期化内容;
```

という書式になります。

また、それぞれの変数には、「アクセス修飾子」を指定することができます。

●変数のアクセス修飾子

public var 変数名:型名; ①

private var 変数名:型名; ②

static var 変数名:型名; ③

①：Unity では、関数の外で宣言された変数はメンバー変数になります。この変数は Unity の「Inspector」からアクセス出来ます。メンバー変数に格納された値は、プロジェクトと同時に保存されます。また、「Public」を指定した変数も、「Inspector」内にパラメーターとして表示され、値を設定することが可能になります。

②：private メンバー変数はスクリプト外部に表示したくない状態を保持しておくのに使います。private メンバー変数はディスクに保存されず、「Inspector」から操作することは出来ません。

③：static キーワードを使うと、グローバル変数になります。

●変数の型

	型	変数の種類
一般的な変数	int	整数
	float	不動少数点型
	Boolean	ブール型
	String	文字列型
	Array	配列型
Unity 独自変数	Transform	オブジェクトの位置、回転、サイズを扱う
	GameObject	Unity での基盤となる型
	Rigidbody	物理演算によりオブジェクトの位置をコントロールし、重力を付けたり、オブジェクト同士の衝突判定を行うことができるようになる
	Animation	アニメーションデータ
	Collider	衝突したオブジェクトの情報を扱う
	Camera	ゲームのプレイ画面を表示するカメラを扱う

●物理挙動

Rigidbody の役割

GameObject が物理特性の制御下で動作するようになります。Rigidbody は、力やトルクを受け、現実的な方向にオブジェクトを動かすことができます。GameObject は、重力の影響を受ける Rigidbody を含めるか、スクリプティングを通じて加えた力の下で動作するか、物理特性エンジンを通じて、その他のオブジェクトと相互作用する必要があります。

簡単にいうと

- ・重力や加速度を使った処理ができる。
- ・衝突した物体を動かすことができる。
- ・衝突判定時に、任意の処理を実行できる

ということです。

設定項目には下記のものがあります。

項目	意味
Mass	オブジェクトの質量 (単位 : kg)
Drag	力により動く際に、オブジェクトに影響する空気抵抗の量
Angular Drag	トルクにより回転する際に、オブジェクトに影響する空気抵抗の量

Use Gravity	有効にすると、オブジェクトは重力の影響を受ける
Is Kinematic	有効にすると、オブジェクトは物理特性エンジンによって駆動されませんが、その Transform によってのみ操作できる
Interpolate	フレーム間の補間
Collision Detection	衝突検知モード
Constraints	リジッドボディの動きに関する制限。移動や回転を軸ごとに無効化する

● 接触判定

Collider の役割

GameObject 同士の衝突判定をするための機能として、「Collider」が用意されています。「Collider」を追加することで、GameObject 同士が衝突したかどうかの判定ができます。

例をあげましょう。

まず「Hierarchy」の「[Create] - [3D Object] - [Plane]」を選択して「Scene」画面に「Plane」を追加してください。同じ手順で「Cube（立方体）」も追加します。「Inspector」から「Transform」の「Scale」の値を操作してサイズを広くしておいてください。

同じ手順で「Sphere（球体）」も追加します。「[Light]」から「[Directional Light]」も追加してください。

「Sphere」は少し上方に配置し、「Camera Preview」にうまく収まるよう「Main Camera」の位置を設定してください。筆者は図 1-4 のように配置しました。

図 1-4 「Cube」と「Sphere」を配置した

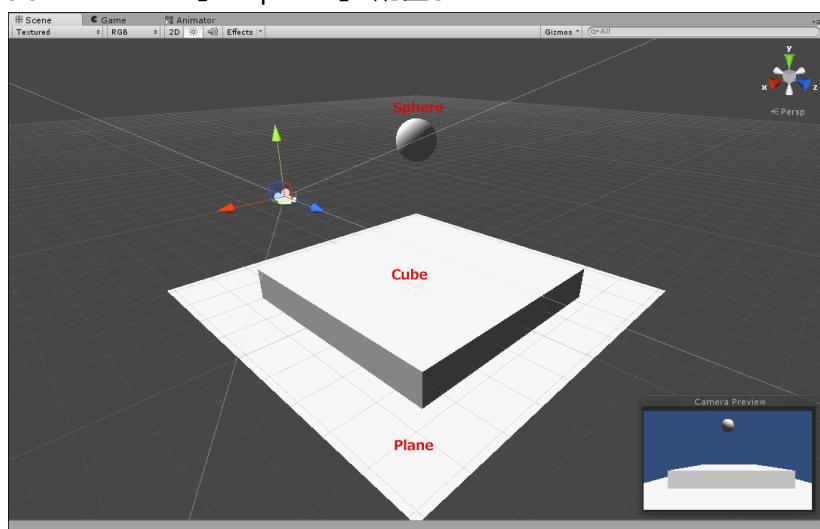
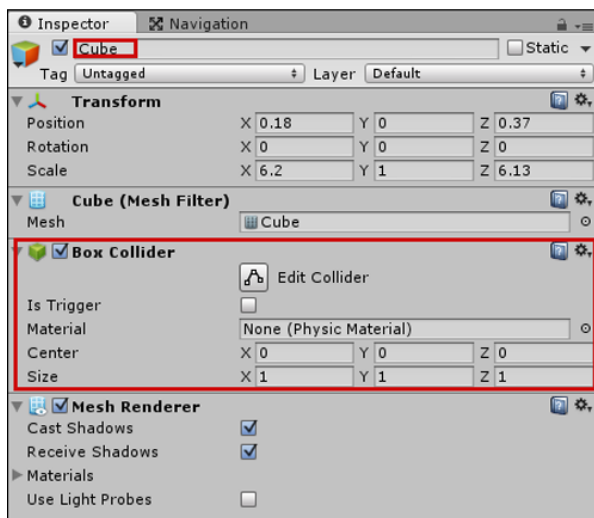
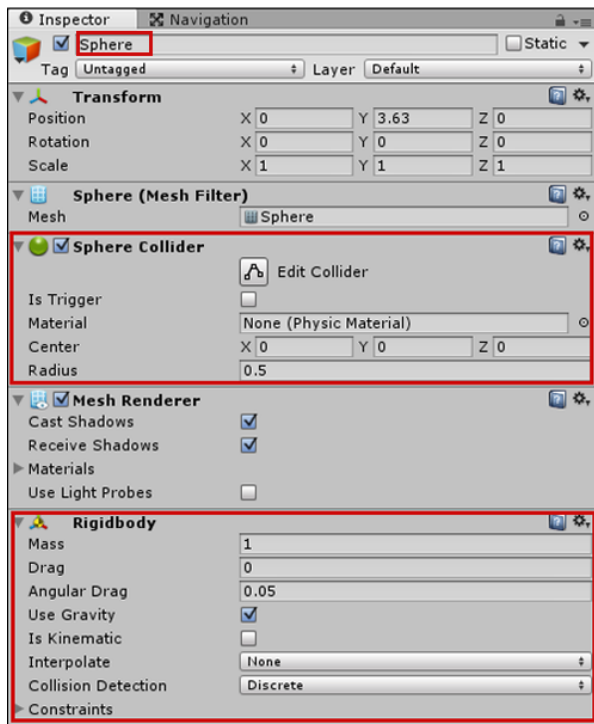


図 1-4 で配置した、「Cube」と「Sphere」の「Inspector」を見ると、どちらにも、自動的に「Collider」が追加されています。「Sphere」を選択し「Add Component」ボタンから「[Physics] - [Rigidbody]」を選択して下さい（図 1-5）。

図 1-5 「Sphere（上図）」に「Collider」と「Rigidbody」が、「Cube（下図）」に「Collider」が追加されている



「Sphere」の「Rigidbody」の「Use Gravity」にチェックが入っているのを確認してください。ここにチェックが入っていないと「Sphere」は落下しません。

では、次に「衝突判定」のスクリプトを書いていきます。

「Sphere」を選択し、「Add Component」から [New Script] と選択し、「Name」に「OnCollisionEnterScript」と指定し、「Language」に「Java Script」を選択して、「Create and Add」をクリックします。「Sphere」の「Inspector」内に「On Collision Enter Script (Script)」が追加されますので、「Script」の「OnCollisionEnterScript」をダブルクリックして、「MonoDevelop」を起動して、リストと 1 のコードを記述してください。

リスト 1 「Sphere」と「Plane」が衝突した際の処理 (OnCollisionEnterScript.js)

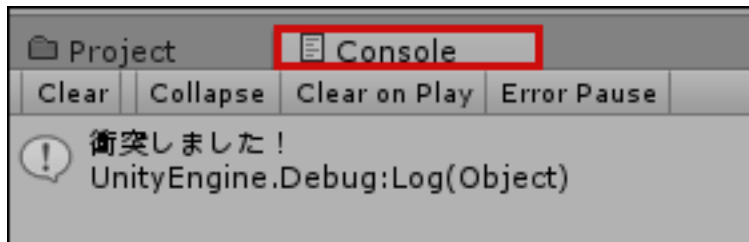
```
function OnCollisionEnter(col : Collision){
    if(col.gameObject.name == "Cube"){
        Debug.Log("衝突しました！");
    }
}
```

```
}  
}
```

OnCollisionEnter で衝突判定を行います。引数 col には衝突情報が格納されています。

衝突したゲームオブジェクトの「name」が「Cube」であった場合は、コンソールに「衝突しました！」と表示させます（図 1-6）。

図 1-6 実行して「Sphere」が「Cube」に衝突した(zu_6.png)



この衝突判定を行う上で、重要な点が 2 点あります。

それが、「Rigidbody」と、「Collider」です。

■「Rigidbody」が必要理由

「Rigidbody」によって、「GameObject」が物理特性の制御下で動作するようになるからです。

■「Collider」が必要な理由

Collider は Rigidbody の物理特性の境界を定義するからです。

つまり、「Rigidbody」で、物理的影響の下で動くようになり、「Collider」で、衝突したかどうかを確認するということです。

図 5 を見るとわかりますが、「Collider」には「Box Collider」、「Sphere Collider」があります。他に「Capsule Collider」や「Mesh Collider」等があります。「Collider」の詳細については、下記の URL の「コライダ」の個所を参照ください。

<http://docs-jp.unity3d.com/Documentation/Components/class-Rigidbody.html>

●MonoBehaviour クラス

Unity でスクリプトを書く上で基本となるクラスです。このクラスには、「Awake」、「Start」、「Update」などの関数が含まれます。

■ 各関数の種類

●Awake 関数

スクリプトが最初に読み込まれるとき、一度だけ Awake 関数が呼び出されます。変数などの初期化する場合に適しています。

構文

```
function Awake(){  
    スクリプトが最初に読み込まれるとき、一度だけ呼ばれる処理  
}
```

●Start 関数

Awake 関数で初期化がなされた後、スクリプトを読み込む際に 1 回だけ実行される関数です。

構文

function Start(){

スクリプトを読み込む際に 1 回だけ実行される処理

}

●Update 関数

毎フレームが呼び出される関数です。オブジェクトを動かす処理はここの中に書きます。フレームとは、ゲームを描画する際の 1 コマ 1 コマを指します。だいたい 1 秒間に 60 フレーム、あるいは 50 フレーム、30 フレームなどが一般的です。

構文

function Update(){

毎フレームが呼び出される処理

}

●OnTriggerEnter

他の衝突がトリガーに入った瞬間に呼び出されます。トリガーとなるゲームオブジェクトの「Is Trigger」には必ずチェックを入れておきます。

リスト 2 のようなコードを書いて 3D キャラクターに追加します。

リスト 2 OnTriggerEnter を発生させるコード(OntriggerEnterScript.js)

```
function OnTriggerEnter (col : Collider) {  
    if(col.gameObject.name=="Cube")  
    {  
        Debug.Log("衝突しました！");  
    }  
}
```

3D キャラクターが「Cube」と接触した瞬間に、一度だけ「衝突しました！」と表示されます。

これを実行すると動画 1 のようになります。左隅下の「Console」の箇所を見ていてください。3D キャラクターが「Cube」と接触した瞬間「衝突しました！」と表示され、3D キャラクターは「Cube」をすり抜けてしまいます。

動画 1

<http://youtu.be/f2YVnXwSHvw>

「Console」に「衝突しました！」と表示されたのがわかったと思います。

●OnCollisionEnter

「●接触判定」の「Collider の役割」で解説していますので、そちらを参照してください。

OnTriggerEnter と OnCollisionEnter の違い

OnTriggerEnter は「Inspector」内の「Is Trigger」にチェックが入っており、Collider や Rigidbody を持つゲームオブジェクト

に「接触した瞬間」に呼び出される関数です。

OnCollisonEnter は、Collider や RigidBody を持つゲームオブジェクトが、「衝突」した時に呼び出される関数です。

● OnControllerColliderHit

移動中にコントローラーが Collider オブジェクトに衝突したときに呼び出される関数です。

リスト 3 のようなコードを書いて 3D キャラクターに追加します。

リスト 3 「OnControllerColliderHit」イベントを発生させるコード(OnControllerColliderHitScript.js)

```
function OnControllerColliderHit(hit:ControllerColliderHit){  
    if(hit.gameObject.tag=="Player")  
    {  
        Debug.Log("ヒットしました！");  
    }  
}
```

「Inspector」の「Tag」名が「Player」であるゲームオブジェクトに衝突した際に、「ヒットしました！」と、「Console」に表示されます。

※Tag の設定方法

「Inspector」内の「Tag」で「Untagged」と表示されている上下「▲」アイコンをクリックします。すると、登録されている「Tag」名が表示されます。もし「Player」という「Tag」名が登録されていない場合は、「Add Tag」から登録してください。登録されている場合は「Player」を選択します。

リスト 3 を実行すると動画 2 のようになります。左隅下の「Console」の個所を見ていてください。3D キャラクターが、別な「3D キャラクター」と接触した瞬間「ヒットしました！」と表示されます。

動画 2

<http://youtu.be/Ixv2hq6UKUA>

3D キャラクター同士が衝突した際に、「Console」に「ヒットしました！」と表示されます。

● OnGUI 関数

UnityGUI コントロールは、OnGUI と呼ばれる特殊な関数を使用します。OnGUI 関数は、Update 関数同様、含んでいるスクリプトが有効になるたびに呼び出されます。ボタンやラベル、入力ボックス、スライダーなどを作成できます。

構文

```
function OnGUI(){  
    GUI 要素を作成する処理  
    GUI 要素が作成された時の処理  
}
```

ボタンを作成するコードはリスト 4 になります。

リスト 4 ボタンを作成し、ボタンをクリックした時の処理(OnGUIScript.js)

```
function OnGUI()  
{  
    if ( GUI.Button( Rect(100, 100, 100, 20), "Button" ) )  
    {  
        Debug.Log("Button がおされました");  
    }  
}
```

このコードを、Unity メニューの [GameObject] – [Create Empty] と選択して、空の「GameObject」を作成し、作成したリスト 4 のスクリプトを「Hierarchy」内の「GameObject」にドラッグ & ドロップしてください。

リスト 4 を実行すると動画 3 になります。「Button」を押すと、「Console」に「Button が押されました」と表示されます。

動画 3

http://youtu.be/7n_SdBS9BPE