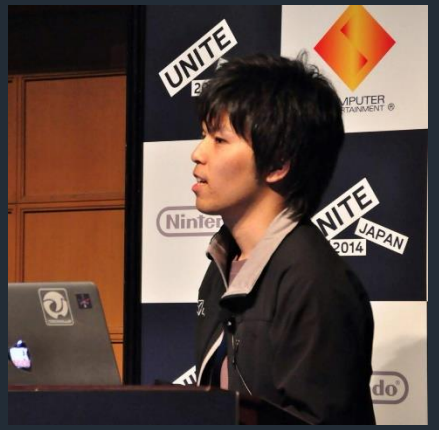
エディター拡張マニアクス 2015

あんどう けいご

安藤 圭吾

ユニティ・テクノロジーズ・ジャパン合同会社



フィールドエンジニア

皆さんのゲーム作りのお手伝い

ドキュメント

ドキュメント翻訳作業・管理

unity

private

あんどう けいご

安藤 圭吾

Unity 部 副部長

もくもく会

コミケ本（Unibook3出すよ！）

SlackにUnityのコミュニティを作成しました

Unity 765 Slack 183

投稿を編集

kyusyukeigoが2015/04/05に投稿(2015/04/05に編集)・編集履歴(11)

unity

ご連絡

Unity 5.1 をベースとして話します。

資料は後日 Unite ページで公開します。

今回使用するソースコードはこちら
https://github.com/anchan828/unite2015tokyo

たぶん質問の時間取れないと思うので...質問は Slack で ﾃﾞｷﾚﾊﾞｲｲﾅ-
→ #editor-extensions

もちろん Unity ブースに居るのでその時に質問どうぞ。

エディター拡張バリバリやってますか？

私はバリバリやってます

エディター拡張バリバリやってますか？

私はバリバリやってます

今日は

年末 〜 今年初めに一番問い合わせの多かった

> アニメーションデータの自動生成（2D寄り）
>
> ステートマシンの自動生成の話はしません

Unity4.6 ~ Unity 5.1 に実装された面白い機能

（Unityスタッフとして話すと怒られそうな）おまけ

話の進め方

話をどんどん掘り下げて深い話をしていきます

深い話が終わったあと何の話をしていたか
思い出しながら進めていきましょう

アジェンダ

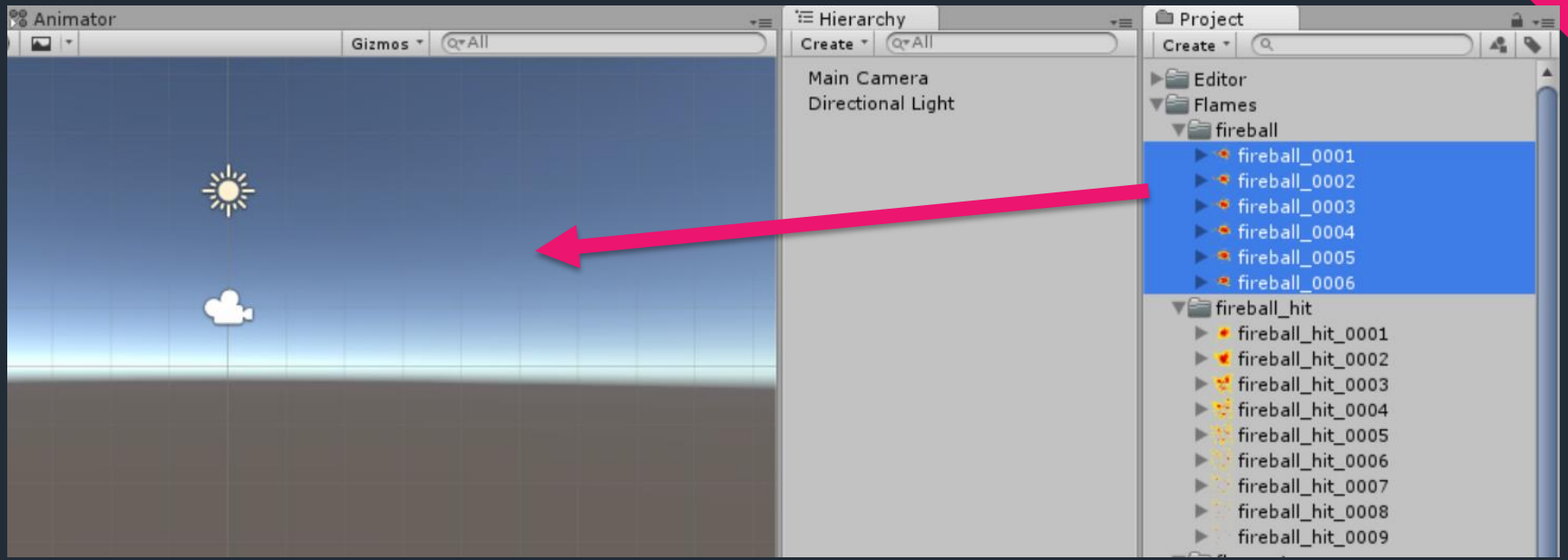
スプライトアニメーションを題材とした
エディター拡張の話

Unity5の新機能

スプライトアニメーションを題材としたエディター拡張の話

Sprite アニメーションの作成 - Unity 4

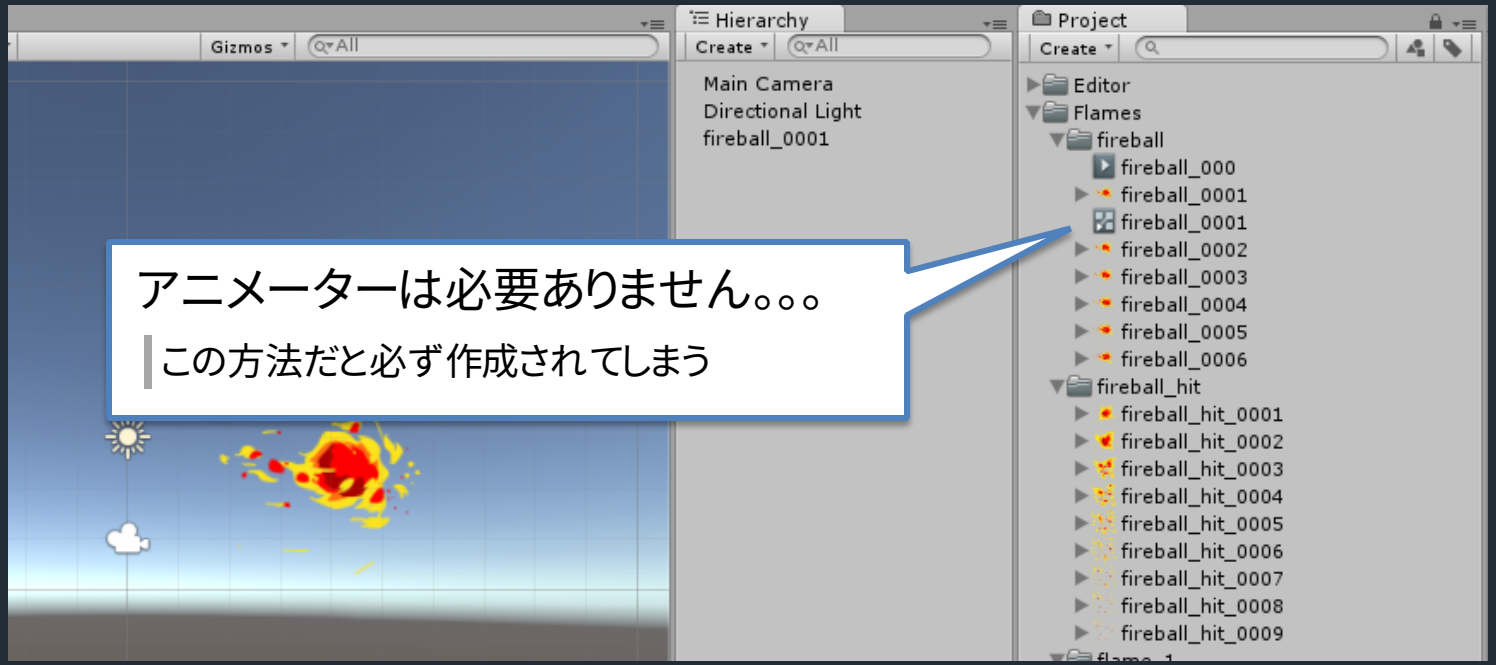
Spriteアニメーション、今までどうやって作っていましたか？



素材: “Pow Studios” http://powstudios.com/

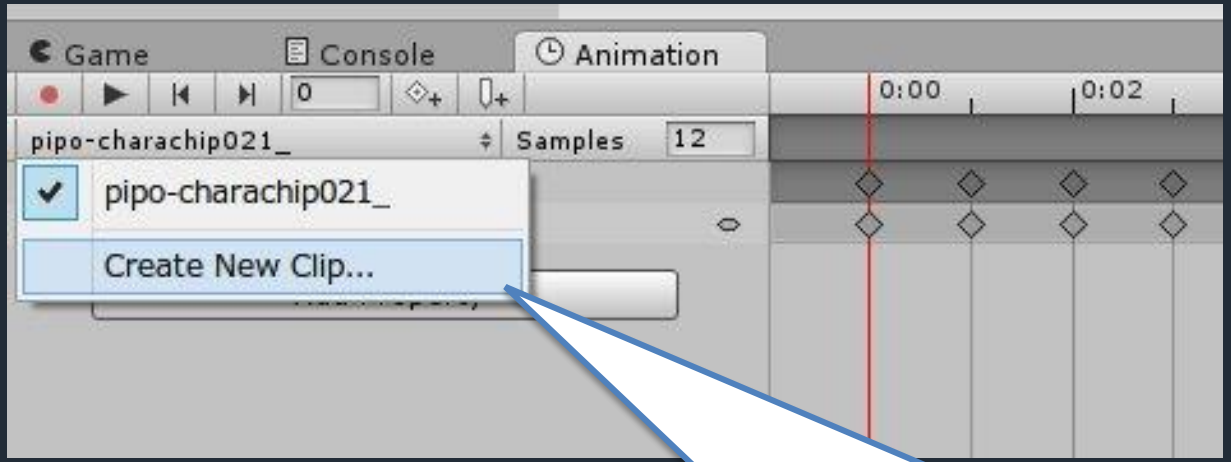
Sprite アニメーションの作成 - Unity 4

Spriteアニメーション、今までどうやって作っていましたか？



Sprite アニメーションの作成 - Unity 4

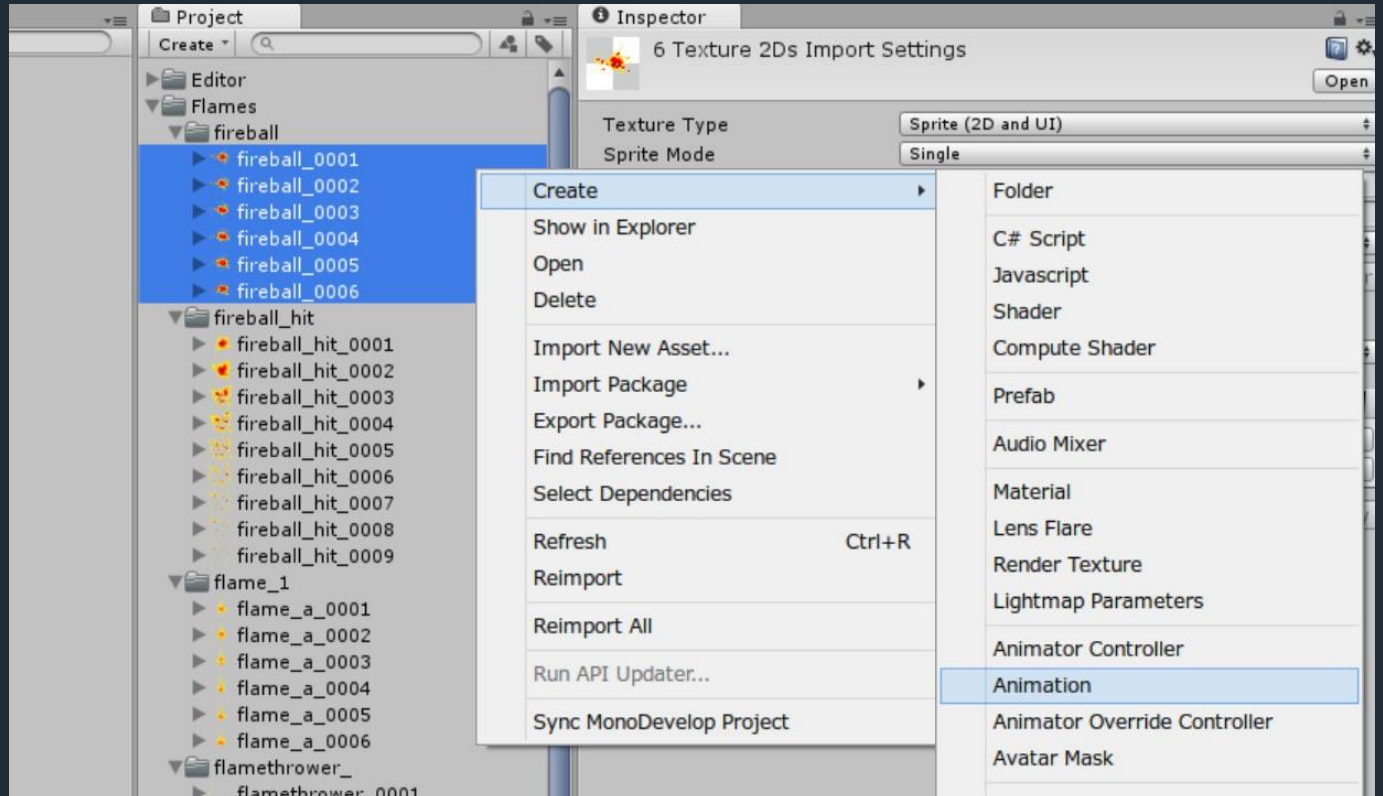
Spriteアニメーション、今までどうやって作っていましたか？



この方法だとSpriteの適用は手作業でしなきゃいけない

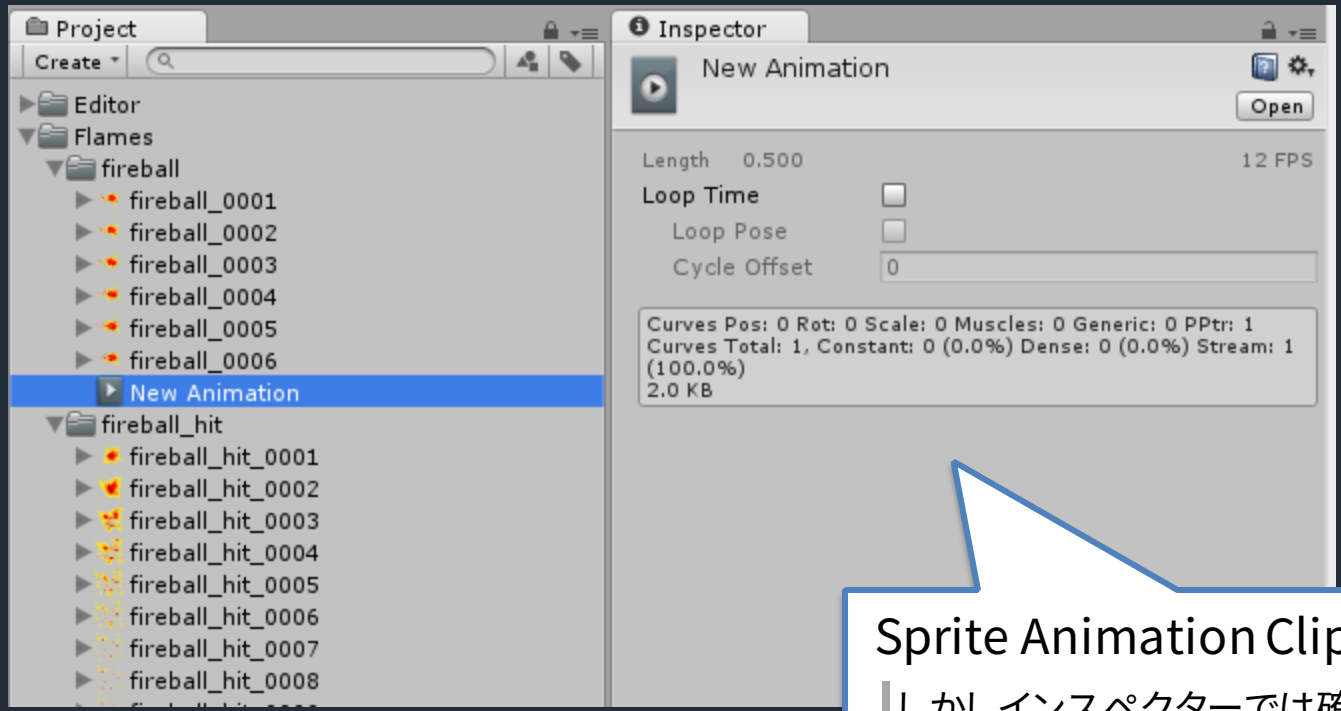
Sprite アニメーションの作成 - Unity 5

Unity5.1 ではスプライトを選択してAnimationClip作成



Sprite アニメーションの作成 - Unity 5

Unity5.1 ではスプライトを選択してAnimationClip作成



Sprite Animation Clipが作成された

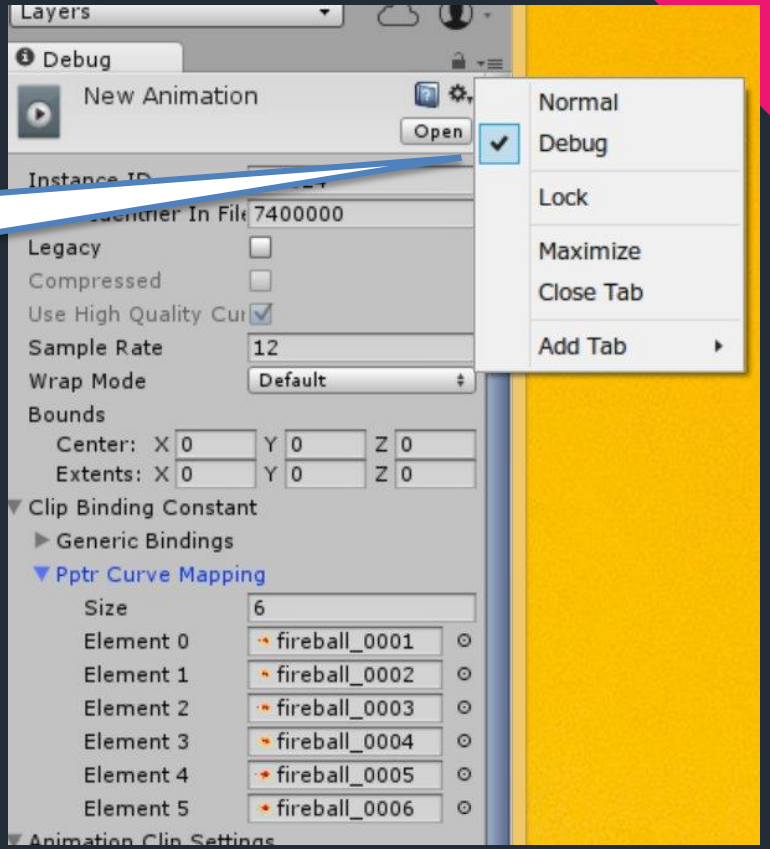
しかしインスペクターでは確認できない

Sprite アニメーションの作成 - Unity 5

スプライトを確認するには

インスペクターをDebugモードにする

- インスペクターウィンドウの ≡ から行える



Sprite アニメーションの作成 - Unity 5

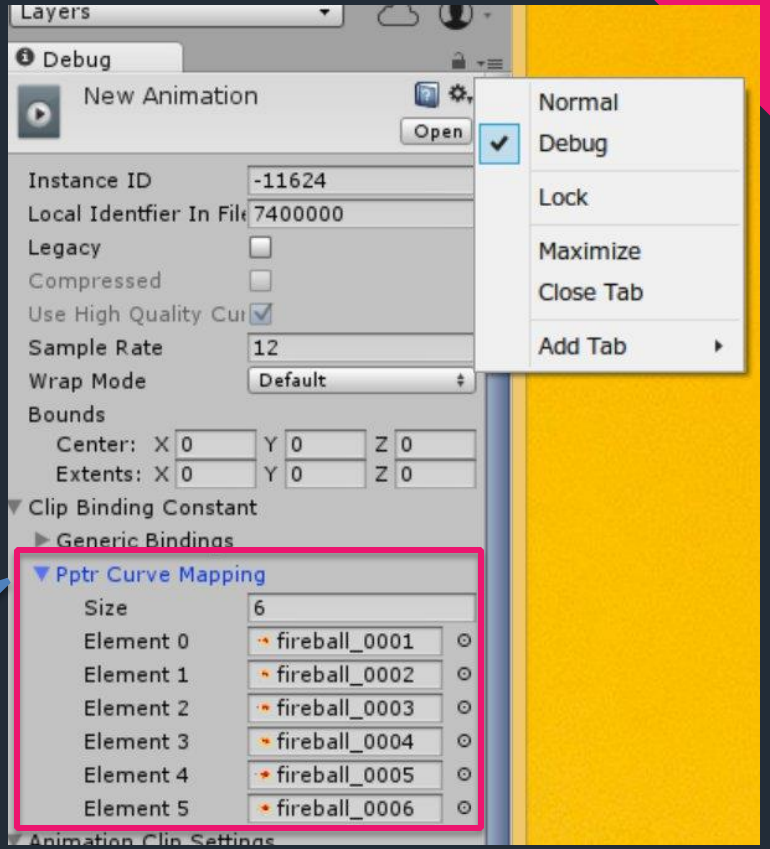
スプライトを確認するには

インスペクターをDebugモードにする

インスペクターウィンドウの ≡ から行える

ここに スプライトが保持されている

ここではスプライトの配列は変更できない

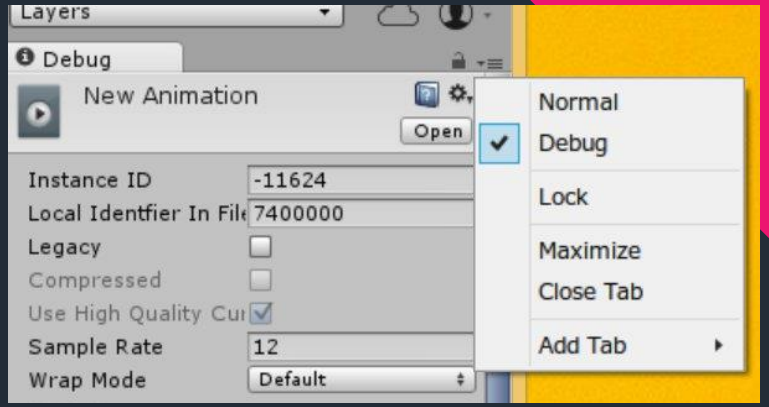


ちょっと寄り道 – Inspector デバッグモード

デバッグモードで見えているものは...

すべてのプロパティが表示されている

> 一部、編集不可あり（disable のやつ）

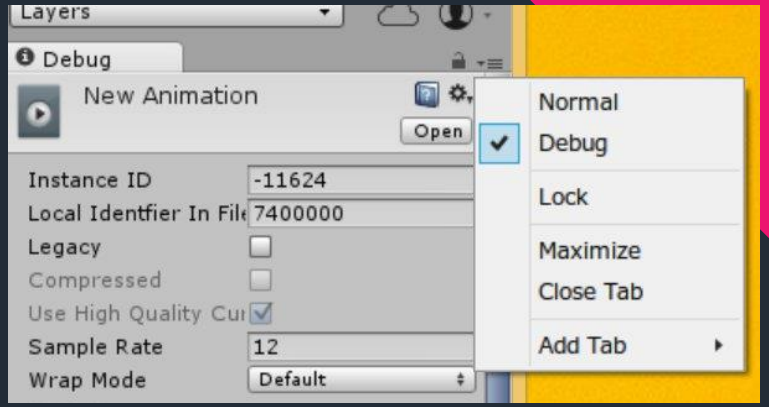


ちょっと寄り道 – Inspector デバッグモード

デバッグモードで見えているものは...

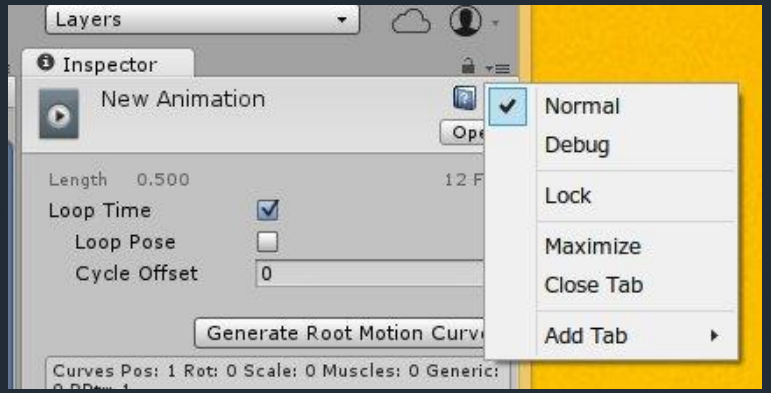
すべてのプロパティが表示されている

- 一部、編集不可あり（disable のやつ）



Normal はカスタムエディターによる表示

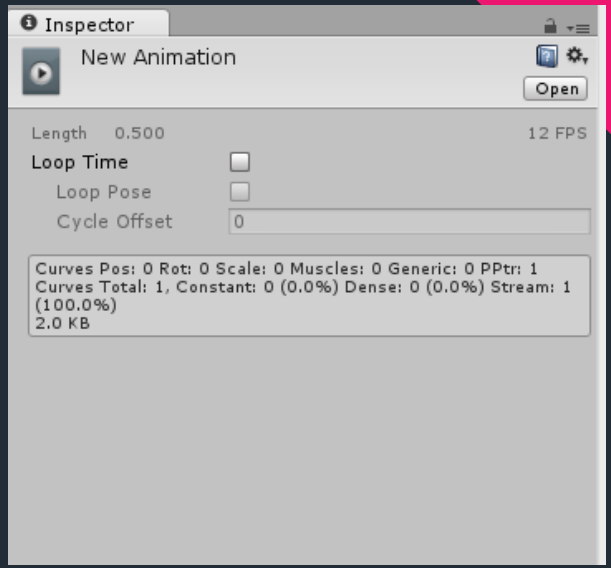
- AnimationClipEditor で表示されている



ちょっと寄り道２ – CustomPreview

インスペクターにはスプライドが表示されない

どうしよう?

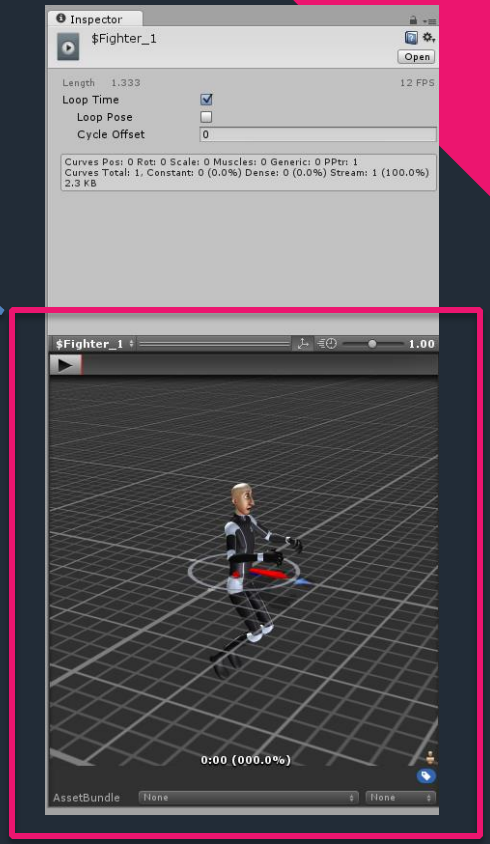


ちょっと寄り道２ – CustomPreview

インスペクターにはスプライドが表示されない

スプライトをココに表示してみれば便利では？

> テクスチャやアニメーションの再生をプレビューする場所



ちょっと寄り道２ – CustomPreview

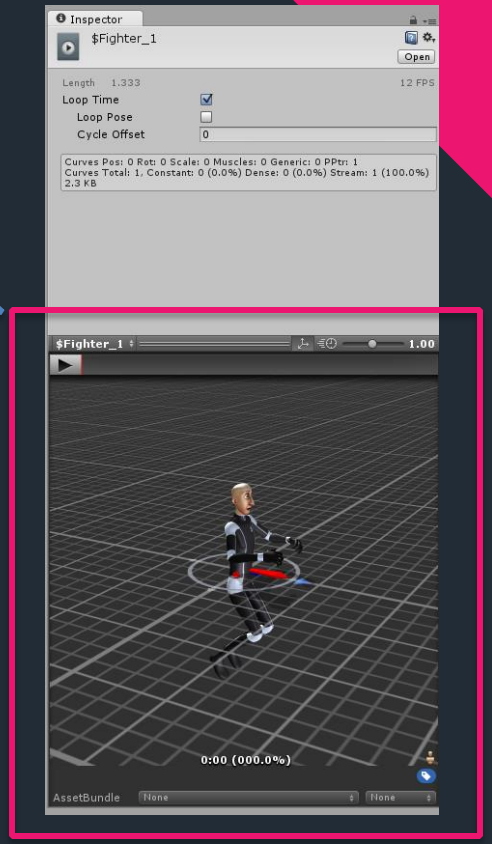
登録しているスプライドが表示されない

スプライトをココに表示してみれば便利では？

> テクスチャやアニメーションの再生をプレビューする場所

カスタムプレビューで実装を行ってみよう

> CustomPreview属性とObjectPreviewクラスを継承する



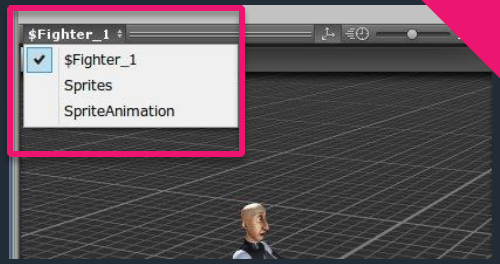

```
[CustomPreview(typeof(AnimationClip))]
public class SpriteAnimationPreview : ObjectPreview
{
}
```

ちょっと寄り道２ – CustomPreview

プレビューは複数のカスタムプレビューを持てる

タイトルの部分がドロップダウンメニューになる

> 2つのカスタムプレビューを持っていることがわかる

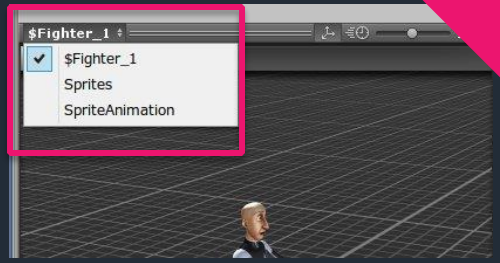


ちょっと寄り道２ – CustomPreview

プレビューは複数のカスタムプレビューを持てる

タイトルの部分がドロップダウンメニューになる

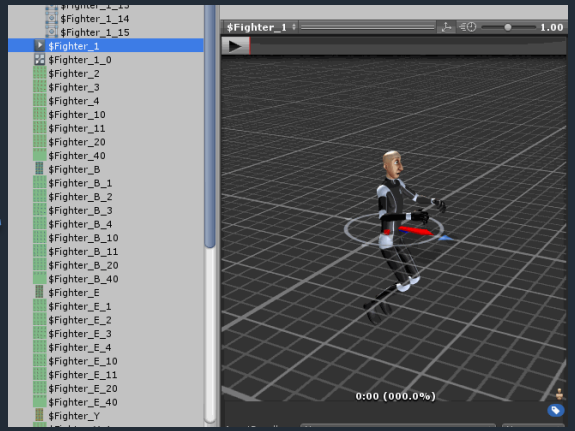
> 2つのカスタムプレビューを持っていることがわかる



ちょっと残念なところ

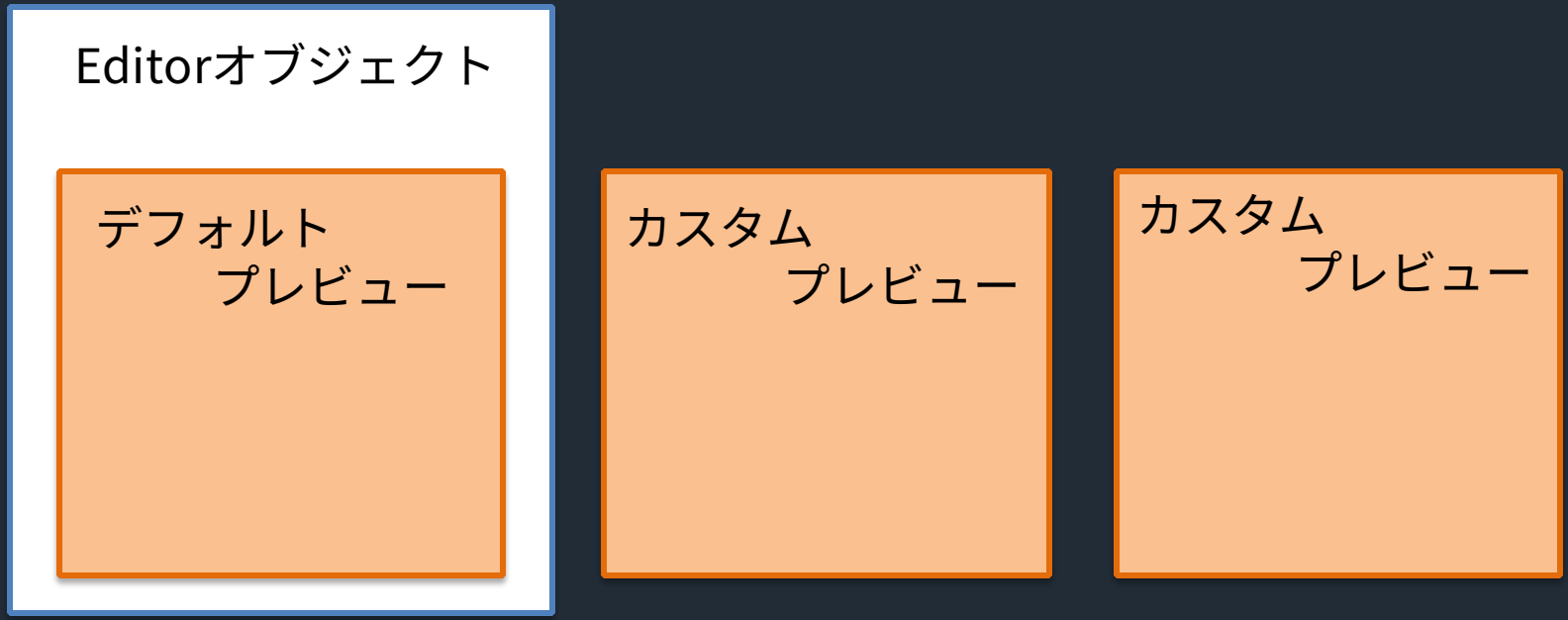
最初（デフォルト）に表示させるプレビューを設定できない

> Unityが用用意しているプレビューが必ず最初に呼び出される



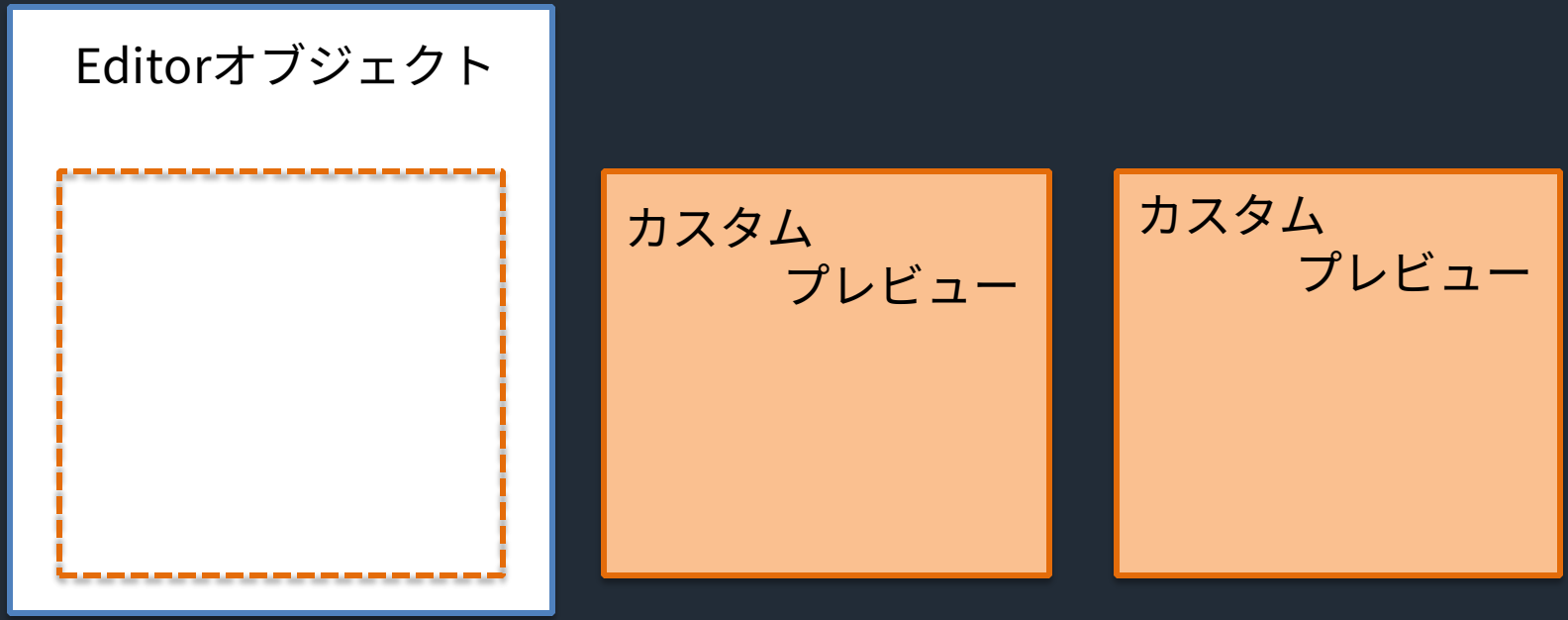
ちょっと寄り道２ – CustomPreview

イメージはこんな感じ



ちょっと寄り道２ – CustomPreview

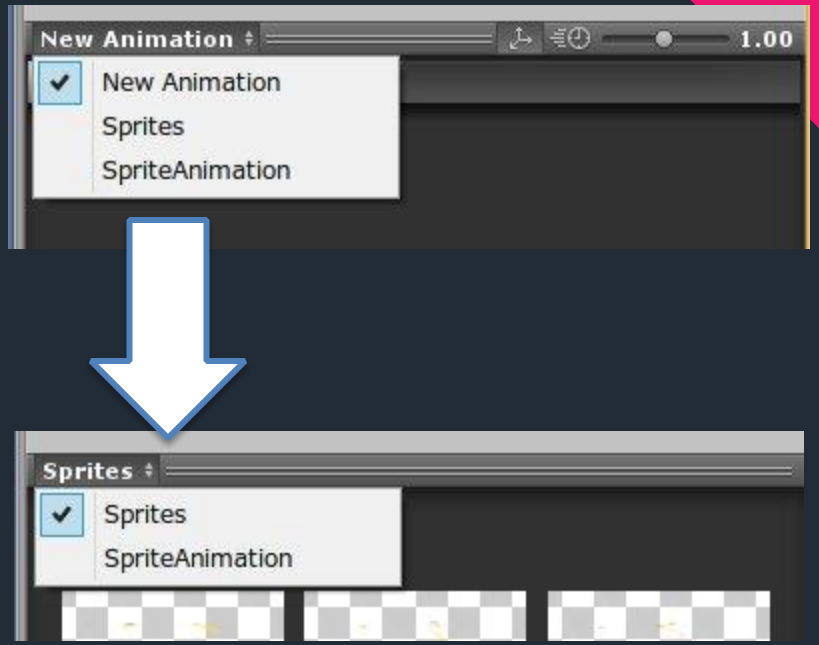
今回の理想はデフォルトプレビューはいらない。消す。



ちょっと寄り道２ – CustomPreview

カスタムエディターをオーバーライド

```
[CustomEditor(typeof(AnimationClip))]
public class AnimationClipEditor : OverrideEditor
{
    public override bool HasPreviewGUI()
    {
        return false;
    }
}
```



デフォルトプレビューをオフにする

> HasPreviewGUIだけオーバーライドしても他に影響が出るので実際は他のメソッドも適切にオーバーライド

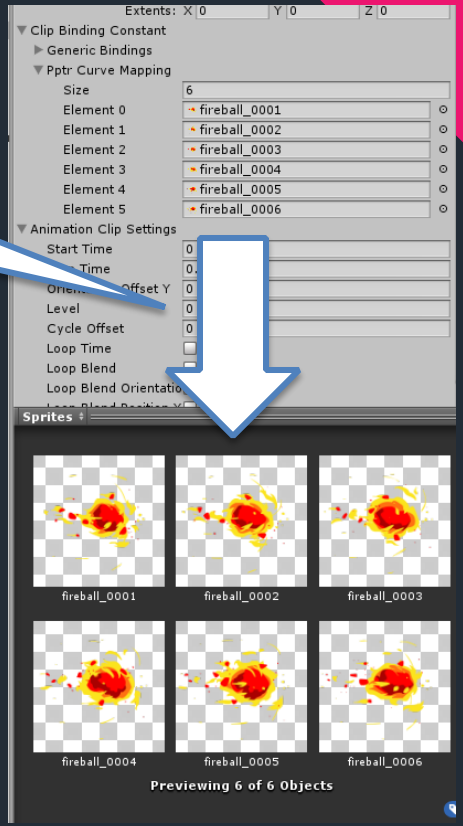
http://qiita.com/kyusyukeigo/private/6130429ab4a78702813f

ちょっと寄り道２ – CustomPreview

作成したカスタムプレビューその１

AnimationClipに登録されているスプライトを表示する

> Unityが用意しているプレビューが必ず最初に呼び出される



ちょっと寄り道２ – CustomPreview

作成したカスタムプレビューその１ – 初期化

```
public override void Initialize(Object[] targets)
{
        base.Initialize(targets);

        var sprites = new Object[0];

        // ここで SerializedProperty から Sprite を取得する

        m_Targets = sprites;
}
```

ちょっと寄り道２ – CustomPreview

作成したカスタムプレビューその１ – プレビュー表示

```
public override void OnPreviewGUI(Rect r, GUIStyle background)
{
    var previewTexture = AssetPreview.GetAssetPreview(target);
    EditorGUI.DrawTextureTransparent(r, previewTexture);
}
```

ちょっと寄り道２ – CustomPreview

作成したカスタムプレビューその１

```
public override void OnPreviewGUI(Rect r, GUIStyle background)
{
    var previewTexture = AssetPreview.GetAssetPreview(target);
    EditorGUI.DrawTextureTransparent(r, previewTexture);
}
```

OnPreviewGUI は m_Targets に設定した要素の数呼び出されます。

ちょっと寄り道２ – CustomPreview

作成したカスタムプレビューその１

```
public override void OnPreviewGUI(Rect r, GUIStyle background)
{
    var previewTexture = AssetPreview.GetAssetPreview(target);
    EditorGUI.DrawTextureTransparent(r, previewTexture);
}
```

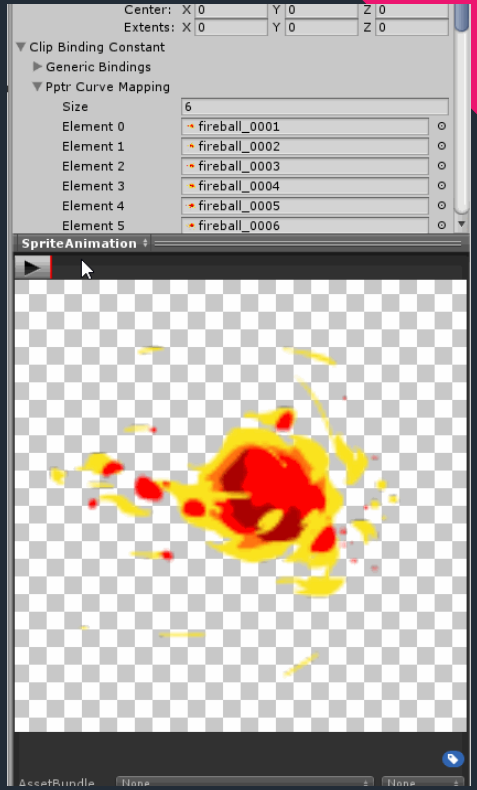
Spriteからプレビュー用のテクスチャを生成するのはこれが便利

> ただし、初回はテクスチャ生成のため null が返ってくるので Initialize メソッドで呼び出しておく

ちょっと寄り道２ – CustomPreview

作成したカスタムプレビューその２

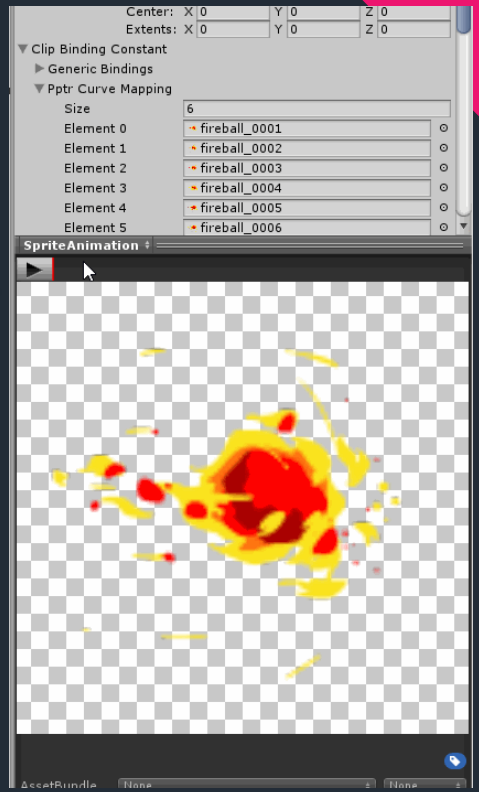
登録されているスプライトをアニメーション再生



ちょっと寄り道２ – CustomPreview

作成したカスタムプレビューその２

登録されているスプライトをアニメーション再生



UnityEditor.TimeControl を参考にすると幸せになるよ

> プライベートなのでデコンパイルしなきゃいけないけど！

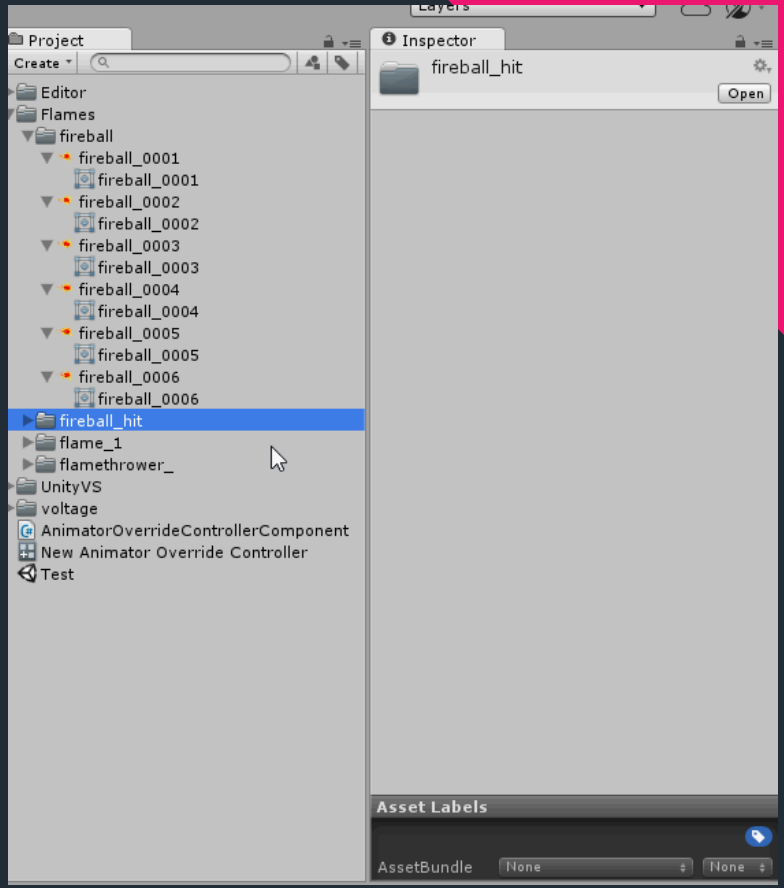
Sprite アニメーションの作成 - Unity 5

まとめ

Sprite選択して「Create / Animation」

プレビューで登録されているスプライトの一覧を表示

スプライトアニメーションを再生



Sprite アニメーションをスクリプトで作成する

作り方は Unity 4 と 5 で変わりません

覚えておくべきなのはこの３つ

ObjectReferenceKeyframe

EditorCurveBinding

AnimationUtility.SetObjectReferenceCurve

Sprite アニメーションをスクリプトで作成する

ObjectReferenceKeyframe

Keyframeのオブジェクト参照版

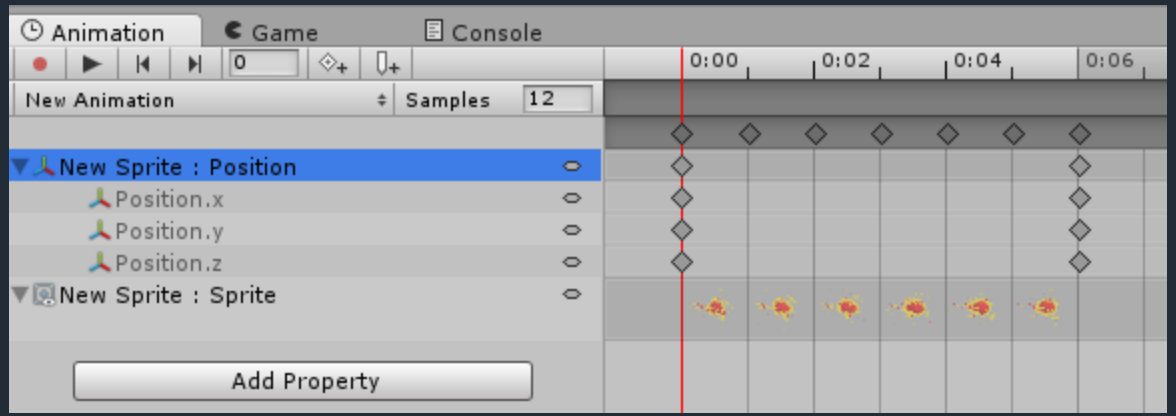
> Float値の代わりにObjectを設定することができる

Sprite アニメーションをスクリプトで作成する

ObjectReferenceKeyframe

Keyframeのオブジェクト参照版

> Float値の代わりにObjectを設定することができる

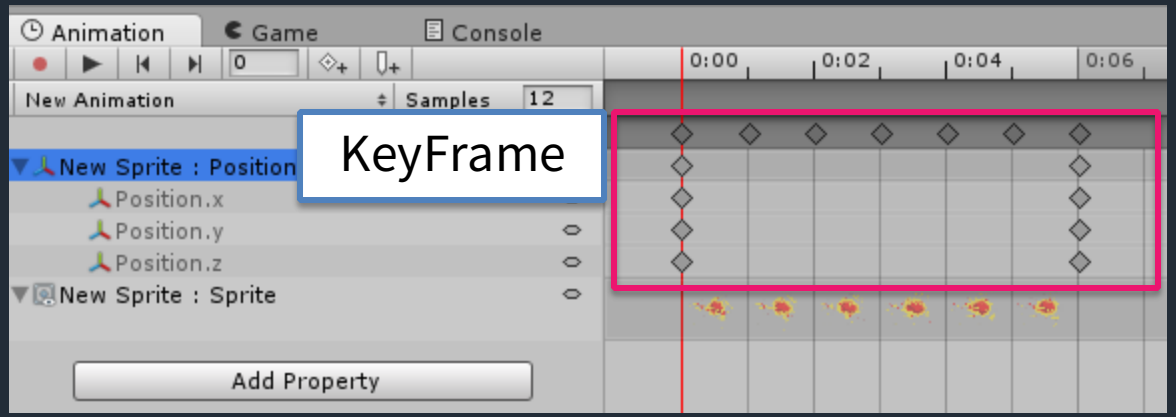


Sprite アニメーションをスクリプトで作成する

ObjectReferenceKeyframe

Keyframeのオブジェクト参照版

- Float値の代わりにObjectを設定することができる

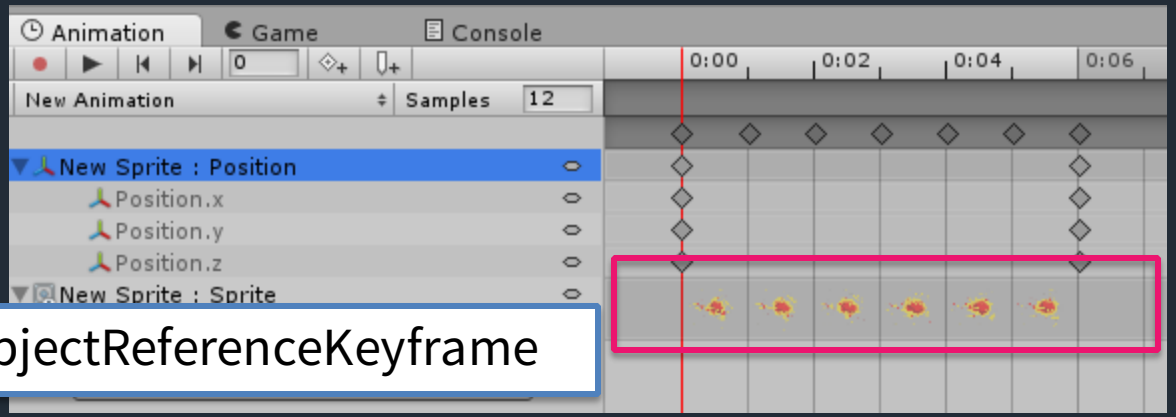


Sprite アニメーションをスクリプトで作成する

ObjectReferenceKeyframe

Keyframeのオブジェクト参照版

- Float値の代わりにObjectを設定することができる

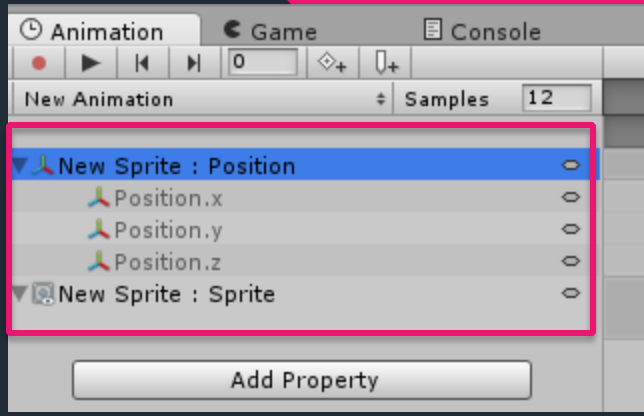


Sprite アニメーションをスクリプトで作成する

EditorCurveBinding

どのプロパティをアニメーションさせるかのキー

> キーはシリアライズ（アニメーション）可能なデータのみ

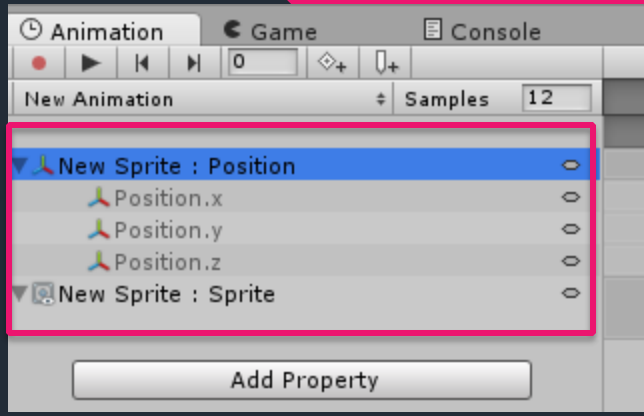


Sprite アニメーションをスクリプトで作成する

EditorCurveBinding

どのプロパティをアニメーションさせるかのキー

> キーはシリアライズ（アニメーション）可能なデータのみ




```
EditorCurveBinding.PPtrCurve("", typeof(SpriteRenderer), "m_Sprite");
```

ルートにある

SpriteRendererの

m_Spriteをアニメーション

Sprite アニメーションをスクリプトで作成する

プロパティ名を知るには？

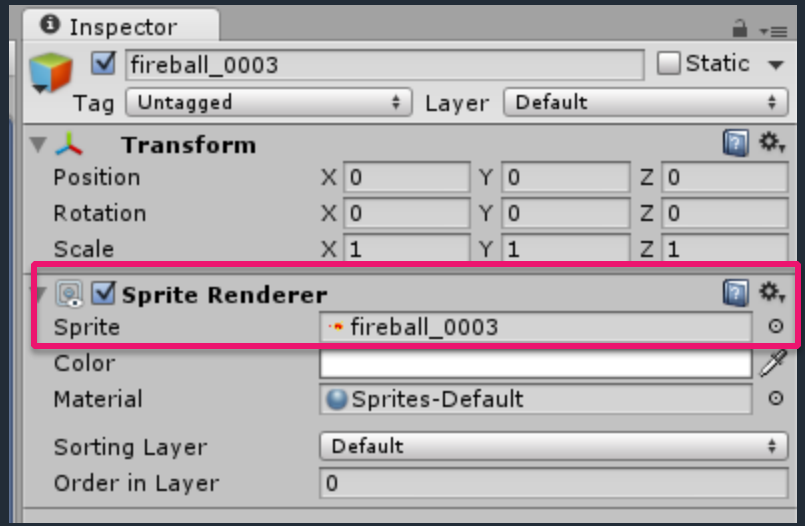
```
EditorCurveBinding.PPtrCurve("", typeof(SpriteRenderer), "m_Sprite");
```

どうやって m_Sprite という名を知るか

Sprite アニメーションをスクリプトで作成する

プロパティ名を知るには？

```
EditorCurveBinding.PPtrCurve("", typeof(SpriteRenderer), "m_Sprite");
```



どうやって m_Sprite という名を知るか

Sprite という名しか見えない！

ObjectNames.NicifyVariableName
によって変換されてしまう

Sprite アニメーションをスクリプトで作成する

プロパティ名を知るには？ - 手っ取り早い方法１

１．コンポーネントを SerializedObject に変換する

２．SerializedObject からすべての SerializedProperty を取得する

３．SerializedProperty.propertyPath がプロパティ名（＝パス）

Sprite アニメーションをスクリプトで作成する

プロパティ名を知るには？ - 手っ取り早い方法１

１．コンポーネントを SerializedObject に変換する

```
var serializedObject = new SerializedObject(component);
```

Sprite アニメーションをスクリプトで作成する

プロパティ名を知るには？ - 手っ取り早い方法１

２．SerializedObjectからすべてのSerializedPropertyを取得する

```
var serializedObject = new SerializedObject(component);

var serializedProperty = serializedObject.GetIterator();

while (serializedProperty.NextVisible(true))
{

}
```

Sprite アニメーションをスクリプトで作成する

プロパティ名を知るには？ - 手っ取り早い方法１

３．SerializedProperty.propertyPath が正確なプロパティ名（＝パス）

```
var serializedObject = new SerializedObject(component);

var serializedProperty = serializedObject.GetIterator();

while (serializedProperty.NextVisible(true))
{
        Debug.Log(serializedProperty.propertyPath);
}
```

Sprite アニメーションをスクリプトで作成する

プロパティ名を知るには？ - 手っ取り早い方法１

３．SerializedProperty.propertyPath が正確なプロパティ名

```
var serializedObject = new SerializedObject(component);

var serializedProperty = serializedObject.GetIterator();

while (serializedProperty.NextVisible(true))
{
        Debug.Log(serialize
}
```

プロパティパス：　m_position.x

プロパティ名：　x

Sprite アニメーションをスクリプトで作成する

プロパティ名を知るには？ - 手っ取り早い方法２

１．コンポーネントをファイルとして保存する

２．ファイルを確認する

Sprite アニメーションをスクリプトで作成する

プロパティ名を知るには？ - 手っ取り早い方法２

１．コンポーネントをファイルとして保存する

```
InternalEditorUtility.SaveToSerializedFileAndForget(
        new Object[] { component },
        filePath,
        true);
```

テキストデータにしたいので必ずtrue

Sprite アニメーションをスクリプトで作成する

プロパティ名を知るには？ - 手っ取り早い方法２

２．ファイルを確認する

m_Materials

m_Sprite

m_Color

```
%YAML 1.1
%TAG !u! tag:unity3d.com,2011:
--- !u!212 &1
SpriteRenderer: m_ObjectHideFlags: 0
m_PrefabParentObject: {fileID: 0}
m_PrefabInternal: {fileID: 0}
m_GameObject: {fileID: 0}
m_Enabled: 1
m_CastShadows: 0
m_ReceiveShadows: 0
m_Materials: - {fileID: 10754, guid: 0000000000000000e000000000000000, type: 0}
m_SubsetIndices:
m_StaticBatchRoot: {fileID: 0}
m_UseLightProbes: 1
m_ReflectionProbeUsage: 1
m_ProbeAnchor: {fileID: 0}
m_ScaleInLightmap: 1
m_PreserveUVs: 0
m_ImportantGI: 0
m_AutoUVMaxDistance: .5
m_AutoUVMaxAngle: 89
m_LightmapParameters: {fileID: 0}
m_SortingLayerID: 0
m_SortingOrder: 0
m_Sprite: {fileID: 0}
m_Color: {r: 1, g: 1, b: 1, a: 1}
```

Sprite アニメーションをスクリプトで作成する

UnityEngine.Object と SerializedObject と YAML

Sprite アニメーションをスクリプトで作成する

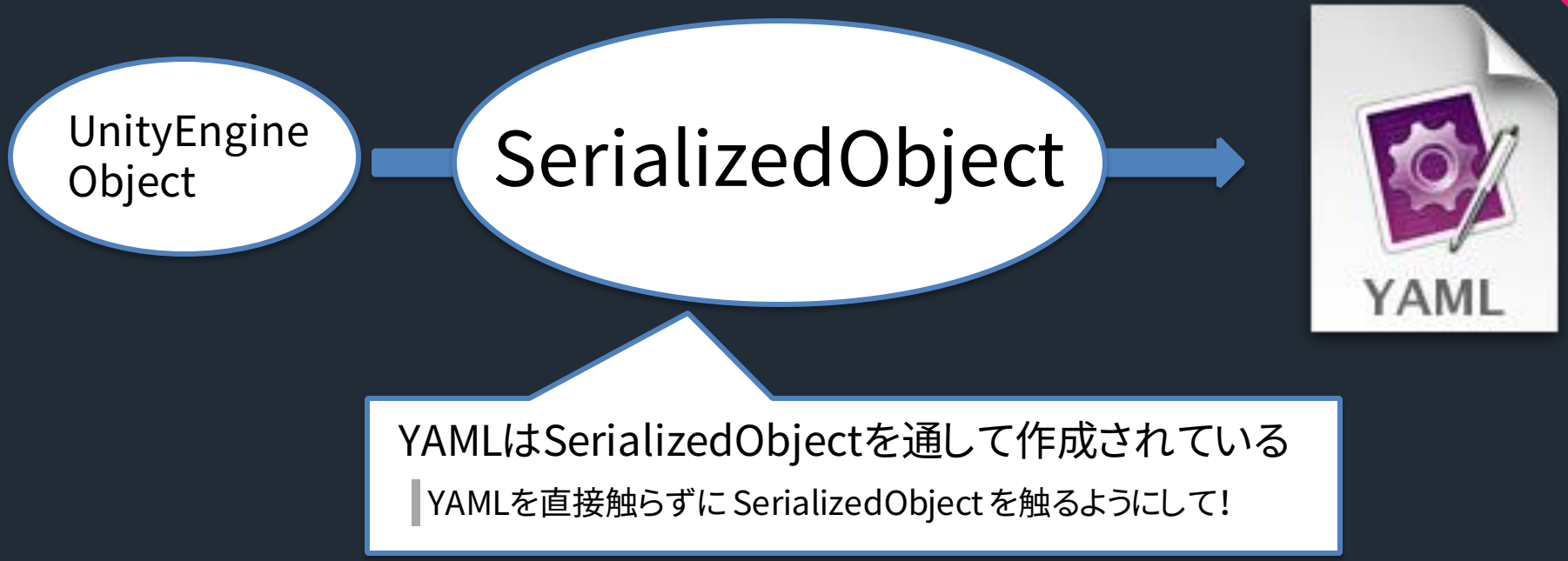
UnityEngine.Object と SerializedObject と YAML

UnityEngine Object

よく YAML を直接触る人がいます

Sprite アニメーションをスクリプトで作成する

UnityEngine.Object と SerializedObject と YAML



Sprite アニメーションをスクリプトで作成する

プレハブも SerializedObject に出来ます

```
var go = AssetDatabase.LoadAssetAtPath<GameObject>("Assets/Main Camera.prefab");

var serializedObject = new SerializedObject(go);

var serializedProperty = serializedObject.GetIterator();

while (serializedProperty.Next(true))
{
     Debug.Log(serializedProperty.propertyPath);
}
```

Sprite アニメーションをスクリプトで作成する

プレハブも SerializedObject に出来ます

```
var go = AssetDatabase.LoadAssetAtPath<GameObject>("Assets/Main Camera.prefab");

var serializedObject = new SerializedObject(go);

var serializedProperty = serializedObject.GetIterator();

while (serializedProperty.Next(true))
{
     Debug.Log(serializedProperty.propertyPath);
}
```

シーンファイルは特殊なことしているので例外

> RenderSettingsをシーンデータに加えたりとかイロイロしてる

Sprite アニメーションをスクリプトで作成する

UnityEngine.Object と SerializedObject と YAML

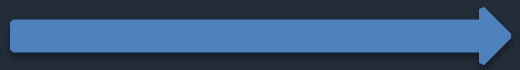
UnityEngine Object

YAMLを直接いじりたいのはすごくわかる

Sprite アニメーションをスクリプトで作成する

UnityEngine.Object と SerializedObject と YAML

UnityEngine Object



YAMLを直接いじりたいのはすごくわかる

俺達のテリトリー

Sprite アニメーションをスクリプトで作成する

作り方は Unity 4 と 5 で変わりません

覚えておくべきなのはこの３つ

ObjectReferenceKeyframe

EditorCurveBinding

AnimationUtility.SetObjectReferenceCurve

話を戻します！
こんな話をしてましたよ！

Sprite アニメーションをスクリプトで作成する

AnimationUtility.SetObjectReferenceCurve

```
AnimationUtility.SetObjectReferenceCurve(
        animationClip,
        editorCurveBinding,
        objectReferenceKeyframes);
```

指定したAnimationClipの

Sprite アニメーションをスクリプトで作成する

AnimationUtility.SetObjectReferenceCurve

```
AnimationUtility.SetObjectReferenceCurve(
        animationClip,
        editorCurveBinding,
        objectReferenceKeyframes);
```

指定したAnimationClipの

特定のキーに

Sprite アニメーションをスクリプトで作成する

AnimationUtility.SetObjectReferenceCurve

```
AnimationUtility.SetObjectReferenceCurve(
        animationClip,
        editorCurveBinding,
        objectReferenceKeyframes);
```

指定したAnimationClipの

特定のキーに

特定の値を設定する

まとめ

スプライトアニメーションの AnimationClip を作成してみる

```
static AnimationClip CreateSpriteAnimation(params Sprite[] sprites)
{
}
```

まとめ

```
static AnimationClip CreateSpriteAnimation(params Sprite[] sprites)
{
    var animationClip = new AnimationClip{ frameRate = 12 };

    var animationClipSettings = new AnimationClipSettings{ loopTime = true };

    AnimationUtility.SetAnimationClipSettings(animationClip, animationClipSettings);

}
```

AnimationClipの初期設定

> フレームレートは12、ループ設定

まとめ

```
static AnimationClip CreateSpriteAnimation(params Sprite[] sprites)
{
    ...

    var objectReferenceKeyframes = new ObjectReferenceKeyframe[sprites.Length];

    for (var i = 0; i < objectReferenceKeyframes.Length; i++)
    {
        objectReferenceKeyframes[i] = new ObjectReferenceKeyframe
        {
            value = sprites[i],
            time = i / animationClip.frameRate
        };
    }

}
```

ObjectReferenceKeyframe の作成

> 作成する分はスプライトの数

まとめ

```
static AnimationClip CreateSpriteAnimation(params Sprite[] sprites)
{
    ...


    var editorCurveBinding = EditorCurveBinding.PPtrCurve("", typeof(SpriteRenderer), "m_Sprite");

}
```

EditorCurveBindingの作成

SpriteRendererの m_Sprite をアニメーションさせる

まとめ

```
static AnimationClip CreateSpriteAnimation(params Sprite[] sprites)
{
    ...


    AnimationUtility.SetObjectReferenceCurve(animationClip, editorCurveBinding, objectReferenceKeyframes);

}
```

AnimationClipにKeyframeを設定する

> クリップ、バインド、キーフレームの3つを使う

まとめ


```
static AnimationClip CreateSpriteAnimation(params Sprite[] sprites)
{
    var animationClip = new AnimationClip{ frameRate = 12 };

    var animationClipSettings = new AnimationClipSettings{ loopTime = true };

    AnimationUtility.SetAnimationClipSettings(animationClip, animationClipSettings);

    var objectReferenceKeyframes = new ObjectReferenceKeyframe[sprites.Length];

    for (var i = 0; i < objectReferenceKeyframes.Length; i++)
    {
        objectReferenceKeyframes[i] = new ObjectReferenceKeyframe
        {
            value = sprites[i],
            time = i / animationClip.frameRate
        };
    }

    var editorCurveBinding = EditorCurveBinding.PPtrCurve("", typeof(SpriteRenderer), "m_Sprite");

    AnimationUtility.SetObjectReferenceCurve(animationClip, editorCurveBinding, objectReferenceKeyframes);

    return animationClip;
}
```

Animator Override Controller

Animator Override Controller

例えば...

ステートマシンはそのままで戦士、格闘家、
魔法使いのようにスプライトだけを変更したい



素材: “P3X-774” http://p3x774.web.fc2.com/

Animator Override Controller

例えば...

ステートマシンはそのままで戦士、格闘家、
魔法使いのようにスプライトだけを変更したい

アニメーターコントローラーのアニメーション
を上書きする オーバーライドコントローラー

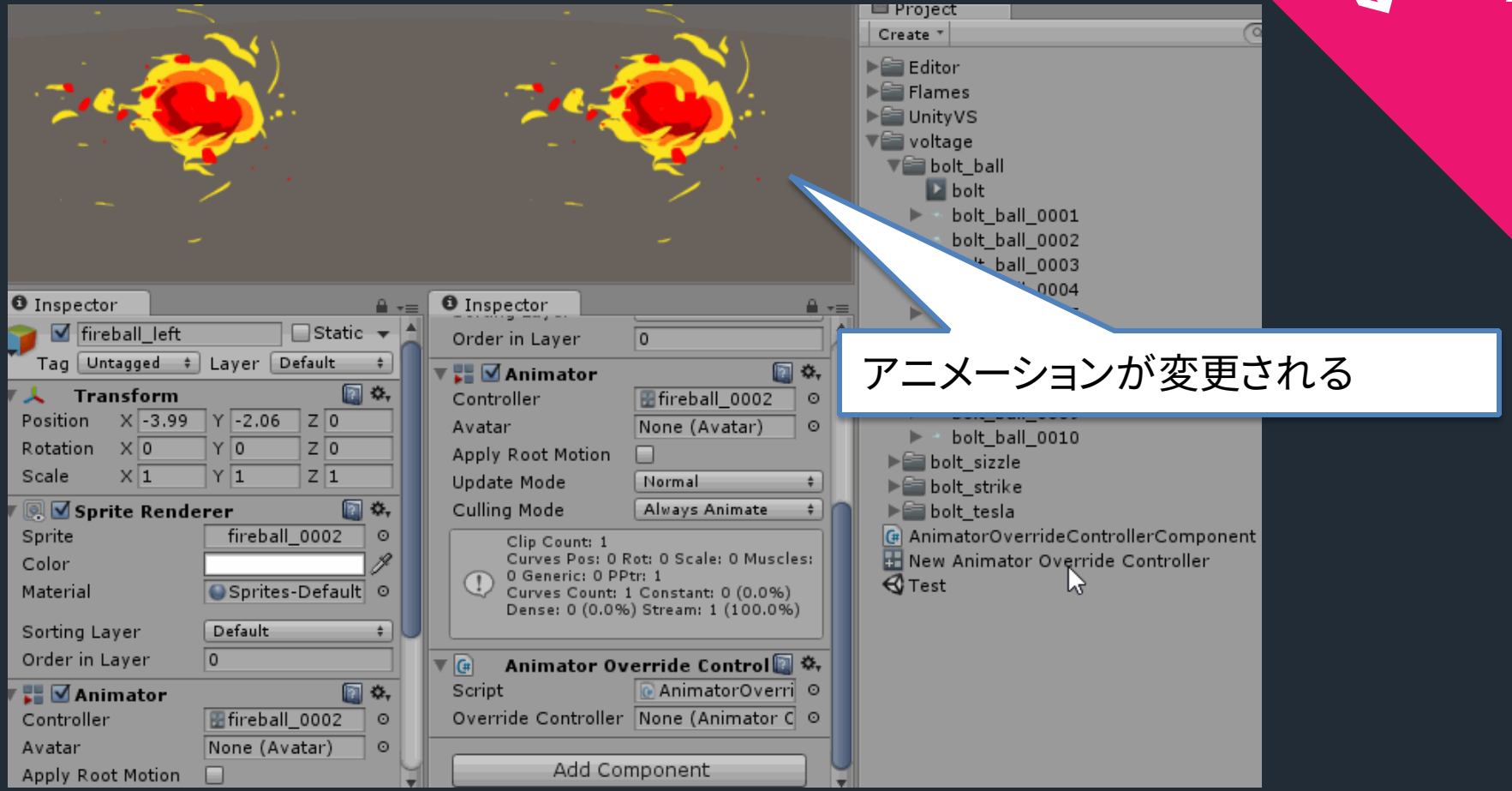
素材: “P3X-774” http://p3x774.web.fc2.com/

Animator Override Controller の使用

```
[SerializeField]
private AnimatorOverrideController overrideController;

void Awake()
{
    if (overrideController != null)
        GetComponent<Animator>().runtimeAnimatorController = overrideController;
}
```

ランタイムでコントローラーを上書きする

Animator Override Controller の使用



Animator Override Controller の作成


```
var _animatorOverrideController = new AnimatorOverrideController
{
    name = "New Animator Override Controller",
    runtimeAnimatorController = animatorController,
    clips = new[]
    {
        new AnimationClipPair
        {
            originalClip = originalClip,
            overrideClip = overrideClip
        }
    }
};
```

Animator Override Controller の作成

```
var _animatorOverrideController = new AnimatorOverrideController
{
    name = "New Animator Override Controller",
    runtimeAnimatorController = animatorController,
    clips = new[]
    {
        new AnimationClipPair
        {
            originalClip = originalClip,
            overrideClip = overrideClip
        }
    }
};
```

オリジナルのアニメーターコントローラー

Animator Override Controller の作成

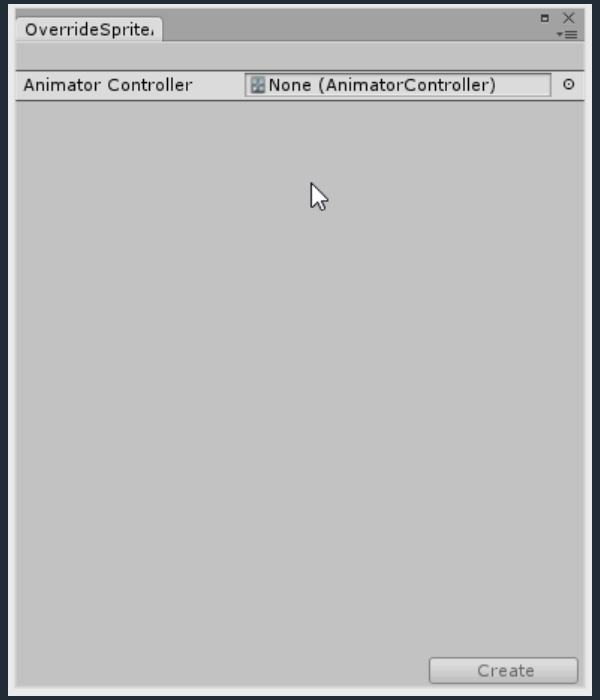
```
var _animatorOverrideController = new AnimatorOverrideController
{
    name = "New Animator Override Controller",
    runtimeAnimatorController = animatorController,
    clips = new[]
    {
        new AnimationClipPair
        {
            originalClip = originalClip,
            overrideClip = overrideClip
        }
    }
};
```

オリジナルのアニメーターコントローラー

オリジナルのアニメーションクリップ
上書きするアニメーションクリップ

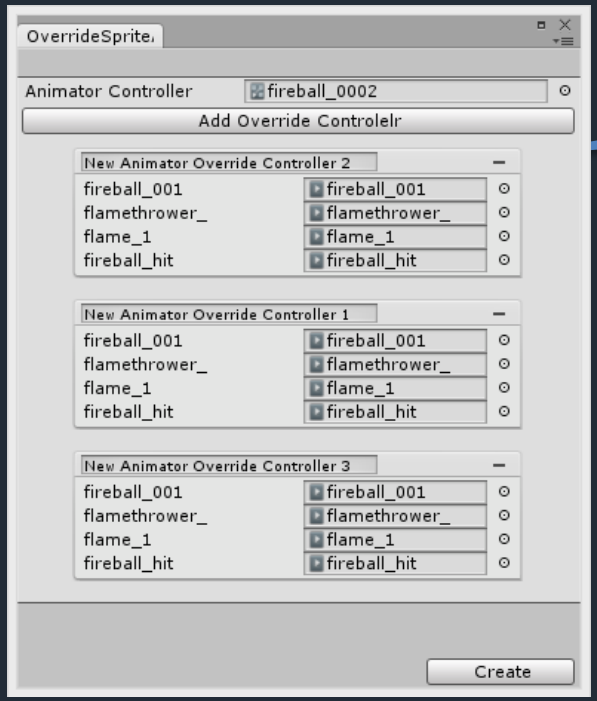
Animator Override Controller の作成

複数の Animator Override Controller を作るなら



Animator Override Controller の作成

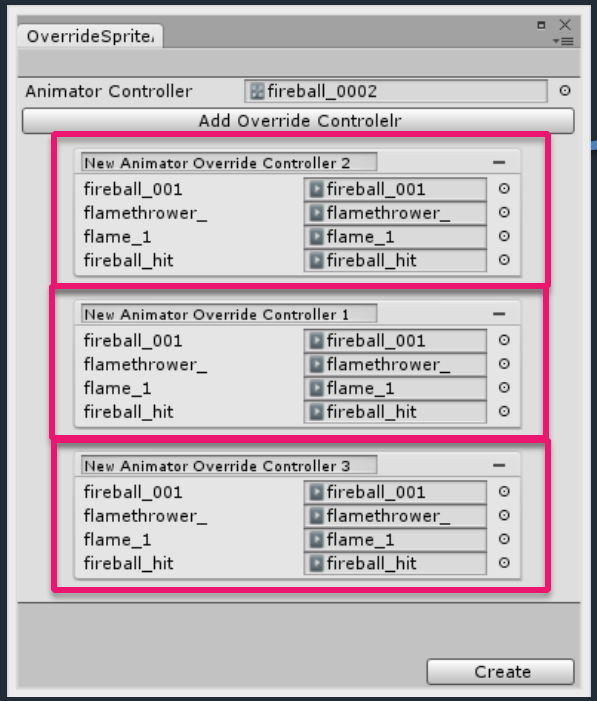
複数の Animator Override Controller を作るなら



ScriptableWizardでささっと作成する

Animator Override Controller の作成

複数の Animator Override Controller を作るなら



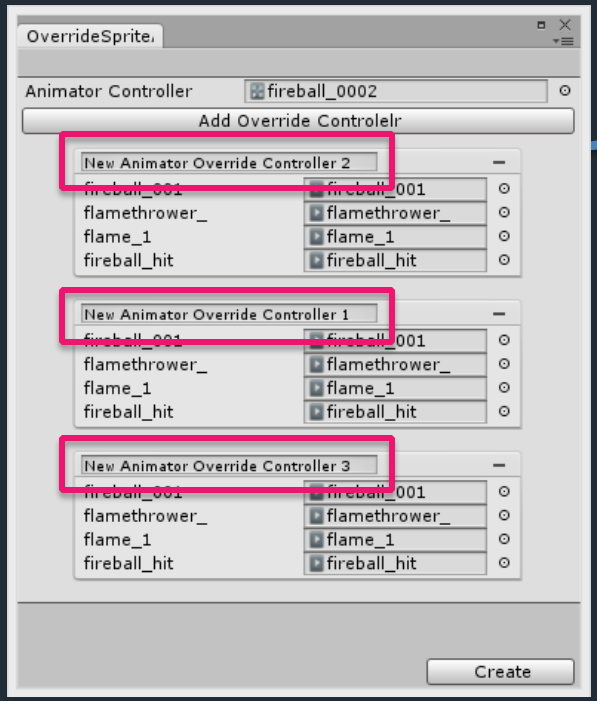
ScriptableWizardでささっと作成する

ReorderableList

使うと配列が見やすくなる

Animator Override Controller の作成

複数の Animator Override Controller を作るなら



ScriptableWizardでささっと作成する

ReorderableList

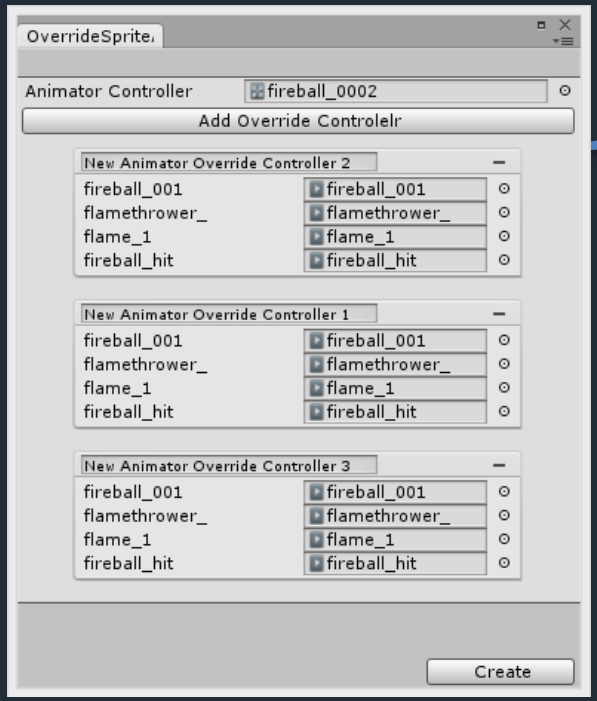
使うと配列が見やすくなる

AssetDatabase.GenerateUniqueAssetPath

同名のアセットがあった場合にユニークな名前にする

Animator Override Controller の作成

複数の Animator Override Controller を作るなら



ScriptableWizardでささっと作成する

ReorderableList

使うと配列が見やすくなる

AssetDatabase.GenerateUniqueAssetPath

同名のアセットがあってはだめなのでユニークな名前にする

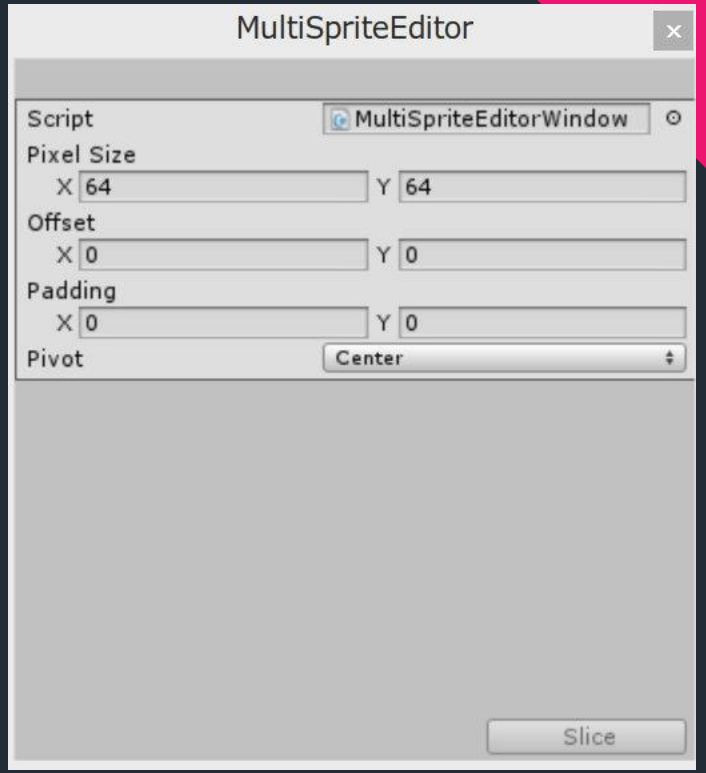
EditorApplication.delayCall

インスペクターアップデート後に呼び出されるデリゲート

ScriptableWizard

対話形式でアセット作成・編集

- isValid が便利
  - False にすることで CreateButton が Disable



ScriptableWizard

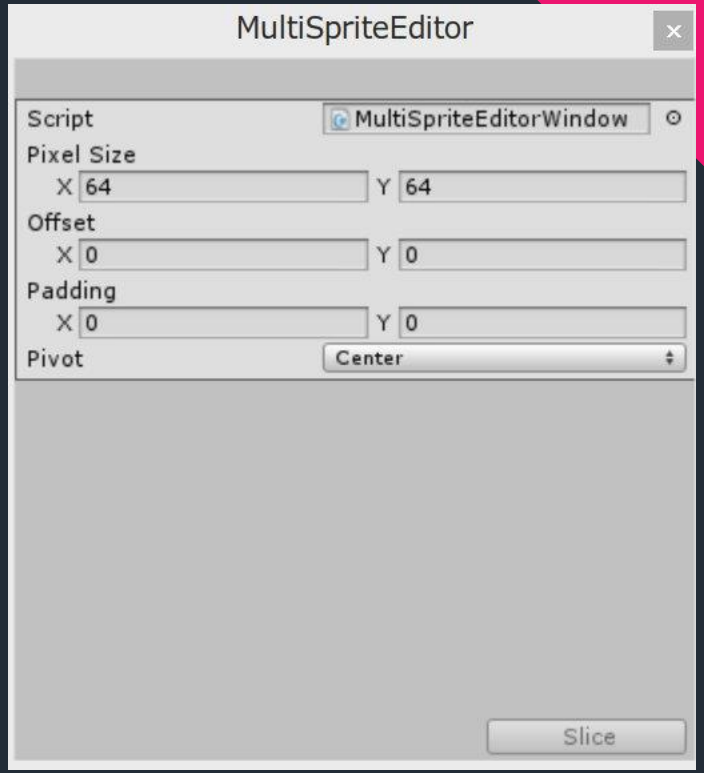
対話形式でアセット作成・編集

isValid が便利

- False にすることで CreateButton が Disable

常に前にポップアップ

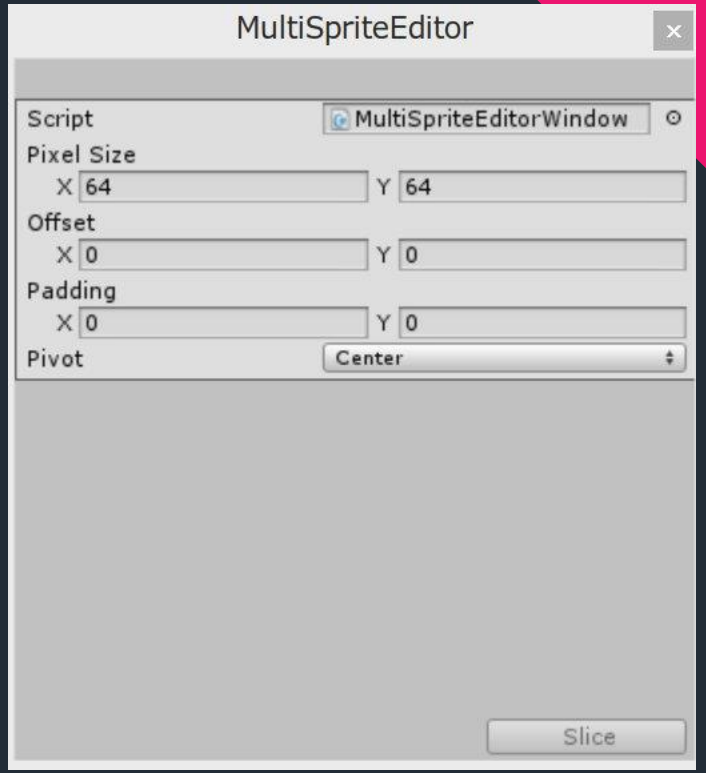
- EditorWindow の ShowUtility が実行されてる



ScriptableWizard

対話形式でアセット作成・編集

- isValid が便利
  - False にすることで CreateButton が Disable
- 常に前にポップアップ
  - EditorWindow の ShowUtility が実行されてる
- 状況に合わせてhelp や error を表示

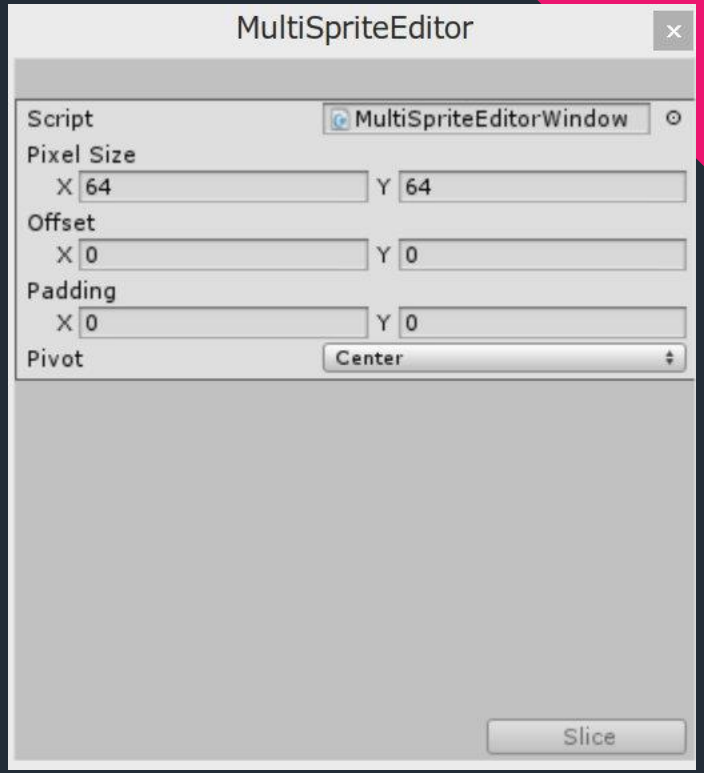


ScriptableWizard - Unity 4 でイケてないトコロ

GUIの表示方法がいまいち

表示はインスペクターと同じ

- Editor.DrawDefaultInspectorを実行してる



ScriptableWizard - Unity 4 でイケてないトコロ

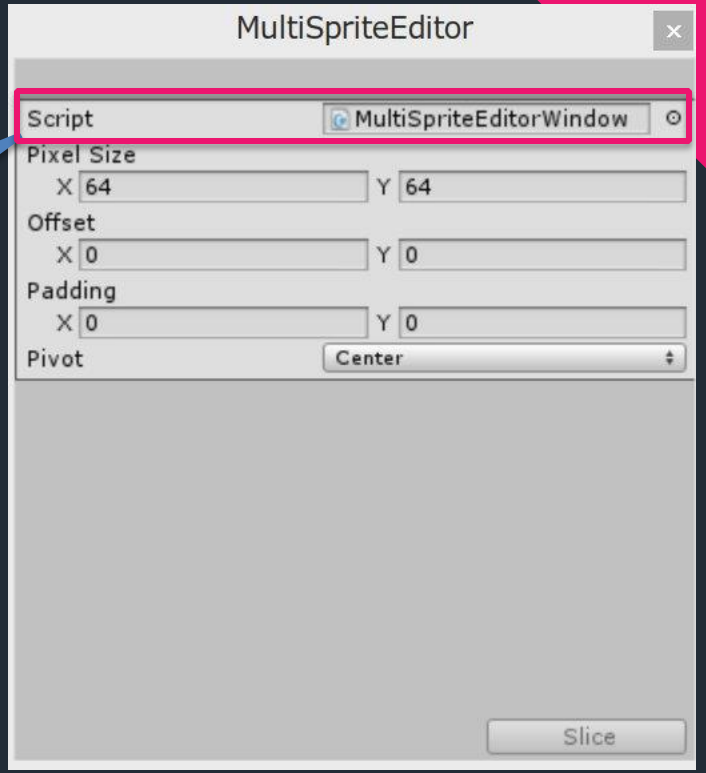
GUIの表示方法がいまいち

表示はインスペクターと同じ

> Editor.DrawDefaultInspectorを実行してる

Script の ObjectField いらないよね...

> インスペクターと同じ表示をしているためこうなる



ScriptableWizard - Unity 4 でイケてないトコロ

GUIの表示方法がいまいち

表示はインスペクターと同じ

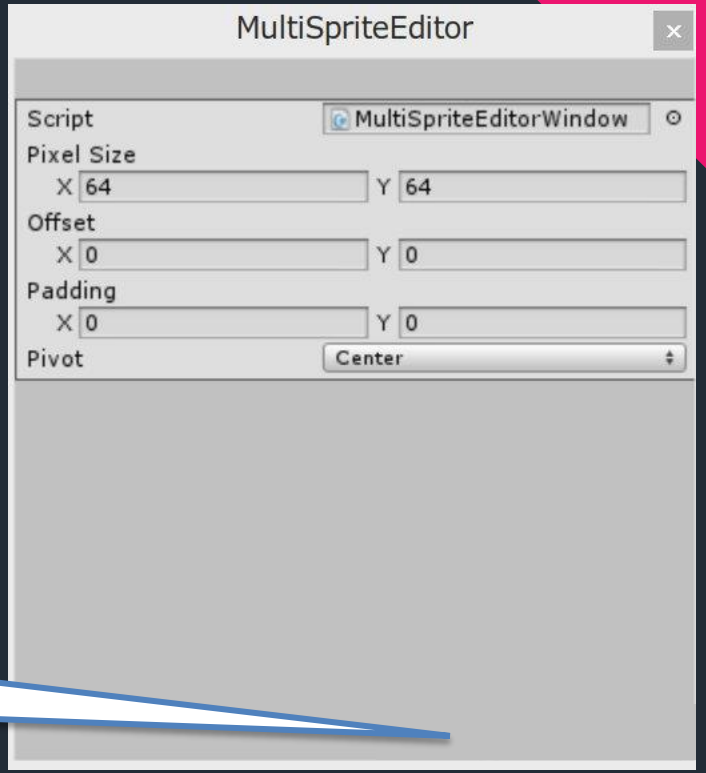
> Editor.DrawDefaultInspectorを実行してる

Script の ObjectField いらないよね...

> インスペクターと同じ表示をしているためこうなる

OnGUI を使うと Button が消える

> ただの EditorWindow になってしまう...



ScriptableWizard - Unity 5 でイケてるトコロ

DrawWizardGUI の追加（解放）

ScriptableWizard - Unity 5 でイケてるトコロ

DrawWizardGUI の追加（解放）

```
protected override bool DrawWizardGUI()
{
    EditorGUILayout.LabelField("Label");


    return true;
}
```

ここに GUI コードを記述する

base.DrawWizardGUI でデフォルトの表示が行われる

ScriptableWizard - Unity 5 でイケてるトコロ

DrawWizardGUI の追加（解放）

```
protected override bool DrawWizardGUI()
{
    EditorGUILayout.LabelField("Label");


    return true;
}
```

ここに GUI コードを記述する

- base.DrawWizardGUI でデフォルトの表示が行われる

True で WizardUpdate が実行される

- isValidate を WizardUpdate 内で使用している場合は注意

ScriptableWizard – まとめ

より柔軟にウィザードを作成できるようになった

- わざわざ PropertyDrawer で拡張するのが不要になった
- GUI の 記述の自由度が上がった

ReorderableList

配列を扱う時にかなり便利な機能

要素の順番を楽に変更できる

左側の＝のつまみで可能

要素の追加・削除も楽

要素を選択して＋や－ボタンをクリック

どんどん使えばいいと思うよ！

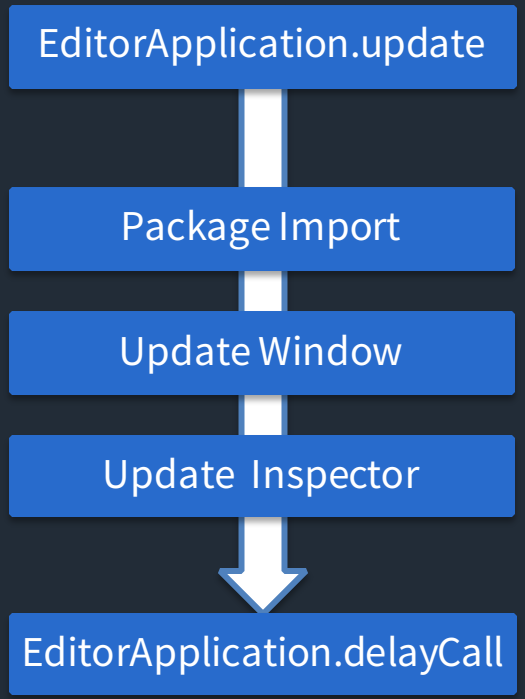
UnityEditorInternalだけど…

EditorApplication.delayCall

インスペクターのアップデート後に呼び出されるデリゲート

EditorApplication.delayCall

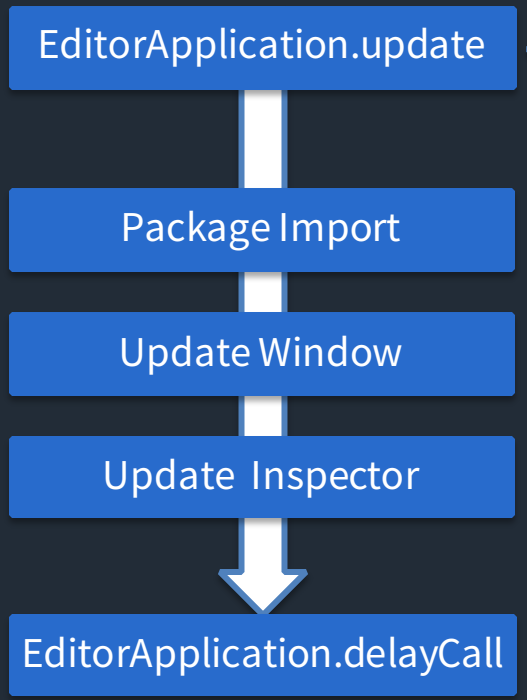
インスペクターのアップデート後に呼び出されるデリゲート



http://qiita.com/kyusyukeigo/private/794b400b549f028974b8

EditorApplication.delayCall

インスペクターのアップデート後に呼び出されるデリゲート



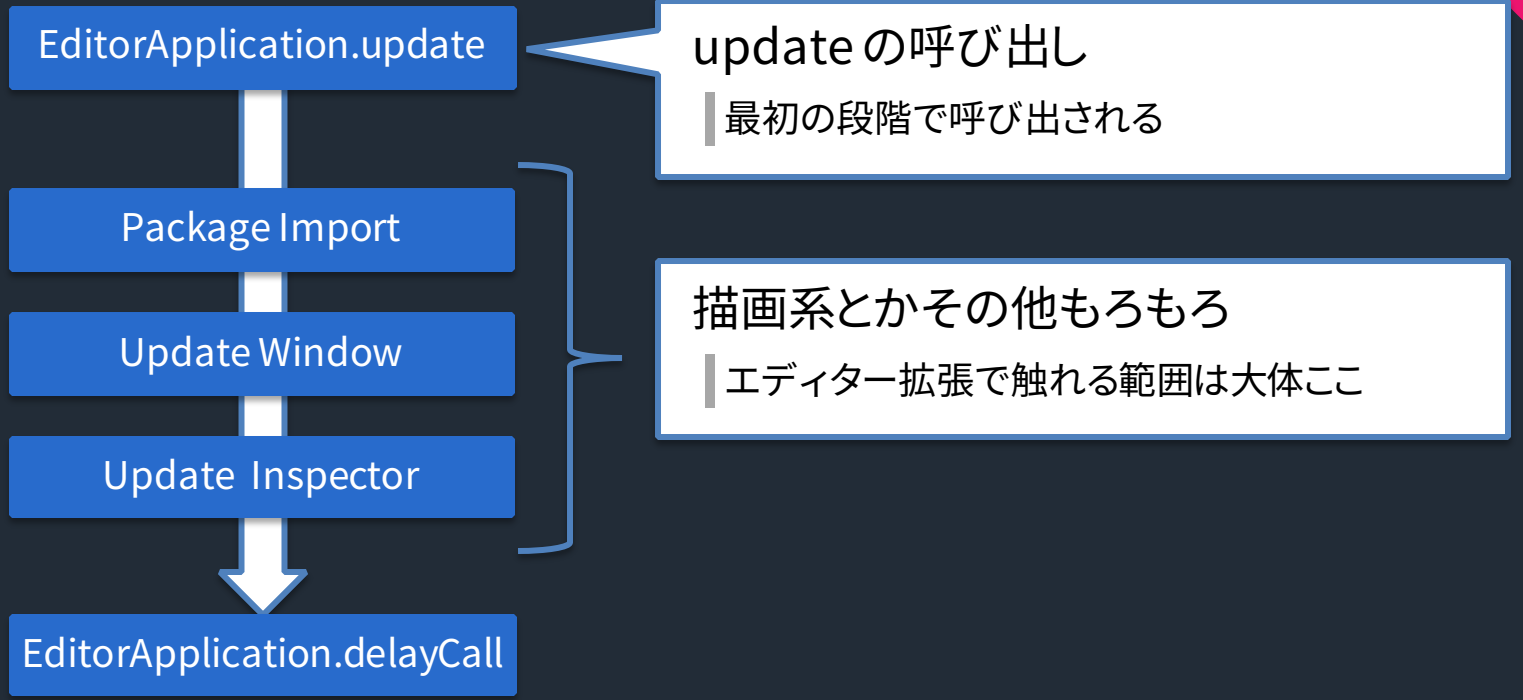
update の呼び出し

> 最初の段階で呼び出される

http://qiita.com/kyusyukeigo/private/794b400b549f028974b8

EditorApplication.delayCall

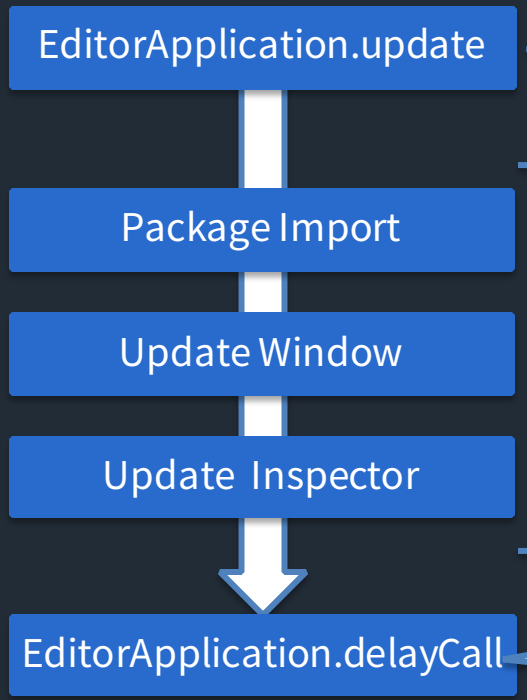
インスペクターのアップデート後に呼び出されるデリゲート



http://qiita.com/kyusyukeigo/private/794b400b549f028974b8

EditorApplication.delayCall

インスペクターのアップデート後に呼び出されるデリゲート



update の呼び出し

最初の段階で呼び出される

描画系とかその他もろもろ

エディター拡張で触れる範囲は大体ここ

delayCall の呼び出し

最後の段階で呼び出される。一度呼び出したら破棄

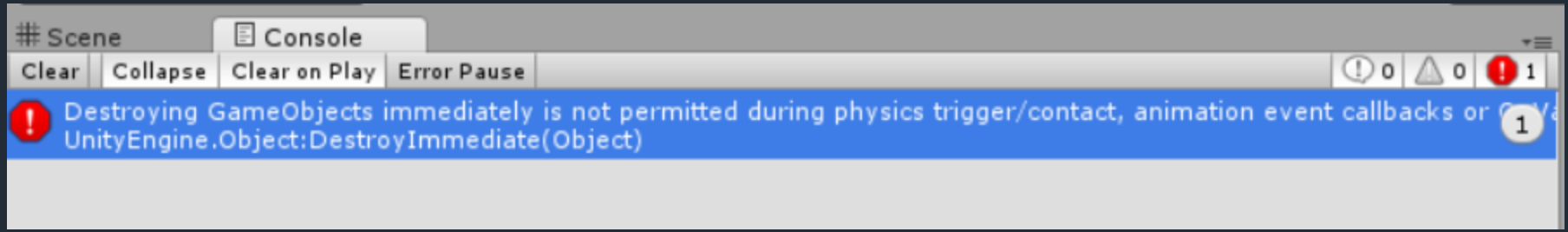
http://qiita.com/kyusyukeigo/private/794b400b549f028974b8

EditorApplication.delayCall

適当な実装（どうしようもない対処）をしたいとき

```
void OnValidate()
{
    DestroyImmediate(go);
    go = null;
}
```

OnValidate内で
変数のGameObjectを削除したいとき



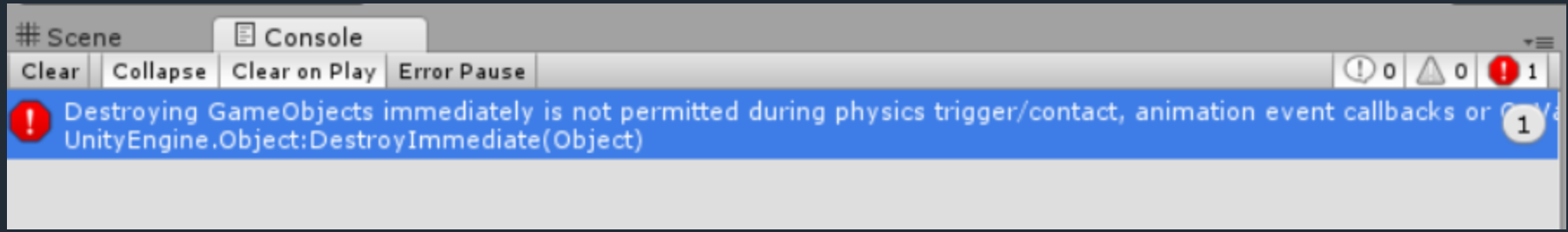
EditorApplication.delayCall

適当な実装（どうしようもない対処）をしたいとき

```
void OnValidate()
{
    DestroyImmediate(go);
    go = null;
}
```

OnValidate内で
変数のGameObjectを削除したいとき

OnValidateはインスペクターの更新中に
呼び出されるため破棄が出来ない



EditorApplication.delayCall

適当な実装（どうしようもない対処）をしたいとき

```
void OnValidate()
{
    EditorApplication.delayCall += () =>
    {
        DestroyImmediate(go);
        go = null;
    };
}
```

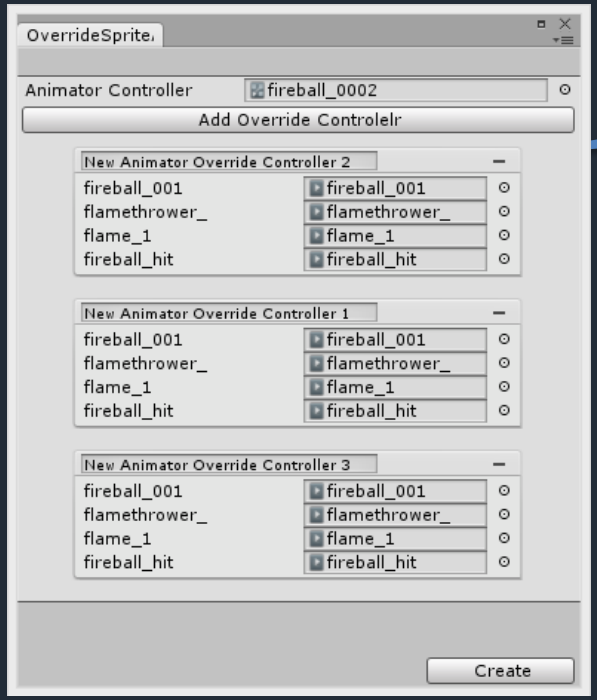
インスペクターの更新後に呼び出されるため問題ない

都合が悪くなった時に使うといいかも

Animator Override Controller の作成

こんな話をしてましたよ！

複数の Animator Override Controller を作るなら



ScriptableWizardでささっと作成する

ReorderableList

使うと配列が見やすくなる

AssetDatabase.GenerateUniqueAssetPath

同名のアセットがあってはだめなのでユニークな名前にする

EditorApplication.delayCall

インスペクターアップデート後に呼び出されるデリゲート

Unity5の新機能

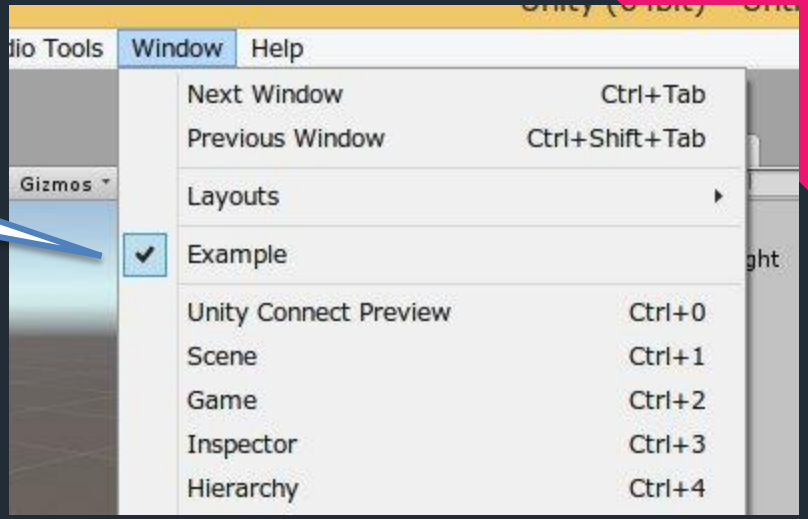
UnityEditor.InitializeOnLoadMethod

InitializeOnLoad のメソッド版

```
public class NewBehaviourScript
{
    [InitializeOnLoadMethod]
    static void Hoge()
    {
        Debug.Log("スクリプトコンパイル後に呼び出される");
    }

    [InitializeOnLoadMethod]
    static void Fuga()
    {
        Debug.Log("スクリプトコンパイル後に呼び出される");
    }
}
```

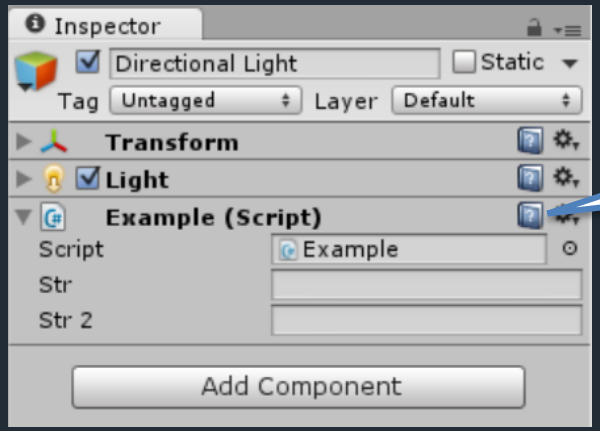
UnityEditor.Menu

チェックつけるための API




```
const string MENU_PATH = "Window/Example";

[MenuItem(MENU_PATH)]
static void Example()
{
    var @checked = Menu.GetChecked(MENU_PATH);
    Menu.SetChecked(MENU_PATH, !@checked);
}
```

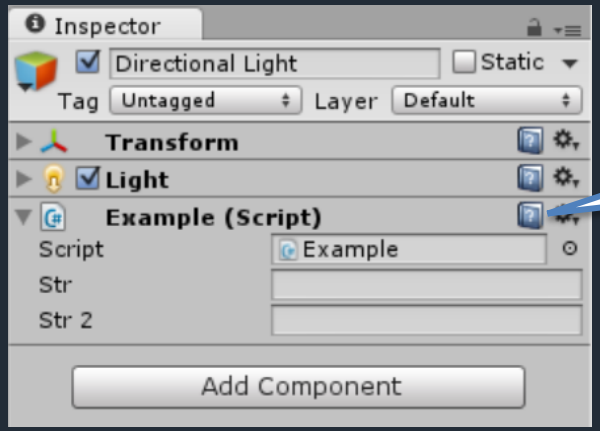
UnityEngine.HelpURLAttribute



これ

> ジャンプ先を変更するための機能

UnityEngine.HelpURLAttribute



これ

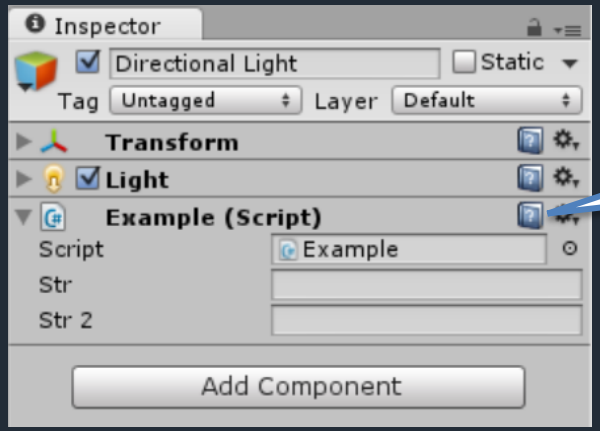
> ジャンプ先を変更するための機能

http , file などを指定

> 機能自体はすごく単純

```
[HelpURL("http://japan.unity3d.com")]
public class Example : MonoBehaviour
{
}
```

UnityEngine.HelpURLAttribute



これ

> ジャンプ先を変更するための機能

http , file などを指定

> 機能自体はすごく単純

```
[HelpURL("http://japan.unity3d.com")]
public class Example : MonoBehaviour
{
}
```

アセット製作者・チーム向け
メモがてらに使う
ドキュメント環境を改善しましょう

GUI.Scope

BeginHorizontalやEndHorizontalをIDisposableでラップしたもの

```
EditorGUILayout.BeginHorizontal();

EditorGUILayout.LabelField("Label");
GUILayout.Button("Button");

EditorGUILayout.EndHorizontal();
```

いままで

> 複雑になってくると段々ややこしくなってくる

GUI.Scope

BeginHorizontalやEndHorizontalをIDisposableでラップしたもの

```
EditorGUILayout.BeginHorizontal();

EditorGUILayout.LabelField("Label");
GUILayout.Button("Button");

EditorGUILayout.EndHorizontal();
```

いままで

> 複雑になってくると段々ややこしくなってくる

```
using (new EditorGUILayout.HorizontalScope())
{
        EditorGUILayout.LabelField("Label");
        GUILayout.Button("Button");
}
```

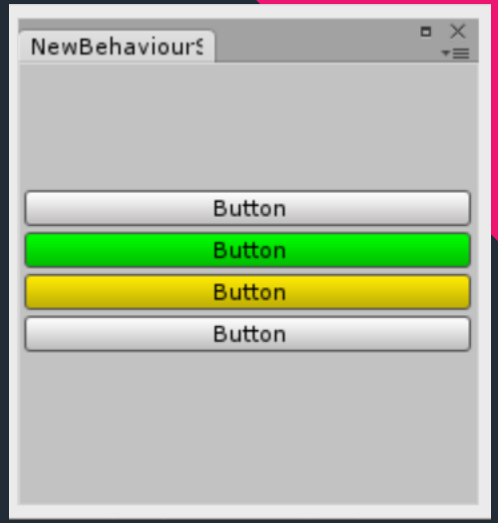
これから

> 見た目的にもスッキリ

GUI.Scope

自作のScopeを作成できる

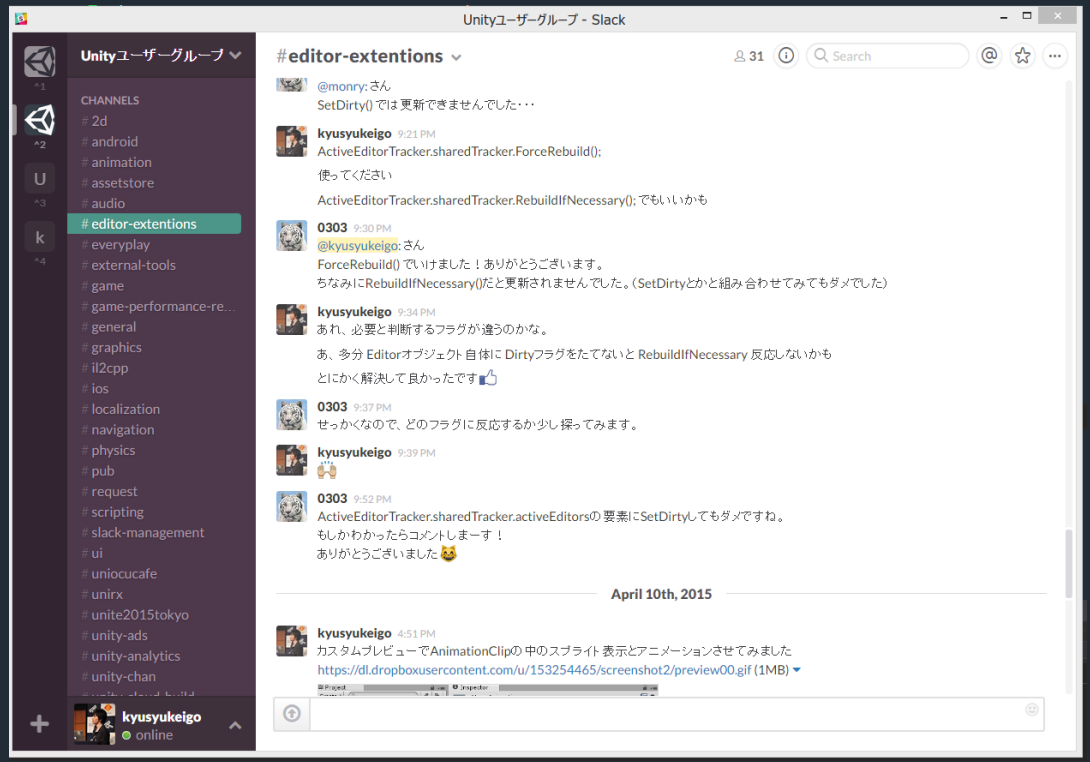
```
using UnityEngine;

public class BackgroundColorScope : GUI.Scope
{
    private Color color;

    public BackgroundColorScope(Color color)
    {
        this.color = GUI.backgroundColor;
        GUI.backgroundColor = color;
    }
    protected override void CloseScope()
    {
        GUI.backgroundColor = color;
    }
}
```




```
using (new BackgroundColorScope(Color.yellow))
{
    GUILayout.Button("Button");
}
```

他に知りたいことある？

Slackの Unity ユーザーグループに参加しましょう



ご清聴ありがとうございました

質問タイム

おまけ

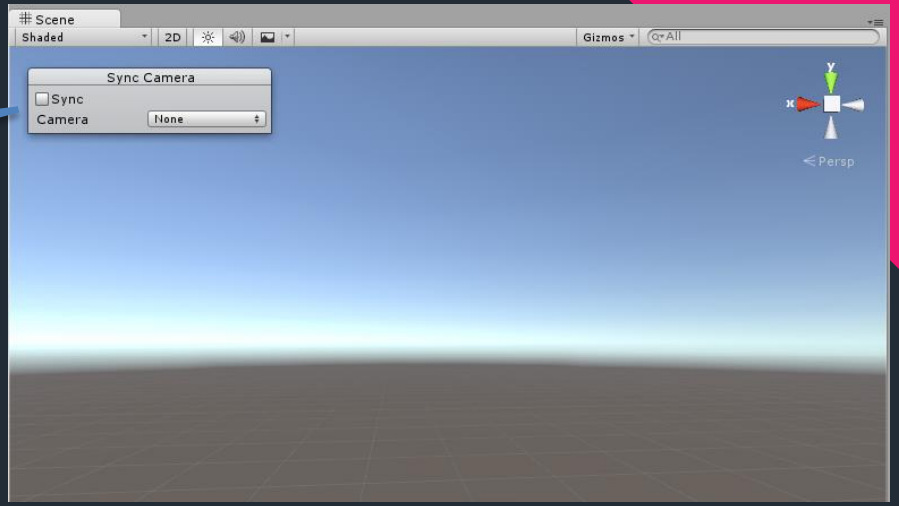
ドキュメントにはないエディター API

Unity にはパブリックなメンバーだけどドキュメント化されていない
APIがたくさんある

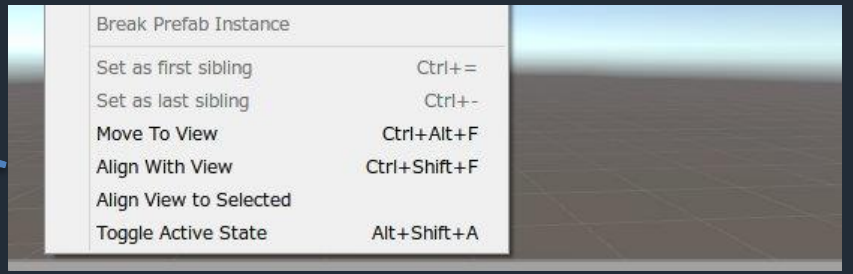
ドキュメントにはないエディター API

SceneView クラス

Editor.OnScene ではなく、グローバルな onSceneGUIDelegate がある



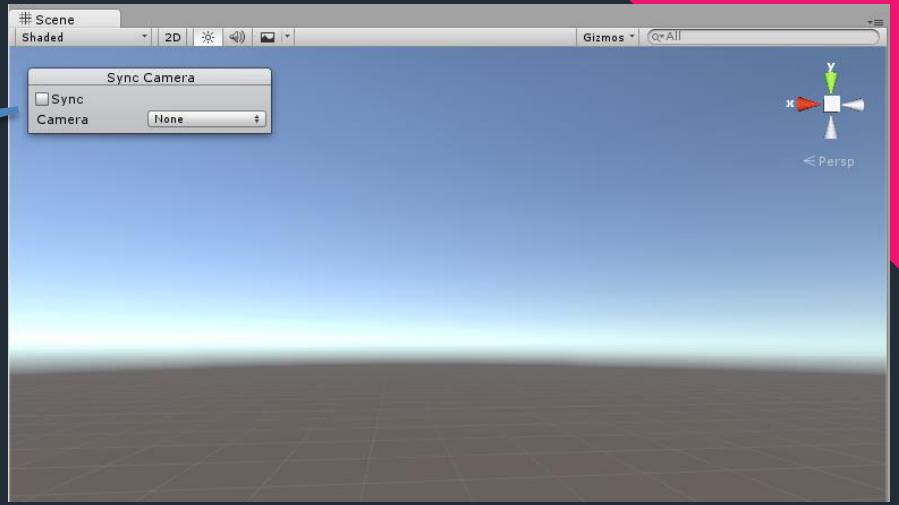
シーンビューを操作できる様々なAPIがある



ドキュメントにはないエディター API

SceneView クラス

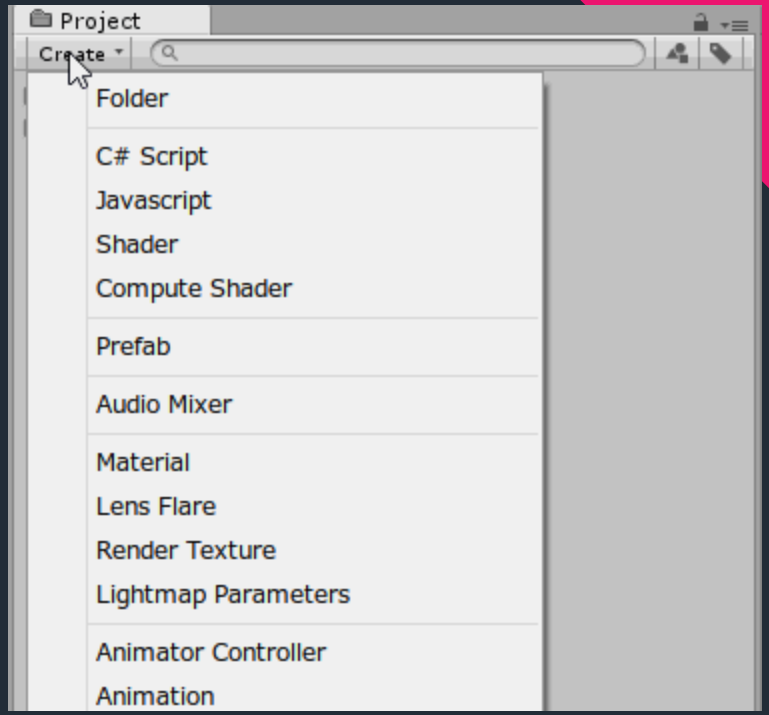
Editor.OnScene ではなく、グローバルな onSceneGUIDelegate がある



ドキュメントにはないエディター API

ProjectWindowUtil クラス

メニューからアセットを作成するときに
使用されるAPI

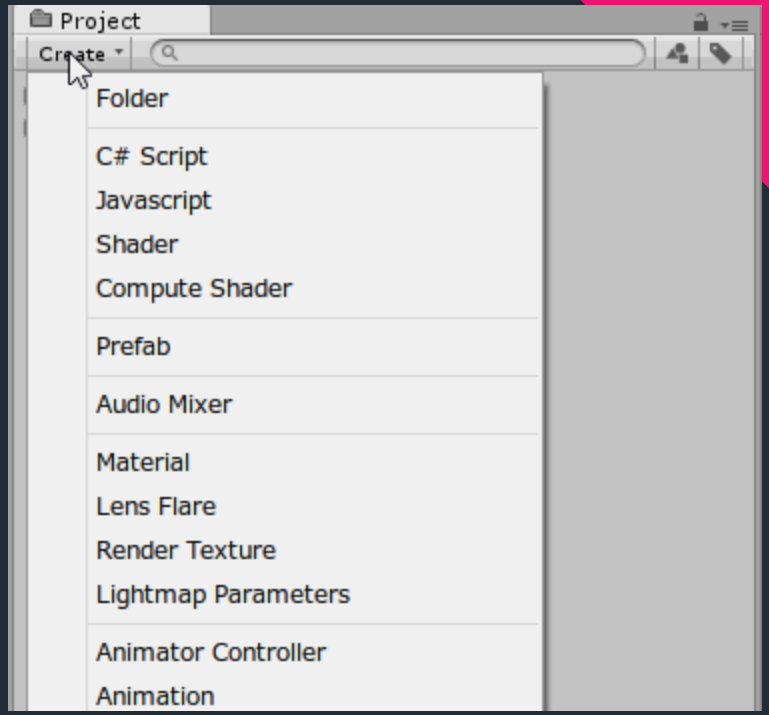


ドキュメントにはないエディター API

ProjectWindowUtil クラス

メニューからアセットを作成するときに
使用されるAPI

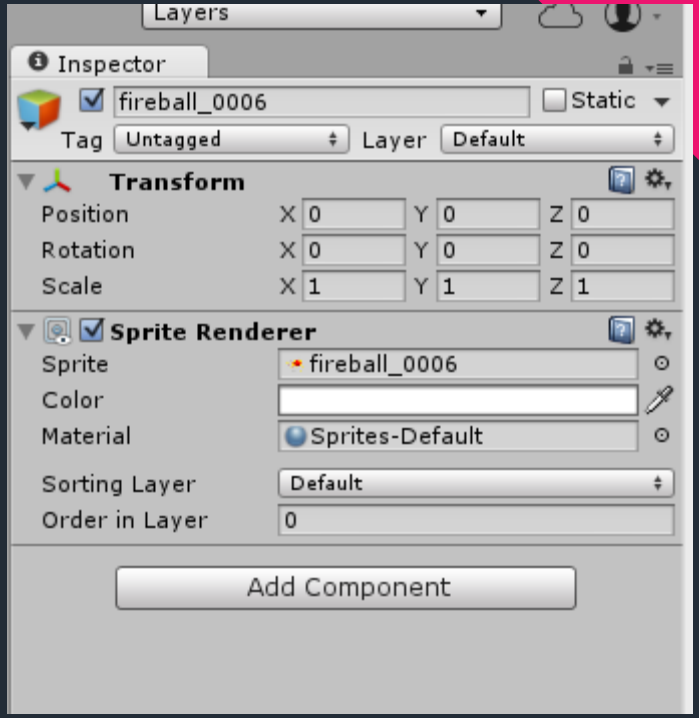
ProjectWindowUtil.CreateAsset は
覚えておくと絶対便利!



ドキュメントにはないエディター API

ActiveEditorTracker クラス

現在インスペクターに表示されている Editor オブジェクトを管理する API

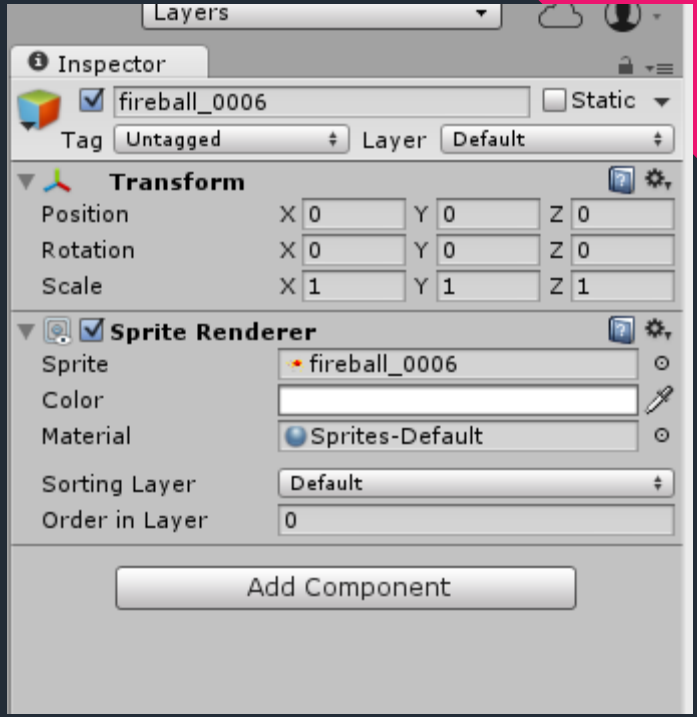


ドキュメントにはないエディター API

ActiveEditorTracker クラス

現在インスペクターに表示されている Editor オブジェクトを管理する API

Editor オブジェクトを操作するときに便利

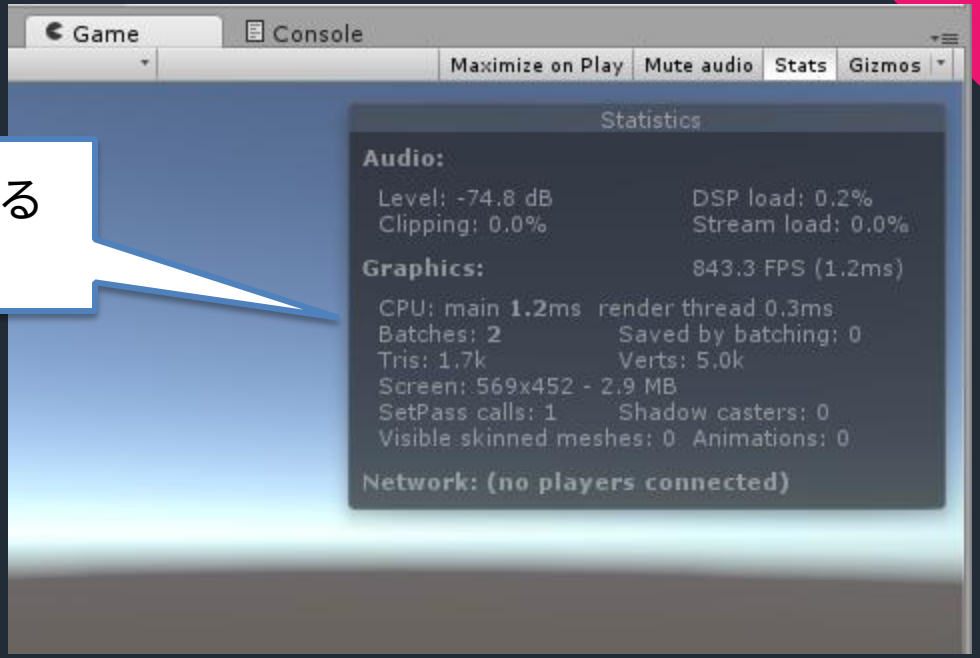



```
var activeEditors =
          ActiveEditorTracker.sharedTracker.activeEditors;

foreach (var editor in activeEditors)
{
      editor.Repaint();
}
```

ドキュメントにはないエディター API

UnityStats クラス

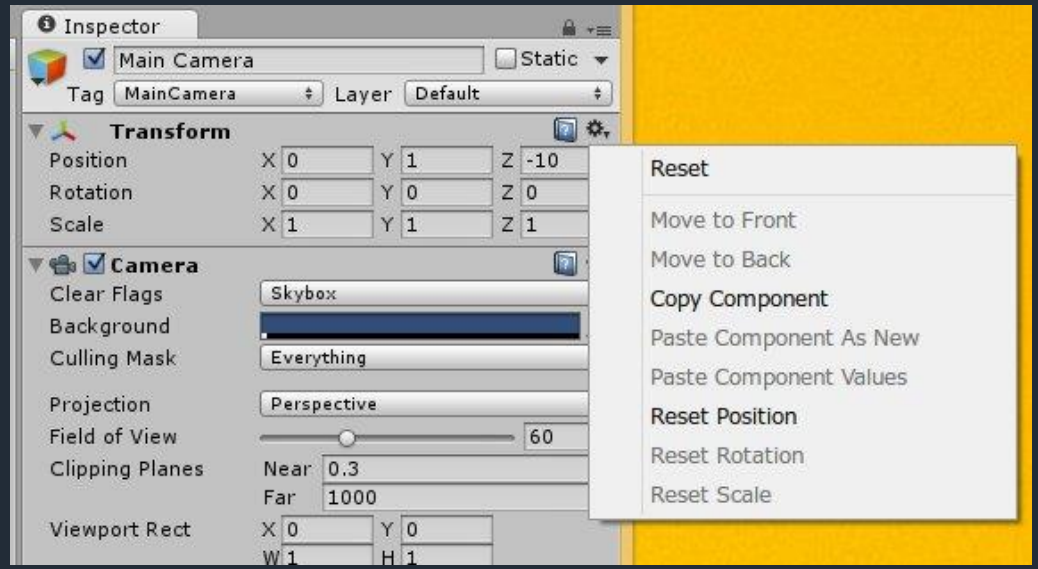
現在の描画に関する情報などを取得するためのAPI



インターナルなエディター API

ComponentUtility クラス

コンポーネントのコピーが行えるAPI



インターナルなエディター API

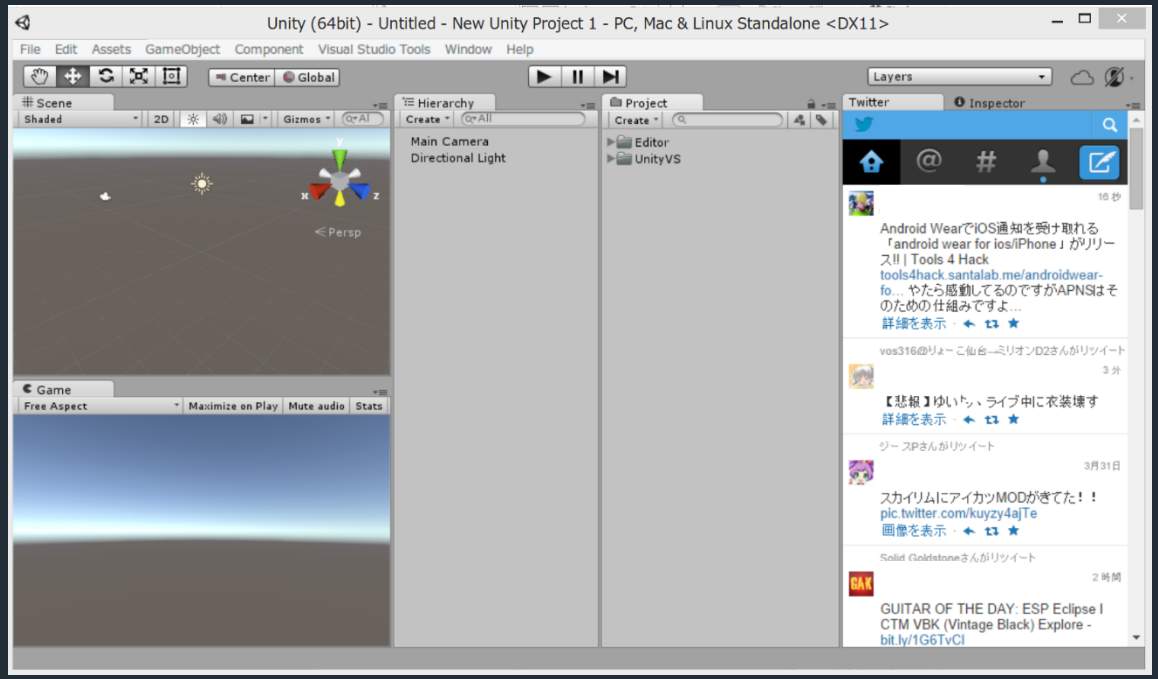
InternalEditorUtilityクラス

- なんか色々おもしろいAPIがあるクラス（雑）
  - タグやレイヤーの一覧が取得できたり
  - UnityのPreferenceフォルダのパスが取得できたり
  - オブジェクトをシリアライズできたり

本来触ることのできないエディターAPI

本来触ることのできないエディターAPI

ああ…!! Unityエディター上でTwitterを見たい…!!!



本来触ることのできないエディターAPI

リフレクションを使えばどうにでもなる!!

本来触ることのできないエディターAPI

リフレクションを使えばどうにでもなる!!

リフレクションとは

実行時にクラス情報を取得して、実行する技術

なんでこれ public クラスじゃないの! 使いたいのに! というための技術

エディター拡張ではこのために使用することが多々

本来触ることのできないエディターAPI

```
public class TwitterWindow
{
    private const BindingFlags Flags = BindingFlags.Public | BindingFlags.Static;

    [MenuItem("Window/Twitter")]
    private static void Open()
    {
        var type = Types.GetType("UnityEditor.Web.WebViewEditorWindow", "UnityEditor.dll");

        var methodInfo = type.GetMethod("Create", Flags);

        methodInfo = methodInfo.MakeGenericMethod(typeof(TwitterWindow));

        methodInfo.Invoke(null, new object[]
        {
            "Twitter",
            "https://mobile.twitter.com/",
            200, 520, 520, 600
        });
    }
}
```

リフレクションを使ってみる

本来触ることのできないエディターAPI

```
public class TwitterWindow : EditorWindow
{
    private const BindingFlags Flags = Bindi

    [MenuItem("Window/Twitter")]
    private static void Open()
    {
        var type = Types.GetType("UnityEditor.Web.WebViewEditorWindow", "UnityEditor.dll");

        var methodInfo = type.GetMethod("Create", Flags);

        methodInfo = methodInfo.MakeGenericMethod(typeof(TwitterWindow));

        methodInfo.Invoke(null, new object[]
        {
            "Twitter",
            "https://mobile.twitter.com/",
            200, 520, 520, 600
        });
    }
}
```

UnityEngine.Typesクラス

> Typeを取得するためのヘルパークラス

本来触ることのできないエディターAPI

```
public class TwitterWindow : EditorWindow
{
    private const BindingFlags Flags = Bindi

    [MenuItem("Window/Twitter")]
    private static void Open()
    {
        var type = Types.GetType("UnityEditor.Web.WebViewEdito

        var methodInfo = type.GetMethod("Create", Flags);

        methodInfo = methodInfo.MakeGenericMethod(typeof(TwitterWindow));

        methodInfo.Invoke(null, new object[]
        {
            "Twitter",
            "https://mobile.twitter.com/",
            200, 520, 520, 600
        });
    }
}
```

UnityEngine.Typesクラス

> Typeを取得するためのヘルパークラス

メソッド情報取得

本来触ることのできないエディターAPI

```
public class TwitterWindow : EditorWindow
{
    private const BindingFlags Flags = Bindi

    [MenuItem("Window/Twitter")]
    private static void Open()
    {
        var type = Types.GetType("UnityEditor.Web.WebViewEdito

        var methodInfo = type.GetMethod("Create", Flags);

        methodInfo = methodInfo.MakeGenericMethod(typeof(TwitterWindow));

        methodInfo.Invoke(null, new object[]
        {
            "Twitter",
            "https://mobile.twitter.com/",
            200, 520, 520, 600
        });
    }
}
```

UnityEngine.Typesクラス

> Typeを取得するためのヘルパークラス

メソッド情報取得

ジェネリックメソッド作成

本来触ることのできないエディターAPI

```
public class TwitterWindow : EditorWindow
{
    private const BindingFlags Flags = Bindi

    [MenuItem("Window/Twitter")]
    private static void Open()
    {
        var type = Types.GetType("UnityEditor.Web.WebViewEdito

        var methodInfo = type.GetMethod("Create", Flags);

        methodInfo = methodInfo.MakeGenericMethod(typeof(TwitterWindow));

        methodInfo.Invoke(null, new object[]
        {
            "Twitter",
            "https://mobile.twitter.com/",
            200, 520, 520, 600
        });
    }
}
```

UnityEngine.Typesクラス

Typeを取得するためのヘルパークラス

メソッド情報取得

ジェネリックメソッド作成

メソッド実行

本来触ることのできないエディターAPI

Types.GetTypeは、よく使うので覚えておくと便利!

```
public static class Types
 {
   public static System.Type GetType(string typeName, string assemblyName)
   {
     try
     {
       return Assembly.Load(assemblyName).GetType(typeName);
     }
     catch (Exception ex)
     {
       return (System.Type) null;
     }
   }
 }
```

Try-catch をしてくれる

本来触ることのできないエディターAPI

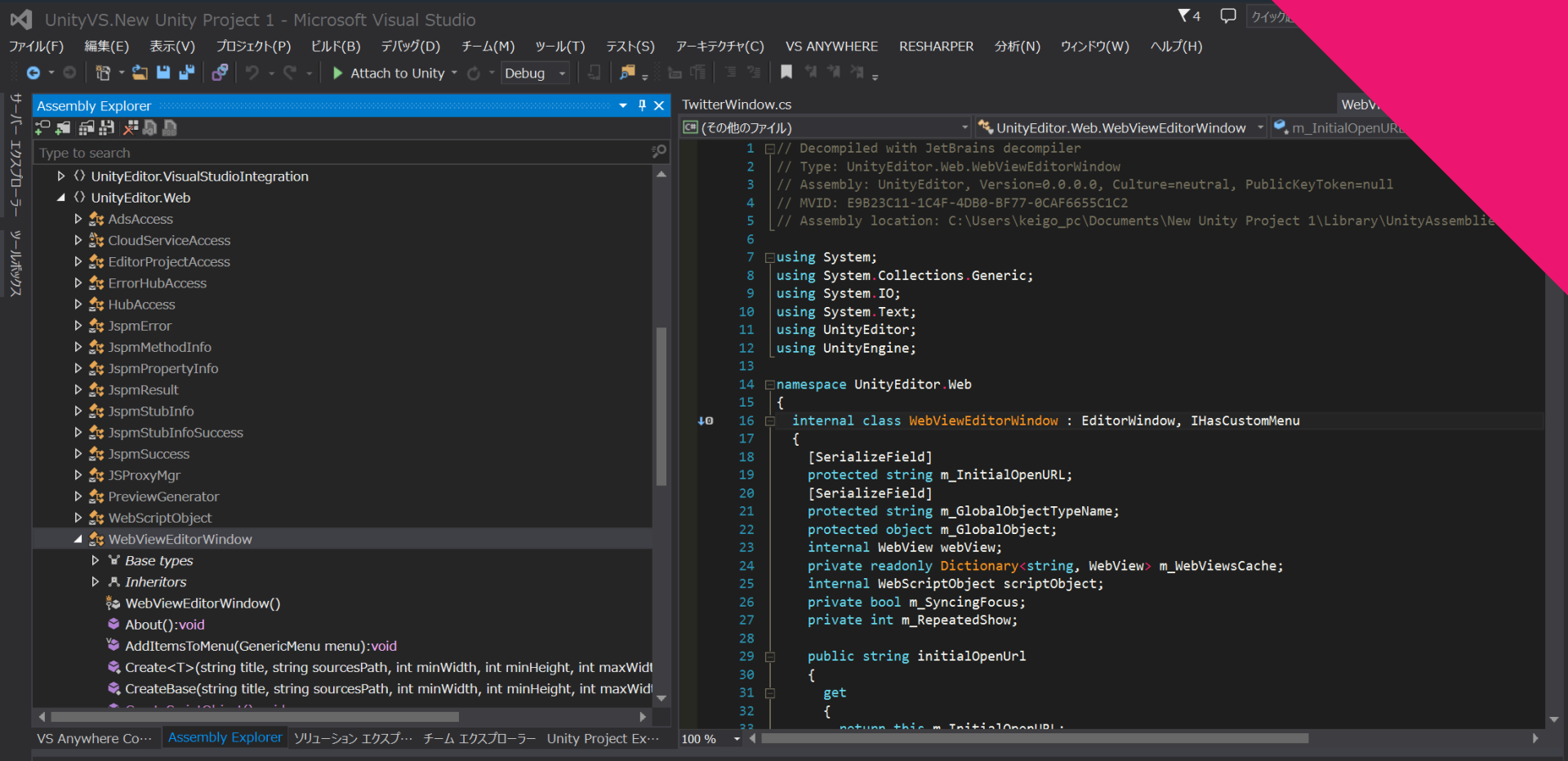
リフレクションって言うけどUnityには

「どのようなクラスが」

「どこにあって」

「どんな変数やメソッドを持っているか」がわからない

本来触ることのできないエディターAPI



本来触ることのできないエディターAPI

アセンブリブラウザとは

アセンブリをデコンパイルして可読性の高いコードを生成
階層構造をツリー状に表示できる
クラスの検索
シンタックスハイライト

アセンブリブラウザ（デコンパイラ）は様々なソフトに実装されている

Monodevelop、Resharper、dotPeek、ILSpy ...

本来触ることのできないエディターAPI

EditorGUIを見たり…

```
13  using UnityEngine;
14  using UnityEngine.Internal;
15
16  namespace UnityEditor
17  {
18    /// <summary/>
19    public sealed class EditorGUI
20    {
21      internal static EditorGUI.DelayedTextEditor s_DelayedTextEditor = new EditorGUI.DelayedTextEditor();
22      internal static EditorGUI.RecycledTextEditor s_RecycledEditor = new EditorGUI.RecycledTextEditor();
23      internal static string s_OriginalText = string.Empty;
24      private static bool bKeyEventActive = false;
25      internal static bool s_DragToPosition = true;
26      internal static bool s_Dragged = false;
27      internal static bool s_PostPoneMove = false;
28      internal static bool s_SelectAllOnMouseUp = true;
29      private static int s_DragUpdatedOverID = 0;
30      private static int s_FoldoutHash = "Foldout".GetHashCode();
31      private static int s_TagFieldHash = "s_TagFieldHash".GetHashCode();
32      private static int s_PPtrHash = "s_PPtrHash".GetHashCode();
33      private static int s_ObjectFieldHash = "s_ObjectFieldHash".GetHashCode();
34      private static int s_ToggleHash = "s_ToggleHash".GetHashCode();
35      private static int s_ColorHash = "s_ColorHash".GetHashCode();
36      private static int s_CurveHash = "s_CurveHash".GetHashCode();
```

本来触ることのできないエディターAPI

UnityEditorInternal を見たり...

```
16  using UnityEngine;
17  using UnityEngine.Internal;
18
19  namespace UnityEditorInternal
20  {
21    public sealed class InternalEditorUtility
22    {
23      public static extern bool inBatchMode { [WrapperlessIcall, MethodImpl(MethodImplOptions.Intern
24
25      public static extern bool isHumanControllingUs { [WrapperlessIcall, MethodImpl(MethodImplOptio
26
27      public static extern int[] expandedProjectWindowItems { [WrapperlessIcall, MethodImpl(MethodIm
28
29      public static extern string[] tags { [WrapperlessIcall, MethodImpl(MethodImplOptions.InternalC
30
31      public static extern string[] layers { [WrapperlessIcall, MethodImpl(MethodImplOptions.Interna
32
33      internal static extern string[] sortingLayerNames { [WrapperlessIcall, MethodImpl(MethodImplOp
```

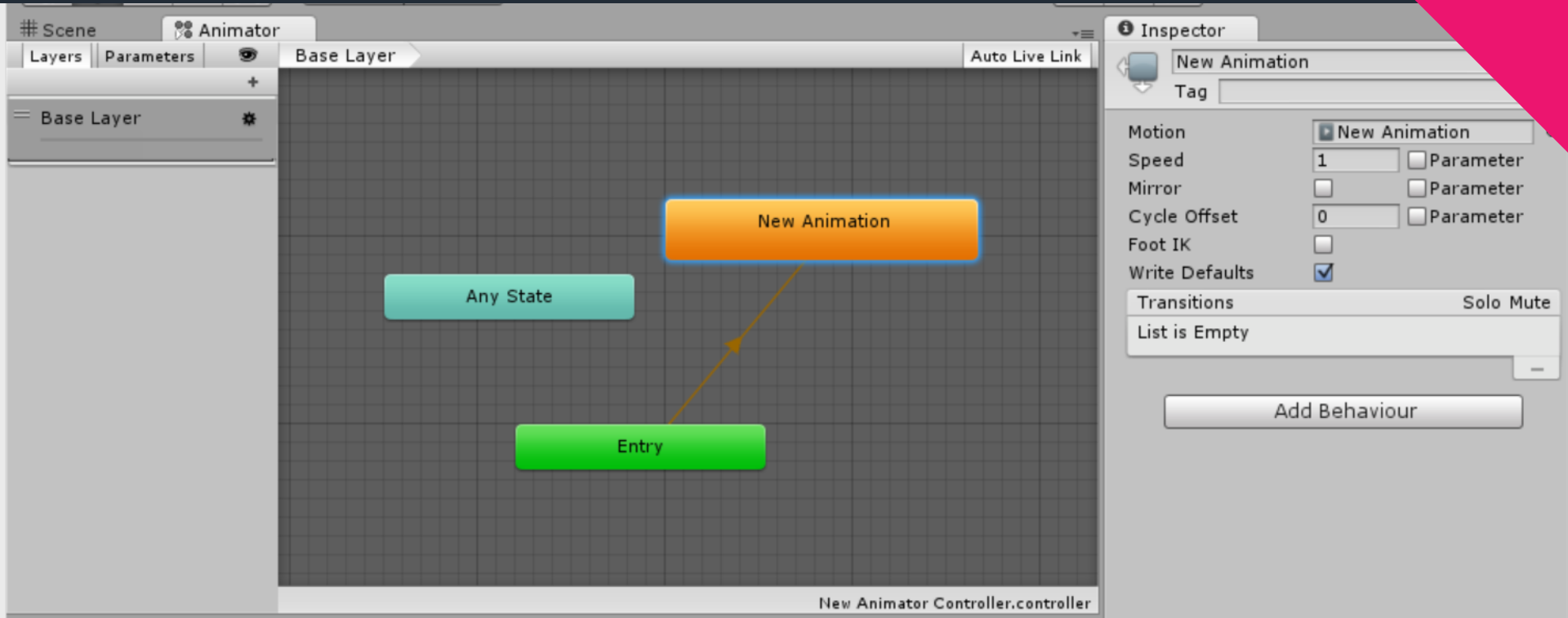
本来触ることのできないエディターAPI

UnityEditor.Graphs を見...なんだこれ

```
using System;
using System.Collections.Generic;
using System.Linq;
using UnityEditor;
using UnityEditorInternal;
using UnityEngine;

namespace UnityEditor.Graphs
{
  public class Graph : ScriptableObject
  {
    [SerializeField]
    public List<Node> nodes = new List<Node>();
    [SerializeField]
    public List<Edge> edges = new List<Edge>();
    [SerializeField]
    internal List<Edge> m_InvalidEdges = new List<Edge>();
    [NonSerialized]
    private List<Slot> m_changingOutputSlotTypesCycleSlots = new List<Slot>();
    [NonSerialized]
    internal Rect graphExtents;
```

UnityEditor.Graphs について

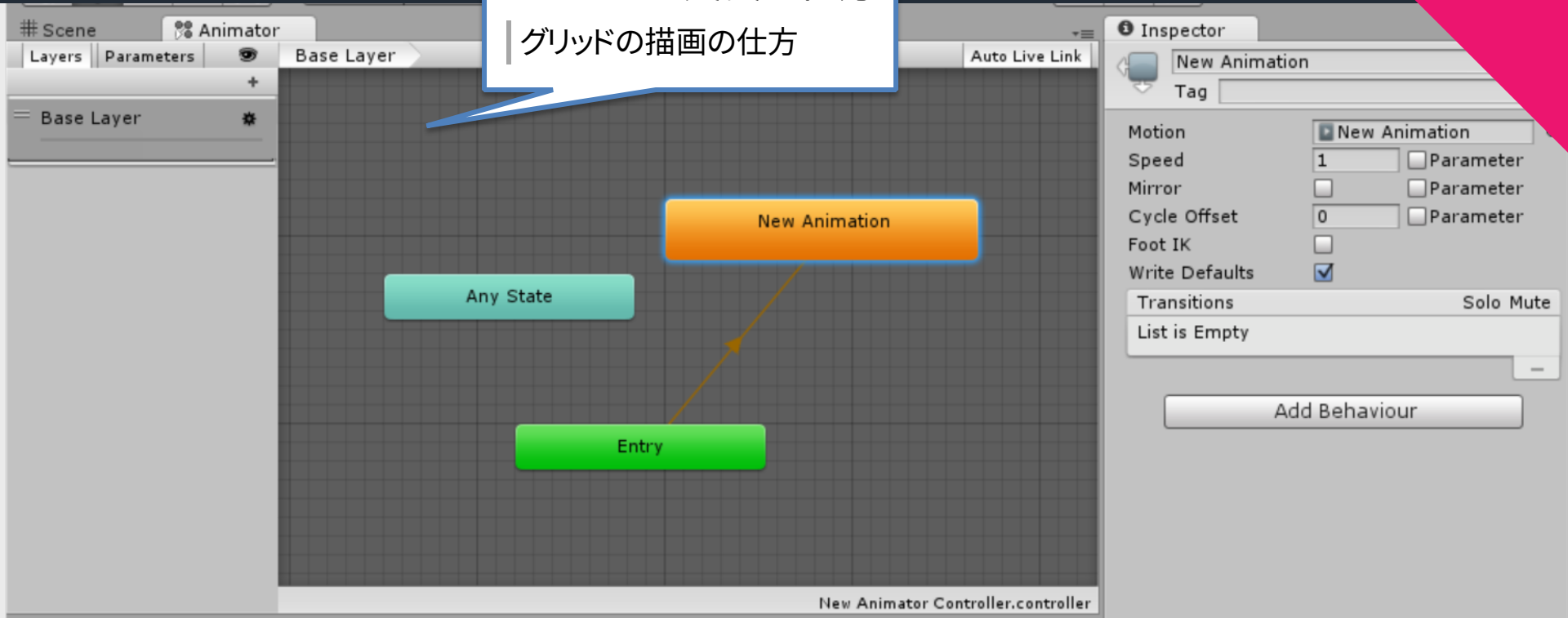
UnityEditor.Graphs はアニメーターで使われているライブラリ



UnityEditor.Graphs について

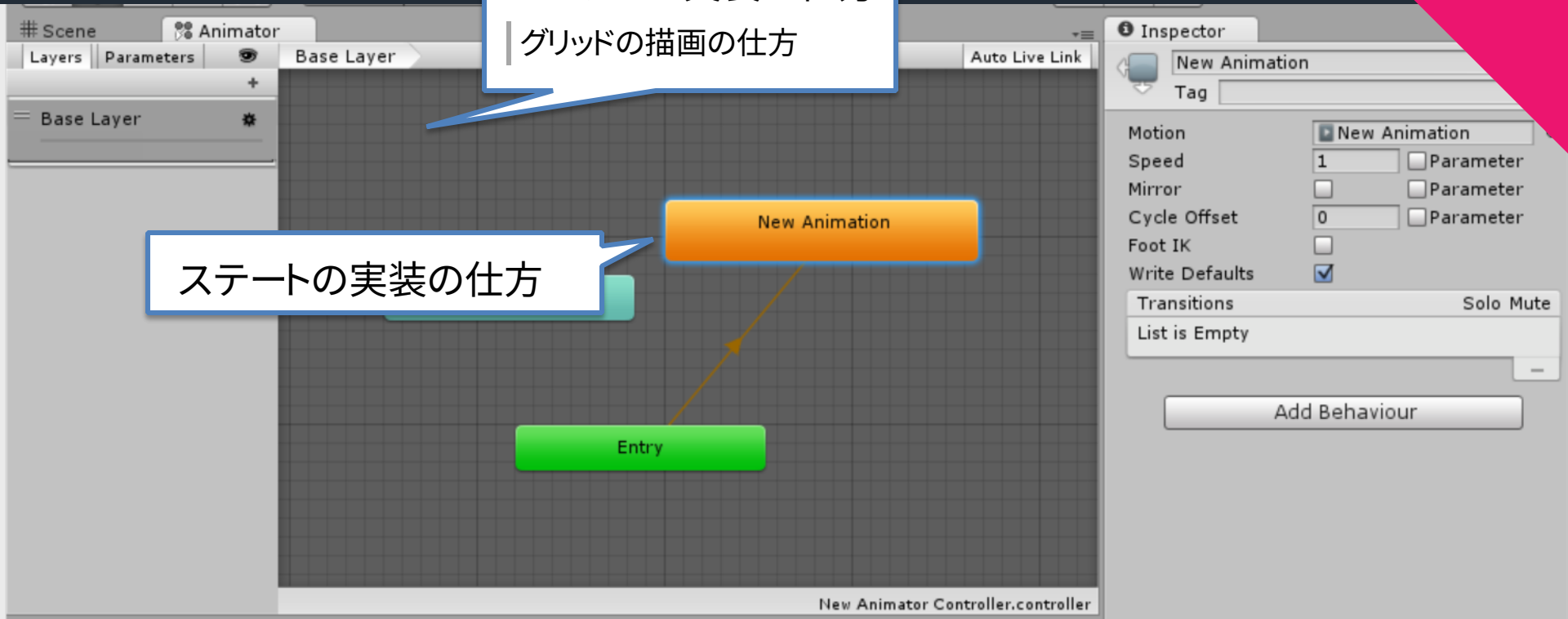
UnityEditor.Graphs はアニメーターで使われているライブラリ

- レイヤーの実装の仕方
  - グリッドの描画の仕方



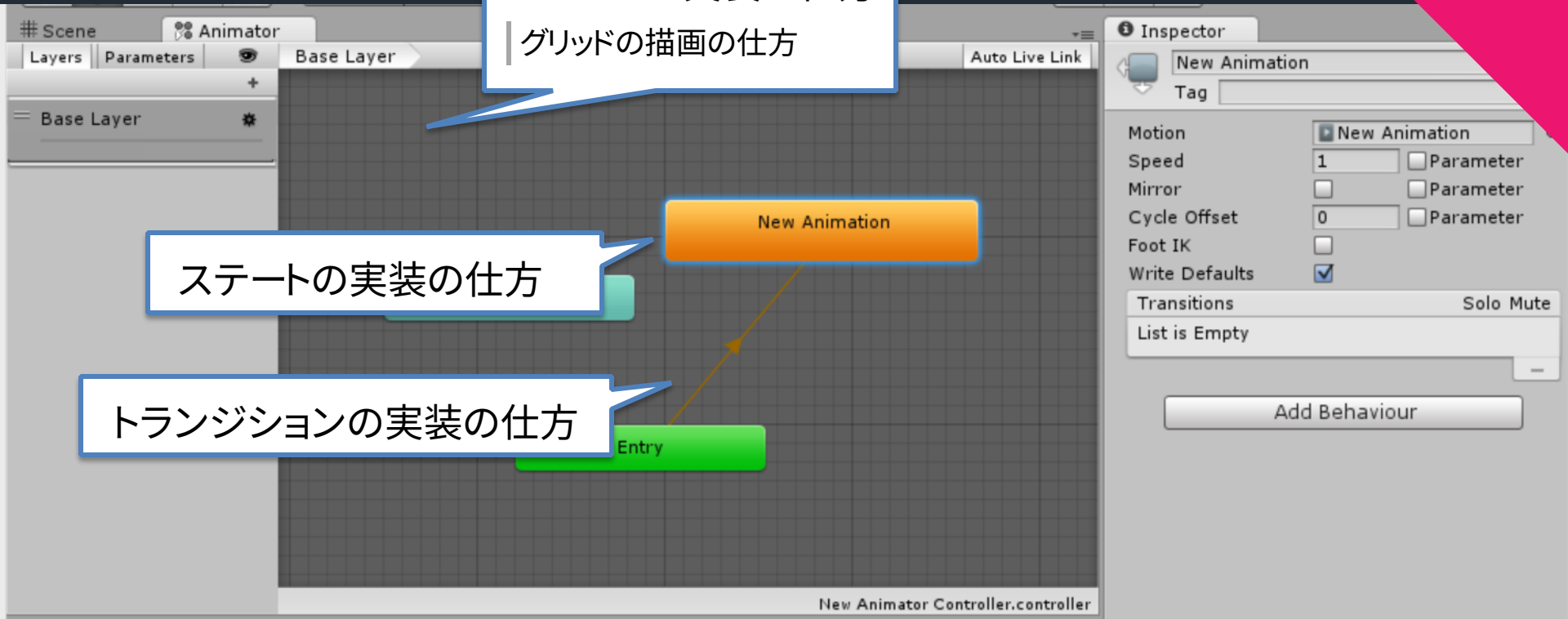
UnityEditor.Graphs について

UnityEditor.Graphs はアニメーターで使われているライブラリ



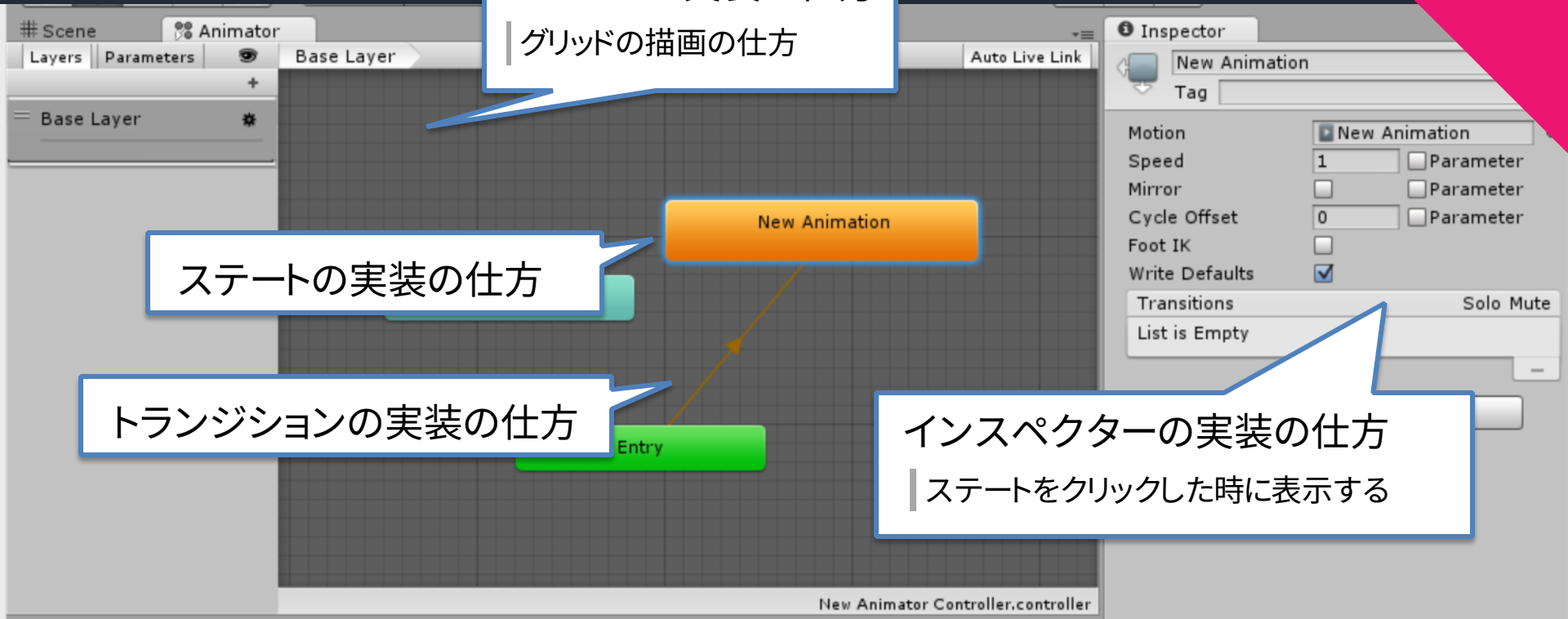
UnityEditor.Graphs について

UnityEditor.Graphs はアニメーターで使われているライブラリ



UnityEditor.Graphs について

UnityEditor.Graphs はアニメーターで使われているライブラリ



レイヤーの実装の仕方

グリッドの描画の仕方

ステートの実装の仕方

トランジションの実装の仕方

インスペクターの実装の仕方

ステートをクリックした時に表示する

UnityEditor.Graphs について

ネタの宝庫なのでコードを眺めるべき

ビジュアル系のエディター拡張を行いたい人は特に！

GUI.Window と ScriptableObject でステートを管理する方法は必見