演習１：UNITYスクリプトの基礎

(01) 04/15

１A｜Unityとエディタの連携

(02) 04/22

１B｜Transform・キーイベント・マウスイベント

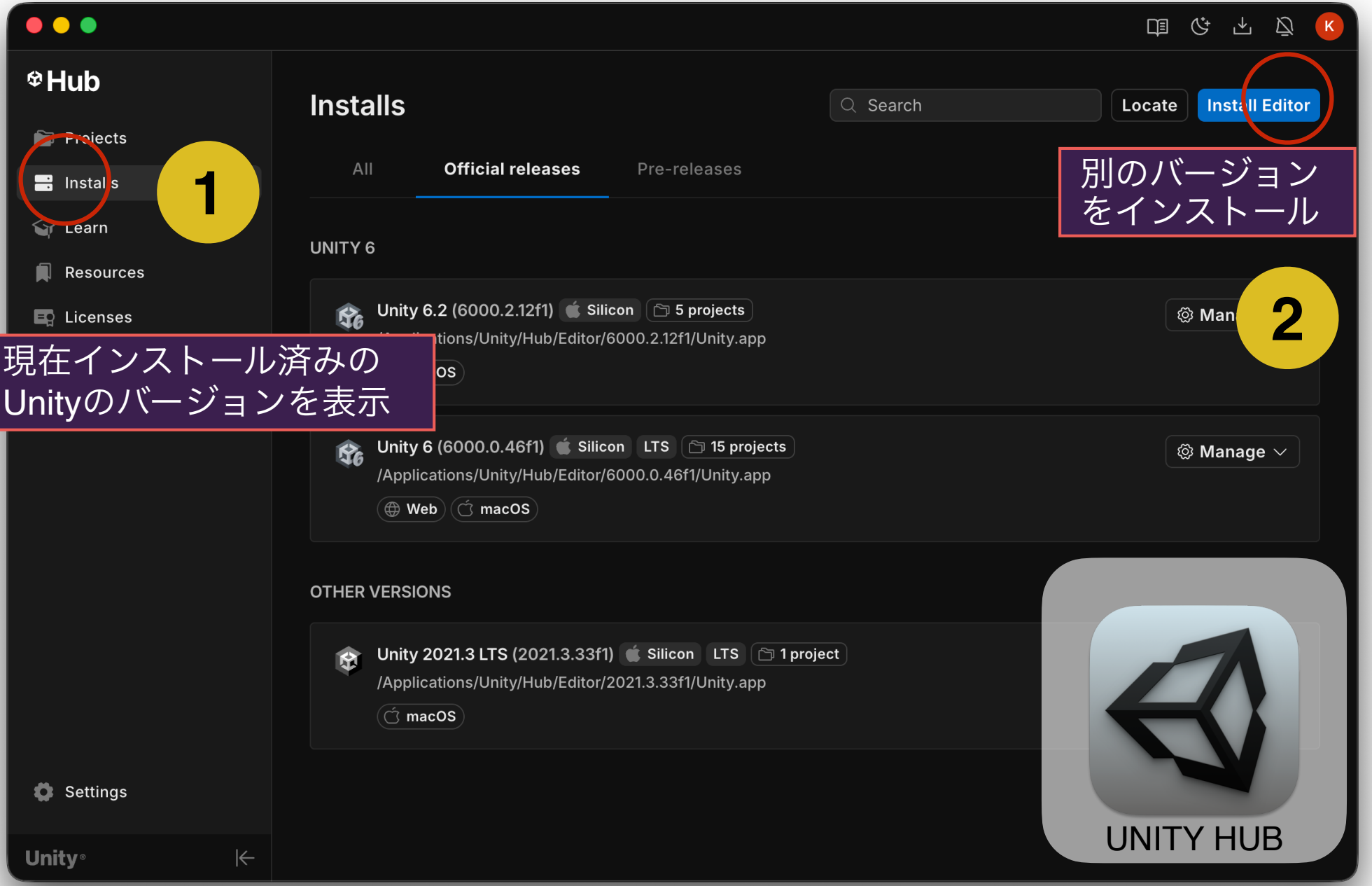
(03) 04/29

１C｜剛体特性・カメラの視点

(04) 05/06

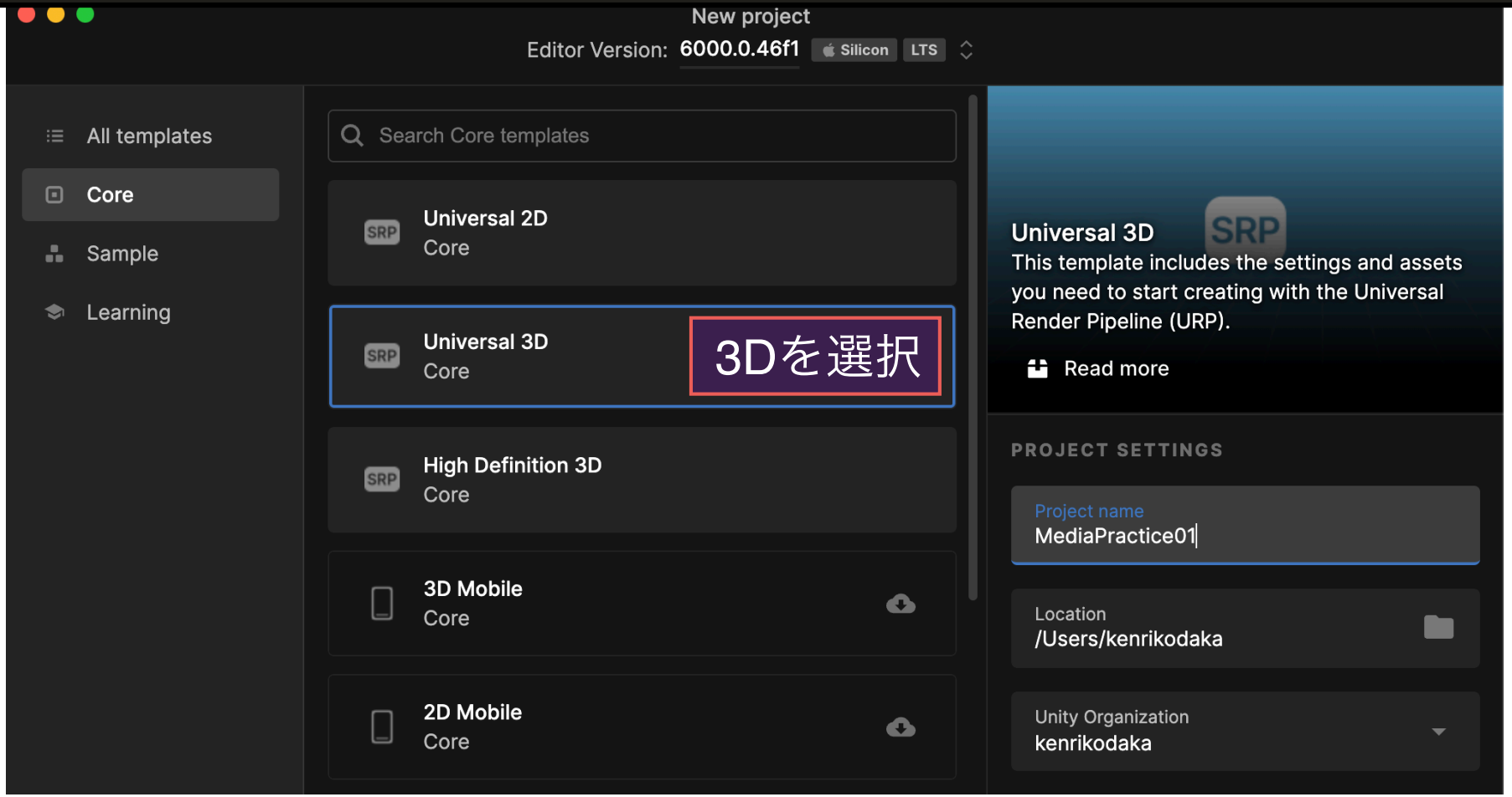
１D｜プレハブ（gameobjectの雛形）, タグ, その他

インストール（1/2）


Hub
Projects
Installs
Learn
Resources
Licenses
Settings
Unity
1
Installs
Search
Locate
Install Editor
All
Official releases
Pre-releases
別のバージョン
をインストール
UNITY 6
Unity 6.2 (6000.2.12f1) Silicon 5 projects
Man
2
現在インストール済み
Unityのバージョンを表示
Unity 6 (6000.0.46f1) Silicon LTS 15 projects
/Applications/Unity/Hub/Editor/6000.0.46f1/Unity.app
Manage
Web
macOS
OTHER VERSIONS
Unity 2021.3 LTS (2021.3.33f1) Silicon LTS 1 project
/Applications/Unity/Hub/Editor/2021.3.33f1/Unity.app
macOS
UNITY HUB

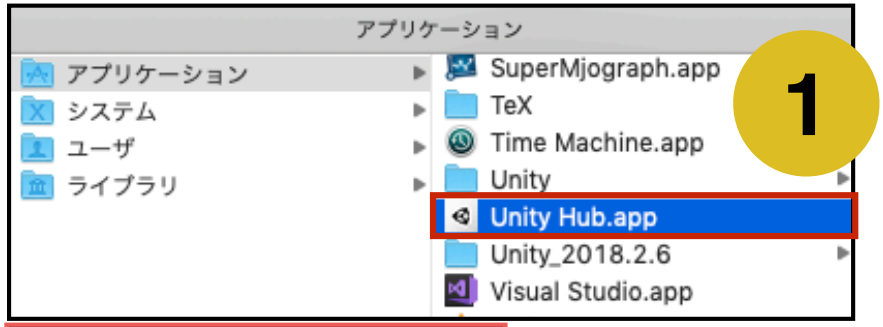
MediaPractice01

Unityとエディタの連携

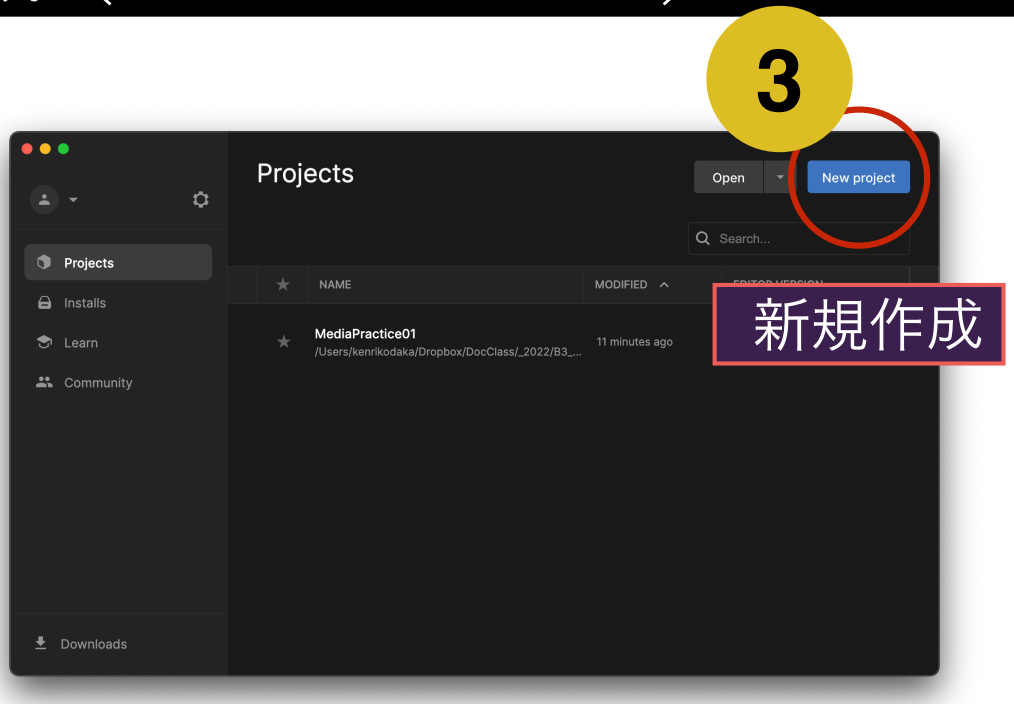


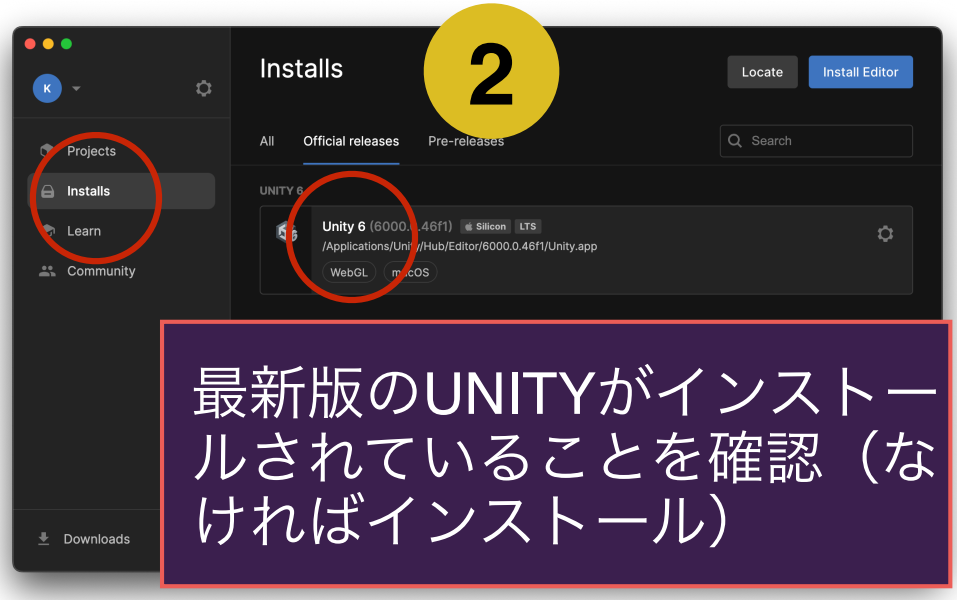
Unityアプリケーションを開いた後, 作業フォルダを適当な場所に作成し, そこをLocationとして, MediaPractice01という名前のProjectをつくりましょう.

新規プロジェクトの作成（MediaPractice01）



Unity Hubを起動

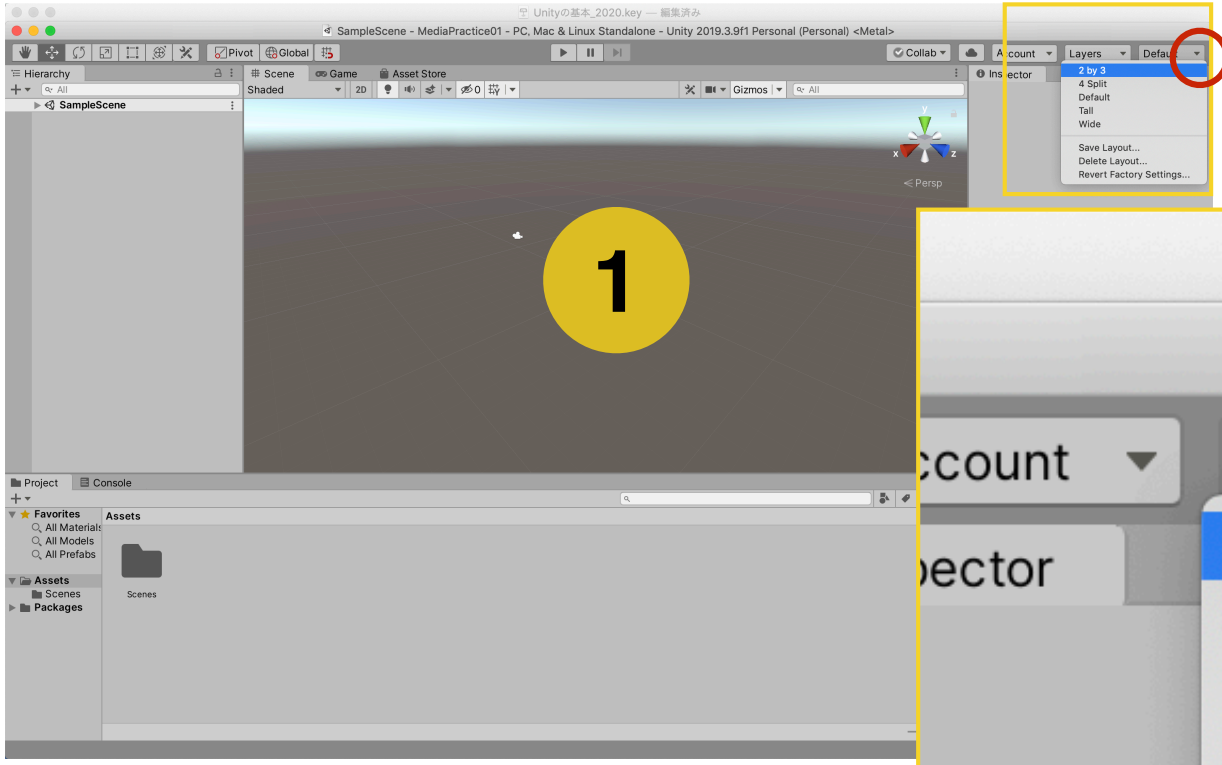




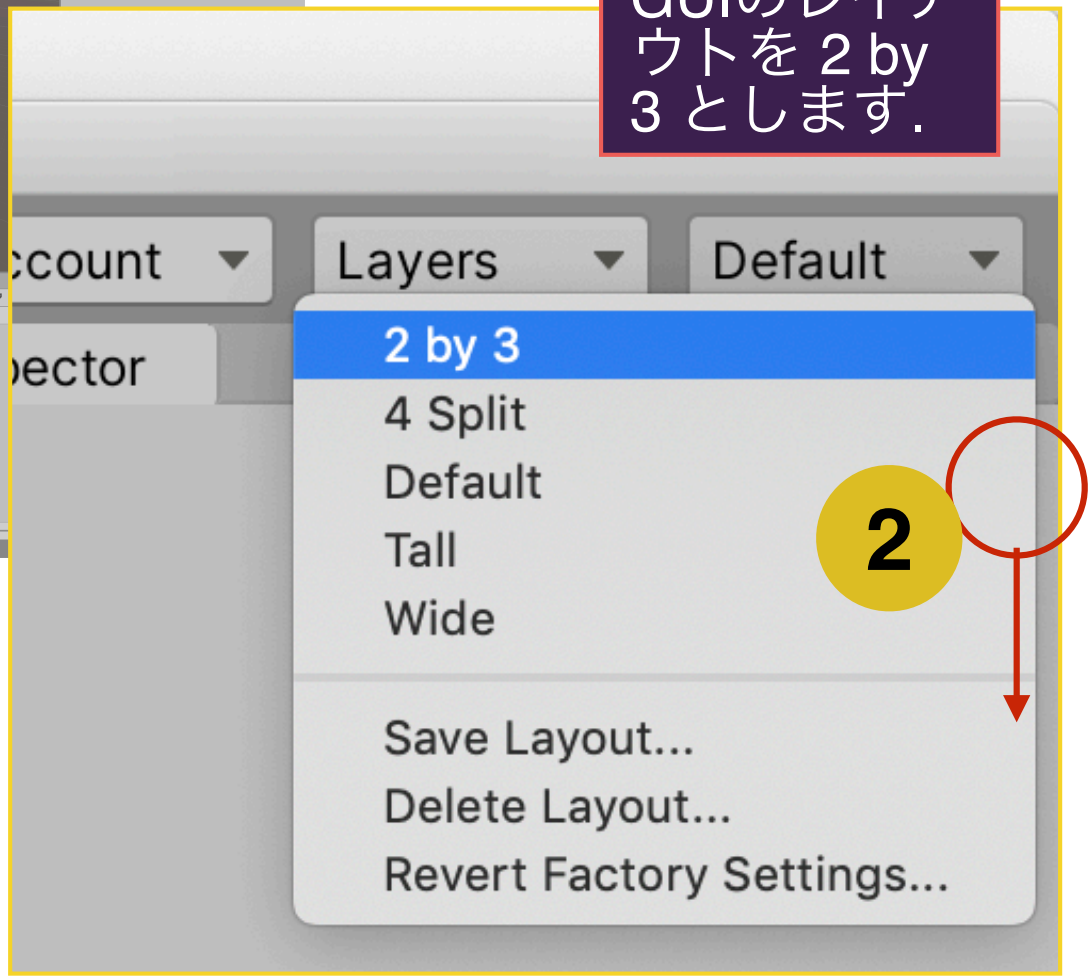
最新版のUNITYがインストールされていることを確認（なければインストール）

2026年4月時点の、最新の安定版は「Unity6.4（6000.0.42f1）」です（for APPLE SILICON）。

GUIレイアウトの変更



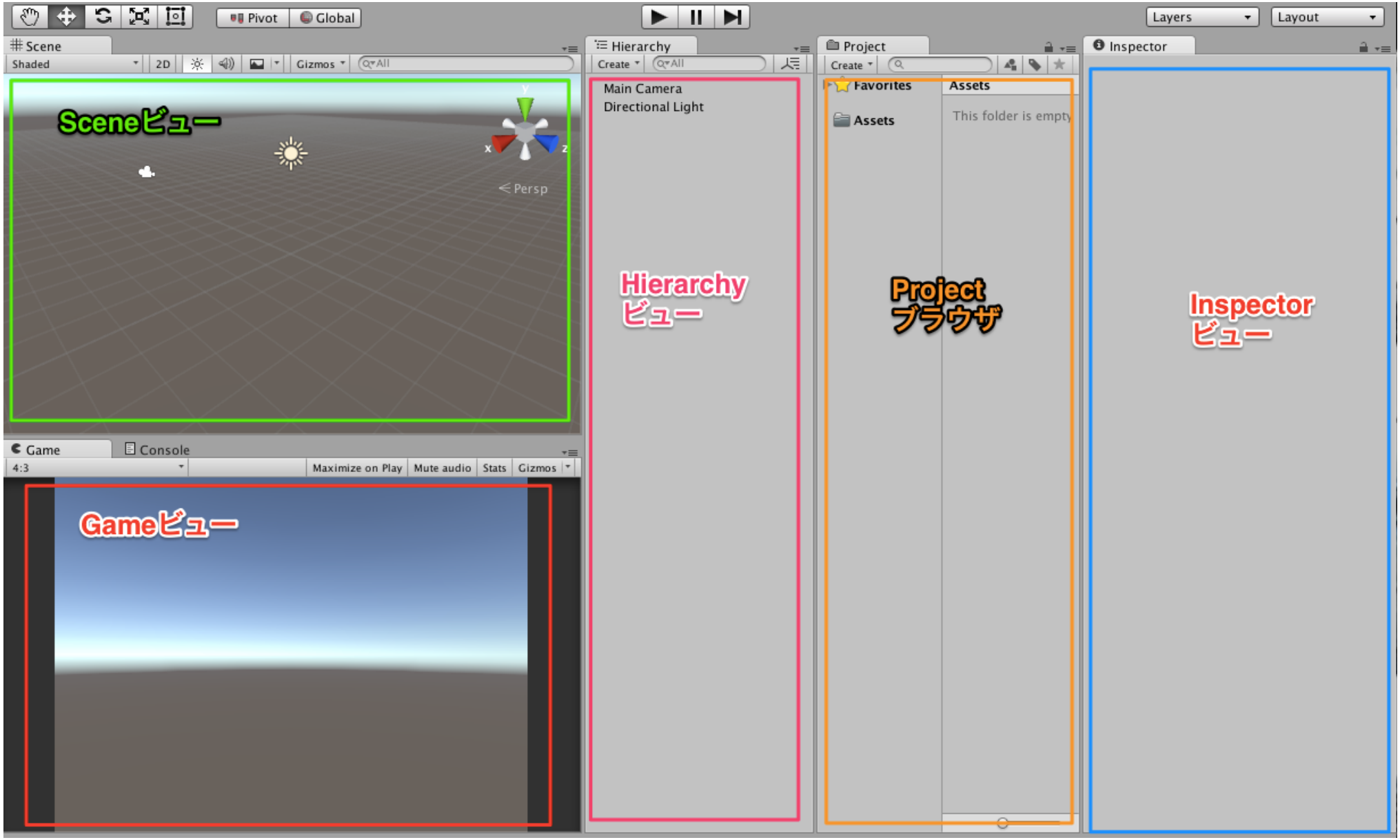
GUIのレイアウトを 2 by 3 とします.



最新版のUNITYでは、DefaultのGUIレイアウトが①となっているので、まずはレイアウトを「2 by 3」に変更してください（②）。

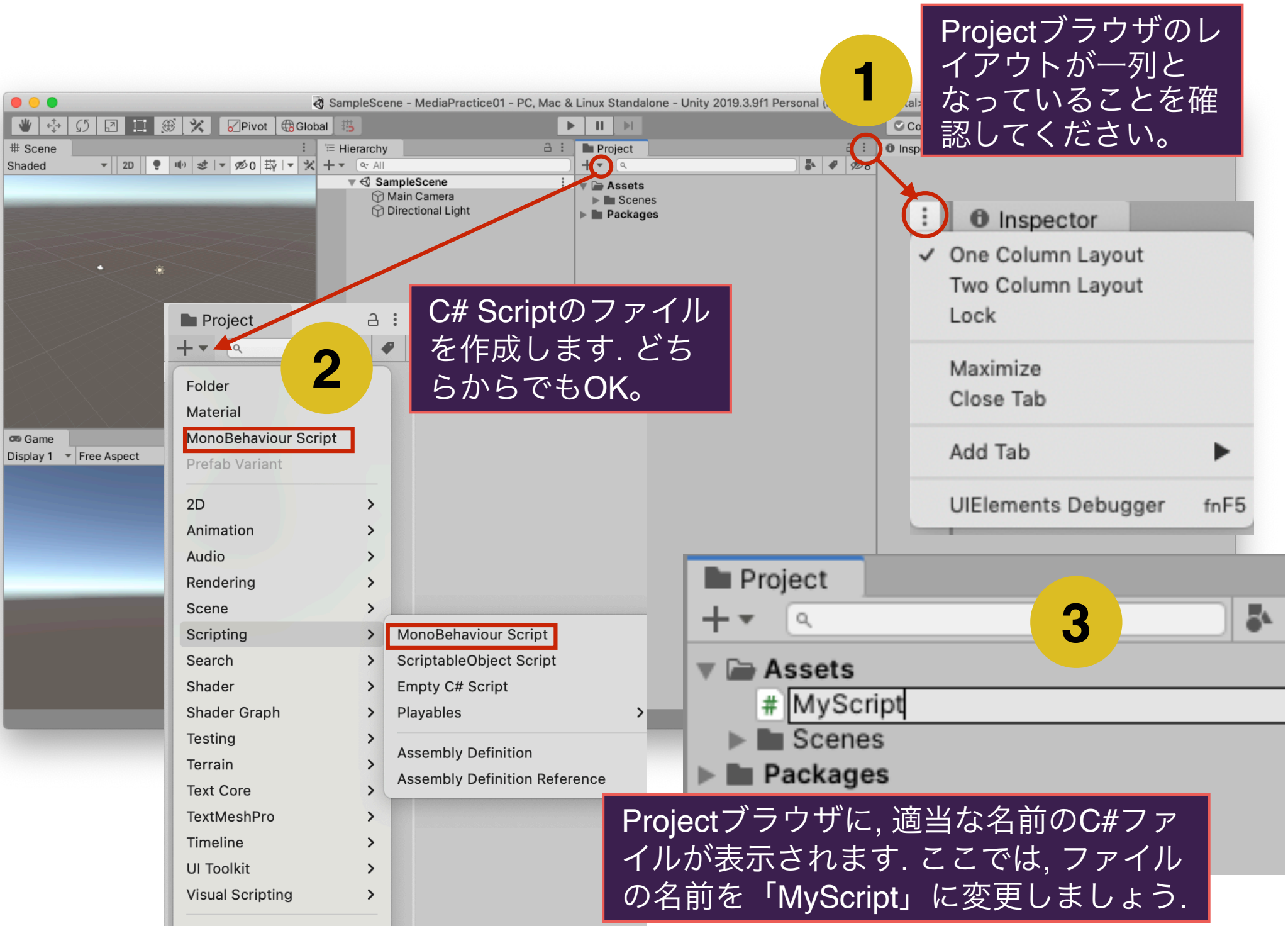
授業資料は、「2 by 3」のレイアウトを前提に進めていきます。

GUIの構造・各パネルの呼び名（2 by 3 レイアウト）

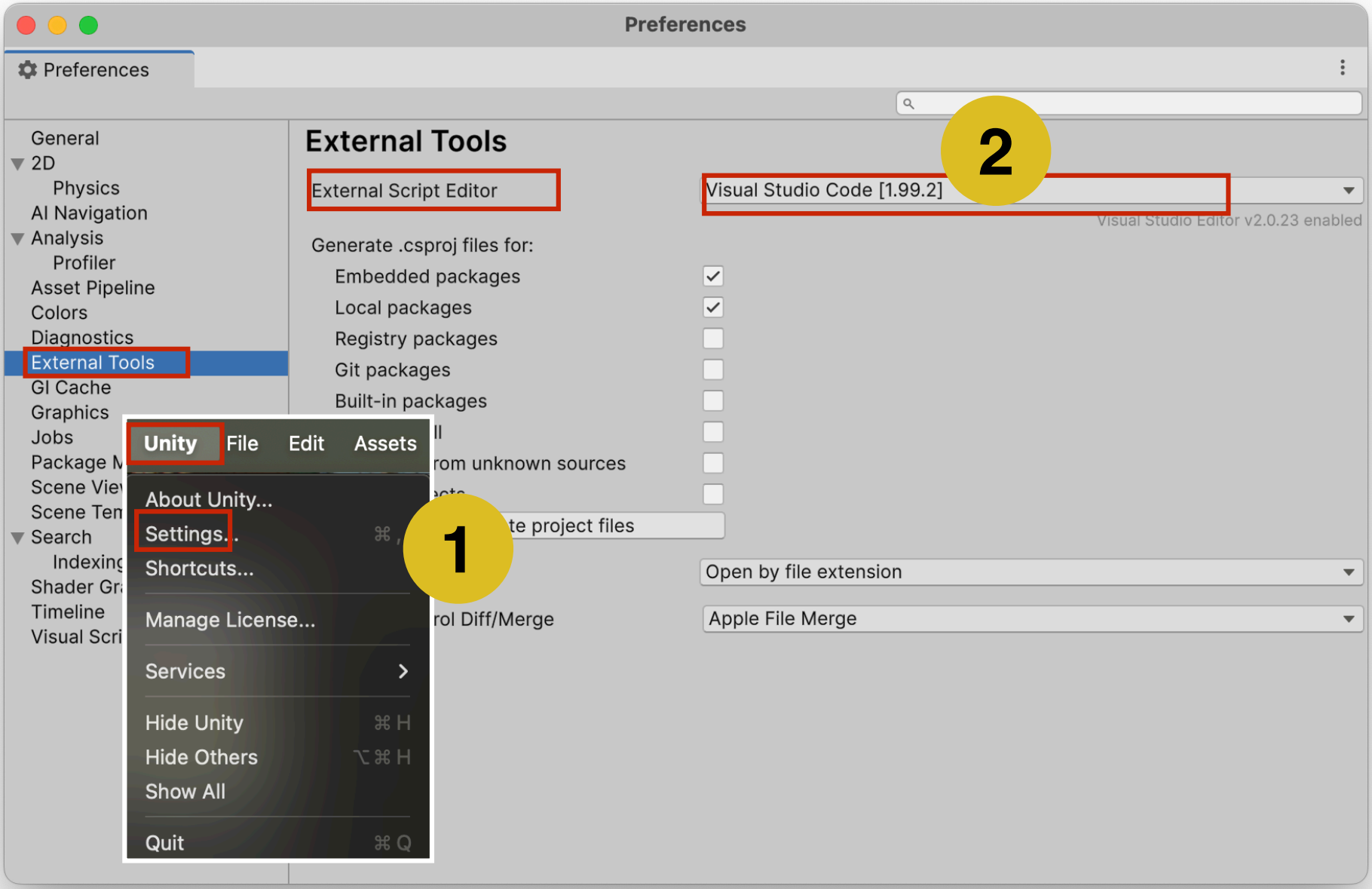


https://github.com/unity3d-jp/FirstTutorial/wiki/0.プロジェクトの作成

スクリプト作成の準備


1
Projectブラウザのレイアウトが一列となっていることを確認してください。
SampleScene - MediaPractice01 - PC, Mac & Linux Standalone - Unity 2019.3.9f1 Personal
Pivot
Global
Scene
Shaded
2D
Hierarchy
SampleScene
Main Camera
Directional Light
Project
Assets
Scenes
Packages
Inspector
One Column Layout
Two Column Layout
Lock
Maximize
Close Tab
Add Tab
UIElements Debugger
fnF5
Game
Display 1
Free Aspect
2
C# Scriptのファイルを作成します. どちらからでもOK。
Folder
Material
MonoBehaviour Script
Prefab Variant
2D
Animation
Audio
Rendering
Scene
Scripting
Search
Shader
Shader Graph
Testing
Terrain
Text Core
TextMeshPro
Timeline
UI Toolkit
Visual Scripting
MonoBehaviour Script
ScriptableObject Script
Empty C# Script
Playables
Assembly Definition
Assembly Definition Reference
3
MyScript
Projectブラウザに, 適当な名前のC#ファイルが表示されます. ここでは, ファイルの名前を「MyScript」に変更しましょう.

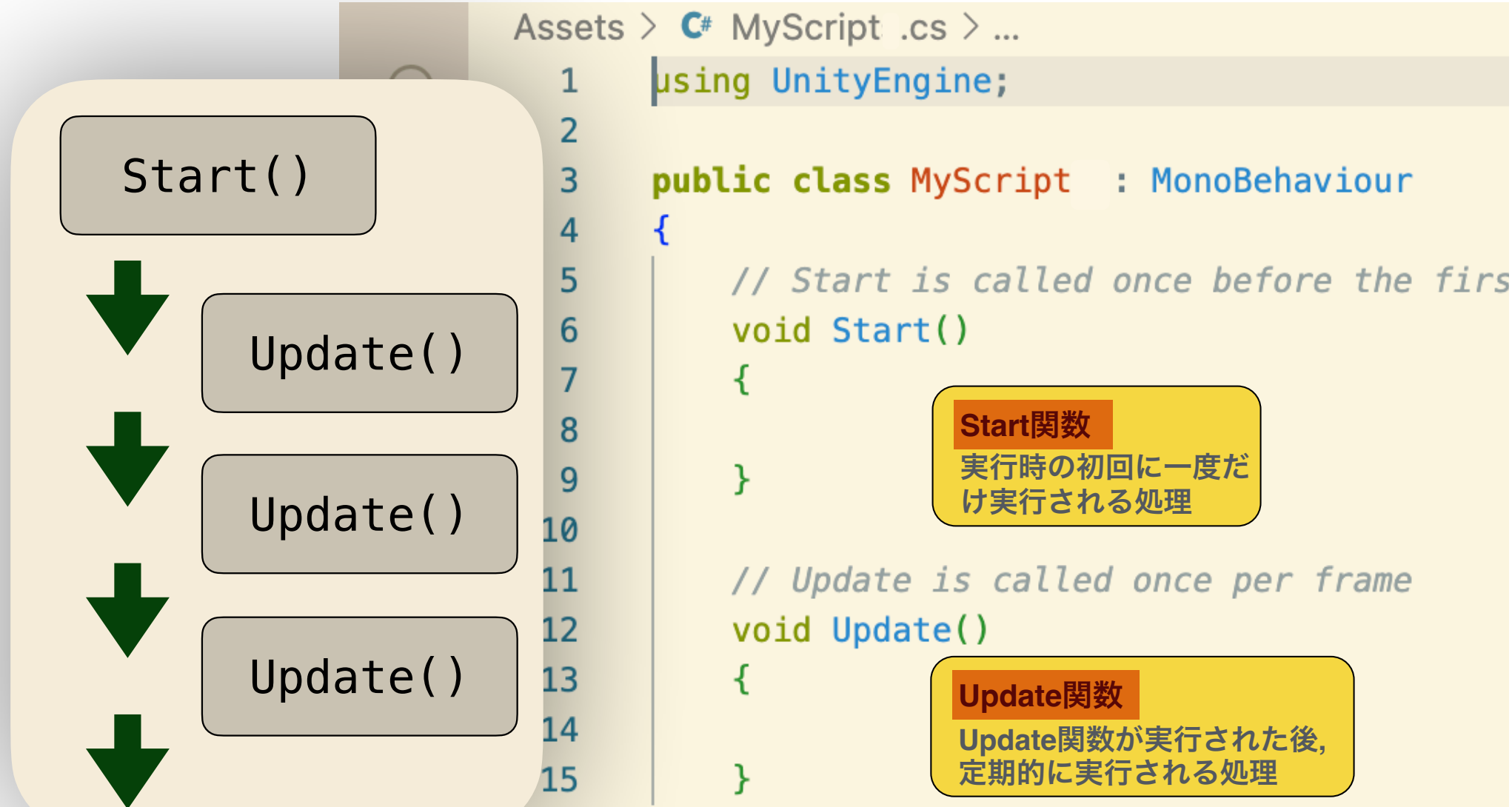
外部スクリプトエディタの設定



はじめに、外部スクリプトのエディタとして「Visual Studio Code」を設定しておきます。これをしないと、プログラム環境として別のテキストエディタが開きます。

エディタを開く

- Project ブラウザの ***MyScript.cs*** をダブルクリックすると, 別アプリケーションのエディタが開き, csファイルの編集が可能となります.
- スクリプトに対応するクラスには, あらかじめ **Start** 関数と **Update** 関数がインクルードされています.



ゲームオブジェクトとスクリプトの関連付け

MyScript.cs

Assets > MyScripts.cs > ...

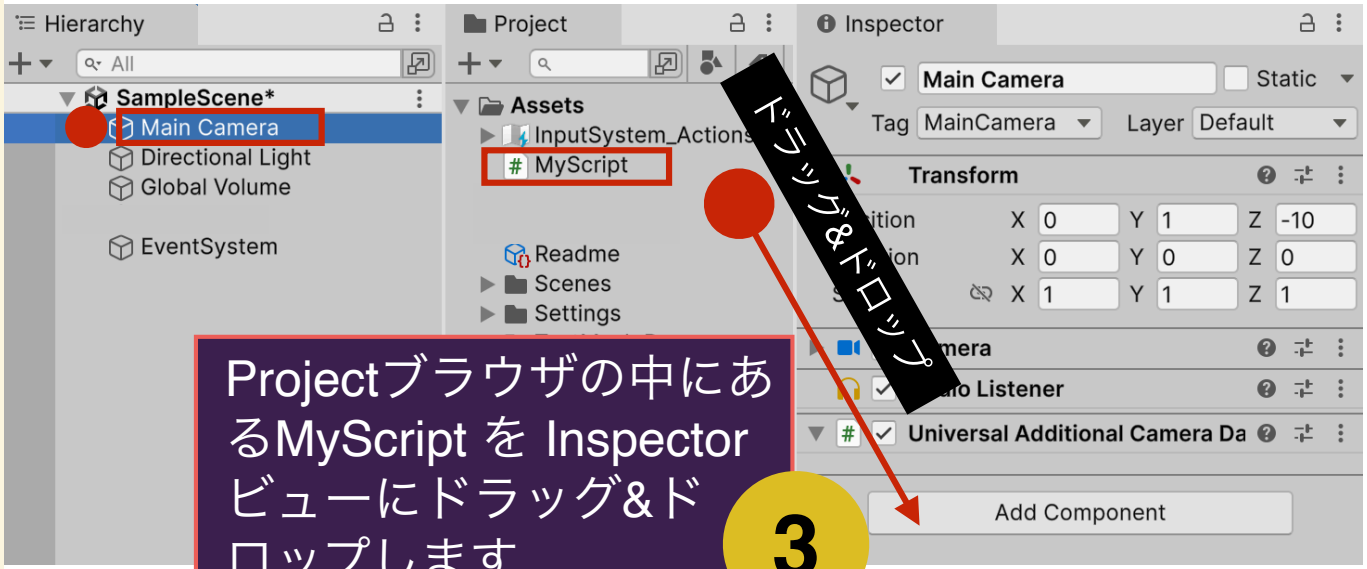
```
1   using UnityEngine;
2   using TMPro;
3
4   public class MyScripts : MonoBehaviour
5   {
6       public TextMeshProUGUI t;
7
8       void Start()
9       {
10
11      }
12
13      void Update()
14      {
15
16      }
17  }
```

1

新たに二つの文を挿入してください.

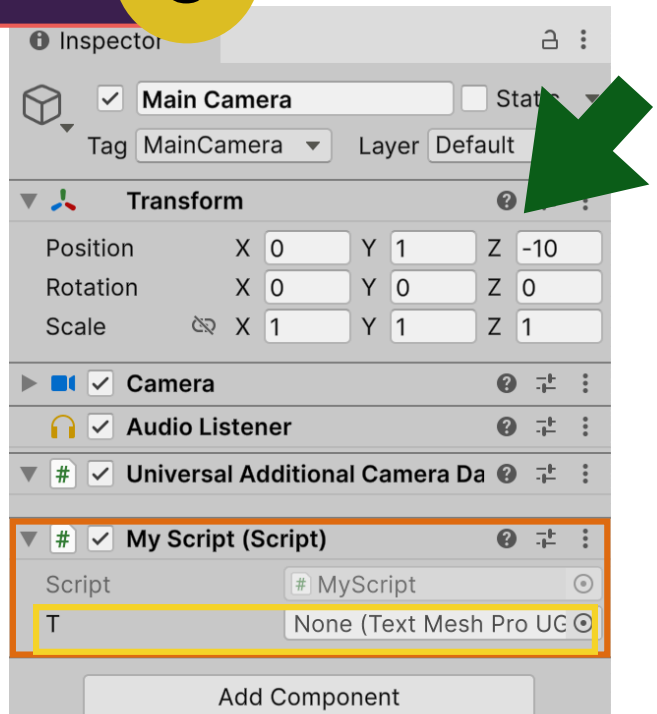
2

Hierarchy ビューの Main Camera を 選択します（Inspector ビューに Main Camera のコンポーネントが表示されます）。



Projectブラウザの中にあるMyScript を Inspector ビューにドラッグ&ドロップします.

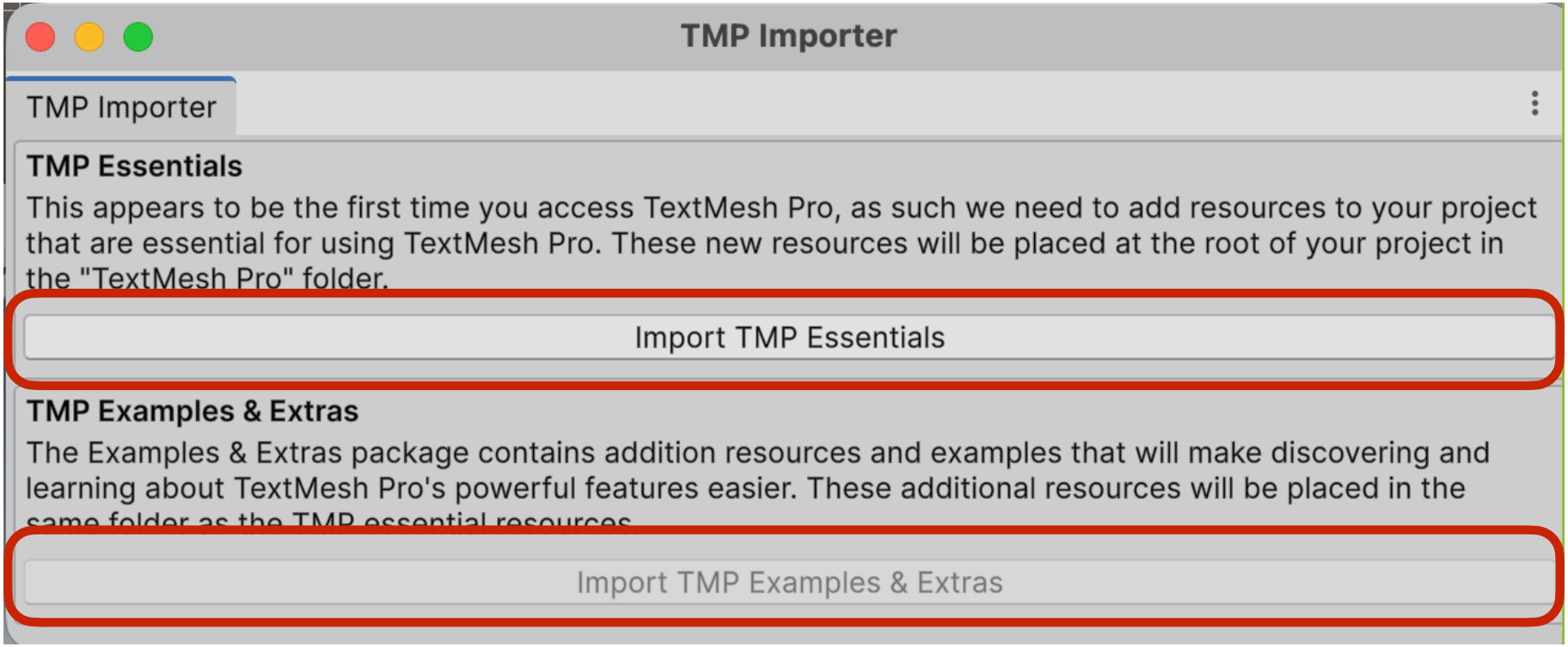
3



inspectorビュー内の「T」は, スクリプト内の変数「t」に対応しています.

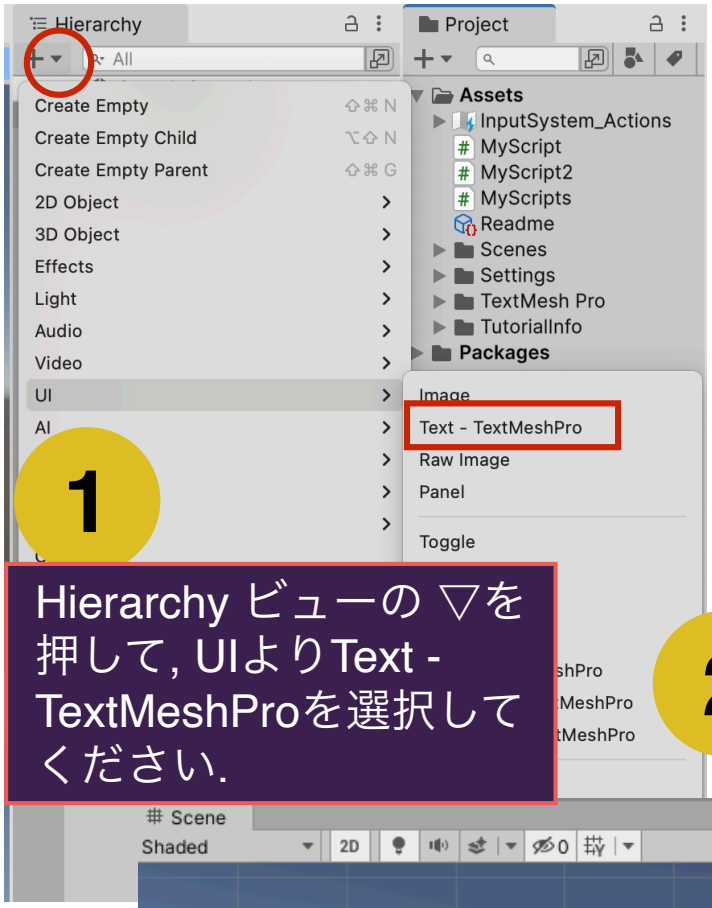
4

結果, Inspectorビューにスクリプトの内容が表示されます（Main Cameraのコンポーネントとして登録されたということです）.



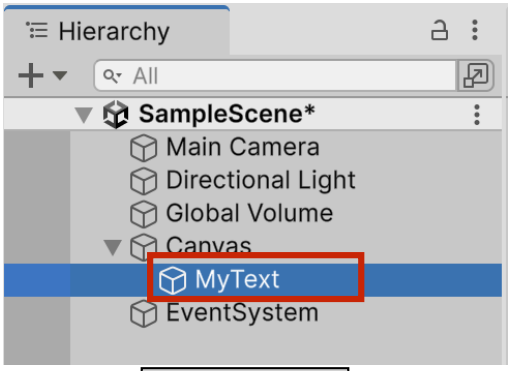
スクリプト（MyScript.cs）の中で、TMProをImportすることで、この画面が現れるはずです。TMPの機能を使うために、どちらもImportしてください。

Text（ゲームオブジェクト）の追加



1

Hierarchy ビューの ▽を押して, UIよりText - TextMeshProを選択してください.

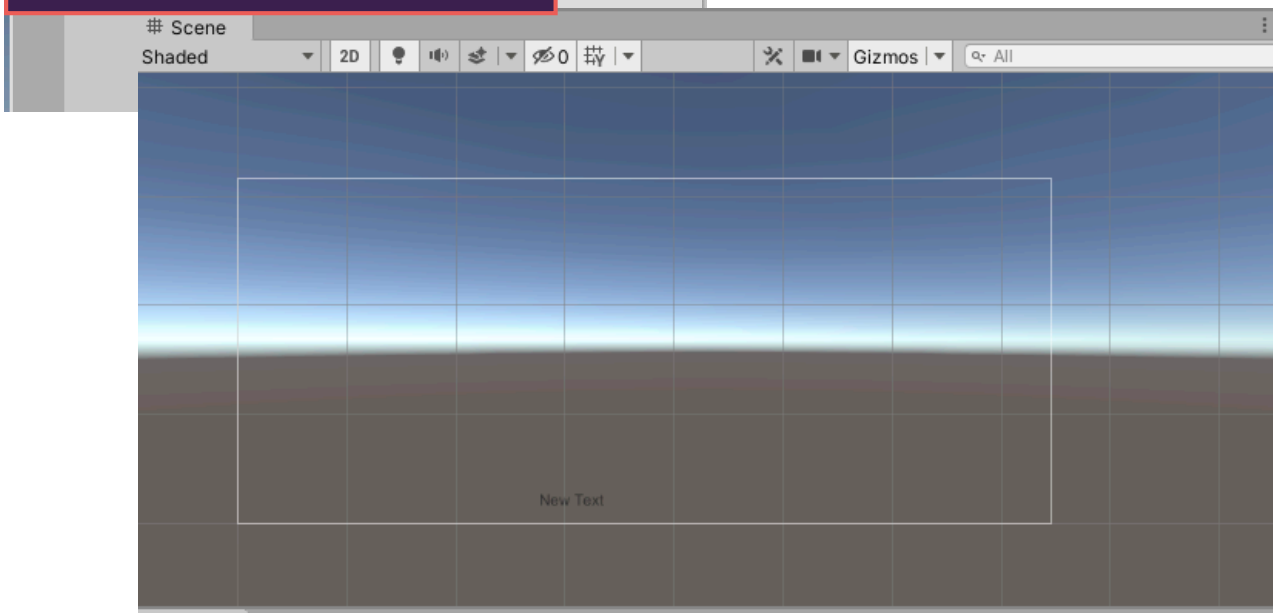


2

ここでは, Text を MyTextにリネームしましょう.

3

二次元表示に切り替えます.



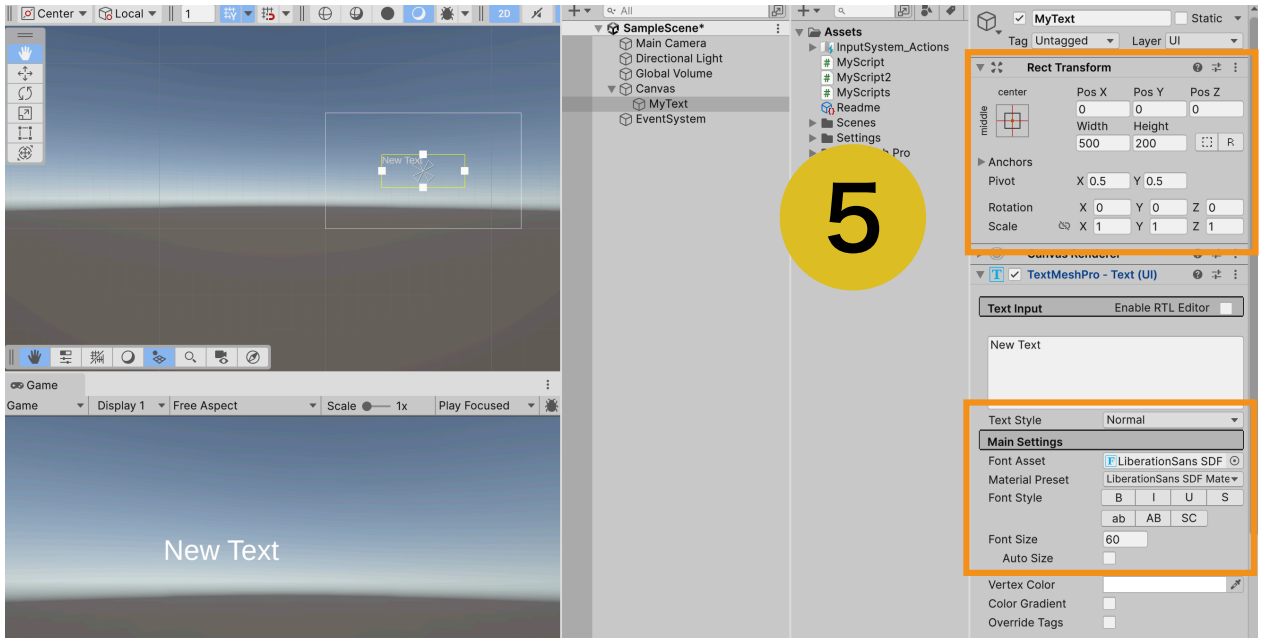
4

白の長方形の枠がすべてSceneビューにおさまるように拡大・縮小（マウスホイールの回転）、移動（option＋ドラッグ）をして調整します.

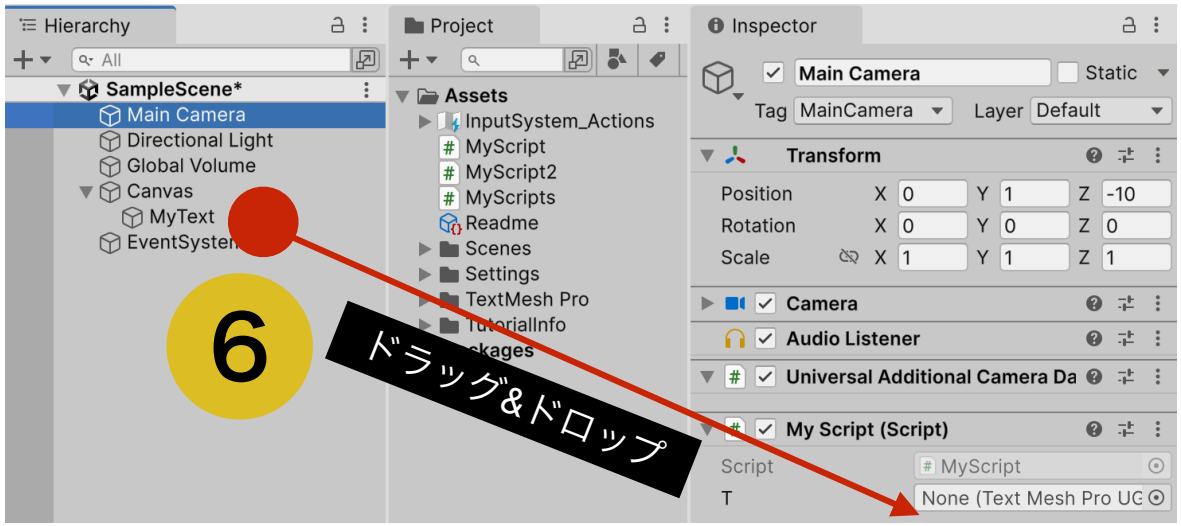
Scene ビューの操作方法については例えば以下を参考にしてください。

https://miyagame.net/scene-view-method/

MyTextのインスペクタを以下のように変更してください. 白の枠内に「New Text」という文字が表示されるはずです.

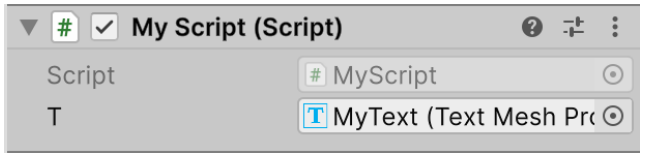


Main Cameraを選択した状態で, My Scriptのpublic変数「T」の項目に, HierarchyビューのMyTextをドラッグ&ドロップします.



7

結果, 「T」の項目がNoneからMyTextへと変更されます.

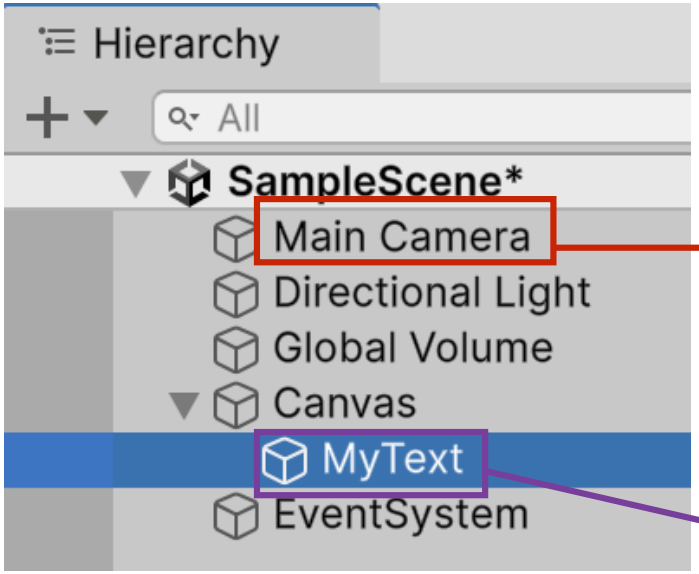


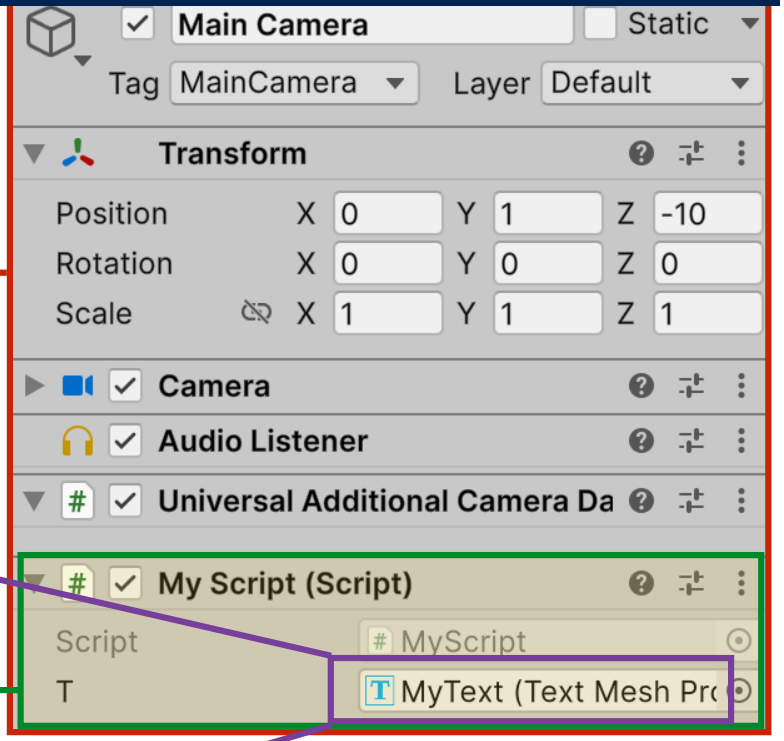
ゲームオブジェクトとスクリプト内変数の関係

シーンの実行

Hierarchyビューの中にある全てのゲームオブジェクトが起動

起動したゲームオブジェクトのコンポーネントに登録されている全てのスクリプトが起動






```
Assets > C# MyScripts.cs > ...
1   using UnityEngine;
2   using TMPro;
3
4   public class MyScripts : MonoBehaviour
5   {
6       public TextMeshProUGUI t;
7
8       void Start()
9       {
10
11      }
12
13      void Update()
```

MyScript.cs

cs内の変数 t は, MyTextで表示される文字列と結びついている.

そのため, 変数 t に対する操作が, シーン内のMyTextの変化として反映される.

C#における変数の型

Assets > MyScript.cs > ...

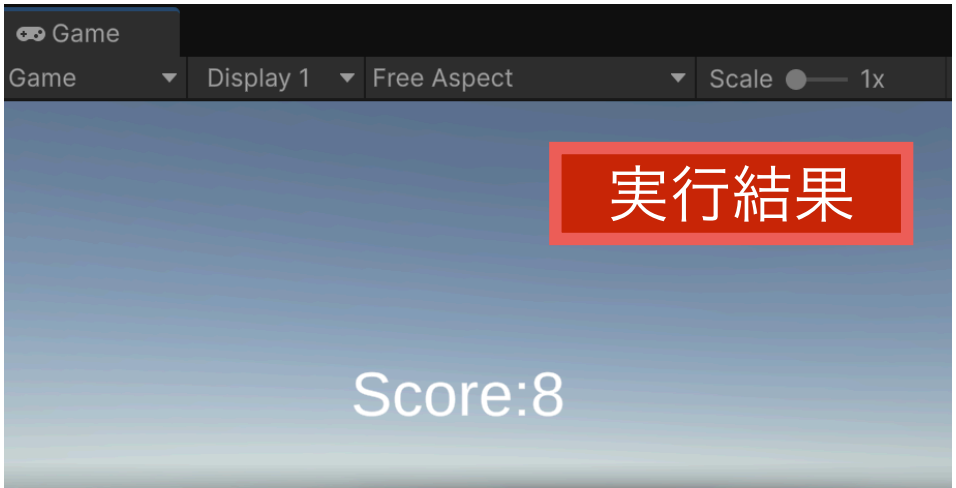
MyScript.cs

```
1   using UnityEngine;
2   using TMPro;
3
4   public class MyScript : MonoBehaviour
5   {
6       public TextMeshProUGUI t;    //Text t; //旧インタフェース用
7
8       int x1 = 3;
9       int x2 = 5; //publicにするとインスペクタで編集可能
10      float y1 = 10.3f; //fはつけなくてもエラーにはならないが推奨。
11      double y2 = 10.3; //doubleは基本的に使わない。
12      string s1 = "abc"; //文字列はstring型
13      bool b1 = true; //真偽値はbool
14
15
16      void Start()
17      {
18          int z1 = x1 + x2;
19          //int z2 = x1 + y1; //エラー:(int)+(float)
20          float z2 = x1 + y1; //(int)+(float)はfloatに揃えられる。
21          int z3 = x1 + (int)y1; //キャストすればintでもOK。
22
23          int score = z1;
24          t.text = "Score:" + score.ToString();
25
26          /* SetTextをつかう別のやり方 */
27          //第一引数の文字を出力、{0}は第二引数の変数の値が埋め込まれる。
28          // t.SetText("Score:{0}", score);
29          //複数の変数を文字列に埋め込むには以下のようにする。
30          //t.SetText("(z1,z2,z3)=({0},{1},{2})", z1,z2,z3);
31      }
```

int	100　-3　98765
double	100.0　-3.0　98765.2
float	100.0f　-3.0f　98765.2f
char	‘a’　‘Z’　‘あ’
string	“abc”　“あいうえお”　“a”
bool	true　false

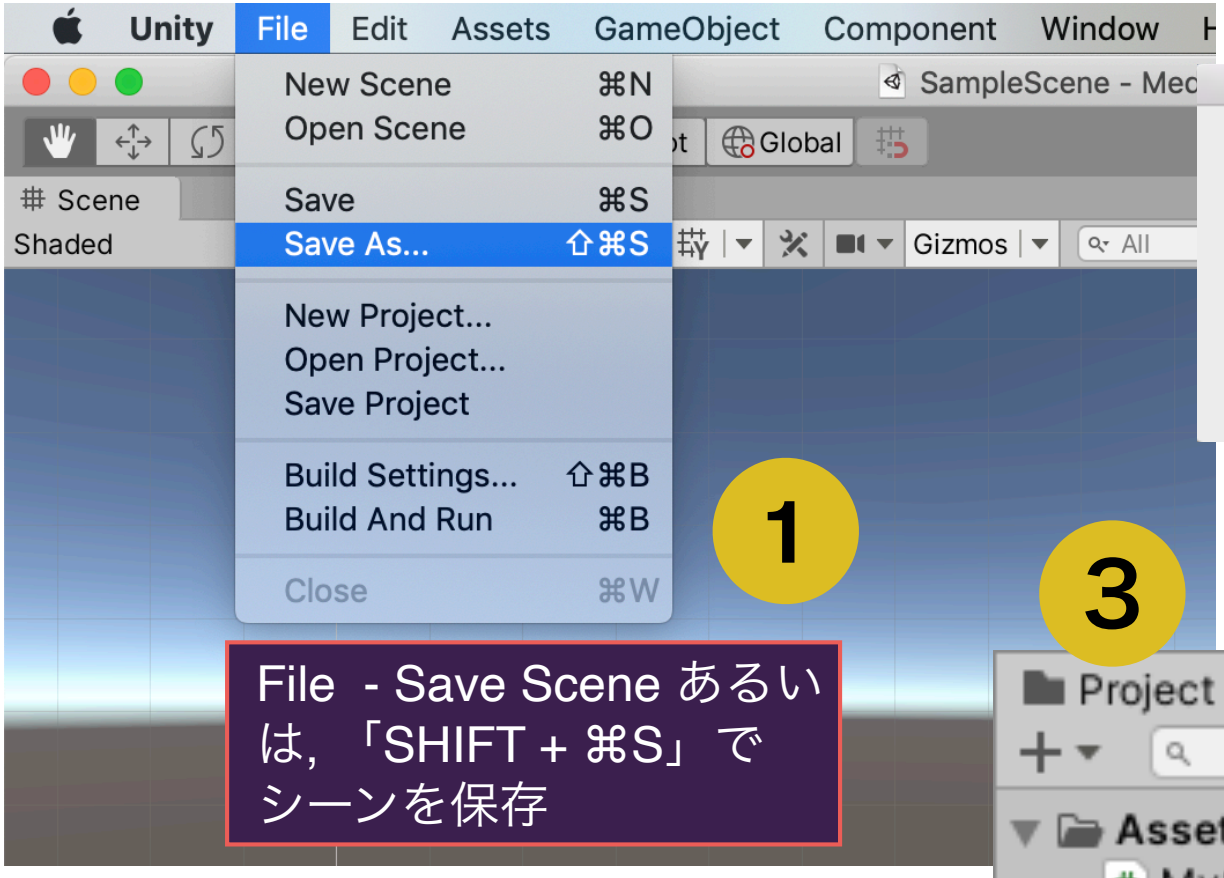
整数と実数には様々な型がありますが, Unityの計算では, 特別な理由がない限り, 「整数は **int** , 実数は **float** を使う」と覚えておきましょう.

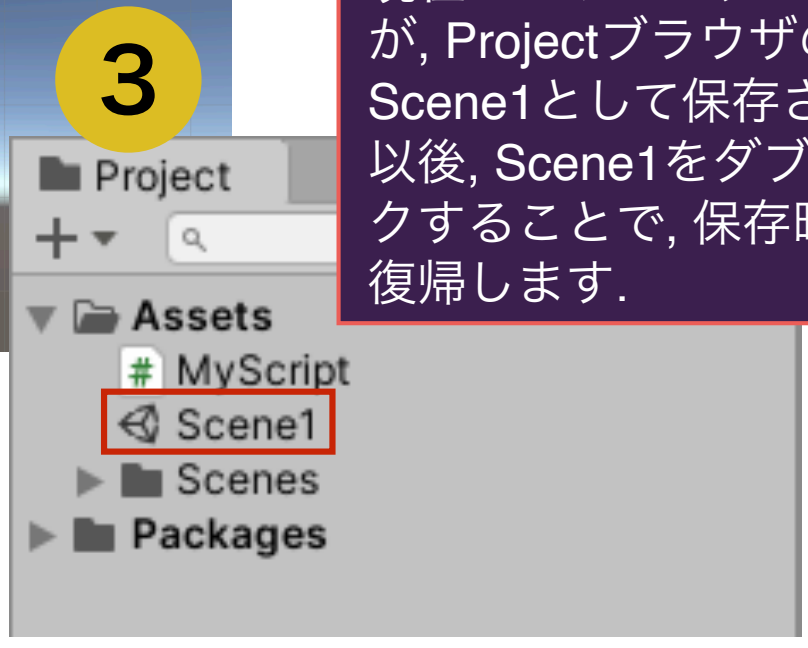
float型には数字の後に「f」をつけることが推奨されています.



シーンの保存

- シーンの現在の状態（Hierarchyビューにどのゲームオブジェクトが配置され，それらがそのようなパラメータを有しているか）を保存することで，特定の状態へとすぐに復帰することができます．





現在のパラメータの状態などが，Projectブラウザの中にScene1として保存されます．以後，Scene1をダブルクリックすることで，保存時の状態が復帰します．

最新版では、初期状態のシーンには「SampleScene」という名前が付けられています。

配列の利用（c#）

```
4   public class MyScript2 : MonoBehaviour
5   {
6       public TextMeshProUGUI t;
7       public int id=0;
8
9       int[] x = {1,3,5};
10      string[] s = {"Hello","Welcome","Bye","Afternoon"};
11
12      void Start(){
13      }
14      void Update()
15      {
16          if(id==0){
17              int ri = (int)Mathf.Floor(x.Length * Random.value);
18              t.text = x[ri].ToString(); //t.SetText("{0}",x[ri]); //これ
19          }else{
20              int ri = (int)Mathf.Floor(s.Length * Random.value);
21              t.text = s[ri];
22          }
23      }
24  }
```

表示の切り替えフラグ

配列の宣言と値の代入

MyScript2.cs

int 配列.Length

配列のサイズ

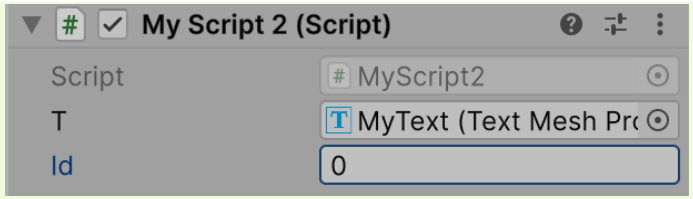
float Mathf.Floor(実数)

引数の値の小数点を切り捨て

float Random.value

0～1のランダムな実数

実行結果



inspectorビューにおいて，idの値を直接変更すると表示する配列が切り替わります．ただ，このままだと，要素の表示の切り替わりが速すぎますね．

ストップウォッチ（時間処理）の実装

```
4   public class MyScripts3 : MonoBehaviour
5   {
6       public TextMeshProUGUI t;
7       public int id=0;
8
9       int[] x = {1,3,5};
10      string[] s = {"Hello","Welcome","Bye","Afternoon"};
11
12      //drawRandom関数処理の時間間隔(秒)
13      public float timeOut = 1f;
14      //前回のdrawRandom関数処理からの累積経過時間(秒)
15      private float timeElapsed = 0f;
16
17      void Start()
18      {
19      }
20      void Update()
21      {
22          //Time.deltaTime:前回のフレームからの経過時間(秒)
23          timeElapsed += Time.deltaTime;
24
25          if(timeElapsed > timeOut){
26              drawRandom();
27              timeElapsed = 0f;
28          }
29      }
30      void drawRandom(){
31          if(id==0){
32              int ri = (int)Mathf.Floor(x.Length * Random.value);
33              t.text = x[ri].ToString(); //t.SetText("{0}",x[ri]);
34          }else{
35              int ri = (int)Mathf.Floor(s.Length * Random.value);
36              t.text = s[ri];
37          }
38      }
```

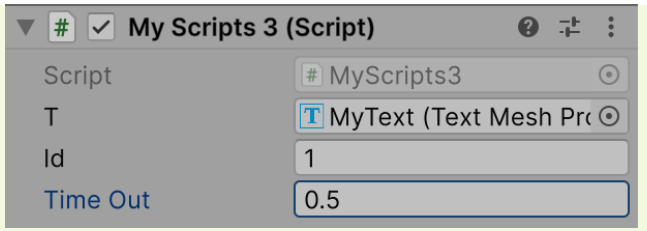
MyScript3.cs

timeElapsed は, ストップウォッチの値を保持する変数であり, 指定時間を過ぎたらdrawRandom関数を実行するようにしています.

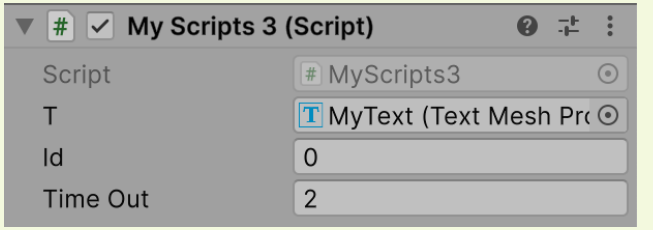
float Time.deltaTime

Update関数実行間のインターバル（秒）

実行結果

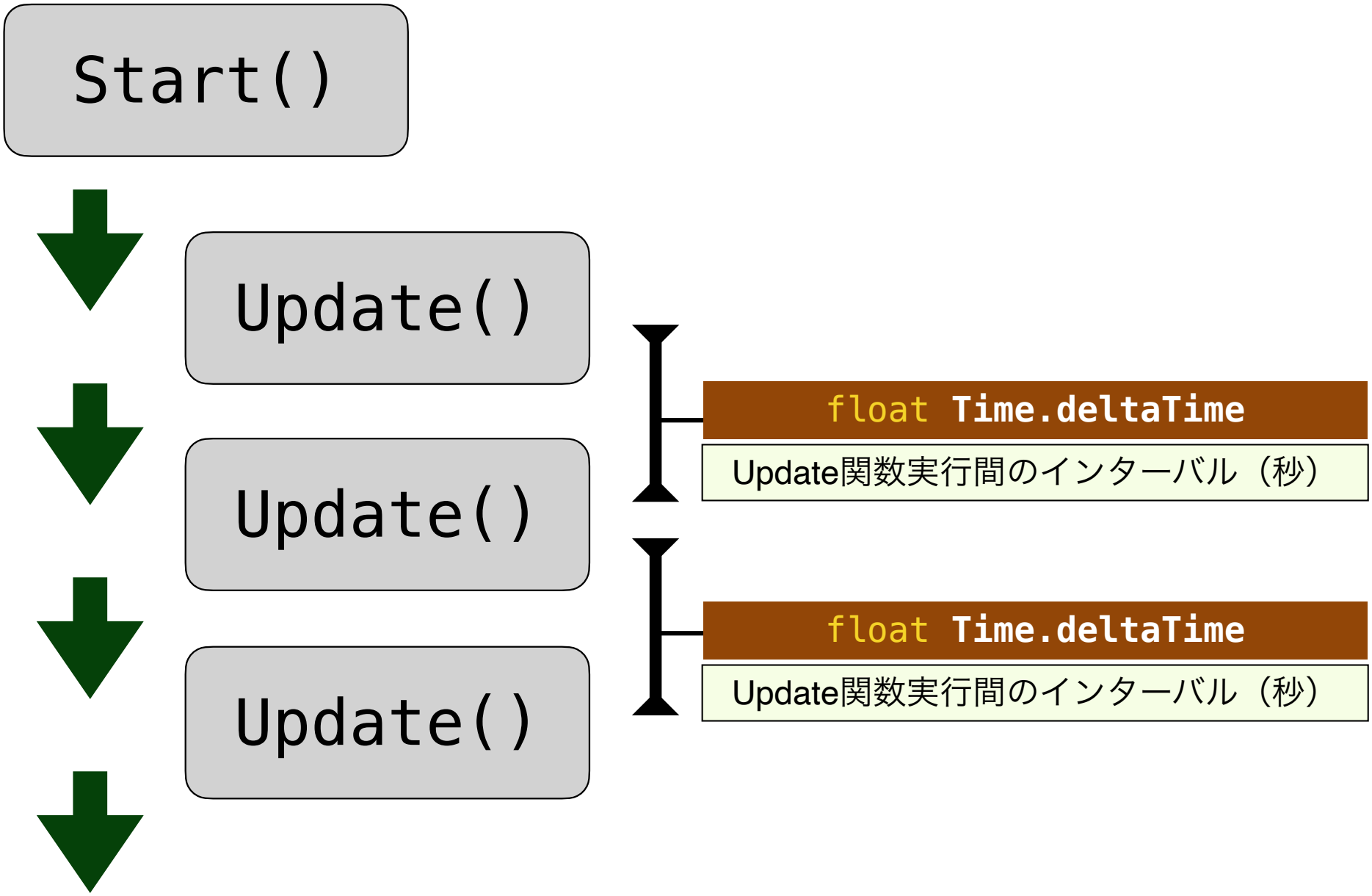


0.5秒おきに挨拶の表示が切り替わります.



2秒おきに数字の表示が切り替わります.

ストップウォッチ（時間処理）の実装



設定

VS Codeでインライン参照を消す

あれですね、**Visual Studio Code** で変数の上に表示される：

> 🔎 `1 reference`
> 🔎 `3 references`

みたいなやつ、地味に気になりますよね。
これは **"インライン参照（inline reference counts）"** という機能です。

✅ 消す方法：設定から無効化する

1. VS Code を開く
2. メニューから
   `Code`（Macの場合）または `File`（Windows） → `Preferences` → `Settings`
3. 検索バーに次を入力：

   コピーする　編集する

   ```
   editor.codeLens
   ```

4. 「**Editor › Code Lens: Enabled**」という設定が出てくるので、
   ✅ チェックを **外す**

これで `1 reference` といった表示が消えます。

Unityのテーマ変更

✅ Unityのテーマ変更方法（Pro/Plus限定）

1. **Unityエディタを開く**
2. 上部メニューから
   Edit → Preferences（Macなら `Unity → Preferences`）
3. 左のリストから **General** を選ぶ
4. 右側にある **Editor Theme** を `Personal`（ライト）か `Professional`（ダーク）から選択

⚠ 注意点

- **ダークテーマ（Professional）は Unity Plus / Pro ライセンス限定機能**
- 無料版（Personal）ではライトテーマ固定です（現在の仕様）