

ストップウォッチ (StopWatch)

時間を計測しないゲームも多くありますが、それでも2秒間の長押しを検出したり、文字の点滅を管理したり、競争の値としていなくてもゲーム内で時間を計測する場面は非常に多く出てきます。時間がハマれなければゲームは作れないと言えるかも知れません。

ここではストップウォッチの制作を通じて、Unity で時間を計測する方法を紹介します。また、計時の開始や停止のボタン押下のタイミングを1秒間に100回程度の回数で監視されることになります。

この2つ、①時間の計測と、②ユーザーの監視、両方を行うのが Update 処理となり、今回は Update 処理の理解を深めることになります。

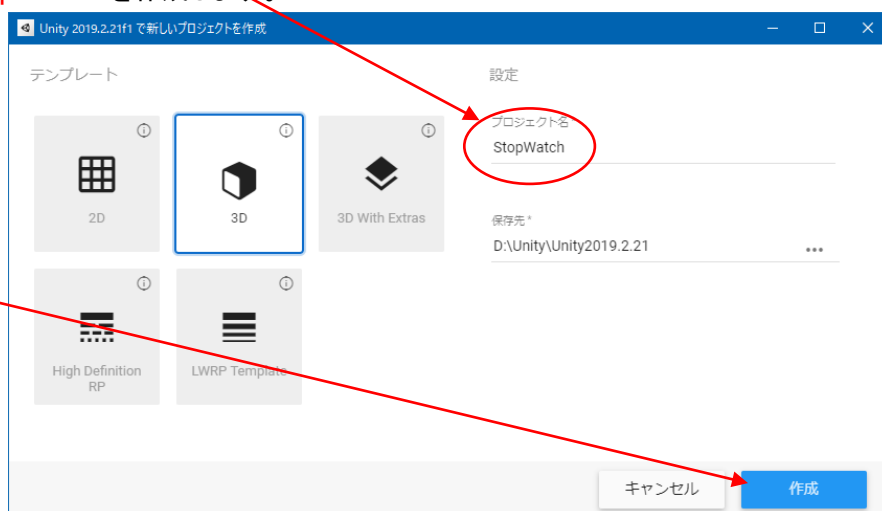
4.13s

制作環境の整備

【STEP1】プロジェクトの準備

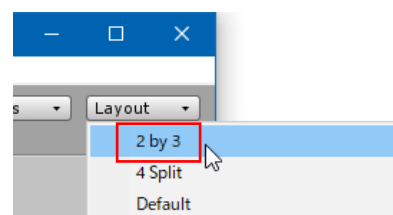
- Unity を起動し、**プロジェクト StopWatch** を作成します。

ボタン作成を押下します。

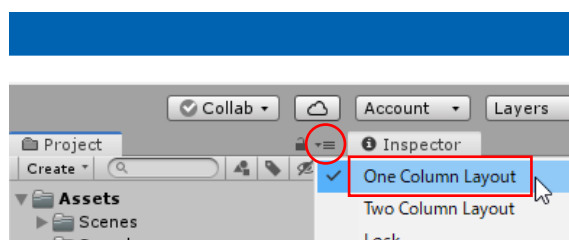


Unity が起動したら、画面のパネルレイアウトを確認します。

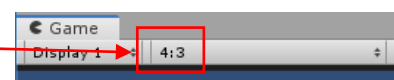
- 画面右上の **Layout** から **2 by 3** を選びます。



- プロジェクト欄のオプションから **One Column Layout** を選びます。



- Gameパネルのサイズを **4:3** にします。



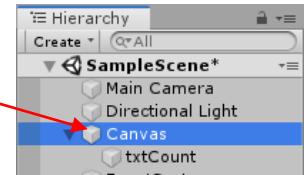
- ヒエラルキー欄の **Main Camera** を選択します。
インスペクタでパラメータを修正します。



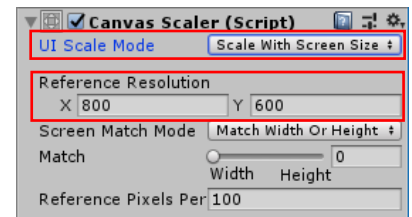
ユーザーインターフェースの構築

【STEP3】 文字の準備

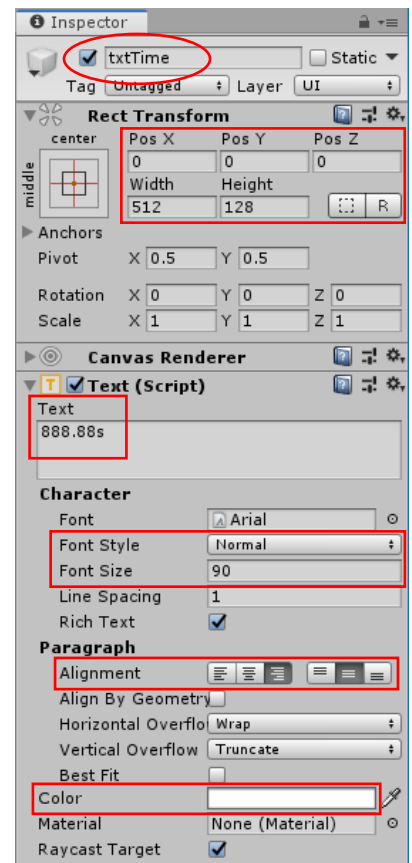
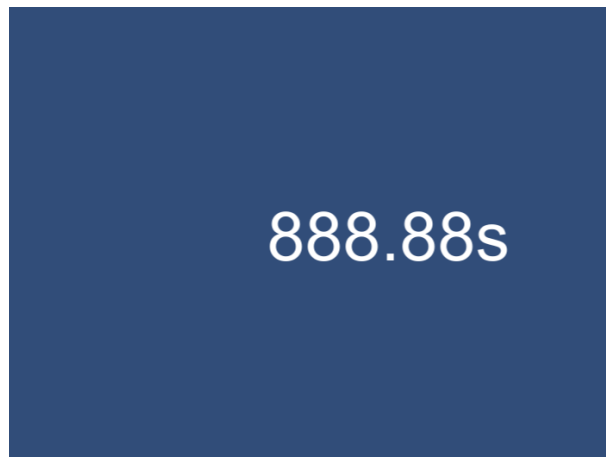
- ヒエラルキー欄の Create から UI > Text を選択します。名称を **txtTime** と命名します。
- 先に、同時にヒエラルキー欄に作られた **Canvas** を選択します。



インスペクタでパラメータを設定します。



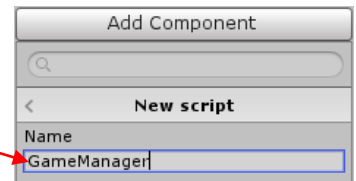
- ヒエラルキー欄の **txtCount** を選択し、インスペクタでパラメータを設定します。



時間を計測する

【STEP4】プログラムの準備

- ヒエラルキー欄の **Main Camera** を選択し、インスペクタの Add Component から **New script** を選択し、名称を **GameManager** とします。



- スクリプト **GameManager** を次のように編集します。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI; //uGUIを利用する際に必要

public class GameManager : MonoBehaviour {

    public Text txtTime; //画面のGUI文字

    void Start() {

    }

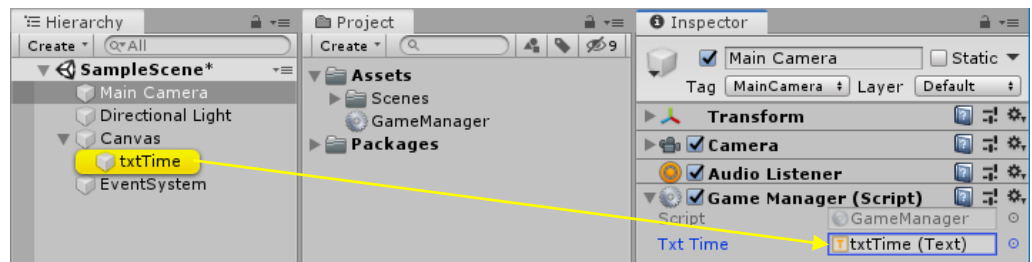
    void Update() {
        txtTime.text = Time.time.ToString();
    }


}
```

【学習項目:Time.time】

プレイボタンを押下してからの経過時間が入っています。小数型なので文字列型に変換しています。

- ヒエラルキー欄の **Main Camera** を選択し、インスペクタに登場した項目にヒエラルキー欄の **txtTime** をドラッグ&ドロップします。



- プレイボタンを押下します。  プレイボタンを押下してから何秒が経過したか？の情報をもらって来て、画面に刻々と転記している状況となります。



【学習項目:Update 処理】

Update 関数や Update メソッドと呼ぶこともあり、極端に言うと、1秒間に約100回程度実行される処理です。間違った使い方をすると、例えば、1秒間に同じ音楽が 100 曲も再生されたり、白い文字の色を白にする無駄な命令が100回も実行されたりするので、注意して扱う場所です。

基本的な考え方は2系統の処理のみを書く場所と考えることです。1つは時間を計測する場所。もう1つは、プレイヤーの操作を監視する場所としての考えを持つことが重要です。

【STEP5】経過時間を測りやすくする

ここで用いた `Time.time` は、プレイボタンを押下してからの単純な時間しか保持していませんでした。ゲームどころか、ストップウォッチですら何度も何度もタイムを測ることができます。プレイボタンを押下してからの時間しか保持していない場合、2人目以降の時間計測が複雑になってしまいます。(やれば出来ませんが、ややこしいです)


そこで、次のプレイヤーの時には、ゼロクリアできる項目を用いて計測を行うように修正し、ゲーム制作に於いて時間を測る一般的な記述に変更します。

- スクリプト `GameManager` を次のように編集します。

```
//～前略～  
  
void Update() {  
    txtTime.text = Time.deltaTime.ToString();  
}  
}
```

【学習項目:Time.deltaTime】

さっき `Update` を実行してから、今回、実行するまでに経過した時間が入っています。1秒間に約100回ですから、おおよそ 0.01 くらいが入っていると考えます。直訳だと極小時間ですが、デルタ秒と呼ぶことはあります。

- プレイボタンを押下します。 
おおよそ 0.01 くらいが入っていることが判ります。

この極小時間を加算する方法でも時間を測ることが出来ます。

0.01 を1秒間に 100 回加算したら1になるからです。

この `Time.deltaTime` を用いて時間を測ってみます。

0.0166672

- スクリプト `GameManager` を次のように編集します。

```
//～前略～  
  
public class GameManager : MonoBehaviour {  
  
    public Text txtTime; //画面のGUI文字  
    float Elapsed; //経過時間  
  
    void Start() {  
        Elapsed = 0.0f; //ゼロクリア  
    }  
  
    void Update() {  
        Elapsed += Time.deltaTime;  
        txtTime.text = Elapsed.ToString();  
    }  
}
```

- プレイボタンを押下します。 

見た目には大した違いはないのですが、一旦、Elapsed というワークエリアに経過時間を加算したものを表示している状況が異なります。

細かく言えば、Time.deltaTime は小数の最後の方が途切れていて、それを加算し続けた Elapsed は、先の Time.time とは異なる経過時間となります。コンピュータは小数を正確には保持できない性質があるからです。


5.476313

【STEP6】 表示を整形する

あまりにも長い小数点以下の数値は不要です。1/100 秒くらいの表示が妥当でしょう。

- スクリプト **GameManager** を次のように編集します。

```
//~前略~  
  
void Update() {  
    Elapsed += Time.deltaTime;  
    txtTime.text = Elapsed.ToString("f2") + "s";  
}  
}
```

- プレイボタンを押下します。 

本来、目指していた表記が出来るようになりました。

4.13s

ユーザーを監視する

時間を計測するのが Update 処理の大きな役目でしたが、もう一方は、ユーザーの監視です。

いきなり計時が始まるのを防ぎ、マウスのLMB押下で計時が始まるようにします。Update 処理を用いて、1 秒間に 100 回もユーザーがマウスを押すのでは？と監視する概念となります。

マウスには通常、ボタンが3つあります。それぞれの呼び方と番号を理解します。

また、ボタン押下には3つのイベントが発生しています。

- ① 押下した瞬間
- ② 押している間ずっと
- ③ 離した瞬間

これらを1秒間に約100回のペースで監視します。



【STEP7】 計時の開始・停止の制御

- スクリプト **GameManager** を次のように編集します。

【学習項目:真偽型】 bool 型変数

値として、true か false の2つのうち、どちらかしか保持が出来ない変数です。


プレイ中なら時間計時する仕組み
等号記号は == となる。

//～前略～

```
bool isPlaying; //プレイ中かどうかの真偽値
```

```
void Start() {  
    Elapsed = 0.0f; //ゼロクリア  
    txtTime.text = "0.00s";  
    isPlaying = false; //プレイ前  
}
```

```
void Update() {  
    if (isPlaying == true) {  
        Elapsed += Time.deltaTime;  
        txtTime.text = Elapsed.ToString( "f2" ) + "s";  
    }  
}
```

- プレイボタンを押下します。 
勝手に計時が始まらないようになりました。

今度は逆に、LMB押下を監視して検出し、計時を始めなくてはなりません。

0.00s

- スクリプト **GameManager** を次のように編集します。

Inputという膨大な入力情報の流れがあり、そこからマウスボタン(0)がDownしたことに耳を澄ましています。(1秒間に約100回)

//～前略～

```
void Update() {  
    if (Input.GetMouseButtonDown( 0 )) {  
        if (isPlaying == true) {  
            isPlaying = false;  
        } else {  
            isPlaying = true;  
        }  
    }  
    if (isPlaying == true) {  
        Elapsed += Time.deltaTime;  
        txtTime.text = Elapsed.ToString( "f2" ) + "s";  
    }  
}
```

- プレイボタンを押下します。



▼LMB 押下

計時中

▼3.67 秒後に LMB 押下

3.67s

表示は停止したまま

▼LMB 押下

計時中

LMBの押下の都度、計時が開始・停止を行うことを確認します。

【STEP8】 真偽型変数の特性

真偽型の特性よりも、条件分岐の制御文である if 文の性質に影響する項目となります。

制御文 if 文の条件式は、直後に続く小括弧 () 内に記述します。この小括弧の内容は、よほど長い計算式が掛かっていると、最終的には、**true か false のどちらか**にならなければ、条件式としては不合格です。

この **true か false のどちらか**という性質は、真偽値そのものとも言えます。よって、if 文で真偽型変数の内容を尋ねる場合は、その性質を最大限に利用し、シンプルな記述方法をする方が良いとされています。

- スクリプト **GameManager** を次のように編集します。

真偽値を入れ替える処理を、トグル式と呼ぶこともあります。

論理否定記号は **!** となります。

```
//～前略～

void Update() {
    if (Input.GetMouseButtonDown( 0 )) {
        isPlaying = !isPlaying;
    }
    if (isPlaying) {
        Elapsed += Time.deltaTime;
        txtTime.text = Elapsed.ToString( "f2" ) + "s";
    }
}
```

【STEP9】 リセット機能

タイムの計測中にリセット(ゼロクリア)する機能も理解できますが、ここでは、計測中では無く、停止している時のみリセットする機能とします。マウスの右ボタン(RMB)クリックでゼロクリアするように構成します。

- スクリプト **GameManager** を次のように編集します。

```
//～前略～

void Update() {
    if (Input.GetMouseButtonDown( 0 )) {
        isPlaying = !isPlaying;
    }
    if (isPlaying) {
        Elapsed += Time.deltaTime;
        txtTime.text = Elapsed.ToString( "f2" ) + "s";
    } else {
        //【命題】
    }
}
```

【命題】

プレイ中ではない時に、RMB押下を検出して、ストップウォッチをリセットしてください。

次のタイムを計り始めても正常であるか？の確認も必要です。

- プレイボタンを押下します。



計時を途中で停めた時、RMB押下でリセットできるようになりました。

0.00s

ラップタイムを運営する

ゲーム制作におけるラップ(Lap)表示とは、サーキットを3周するレースで考えると、1周ごとのタイムのことを指すのが一般的です。



これを、ストップウォッチの構造で考えます。

整理すると、時間の計測は継続しつつも、ある出来事のタイミングで表示を停めておく、これがラップ表示に求められる処理となります。

キーボードのエスケープキー「Esc」を押下すると、計時は続けながら、表示のみ停止させます。

- スクリプト **GameManager** を次のように編集します。

```
//～前略～

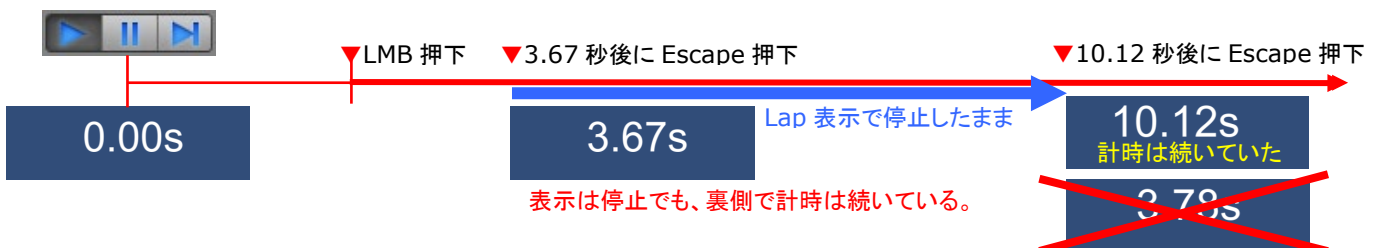
bool isPlaying; //プレイ中かどうかの真偽値
bool isLapStop; //Lap表示中かどうかの真偽値

void Start() {
    Elapsed = 0.0f; //ゼロクリア
    txtTime.text = "0.00s";
    isPlaying = false; //プレイ前
    isLapStop = false; //Lap中ではない
}

void Update() {
    if (Input.GetKeyDown( KeyCode.Escape )) {
        isLapStop = !isLapStop;
    }
    if (Input.GetMouseButtonDown( 0 )) {
        isPlaying = !isPlaying;
    }
    if (isPlaying) {
        Elapsed += Time.deltaTime;
        if (!isLapStop) {
            txtTime.text = Elapsed.ToString( "f2" ) + "s";
        }
    } else {
        if (Input.GetMouseButtonDown( 1 )) {
            Elapsed = 0.0f; //ゼロクリア
            txtTime.text = "0.00s";
            isLapStop = false; //Lap中ではない
        }
    }
}
```

キーボードのキーには、全て KeyCode が割り当てられており、全てを検出可能になっています。

- プレイボタンを押下します。



計時開始後、Escape 押下の都度、計時表示が停止・解除となることを確認します。

以上