今回のレッスンで学ぶ概念

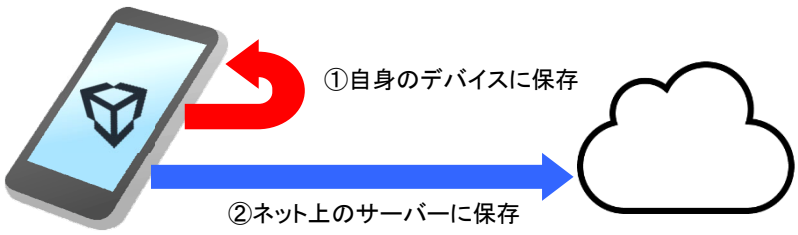
unity for 2019.2.21

データの永続性（ユーザーデフォルト領域の利用）

ゲームに関わらずビジネスアプリやホームページでも、一度利用した状況や情報を保存しておき、次に使う時に素早く取り出して用いる（表示などをする）データ運営は、一般的な概念となっています。そうでなきゃ不便です。

ブラウザで利用した情報はクッキーという仕組みで保存する概念もあり、また、優秀なブラウザであれば、ブラウザソフトそのものがクラウド上にデータを保存するなど、様々な手法が日々編み出されています。

次回に起動した時に使うデータを、どこかに温存させておく考え方を「データの永続性」と言い、Unityでも大きく2つの概念でデータ保存を実現できます。



【①自身のデバイスに保存】

今回取り組むのは、自身のデバイスに保存する概念となります。

軽微な変数データに限られますが、ゲームの進行状況や設定値、ハイスコアなどの保存と読み出しに用いることができます。但し、重要なデータや課金データなどをデバイスに保存してしまうと、書き換え行為などの懸念が出て来て、管理するデータの決定は十分に考慮した設計が必要です。

【②ネット上のサーバーに保存】

ネットワーク上のサーバーに保存する方法は、各社より様々なサービスが提供されており、スマホアプリ開発ではモバイル後方支援サービス（mbaaS：Mobile Backend as a Service：エムバース）が有名です。日本語で利用できるニフティクラウド・モバイルバックエンドやStrix Cloudなど、世界規模で様々なものがあります。実験的なものであれば、Googleスプレッドシートという表計算サービスでデータをセーブ・ロードする挑戦も聞きます。

PlayerPrefsの概念

UnityではPlayerPrefs（プレイヤープレフス）という機能を用いてデータを保存・取り出しを行います。プレイを終了し、再プレイしてもセーブデータがロードできます。PCのシャットダウンはもちろん、次の日だって大丈夫です。

つまりデバイス（機器本体）自体の奥深くにアプリが使える領域があり、そこに保存している訳です。ユーザーデフォルト領域と呼んだりします。

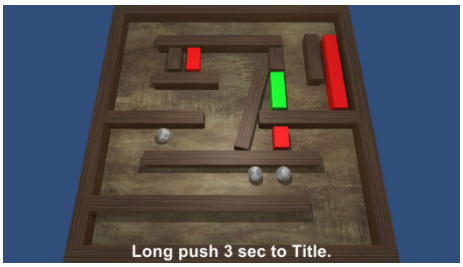
スマート機器でアプリを削除する時に、データも削除される確認を問われます。あれです。

概念的な話ですが、データを入れるタンスの引き出しに名前（キー）が付いていると考えます。キー名を指定して引き出しを開き、データを出し入れします。全部で10個の命令がありますが、次の9個だけで十分です。

「キー」を「引き出し」と読み替えてみて下さい。	セーブ系	ロード系
キーに対応する実数を格納・取得します。	SetFloat	GetFloat
キーに対応する整数を格納・取得します。	SetInt	GetInt
キーに対応する文字列を格納・取得します。	SetString	GetString
全てのキーと中身を削除します。注意して使います。	DeleteAll	
キーと対応する中身を削除します。	DeleteKey	
キーが存在するか？確認します。	HasKey	

今回は、ゲーム構築の手間にスポットライトを当てないので、既出のゲーム「BallMaze」を流用します。（都合上、ユーザーインターフェースに少量の差異があります。）

このゲームの完成状態を配布しますので、これを自身で作ったゲームであると仮定して、5位までのハイスコア管理機能を追加する手順を中心に解説を行います。



追加

一般的なゲームの場合に於いて「5位までのハイスコア管理」とは、以下の3パターンになると思われます。

1. 得点の高さを整数で競う。 順位は降順となる。 例：コイン取得の状況を、得点に換算する。
2. 時間の長さを小数で競う。 順位は降順となる。 例：自機3機が全滅するまで耐えきった時間。
3. 時間の短さを小数で競う。 順位は昇順となる。 例：ボールがゴールに到着するまでの時間。

（整数の昇順がないですねぇ。何か思い付くゲームはありますか？使った弾数の少なさ？無理かな？）

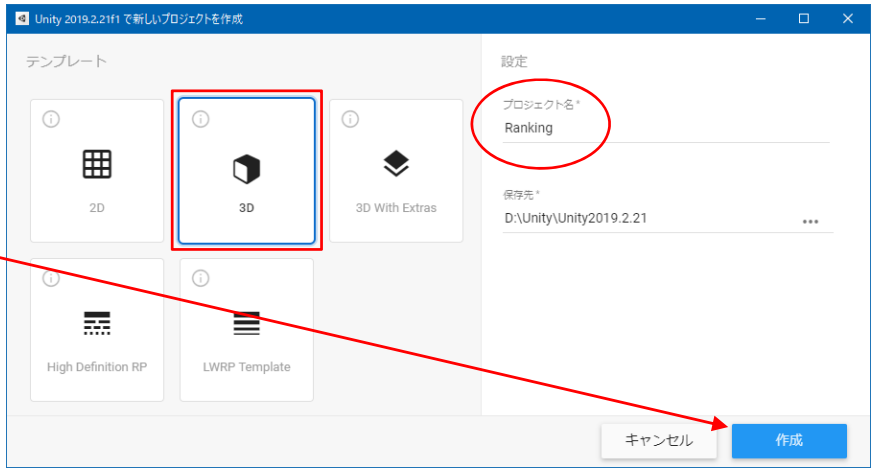
ゲーム「BallMaze」では、ボール3つがゴールに到着するまでの時間ですから、項番3の「時間の短さ」を小数の昇順でランキングすることになります。

プロジェクトの準備

【STEP1】 完成状態のインポート

- UnityHUB を起動して新しいプロジェクト Ranking（ランキング）を 3D モードで準備します。

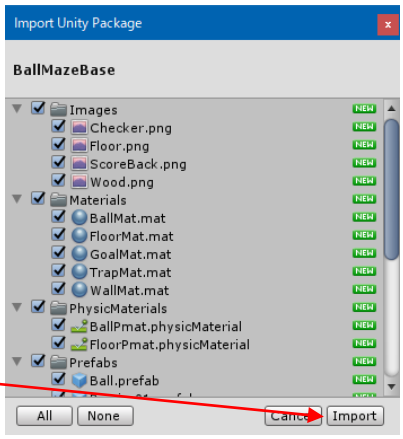
  ボタン作成を押下します。



- 配布された素材を読み込みます。

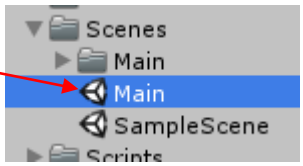
  メニューAssets（アセッツ）から Import Package（インポートパッケージ）＞ Custom Package（カスタムパッケージ）と進み、今回のフォルダ内にある BallMazeBase.unitypackage を指定します。

- 内容物が表示されたら、ボタン Import（インポート）を押下します。



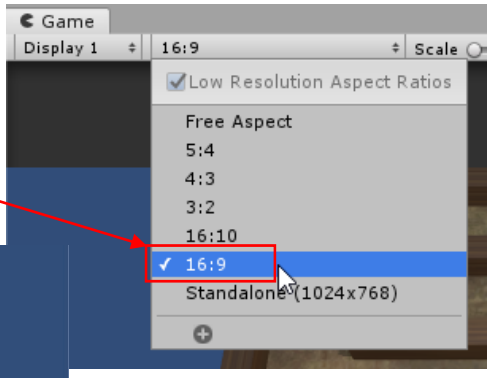
【STEP2】 完成ゲームのプレイ確認

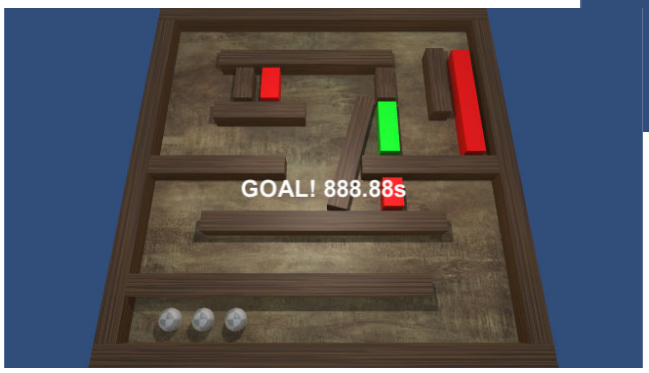
- プロジェクト欄の Scenes > Main をダブルクリックして表示します。



- 各人の制作プラットフォームの比率に応じ、ゲーム画面の比率を調整します。

ここでは、PCの全画面でプレイすることを想定して、16:9 の比率としています。iPad なら 4:3 となります。





- プレイボタンを押下します。 

このシーン Main は完成状態なので、通常にプレイ操作が可能です。ボール3個を同時にゴールに突入させるまでの時間を計測し、ゴール時にタイムを表示します。実際にゴールするまでの時間を測ってみます。

＜PCでのプレイ＞

画面をマウスでクリックするとゲームが開始され、キーボードの上下左右キーを操作してボールを操作します。ゴール後は画面を3秒長押しすることでタイトル画面に戻ります。これはプレイ中であっても可能です。

＜Xbox ゲームコントローラーfor Windows でのプレイ＞

コントローラーのAボタン押下でゲームが開始され、左スティックの操作でボールが移動します。ゴール後はAボタンを3秒長押しすることでタイトル画面に戻ります。これはプレイ中であっても可能です。

＜スマート機器でのプレイ＞

画面のどこでもタッチでゲームが開始され、スマート機器本体を傾ける操作で、ボールが転がる動きとして同調します。ゴール後は画面を3秒長押しすることでタイトル画面に戻ります。これはプレイ中であっても可能です。

ユーザーデフォルト領域の改廃作業

【STEP3】 保存領域の有無による対応

● コンテンツが始まった直後に、ユーザーデフォルト領域が既に存在するか？を確認します。データ領域が存在すれば、作業エリアにその値を取り込みます。存在しなければ、5件のタイム全てを初期化してデータ領域に保存し、作業エリアの内容も同じ値にします。

- 作業エリアを「0～4位」ではなく「1～5位」で表現する目的で、配列を6個で定義し、添え字「ゼロ」で指定されるデータ領域は未使用とする運用を採用します。
- 開発中のミスやトラブルを想定し、データ領域は完全破棄できるボタン（今回はエスケープキー）を設置するのが賢明です。

スクリプト GameManager を次のように編集します。

```
//～前略～
    float[] Rank = new float[ 6 ]; // 作業エリア

    void Start() {
        // アプリのデータ領域が存在するか
        if (PlayerPrefs.HasKey( "R1" )) {
            Debug.Log( "データ領域を読み込みました。" );
            for (int idx = 1; idx <= 5; idx++) {
                Rank[ idx ] = PlayerPrefs.GetFloat( "R" + idx ); // データ領域読み込み
            }
        } else {
            Debug.Log( "データ領域を初期化しました。" );
            for (int idx = 1;idx <= 5;idx ++) {
                Rank[ idx ] = float.MaxValue;
                PlayerPrefs.SetFloat("R" + idx, float.MaxValue); // 最大値を格納する
            }
        }
        Ready();
    }

//～中略～

    void Update() {
        switch (GameStatus) {
            case STS.INIT:
                //開発用：データ領域の初期化
                if (Input.GetKeyDown( KeyCode.Escape )) {
                    PlayerPrefs.DeleteAll();
                    Debug.Log( "データ領域を削除しました。" );
                }

//～後略～
```

今回は時間を小数（float）で扱う必要があるので、作業領域 Rank も小数（float）で定義されており、

- セーブの命令は SetFloat
- ロードの命令は GetFloat

となっていますが、得点などを扱う場合は整数（int）となるので、その場合は、

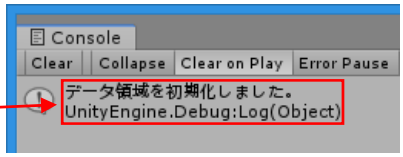
- セーブの命令は SetInt
- ロードの命令は GetInt

となります。

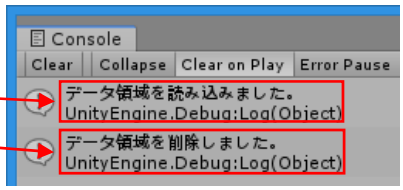
今回は順位が昇順になるので、初期化時に最大値（float.MaxValue）を代入していますが、大きい値が上位になる降順の場合は、小数であれ整数であれ、単純にゼロを代入することになります。

● プレイボタンを押下します。 

開始と同時に、初めてのプレイであることを検出し、データ領域（R1～R5）を作成し、内容を初期値で登録しています。



ここで一度、プレイを中断します。再度プレイボタンを押下すると、既に存在するデータをロードすることが判ります。
キーボードのエスケープキーを押下して、データ領域を強制的に削除できることも確認します。



【STEP4】 実際のスコアを登録する

● このゲームでは、ゴールオブジェクト（緑の直方体）が GameManager にメッセージ「ClearedAction」を告げたタイミングが所要タイム（経過時間 Elapsed）の確定です。この時に、①作業領域のランキングに入るかを検査します。ランクインする場合に、②タイムを作業領域に挿入し、データ領域にも転記します。
スクリプト GameManager を次のように編集します。

①の処理

②の処理

```
//～前略～
    void ClearedAction() {
        txtMessage.text = "GOAL! " + Elapsed.ToString("f2") + "s. Long push 3 sec to Title.";
        GameStatus = STS.CLEARED;

        int newRank = 0; //まず今回のタイムを0位と仮定する
        for (int idx = 5; idx > 0; idx--) {   //逆順 5...1
            if (Rank[ idx ] > Elapsed) { // 不等号（＊）
                newRank = idx; // 新しいランクとして判定する
            }
        }
        if (newRank != 0) { // 0位のままでなかったらランクイン確定
            for (int idx = 5; idx > newRank; idx--) { // 不等号（＊）
                Rank[ idx ] = Rank[ idx - 1 ];   // 繰り下げ処理
            }
            Rank[ newRank ] = Elapsed; // 新ランクに登録
            for (int idx = 1; idx <= 5; idx++) {
                PlayerPrefs.SetFloat("R" + idx, Rank[ idx ] ); // データ領域に保存
            }
        }
    }

//～後略～
```

（＊）大小関係を比較する2か所で、昇順の場合は不等号記号が「＞」となります。逆に、降順になる場合は、記号が逆転して「＜」となります。

● プレイボタンを押下します。 

実際にプレイを2回ほど行い、プレイタイムを2件ほど登録しておきます。きちんと順位通りにデータ領域に格納されているかどうかは、後ほど判明します。

ランキング画面の運営

【STEP5】 ランキング画面（シーン）へ遷移する

● ゴールしたあとは、画面の3秒長押しでタイトルに戻っていましたが、これを改造します。画面のタッチで、新設するランキング画面（Ranking）に遷移することとします。

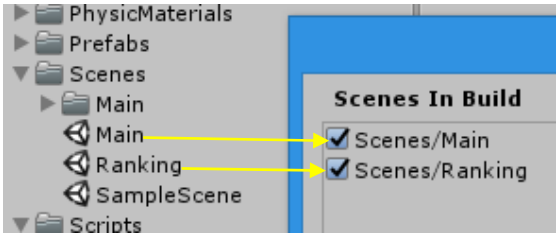
スクリプト GameManager を次のように編集します。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI; //uGUIを扱うのに必要
using UnityEngine.SceneManagement; //シーンのロードに必要

//～中略～

    void ClearedAction() {
        txtMessage.text = "GOAL! " + Elapsed.ToString("f2") + "s. Please touch to Ranking.";
        GameStatus = STS.CLEARED;

//～中略～
            case STS.CLEARED:
                //LongPushDetect(); //コメントアウトします
                if (Input.GetButtonDown("Fire1" )) {
                    SceneManager.LoadScene("Ranking");
                }
                break;
            default:
                break;
        }
    }
}
```

【STEP6】 ランキング画面の新設

● メニューFile から New Scene を選択します。何も設定をしていませんが、保存（Ctrl + S）を行います。シーン名を Ranking とします。

● メニューFile から Build Settings…を選択します。
プロジェクト欄のシーン Main と Ranking をビルド一覧（Scenes In Build）にドラッグ＆ドロップします。1行目に Main が来るようにします。
このパネルは閉じておきます。

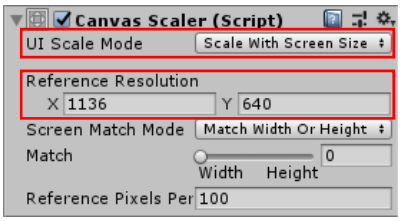


【履修項目】 別シーンへの画面遷移

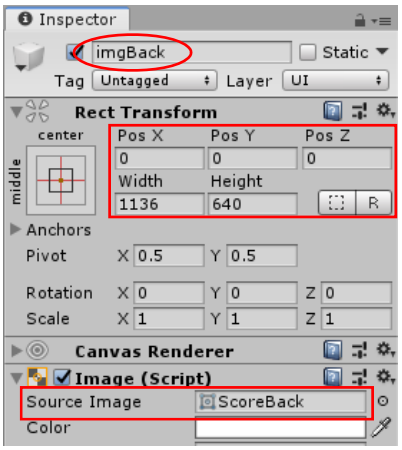
別のシーンをロードする考え方で、次の画面（シーン）を登場させることが出来ます。①ロード命令は、名前空間の定義が必要であること、②ビルド一覧にシーンを登録しておくこと、この2点を忘れずに。

【STEP7】ランキング画面のユーザーインターフェース作成

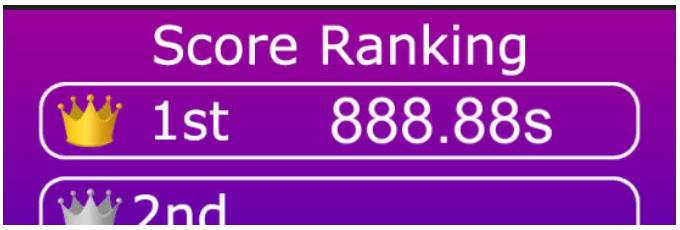
- ヒエラルキー欄の Create から UI ＞ Image を選択し、imgBack と命名します。
- 同時に生成されたヒエラルキー欄の Canvas を選択し、インスペクタでパラメータを設定します。

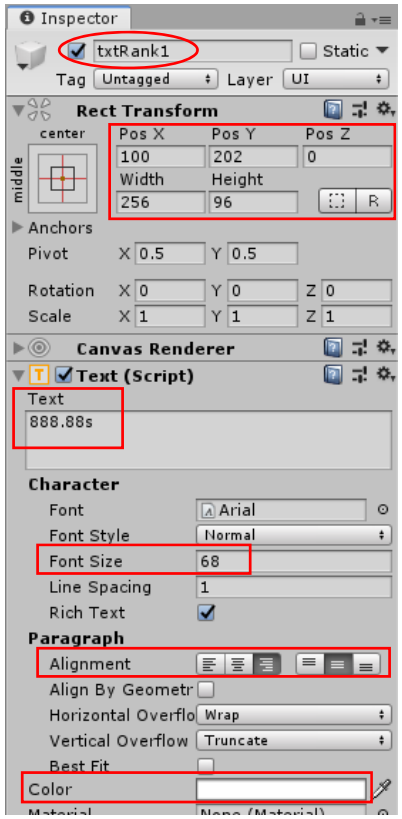


- ヒエラルキー欄の imgBack を選択し、インスペクタでパラメータを設定します。



- ヒエラルキー欄の Create から UI ＞ Text を選択し、txtRank1 と命名します。
  インスペクタでパラメータを設定します。



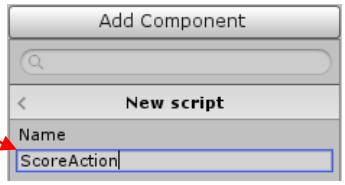


- ヒエラルキー欄の txtRank1 を選択し、4回複製（Ctrl ＋ D）して、計5個にします。それぞれ名称とポジションY座標値を設定します。

  - txtRank1　　Position-Y：202
  - txtRank2　　Position-Y：102
  - txtRank3　　Position-Y：2
  - txtRank4　　Position-Y：-98
  - txtRank5　　Position-Y：-198

【STEP8】ハイスコアの表示と画面遷移

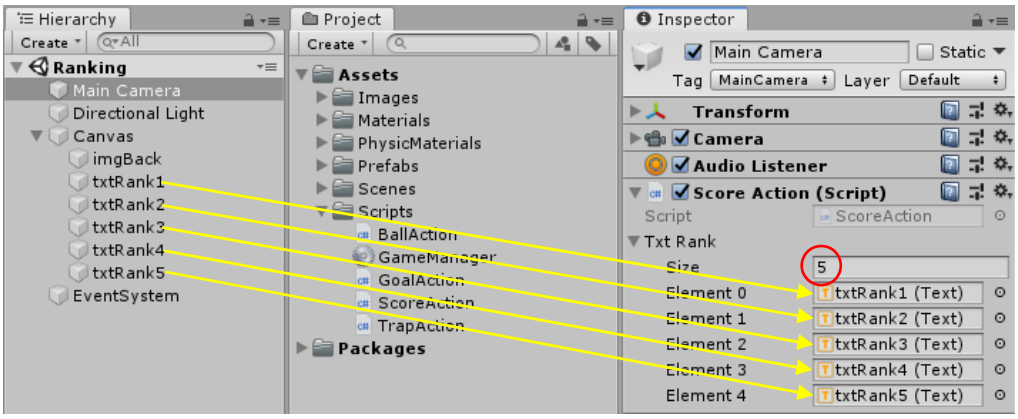
- ヒエラルキー欄の Main Camera を選択し、インスペクタの Add Component から New script を選択します。名称を ScoreAction と命名します。



スクリプト ScoreAction を次のように編集します。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI; //uGUIを扱うのに必要
using UnityEngine.SceneManagement; //シーンのロードに必要

public class ScoreAction : MonoBehaviour {

    public Text[] txtRank;
    float Elapsed = 0.0f;

    void Start() {
        for (int idx = 1;idx <= 5;idx ++) {
            if (PlayerPrefs.GetFloat( "R" + idx ) >= float.MaxValue) {
                txtRank[ idx - 1 ].text = "_.__s";
            } else {
                txtRank[ idx - 1 ].text = PlayerPrefs.GetFloat( "R" + idx ).ToString( "f2" ) + "s";
            }
        }
    }

    void Update() {
        if (Input.GetButton( "Fire1" )) {
            Elapsed += Time.deltaTime; //画面を押し続けている間ずっと
            if (Elapsed > 3.0f) {
                SceneManager.LoadScene("Main");
            }
        }
        if (Input.GetButtonUp( "Fire1" )) {
            Elapsed = 0.0f; //画面押しを離した
        }
    }
}
```

- 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を設定します。



- 必ずここで現在のシーン Ranking を保存（Ctrl ＋ S）します。
- プロジェクト欄のシーン Main をダブルクリックして読み込みます。
- プレイボタンを押下します。

ゲームプレイを行い、ゴール後にランキング画面に進み、正しくスコアが表示されていることを確認します。

付録

【パターン別の差異】

■ 時間の短さを小数で競う場合 （この BallMaze の通り）

作業域 Rank を小数（float）で管理し、初期値に float.MaxValue を代入する。

データ領域の入出力には、SetFloat と GetFloat を使う。

現在のランキングと照合する際の不等号記号「＞」

画面に表示時に「s」を付加する。float.MaxValue の表示を避け、「__.____s」などの工夫が必要。

■ 時間の長さを小数で競う場合

作業域 Rank を小数（float）で管理し、初期値にゼロを代入する。

データ領域の入出力には、SetFloat と GetFloat を使う。

現在のランキングと照合する際の不等号記号「＜」

画面に表示時に「s」を付加する。

■ 得点の高さを整数で競う場合

作業域 Rank を整数（int）で管理し、初期値にゼロを代入する。

データ領域の入出力には、SetInt と GetInt を使う。

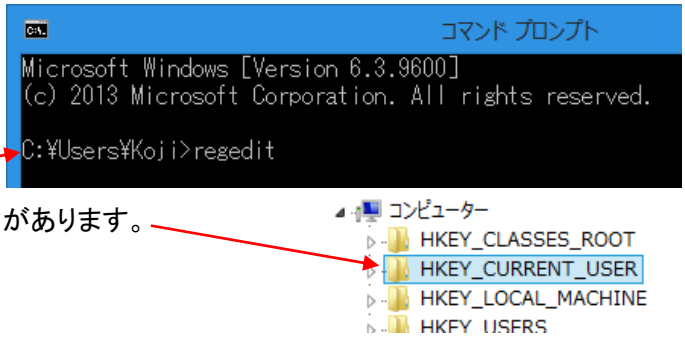
現在のランキングと照合する際の不等号記号「＜」

画面に表示時に「G」や「点」などの単位を付加することが望ましい。

【データ保存の場所】

ここからはPCの場合でのデータ保存場所を確認します。PCが起動しなくなる可能性が高いので、レジストリエディタはむやみに変更を加えないことに注意しましょう。

- コマンドプロンプトを起動します。アイコンが見つからなければプログラム cmd を探して実行します。
- レジストリエディタ regedit を起動します。
- 項目HKEY_CURRENT_USER内に保存したデータがあります。



Windows機器での保存場所を紹介しましたが、iOSやAndroidなど、OSが変われば保存場所やそのアクセス方法も変わります。PlayerPrefsについての詳しい情報は、以下のオンライン・マニュアル（日本語）を参考にして下さい。

https://docs.unity3d.com/ja/current/ScriptReference/PlayerPrefs.html

以上