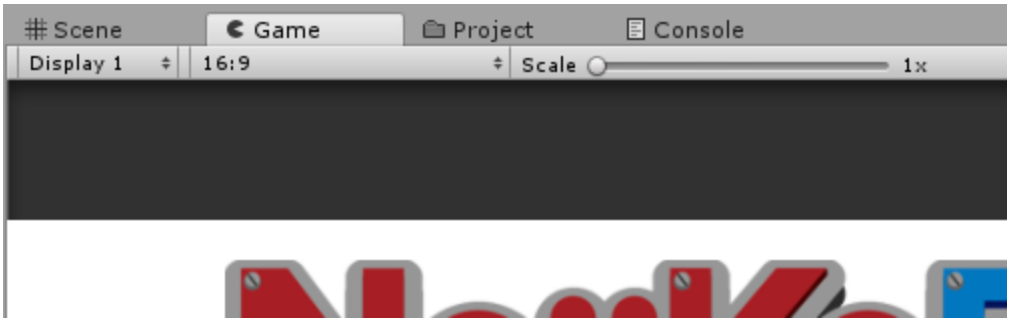
Webプログラミング実習Ⅱ　授業用レジュメ③

■Unityでの作業

※以下、教科書で制作した『NejikoRun』を改変する形で作業します。

■下準備

●ゲームビュー左上のドロップダウンから、アスペクト比(画面の縦横比)を「16:9」にしておく
※下の画像のようになっていればＯＫ



■Titleシーンの編集

●エンティティクラス、JsonHelper、ConnectManagerの作成
(MonoBehaviorを継承してない通常のクラスであることに注意！)

【NejikoMileage】：ユーザー名とスコアの情報ひと組を保持するエンティティクラス

```
using System.Collections;
using System.Collections.Generic;

[System.Serializable]
public class NejikoMileage
```

```
{
    public string name;
    public int score;

    //コンストラクタ
    public NejikoMileage(string name, int score)
    {
        this.name = name;
        this.score = score;
    }
}
```

【JsonHelper】：サーブレットから送られてくるJsonを正しく読み込む手助けをするクラス

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System;

/// <summary>
/// クラスのList型コレクションをJsonUtilityでデシリアライズするヘルパー
/// http://forum.unity3d.com/threads/how-to-load-an-array-with-jsonutility.375735/
/// </summary>
public class JsonHelper
{
    /// <summary>
    /// List(配列含む)になっているJSONデータをデシリアライズする
    /// </summary>
```

```
    /// <typeparam name="T">List(配列含む)になっているデータの型</typeparam>
    /// <param name="json">json文字列</param>
    /// <returns>FromJsonで生成されたオブジェクトのList</returns>
    public static List<T> GetJsonArray<T>(string json)
    {
        string newJson = "{ \"array\": " + json + "}";
        Wrapper<T> wrapper = JsonUtility.FromJson<Wrapper<T>>(newJson);
        return wrapper.array;
    }

    [Serializable]
    private class Wrapper<T>
    {
        //『#pragma ~』は `プログラムの動きとは直接関係ない記述
        // (『この番号の警告を無視しろ』というコンパイラへの命令文)
        #pragma warning disable 649
        public List<T> array;
    }
}
```

【ConnectManager】：Webへの接続処理全てを担うクラス

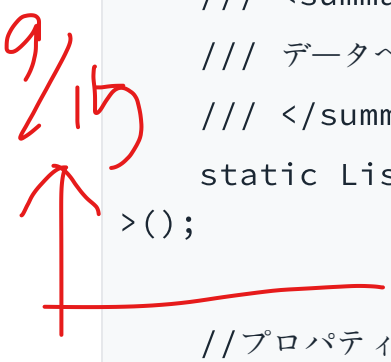
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public class ConnectManager
{
```

```
    /// <summary>
    /// 接続するサーブレットへのルートパス
    /// </summary>
    const string ROOTPATH = "http://localhost:8080/nejikoRunOnlineServer/NejikoMileageServlet";
    /// <summary>
    /// サーブレットから返ってきた文字列
    /// </summary>
    static string responseText = "";
    /// <summary>
    /// 接続中にエラーが起こったかどうかのフラグ
    /// </summary>
    static bool isError = true;
    /// <summary>
    /// データベースから取り出した上位3位の走行記録
    /// </summary>
    static List<NejikoMileage> result = new List<NejikoMileage>();

    //プロパティ
    public static bool IsError
    {
        get { return isError; }
    }

    public static List<NejikoMileage> Result
    {
        get { return result; }
    }
```



```
    /// <summary>
    /// ランキング情報を取得する際に呼び出すコルーチン
    /// </summary>
    /// <returns></returns>
    public static IEnumerator FindMileage()
    {
        //Webに接続してデータを取得
        yield return GetRequest();

        if (isError)
        {
            Debug.Log("GetRanking:通信エラー");
        }
        else
        {
            result = JsonHelper.GetJsonArray<NejikoMileage>(re
sponseText);
            foreach (NejikoMileage nm in result)
            {
                Debug.Log(nm.name + "/" + nm.score);
            }
            Debug.Log("正常終了　取得したListの内容：" + result +
"/Listの長さ：" + result.Count);
        }
    }

    /// <summary>
    /// 走行記録の書き込みを行う際に呼び出すコルーチン
    /// </summary>
    /// <returns></returns>
```

```
    public static IEnumerator RegistMileage()
    {
        //ゲーム中のセーブデータをもとに送信用ファイルを生成
        NejikoMileage sendRecord = MakeNejikoMileage();

        //サーブレットへPostリクエストを行い`送信用ファイルの内容を送信
        yield return PostRequest(sendRecord);

        //実行結果に応じてログを表示
        if (isError)
        {
            Debug.Log("SetRecord:通信エラー");
        }
        else
        {
            Debug.Log("SetRecord:正常終了");
        }
    }

    /// <summary>
    /// 実際にサーブレットへGETリクエストを行うコルーチン
    /// </summary>
    /// <returns></returns>
    static IEnumerator GetRequest()
    {
        //変数の初期化
        InitVar();

        UnityWebRequest request = new UnityWebRequest(ROOTPAT
H);
```

```
        request.downloadHandler = new DownloadHandlerBuffer();
        request.SetRequestHeader("Content-Type","application/j
son");

        //レスポンスが返ってくるまで実行を待つ
        yield return request.SendWebRequest();

        //接続の途中でエラーが起きた場合はログを返す
        if (request.isNetworkError || request.isHttpError)
        {
            isError = true;
            Debug.Log("WebConnectionGet:通信エラー / " + reques
t.error);
            Debug.Log("WebConnectionGet:エラーコード / " + reque
st.responseCode);
        }

        //正常に接続できた場合
        else
        {
            responseText = request.downloadHandler.text;
            isError = false;

            //デバッグ用表示
            Debug.Log("WebConnectGet.サーブレットから受け取ったJSO
N：" + responseText);

            result = JsonHelper.GetJsonArray<NejikoMileage>(re
sponseText);
            foreach (NejikoMileage nm in result)
            {
```

```
                Debug.Log(nm.name + "/" + nm.score);
            }
            Debug.Log("正常終了　取得したList内容：" + result + "/Listの長さ：" + result.Count);
        }
    }

    /// <summary>
    /// 実際にサ―ブレットへPOSTリクエストを行うコル―チン
    /// </summary>
    /// <param name="mileage"></param>
    /// <returns></returns>
    static IEnumerator PostRequest(NejikoMileage mileage)
    {
        //変数を初期化
        InitVar();

        //引数で渡されたクラスの情報を引き出し`送りたいPOSTリクエストのリクエストパラメ―タに入れる
        WWWForm form = new WWWForm();
        form.AddField("name", mileage.name);
        form.AddField("score", mileage.score);
        UnityWebRequest request = UnityWebRequest.Post(ROOTPATH, form);
        yield return request.SendWebRequest();

        //接続の途中でエラ―が起きた場合はログを返す
        if (request.isNetworkError || request.isHttpError)
        {
            isError = true;
```

```
            Debug.Log("WebConnectionPost：通信エラー / " + reque
st.error);
            Debug.Log("WebConnectionPost：エラーコード / " + requ
est.responseCode);
        }
        //正常に接続できた場合
        else
        {
            //成功/失敗の文字列が返ってくるので受け取る
            responseText = request.downloadHandler.text;
            //書き込み成功なら
            if (responseText == "SUCCESS")
            {
                Debug.Log("WebConnectionPost:レコード書き込み完
了");
                isError = false;

            }
            //失敗なら
            else
            {
                Debug.Log("WebConnectionPost:レコード書き込みエラ
ー");
                isError = true;
            }
        }
    }

    /// <summary>
    /// 取得したJSON文字列`エラー発生フラグの初期化用メソッド
    /// </summary>
```

```
    public static void InitVar()
    {
        responseText = "";
        isError = true;
    }

    /// <summary>
    /// ゲーム中のセーブデータをもとに送信用ファイルを生成するメソッド
    /// </summary>
    static NejikoMileage MakeNejikoMileage()
    {

        Debug.Log("name:" + PlayerPrefs.GetString("Name") + "/
score:" + PlayerPrefs.GetInt("HighScore"));
        string name = PlayerPrefs.GetString("Name");
        int score = PlayerPrefs.GetInt("HighScore");

        return new NejikoMileage(name, score);
    }
}
```

●Canvas内へのゲームオブジェクト追加
（★マークのゲームオブジェクトは、作り終わった後非アクティブにしておく）

★(Panel)RankingTablePanel：ランキング全体の位置を定めるPanel

【Rect TransFormの値】

・アンカー　　：middle-right
・PosX,Y,Z　　：-115 , -100, 0
・Width,Height: 210 , 90

[RankingTablePanelの子オブジェクト]

※この項の『(Text)』と書かれたゲームオブジェクトについては、
「Text」コンポーネント「Paragraph」内の設定項目「Alignment」で、
文字列の合わせ方向を「中央寄せ・中央合わせ」に変えておくこと

下の画像のようになっていればＯＫ

●(Text)HeadLine：見出し（Textの内容を『ランキング』に変更）

【Rect TransFormの値】
・アンカー　　：left-top
・PosX , Y , Z　　：35 , 15 , 0
・Width , Height ：120 , 20

★(Panel)1stRow、2ndRow、3rdRow

※各順位の情報表示欄。同じ構造のものを3つ作るので、
１セット作った時点で下記の子オブジェクトごとプレファブ化しておくと楽

【Rect TransFormの値】
・アンカー　　：left-top
・PosX , Y , Z　　：105 , -15 (1st) / -45 (2nd) / -75 (3rd) , 0
・Width , Height ：210 , 30

[1st～3rdRowの子オブジェクト(それぞれに1セットずつ用意する)]

●(Text)Rank：(各順位に合わせて、Textの内容を『1位』～『3位』に変更)

【Rect TransFormの値】

・アンカー　　：left-top
・PosX , Y , Z　　：25 , -15 , 0
・Width , Height ：40 , 25

●(Text)Name：ここにユーザー名が表示される(Textの内容を『ユーザー』に変更)

【Rect TransFormの値】

・アンカー　　：left-top
・PosX , Y , Z　　：90 , -15 , 0
・Width , Height ：90 , 30

●(Text)Score：ここに得点(走行距離)が表示される(Textの内容を『9999』に変更)

【Rect TransFormの値】

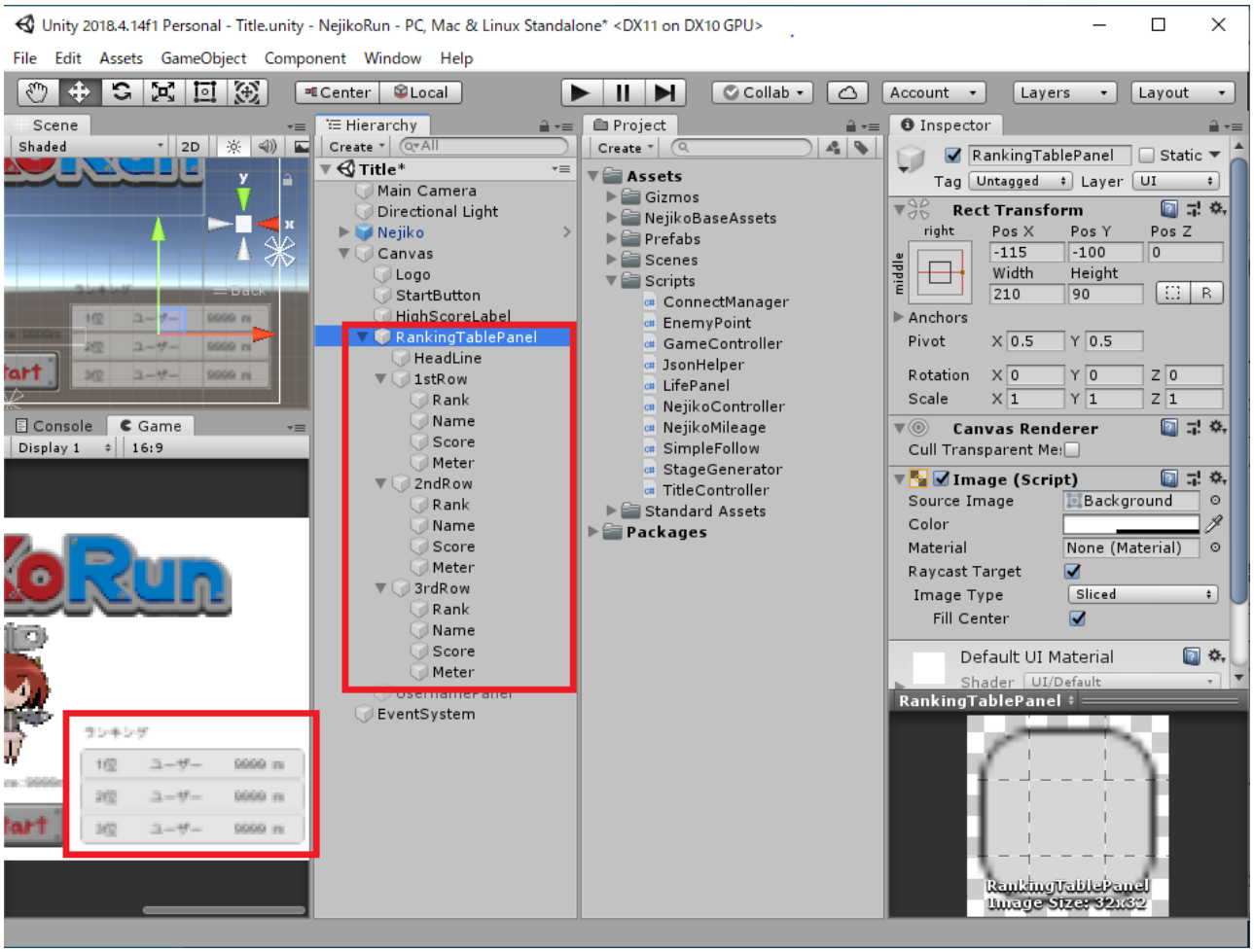
・アンカー　　：left-top
・PosX , Y , Z　　：159 , -15 , 0
・Width , Height ：40 , 30

●(Text)Meter：走行距離の単位表記(Textの内容を『m』に変更)

【Rect TransFormの値】

・アンカー　　：left-top
・PosX , Y , Z　　：185 , -15 , 0
・Width , Height ：15 , 30

【RankingTablePanelの完成図】



●(Panel)UsernamePanel：ユーザーの名前入力欄の位置を定めるPanel

【Rect TransFormの値】

・アンカー　　：right-middle
・PosX , Y , Z　　：-115 , 5 , 0
・Width , Height ：210 , 70

[UsernamePanelの子オブジェクト]

※『(Text)』と書かれたゲームオブジェクトについては、
RankingTablePanelと同様に文字列の合わせ方向を「中央寄せ・中央合わせ」に変えておくこと

●(Text)HeadLine：見出し。Textの内容を「ユーザー名(6文字まで)」に変更しておく

【Rect TransFormの値】

・アンカー　　：left-top
・PosX , Y , Z　　：70 , 10 , 0
・Width , Height ：150 , 20

●(Button)SubmitButton:「登録」ボタン。Textの内容を「登録」に変更しておく

【Rect TransFormの値】
・アンカー　　：left-top
・PosX , Y , Z　　：190 , -23 , 0
・Width , Height ：35 , 30

●(Text)UserNameLabel：入力名やエラーの表示欄。Textの内容を「エラーメッセージ」に変更しておく

【Rect TransFormの値】
・アンカー　　：left-top
・PosX , Y , Z　　：100 , -55 , 0
・Width , Height ：180 , 30

●(InputField)InputField：ユーザー名の入力欄

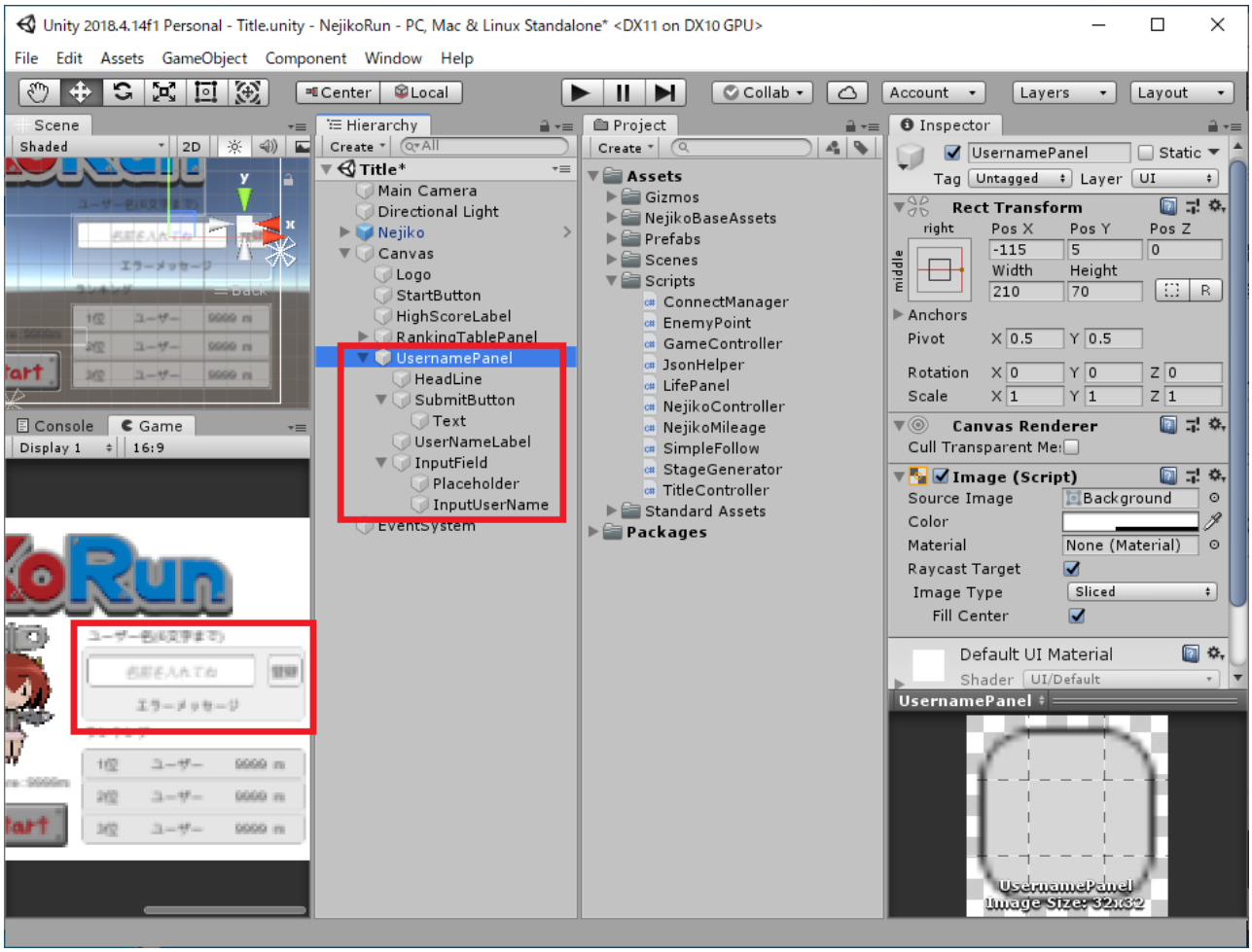
【Rect TransFormの値】
・アンカー　　：left-top
・PosX , Y , Z　　：85 , -23 , 0
・Width , Height ：160 , 30

[InputFieldの子オブジェクト(最初から作成されているので名前変更などを行う)]

●(Text)InputUserName(『Text』から名前を変更)：入力欄へ入力された文字列がここに入る

●(PlaceHolder)PlaceHolder：Textの内容を「名前を入れてね」に変更

【UsernamePanelの完成図】



（▲マークのゲームオブジェクトは、作り終わった後「Text」コンポーネントを無効にしておく）

▲(Text)NetworkState：ネットワーク接続の結果を表示

(Textコンポーネントの設定項目『Color』から**文字色を赤色に、**
中身のテキストを『**state:ランキング情報取得中・・・**』にしておく)

【Rect TransFormの値】

・アンカー　　：left-bottom
・PosX , Y , Z　　：120 , 30 , 0
・Width , Height ：220 , 30

●TitleControllerの追記

青色になっている部分を追記してください。

【TitleController(Web対応＆シーン転換処理変更版)】

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class TitleController : MonoBehaviour
{

    public Text highScoreLabel;
    public bool isDebug;//デバッグ用
    public Text userNameLabel;
    public Text inputUserName;

    /// <summary>
    /// 通信状態表示用の文字列
    /// </summary>
    public Text networkState;

    /// <summary>
    /// ランキング情報表示用のパネル
    /// </summary>
    public GameObject rankingTablePanel;

    /// <summary>
    /// 1位~3位の表示欄が入った配列
    /// </summary>
    public GameObject[] rankingRows;
```

```
    public void Start ()
        {
        //デバッグ用
        //(インスペクタウィンドウ上の「isDebug」にチェックを入れると`ゲーム起動毎にデータが消える)
        if (isDebug)
        {
            //isDebugにチェックが入っていたら`すべてのPrayerPrefsを削除
            PlayerPrefs.DeleteAll();
        }

                // ハイスコアを表示
                highScoreLabel.text = "High Score : " + PlayerPrefs.GetInt("HighScore") + "m";

        //登録しているユーザー名を表示
        UpdateUserName();

        //取得したConnectManagerから情報を引き出し`各種画面表示を行う
        StartCoroutine(InitWebUI());
    }

        public void OnStartButtonClicked ()
        {
                SceneManager.LoadScene("Main");
        }

    /// <summary>
    /// 「登録」ボタンを押した時の処理
```

```
    /// </summary>
    public void OnSubmitButtonClicked()
    {
        //正常に入力されていたらユーザー名を更新
        if (inputUserName.text.Length < 7)
        {
            PlayerPrefs.SetString("Name", inputUserName.text);
            UpdateUserName();
        }
        else
        {
            Debug.Log("文字数: " + inputUserName.text.Length);
            userNameLabel.text = "6文字を超えています";
        }
    }

    /// <summary>
    /// ユーザー名の画面表示･更新処理
    /// </summary>
    void UpdateUserName()
    {
        string name = PlayerPrefs.GetString("Name");
        Debug.Log("TitleController : 現在登録されているユーザー名 /
" + name);

        //初回起動時（プレイヤー名が登録されていない時）や未入力で登録時、
ユーザー名を「Noname」に
        if (string.IsNullOrEmpty(name))
        {
            PlayerPrefs.SetString("Name", "Noname");
```

```
            name = "Noname";
        }
        //ユーザー名入力フォームに現在のユーザー名を入れておく
        userNameLabel.text = "ユーザー名：" + name;
    }

    /// <summary>
    /// サーバへ接続し`取得した情報を元にUIを表示させるコルーチン
    /// </summary>
    /// <returns></returns>
    IEnumerator InitWebUI()
    {
        //ネットワークに接続、ランキング情報を取得
        yield return StartCoroutine(ConnectManager.FindMileage
());
        //情報に合わせて各種画面表示
        ShowWebUI();
    }

    /// <summary>
    /// 通信状態･ランキング情報をシーンへ表示するメソッド
    /// </summary>
    void ShowWebUI()
    {
        string baseText = "state:";

        //エラーならランキングを表示しない
        if (ConnectManager.IsError)
        {
            networkState.enabled = true;
```

```
            networkState.text = baseText + "通信エラーが発生しました";
        }
        else
        {
            networkState.enabled = true;
            networkState.text = baseText + "ランキング取得完了";
            WriteRankingTable();
        }
    }

    /// <summary>
    /// ランキング情報部分の表示処理
    /// </summary>
    void WriteRankingTable()
    {
        //各順位の情報を取得
        List<NejikoMileage> mileages = ConnectManager.Result;

        //表示用パネルをアクティブ化
        rankingTablePanel.SetActive(true);

        //各順位の情報を表示
        for (int i = 0; i < mileages.Count; i++)
        {
            ShowRankingRow(i);
            WriteRankingRow(rankingRows[i], mileages[i]);
        }
    }
```

```
    /// <summary>
    /// 指定したindexのランキング情報をアクティブ化するメソッド
    /// </summary>
    /// <param name="index"></param>
    void ShowRankingRow(int index)
    {
        rankingRows[index].SetActive(true);
    }

    /// <summary>
    /// 指定したcolumに指定したmileageの情報を表示するメソッド
    /// </summary>
    /// <param name="row">走行記録の情報を表示する行</param>
    /// <param name="mileage">ランキングから取り出した走行記録</param>
    void WriteRankingRow(GameObject row, NejikoMileage mileage)
    {
        Text[] topics = row.GetComponentsInChildren<Text>();
        //Name·Scoreの表示内容をrecord通りに変更
        topics[1].text = mileage.name;
        topics[2].text = mileage.score.ToString();
    }
}
```

●TitleControllerオブジェクトを選択、インスペクタービューでアタッチされている【TitleController】内のpublic変数へ以下のゲームオブジェクトをドラッグ&ドロップ

- **User Name Label ← UserNameLabel**
- **High Score Label ← HighScoreLabel**

・**Input User Name ← InputUserName**

・**NetWork State　← NetWorkState**

・**Ranking Table Panel ← RankingTablePanel**

・**Ranking Rows** ← Sizeを「3」にした後、
「Element0」～「Element2」に「1stRow」～「3rdRow」をドロップ

●ヒエラルキーウィンドウでSubmitButtonを選択、
　インスペクタービューの「Button」コンポーネント内「OnClick」に項目を追加(＋をクリック)。
　項目左下にTitleControllerをドラッグ&ドロップし、呼び出すメソッドを選ぶドロップダウンで
　「TitleController」→「OnSubmitButtonClicked」を選択

■Mainシーンの編集

●Canvas内へゲームオブジェクトを追加
（▲マークのゲームオブジェクトは、作り終わった後「Text」コンポーネントを無効にしておく）

▲(Text)UpdateInfoLabel：Textの内容を「ハイスコア更新！」にしておく
【Rect TransFormの値】
・アンカー　　：middle-center
・PosX , Y , Z　：0 , 37 , 0
・Width , Height ：160 , 30

▲(Text)ResistInfoLabel：Textの内容を「データベースに登録しました」にしておく
【Rect TransFormの値】
・アンカー　　：middle-center
・PosX , Y , Z　：0 , 0 , 0
・Width , Height ：240 , 30

●GameControllerの内容変更

青色になっている部分を追記・変更してください。

【GameController】：ネットワーク通信対応後

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class GameController : MonoBehaviour
{
    //ライフを表す変数(life)の持ち主
    public NejikoController nejiko;
    //スコアを表示したいTextオブジェクトが持つTextコンポーネント
    public Text scoreText;
    //ライフの画像が入ったパネルが持つライフ表示更新用スクリプト
    public LifePanel lifePanel;

    /// <summary>
    /// 「ハイスコア更新！」表示用テキスト
    /// </summary>
    public Text UpdateInfoLabel;

    /// <summary>
    /// 「データベースに登録しました」表示用テキスト
    /// </summary>
    public Text RegistInfoLabel;

        // Update is called once per frame
    void Update()
    {
```

```
        //現在スコアを計算｀スコアに合わせてスコア表示用Textの内容を更新
        int score = CalcScore();
        scoreText.text = "Score : " + score + "m";

        //lifepanel内のメソッドを使い｀アイコン表示を更新
        lifePanel.UpdateLife(nejiko.Life());

        //ねじ子のライフが無くなっていたらゲームオーバー｀タイトル画面に遷移
        if (nejiko.Life() <= 0)
        {
            //このスクリプト自体を非アクティブに
            //(これ以上このスクリプトの処理を行わないように)
            //【重要なポイント】を参照
            enabled = false;


            //コルーチンにしたゲーム終了処理を開始
            StartCoroutine(ReturnToTitle(score));
        }

    }

    //スコア計算用メソッド
    int CalcScore()
    {
        //ねじ子の現在地＝ねじ子の走行距離(小数切り捨て)がそのままスコアとなる
        //int型にキャストオフしている点に注意
        return (int)nejiko.transform.position.z;
```

```
    }

    /// <summary>
    /// ハイスコアの登録処理を行い`タイトルへ戻るコルーチン
    /// </summary>
    /// <param name="score">今回の走行距離</param>
    /// <returns></returns>
    IEnumerator ReturnToTitle(int score)
    {
        // ハイスコアを更新
        if (PlayerPrefs.GetInt("HighScore") < score)
        {
            PlayerPrefs.SetInt("HighScore", score);

            //「ハイスコア更新！」の文字列を表示
            UpdateInfoLabel.enabled = true;

            //走行記録をDBへ書き込み
            yield return StartCoroutine(ConnectManager.RegistM
ileage());

            //正常に書き込みできていたらゲーム画面に通知を表示
            if (!ConnectManager.IsError)
            {
                RegistInfoLabel.enabled = true;
            }
        }

        // 3秒待機
        yield return new WaitForSeconds(3.0f);
```

```
        //  タイトルシーンに切り替え
        SceneManager.LoadScene("Title");
    }
}
```

●追加したフィールド変数に同名のゲームオブジェクトをアタッチ

■試験運用
MAMP・Eclipse側のプログラムを立ち上げた状態でゲームをプレイ

●ゲームを立ち上げた際
・タイトル画面にランキングが表示されているか確認する

・一度MAMPを落としてみて、立ち上げた際にランキングが表示されず、「通信エラー」の文字が出るか確認する

●ハイスコアを更新した際
・「ハイスコア更新！」「データベースに登録しました」という文字列が表示されているか確認する

・データベース(phpMyAdmin)を開き、入力した名前・その際のスコア通りの情報が入力されているか確認する