OFFLINE LEADERBOARDS

BY JEFF W. MURRAY (PSYCHICPARROT)

WELCOME

Thank you for purchasing! I hope this is useful for your games.

The Offline Leaderboard pack includes everything you need to add local (as in, offline) high scores to your games. Also included is an example project to help get you started. Feel free to copy and paste anything you want out of it and use it in your own games.

INCLUDED IN THE ZIP FILE:

1. This document, in .pdf and .rtf formats.
2. An example unity project file, containing the high score scripts. (NOTE: All scripts and classes are prefixed with LB_ so that they don't clash with your own scripts and you can easily see which ones belong to this leaderboard pack)
3. Good karma.

THE EXAMPLE

OK, so I wanted to give you something that would make it as easy as possible to understand how this leaderboard code might be used. There was no need for a full game example, but I wanted to (at the very least) have an overall structure in place. That is; a menu with a link to the leaderboard screen, a show leaderboard screen and a really simple example of how a game might 'talk' to the leaderboard.

I hope I managed to make something that supports this document in outlining how it works.

HOW TO USE THE HIGH SCORE SCRIPTS

Everything you need, to add high scores into your own game, is inside the *LB_Leaderboard.cs* script. This contains all of the functions you need to create, update, save and load high scores. It also contains a few functions to help you get to the information you need, like the 'GetHighscoreRank' and 'DidGetHighScore' functions.

LB_Leaderboard.cs doesn't do anything without you asking it to. It's just a collection of functions you'll need. In the example, it's the LB_*StandardHighScores.cs* script that deals with managing everything around the Leaderboard script.

The easiest way to use this script is to add it as a Component to an empty GameObject in your scene. Create a reference to the Leaderboard Component in your own script and access it via the reference. That's exactly how I did it in *LB_StandardHighScores.cs* when I made a variable like this:

```
// reference to our high score controller gameObject

public LB_Leaderboard _scoreManager;
```

I would suggest taking a look at the example project to see how this work in action.

LEADERBOARD.CS CLASS FUNCTIONS

Here is a list of the functions available within that *LB_Leaderboard.cs* script:

SETUPSCORES()

Before we do anything to the leaderboard, we need to make sure that everything is set up. By set up, I mean that we need to either create a new leaderboard or load leaderboard data in from disk. The SetUpScores() function takes care of this, checking to see if the leaderboard exists before we start. Call this function before you do anything else.

If you wanted to have more than one leaderboard, we got it covered. Simply supply the SetUpScores function with an index number, for each score board. There is no hard limit to how many score boards the lib supports, since it's just grabbing the data from different PlayerPrefs values.

Whenever you call SetUpScores with an index number, you are switching leaderboards to whichever one you index. Every function you call after that will act against *that* leaderboard and not any other. Remember that whenever you want to 'switch' between leaderboards, you will need to call 'SetUpScores (<which leaderboard index num>)'

RESETALLSCORES()

Resets all scores to default (rebuilds the leaderboard as if it were a new one).

GETNAMEAT(RANK)

This function returns the player name at the rank you pass in, as a string.

GETSCOREAT(RANK)

This function returns the score at the rank you pass in, as a string.

DIDGETHIGHSCORE(SCORE)

This function will return TRUE if the player achieved a high score suitable of making it on to the leaderboard, or FALSE when the score was not high enough. Use this as a 'quick check' and pass in the final score as an integer.

GETHIGHSCORERANK()

This function returns an integer containing the rank that the player has achieved on the leaderboard. Pass in the final score as an integer to this function to find out where the player ranks.

SUBMITLOCALSCORE(NAME, SCORE)

Use this function to commit a score to the leaderboard. If the player does not place (if the score is too low to rank on the board) this function will work just fine, but do nothing with the score entry. Pass in the player name as a string and the score as an integer. The SubmitLocalScore function will take care of the rest!

USING THE EXAMPLE LEADERBOARD SCENE IN YOUR OWN GAME

Possibly the easiest way to use this leaderboard pack is to use the included scene and customize it to how you would like it. Integration is a case of writing the final score to a PlayerPref and loading the leaderboard scene. Here are some step by step instructions to help make it happen:

1. Add everything into your project.

2. When your game ends, save out the final score to the finalScore PlayerPref like this:

   ```
   PlayerPrefs.SetInt("finalScore", finalScore);
   ```

   (where finalScore is replaced with whatever variable your game is using for score)

3. Add code to your game to jump to the ShowLeaderboard scene whenever you want to display the leaderboard (such as from your main menu or after the game over screen). You can load the scene with the scene manager like this:

   ```
   SceneManager.LoadScene("ShowLeaderboard");
   ```

   Note that if you get errors about the SceneManager you may need to add a using statement to the first line of your script, to tell Unity to access the SceneManager library:

   ```
   using UnityEngine.SceneManagement;
   ```

   (You can see this scene loading in action in the example project script named LB_MainMenuController.cs)

4. Open the script LB_StandardHighScores.cs and look at the top of the script for the two scene loading functions, GotoMainMenu() and PlayAgain(). You'll need to change the scene names to your own for the buttons to go to different scenes than those in the examples.

   For example, say your main game scene was called SpaceFace. When the user clicks the Play Again button on the leaderboard screen, you want the game to jump to the SpaceFace scene. To do that, first you would find PlayAgain() and find this line:

   ```
   SceneManager.LoadScene("Game");
   ```

   Then change it to

   ```
   SceneManager.LoadScene("SpaceFace");
   ```

IMPORTANT NOTE ABOUT SCENES:

Go to the menu **File > Build Settings** and check that the scenes are all inside the **Scenes In Build** section.

You need to make sure that all of the scenes are added to Build Settings, otherwise the game won't be able to find them.

EXAMPLE PROJECT JOYSTICK / CONTROLLER SUPPORT:

If you need the 'enter name' system to use joystick input instead of keyboard, open up the ShowLeaderboard scene and click on the Leaderboard_Controller GameObject in the Hierarchy. In the Inspector, check the 'Use Joystick Enter Name' checkbox. A joystick-based name input pop up will now be used instead of the keyboard one.

If you are using this example to call your own code whenever the name has been entered (rather than the submit code in the example project), you can tell the joystick code what to call when the user finishes entering their name. To do this, find the InputField_Joystick GameObject in the Hierarchy of the ShowLeaderboard scene. It has a Component attached to it called LB_JoystickTextInputController. On the LB_JoystickTextInputController Component, you can find an OnClick() Unityevent that you can set up to call whatever function on whatever GameObject you would like. In the Inspector, drag in your own GameObject into the target box and choose your script and function from the dropdown menu.

The default setup for my controller input code example is that it will let you enter in the letters with up/down and fire will advance to the next letter. To end text input, you press Fire2 (in Unity's input manager, Fire2 can be customized to be whatever button, key or mouse thing you want it to be). If you don't want to allow Fire2, take a look at the LB_JoystickTextInputController Component and find the useFire2ForSubmit checkbox. Obviously, check it on to enable fire2 or off to disable it. Also on that Component is another checkbox named autoSubmitOnMaxLength. When autoSubmitOnMaxLength is checked, it will automatically submit the name whenever the 'advance' button (fire button) is pressed and the name is at max length.

CHANGING THE RANKING TYPE - LOWEST SCORES FIRST OR HIGHEST SCORES FIRST?

It is possible to change the way your leaderboard works (ascending or descending) via the variable named lowestFirst in the LB_Leaderboard.cs script. If you are using the Leaderboard_Controller GameObject from the example, the Lowest First checkbox is accessible via the Inspector on Leaderboard_Controller. Check or uncheck it as you need and reset the leaderboard to make it work.

USING YOUR OWN CONTROLLER / A DIFFERENT INPUT METHOD

If you want to use some sort of custom input system that doesn't use my controller code, you can disable my controller code and talk directly to the text field on the joystick-based submit name popup. I've provided three functions to help you implement your own input control. You can talk to the script controlling the text field directly to add letters, delete or submit.

To disable my controller code on the submit name popup, find the InputField_Joystick GameObject in the Hierarchy of the ShowLeaderboard scene. It has a Component attached to it called LB_JoystickTextInputController. Find it in the Inspector and check the Skip Joystick Input box. This means that the leaderboard will show this joystick-based enter name popup, but it won't use my input code. Once my controller code is disabled, you can talk to the text field through your own code.

To talk to the text field from your own scripts, make a reference to this LB_JoystickTextInputController Component – something like:

```
public LB_JoystickTextInputController theInputController;
```

Then, use the following functions to talk to it:

SetLetterAndAdvance(string whichLetter); // < ---to add a letter to the text field

DeleteLetter(); // <---to delete a letter

FinishedInput(); // <---when you are done, this will call the event set up for OnClick

For my testing purposes, I used the following script. I thought I'd provide it here just to help you see how I had my script set up to talk to the JoystickTextInputController attached to the input field of the submit score popup. What I did to test was to make three buttons, one for add letter, one for delete letter and one to submit. Each button called one of the functions in this script, which in turn talked to the input field:

```
using UnityEngine;
using System.Collections;

public class TestOutsideInput : MonoBehaviour {

    public LB_JoystickTextInputController theInputController;

       public void TestAddLetter()
    {
        theInputController.SetLetterAndAdvance("F"); // F is just a test letter
    }

    public void TestDeleteLetter ()
    {
        theInputController.DeleteLetter();
    }

    public void TestDone ()
    {
        theInputController.FinishedInput();
    }
}
```

FINAL NOTE

I really hope that this pack helps you out in your project and that you have fun with it. I've tried to keep it simple so that it's easy to integrate. That said, if there's any trouble using it you can drop me an email talk@psychicparrot.com and I'll try to help out. I may not be able to do all of the work for you, but I'll try my best to help ☺

Have fun making games!

Jeff.