DTT
APPS WEB GAMES

SINCE 2010

Rankings

Documentation | 19-05-2022

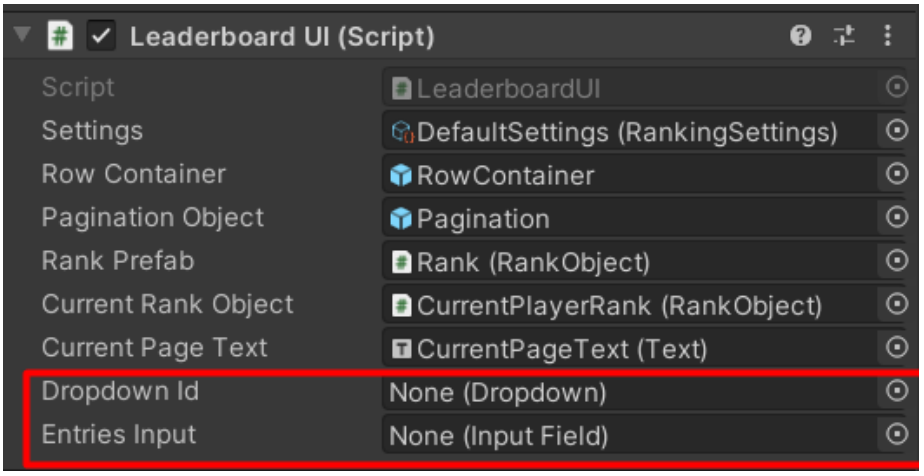
Table of Contents

1. Get started quickly

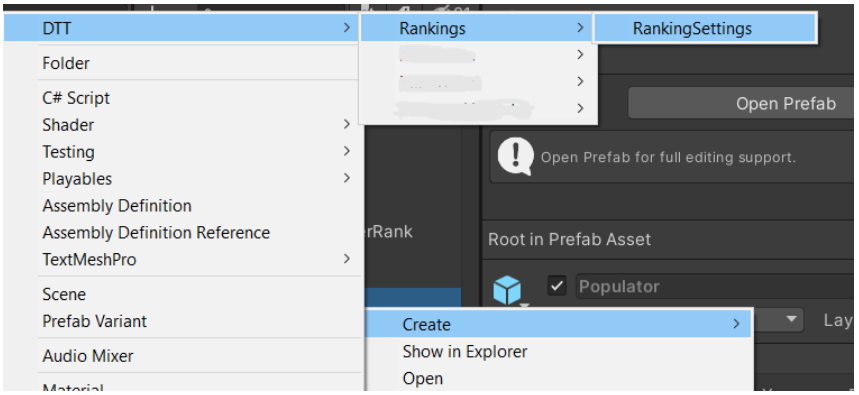
This is a summary to help you get started quickly. More details are provided below.

Open a scene, create a Canvas and add to it the RankingsUI prefab. It can be found in 'Demo/Prefab/'. All references should be correctly set by default.

- In case a reference is missing you can navigate to the LeaderboardUI component and add a reference to the ID Dropdown and the Populator InputField. Then change the OnClick() event on the three buttons (PopulateButton, RestartButton and ChooseIdButton), see Default setup for the correct methods.



Configure the RankingSettings scriptable object to fit your needs.
Or you can create a new one by right-clicking and selecting 'Create/DTT/Rankings/RankingSettings'.



The LeaderboardUI script is an example of a UI implementation. It fills the data file with demo Ranks by calling the AddRows method. You can get the rankings you want by accessing the Rankings, SortedRankings or AllRankings properties of the leaderboard.

```
/// <summary>
/// Fills the leaderboard with ranks.
/// </summary>
public void RefreshTable()
{
    ClearTable();
    _leaderboard.GetData(PATH_RANK);

    _currentPageText.text = $"1 / {_leaderboard.TotalPages}";

    foreach (Rank rank in _leaderboard.Rankings)
    {
        RankObject rankObject = Instantiate(_rankPrefab, _rowContainer.transform, false);
        rankObject.SetData(rank.Id, rank.RankPosition, rank.Name, rank.Score);
        _rankObjects.Add(rankObject);
    }

    FillDropdown();
}
```

- You can sort the data by calling SortData or SortDataDescending and passing the property you want, the sorted list, and the list you would like to sort.

```
/// <summary>
/// Sorts the ranks by name.
/// </summary>
public void SortByName()
{
    ClearTable();
    _leaderboard.SortData(x => x.Name, _leaderboard.SortedRankings, _leaderboard.Rankings);

    foreach (Rank rank in _leaderboard.SortedRankings)
    {
        RankObject rankObject = Instantiate(_rankPrefab, _rowContainer.transform, false);
        rankObject.SetData(rank.Id, rank.RankPosition, rank.Name, rank.Score);
        _rankObjects.Add(rankObject);
    }
}

/// <summary>
/// Sorts the rank by score.
/// </summary>
public void SortByScore()
{
    ClearTable();
    _leaderboard.SortDataDescending(x => x.Score, _leaderboard.SortedRankings, _leaderboard.Rankings);

    foreach (Rank rank in _leaderboard.SortedRankings)
    {
        RankObject rankObject = Instantiate(_rankPrefab, _rowContainer.transform, false);
        rankObject.SetData(rank.Id, rank.RankPosition, rank.Name, rank.Score);
        _rankObjects.Add(rankObject);
    }
}
```

- NextPage and PreviousPage allow you to go through the different pages of all rankings.

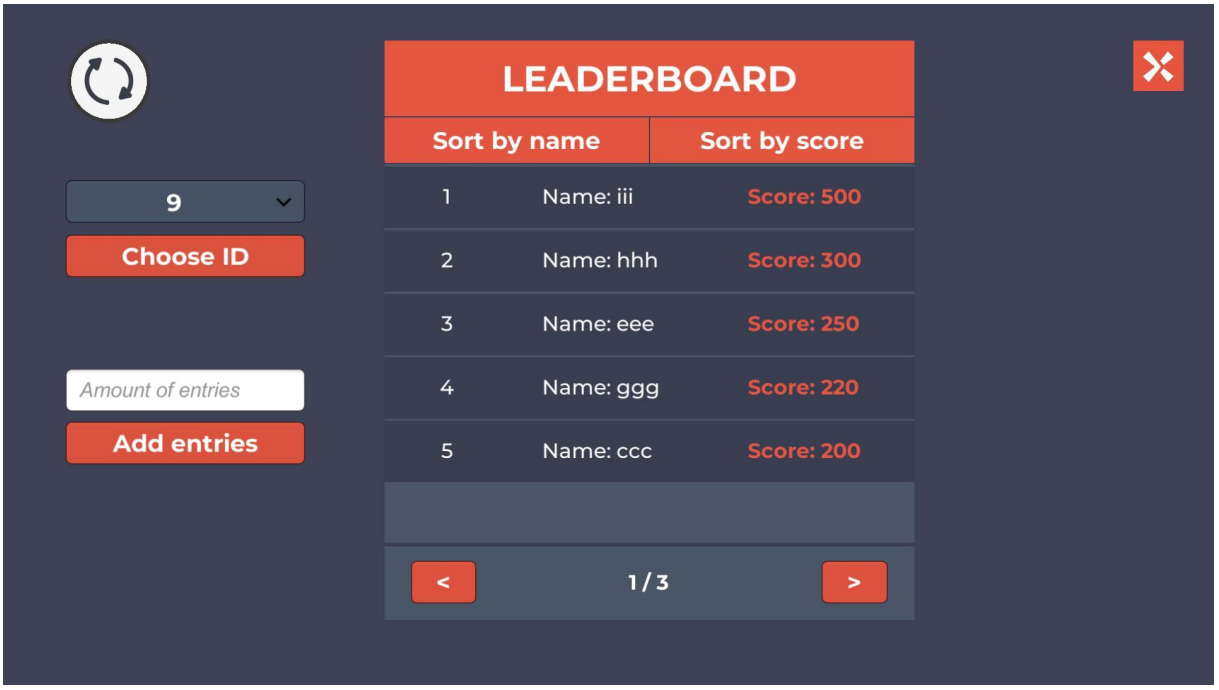
```
/// <summary>
/// Displays the next page.
/// </summary>
public void NextPage()
{
    _leaderboard.NextPage(_leaderboard.TotalPages);
    ClearTable();

    foreach (Rank rank in _leaderboard.Rankings)
    {
        RankObject rankObject = Instantiate(_rankPrefab, _rowContainer.transform, false);
        rankObject.SetData(rank.Id, rank.RankPosition, rank.Name, rank.Score);
        _rankObjects.Add(rankObject);
    }
}

/// <summary>
/// Displays the previous page.
/// </summary>
public void PreviousPage()
{
    _leaderboard.PreviousPage();
    ClearTable();

    foreach (Rank rank in _leaderboard.Rankings)
    {
        RankObject rankObject = Instantiate(_rankPrefab, _rowContainer.transform, false);
        rankObject.SetData(rank.Id, rank.RankPosition, rank.Name, rank.Score);
        _rankObjects.Add(rankObject);
    }
}
```

- ChooseID makes use of the GetCurrentPlayerRank method by passing an ID and the list with all rankings to get the Rank of that ID.
- PopulateBoard is used for demo purposes to fill the board with random entries.

```
/// <summary>
/// Populates the leaderboard with random data.
/// </summary>
/// <param name="amount">Amount of ranks.</param>
public void PopulateBoard()
{
    if (string.IsNullOrEmpty(_entriesInput.text))
        return;

    List<Rank> newRanks = new List<Rank>();
    int amount = Convert.ToInt32(_entriesInput.text);

    for (int i = 0; i < amount; i++)
    {
        Rank rank = new Rank(i, "user_" + i, Random.Range(0, 999));
        newRanks.Add(rank);
    }

    _leaderboard.AddRows(PATH_RANK, newRanks.ToArray());
}
```

2. Introduction

DTT Rankings is a Unity asset, which aims to provide you with the foundation needed to create your own leaderboard implementation. The asset includes a demo scene with UI classes, which gives you an example of how you can use the framework in your game. It includes pagination, sorting, saving and loading logic. The code is extendable, so you can use it for your own custom rankings and saving format.

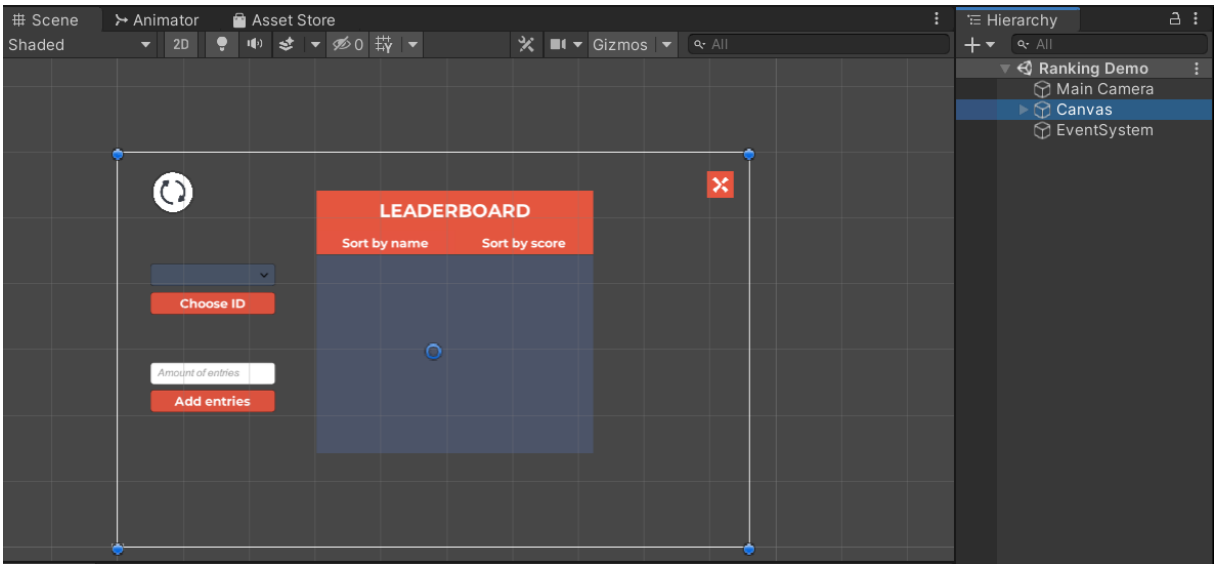


3. Set-Up

Default setup

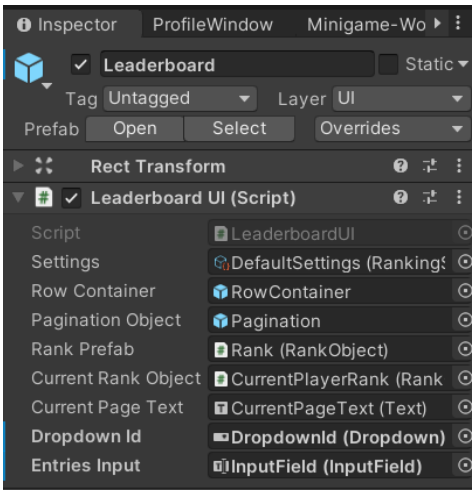
User Interface

Open the game demo scene located in *'Demo/Scenes/Ranking Demo'*. In the **Scene View** you can find an example of a visual representation for this package. In the **Hierarchy**, you will see a **Canvas** object. There you can find **RankingsUI** which contains several **GameObjects**: **ResetButton**, **IdChooser**, **Populator** and **Leaderboard**.



The most important object here is **Leaderboard**, it contains the **LeaderboardUI** component.

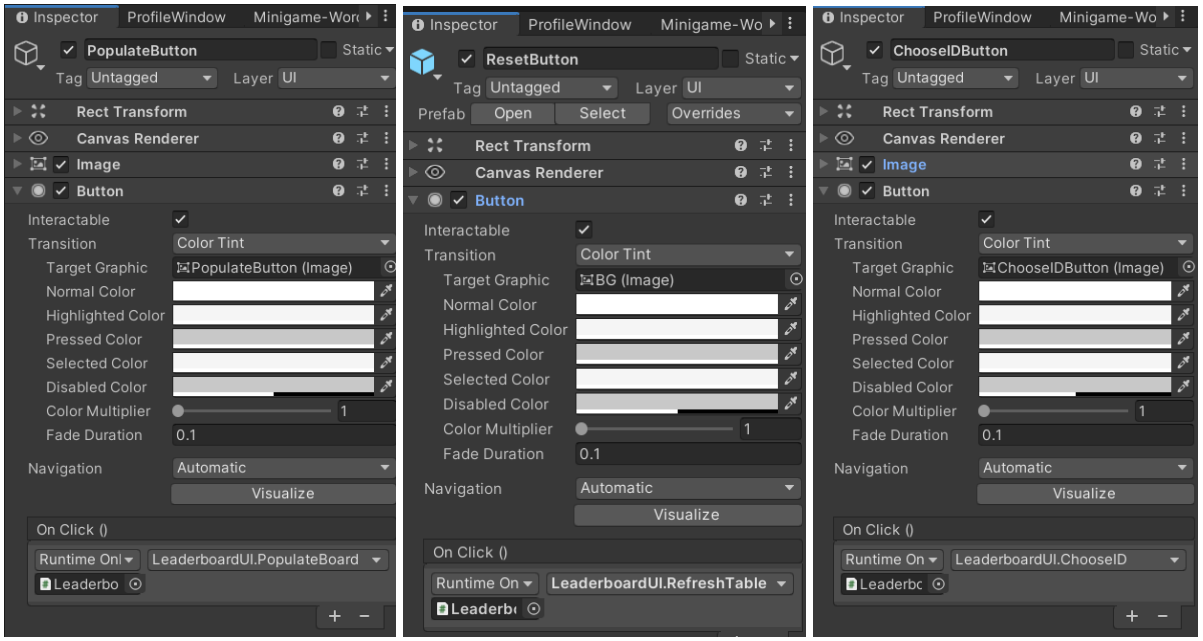
- **LeaderboardUI** is responsible for controlling and displaying the user interface and communicating with the **Leaderboard** class. It is configured by a **ScriptableObject** with settings. You can create a new one by clicking in the project windows and selecting *'Create/DTT/Rankings/RankingSettings/'*.



- The **Pagination** and **CurrentPlayerRank** objects are disabled by default and are only enabled based on the **RankingSettings**.

The other objects inside of the scene are interactable buttons and inputs.

- **ResetButton** calls **LeaderboardUI.RefreshTable** when clicked which starts/refreshes the leaderboard.
- **IdChooser** allows you to select an ID from the dropdown and then if the **HighlightCurrentPlayer** setting is enabled, it is going to display the rank with that ID separately from the rest. On click it calls the **LeaderboardUI.ChooseID** method.
- **Populator** allows you to fill the leaderboard with random entries for testing purposes. After adding them, you have to refresh the board to see them. It calls the **LeaderboardUI.PopulateBoard** method.



Logic

The default setup uses the Rank class for storing ranking information.

```
/// <summary>
/// Rank entity. Custom rankings must derive from this class.
/// </summary>
[Serializable]
public class Rank
{
    /// <summary>
    /// Rank ID.
    /// </summary>
    public int _id;

    /// <summary>
    /// Player rank.
    /// </summary>
    public int _rankPosition;

    /// <summary>
    /// Player name.
    /// </summary>
    public string _name;

    /// <summary>
    /// Final score.
    /// </summary>
    public int _score;

    /// <summary>
    /// Rank ID.
    /// </summary>
    public int Id => _id;

    /// <summary>
    /// Player rank.
    /// </summary>
    public int RankPosition => _rankPosition;

    /// <summary>
    /// Player name.
```

Leaderboard<Rank> handles API communication and controls the ranking collections. You can find an overview of its methods in 6. API.

```
/// <summary>
/// Initializes the leaderboard.
/// </summary>
/// <param name="maxRows">The maximum amount of ranks on the board.</param>
/// <param name="saver">Class for saving and loading ranks.</param>
public Leaderboard(int maxRows, IRankingSaver<T> saver)
{
    _maxRows = maxRows;
    _currentPage = 0;

    _rankings = new List<T>();
    _sortedRankings = new List<T>();
    _allRankings = new List<T>();

    _rankSaver = saver;

    callback += OnCallback;
}
```

RankingSaverJSON is a simple demo API for saving and loading data locally in a JSON format. It includes a wrapper class to allow array saving.

```
/// <summary>
/// Loads the data and calls the callback event.
/// </summary>
/// <param name="path">File location.</param>
public void Load(string path, Action<T[]> callback)
{
    if (File.Exists(path))
    {
        string content = File.ReadAllText(path);
        Wrapper<T> wrapper = JsonUtility.FromJson<Wrapper<T>>(content);
        T[] rows = wrapper.Items;

        callback?.Invoke(rows);
    }
    else
    {
        Debug.LogError("File doesn't exist.");
    }
}

/// <summary>
/// Saves the data.
/// </summary>
/// <param name="path">File location.</param>
/// <param name="rows">Collection with rows.</param>
public void Save(string path, T[] rows)
{
    Wrapper<T> wrapper = new Wrapper<T>();
    wrapper.Items = rows;
    string json = JsonUtility.ToJson(wrapper, true);

    File.WriteAllText(path, json);
}
```

Leaderboard loads the results by passing a callback event to RankingSaverJSON.Load and then listening to that event to fill the data.

```
/// <summary>
/// Event for loading data.
/// </summary>
public Action<T[]> callback;
```

```
/// <summary>
/// Gets the ranking data from a file.
/// </summary>
/// <param name="path">The file location.</param>
public void GetData(string path) => _rankSaver.Load(path, callback);
```

```
/// <summary>
/// Listens to the callback event and fills the list with rankings.
/// </summary>
/// <param name="data">The data to receive.</param>
public void OnCallback(T[] data)
{
    if (data == null)
        return;

    _currentPage = 0;
    _allRankings.Clear();
    _allRankings.AddRange(data.ToList());

    AssignRankPositions(true, x => x.Score, _allRankings);

    _rankings.Clear();
    _totalPages = (_allRankings.Count + _maxRows - 1) / _maxRows;

    for (int i = 0; i < _maxRows; i++)
    {
        if (i >= _allRankings.Count)
            return;

        _rankings.Add(_allRankings[i]);
    }
}
```

Custom Setup

1. Open a scene in Unity. Create a **GameObject** with a script for managing the UI of the rankings. Create a new class that derives from Rank for holding the different properties you would like each ranking to have.

```
/// <summary>
/// Custom ranking with a country field.
/// </summary>
[Serializable]
public class RankCountry : Rank
{
    /// <summary>
    /// Player country.
    /// </summary>
    public string _country;

    /// <summary>
    /// Player country.
    /// </summary>
    public string Country => _country;

    /// <summary>
    /// Initializes the rank with the given data.
    /// </summary>
    /// <param name="id">Rank ID.</param>
    /// <param name="name">Player name.</param>
    /// <param name="score">Player score.</param>
    /// <param name="country">Player country.</param>
    public RankCountry(int id, string name, int score, string country) : base(id, name, score) => _country = country;

    /// <summary>
    /// Returns a string with all rank details.
    /// </summary>
    public override string ToString() => $"ID: {Id} Rank: {RankPosition} Name: {Name} Score: {Score} Country: {_country}";
}
```

Then in the UI manager, add a new Leaderboard<T> where "T" is the Rank class you created.

```
/// <summary>
/// Leaderboard with ranks that have an additional country property.
/// </summary>
private Leaderboard<RankCountry> _leaderboard;
```

2. Now create or add your own API for saving and loading. The class should implement the **IRankingSaver<T>** interface, where **T** is your **Rank** class.

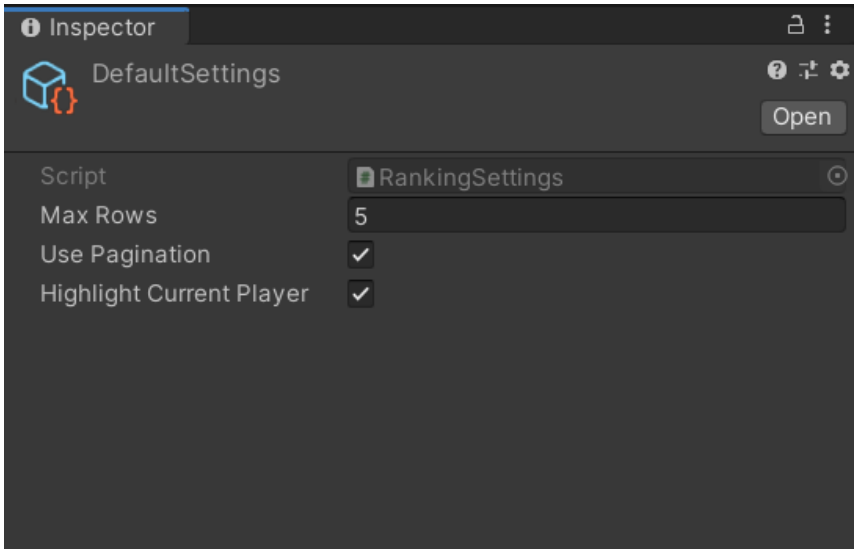
```
/// <summary>
/// Loads and saves rank data in JSON.
/// </summary>
/// <typeparam name="T">Class that derives from Rank.</typeparam>
public class RankingSaverJSON<T> : IRankingSaver<T> where T : Rank
{
    /// <summary>
    /// Loads the data and calls the callback event.
    /// </summary>
    /// <param name="path">File location.</param>
    public void Load(string path, Action<T[]> callback)...

    /// <summary>
    /// Saves the data.
    /// </summary>
    /// <param name="path">File location.</param>
    /// <param name="rows">Collection with rows.</param>
    public void Save(string path, T[] rows)...

    /// <summary>
    /// Wrapper allows arrays to be serialized.
    /// </summary>
    /// <typeparam name="TObject">the type of the object to be serialised</typeparam>
    [Serializable]
    private class Wrapper<TObject>...
}
```

3. Then initialize the **Leaderboard** by passing the amount of rows it should have on a page and your saver class as arguments to the constructor.

```
_leaderboard = new Leaderboard<RankCountry>(_settings.MaxRows, new RankingSaverJSON<RankCountry>());
```

4. You are now ready to use all the public methods that are found in **Leaderboard** to create your own implementation of a leaderboard. You can find an example of this in the **LeaderboardUI** script explained in 1. Get started quickly.

4. Editor



The RankingSettings **ScriptableObject** is used to configure the demo UI.

1. Max Rows

   The amount of ranks to fit in one page.

2. Use Pagination

   Whether the pagination object is enabled.

3. Highlight Current Player

   If selected, the rank of the player with the selected ID will be shown separately.

5. API

Leaderboard

Method name	Parameters	Description
GetData	string path	Used for retrieving data by calling the IRankSaver.Load method, passing a callback event.
OnCallback	T[] data	Listens to the callback event and fills the list with rankings.
AddRows	string path T[] rows	Used for saving data by calling the IRankSaver.Save method.
SortData	Func<T, IComparable> getProp List<T> sortedRankings List<T> rankingsToSort	Sorts data ascending by any given property.
SortDataDescending	Func<T, IComparable> getProp List<T> sortedRankings List<T> rankingsToSort	Sorts data descending by any given property.
GetCurrentPlayerRank	int id List<T> allRankings	Returns the Rank of the received ID.
NextPage	int totalPages	Increments the page counter and calls the Paginate method.
PreviousPage		Decrements the page counter and calls the Paginate method.

IRankingSaver

Method name	Parameters	Description
Load	string path Action<T[]> callback	Loads the data from the given path and calls the callback event with an array of rankings.
Save	string path T[] rows	Saves the data in a JSON format to the given path.

6. Known Limitations

- Does not inherently support remote API's.

7. Support and feedback

If you have any questions regarding the use of this asset, we are happy to help you out. Always feel free to contact us at:

unity-support@d-tt.nl

(We typically respond within 1-2 business days)

We are actively developing this asset, with many future updates and extensions already planned. We are eager to include feedback from our users in future updates, be they 'quality of life' improvements, new features, bug fixes or anything else that can help you improve your experience with this asset. You can reach us at the email above.

Reviews and ratings are very much appreciated as they help us raise awareness and to improve our assets.

DTT stands for Doing Things Together

DTT is an app, web and game development agency based in the centre of Amsterdam. Established in 2010, DTT has over a decade of experience in mobile, game, and web based technology.

Our game department primarily works in Unity where we put significant emphasis on the development of internal packages, allowing us to efficiently reuse code between projects. To support the Unity community, we are publishing a selection of our internal packages on the Asset Store, including this one.

More information about DTT (including our clients, projects and vacancies) can be found here:

https://www.d-tt.nl/en/