DOCUMENTATION

The code is documented only in core areas. No external code reference can be provided, since the package is pretty small at the moment

USAGE INSTRUCTIONS

`ClassicSRIA` and **`CAbstractViewsHolder`** are the 2 core classes in this small plugin dedicated to programmatically manage a ScrollRect's contents.

You can use it both for a horizontal or vertical ScrollView.

`ClassicSRIA i`t's an abstract, generic MonoBehaviour, which you need to extend and provide at least the implementation of **`ClassicSRIA.CreateItemViewHolder()`** and **`ClassicSRIA.UpdateItemViewHolder()`**.

It's recommended to manually go through example code provided in order to fully understand the mechanism. You'll find detailed comments in core areas. You may even use those scripts directly without implementing your own, in some simple scenarios, by stripping the demo code, but it's not recommended.

(Some may find it more easy to consult the example code, without reading this document)

IMPLEMENTATION

UI setup:

3 prefabs are provided, which only include the UI data (the code you need to implement is described below):

- VerticalListView (contains a classic ScrollView setup, with a ContentSizeFitter on the 'content' and a VerticalLayoutGroup)

- HorizontalListView (analogue, with a HorizontalLayoutGroup)

- VerticalGridView (analogue, with a GridLayout) - if you want a Horizontal GridView, it's similar to this. Look at how the HorizontalListView was made.

Code setup:

(Follow these steps while constantly looking at how it's done in the example code in ***`HorizontalClassicListView.cs`****)*

Here's the normal flow you'll follow after you've created a Scroll View using **`GameObject->UI->Scroll View:`**

1. create your own implementation of **`CAbstractViewsHolder`**, let's name it **`MyItemViewsHolder`**

2. create your own implementation of Classic**`SRIA<MyItemViewsHolder>`**, let's name it **`MyClassicSRIA`**

3. override **`Start(),`** call base.Start(), after which:

4. call **`MyClassicSRIA.ResetItems(int count)`** once (and any time your dataset is changed) and **`CreateViewsHolder(int)`** will be called for each view that needs creation.

- **`newOrRecycledViewsHolder.root`** will be null, so you need to instantiate your prefab), assign it and call **`newOrRecycledViewsHolder.CollectViews().`** Alternatively, you can call its **`AbstractViewHolder.Init(..)`** method, which can do a lot of things for you, mainly instantiate the prefab and (if you want) call **`CollectViews()`**

- after creation, only **`MyClassicSRIA.UpdateItemViewHolder()`** will be called for it when its represented item changes

- **`MyClassicSRIA.UpdateViewsHolder(MyItemViewsHolder)`** will be called when an item is to be displayed or simply needs updating:

- use **`newOrRecycledViewsHolder.ItemIndex`** to get the item index, so you can retrieve its associated model from your data set

- **`newOrRecycledViewsHolder.root`** is not null here (given the view holder was properly created in **`CreateViewsHolder(..)`**). It's assigned a valid object whose UI elements only need their values changed (common practice is to implement helper methods in the view holder that take the model and update the views themselves)

EXAMPLE SCENES & UTILITIES

All the example scenes & the utlity scripts are provided on an "as-is" base. Their main purpose is to demonstrate the feature-set and show you the basic code-flow when implementing the adapter, following the recommended best-practices & conventions.

KNOWN ISSUES & WORKAROUNDS

- Not actually related to the plugin itself, but worth mentioning: some lower-end devices have terrible performance with Open GL 3 and/or Auto Graphics API. If you experience oddly low FPS, untick Auto Graphics API and use Open GL 2 instead.