Super ScrollView for UGUI 2.4

Package Overview

In the SuperScrollView package, there are three main components: **LoopListView2**, **LoopGridView** and **LoopStaggeredGridView**. LoopListView2 is mainly for ListView, and LoopGridView is mainly for GridView that all the items have same size, and LoopStaggeredGridView is mainly for StaggeredGridView. StaggeredGridView is GridView that the items have different height for an vertical GridView (or different width for an horizontal GridView), such as www.**pinterest**.com looks like.

LoopListView2 and LoopGridView and LoopStaggeredGridView are components attaching to the same gameobject of UGUI ScrollRect. They help the UGUI ScrollRect to support any count items with high performance and memory-saving.

This document would firstly introduce the LoopListView2 component and then introduce the LoopGridView component and then LoopStaggeredGridView component.

LoopListView2 overview

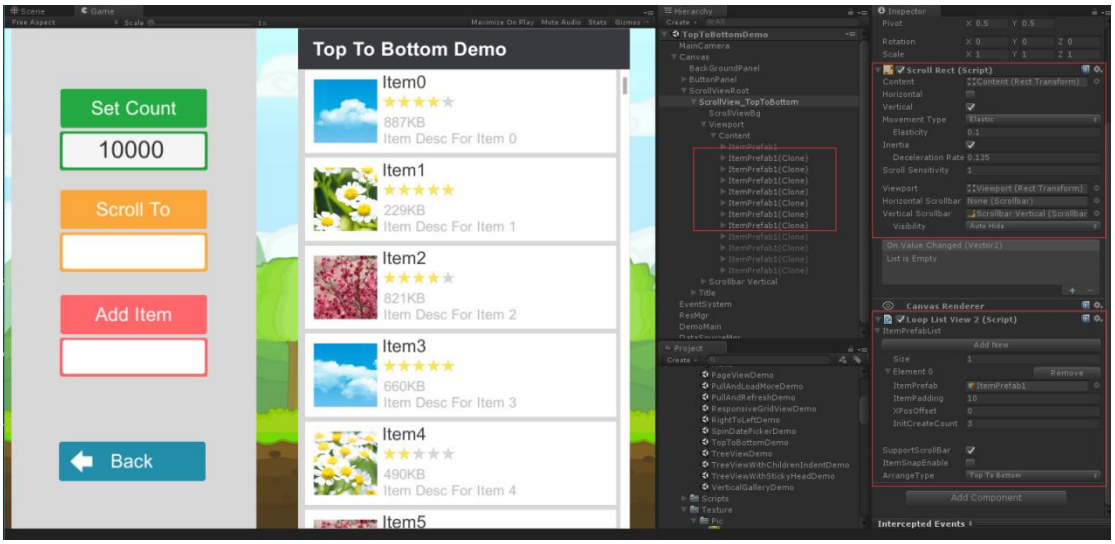
For a ScrollRect with 10,000 items, LoopListView2 does not really create 10,000 items, but create a few items based on the size of the viewport.

When the ScrollRect moving up, for example, the LoopListView2 component would check the topmost item's position, and once the topmost item is out of the viewport, then the LoopListView2 component would recycle the topmost item, and at the same time check the downmost item's position, and once the downmost item is near the bottom of the viewport , the LoopListView2 component would call the **onGetItemByIndex** handler to create a new item and then positon the new created item under the downmost item, so the new created item becomes the new downmost item.

Every item can use a different prefab and can have different height/width and padding.

There are several examples to help you learning the LoopListView2 component, in the folder with path : Assets -> SuperScrollView ->Demo-> Scenes -> ListView.

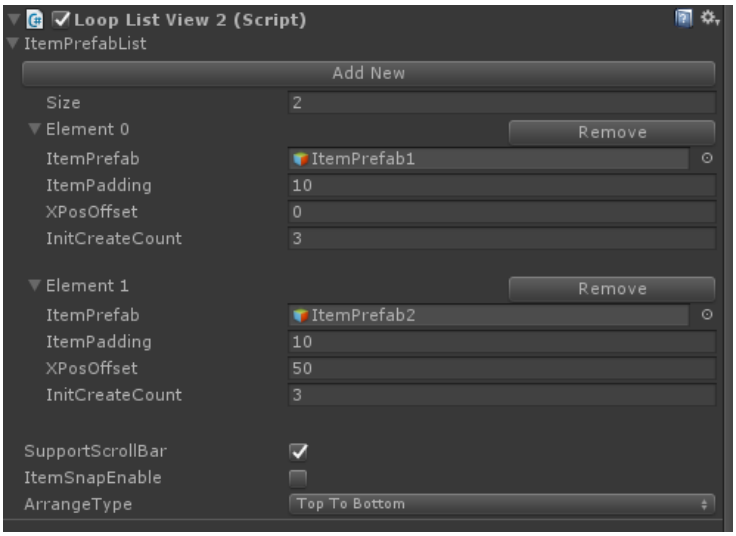
The following picture is what a TopToButtom arranged scrollrect looks like:



In the above picture, the scrollrect have 10000 items, but in fact, only 7 items really created.

LoopListView2 Inspector Settings

In the Inspector, to make sure the LoopListView2 component works well, there are several parameters need to set:



ItemPrefabList: this is the existing items to clone.

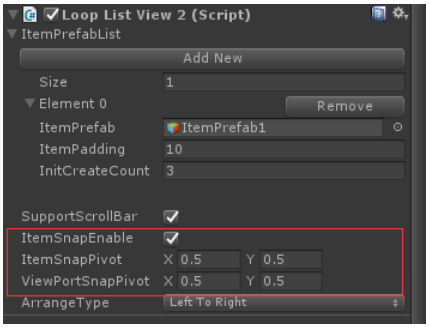
Every item can use a different prefab and every prefab can have different **default padding**(the amount of spacing between each item in the scrollrect). And also, every prefab can have different default x local pos(for **Vertical** scrollrect) or default y local pos(for **Horizontal** scrollrect), and here, is called (X/Y)PosOffset. Every prefab has a pool for getting and recycling action, and the **InitCreateCount** is the count created in pool at start.

In fact, in runtime, every item can have different padding and (X/Y)PosOffset, you can change an item's padding and (X/Y)PosOffset in your `onGetItemByIndex` callback. You can get/set them by LoopListViewItem2. Padding and LoopListViewItem2. StartPosOffset.

Important note: All the itemPrefab's localScale need to be (1,1,1).

SupportScrollbar: if checked, the LoopListView2 component would support scrollbar.

ItemSnapEnable:



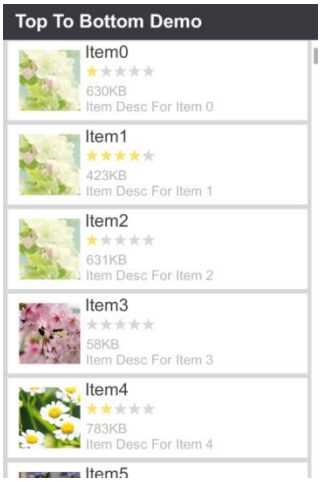
if checked, the LoopListView2 component would try to snap item to the configed location in viewport.

ItemSnapPivot is the location of the snap pivot point of the item, defined as a fraction of the size of the rectangle itself. 0,0 corresponds to the lower left corner while 1,1 corresponds to the upper right corner.

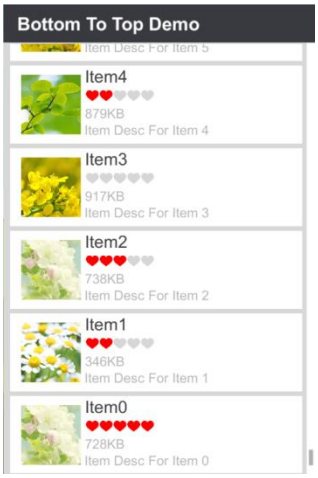
ViewPortSnapPivot is the location of the snap pivot point of the ScrollRect ViewPort, defined as a fraction of the size of the rectangle itself. 0,0 corresponds to the lower left corner while 1,1 corresponds to the upper right corner.

ArrangeType: the scroll direction, there are four types:

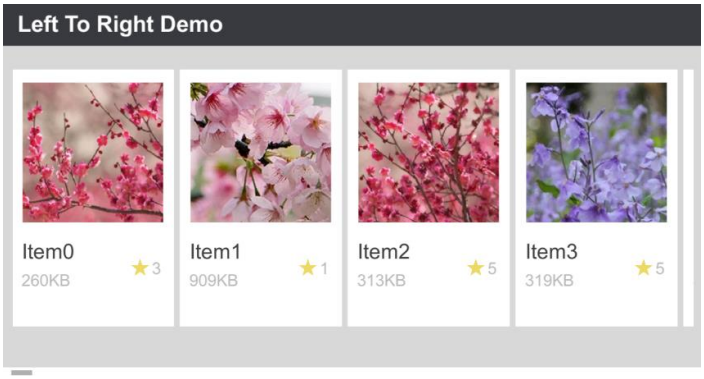
(1) **TopToBottom**: this type is used for a vertical scrollrect, and the item0,item1,…itemN are positioned one by one **from top to bottom** in the scrollrect viewport, just like the following:



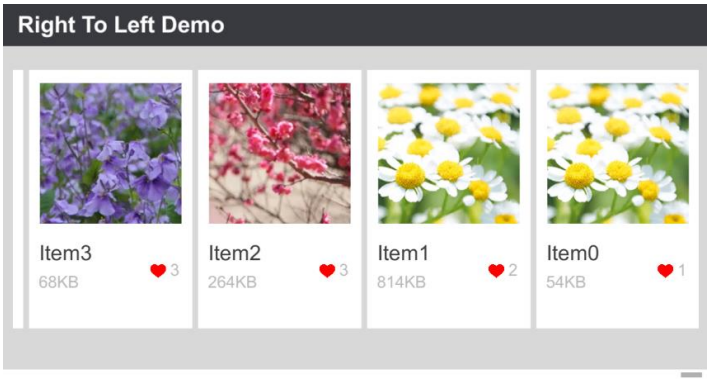
(2) **BottomToTop**: this type is used for a vertical scrollrect, and the item0,item1,…itemN are positioned one by one **from bottom to top** in the scrollrect viewport, just like the following:



(3) **LeftToRight**: this type is used for a horizontal scrollrect, and the item0,item1,…itemN are positioned one by one **from left to right** in the scrollrect viewport.



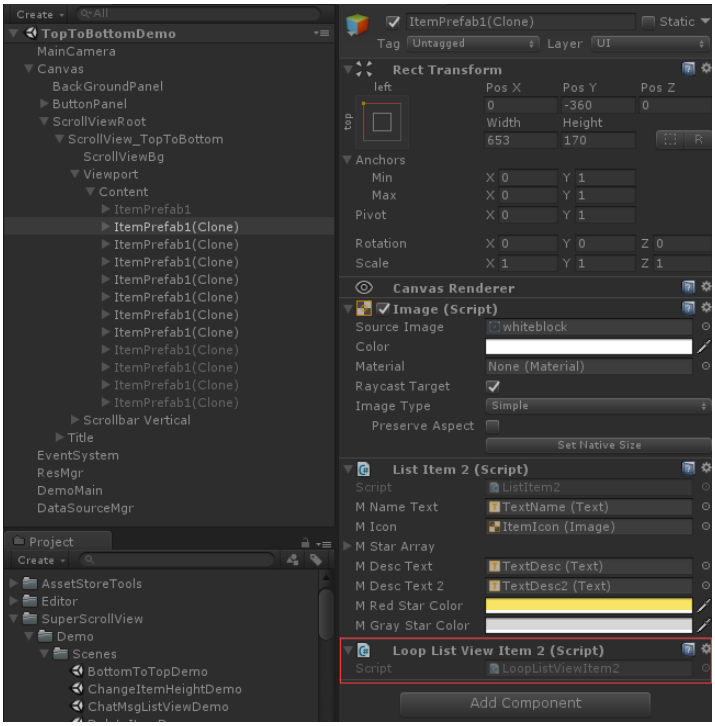
(4) **RightToLeft:** this type is used for a horizontal scrollrect, and the item0,item1,…itemN are positioned one by one **from right to left** in the scrollrect viewport.



LoopListView2 Important Public Method

```
public void InitListView(int itemTotalCount,
    System.Func<LoopListView2, int, LoopListViewItem2> onGetItemByIndex,
    LoopListViewInitParam initParam = null)
```

InitListView method is to initiate the LoopListView2 component. There are 3 parameters:
itemTotalCount: the total item count in the scrollview. If this parameter is set -1, then means there are infinite items, and scrollbar would not be supported, and the ItemIndex can be from **–MaxInt** to +**MaxInt**. If this parameter is set a value >=0 , then the ItemIndex can only be from 0 to **itemTotalCount -1**.
onGetItemByIndex: when an item is getting in the scrollrect viewport, this Action will be called with the item' index as a parameter, to let you create the item and update its content.
LoopListViewItem2 is the return value of onGetItemByIndex
Every created item has a LoopListViewItem2 component auto attached:



LoopListViewItem2 component is very sample:

```
public class LoopListViewItem2 : MonoBehaviour
{
    int mItemIndex = -1;
    int mItemId = -1;
    LoopListView2 mParentListView = null;
    bool mIsInitHandlerCalled = false;
    string mItemPrefabName;
    RectTransform mCachedRectTransform;
    float mPadding;
```

The mItemIndex property indicates the item's index in the list, as mentioned above, if **itemTotalCount** is set -1, then the **mItemIndex** can be from **–MaxInt** to +**MaxInt**. If

itemTotalCount is set a value >=0 , then the **mItemIndex** can only be from 0 to **itemTotalCount -1**.

The mItemId property indicates the item's id. This property is set when the item is created or fetched from pool, and will no longer change until the item is recycled back to pool.

The following codes is an example of **onGetItemByIndex**:

```
LoopListViewItem2 OnGetItemByIndex(LoopListView2 listView, int itemIndex)
{
    if (itemIndex < 0 || itemIndex >= DataSourceMgr.Get.TotalItemCount)
    {
        return null;
    }
    //get the data to showing
    ItemData itemData = DataSourceMgr.Get.GetItemDataByIndex(itemIndex);
    if(itemData == null)
    {
        return null;
    }
    /*get a new item. Every item can use a different prefab, the parameter of the NewListViewItem is
    the prefab' name.
    And all the prefabs should be listed in ItemPrefabList in LoopListView2 Inspector Setting */
    LoopListViewItem2 item = listView.NewListViewItem("ItemPrefab1");
    ListItem2 itemScript = item.gameObject.GetComponent<ListItem2>(); //get your own component
    // IsInitHandlerCalled is false means this item is new created but not fetched from pool.
    if (item.IsInitHandlerCalled == false)
    {
        item.IsInitHandlerCalled = true;
        itemScript.Init();// here to init the item, such as add button click event listener.
    }
    //update the item' s content for showing, such as image,text.
    itemScript.SetItemData(itemData);
    return item;
}
```

```
public LoopListViewItem2 NewListViewItem(string itemPrefabName)
```

This method is used to get a new item, and the new item is a clone from the prefab named itemPrefabName. This method is usually used in onGetItemByIndex.

```
public void SetListItemCount(int itemCount, bool resetPos = true)
```

This method may use to set the item total count of the scrollview at runtime. If this parameter is set -1, then means there are infinite items, and scrollbar would not be supported, and the ItemIndex can be from **–MaxInt** to +**MaxInt**. If this parameter is set a value >=0 , then the ItemIndex can only be from 0 to **itemTotalCount -1**.
If resetPos is set false, then the scrollrect's content position will not changed after this method finished.

```
public LoopListViewItem2 GetShownItemByItemIndex(int itemIndex)
```

To get the visible item by itemIndex. If the item is not visible, then this method return null.

```
public LoopListViewItem2 GetShownItemByIndex(int index)
```

All visible items is stored in a List<LoopListViewItem2> , which is named `mItemList` ; this method is to get the visible item by the index in visible items list. The parameter index is from 0 to `mItemList.Count`.

```
public int ShownItemCount
{
    get
    {
        return mItemList.Count;
    }
}
```

To get the total count of all visible items.

```
public void RefreshItemByItemIndex(int itemIndex)
```

To update a item by itemIndex. if the itemIndex-th item is not visible, then this method will do nothing. Otherwise this method will first call `onGetItemByIndex(itemIndex)` to get an updated item and then reposition all visible items' position.

```
public void RefreshAllShownItem()
```

This method will update all visible items.

```
public void MovePanelToItemIndex(int itemIndex, float offset)
```

This method will move the scrollrect content's position to (the positon of itemIndex-th **item** + offset), and **in current version the itemIndex is from 0 to MaxInt, offset is from 0 to scrollrect viewport size.**

```
public void OnItemSizeChanged(int itemIndex)
```

For a vertical scrollrect, when a visible item's height changed at runtime, then this method should be called to let the LoopListView2 component reposition all visible items' position.
For a horizontal scrollrect, when a visible item's width changed at runtime, then this method

should be called to let the LoopListView2 component reposition all visible items' position.

```
public void FinishSnapImmediately()
```

Snap move will finish at once.

```
public int CurSnapNearestItemIndex
```

Get the nearest item index with the viewport snap point.

```
public void SetSnapTargetItemIndex(int itemIndex)
```

Set the snap target item index. This method is used in PageViewDemo.

```
public void ClearSnapData()
```

Clear current snap target and then the LoopScrollView2 will auto snap to the CurSnapNearestItemIndex.

LoopGridView overview

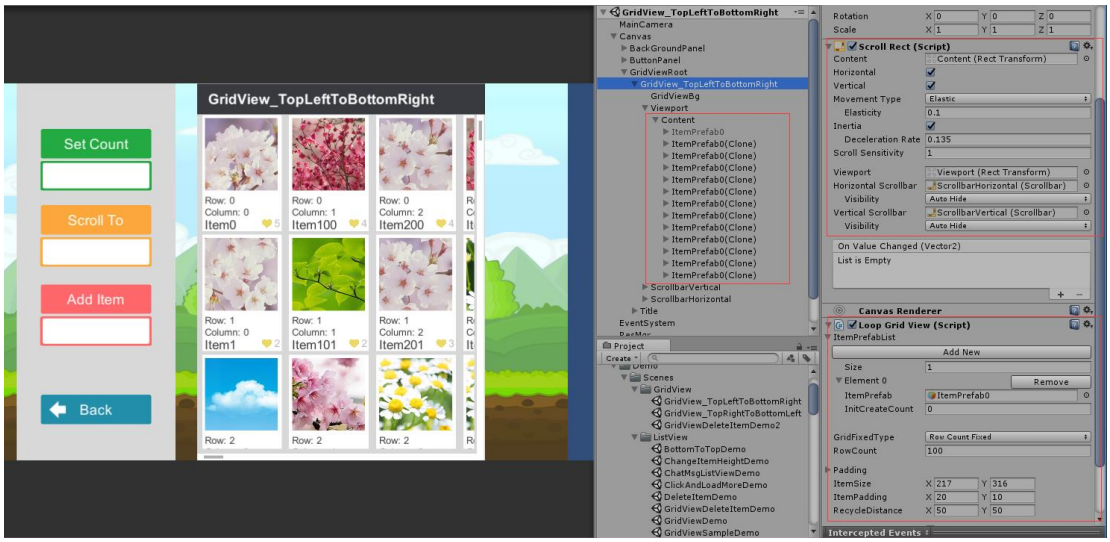
LoopGridView component is attaching to the same gameobject of UGUI ScrollRect, and is mainly for the GridView that all the items have same size, and has fixed count Row or fixed count Column. The ScrollRect can scroll horizontal and vertical at same time. For a GridView with 10,000 items, for example, 100 rows*100 columns, the LoopGridView does not really create 10,000 items, but create a few items based on the size of the viewport.

When the ScrollRect moving up, for example, the LoopGridView component would check the topmost row's position, and once the topmost row is out of the viewport, then the LoopGridView component would recycle all the items of the topmost row, and at the same time check the downmost row's position, and once the downmost row is near the bottom of the viewport , the LoopGridView component would call the **onGetItemByRowColumn** handler to create a new row and all the items in the viewport in the new row and then position the new created row under the downmost row, so the new created row becomes the new downmost row.

Every item can use a different prefab.

There are several examples to help you learning the LoopGridView component, in the folder with path : Assets -> SuperScrollView -> Demo -> Scenes -> GridView.

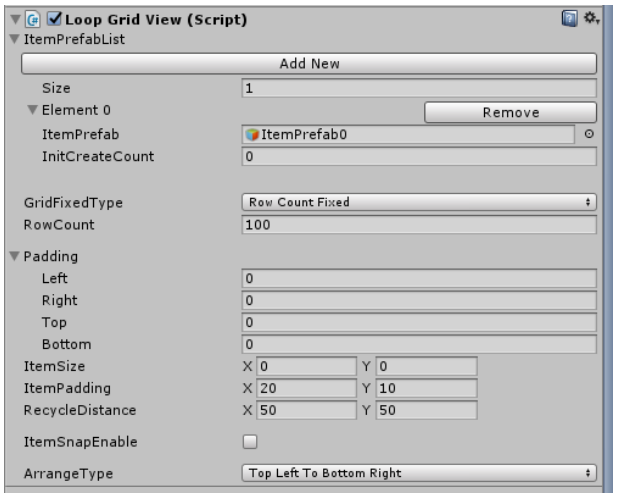
The following picture is what a TopLeft To BottomRight (100 rows * 100 columns) arranged ScrollRect looks like:



In the above picture, the GridView have 10000 items, but in fact, only a few items really created.

LoopGridView Inspector Settings

In the Inspector, to make sure the LoopGridView2 component works well, there are several parameters need to set:



ItemPrefabList: this is the existing items to clone.

Every item can use a different prefab. Every prefab has a pool for getting and recycling action, and the **InitCreateCount** is the count created in pool at start.

Important note: All the itemPrefab's localScale need to be (1,1,1). And the localScale of the content gameobject of the ScrollRect also need to be (1,1,1).

GridFixedType: the value can be ColumnCountFixed or RowCountFixed. This property is similar to the UGUI GridLayoutGroup.constraint and GridLayoutGroup.startAxis, that is which axis should items be placed along first.

RowCount/ColumnCount: if GridFixedType is ColumnCountFixed, then here you would set the column count of the GridView; if GridFixedType is RowCountFixed, then here you would set the row count of the GridView, This property is similar to the UGUI GridLayoutGroup. constraintCount.

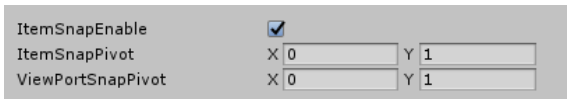
Padding: the space between the items and the viewport, this property is similar to the UGUI GridLayoutGroup.Padding.

ItemSize: the size of items. If the x or y is set to 0, then the ItemSize would be auto set the size of the first item of the **ItemPrefabList** when the LoopGridView component is inited.

ItemPadding: the amount of spacing between each item in the GridView.

RecycleDistance: how much distance an item leaves the viewport when the item would be recycled.

ItemSnapEnable:



if checked, the LoopGridView component would try to snap item to the configed location in the viewport.

ItemSnapPivot is the location of the snap pivot point of the item, defined as a fraction of the size of the rectangle itself. 0,0 corresponds to the lower left corner while 1,1 corresponds to the upper right corner.

ViewPortSnapPivot is the location of the snap pivot point of the ScrollRect ViewPort, defined as a fraction of the size of the rectangle itself. 0,0 corresponds to the lower left corner while 1,1 corresponds to the upper right corner.

ArrangeType: the scroll direction, there are four types:
TopLeftToBottomRight , BottomLeftToTopRight, TopRightToBottomLeft, BottomRightToTopLeft.

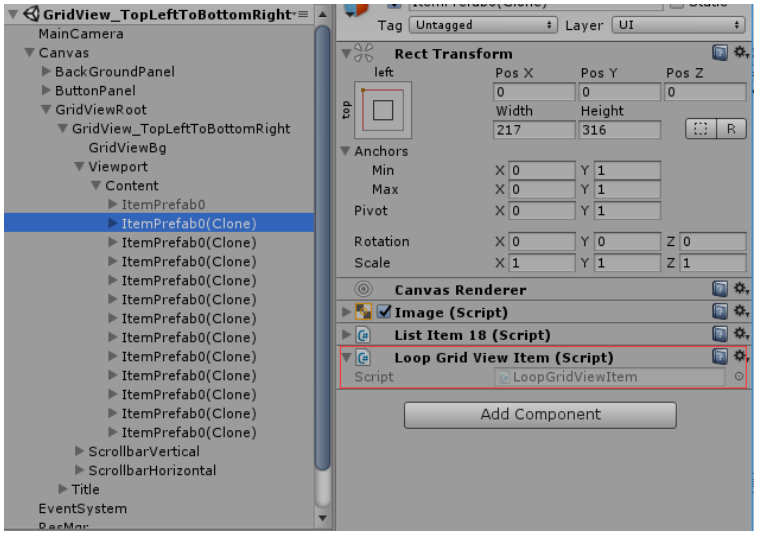
LoopGridView Important Public Method

```
public void InitGridView(int itemTotalCount,
    System.Func<LoopGridView, int, int, int, LoopGridViewItem> onGetItemByRowColumn,
    LoopGridViewSettingParam settingParam = null,
    LoopGridViewInitParam initParam = null)
```

InitGridView method is to initiate the LoopGridView component. There are 4 parameters:
itemTotalCount: the total item count in the GridView. This parameter must be set a value >=0, and the ItemIndex can be from 0 to **itemTotalCount -1**.
onGetItemByRowColumn: when an item is getting in the ScrollRect viewport, this action will be called with the item' (itmeIndex,row,column) as parameters, to let you create the item and update its content.
settingParam: this parameter is a LoopGridViewSettingParam object. You can use this parameter to override the values in the Inspector Setting, for example:

```
LoopGridViewSettingParam settingParam = new LoopGridViewSettingParam();
settingParam.mItemSize = new Vector2(500, 500);
settingParam.mItemPadding = new Vector2(40, 40);
settingParam.mPadding = new RectOffset(10, 20, 30, 40);
settingParam.mGridFixedType = GridFixedType.RowCountFixed;
settingParam.mFixedRowOrColumnCount = 6;
mLoopGridView.InitGridView(DataSourceMgr.Get.TotalItemCount, OnGetItemByIndex, settingParam);
```

LoopGridViewItem is the return value of onGetItemByRowColumn
Every created item has a LoopGridViewItem component auto attached:



LoopGridViewItem component is very sample:

```
public class LoopGridViewItem : MonoBehaviour
{
    // indicates the item's index in the list the mItemIndex can only be from 0 t
    int mItemIndex = -1;
    // the row index, the item is in. starting from 0.
    int mRow = -1;
    // the column index, the item is in. starting from 0.
    int mColumn = -1;
    //indicates the item's id.
    //This property is set when the item is created or fetched from pool,
    //and will no longer change until the item is recycled back to pool.
    int mItemId = -1;
    LoopGridView mParentGridView = null;
    bool mIsInitHandlerCalled = false;
    string mItemPrefabName;
    RectTransform mCachedRectTransform;
```

The mItemIndex property indicates the item's index in the GridView.

The mItemId property indicates the item's id. This property is set when the item is created or fetched from the pool, and will no longer change until the item is recycled back to the pool.

The following codes is an example of **onGetItemByRowColumn**:

```
LoopGridViewItem OnGetItemByRowColumn(LoopGridView gridView, int itemIndex,int row,int column)
{
    //get the data to showing
    ItemData itemData = DataSourceMgr.Get.GetItemDataByIndex(itemIndex);
    if (itemData == null)
    {
        return null;
    }
    /*
    get a new item. Every item can use a different prefab,
    the parameter of the NewListViewItem is the prefab' name.
    And all the prefabs should be listed in ItemPrefabList in LoopGridView Inspector Setting
    */
    LoopGridViewItem item = gridView.NewListViewItem("ItemPrefab0");
    ListItem18 itemScript = item.GetComponent<ListItem18>();//get your own component
    // IsInitHandlerCalled is false means this item is new created but not fetched from pool.
    if (item.IsInitHandlerCalled == false)
    {
        item.IsInitHandlerCalled = true;
        itemScript.Init();// here to init the item, such as add button click event listener.
    }
    //update the item' s content for showing, such as image,text.
    itemScript.SetItemData(itemData, itemIndex, row, column);
    return item;
}
```

```
public LoopGridViewItem NewListViewItem(string itemPrefabName)
```

This method is used to get a new item, and the new item is a clone from the prefab named

itemPrefabName. This method is usually used in onGetItemByRowColumn.

```
public void SetListItemCount(int itemCount, bool resetPos = true)
```

This method may use to set the item total count of the GridView at runtime. If resetPos is set false, then the ScrollRect's content position will not change after this method finished.

```
public LoopGridViewItem GetShownItemByItemIndex(int itemIndex)
```

To get the visible item by itemIndex. If the item is not visible, then this method return null.

```
public LoopGridViewItem GetShownItemByRowColumn(int row, int column)
```

To get the visible item by (row, column). If the item is not visible, then this method return null.

```
public void RefreshItemByItemIndex(int itemIndex)
```

To update an item by itemIndex. if the itemIndex-th item is not visible, then this method will do nothing. Otherwise this method will call onGetItemByRowColumn to get an updated item

```
public void RefreshItemByRowColumn(int row,int column)
```

To update an item by (row, column). if the item is not visible, then this method will do nothing. Otherwise this method will call onGetItemByRowColumn to get an updated item

```
public void RefreshAllShownItem()
```

This method will update all visible items.

```
public void MovePanelToItemByIndex(int itemIndex, float offsetX = 0, float offsetY = 0)
```

This method will move the ScrollRect content's position to (the positon of itemIndex-th **item** + offset).

```
public void MovePanelToItemByRowColumn(int row,int column, float offsetX = 0,float offsetY = 0)
```

This method will move the panel's position to (the position of (row,column) item + offset).

```
public void SetGridFixedGroupCount(GridFixedType fixedType,int count)
```

Change the GridFixedType and fixed row/column count at runtime.
You can also change the itemSize,itemPadding,contentPadding at runtime by SetItemSize(Vector2), SetItemPadding(Vector2), SetPadding(RectOffset).

```
public void FinishSnapImmediately()
```

Snap move will finish at once.

```
public RowColumnPair CurSnapNearestItemRowColumn
```

Get the nearest item' row and column with the viewport snap point.

```
public void SetSnapTargetItemRowColumn(int row, int column)
```

Set the snap target item' row and column.

```
public void ClearSnapData()
```

Clear current snap target and then the LoopGridView will auto snap to the CurSnapNearestItemRowColumn.

LoopStaggeredGridView overview

LoopStaggeredGridView is mainly for StaggeredGridView that the items have different height for an vertical GridView (or different width for an horizontal GridView). So if all the items of a GridView have same size, then you had better use LoopGridView.

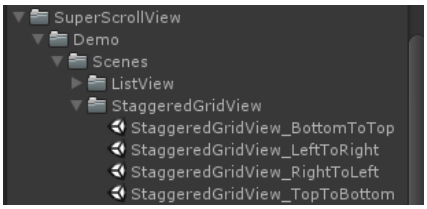
LoopStaggeredGridView is similar to the LoopListView2, for a ScrollRect with 10,000 items, LoopStaggeredGridView does not really create 10,000 items, but create a few items based on the size of the viewport.

When the ScrollRect moving up, for an vertical GridView, the LoopStaggeredGridView component would check the topmost item's position of every column, and once the topmost item of a column is out of the viewport, then the LoopStaggeredGridView component would recycle the topmost item.

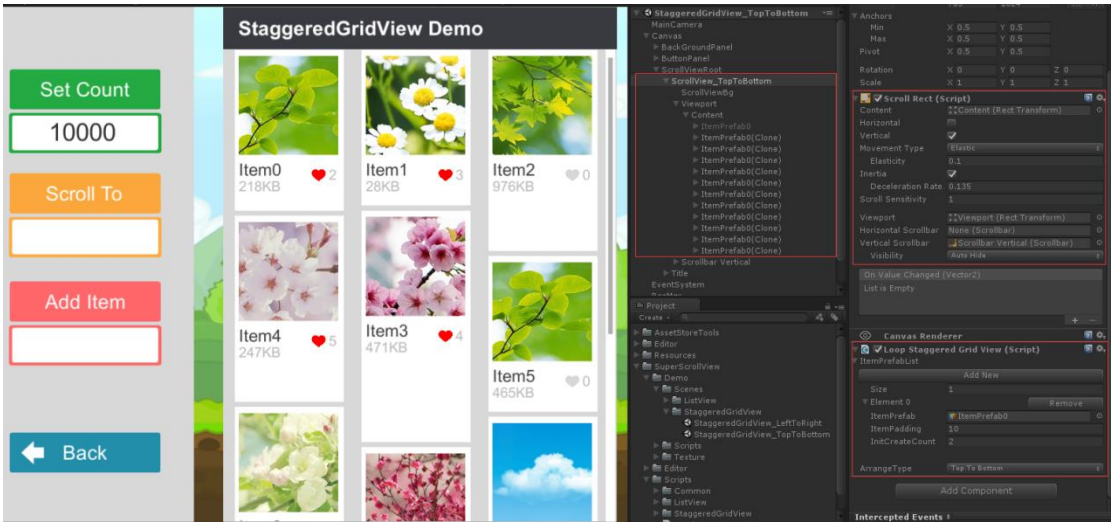
And at the same time check the downmost item's position of every column, and once the downmost item of a column is near the bottom of the viewport , the LoopStaggeredGridView component would call the **onGetItemByIndex** handler to create a new item and then **positon the new created item under the downmost item of the shortest column**, that is the column whose downmost item's bottom position is the top most position in all the columns' downmost positon. In the whole, all the items are arranged from left to right, from top to bottom.

Every item can use a different prefab. But for an vertical GridView, items can have different height but the width must be same. For an horizontal GridView, items can have different width but the height must be same.

There are two examples to help you learning the LoopStaggeredGridView component, in the folder with path : Assets -> SuperScrollView -> Demo -> Scenes -> StaggeredGridView.



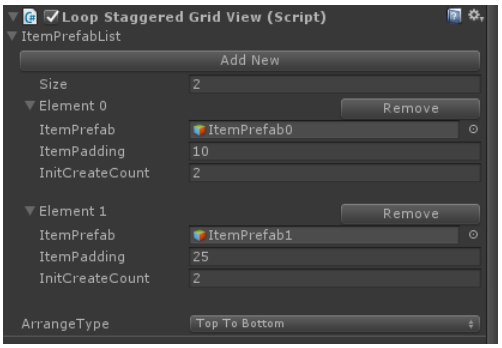
The following picture is what a TopToButtom arranged StaggeredGridView with 3 columns looks like:



In the above picture, the scrollrect have 10,000 items, but in fact, only 11 items really created.

LoopStaggeredGridView Inspector Settings

In the Inspector, to make sure the LoopStaggeredGridView component works well, there are several parameters need to set:



ItemPrefabList: this is the existing items to clone.
Every item can use a different prefab and every prefab can have different **default padding**(the amount of spacing between each item in the scrollrect). Every prefab has a pool for getting and recycling action, and the **InitCreateCount** is the count created in pool at start.
In fact, in runtime, every item can have different padding, you can change an item's padding in your `onGetItemByIndex` callback. You can get/set them by LoopStaggeredGridViewItem.Padding
Important note: All the itemPrefab's localScale need to be (1,1,1).

ArrangeType: the scroll direction, this is same to ListView. there are four types: **TopToBottom, BottomToTop, LeftToRight, RightToLeft**.

LoopStaggeredGridView Important Public Method

```
public void InitListView(int itemTotalCount, GridViewLayoutParam layoutParam,
    System.Func<LoopStaggeredGridView, int, LoopStaggeredGridViewItem> onGetItemByItemIndex,
    StaggeredGridViewInitParam initParam = null)
```

InitListView method is to initiate the LoopStaggeredGridView component. There are 4 parameters:
itemTotalCount: the total item count in the scrollview, this parameter should be >=0.

```
public class GridViewLayoutParam
{
    public int mColumnOrRowCount = 0;
    public float mItemWidthOrHeight = 0;
    public float mPadding1 = 0;
    public float mPadding2 = 0;
    public float[] mCustomColumnOrRowOffsetArray = null;
```

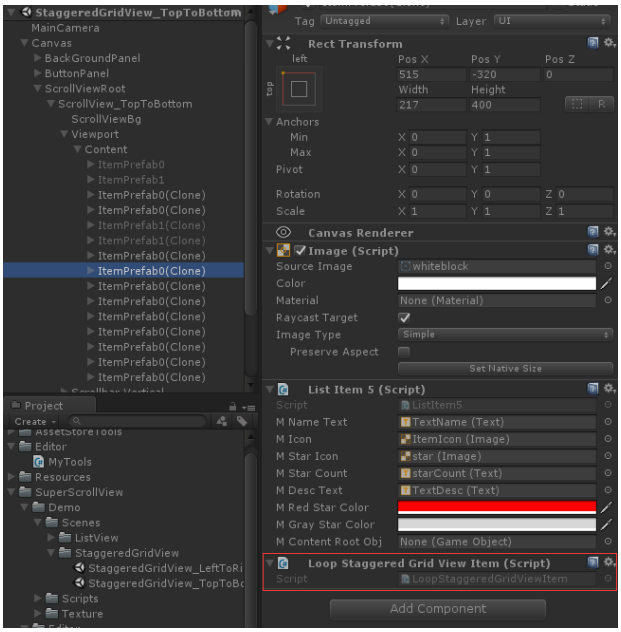
layoutParam: this class is very sample, and you need new a GridViewLayoutParam instance and set the values you want. For an vertical GridView, mColumnOrRowCount is the column count,

mItemWidthOrHeight is the item's width, mPadding1 is the viewport left margin, mPadding2 is the viewport right margin. For an horizontal GridView, mColumnOrRowCount is the row count, mItemWidthOrHeight is the item's height, mPadding1 is the viewport top margin, mPadding2 is the viewport bottom margin. If mCustomColumnOrRowOffsetArray is null, that is to say, you do not set value for this parameter, then the GiriView would arrange all the columns or rows averaged. If mCustomColumnOrRowOffsetArray is not null, the values of the array is the XOffset/YOffset of each column/row, and mCustomColumnOrRowOffsetArray.length must be same to mColumnOrRowCount.

onGetItemByItemIndex: when an item is getting in the scrollrect viewport, this Action will be called with the item' index as a parameter, to let you create the item and update its content.

LoopStaggeredGridViewItem is the return value of onGetItemByItemIndex

Every created item has a LoopStaggeredGridViewItem component auto attached:



LoopStaggeredGridViewItem component is very sample:

```
public class LoopStaggeredGridViewItem : MonoBehaviour
{
    int mItemIndex = -1;
    int mItemIndexInGroup = -1;
    int mItemId = -1;
    float mPadding;
    bool mIsInitHandlerCalled = false;
    string mItemPrefabName;
```

The mItemIndex property indicates the item's index in the GridView, can be from 0 to **itemTotalCount -1**.

The mItemIndexInGroup property indicates the item's index in a column or row. Here the word "Group" means: for an vertical GridView, a group means a column and for an horizontal GridView, a group means a row.

The mItemId property indicates the item's id. This property is set when the item is created or fetched from pool, and will no longer change until the item is recycled back to pool.

The following codes is an example of **onGetItemByItemIndex**:

```
LoopStaggeredGridViewItem OnGetItemByItemIndex(LoopStaggeredGridView gridView, int itemIndex)
{
    if (itemIndex < 0 || itemIndex >= DataSourceMgr.Get.TotalItemCount)
    {
        return null;
    }
    //get the data to showing
    ItemData itemData = DataSourceMgr.Get.GetItemDataByIndex(itemIndex);
    if(itemData == null)
    {
        return null;
    }
    /*get a new item. Every item can use a different prefab, the parameter of the NewListViewItem is
    the prefab' name.
    All the prefabs should be listed in ItemPrefabList in LoopStaggeredGridView Inspector Setting */
    LoopStaggeredGridViewItem item = gridView.NewListViewItem("ItemPrefab1");
    ListItem2 itemScript = item.gameObject.GetComponent<ListItem2>(); //get your own component
    // IsInitHandlerCalled is false means this item is new created but not fetched from pool.
    if (item.IsInitHandlerCalled == false)
    {
        item.IsInitHandlerCalled = true;
        itemScript.Init();// here to init the item, such as add button click event listener.
    }
    //update the item' s content for showing, such as image,text.
    itemScript.SetItemData(itemData);
    return item;
}
```

```
public LoopStaggeredGridViewItem NewListViewItem(string itemPrefabName)
```

This method is used to get a new item, and the new item is a clone from the prefab named itemPrefabName. This method is usually used in onGetItemByItemIndex.

```
public void SetListItemCount(int itemCount, bool resetPos = true)
```

This method may use to set the item total count of the GridView at runtime. If resetPos is set false, then the scrollrect's content position will not changed after this method finished.

```
public LoopStaggeredGridViewItem GetShownItemByItemIndex(int itemIndex)
```

To get the visible item by itemIndex. If the item is not visible, then this method return null.

```
public void RefreshItemByItemIndex(int itemIndex)
```

To update a item by itemIndex. if the itemIndex-th item is not visible, then this method will do nothing. Otherwise this method will first call `onGetItemByItemIndex(itemIndex)` to get an updated item and then reposition all visible items' position of the same group (that is the same column / row).

```
public void RefreshAllShownItem()
```

This method will update all visible items.

```
public void MovePanelToItemIndex(int itemIndex, float offset)
```

This method will move the scrollrect content's position to (the positon of itemIndex-th **item** + offset)

```
public void OnItemSizeChanged(int itemIndex)
```

For a vertical scrollrect, when a visible item's height changed at runtime, then this method should be called to let the LoopStaggeredGridView component reposition all visible items' position of the same group (that is the same column / row).
For a horizontal scrollrect, when a visible item's width changed at runtime, then this method should be called to let the LoopStaggeredGridView component reposition all visible items' position of the same group (that is the same column / row).