Super SCROLL VIEW For UGUI

Contents

1. Package Overview

In the UGUI Super ScrollView package, there are three main components: **LoopListView2**, **LoopGridView** and **LoopStaggeredGridView**.

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

This package has 88 demoes, these demos use one of these three components.

1.1 List View

List View uses the LoopListView2 component.

1) List View Top To Bottom Demo
2) List View Left To Right Demo
3) List View Bottom To Top Demo
4) List View Right To Left Demo
5) List View Select Delete Demo
6) List View Pull Down Refresh Demo
7) List View Pull Up Load More Demo
8) List View Click Load More Demo
9) List View Filter Demo
10) List View Expand Demo
11) List View Content Fitter Demo
12) List View Multiple Prefab Top To Bottom Demo
13) List View Multiple Prefab Left To Right Demo
14) List View Simple Top To Bottom Demo
15) List View Simple Left To Right Demo
16) List View Simple Load More Demo
17) List View Simple Loop Top To Bottom Demo
18) List View Simple Loop Left To Right Demo

1.2 Grid View

Grid View uses the the LoopGridView component.

1) Grid View Top To Bottom Demo

2) Grid View Left To Right Demo
3) Grid View Bottom To Top Demo
4) Grid View Right To Left Demo
5) Grid View Click Load More Demo
6) Grid View Select Delete Demo
7) Grid View Diagonal Top Left Demo
8) Grid View Diagonal Bottom Right Demo
9) Grid View Diagonal Select Delete Demo
10) Grid View Multiple Prefab Top To Bottom Demo
11) Grid View Multiple Prefab Left To Right Demo
12) Grid View Simple Top To Bottom Demo
13) Grid View Simple Left To Right Demo
14) Grid View Simple Filter Demo
15) Grid View Simple Diagonal Top Right Demo
16) Grid View Simple Diagonal Bottom Left Demo

1.3 Staggered View

Staggered View uses the LoopStaggeredGridView component. Staggered View is a Grid View that the items have different height for a vertical Grid View, or different width for a horizontal Grid View, such as 'www.pinterest.com' looks like.

1) Staggered View Top To Bottom Demo
2) Staggered View Left To Right Demo
3) Staggered View Bottom To Top Demo
4) Staggered View Right To Left Demo
5) Staggered View Simple Top To Bottom Demo
6) Staggered View Simple Left To Right Demo

1.4 Special Grid View

Special Grid View uses the LoopListView2 component.

1) Special Grid View Top To Bottom Demo
2) Special Grid View Left To Right Demo
3) Special Grid View Select Delete Demo
4) Special Grid View Pull Down Refresh Demo
5) Special Grid View Pull Up Load More Demo
6) Special Grid View Feature Top To Bottom Demo
7) Special Grid View Feature Left To Right Demo
8) Special Grid View Simple Top To Bottom Demo
9) Special Grid View Simple Left To Right Demo

1.5 Responsive View

Responsive View uses the LoopListView2 component.

1) Responsive View Demo
2) Responsive View Refresh Load Demo

1.6 Chat View

Chat View uses the LoopListView2 component.

1) Chat View Demo
2) Chat View Change Viewport Height Demo

1.7 Gallery

Gallery uses the LoopListView2 component.

1) Gallery Horizontal Demo
2) Gallery Vertical Demo

1.8 Tree View

Tree View uses the LoopListView2 component.

1) Tree View Demo
2) Tree View With Child Indent Demo
3) Tree View With Sticky Head Demo
4) Tree View Simple Demo

1.9 Page View

Page View uses the LoopListView2 component.

1) Page View Demo
2) Page View Simple Demo

1.10 Spin View

Spin View uses the LoopListView2 component.

1) Spin Date Picker Demo
2) Spin Time Picker Demo
3) Spin Date Time Picker Demo

1.11 Nested View

Nested View uses the LoopListView2 or the LoopGridView component.

1) Nested Top To Bottom Demo
2) Nested Left To Right Demo
3) Nested Grid View Top To Bottom Demo
4) Nested Grid View Left To Right Demo
5) Nested Simple List View Demo
6) Nested Simple Grid View Demo
7) Nested Simple Special Grid View Demo

1.12 List View Animation

List View Animation uses the LoopListView2 component.

1) List View Add Clip Animation Demo
2) List View Add Fade Animation Demo
3) List View Add Clip Fade Animation Demo
4) List View Add Slide Left Animation Demo
5) List View Add Slide Right Animation Demo
6) List View Delete Clip Animation Demo
7) List View Delete Fade Animation Demo
8) List View Delete Clip Fade Animation Demo
9) List View Delete Slide Left Animation Demo
10) List View Delete Slide Right Animation Demo
11) List View Expand Clip Animation Demo
12) List View Expand Fade Animation Demo
13) List View Expand Clip Fade Animation Demo

1.13 Draggable View

Draggable View uses the LoopListView2 component.

1) Draggable View Fade Top To Bottom Demo
2) Draggable View Fade Left To Right Demo
3) Draggable View Top To Bottom Demo
4) Draggable View Left To Right Demo

2. LoopListView2

2.1 LoopListView2 overview

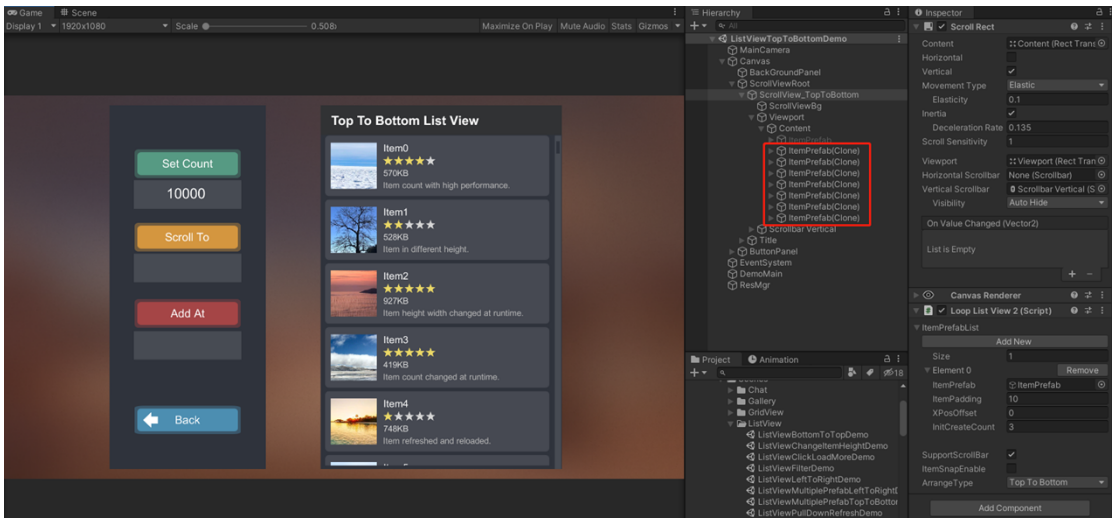
For a ScrollRect with 10,000 items, LoopListView2 does not really create 10,000 items, but create a few items based on the size of the viewport.

When the ScrollRect moving up, 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.

At the same time, it checks the bottommost item's position, and once the bottommost 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 bottommost item, so the new created item becomes the new bottommost item.

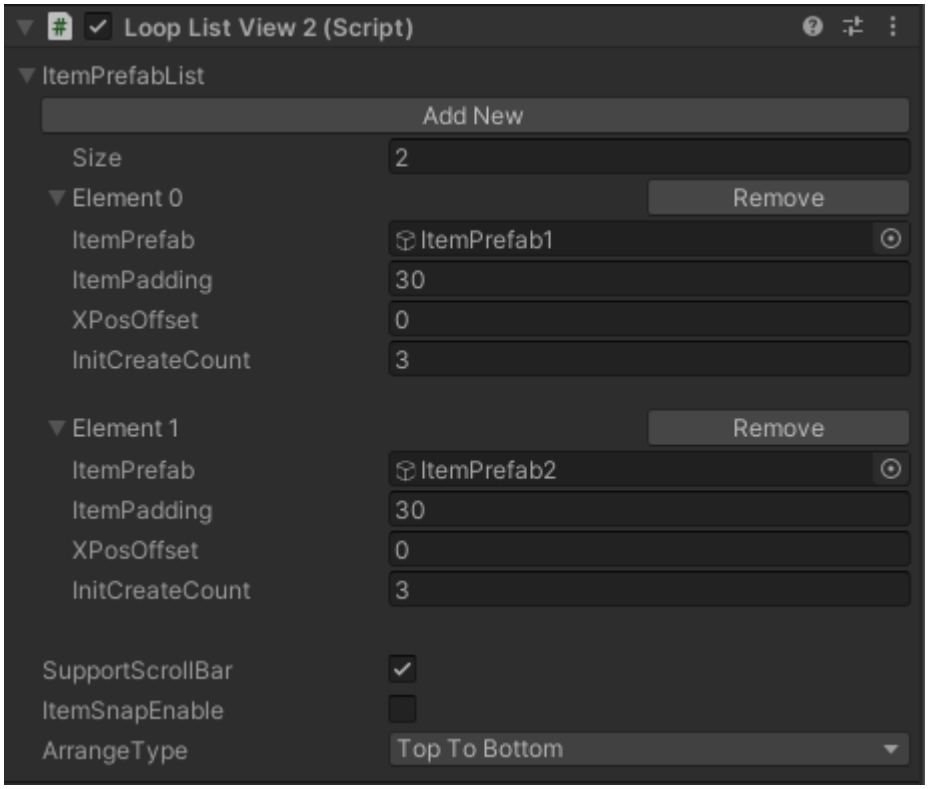
Every item can use a different prefab and can have different height/width and padding.

The ScrollRect have 10000 items, but in fact, only 7 items really created. The following figure is what a TopToBottom arranged ScrollRect looks like:



2.2 LoopListView2 Inspector Settings

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



1) ItemPrefabList

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 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.

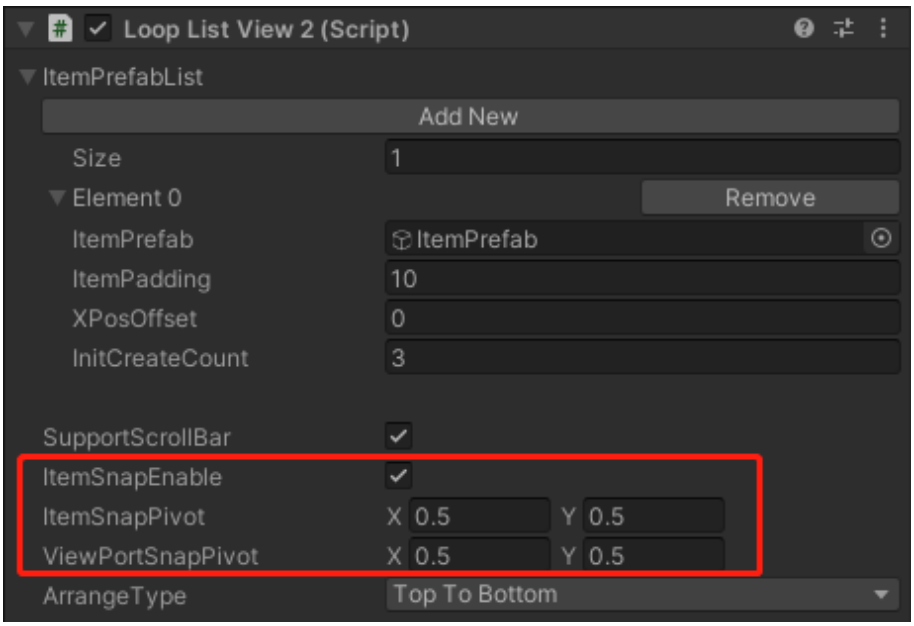
Important note: All the ItemPrefab's localScale need to be (1,1,1).

2) SupportScrollbar

If it is checked, the LoopListView2 component would support scrollbar.

3) ItemSnapEnable:

If it is checked, the LoopListView2 component would try to snap item to the configured location in viewport.



4) ItemSnapPivot

It 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.

5) ViewPortSnapPivot

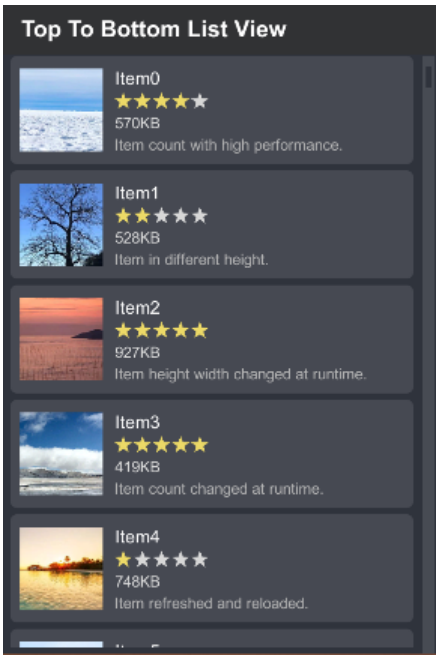
It is the location of the snap pivot point of the ScrollRect ViewPort, defined as a fraction of the size of the rectangle itself. (0,0) corresponds to the lower left corner while (1,1) corresponds to the upper right corner.

6) ArrangeType

It is the scroll direction, there are four types:

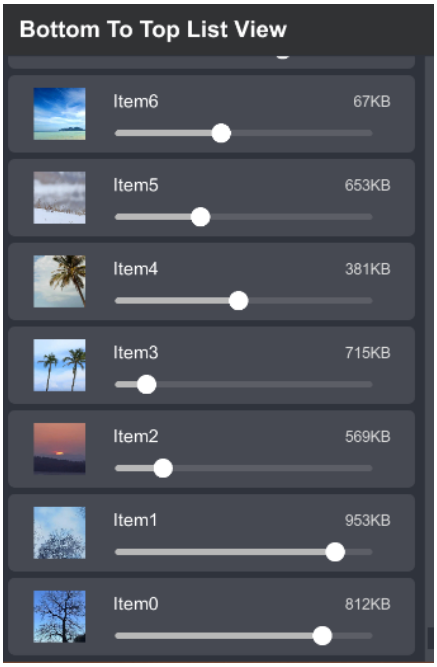
- **TopToBottom**

  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.



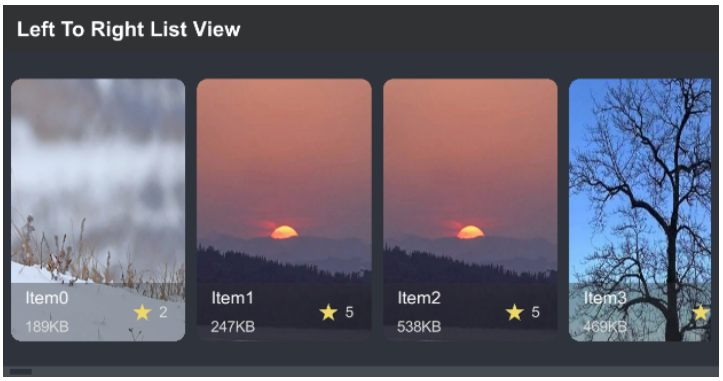
- **BottomToTop**

  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.



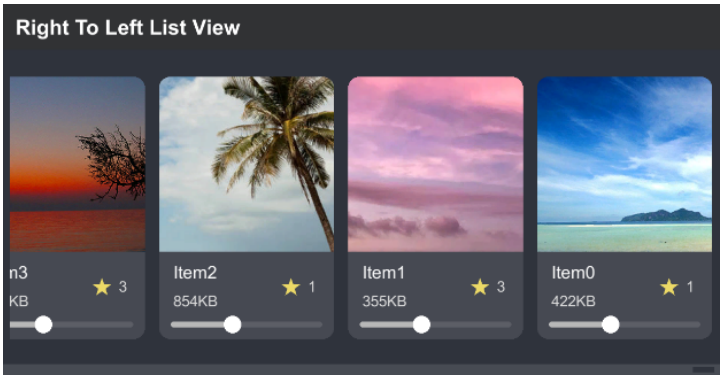
- **LeftToRight**

  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.



- **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.



2.3 LoopListView2 Important Public Method

1) **InitListView** Method

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

This method can be used 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**.

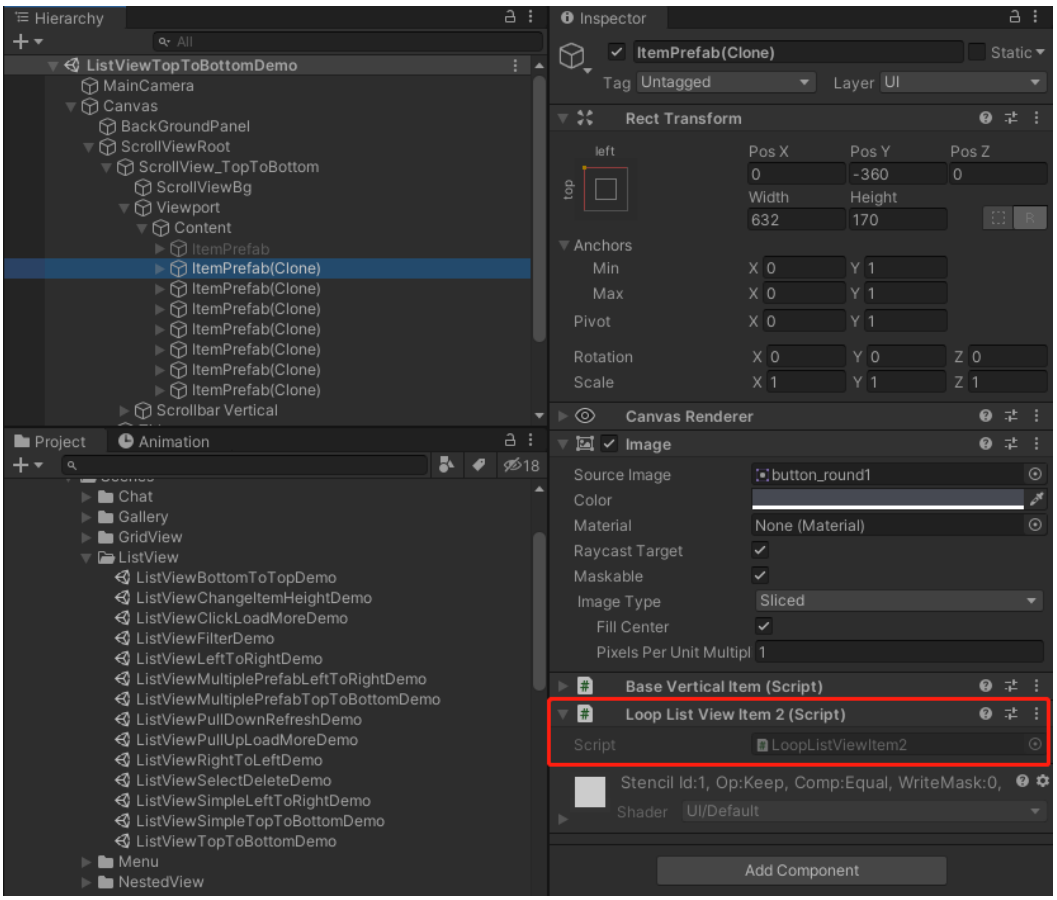
- **onGetItemByIndex**

  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**

  The return value of `onGetItemByIndex`.

  Every created item has a LoopListViewItem2 component auto attached.




```
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.
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 user defined id of the item. This property can be set to any user defined integer value, for any purpose, such as to help searching items.

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

```
LoopListViewItem2 OnGetItemByIndex(LoopListView2 listView, int index)
{
    if (index < 0 || index >= mDataSourceMgr.TotalItemCount)
    {
        return null;
    }
    //get the data to showing
    ItemData itemData = mDataSourceMgr.GetItemDataByIndex(index);
    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("ItemPrefab");
    //get your own component
    BaseVerticalItem itemScript = item.GetComponent<BaseVerticalItem>();
    //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,index);
    return item;
}
```

2) **NewListViewItem** Method

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

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

3) **SetListItemCount** Method

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

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 change after this method finished.

4) **GetShownItemByItemIndex** Method

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

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

5) **GetShownItemByItemId** Method

```
public LoopListViewItem2 GetShownItemByItemId(int itemId)
```

To get the visible item by a user defined itemId. If the item is not visible, then this method return null.

6) **GetShownItemByIndex** Method

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

Get the visible item by the index in visible items list.

All visible items are stored in a List<LoopListViewItem2>, which is named `mItemList`.

The parameter index is from 0 to `mItemList.Count`.

7) **ShowItemCount** Method

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

Get the total count of all visible items.

8) **RefreshItemByItemIndex** Method

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

Update an 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.

9) **RefreshAllShownItem** Method

```
public void RefreshAllShownItem()
```

Update all visible items.

10) **MovePanelToItemIndex** Method

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

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.**

11) **OnItemSizeChanged** Method

```
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.

12) FinishSnapImmediately Method

```
public void FinishSnapImmediately()
```

Snap move will finish at once.

13) CurSnapNearestItemIndex Method

```
public int CurSnapNearestItemIndex
```

Get the nearest item index with the viewport snap point.

14) SetSnapTargetItemIndex Method

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

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

15) ClearSnapData Method

```
public void ClearSnapData()
```

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

3. LoopGridView

3.1 LoopGridView overview

LoopGridView component is attaching to the same gameobject of UGUI ScrollRect

It is mainly for the Grid View 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 Grid View 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.

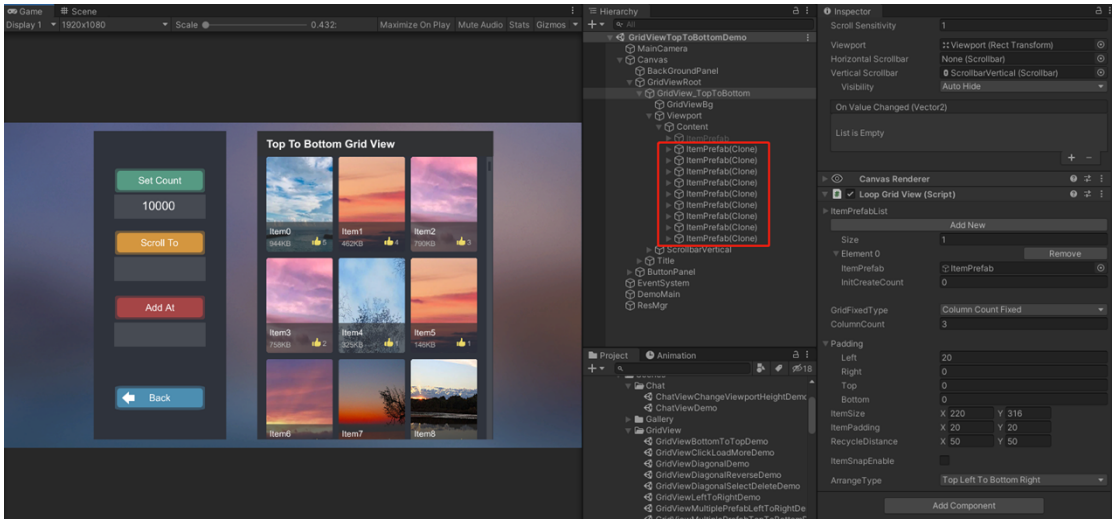
When the ScrollRect moving up, for example, the LoopGridView component would check the topmost row's position, and once the topmost row is out of the viewport, then the LoopGridView component would recycle all the items of the topmost row.

At the same time, it checks the bottommost row's position, and once the bottommost 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 bottommost row, so the new created row becomes the new bottommost row.

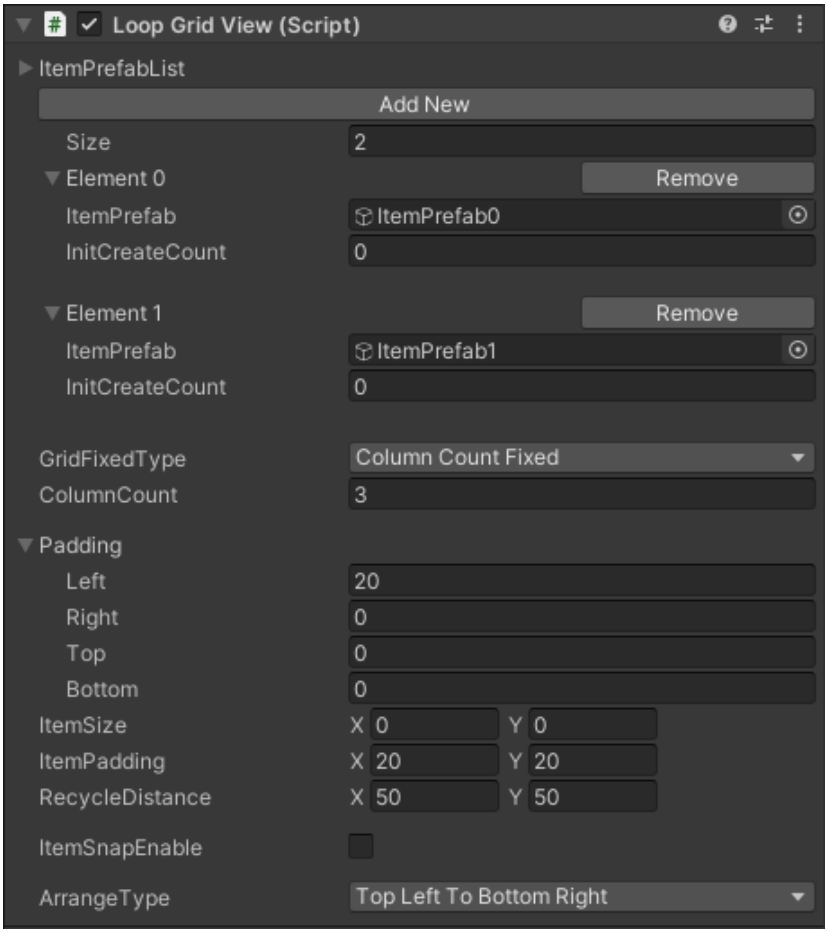
Every item can use a different prefab.

The Grid View have 10000 items, but in fact, only a few items really created. The following figure is what a TopLeft To BottomRight (100 rows * 100 columns) arranged ScrollRect looks like:



3.2 LoopGridView Inspector Settings

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



1) ItemPrefabList

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).

2) 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.

3) 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.

4) Padding

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

5) **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.

6) **ItemPadding**

The amount of spacing between each item in the GridView.

7) **RecycleDistance**

How much distance an item leaves the viewport when the item would be recycled.

8) **ItemSnapEnable**:

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

9) **ItemSnapPivot**

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.

10) **ViewPortSnapPivot**

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.

11) **ArrangeType**:

The scroll direction, there are four types such as: TopLeftToBottomRight, BottomLeftToTopRight, TopRightToBottomLeft and BottomRightToTopLeft.

3.3 LoopGridView Important Public Method

1) **InitGridView** method

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

Initiate the LoopGridView component. There are 4 parameters:

- **itemTotalCount**

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**

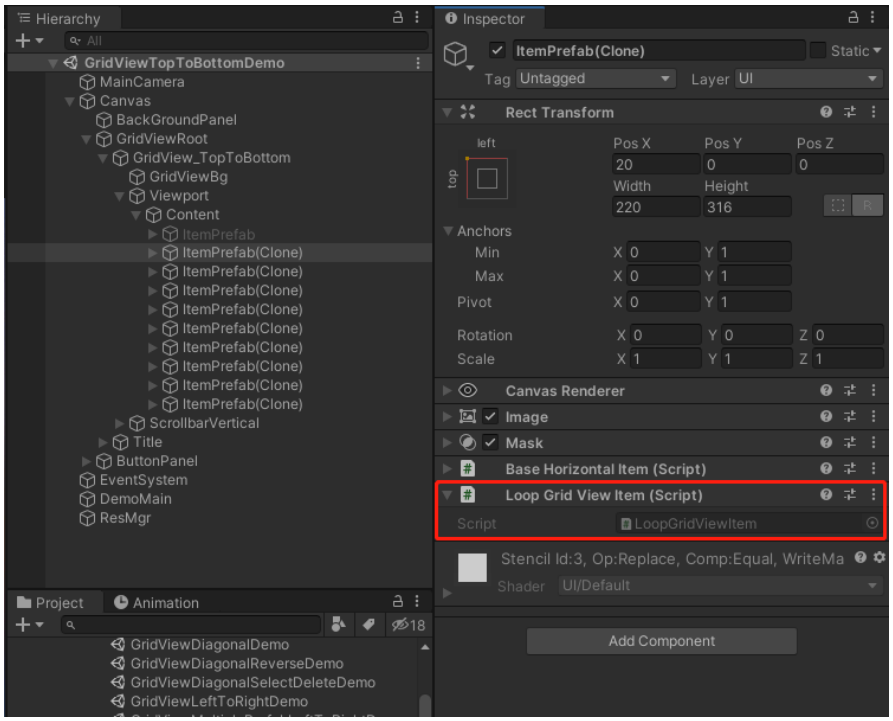
  This parameter is a LoopGridViewSettingParam object. You can use this parameter to override the values in the Inspector Setting, for example:

```
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**

  The return value of onGetItemByRowColumn.

  Every created item has a LoopGridViewItem component auto attached.



```
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.

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 index,int row,int column)
{
    if (index < 0 || index >= mDataSourceMgr.TotalItemCount)
    {
        return null;
    }
    //get the data to showing
    ItemData itemData = mDataSourceMgr.GetItemDataByIndex(index);
    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("ItemPrefab");
    //get your own component
    BaseHorizontalItem itemScript = item.GetComponent<BaseHorizontalItem>();
    // 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, index);
    return item;
}
```

2) **NewListViewItem** method

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

Get a new item, and the new item is a clone from the prefab named itemPrefabName. This method is usually used in onGetItemByRowColumn.

3) **SetListItemCount** method

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

Set the item total count of the Grid View at runtime. If resetPos is false, then the ScrollRect's content position will not change after this method finished.

4) **GetShownItemByItemIndex** method

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

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

5) **GetShownItemByItemIndex** method

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

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

6) **RefreshItemByItemIndex** method

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

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.

7) **RefreshItemByRowColumn** method

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

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.

8) **RefreshAllShownItem** method

```
public void RefreshAllShownItem()
```

Update all visible items.

9) **MovePanelToItemByIndex** method

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

Move the ScrollRect content's position to (the positon of itemIndex-th **item** + offset).

10) **MovePanelToItemByRowColumn** method

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

Move the panel's position to (the position of (row,column) item + offset).

11) **SetGridFixedGroupCount** method

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

Change the GridFixedType and fixed row/column count at runtime. Also it can change the

itemSize, itemPadding, contentPadding at runtime by SetItemSize(Vector2), SetItemPadding(Vector2), SetPadding(RectOffset).

12) **FinishSnapImmediately** method

```
public void FinishSnapImmediately()
```

Snap move will finish at once.

13) **RowColumnPair** method

```
public RowColumnPair CurSnapNearestItemRowColumn
```

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

14) **SetSnapTargetItemRowColumn** method

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

Set the snap target item' row and column.

15) **ClearSnapData** method

```
public void ClearSnapData()
```

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

4. LoopStaggeredGridView

4.1 LoopStaggeredGridView overview

LoopStaggeredGridView is mainly for StaggeredGridView that the items have different height for a vertical Grid View (or different width for a horizontal GridView). So if all the items of a Grid View 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.

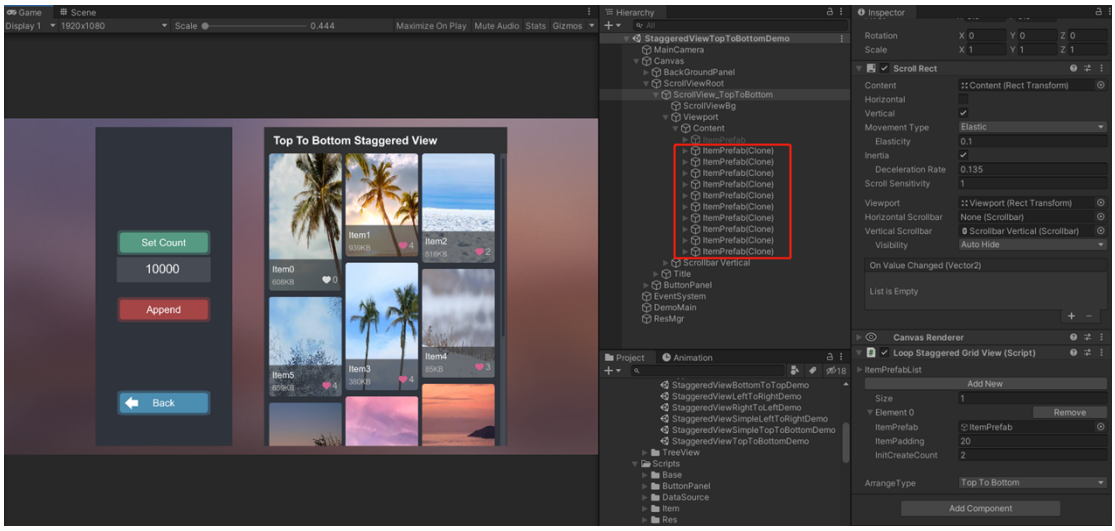
When the ScrollRect moving up, for a vertical GridView, the LoopStaggeredGridView component would check the topmost item's position of every column, and once the topmost item of a column is out of the viewport, then the LoopStaggeredGridView component would recycle the topmost item.

And at the same time, check the bottommost item's position of every column, and once the

bottommost 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 bottommost item of the shortest column**, that is the column whose bottommost item's bottom position is the top most position in all the columns' bottommost positon. In the whole, all the items are arranged from left to right, from top to bottom.

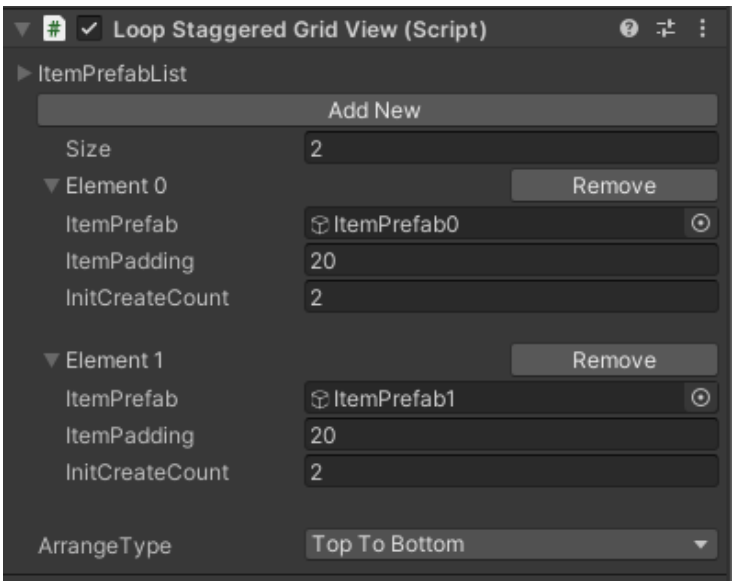
Every item can use a different prefab. But for a vertical Grid View, items can have different height but the width must be same. For a horizontal Grid View, items can have different width but the height must be same.

The ScrollRect has 10,000 items, but in fact, only a few items really created. The following figure is what a TopToBottom arranged StaggeredGridView with 3 columns looks like:



4.2 LoopStaggeredGridView Inspector Settings

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



1) **ItemPrefabList**

The existing items to clone.

Every item can use a different prefab and every prefab can have different **default padding** which is 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).

2) **ArrangeType**

The scroll direction, this is same to List View.

There are four types: **TopToBottom, BottomToTop, LeftToRight, RightToLeft**.

4.3 LoopStaggeredGridView Important Public Method

1) **InitListView** method

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

Initiate the LoopStaggeredGridView component. It has four 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**

  You need new a GridViewLayoutParam instance and set the values you want.

  For a vertical Grid View, mColumnOrRowCount is the column count, mItemWidthOrHeight is the item's width, mPadding1 is the viewport left margin, mPadding2 is the viewport right margin.

  For a horizontal Grid View, 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 Grid View 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.

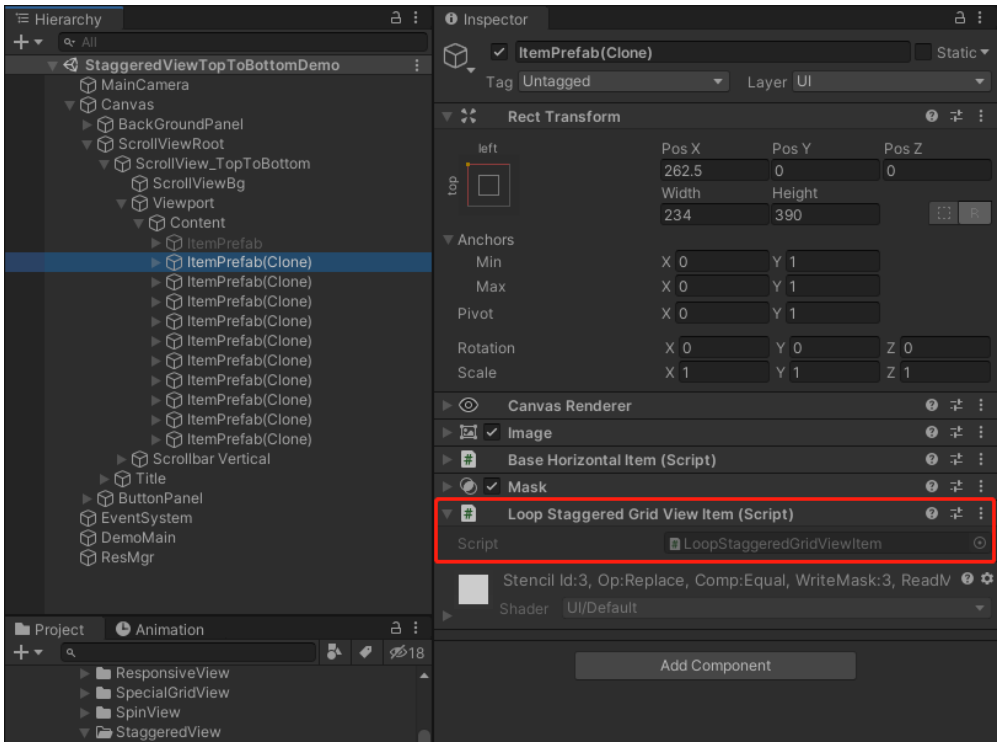
- onGetItemByItemIndex

  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**

  The return value of onGetItemByItemIndex.

  Every created item has a LoopStaggeredGridViewItem component auto attached.




```
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. It can be from 0 to itemTotalCount -1.

The mItemIndexInGroup property indicates the item's index in a column or row. For a vertical Grid View, a group means a column and for a horizontal Grid View, 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 listView, int index)
{
    if (index < 0 || index >= mDataSourceMgr.TotalItemCount)
    {
        return null;
    }
    //get the data to showing
    ItemData itemData = mDataSourceMgr.GetItemDataByIndex(index);
    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 LoopStaggeredGridView Inspector Setting */
    LoopStaggeredGridViewItem item = listView.NewListViewItem("ItemPrefab");
    //get your own component
    BaseHorizontalItem itemScript = item.GetComponent<BaseHorizontalItem>();
    // 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, index);
    return item;
}
```

2) **NewListViewItem** method

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

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

3) **SetListItemCount** method

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

Set the item total count of the Grid View at runtime. If resetPos is set false, then the ScrollRect's content position will not change after this method finished.

4) **GetShownItemByItemIndex** method

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

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

5) **RefreshItemByItemIndex** method

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

Update an 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).

6) **RefreshAllShownItem** method

```
public void RefreshAllShownItem()
```

Update all visible items.

7) **MovePanelToItemIndex** method

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

Move the ScrollRect content's position to (the positon of itemIndex-th **item** + offset)

8) **OnItemSizeChanged** method

```
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).