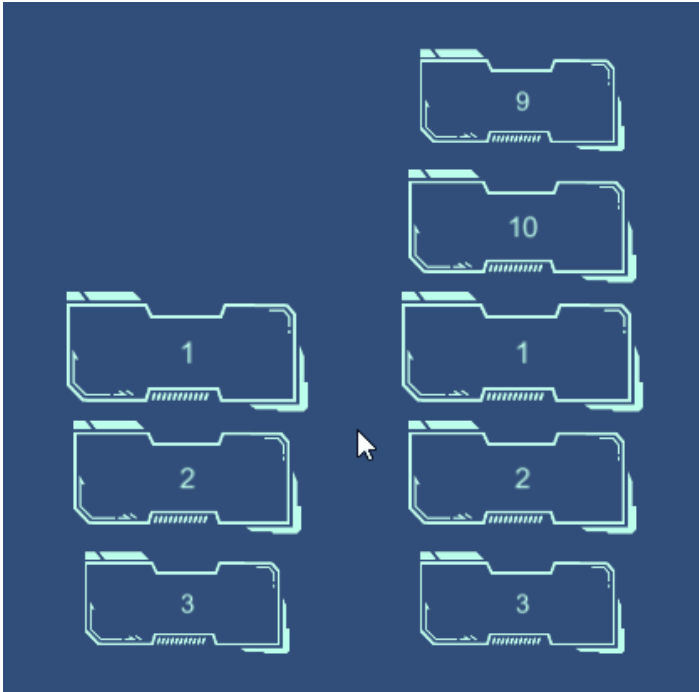


Circular Scrolling List



The quick overview of version 5 - [Demo video](#)

Outline

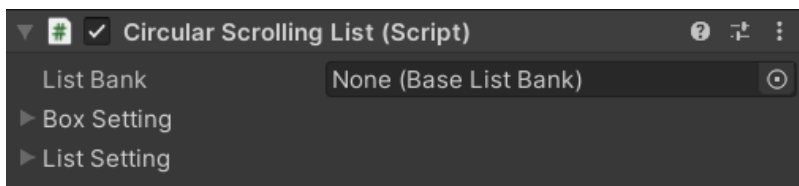
- [Circular Scrolling List](#)
 - [Outline](#)
 - [Features](#)
 - [Setting](#)
 - [Box Setting](#)
 - [List Setting](#)
 - [List Mode](#)
 - [List Appearance](#)
 - [List Events](#)
 - [How to Use](#)
 - [Set Up the List](#)
 - [Set the Layout Area](#)
 - [Set the Control Mode](#)
 - [Set the Focusing Position](#)
 - [Appearance Curves](#)
 - [Curve Presets](#)

- ListBank and ListBox
 - Custom ListBank
 - Custom ListBox
 - Use Them in the List
 - Avoid Boxing/Unboxing Problem
- Get the ID of the Selected Content
 - OnBoxClick Event
 - OnCenteredContentChanged Event
 - Manually Get the Centered Content ID
- Select the Content from Script
- Refresh the List

Features

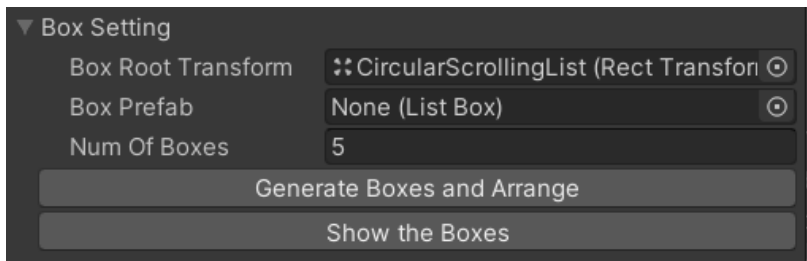
- Use finite list boxes to display infinite contents
- 2 list types: Circular or Linear mode
- 3 control modes: Pointer, Mouse wheel, and Script
- 3 focusing (ending) position: Top, Center, and Bottom
- Support both vertical and horizontal scrolling
- Support all three render modes of the canvas plane
- Custom layout and movement, and layout preview in the editor
- Custom displaying contents
- Support dynamic list contents
- Script interacting
- Image sorting - The box which is closest to the focusing position will be popped up
- Callback events
- Support Unity 2018.4+ (Tested in Unity 2018.4.15f1. The demo scenes in the project are made in Unity 2019.4.16f1)

Setting



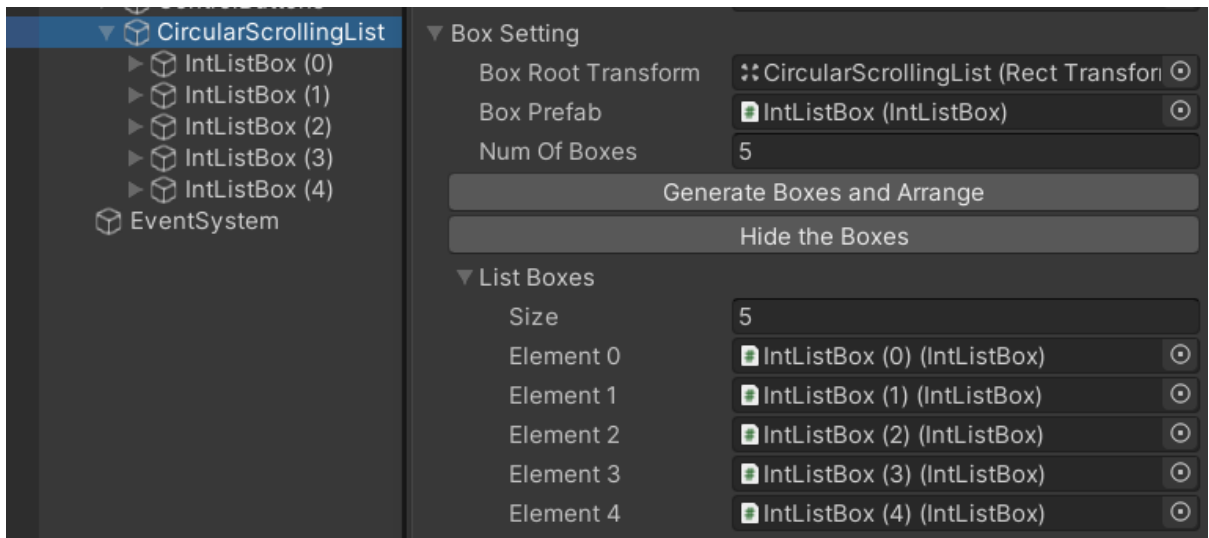
Property	Description
List Bank	The game object that stores the contents for the list to display
Box Setting	The setting of the list box. See Box Setting section
List Setting	The setting of the list. See List Setting section

Box Setting



Property	Description
Box Root Transform	The root rect transform that holding the list boxes. Default to the gameobject where the script is attached to
Box Prefab	The prefab of the list box
Num Of Boxes	The number of boxes to be generated
Generate Boxes and Arrange	Generate the boxes under the "Box Root Transform" and arrange them according to the list appearance
Show/Hide the Boxes	Show or hide the reference of managed boxes

The managed boxes will be shown when click the "Show the Boxes" button, and be hidden by clicking the button again:



List Setting

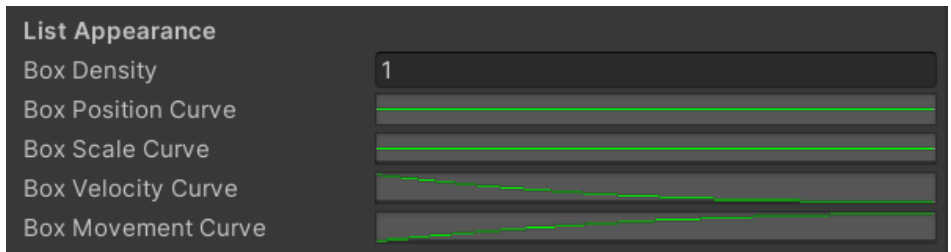
List Mode

List Mode	
List Type	Circular
Direction	Vertical
Control Mode	Everything
Align At Focusing Position	<input type="checkbox"/>
Reverse Scrolling Direction	<input type="checkbox"/>
Focusing Position	Center
Reverse Content Order	<input type="checkbox"/>
Init Focusing Content ID	0
Focus Selected Box	<input type="checkbox"/>
Initialize On Start	<input checked="" type="checkbox"/>

Property	Description
List Type	The type of the list. Could be Circular or Linear
Direction	The major scrolling direction. Could be Vertical or Horizontal
Control Mode	The controlling mode. Could be Nothing , or Everthing , Pointer , and Mouse Wheel See Set the Control Mode for more information
└ Align At Focusing Position	Whether to align a box at the focusing position after sliding or not. Available if the control mode has Pointer set.
└ Reverse Scrolling Direction	Whether to reverse the scrolling direction or not. Available if the control mode has Mouse Wheel set.
Focusing Position	The focusing (ending) position of the list. Could be Top , Center , or Bottom See Set the Focusing Position for more information
└ Reverse Content Order	Whether to reverse the content displaying order or not. Available if the focusing position is Center .
Init Focusing Content ID	The initial content ID to be displayed in the focusing box
Focus Selected Box	Whether to move the selected box to the focusing position or not. The list box must be a button to make this function take effect.

Property	Description
Initialize On Start	Whether to initialize the list in its <code>Start()</code> or not If it is false, manually initialize the list by invoking <code>CircularScrollingList.Initialize()</code>

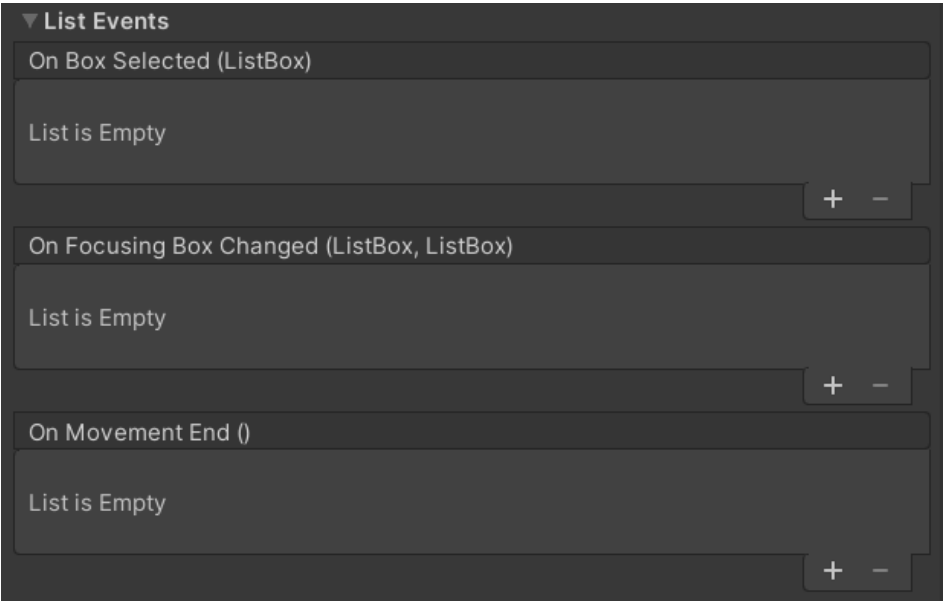
List Appearance



Property	Description
Box Density	The factor for adjusting the distance between boxes. The larger, the closer
Box Position Curve	The curve specifying the passive position of the box
Box Scale Curve	The curve specifying the box scale
Box Velocity Curve	The curve specifying the velocity factor of the box after releasing. Available if the control mode has Pointer set.
Box Movement Curve	The curve specifying the movement factor of the box

For the detailed information of the curves, see [Appearance Curves](#).

List Events

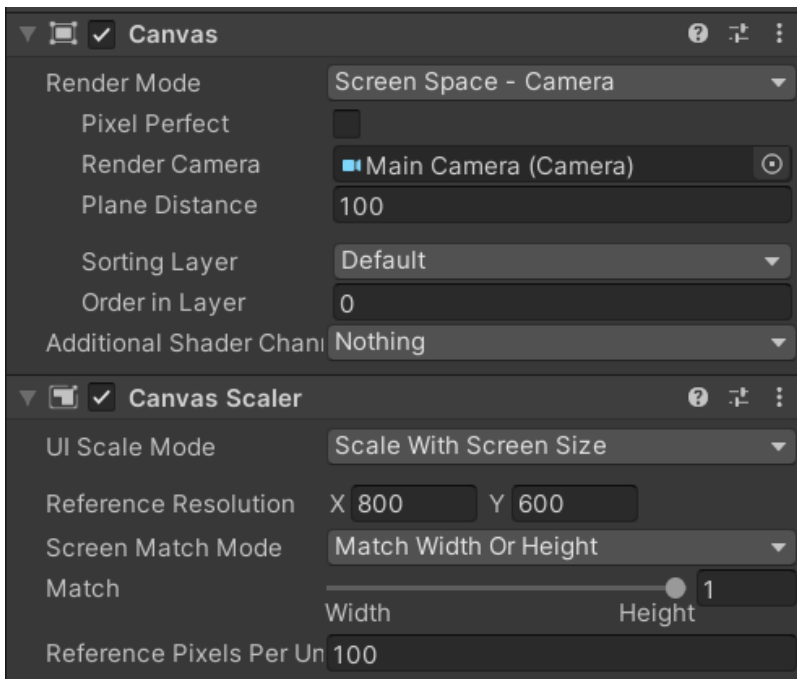


Property	Description
On Box Selected	The callback to be invoked when a box is selected by clicking. The <code>ListBox</code> parameter is the selected box.
On Focusing Box Changed	The callback to be invoked when the focusing box is changed. The first parameter is the previous focusing box, and the second parameter is the current one.
On Movement End	The callback to be invoked when the list movement is ended

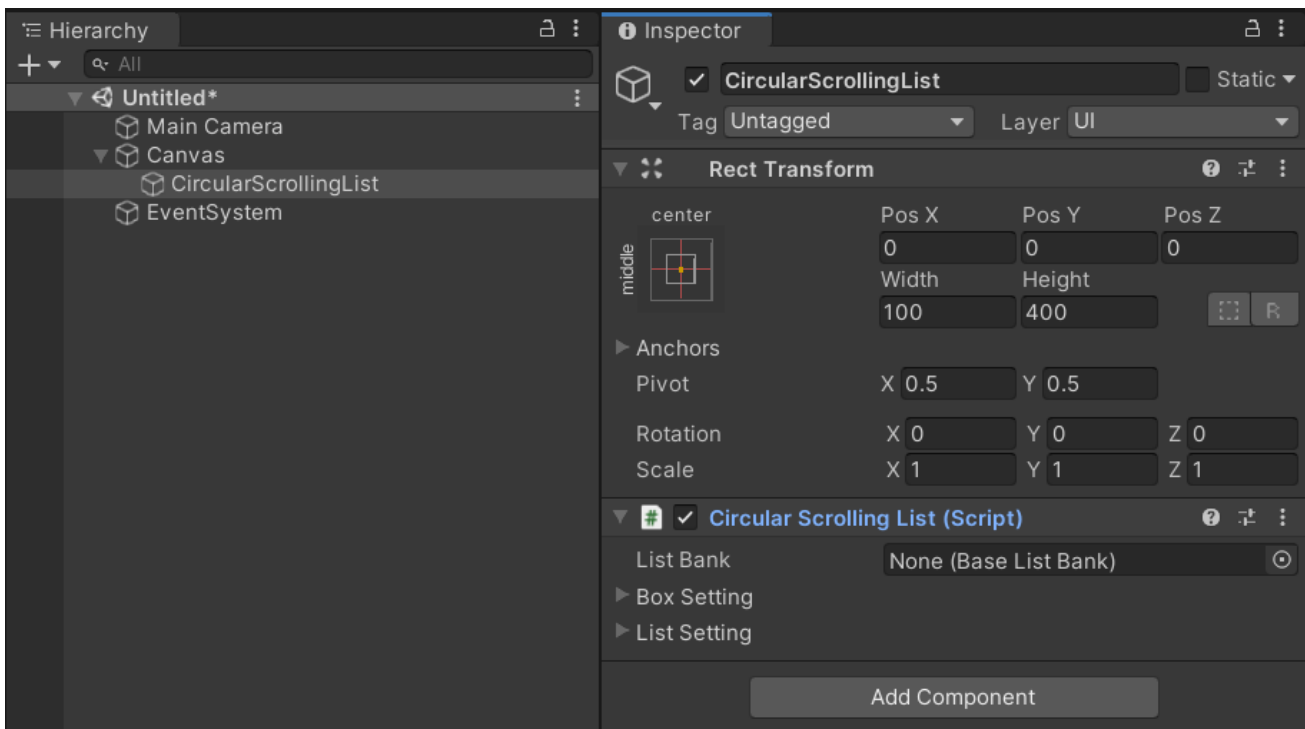
How to Use

Set Up the List

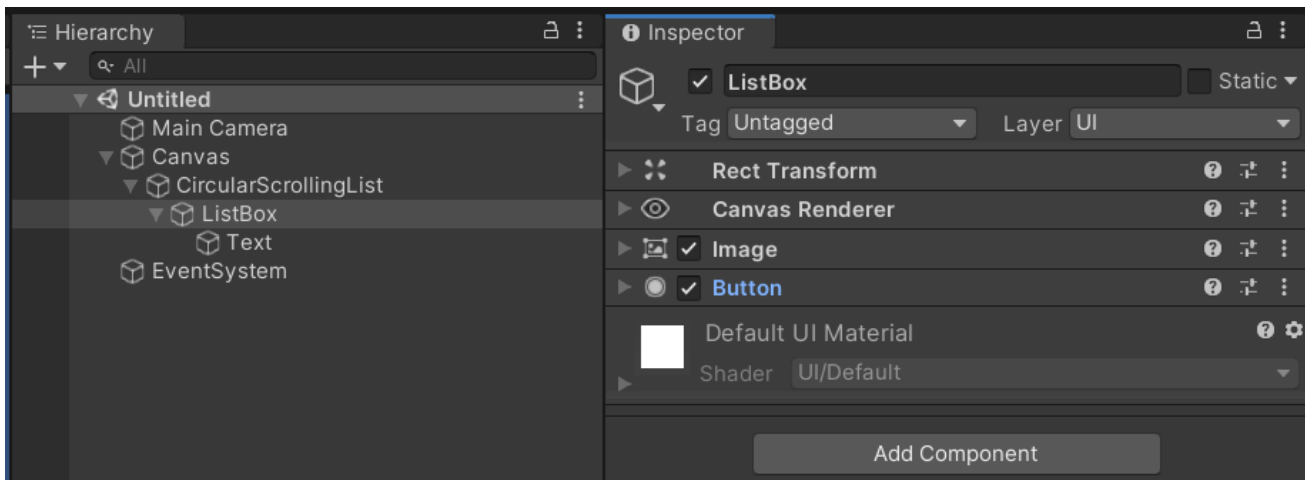
1. Add a Canvas plane to the scene. Set the render mode to "Screen Space - Camera" for example, and assign the "Main Camera" to the "Render Camera". Set the ui scale mode to "Scale With Screen Size", and the "Match" to 1.



2. Create an empty gameobject as the child of the canvas plane, rename it to "CircularScrollingList" (or other name you like), and set the height to 400. It will define the reference area of the list (See [Set the Layout Area](#) for more information). Then attach the script `ListPositionCtrl.cs` to it.



3. Create a Button gameobject as the child of the "CircularScrollingList", rename it to "ListBox", and adjust the image or text size if needed.



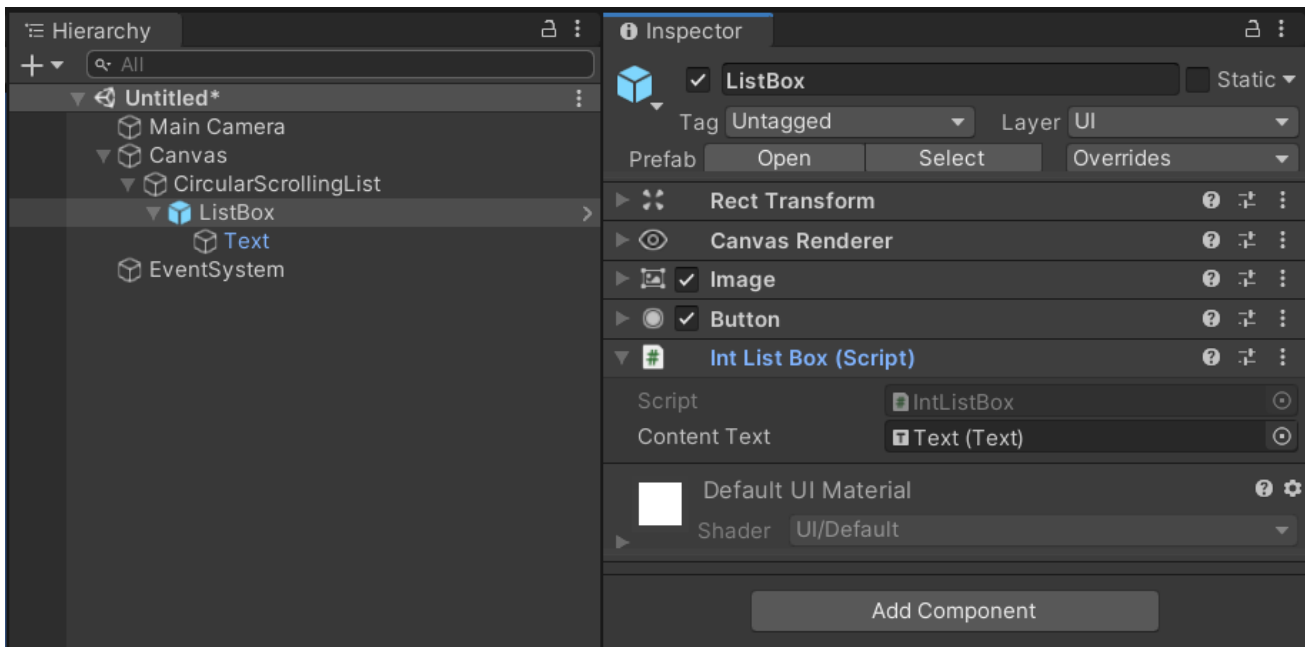
4. Create a new script `IntListBox.cs` and add the following code. For more information, see [ListBank and ListBox](#) section.

```
using AirFishLab.ScrollingList.ContentManagement;
using UnityEngine;
using UnityEngine.UI;

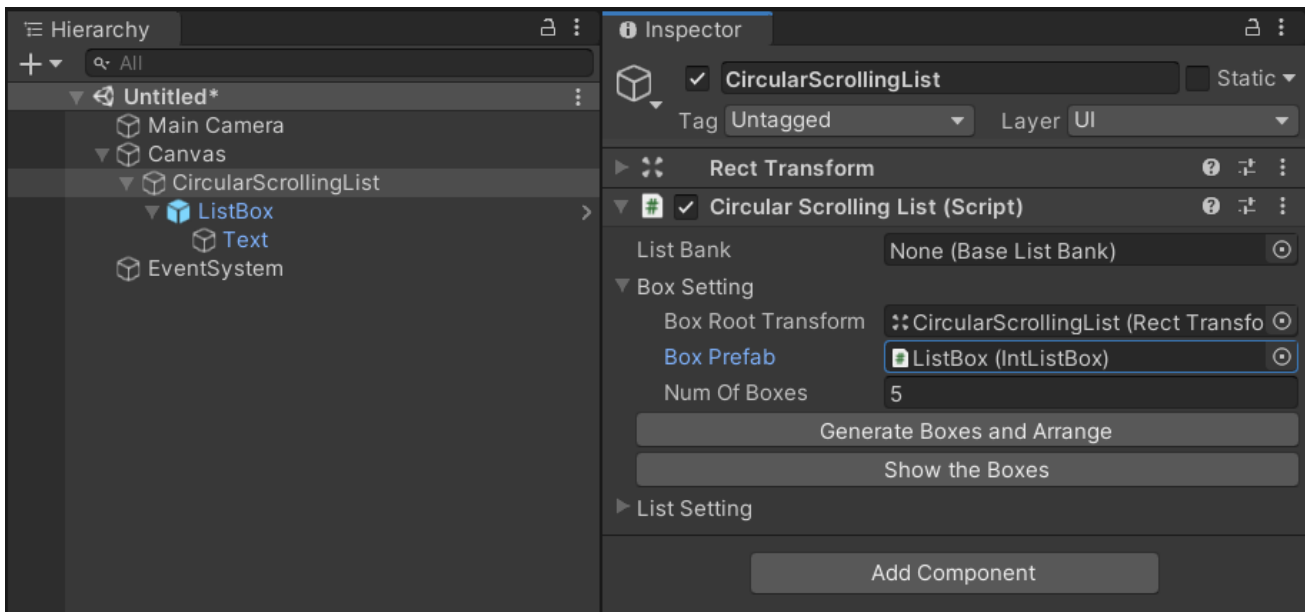
// The box used for displaying the content
// Must inherit from the class `ListBox`
public class IntListBox : ListBox
{
    [SerializeField]
    private Text _contentText;

    // This function is invoked by the `CircularScrollingList` for updating the list content
    protected override void UpdateDisplayContent(IListContent listContent)
    {
        // Code will be added later
    }
}
```

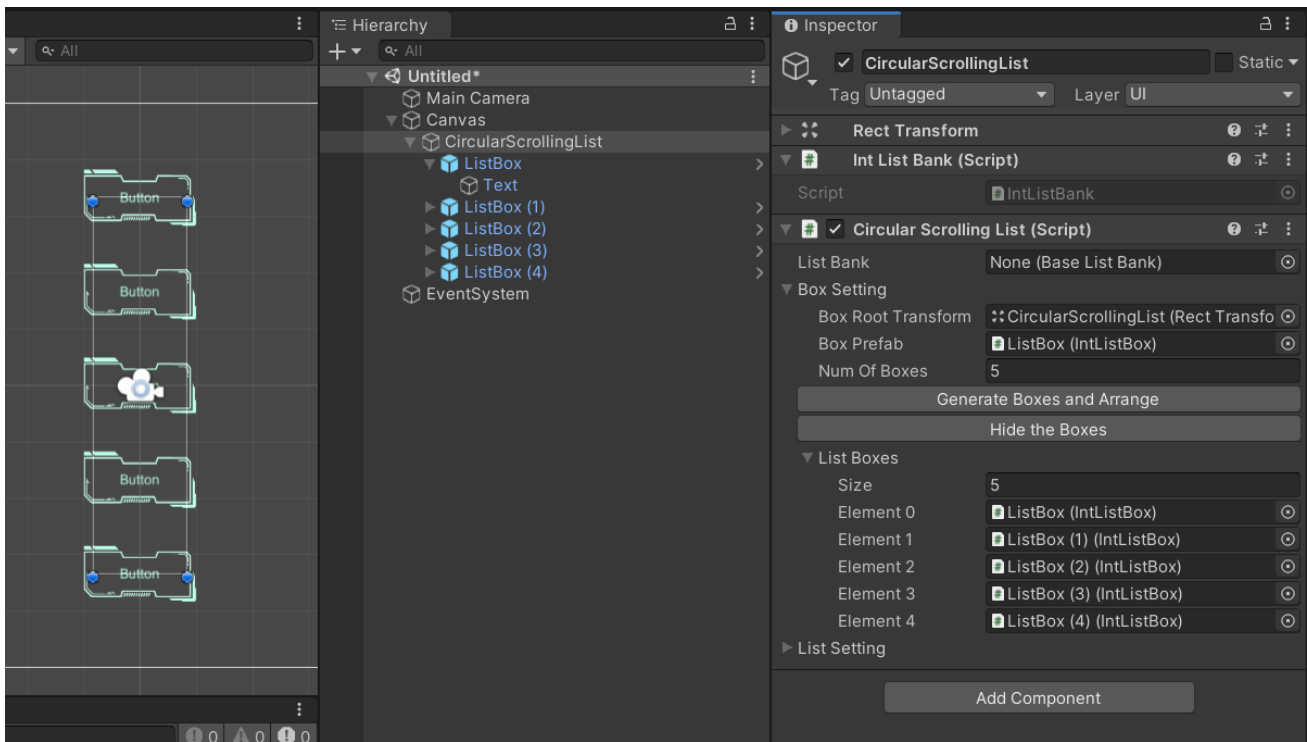
5. Attach the script `IntListBox.cs` to it, assign the gameobject "Text" of the Button to the "Content Text" of the `ListBox.cs`, and then create a prefab of it.



6. Assign the created prefab to the "Box Prefab" in the "Box Setting" of the CircularScrollingList.cs .



7. Click the "Generate Boxes and Arrange" button, and 4 more boxes will be generated and arranged. Click "Show the Boxes" button to view the referenced boxes.



8. Create a new script `IntListBank.cs` and add the following code. For more information, see [ListBank](#) and [ListBox](#) section.

```

using AirFishLab.ScrollingList.ContentManagement;

// The bank for providing the content for the box to display
// Must be inherit from the class BaseListBank
public class IntListBank : BaseListBank
{
    // The content to be passed to the list box
    // must inherit from the class `IListContent`.
    public class Content : IListContent
    {
        public int Value;
    }

    private readonly int[] _contents = {
        1, 2, 3, 4, 5, 6, 7, 8, 9, 10
    };

    // This function will be invoked by the `CircularScrollingList`
    // to get the content to display.
    public override IListContent GetListContent(int index)
    {
        var content = new Content {
            Value = _contents[index]
        };

        return content;
    }

    public override int GetContentCount()
    {
        return _contents.Length;
    }
}

```

9. In the script `IntListBox.cs` , add the code to the function `UpdateDisplayContent()` to receive the content.

```

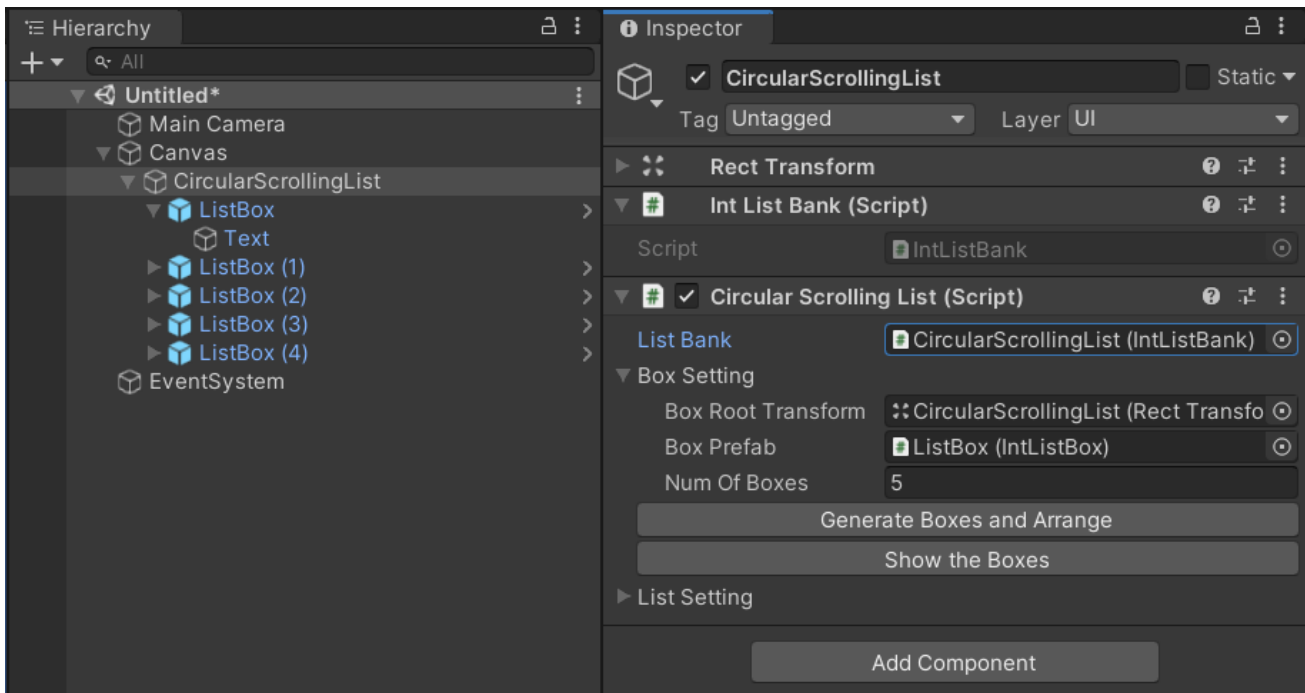
using AirFishLab.ScrollingList.ContentManagement;
using UnityEngine;
using UnityEngine.UI;

// The box used for displaying the content
// Must inherit from the class `ListBox`
public class IntListBox : ListBox
{
    [SerializeField]
    private Text _contentText;

    // This function is invoked by the `CircularScrollingList` for updating the list content
    protected override void UpdateDisplayContent(IListContent listContent)
    {
        var content = (IntListBank.Content)listContent;
        _contentText.text = content.Value;
    }
}

```

10. Attach the script `IntListBank.cs` to the gameobject "CircularScrollingList" (or another gameobject you like), and assign the reference to the "List Bank" of the `CircularScrollingList.cs`.



11. Click "Play" to see the result

Set the Layout Area

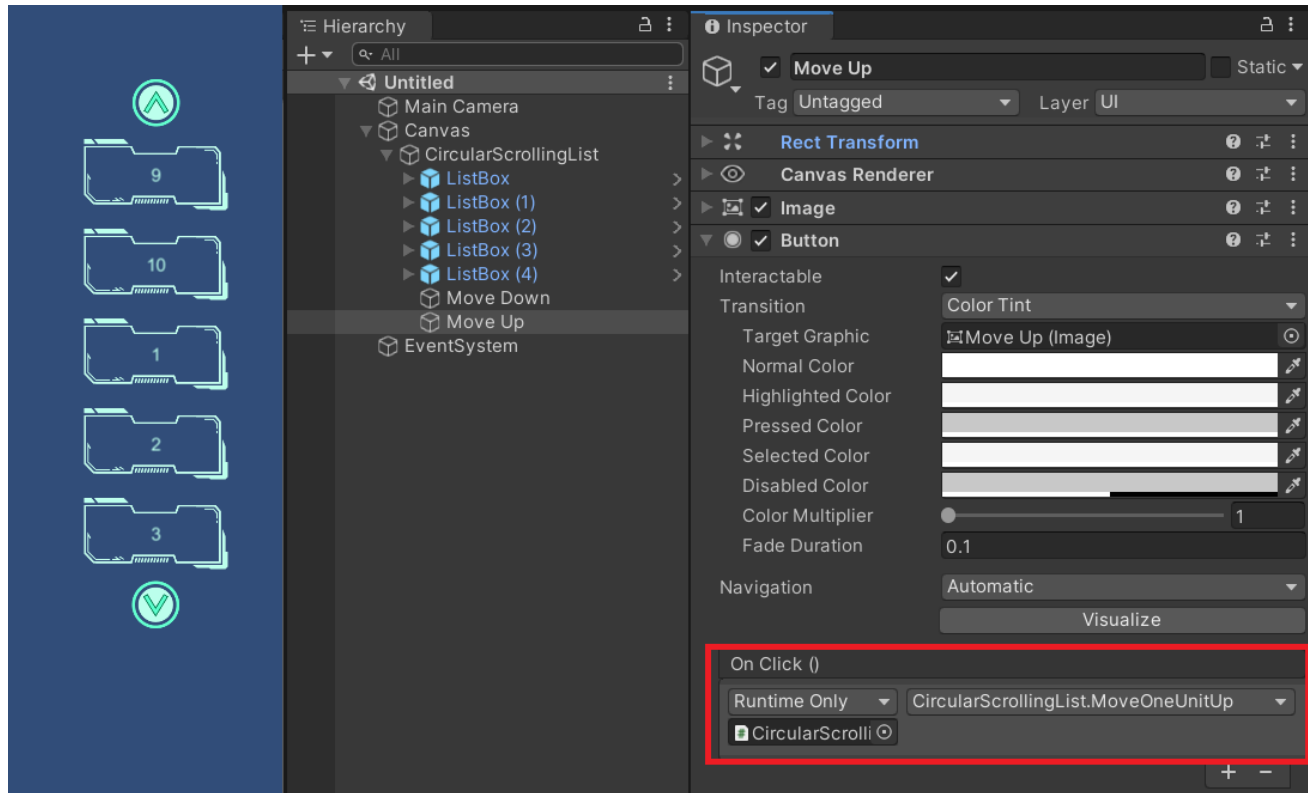
Set the Control Mode

There are 3 control mode for the list:

- **Drag:** The list can be moved by dragging it.
- **Function:** The list can be moved by invoking `CircularScrollingList.MoveOneUnitUp()` or `CircularScrollingList.MoveOneUnitDown()`.

For the **horizontally** scrolling list, invoking `CircularScrollingList.MoveOneUnitUp()` will move the list one unit right, and one unit left by invoking `CircularScrollingList.MoveOneUnitDown()`.

In this mode, the list can be moved by additional buttons by assigning these two function to them.



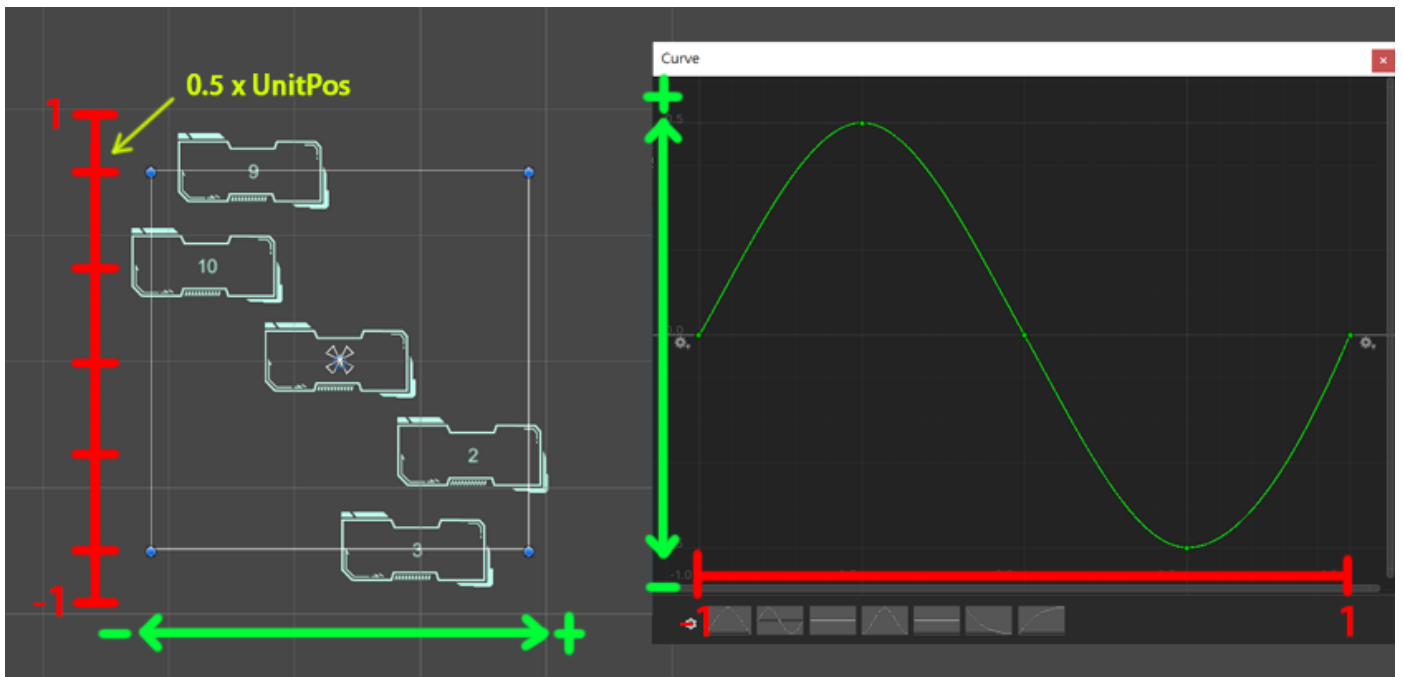
- **Mouse Wheel:** The list can be moved by scrolling the mouse wheel.

Set the Focusing Position

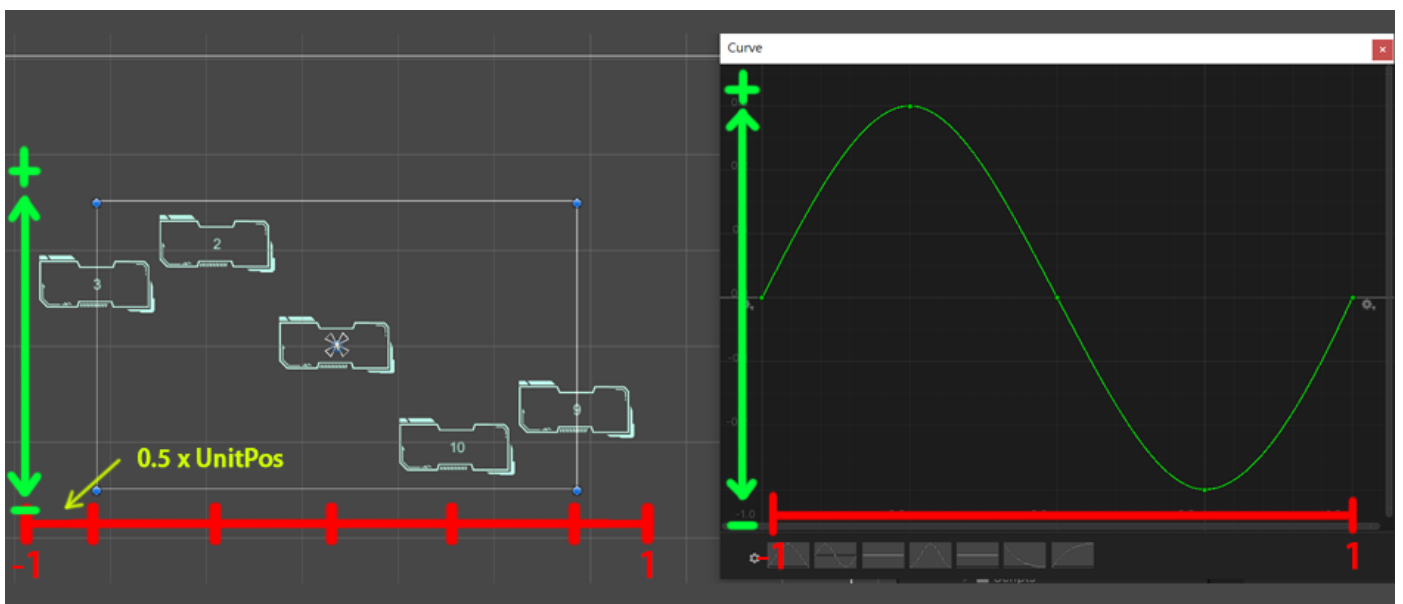
Appearance Curves

- **Box Position Curve:** The curve specifying the passive position of the box
 - X axis: The major position of the box, which is mapped to $[-1, 1]$ (from the smallest value to the largest value).
 - Y axis: The factor of the passive position.

For example, in the vertical mode, the major position is the y position and the passive position is the x position:



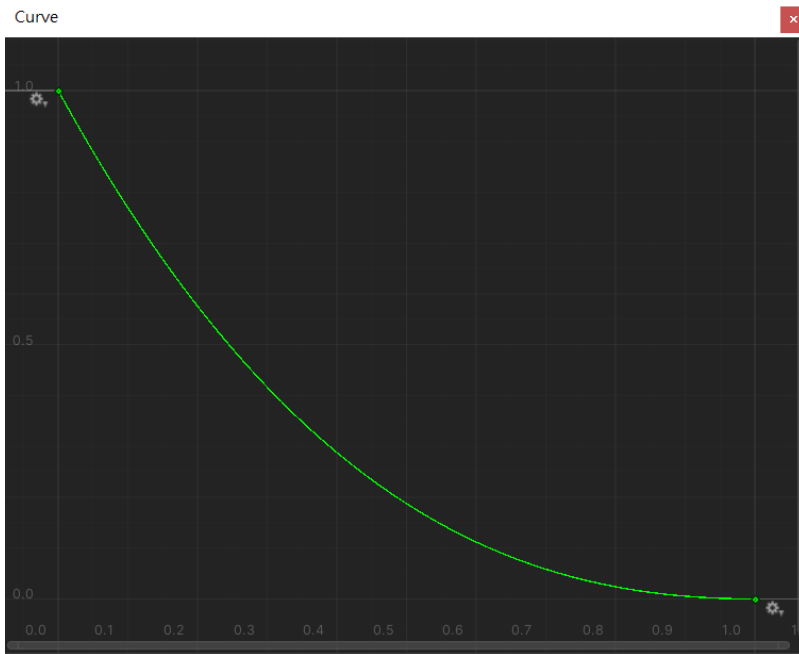
It is intuitive in the horizontal mode:



Note that "1" in the curve equals to $(\text{number of boxes} / 2) * \text{unitPos}$, where unitPos equals to $(\text{width/length of rect} / (\text{number of boxes} - 1))$.

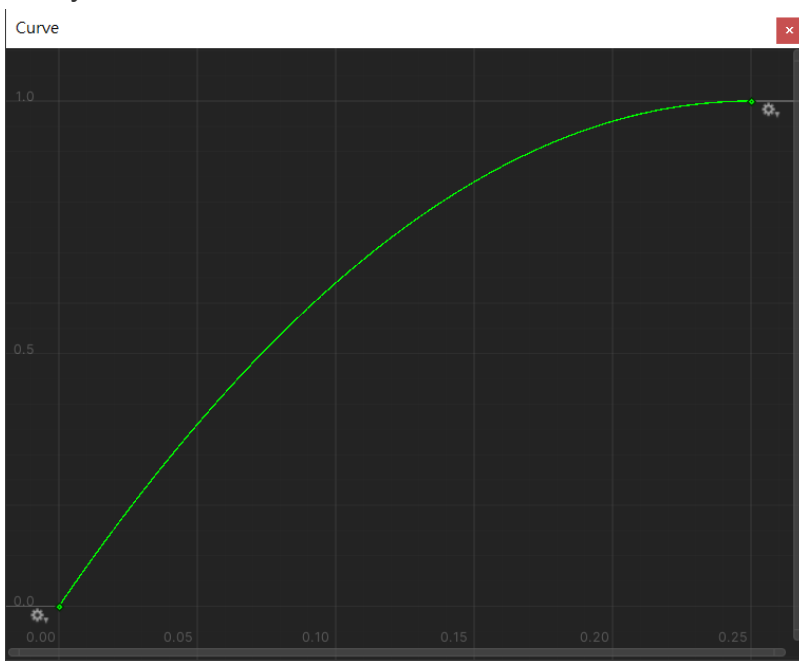
- **Box Scale Curve:** The curve specifying the box scale
 - X axis: Same as the box position curve
 - Y axis: The scale value of the box at that major position
- **Box Velocity Curve:** The curve specifying the velocity factor of the box after releasing
 - X axis: The movement duration in seconds, which starts from 0.
 - Y axis: The factor relative to the releasing velocity

The y value of curve should **start from 1 and end with 0**.



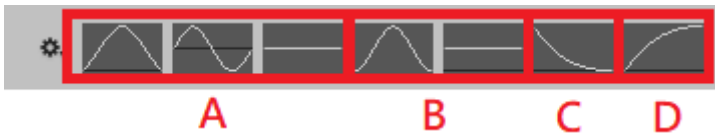
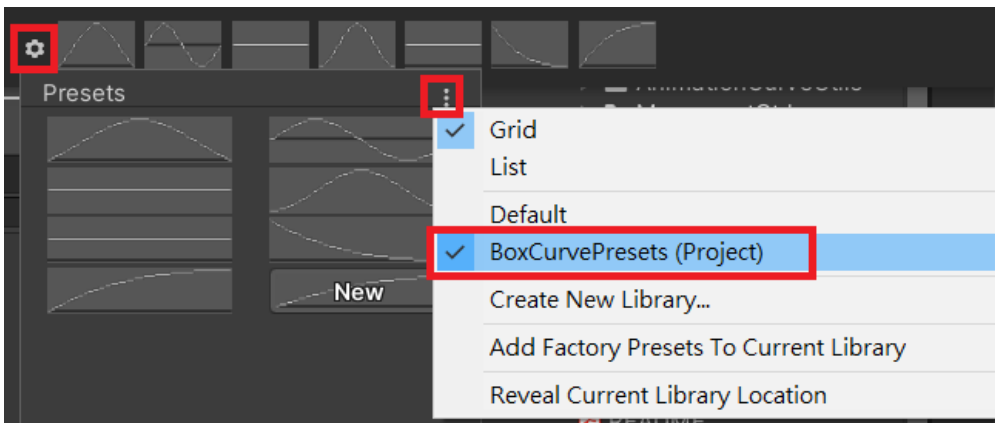
- **Box Movement Curve:** The curve specifying the movement factor of the box.
 - X axis: Same as the box velocity curve
 - Y axis: The factor relative to the target position.

The y value of curve should **start from 0 and end with 1**.



Curve Presets

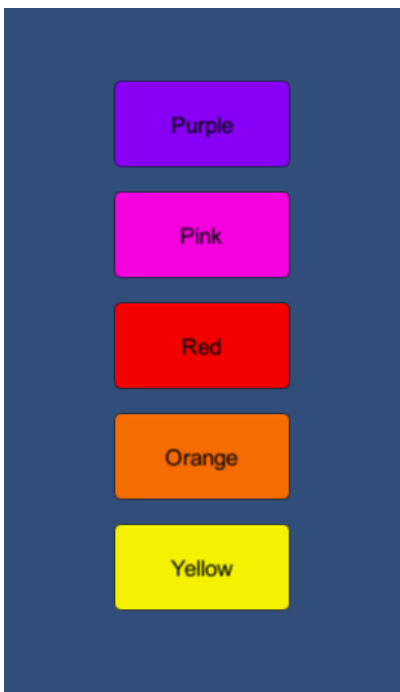
The project provides curve presets. Open the curve editing panel and select the `BoxCurvePresets` to use them.



Part A are position curves, part B are scale curves, part C is a velocity curve, and part D is a movement curve.

ListBank and ListBox

Scene version 5, the list supports custom content type. Different type of ListBank and ListBox can be used in the different list. In this section mentions how to implement your own ListBank and ListBox .



Custom ListBank

Here is the example of the custom ColorStrListBank :

```
public class ColorStrListBank : BaseListBank
{
    [SerializeField]
    private ColorString[] _contents;

    public override object GetListContent(int index)
    {
        return _contents[index];
    }

    public override int GetListLength()
    {
        return _contents.Length;
    }
}

[Serializable]
public class ColorString
{
    public Color color;
    public string name;
}
```

The class must inherit from the class BaseListBank , and there are 2 methods to be implemented:

- public override object GetListContent(int index) : The function for the list to request the content to display. This function always convert the returned content to type object , and it should be converted back to its original type for being used in the custom ListBox .
- public override int GetListLength() : Get the number of the contents.

Custom ListBox

Here is the example of the corresponding ColorStrListBox :

```

using AirFishLab.ScrollingList;
using UnityEngine;
using UnityEngine.UI;

public class ColorStrListBox : ListBox
{
    [SerializeField]
    private Image _contentImage;
    [SerializeField]
    private Text _contentText;

    protected override void UpdateDisplayContent(object content)
    {
        var colorString = (ColorString) content;
        _contentImage.color = colorString.color;
        _contentText.text = colorString.name;
    }
}

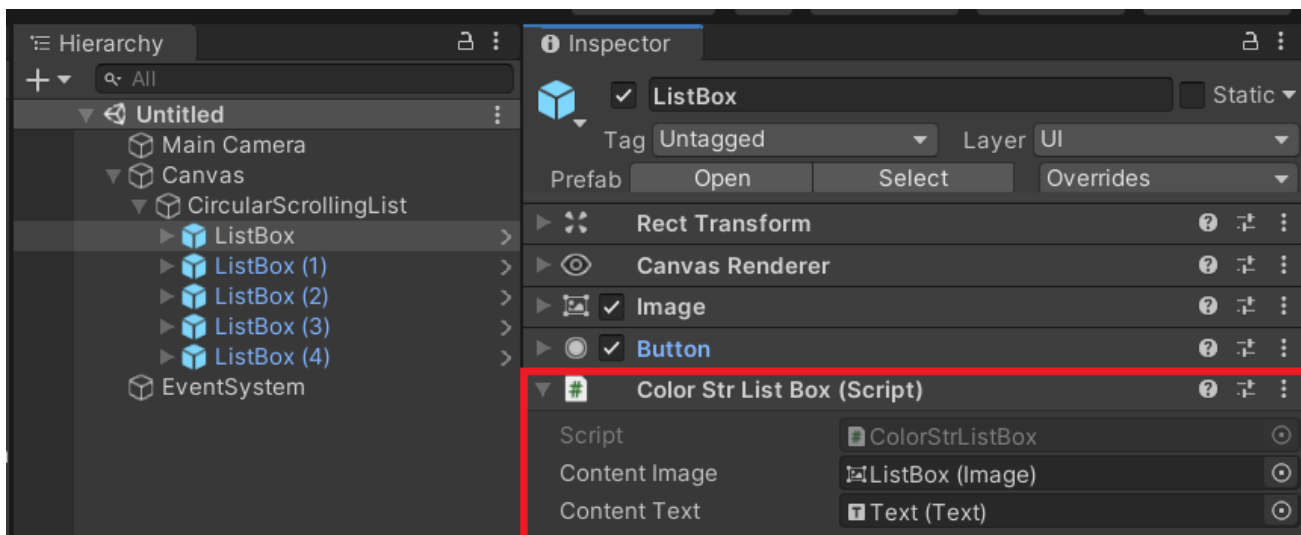
```

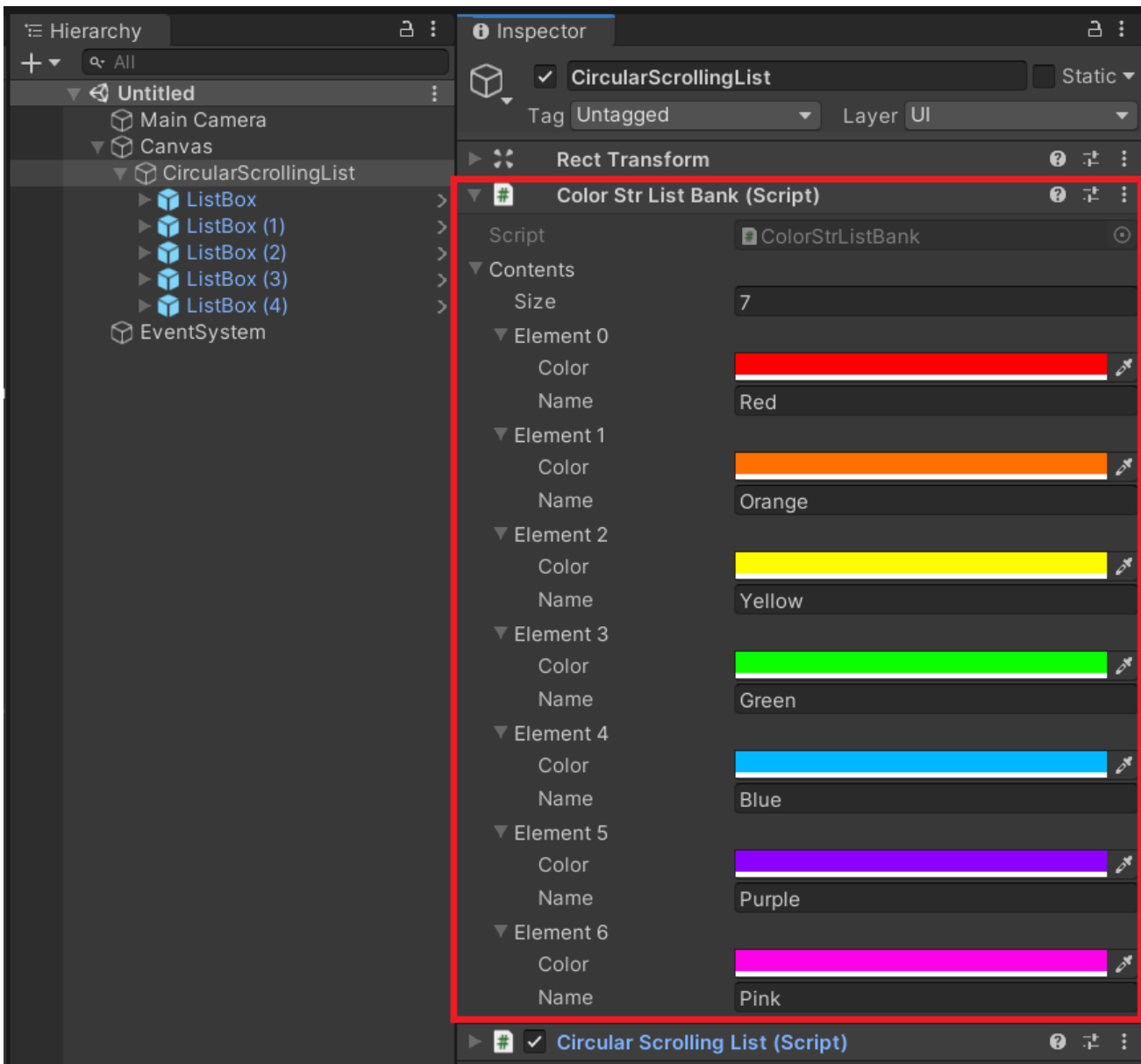
The class must inherit from the class `ListBox` , and there are 1 method to be implemented:

- `protected override void UpdateDisplayContent(object content)` : The function for the list to update the content of the box. `content` is the content requested from the custom list bank, and it should be converted back to its original type for being used.

Use Them in the List

Same as the setup steps in the [Set up the List](#) section but replacing the `IntListBox` and `IntListBank` with your own version of `ListBox` and `ListBank` .





Avoid Boxing/Unboxing Problem

According to [this C# programming guide](#), converting a value type to `object` type is called boxing, and converting `object` type to a value type is called unboxing, which causes a performance problem. To avoid this situation, create a data class to carry the data of value type.

The modified version of `IntListBank` :

```

using AirFishLab.ScrollingList;

public class IntListBank : BaseListBank
{
    private readonly int[] _contents = {
        1, 2, 3, 4, 5, 6, 7, 8, 9, 10
    };

    // Create a data wrapper for carrying the data
    private DataWrapper _dataWrapper = new DataWrapper();

    public override object GetListContent(int index)
    {
        _dataWrapper.value = _contents[index];
        return _dataWrapper;
    }

    public override int GetListLength()
    {
        return _contents.Length;
    }
}

public class DataWrapper
{
    public int value;
}

```

The modified version of IntListBox :

```

using AirFishLab.ScrollingList;

public class IntListBox : ListBox
{
    [SerializeField]
    private Text _contentText;

    protected override void UpdateDisplayContent(object content)
    {
        var data = (DataWrapper) content;
        _contentText.text = (string) data.value;
    }
}

```

Get the ID of the Selected Content

There are three ways to get ID of the selected content.

1. OnBoxClicked event
2. OnCenteredContentChanged event
3. Manually get the centered content ID

OnBoxClick Event

When a box is clicked, the `CircularScrollingList` will launch the event `OnBoxClick` (actually launch from the `Button.onClick` event). The callback function (or the listener) for the event must have 1 parameter for receiving the ID of the selected content.

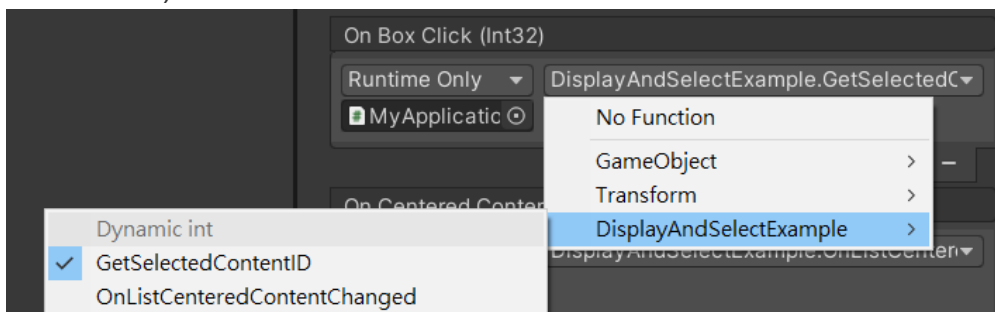
Here is an example of the callback function:

```
using AirFishLab.ScrollingList;

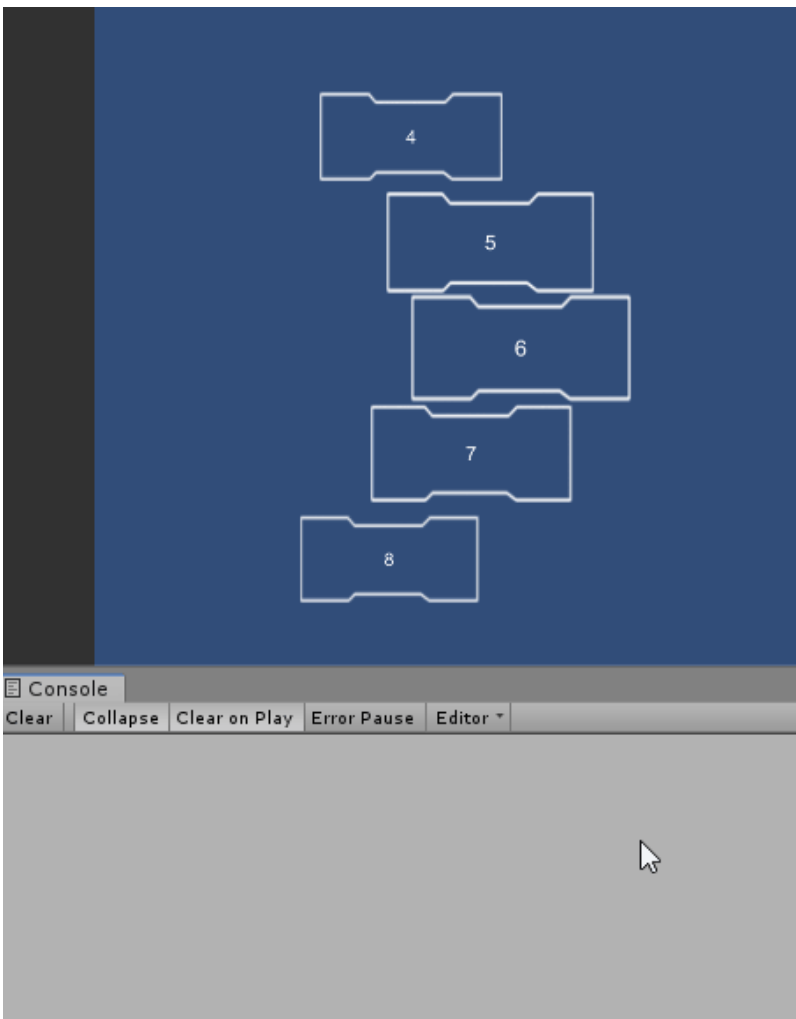
public class DisplayAndSelectExample : MonoBehaviour
{
    [SerializeField]
    private CircularScrollingList _list;

    public void GetSelectedContentID(int selectedContentID)
    {
        var content = (int) _list.listBank.GetListContent(selectedContentID);
        Debug.Log("Selected content ID: " + selectedContentID +
            ", Content: " + content);
    }
}
```

Then, assign it to the property "On Box Click (Int32)". (Note that select the function in the "dynamic int" section)



It will be like:



OnCenteredContentChanged Event

The `OnCenteredContentChanged` event will be invoked when the centered content is changed. The callbacks for this event are similar to the `OnBoxClicked` event.

Here is an example of the callback function:

```

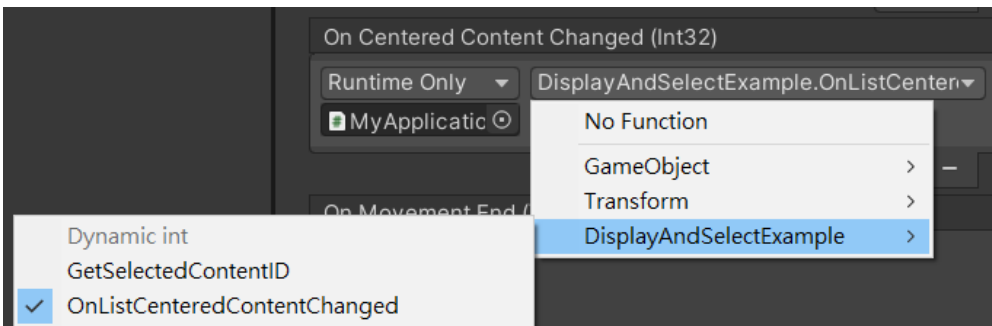
using AirFishLab.ScrollingList;

public class DisplayAndSelectExample : MonoBehaviour
{
    [SerializeField]
    private CircularScrollingList _list;
    [SerializeField]
    private Text _centeredContentText;

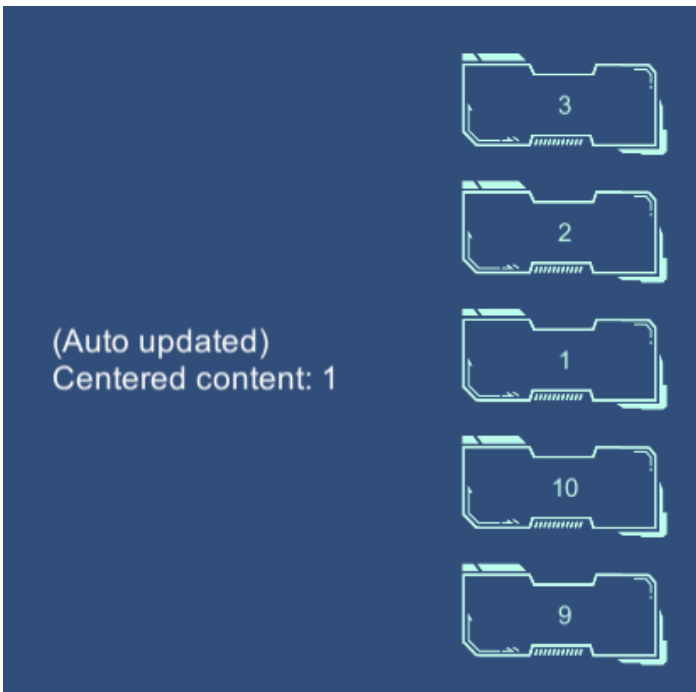
    public void OnListCenteredContentChanged(int centeredContentID)
    {
        var content = (int) _list.listBank.GetListContent(centeredContentID);
        _centeredContentText.text = "(Auto updated)\nCentered content: " + content;
    }
}

```

Assign it to the property "On Centered Content Changed (Int 32)"



It will be like:



Manually Get the Centered Content ID

The other way is to invoke the function `CircularScrollingList.GetCenteredContentID()` to manually get the centered content ID.

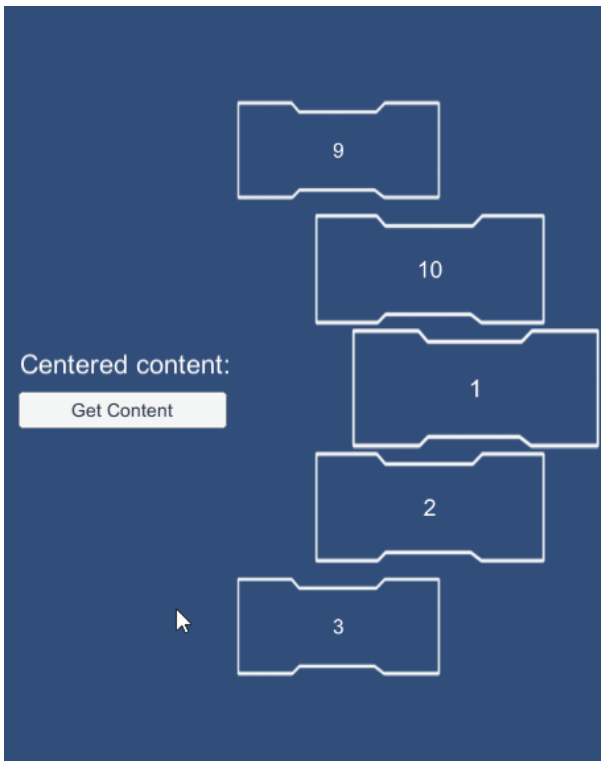
For example, create a function which will update the content of the centered box to the Text, and use a Button to invoke it.

```
using AirFishLab.ScrollingList;

public class DisplayAndSelectExample : MonoBehaviour
{
    [SerializeField]
    private CircularScrollingList _list;
    [SerializeField]
    private Text _displayText;

    public void DisplayCenteredContent()
    {
        var contentID = _list.GetCenteredContentID();
        var centeredContent = (int) _list.listBank.GetListContent(contentID);
        _displayText.text = "Centered content: " + centeredContent;
    }
}
```

It will be like:



Select the Content from Script

The list content could be selected from the script by invoking:

```
CircularScrollingList.SelectContentID(int contentID)
```

Whether the "Centered Selected Box" is on or off, the selected content will always be centered.

If the specified `contentID` is not valid, it will raise `IndexOutOfRangeException`. If the list has no content to display, this function has no effect, no matter what the value of `contentID` is.

Here is an example for iteration through the list contents by selecting each content:

```
using AirFishLab.ScrollingList;

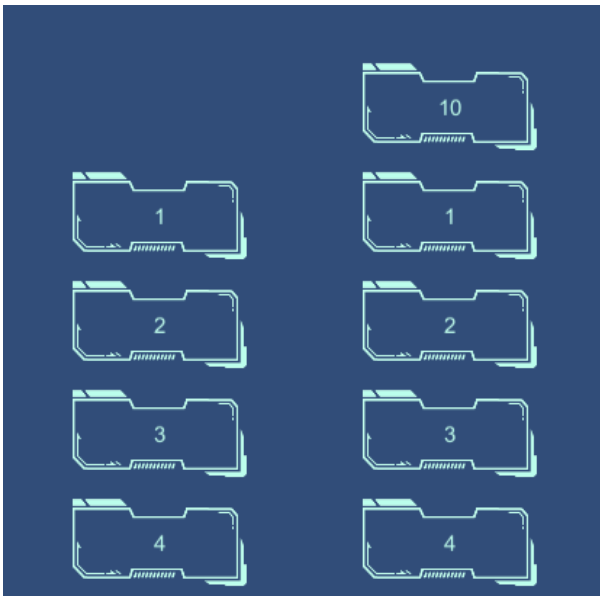
public class ListIteration : MonoBehaviour
{
    [SerializeField]
    private CircularScrollingList _list;
    [SerializeField]
    private float _stepInterval = 0.1f;

    private int _currentID;

    private void Start()
    {
        StartCoroutine(IterationLoop());
    }

    private IEnumerator IterationLoop()
    {
        while (true) {
            _list.SelectContentID(_currentID);
            _currentID =
                (int) Mathf.Repeat(_currentID + 1, _list.listBank.GetListLength());
            yield return new WaitForSeconds(_stepInterval);
        }
    }
}
```

It will be like:



Refresh the List

When any content in the list bank is changed, make the list refresh its displaying contents by invoking:

```
CircularScrollingList.Refresh(int centeredContentID = -1)
```

The boxes in the list will recalculate their content ID and reacquire the content from the list bank.

The `centeredContentID` specifies the ID of the centered content after the list is refreshed. If its value is invalid, the function will raise `IndexOutOfRangeException`.

If the `centeredContentID` is negative, whose default value is -1, the list will use the current centered content ID as the content ID of the centered box (Note that it uses ID, not content). If the current centered content ID is larger than the number of contents, it will be the ID of the last item of them. If there is no content to be displayed before calling `Refresh()`, the ID of the centered content will be 0.

Here is an example for extracting new contents and refresh the list:

```

using AirFishLab.ScrollingList;

public class VariableStringListBank : BaseListBank
{
    [SerializeField]
    private InputField _contentInputField;
    [SerializeField]
    private string[] _contents = {"a", "b", "c", "d", "e"};
    [SerializeField]
    private CircularScrollingList _list;

    /// <summary>
    /// Extract the contents from the input field and refresh the list
    /// </summary>
    /// This function is assigned to a button.
    public void ChangeContents()
    {
        _contents =
            _contentInputField.text.Split(
                new[] {',', ' '}, StringSplitOptions.RemoveEmptyEntries);
        _list.Refresh();
    }

    public override object GetListContent(int index)
    {
        return _contents[index];
    }

    public override int GetListLength()
    {
        return _contents.Length;
    }
}

```

It will be like:

Next List Content:

a,b,c,d,e

Refresh

