UBindr Tutorial 3

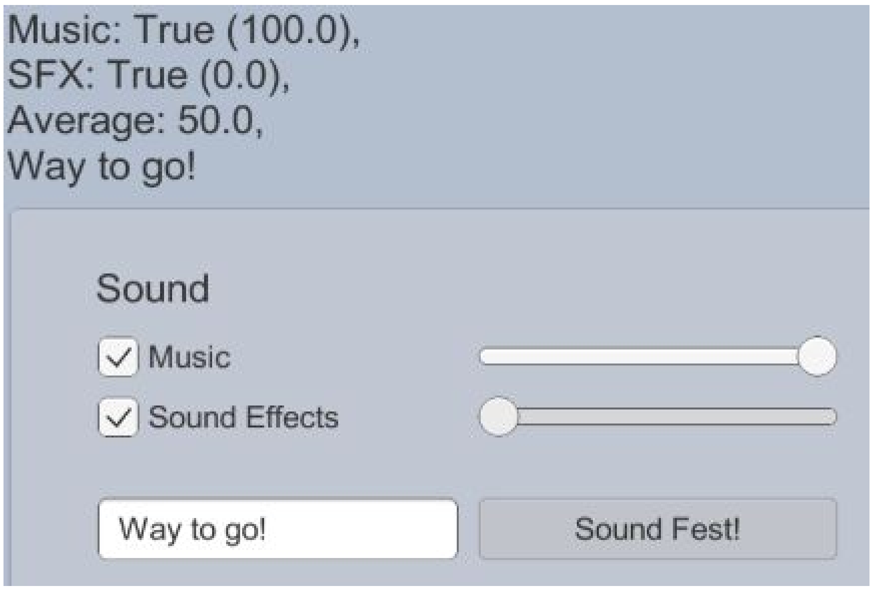
UBindr allows you to publish data from your C# code to the Unity UI with minimal to no code changes. It supports two way binding, so that changes made by the user are passed back to the C# code. It fills some of the same need that Angular, Knockout or React fills for Web applications. It makes building UIs easier and leaves the code cleaner.

Overview
Tutorial 1
Tutorial 2
Tutorial 3
Tutorial Movie

Still Trickier Bindings

In this tutorial, we'll hook up a Dropdown component and we'll create a list of objects where a Prefab is used as a template for each row. And we'll sort the list. Really. It'll be glorious.

I'm assuming you already have a UI that looks like this (from Tutorial 1);

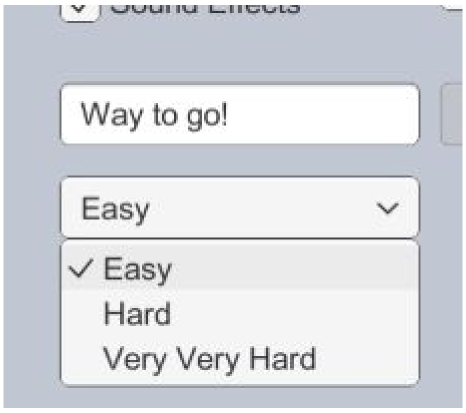


Binding to a Dropdown

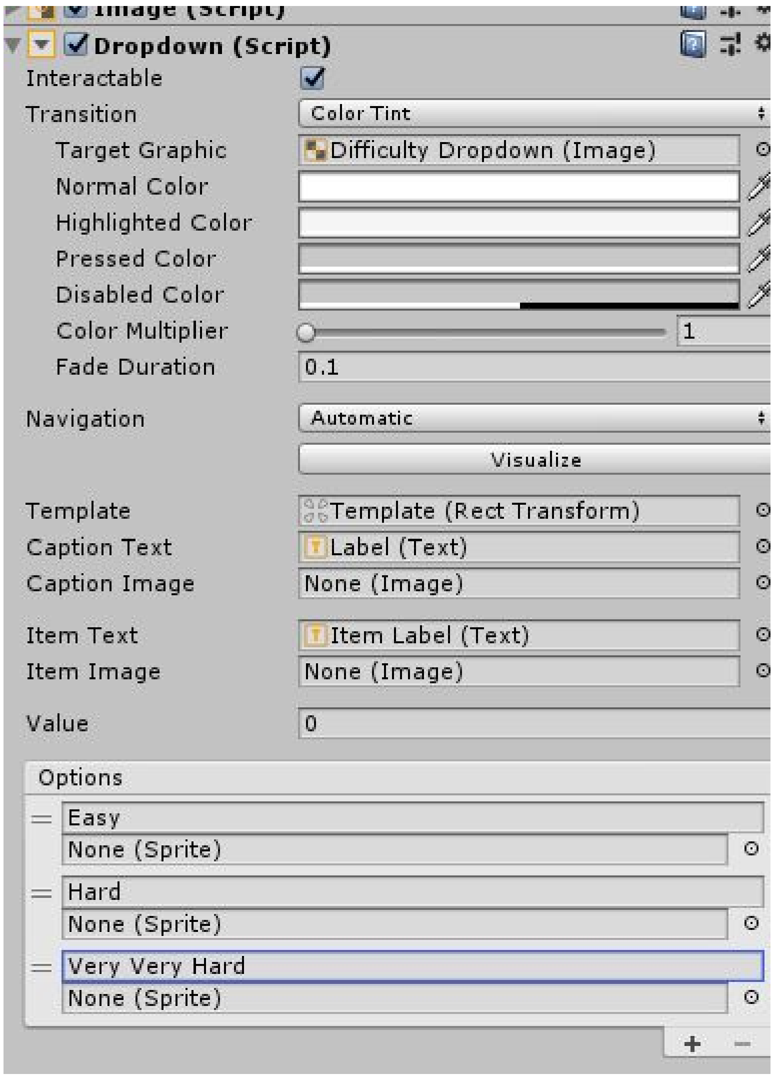
- Create an Dropdown component
  - Select the Setting Panel in the UI
  - Right click
  - Select UI|Dropdown
- Rename the Dropdown "Difficulty Dropdown"
- Place below the Message InputField
- Change the options to

- Easy
- Hard
- Very Very Hard

It should look like this when you run the game;



And the component properties should look something like this;



Dropdown (Script)

Interactable	✓
Transition	Color Tint
Target Graphic	Difficulty Dropdown (Image)
Normal Color	
Highlighted Color	
Pressed Color	
Disabled Color	
Color Multiplier	1
Fade Duration	0.1
Navigation	Automatic
	Visualize
Template	Template (Rect Transform)
Caption Text	Label (Text)
Caption Image	None (Image)
Item Text	Item Label (Text)
Item Image	None (Image)
Value	0

Options

Easy	None (Sprite)
Hard	None (Sprite)
Very Very Hard	None (Sprite)

- Edit the TutorialSettings script

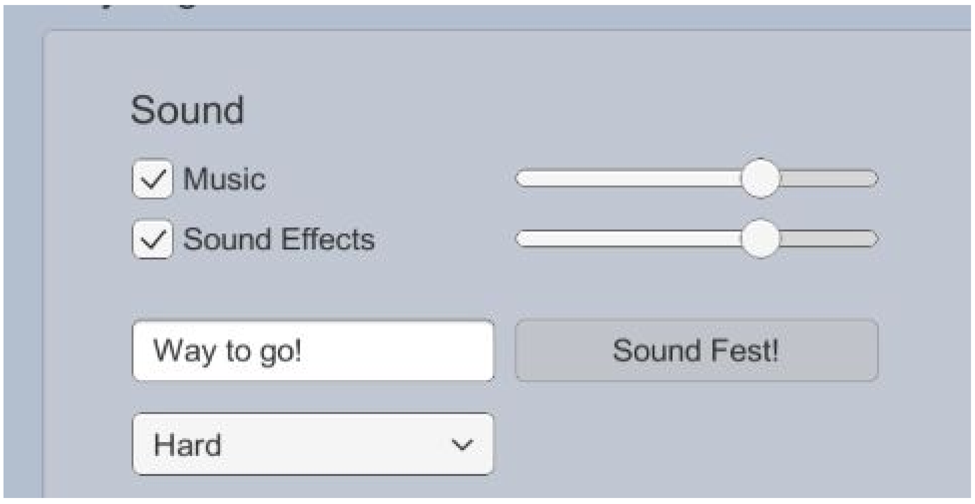
- Create an int field called “difficulty”

The class should look like this;

```
public class TutorialSettings : MonoBehaviour
{
        public bool soundEffectsEnabled = true;
        [Range(0, 100)]
        public float soundEffectsVolume = 70;

        public bool musicEnabled = true;
        [Range(0, 100)]
        public float musicVolume = 70;

        public string message = "Way to go!";

        public int difficulty=1;

        public void SoundFest()
        {
        message = "FIESTA!";
        musicVolume += 1;
        soundEffectsVolume -= 1;
        }
}
```

- Add a UI Binding to the Difficulty Dropdown component
- Set the UI Binding Source Expression to settings.difficulty (use the Binding Helper)

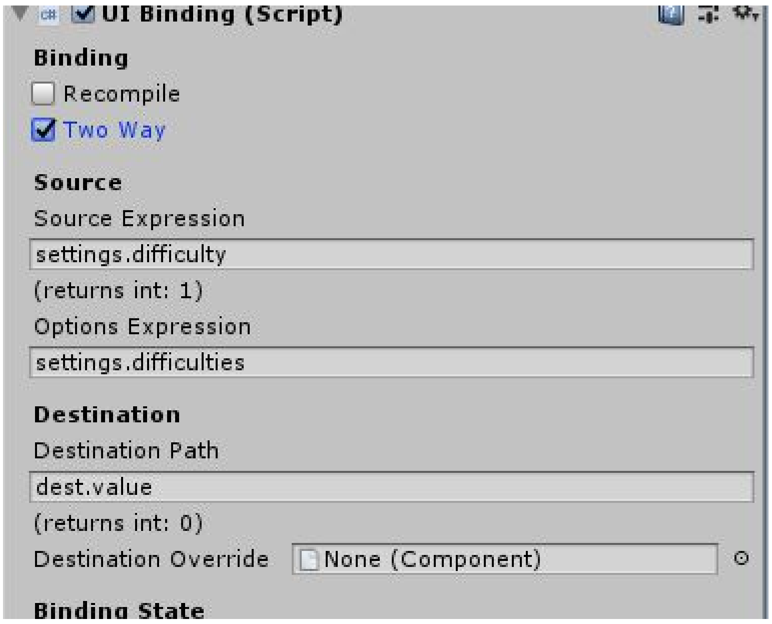
Running the game should look like this (notice how Hard is selected, because we made difficulty default to 1)



Rev 1.0. For support contact mattias.fagerlund@carretera.se.

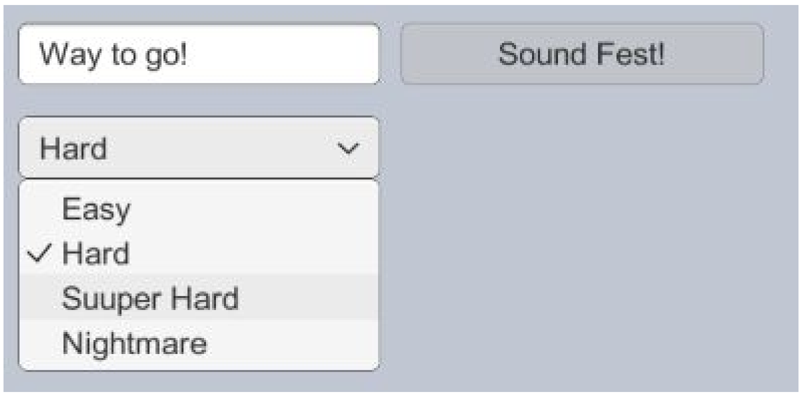
How About Some Options?

- Edit the TutorialSettings again
- Add a field called difficulties, like this;
  public string[] difficulties = new[] {"Easy", "Hard", "Suuper Hard", "Nightmare"};
- Edit the UI Binding for the Difficulty Dropdown component
- Set the Options Expression to “settings.difficulties”
    - You can use the Binding Helper,
    - there’s no button to copy the text to the Options Expression value.
    - click on settings.difficulties (it’s now in the copy/paste buffer)
    - paste the text into the Options Expression value.
- It should now look like this



```
UI Binding (Script)
Binding
[ ] Recompile
[✓] Two Way
Source
Source Expression
settings.difficulty
(returns int: 1)
Options Expression
settings.difficulties
Destination
Destination Path
dest.value
(returns int: 0)
Destination Override   None (Component)
Binding State
```

When you run the game, you should have four difficulties to choose from;



- Edit the SoundFest method, make it add a new difficulty everytime you click the button, something like this;

```
public void SoundFest()
{
        message = "FIESTA!";
        musicVolume += 1;
        soundEffectsVolume -= 1;
        var diff = difficulties.ToList();
        diff.Add("Festive!");
        difficulties = diff.ToArray();
}
```

- Run the game again, the Options list should update as new items are added.

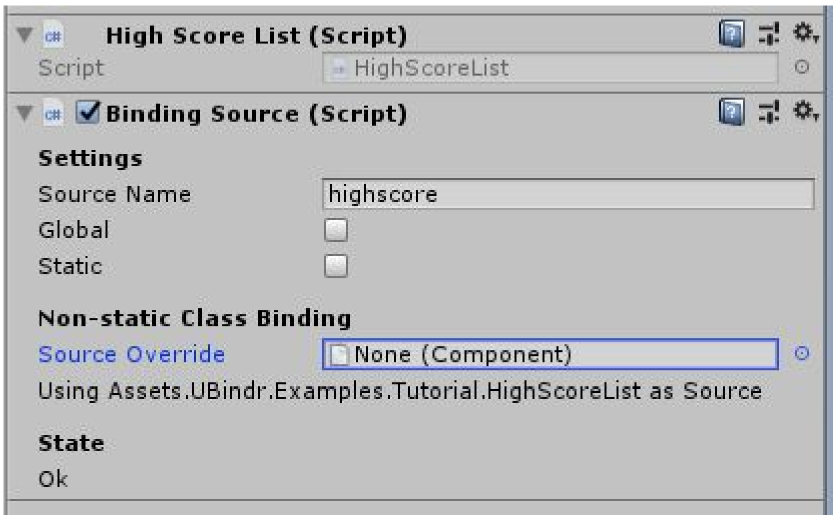
A High Score List

Now we'll add a High Score list to our game.

- Add this script to your game;

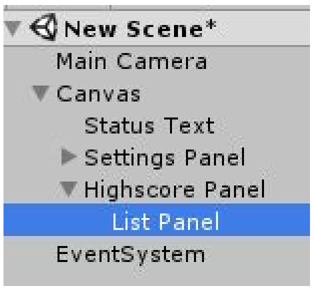
```
public class HighScoreList : MonoBehaviour
{
        public HighScoreList()
        {
        Singleton = this;

        Players = new List<Player>
        {
        new Player{Name="Mr Splendid", Score = 75},
        new Player{Name="Xerxes", Score = 15},
        new Player{Name="Madam GodMode", Score = 100},
        };
        selectedPlayer = Players[0];
        }

        public static HighScoreList Singleton { get; set; }
        public List<Player> Players { get; set; }
        public Player selectedPlayer;

        public class Player
        {
        public string Name;
        public float Score;
        public Settings Settings;

        public void SelectMe()
        {
        HighScoreList.Singleton.selectedPlayer = this;
        }

        public void DeleteMe()
        {
        HighScoreList.Singleton.Players.Remove(this);
        }

        public void ChangeScore(float delta)
        {
        Score += delta;
        SelectMe();
        }
        }
}
```

- Select the Canvas Component
- Add an HighScoreList Component
- Add a Binding Source to the HighScoreList Component
- Change the binding name to highscore

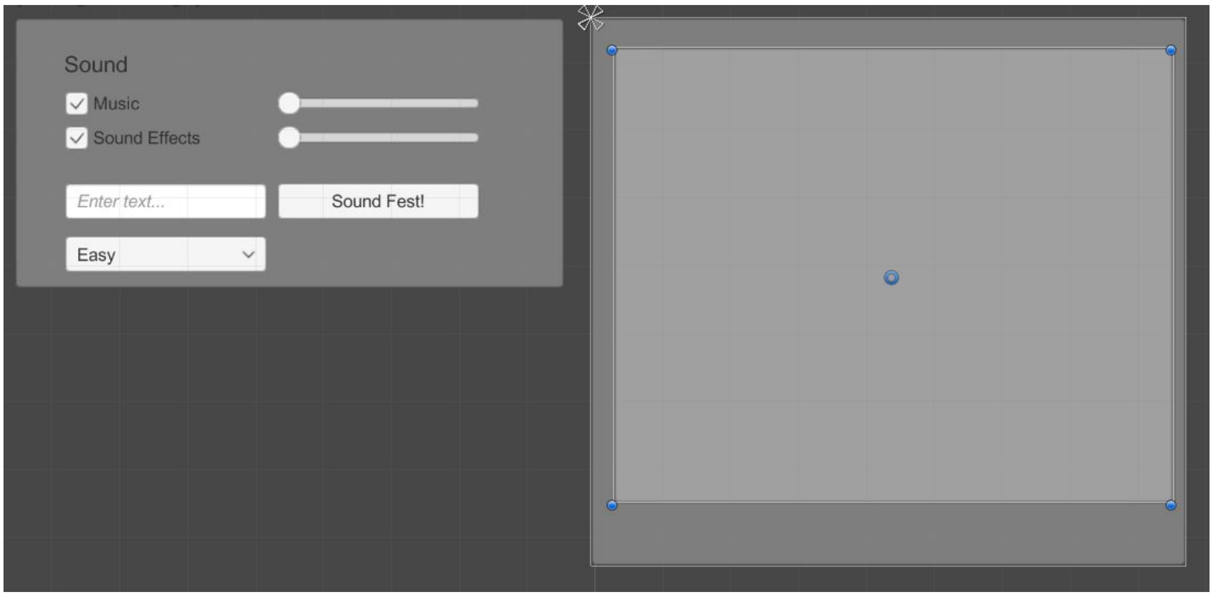


High Score List (Script)
Script: HighScoreList
Binding Source (Script)
Settings
Source Name: highscore
Global: ☐
Static: ☐
Non-static Class Binding
Source Override: None (Component)
Using Assets.UBindr.Examples.Tutorial.HighScoreList as Source
State
Ok

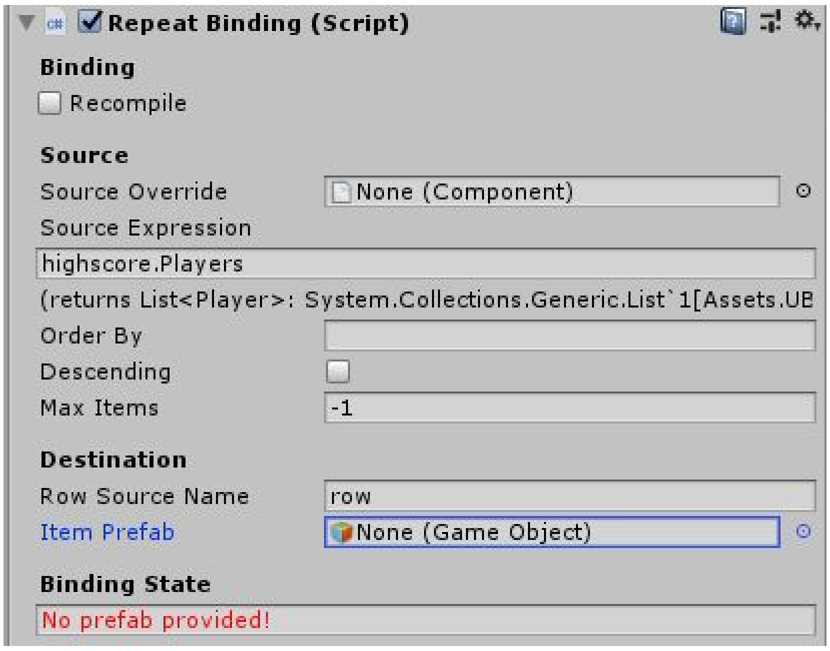
- Create a new Panel and call it Highscore Panel
- Create another Panel inside the Highscore Panel, call it List Panel
- It should look something like this



```
New Scene*
  Main Camera
  Canvas
    Status Text
    Settings Panel
    Highscore Panel
      List Panel
  EventSystem
```



- In the List Panel, add a Repeat Binding
- Bind it to highscore.Players
- Note that there's a "Row Source Name" property that's pre-populated with "row".



Repeat Binding (Script)

Binding

☐ Recompile

Source

Source Override: None (Component)

Source Expression

highscore.Players

(returns List<Player>: System.Collections.Generic.List`1[Assets.UB

Order By:

Descending: ☐

Max Items: -1

Destination

Row Source Name: row

Item Prefab: None (Game Object)

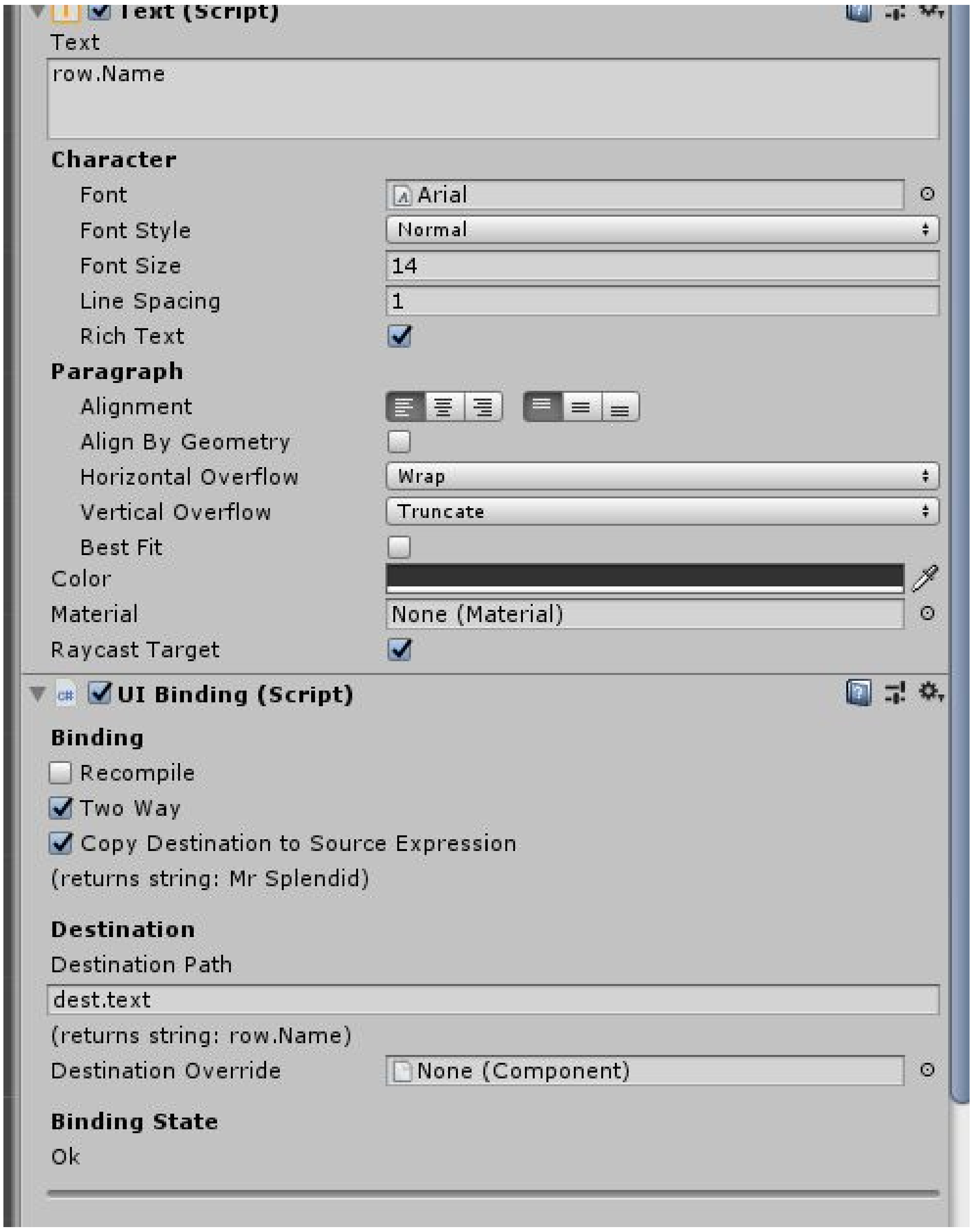
Binding State

No prefab provided!

That's important, because that's how the rows of the list will access the data from the list. You could name it "player", "currentPlayer" or some such, and that would be available for binding. But for now, we're sticking with the text: row.

- In the List Panel,
  - Create another panel,
  - Call it PlayerPrefab
  - Make it 30 pixels high (you may have to change it's anchors to top left first)
- Add a Text component to the PlayerPrefab panel
  - Move one to the left

- Change the name to row.Name
- Add a UI Binding to it
- It should be all happy



Text (Script)

Text

row.Name

Character

Font: Arial

Font Style: Normal

Font Size: 14

Line Spacing: 1

Rich Text: ✓

Paragraph

Alignment

Align By Geometry: ☐

Horizontal Overflow: Wrap

Vertical Overflow: Truncate

Best Fit: ☐

Color

Material: None (Material)

Raycast Target: ✓

UI Binding (Script)

Binding

☐ Recompile

✓ Two Way

✓ Copy Destination to Source Expression

(returns string: Mr Splendid)

Destination

Destination Path

dest.text

(returns string: row.Name)

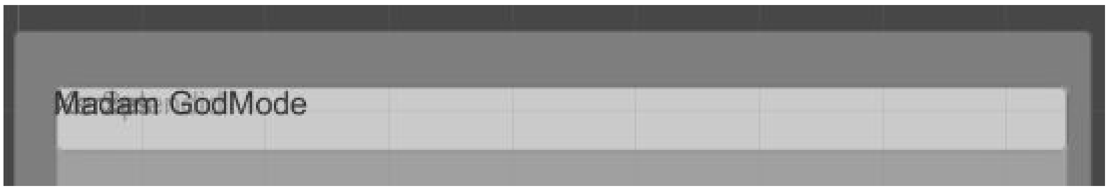
Destination Override: None (Component)

Binding State

Ok

- Make the Prefab Panel into a prefab
  - select it in the hierarchy
  - drag it into your project
- Edit the Repeat Binding
  - Set the Item Prefab to your new prefab (Dragondropit in the Item Prefab box)

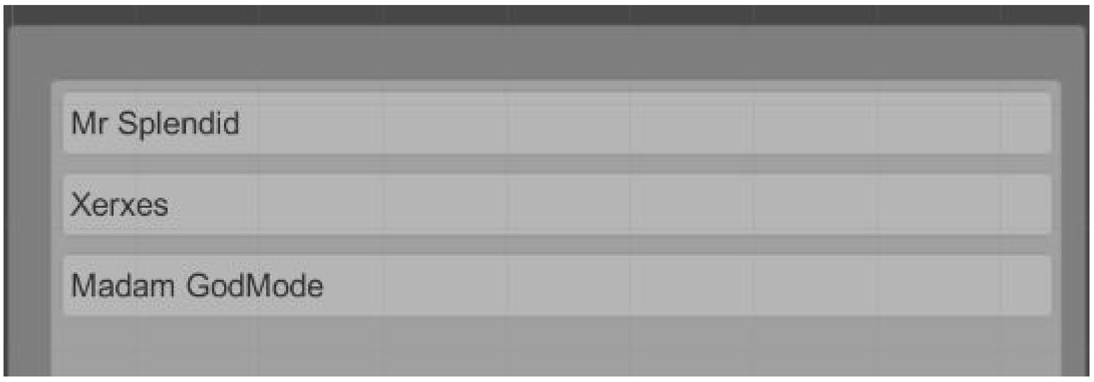
When you run the game, you should get three rows, but stacked on top of eachother



To fix this;

- Select the List Panel
- Add a vertical layout group
- Disable Child Force Expand
- Set Spacing to 5
- Set Paddings to 5
- Adjust the PlayerPrefab so that the row.Name looks nice and centered
- Remember to Apply the changes to the prefab

When you run the game, you should get three rows, but **not** stacked on top of eachother



- Copy the Text field of the Player Prefab
- Move it to the right side of the Player Prefab
- Bind it to "{row.Score:0.00}" (include the quotation marks)
- Remember to Apply the changes to the prefab

You should now have a new field with the score of the players.

Sort The Rows

To sort the rows, you could add a property to the TutorialSettings class which publishes a sorted version of the players array, and bind to that. Something like this;

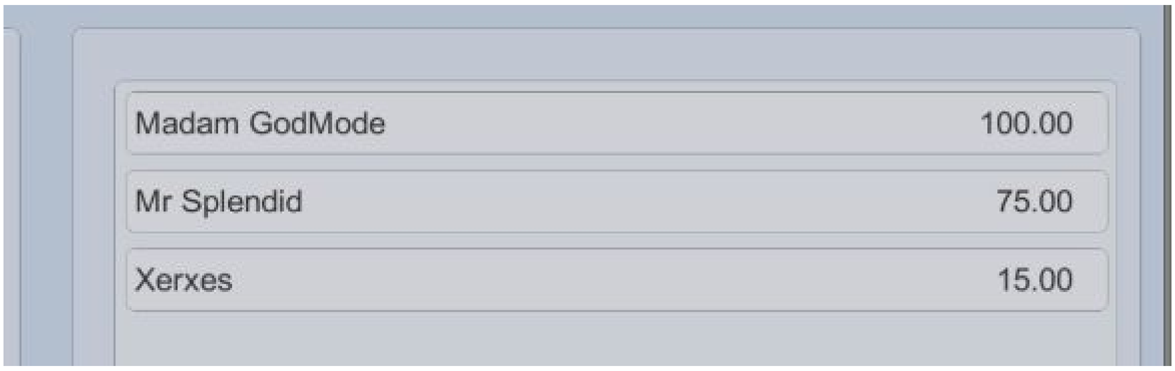
```
public IEnumerable<Player> SortedPlayers { get { return Players.OrderByDescending(x =>
x.Score); } }
```

But let's don't do that! Instead we'll edit the Repeat Binding

- Edit the repeat binding
- Set the Order By field to row.Score
- Determine whether you want it in descending order or not

Rev 1.0. For support contact mattias.fagerlund@carretera.se.

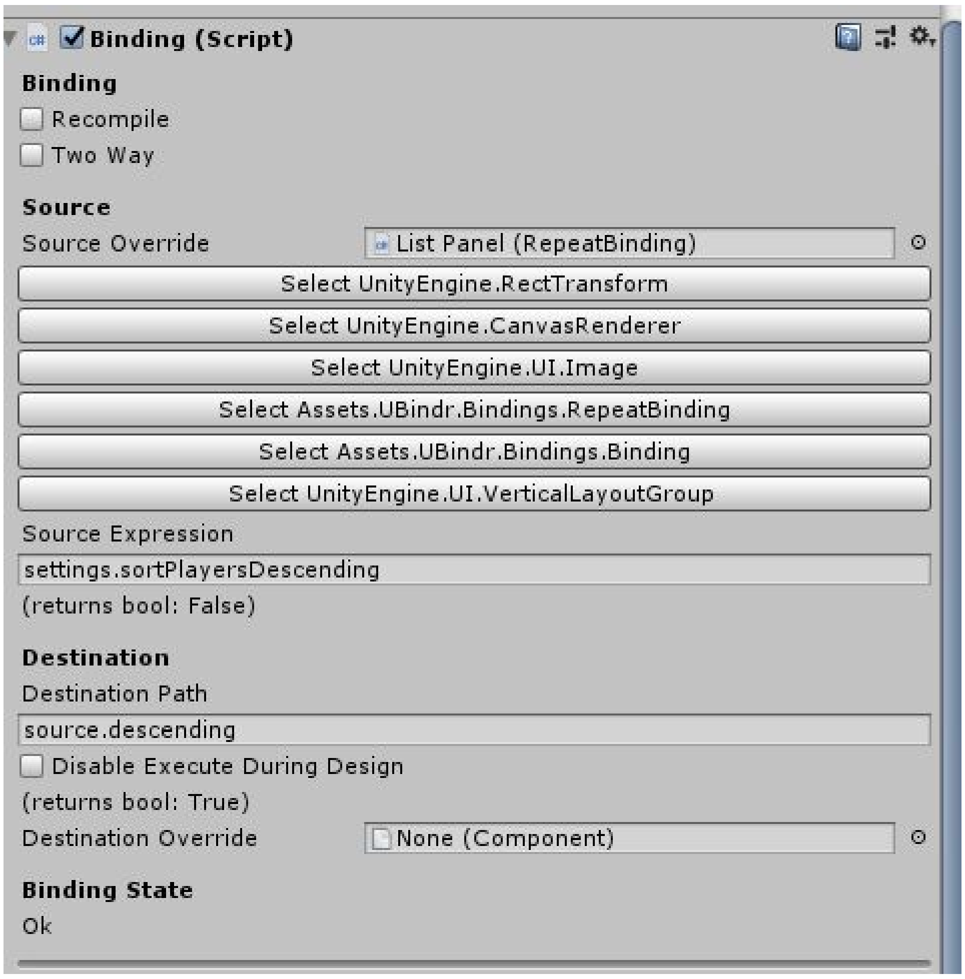
Now when you run the game, you should have a sorded list;


Madam GodMode 100.00
Mr Splendid 75.00
Xerxes 15.00

Control The Sorting

Let’s get the user control over the sorting.There are two main ways of doing this. We could add a boolean property called something like “sortPlayersDescending” to the TutorialSettings class. Then we could bind our Descending Toggle to that boolean. Then we’d create another binding, after the Repeat Binding, that would bind settings.sortPlayersDescending to source.descending (note how we can bind to the source just as well as we can to the destination).

It would look like this;


Binding (Script)

Binding

Recompile

Two Way

Source

Source Override: List Panel (RepeatBinding)

Select UnityEngine.RectTransform

Select UnityEngine.CanvasRenderer

Select UnityEngine.UI.Image

Select Assets.UBindr.Bindings.RepeatBinding

Select Assets.UBindr.Bindings.Binding

Select UnityEngine.UI.VerticalLayoutGroup

Source Expression

settings.sortPlayersDescending

(returns bool: False)

Destination

Destination Path

source.descending

Disable Execute During Design

(returns bool: True)

Destination Override: None (Component)

Binding State

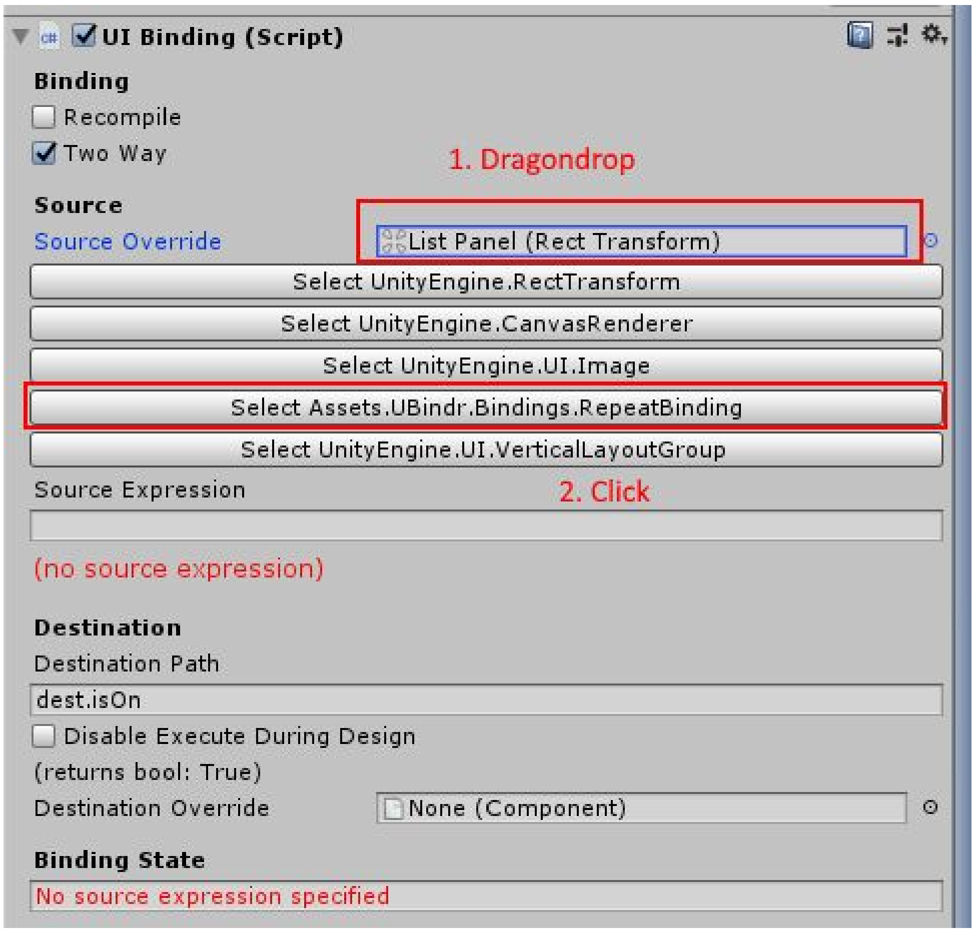
Ok

But the disadvantage of this is that the model (TutorialSettings) doesn't really care about the sort order, and it shouldn't. It complicates the model to help the view - and that's something we should avoid.

So instead, we'll bind our toggle directly to the descending field of the repeat binder.

- Add a new Toggle above the list
- Place it in the Highscore Panel, **not** in the List Panel
- Call it Descending Toggle and set the Title to Descending
- Add a UI Binding to the Descending Toggle
- It should automatically use dest.isOn as the Destination Path
- Now, dragondrop the List Panel, from the hierarchy, to the Source Override field of the UI Binding

It should look like this;


UI Binding (Script)

Binding

☐ Recompile

☑ Two Way

Source

Source Override: List Panel (Rect Transform)

Select UnityEngine.RectTransform

Select UnityEngine.CanvasRenderer

Select UnityEngine.UI.Image

Select Assets.UBindr.Bindings.RepeatBinding

Select UnityEngine.UI.VerticalLayoutGroup

Source Expression

(no source expression)

Destination

Destination Path

dest.isOn

☐ Disable Execute During Design

(returns bool: True)

Destination Override: None (Component)

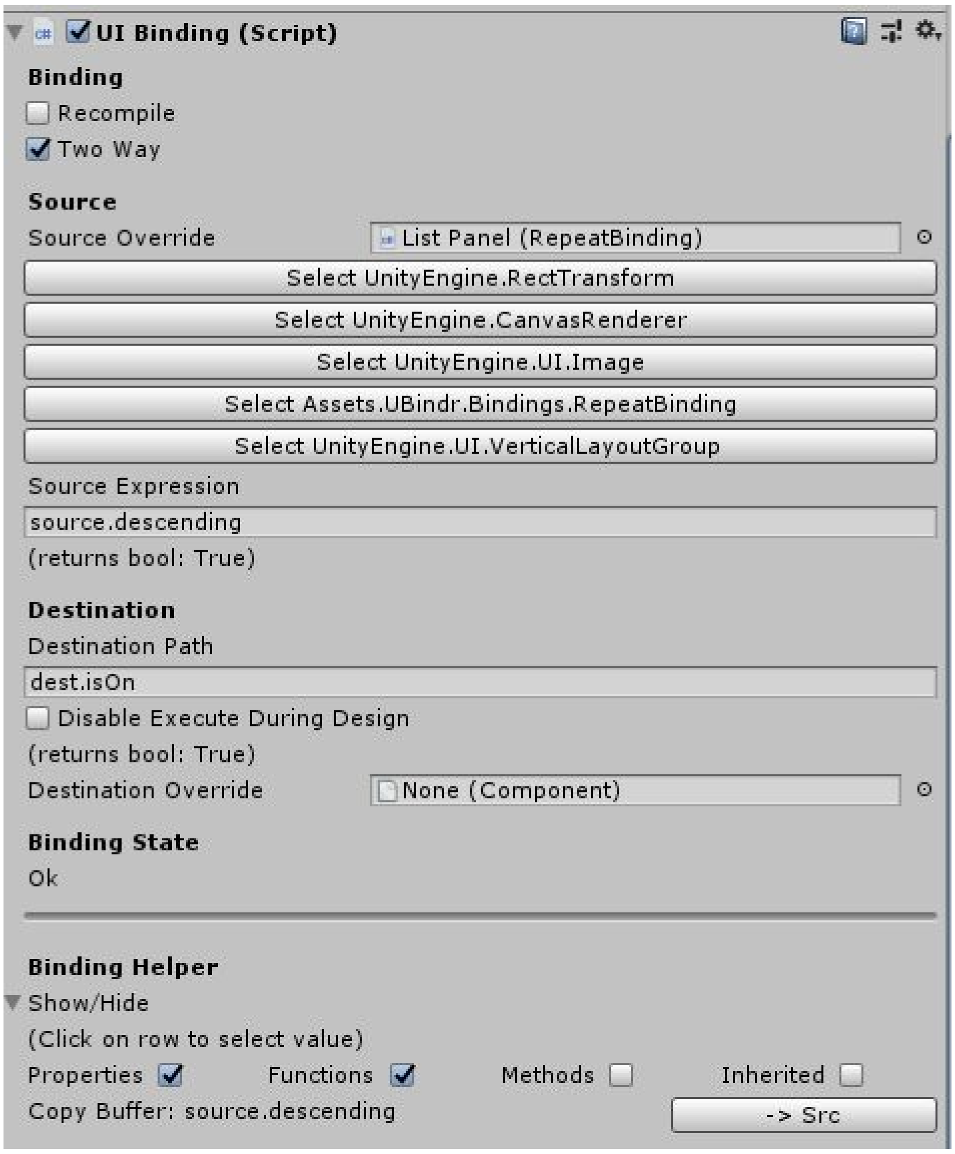
Binding State

No source expression specified

It assumes we want to bind to the Rect Transform on that game object. But we want the Repeat Binding, so click on that button. You should now have a “source” option in the Binding Helper.

- Click “source” in the Binding Helper to expand it
- Click the “descending” to select it
- Click the -> Src field to make it the source of the binding

It should now look like this;



UI Binding (Script)

Binding

- [ ] Recompile
- [✓] Two Way

Source

Source Override: List Panel (RepeatBinding)

Select UnityEngine.RectTransform
Select UnityEngine.CanvasRenderer
Select UnityEngine.UI.Image
Select Assets.UBindr.Bindings.RepeatBinding
Select UnityEngine.UI.VerticalLayoutGroup

Source Expression
source.descending
(returns bool: True)

Destination

Destination Path
dest.isOn
- [ ] Disable Execute During Design

(returns bool: True)
Destination Override: None (Component)

Binding State

Ok

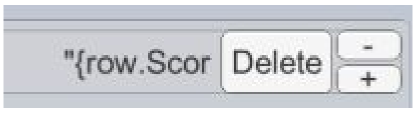
Binding Helper

Show/Hide
(Click on row to select value)
Properties [✓] Functions [✓] Methods [ ] Inherited [ ]
Copy Buffer: source.descending

When running the game, clicking on “Descending” should resort the list.

Let’s Change the Data

- Add three buttons to the player prefab, place them like this



- Add a method binding to each of them
    - Delete => row.DeleteMe()
    - + => row.ChangeScore(1)
    - - => row.ChangeScore(-1)
- Apply the changes to the prefab

If you run the game, you should be able to change the scores and delete the rows. The list should automatically be re-sorted whenever needed.

- Add a new Panel inside the PlayerPrefab
- Make sure it covers the entire Player Prefab
- Rename it to Selected Panel
- Place it first in the PlayerPrefab
- Give it a color that indicates that the row is selected
- Add a UI Enabled Binding to the new panel
- Uncheck “Control Game Object” - we want to control the image Component
- Set the Source Expression to highscore.selectedPlayer==row
- Apply the changes to the prefab

If you’ve set things up correctly, your binding should enable/disable the Image Component when the row is selected. When you click +/-, the clicked row should get selected.

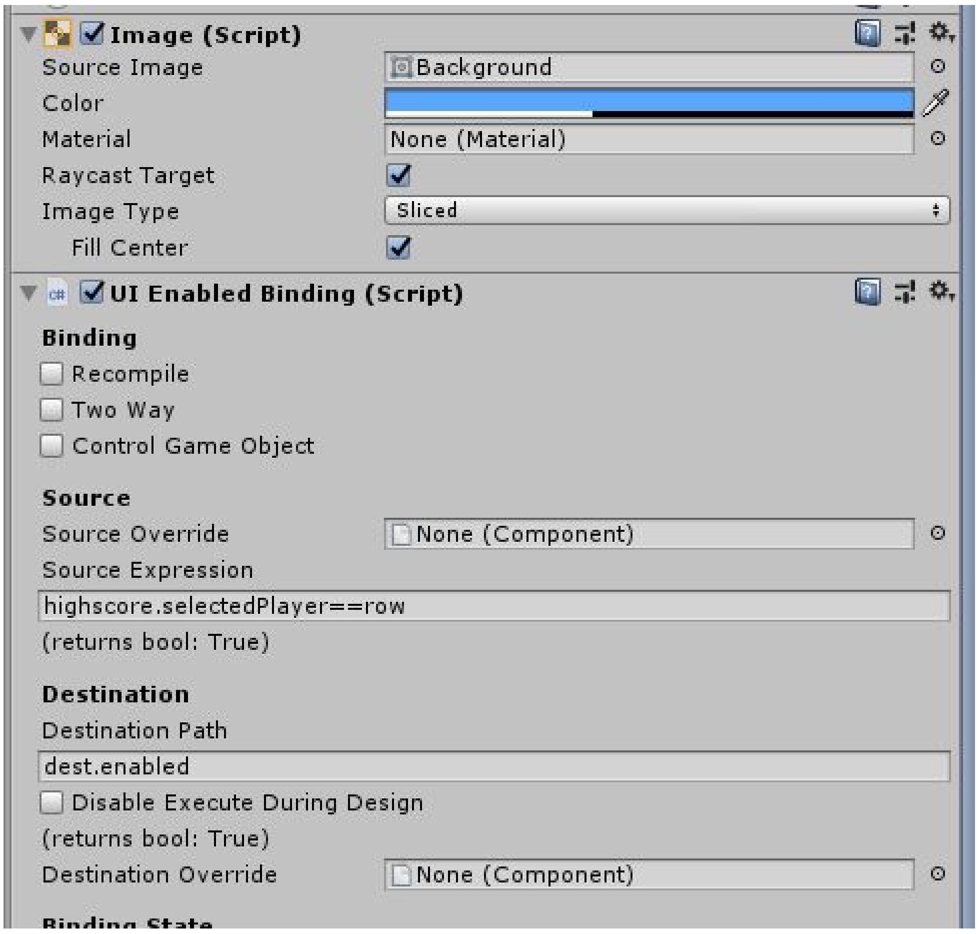
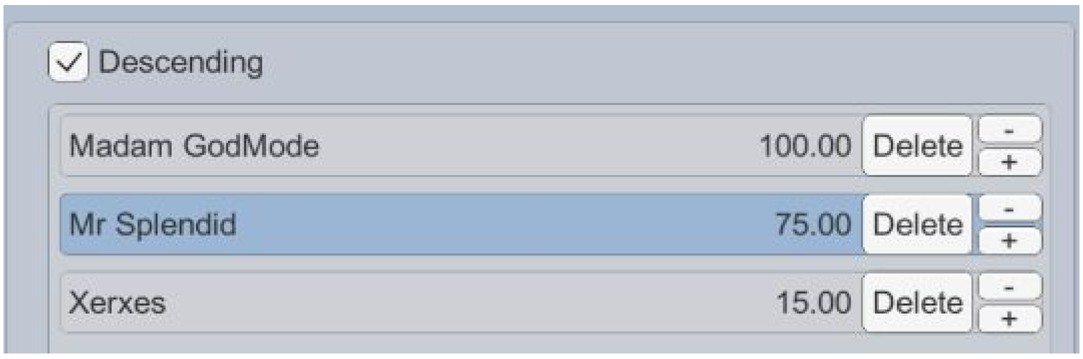


Image (Script)
Source Image: Background
Color
Material: None (Material)
Raycast Target: ✓
Image Type: Sliced
Fill Center: ✓

UI Enabled Binding (Script)
Binding
☐ Recompile
☐ Two Way
☐ Control Game Object
Source
Source Override: None (Component)
Source Expression
highscore.selectedPlayer==row
(returns bool: True)
Destination
Destination Path
dest.enabled
☐ Disable Execute During Design
(returns bool: True)
Destination Override: None (Component)

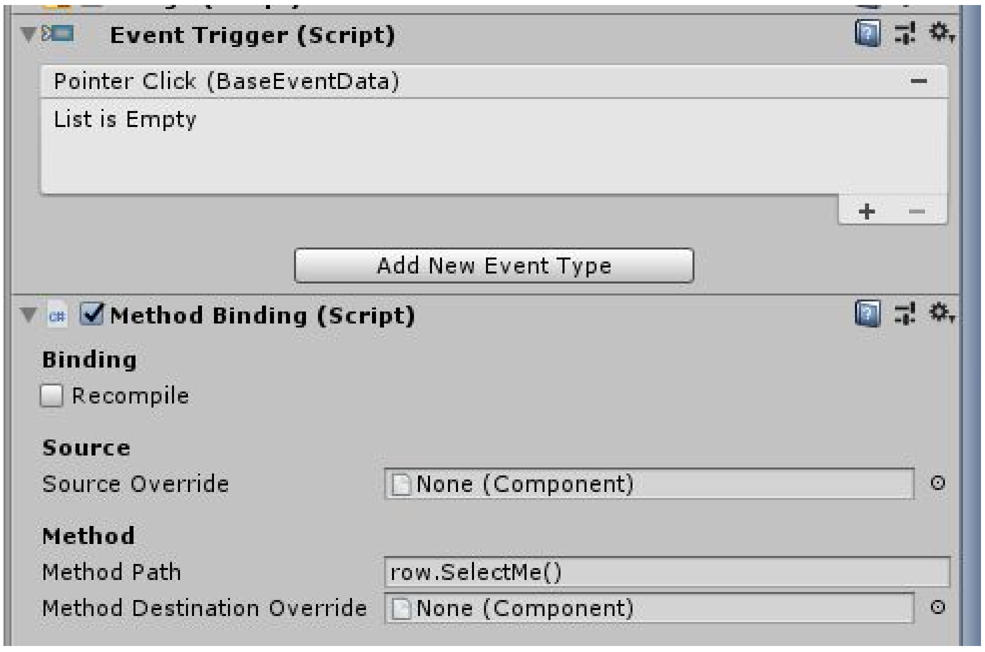


Click to Select!

Let's add the ability to click on a row to select it.

- Add an Event Trigger component
- Add a Pointer Click Event
- Add a method binding
- Bind the method binding to row.SelectMe()
- Apply the changes to the prefab

It should look like this;



Event Trigger (Script)

Pointer Click (BaseEventData)

List is Empty

Method Binding (Script)

Binding

Recompile

Source

Source Override: None (Component)

Method

Method Path: row.SelectMe()

Method Destination Override: None (Component)

Note how the Method Binding comes after the Event Trigger - if you place it after the Image instead, it will not work properly.

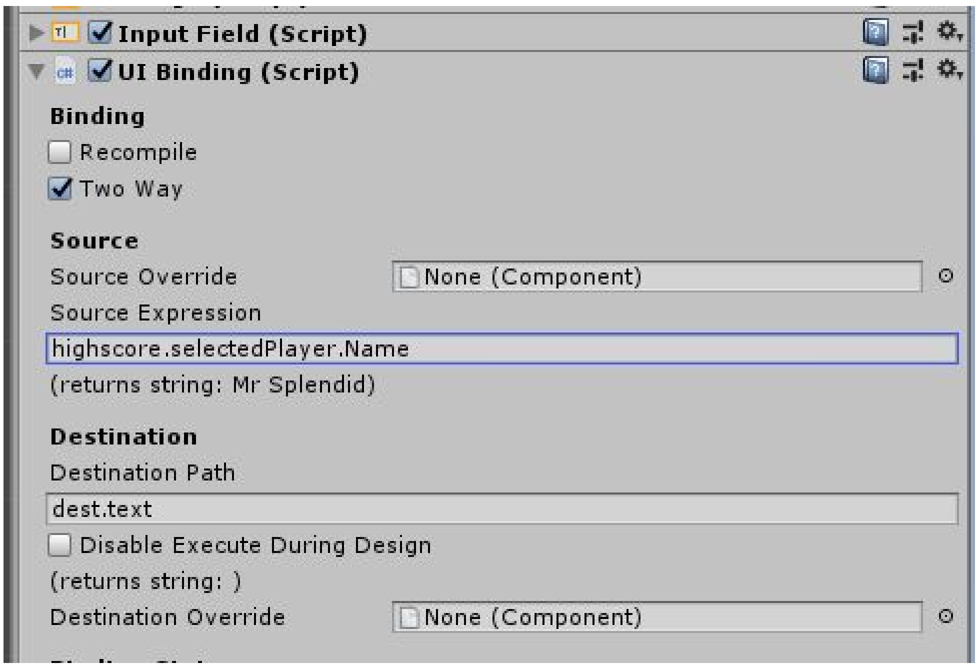
What's in a name?

Let's edit the selected player

- Add an input field to the Highscore Panel
- Place it underneath the List Panel

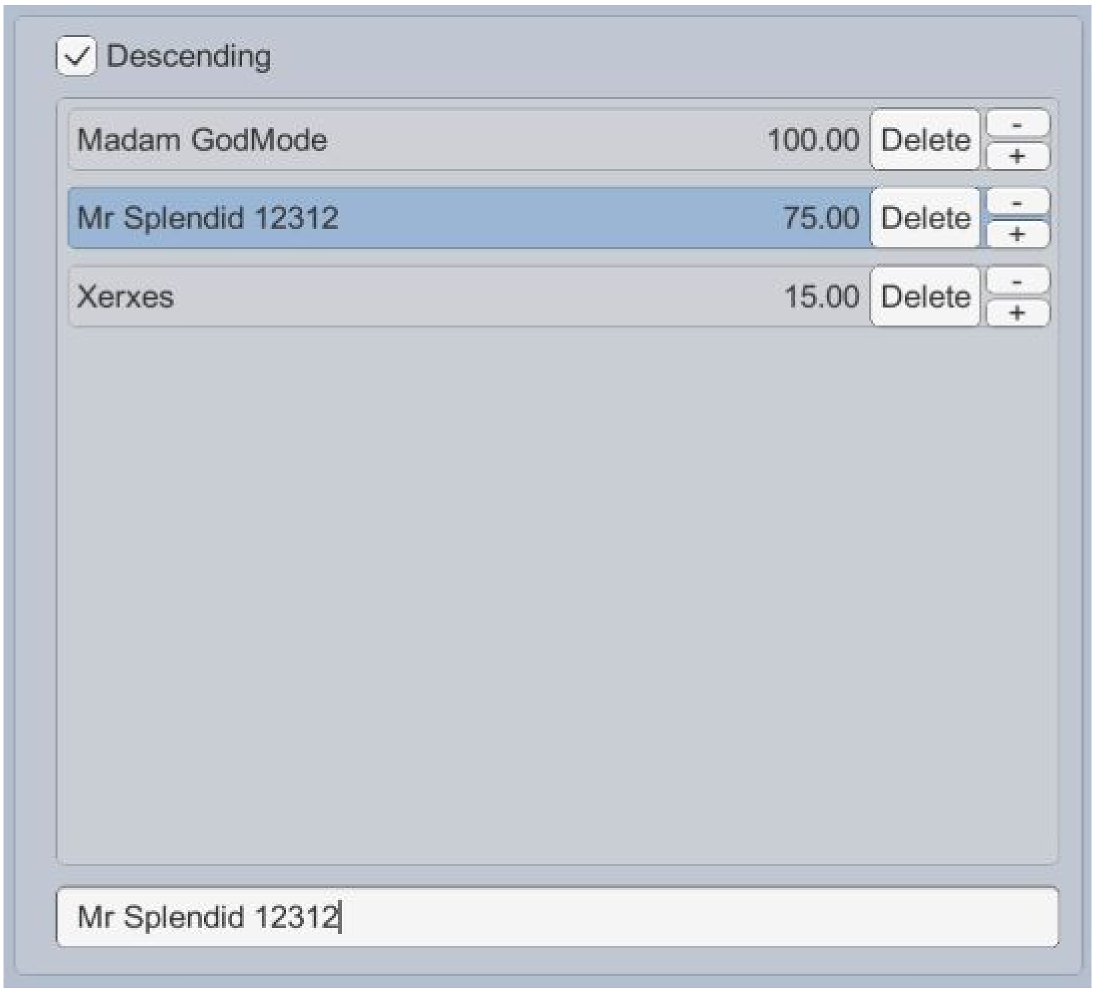
- Name it InputField Player Name
- Add a UI Binding to the InputField Player Name
- Bind it to highscore.selectedPlayer.Name.

It should look like this;



```
Input Field (Script)
UI Binding (Script)
Binding
☐ Recompile
☑ Two Way
Source
Source Override          None (Component)
Source Expression
highscore.selectedPlayer.Name
(returns string: Mr Splendid)
Destination
Destination Path
dest.text
☐ Disable Execute During Design
(returns string: )
Destination Override     None (Component)
```

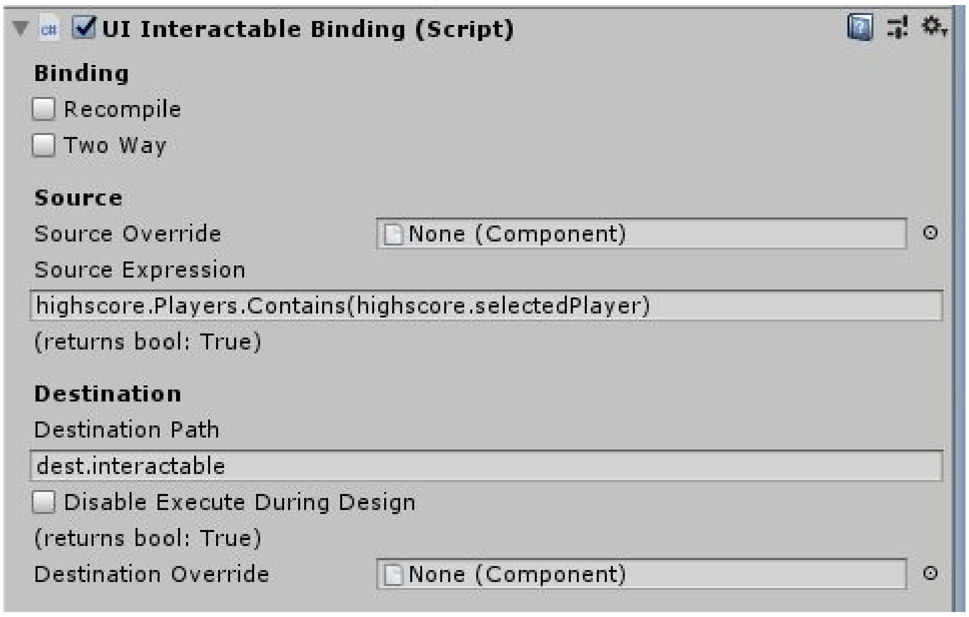
And now you should be able to edit the name of the selected player;



Descending
Madam GodMode 100.00 Delete - +
Mr Splendid 12312 75.00 Delete - +
Xerxes 15.00 Delete - +
Mr Splendid 12312

If you delete the player, you should no longer be able to edit it!

- On the InputField Player Name component
- Add a UI Interactable Binding
- Set the Source Expression to highscore.Players.Contains(highscore.selectedPlayer)


UI Interactable Binding (Script)

Binding

Recompile

Two Way

Source

Source Override: None (Component)

Source Expression

```
highscore.Players.Contains(highscore.selectedPlayer)
```

(returns bool: True)

Destination

Destination Path

```
dest.interactable
```

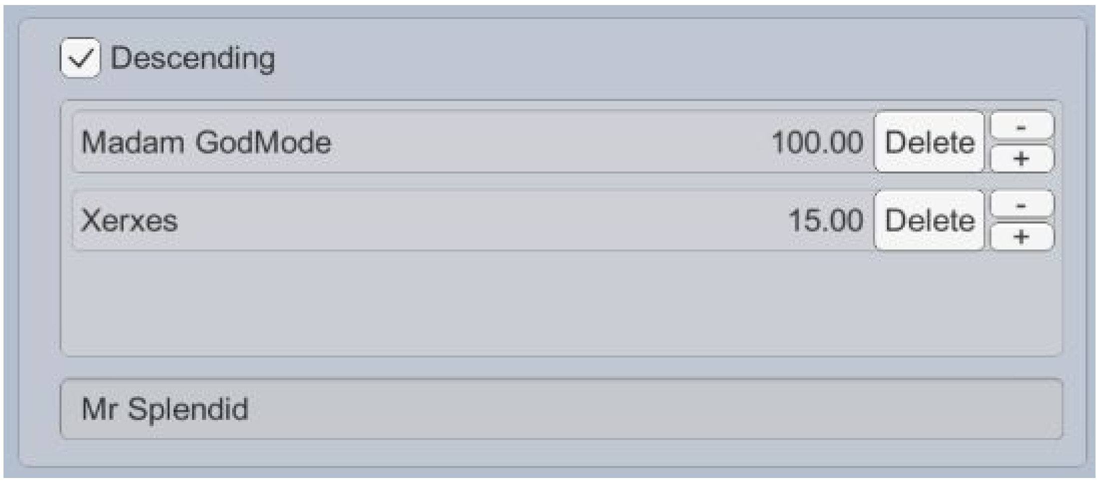
Disable Execute During Design

(returns bool: True)

Destination Override: None (Component)

Deleting the player doesn't set highscore.selectedPlayer to null, so we can't just bind to "highscore.selectedPlayer!=null", which would be neater. If we change our model to update selectedPlayer to null if it's no longer in the list, then that would work too.

Now the input becomes uninteractable when we delete the player;



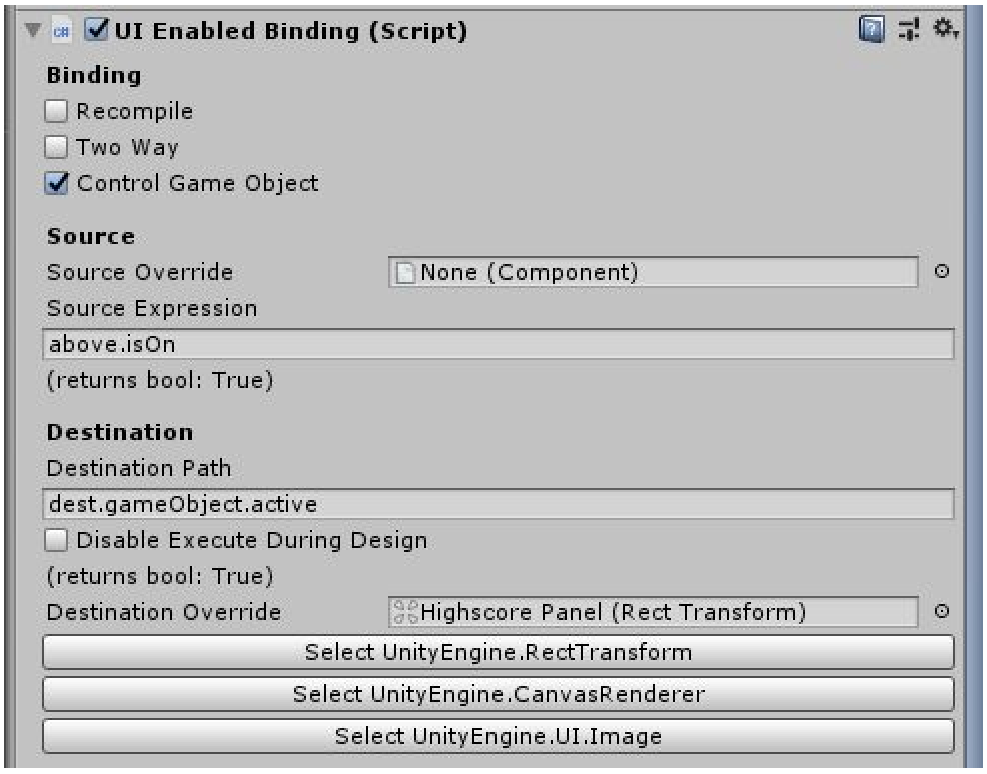
Hide the List

Maybe we don't want the list to be visible all the time, so let's add a toggle to show/hide it

- On the Canvas
- Add a Toggle
- Rename it Highscore Visible Toggle
- Change the Title to Highscore Visible
- On the Highscore Visible Toggle
- Add a UI Enabled Binding
- Dragondrop the Highscore Panel into the Destination Override field

- Bind Source Expression to above.isOn

It should look like this;



UI Enabled Binding (Script)

Binding

- [ ] Recompile
- [ ] Two Way
- [x] Control Game Object

Source

Source Override: None (Component)

Source Expression

above.isOn

(returns bool: True)

Destination

Destination Path

dest.gameObject.active

- [ ] Disable Execute During Design

(returns bool: True)

Destination Override: Highscore Panel (Rect Transform)

Select UnityEngine.RectTransform
Select UnityEngine.CanvasRenderer
Select UnityEngine.UI.Image

Note that we want to control the GameObject - so that all its children are hidden as well.

Toggling the Highscore Visible Toggle should show / hide the Highscore Panel.

That's it for UBindr Tutorial 3. We've covered most of the functionality of the Asset. The asset includes an example project (AllInOne) that demonstrates even more advanced use cases. Have a look at how bindings are used there.