Settings Carousel UI

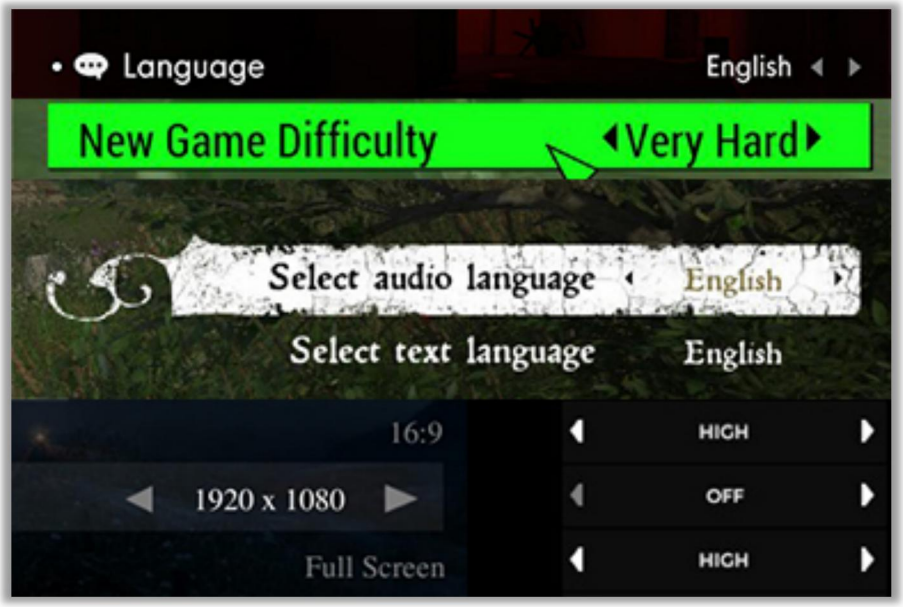
User Guide
1.0.0

By Billion Surya Lioe

Table of Contents

Introduction

This asset is meant to serve as a base for creating carousel UI elements that shows a single option with every step. Even if the input methods differ (the demo included makes use of Unity's built in input manager), the scripts provided were written to remain usable in creating similarly behaving UI elements.



Examples of Carousel settings UI from a number of videogames (Top to Bottom: *The Stanley Parable*, *Fallout 4*, *Kingdom Come: Deliverance*, *Dear Esther*, and *The Forest*).

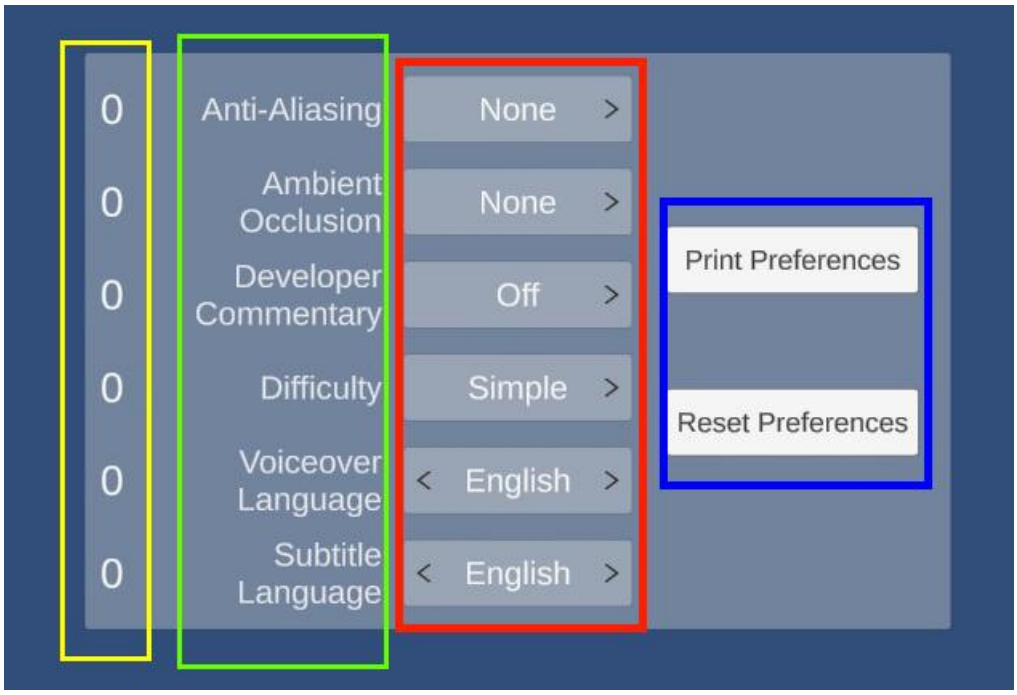
Installation

Simply import the package using the package manager (or manually). On a fresh project, opening the demo scene will prompt Text Mesh Pro to display a window regarding the installation of TMP Essentials. This is because the demo as well as the basic prefab makes use of TMP text components.

As a codebase, the demo scene and any scripts in the demo folder (also under the CarouselUI.Demo namespace) will not be required. Naturally, include your custom scripts as well ;p

Demo

This asset comes with a demo scene that demonstrates the UI elements at work. A screenshot is provided below for reference:



Red

The carousel UI element. Buttons on either side allow swapping between options. This consists of a variant of the base prefab included in the Prefabs folder of the asset. Every interaction saves the new setting as a player preference variable.

Green

Static text that signifies what the UI elements controls. Does not have code functionality.

Yellow

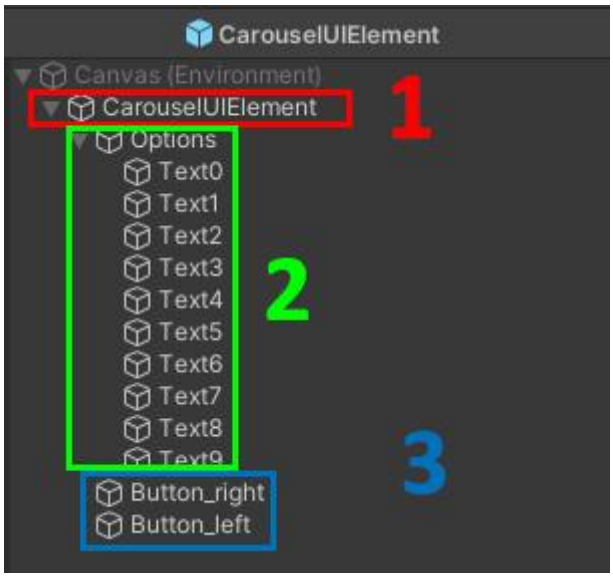
Debug texts that show the saved values controlled by their respective carousel element. These gameobjects will fade after 3 seconds, but are prompted to reappear when the Print Preferences button on the far-right side of the panel is clicked. These work by reading the respective player preference variables the carousel element of the same row controls.

Blue

Control buttons. Print Preferences toggle the debug texts and prints into the console. Reset Preferences reset all saved preferences back to 0, and resets the UI elements in the scene.

Technical Details

UI element Prefab



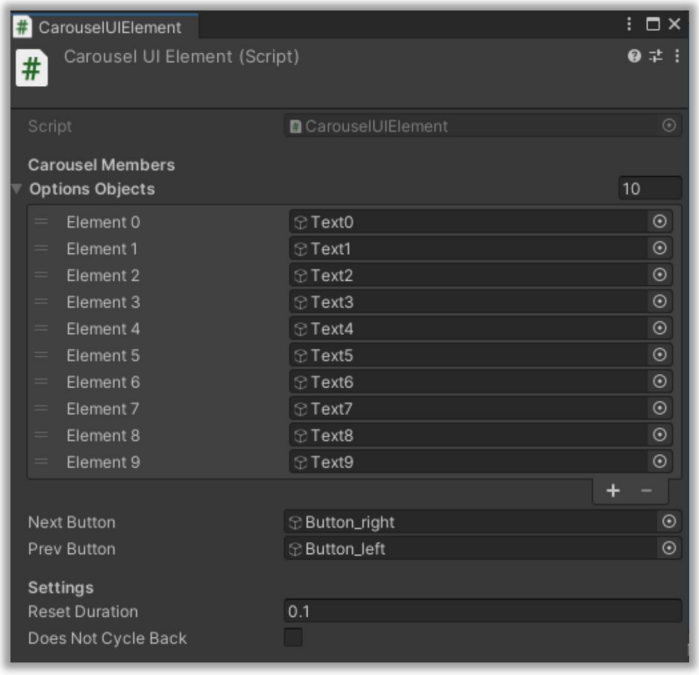
The CarouselUIElement prefab is a basic example for how to implement the CarouselUI script. It consists of:

1) A root panel

2) A set of options in the form of TMPro texts

3) Buttons to advance the selection forward or backward (using Unity's built-in button UI component)

The CarouselUIElement script contained within the root has been set up to use the gameobjects of contained in the prefab, under Options (2).



Additionally, it by default has Reset Duration set to **0.1** (seconds), and Does Not Cycle Back on **False**.

It is not necessary to make use of the prefab directly, as long as the Carousel Element script is properly employed. Additional details are in the next section regarding scripts.

While the prefab is perfectly functional on its own, **it still requires the addition of a Carousel Processor script in order to have an effect on anything**.

Carousel Processor

The processor takes the inputs that the carousel element receives and directs them to other systems. An abstract class is provided as a base class to inherit from. The idea here is to be able to keep all "functional" code that connects the processor to the carousel separate from any additional functionality desired by the user. A processor script that inherits the base is included for use in the demo scene to demonstrate usage and implementation. Another "blank" template script is also provided for use as a quickstart template.

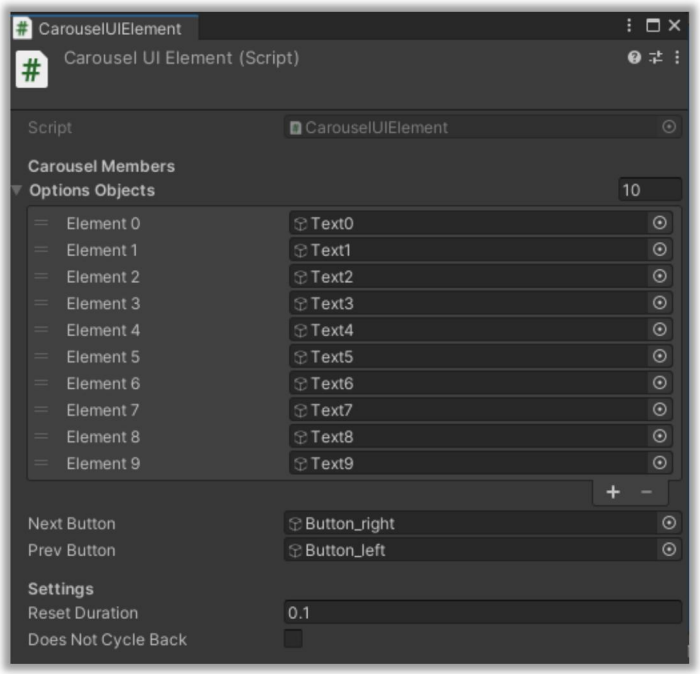
The base script for the processor will not require additional changes in most cases; and the template script can instead be directly modified and saved as a new variant.

Note that the processor does not function in the absence of a CarouselUIElement script, and will create an error message if it fails to find one on the gameobject it is assigned to.

Scripts

CarouselUI Namespace

CarouselUIElement



The functional script for the UI assembly. It controls the UI behaviour and displays which option is currently active. It stores the currently active element index as a private integer.

Variables

- **Options Objects** - Array of options for the carousel element to consider. The number of elements present corresponds to the number of "slides" the carousel will have.
- **Next/Previous Button** - References to the buttons used for interacting with the carousel. The buttons themselves do not have to be of the unity UI button component, as the reference here is simply for the gameobject.
- **Reset Duration** - Cooldown time until the component is able to receive inputs once more.
- **Does Not Cycle Back** - If true, when the carousel reaches either the first or last element, it will hide the previous/next button respectively.

API

- **PressNext**() - Used to increment the current element index by 1.
- **PressPrevious**() - Used to decrement the current element index by 1.
- **UpdateIndex**(int) - Immediately updates the current element index to the input.

CarouselUIProcessor_Base

An abstract class that contains methods that read inputs from the carousel element.

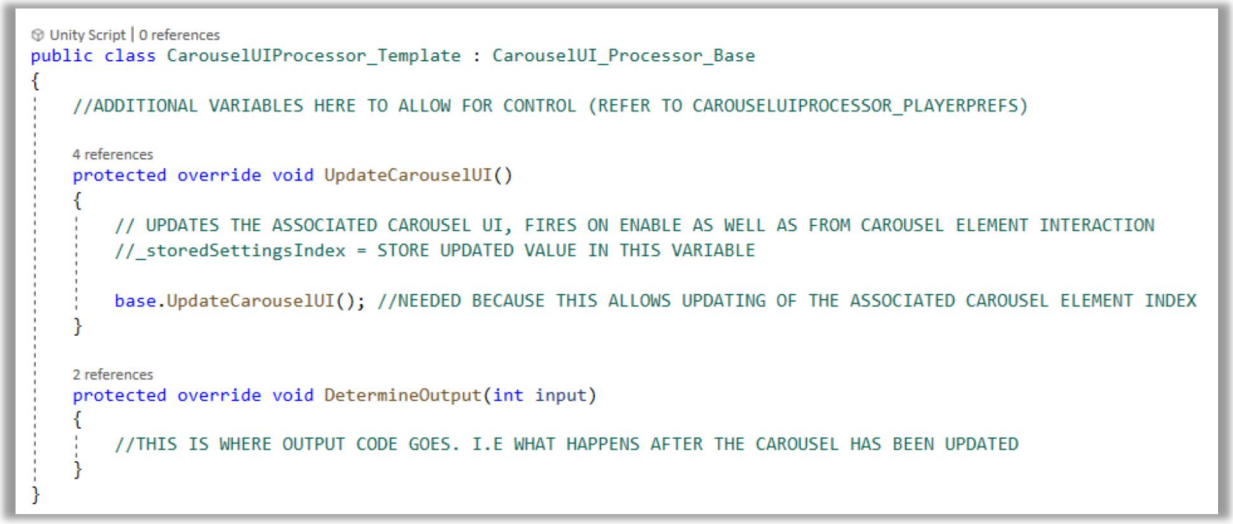
Variables

- AssociatedCarousel - Cached reference to the carousel element that the processor instance would be tracking. Will be automatically cached if available, otherwise will generate an error.

API

- UpdateCarouselUI() - Will update the associated carousel to the current settings in place. Custom code should be placed here to return values from the saved settings and display them on the associated carousel.
- DetermineOutput() - Used by the associated carousel to update the settings that the processor controls. Custom code should be placed here to save into settings.

CarouselUIProcessor_Template

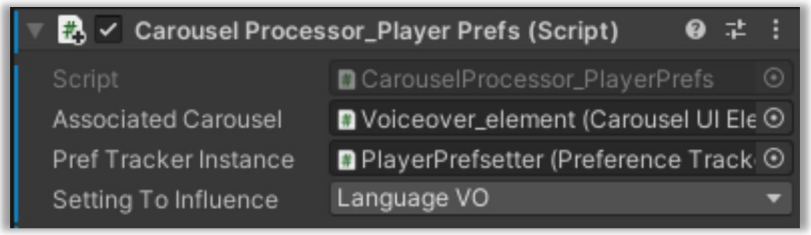


```
Unity Script | 0 references
public class CarouselUIProcessor_Template : CarouselUI_Processor_Base
{
    //ADDITIONAL VARIABLES HERE TO ALLOW FOR CONTROL (REFER TO CAROUSELUIPROCESSOR_PLAYERPREFS)

    4 references
    protected override void UpdateCarouselUI()
    {
        // UPDATES THE ASSOCIATED CAROUSEL UI, FIRES ON ENABLE AS WELL AS FROM CAROUSEL ELEMENT INTERACTION
        //_storedSettingsIndex = STORE UPDATED VALUE IN THIS VARIABLE

        base.UpdateCarouselUI(); //NEEDED BECAUSE THIS ALLOWS UPDATING OF THE ASSOCIATED CAROUSEL ELEMENT INDEX
    }

    2 references
    protected override void DetermineOutput(int input)
    {
        //THIS IS WHERE OUTPUT CODE GOES. I.E WHAT HAPPENS AFTER THE CAROUSEL HAS BEEN UPDATED
    }
}
```

Barebones processor script that inherits from CarouselUIProcessor_Base. Contains overrides for the 2 main methods that needs to be replaced for proper functionality.

CarouselUI.Demo Namespace

CarouselProcessor_PlayerPrefs



Processor script variant used to demonstrate functionality in the demo scene. It makes use of custom enums to control setting variables through player preferences.

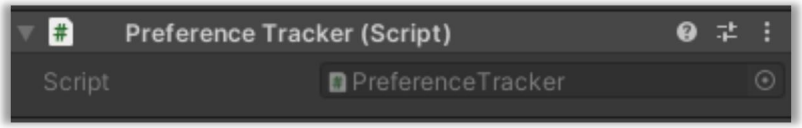
Note: the exposed associated carousel variable will be a feature of any scripts inheriting the base processor script.

Enums_PlayerPrefs

```
40 references
public enum PreferenceEnum { AO, LanguageVO, LanguageSub, AntiAlias, DifficultyMode, DeveloperComm };
```

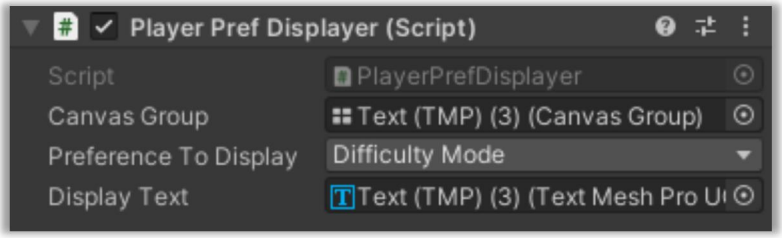
Contains enums for player preference saving. Referenced by a number of the demo scripts.

PreferenceTracker



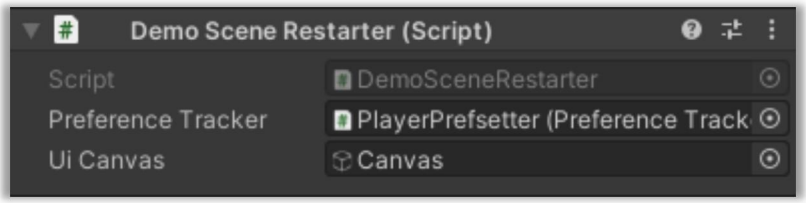
This is a simplified settings manager that controls player preference interactions.

PlayerPrefDisplayer



Works to display the player preference setting set on a text object.

DemoSceneRestarter

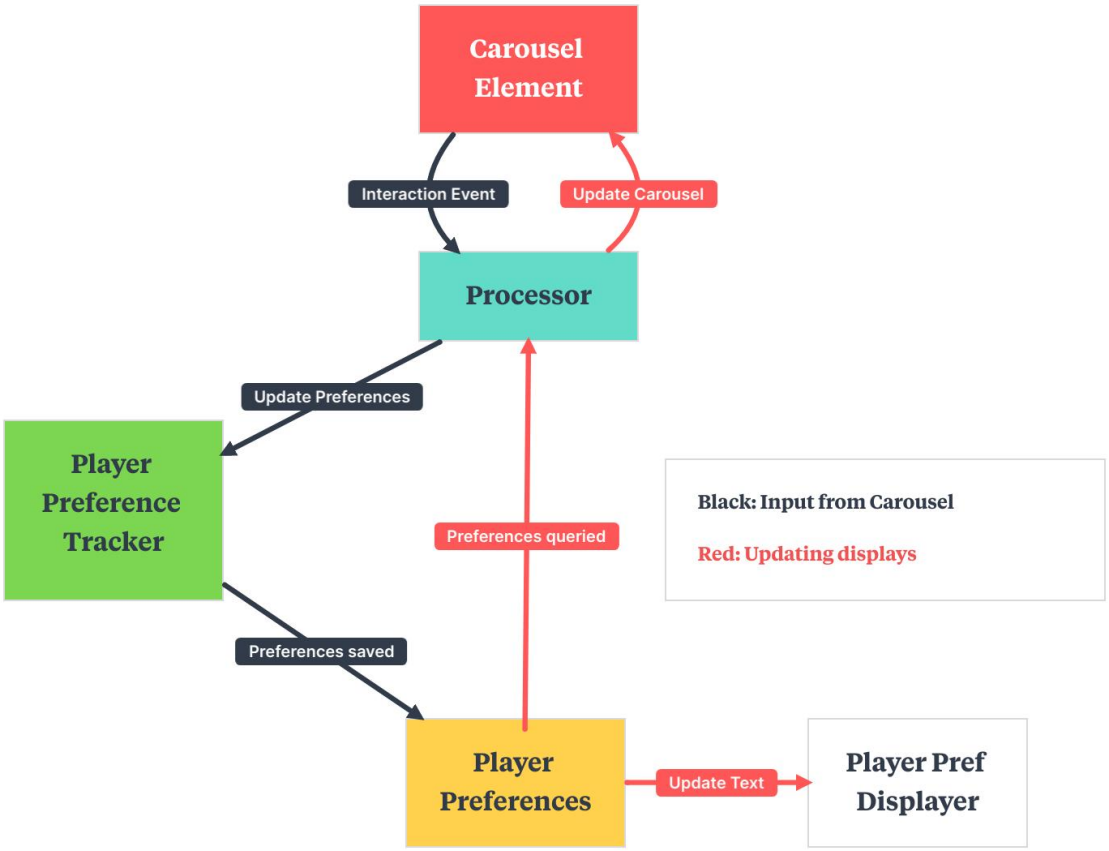


Simple script that quickly disables and enables a gameobject. Used to reset scene displays.

Saving Inputs

While the finer details of this aspect is out of the scope of this user manual, I will briefly describe the player preferences approach used in the demo.

In the demo implementation, player preferences are managed by the PreferenceTracker script - it is referenced by every CarouselUIProcessor in the scene. Each interaction with the Carousel elements triggers the associated processors which in turn sets values according to the setting it controls via the PreferenceTracker script. This change is saved by the built-in player preferences system, which allows the saved values to persist even after the scene is unloaded/application is exited.



When the demo scene is reloaded (or if OnEnable() is called on the processor), the carousel processor script looks up the settings value it controls via the preference tracker. From there, it sets its internal data as well as its associated carousel to the appropriate stored values.

While making use of player preferences is one of the easier ways of keeping settings data, there are some alternatives such as JSON. In that case, a custom script that would control JSON read/writes would be required, but would be easily connected to the carousel system by inheriting the base processor script.

Final Thoughts

This asset can be extended further outside of the scope of the demo. For example, the options for the carousel can use images instead of text, and no actual change to the code would be required to accommodate this. It could also be extended as a “classic” carousel with the options preceding and following being displayed at the same time on either side of the active option. This would require some code modifications however, although not major.

You are more than welcome to build upon the resources provided by this asset to fit your needs.

Tips

1. When making use of the prefab, it would be a good idea to simply remove unrequired references in the options array and hiding the unused gameobjects - instead of unpacking the prefab and deleting the options manually.
2. The visual elements of the UI prefab can be switched out easily as they are standard Unity components for the most part. For the sake of interportability, I had limited the references the demo/prefabs use to ones that come with Unity by default. Custom versions of course do not need to be constrained by this limitation.
3. The way the carousel UI element as well as carousel processor script is designed, values are tracked as integers internally. While it is recommended to stick to using integers for storing settings values when using these scripts, it is not particularly difficult to alter some of the code in either script to enable it to support enums or strings.