FRONTIER UI ASSET

Documentation - v1.0

Content

1. Common Questions & Support (FAQ)

- I need assistance — what should I do?

If this documentation doesn't address your issue, **feel free to reach out to our Discord community directly**. We'd be more than happy to help you sort it out.

- Why can't I change an element? The values remain unchanged.

Some elements are controlled by the **UI Manager**, which applies universal settings across any object with the **UI Manager**' component. To assign unique values to a specific element, simply remove the manager component from it. Or check the appropriate boxes in the component. For example, "**Retain Alpha**" to change the alpha and "**Color For Text/Image**" to set a custom color.

- I'm encountering errors — what might be the issue, and how can I resolve it?

Errors can stem from various causes. First, ensure that you've imported TextMesh Pro via the Package Manager and installed its essentials through **Window > TextMesh Pro**. If problems persist, please contact me with further details.

- Is Frontier UI compatible with URP/HDRP rendering?

Yes, it is. There are scenes available for both URP and HDRP.

- Will you continue to support and update the package?

Absolutely! We plan to provide ongoing updates and enhancements for a significant period, including major revisions when needed.

- Which platforms can I build for?

Frontier UI supports builds for all platforms shown in the Unity build window. However, the demo scenes are preconfigured to run smoothly only on Desktop, WebGL, and Console (primarily Xbox). With a few adjustments, they can be adapted for other platforms as well.

- I'm experiencing performance issues — what could be the reason?

If the slowdown only occurs in the editor, check that the '**DynamicUpdate**' option in the UI Manager is disabled when you're not actively tweaking settings. However, if you notice sluggish performance in your builds too, it may be due to outdated hardware.

2. Quick Setup

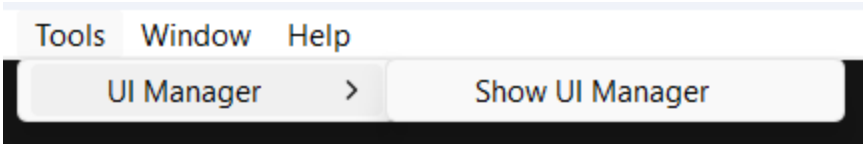
Thank you for choosing our package! If you're looking for guidance on

how to begin, you've come to the right spot.

2.1. Managing the UI System

Want to update the look of your entire UI in one move? The UI Manager has you covered—it lets you modify every UI element at once, so you don't have to adjust them individually.

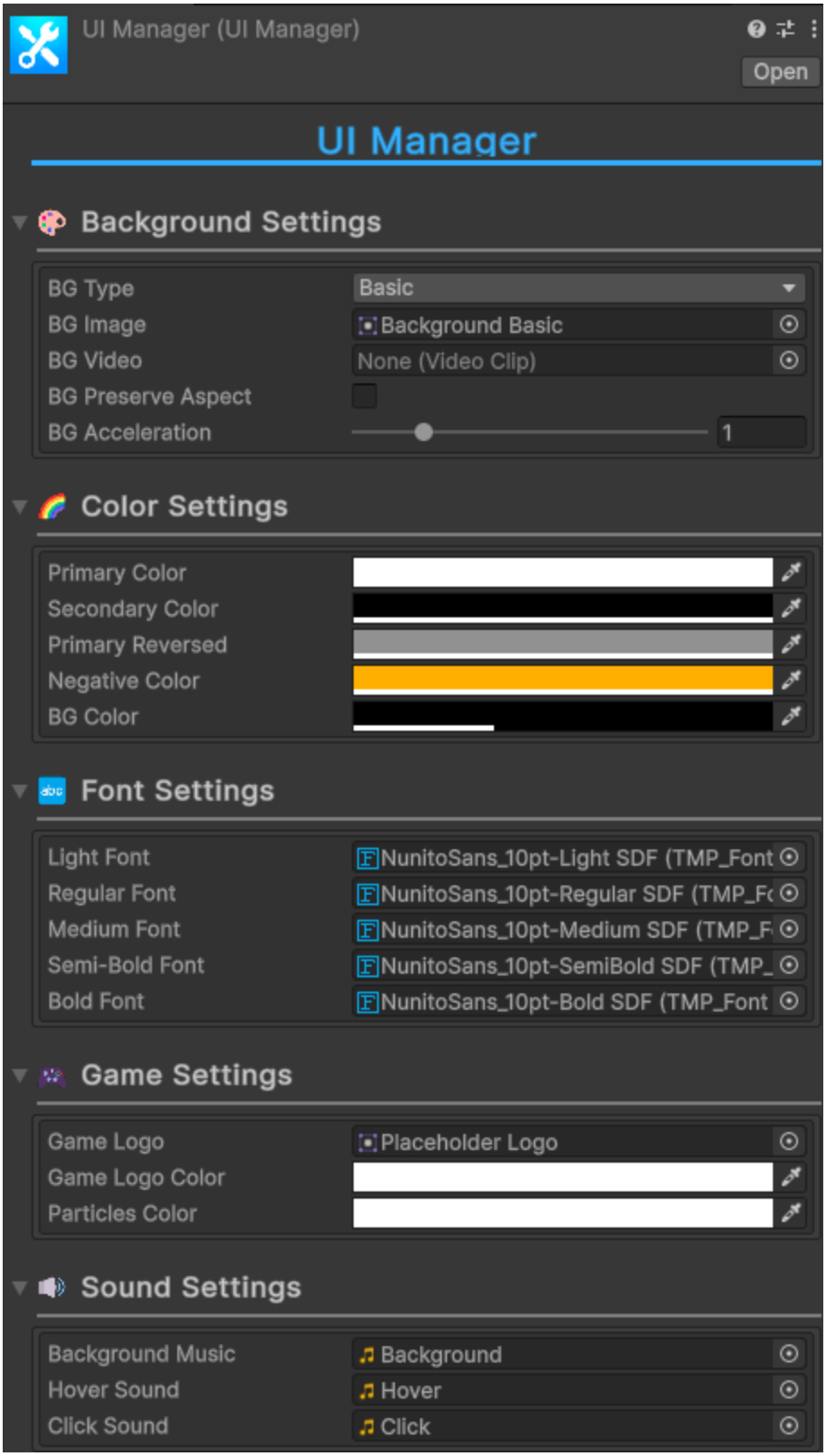
How to Open:
Simply go to **Tools > UI Manager > Show UI Manager** to open the window. From there, expand the different categories and begin tweaking the settings.



Update Values:
When the **Dynamic Update** option is active, the UI Manager will update the UI elements in real time. If it's not enabled, any changes you make won't show up until you run the application. You can disable it to improve performance in the editor, but remember to re-enable it when you need to see your modifications. Note that this feature is not active in builds, so it won't affect your final product.

UI Manager Customization:

You can customize the UI Manager any way you want in UI Manager.


UI Manager (UI Manager)
Open
UI Manager
Background Settings
BG Type
Basic
BG Image
Background Basic
BG Video
None (Video Clip)
BG Preserve Aspect
BG Acceleration
1
Color Settings
Primary Color
Secondary Color
Primary Reversed
Negative Color
BG Color
Font Settings
Light Font
NunitoSans_10pt-Light SDF (TMP_Font
Regular Font
NunitoSans_10pt-Regular SDF (TMP_F
Medium Font
NunitoSans_10pt-Medium SDF (TMP_F
Semi-Bold Font
NunitoSans_10pt-SemiBold SDF (TMP_
Bold Font
NunitoSans_10pt-Bold SDF (TMP_Font
Game Settings
Game Logo
Placeholder Logo
Game Logo Color
Particles Color
Sound Settings
Background Music
Background
Hover Sound
Hover
Click Sound
Click

Important:

Changes made via the UI Manager are universal and will apply to any object that has the UI Manager component attached.

2.2. Getting Started with Demo Scenes

This package comes with several demo scenes you can start experimenting with right away.

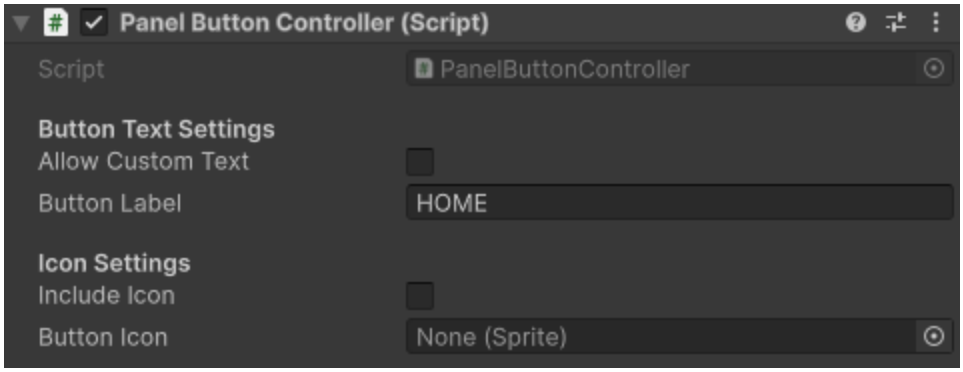
Keep in mind that numerous panels include **Canvas Group** components. To make these panels visible, their alpha value must be set to 1; conversely, setting the alpha to 0 will render them invisible. Essentially, when updating a panel's content, ensure its alpha value is maximized so it can be seen.

3. UI Components

Shift offers a wide variety of UI components. Some of these elements are custom-built, while others utilize the standard Unity UI system. This section focuses on the custom components. For more information about the default Unity elements, please refer to the official Unity tutorials or documentation.

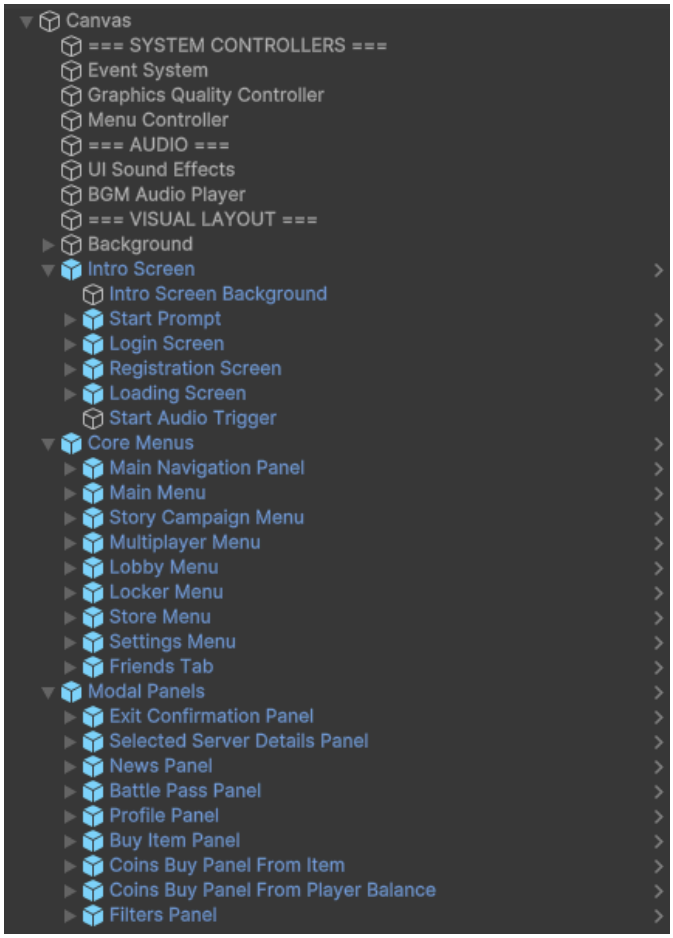
3.1. Updating Content

In many cases, you can modify the content directly in the Inspector. Instead of searching for the specific text object, simply edit its value within the Inspector. This approach can significantly streamline your workflow when working with complex UI elements. For example:



3.2. Modifying UI Elements

Every UI element in this package is constructed using Unity's native UI system, which means you have full flexibility to customize or modify them as needed. The objects are neatly organized into categories, allowing you to quickly locate and adjust the component you're interested in. Once you've made your changes, use the **'Apply'** function on the prefab to update all connected instances at once.

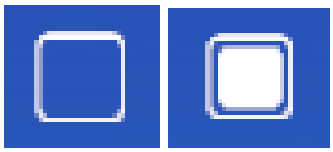


3.3 Custom Components

This package includes some unique UI components that aren't part of the default Unity UI library. The following sections explain these custom elements in more detail.

3.3.1. Switch

This component operates much like a traditional toggle, but with a modernized design and enhanced functionality.



• **Events:**
With the **UI Switch Controller**, you can assign separate events for the Off and On states, which can even be set to trigger at startup.

• **Saving**
By checking the "**Save Value**" option, the switch will retain its last selected state. Each switch must have a unique "**Switch Tag**" to enable proper saving.

3.3.2. Paginator

This component functions as a dynamic pagination system, allowing you to specify the total number of pages, display relevant text, and integrate navigation buttons to seamlessly switch between pages.

• **Events:**
With the Paginator Controller, you can assign custom events to the navigation buttons. Each button can trigger a function that updates the current page number and refreshes the displayed content, ensuring a smooth and interactive user experience.

PaginatorController.PreviousPage/NextPage();
HorizontalScrollButtons.ScrollLeft/ScrollRight();

3.3.3. Slider

The slider component in Frontier UI is based on Unity's built-in slider, but it comes with some added features to enhance its functionality.

• **Enhanced Features:**
These additional features are managed through the Slider Manager component attached to each slider.

• **Saving:**
You can save the last used slider value by enabling the "**Have Saving**" option. Be sure that every slider has a unique "**Slider Tag**" to ensure the value is saved correctly.

3.3.4. Horizontal Option Selector

Think of the horizontal selector as an alternative to a dropdown menu; however, navigation is handled via dedicated "next" and "previous" buttons, which in many cases can offer a more intuitive experience.

• **Items:**
You can add multiple items to the horizontal selector. Additionally, you have the option to assign functions to each individual item.

• **Highlighted:**
When you hover over a field that has a Horizontal Option Selector, it is highlighted by a frame and buttons appear to switch items

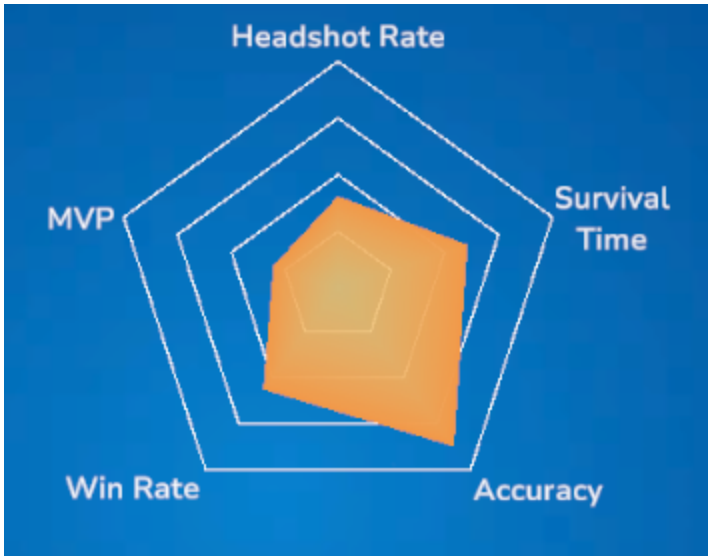
• **Saving:**
To remember the last selected value, simply enable the saving option. Just ensure that each horizontal selector is assigned a unique "**Selector Tag**" for proper functionality.

• **Optional:**
You can use Horizontal Option Selector as a more interesting version of Toggle with On/Off items

• **Usage:**
The horizontal selector can be activated either via an OnClick event or programmatically. For instance, you can call **HorizontalOptionPicker.ForwardClick();** or **HorizontalOptionPicker.PreviousClick();** from your script.

3.3.5 Radar Chart

The Radar Chart component provides a visually engaging way to display multiple statistical values in a single, easy-to-interpret chart. It is particularly useful for comparing different attributes of a player's performance in a game.



• **Customizable Parameters:**
You can change the number of axes. Accordingly, you can change the shape of the chart (5 axes - pentagon, 6 axes - hexagon, and so on). You can set separate values for each axis and the statistics text that corresponds to it. The chart is dynamically updated based on the specified data.

• **Dynamic Scaling:**
The Radar Chart is designed to automatically scale values between 0 and 1, ensuring that all statistics remain proportionally balanced and readable.

• **Color Customization:**
Both the background grid and the statistical shape can be customized with different colors and gradients to match your game's theme.

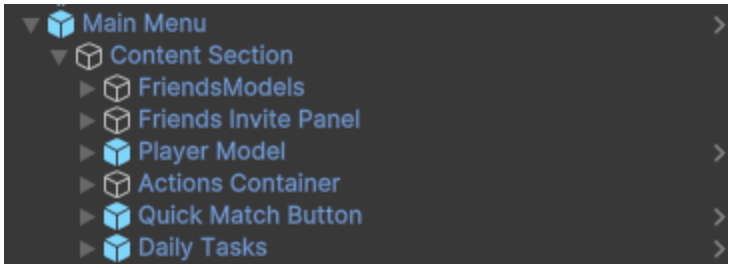
4. Menus & Panels

Frontier includes a variety of UI components — some are custom-built while others rely on Unity's default UI system. This section focuses on the custom panels and windows. For more details on the default elements, please refer to Unity's official tutorials or documentation.

4.1. Modifying Menus

Want to add your own content to an existing menu? It's straightforward:

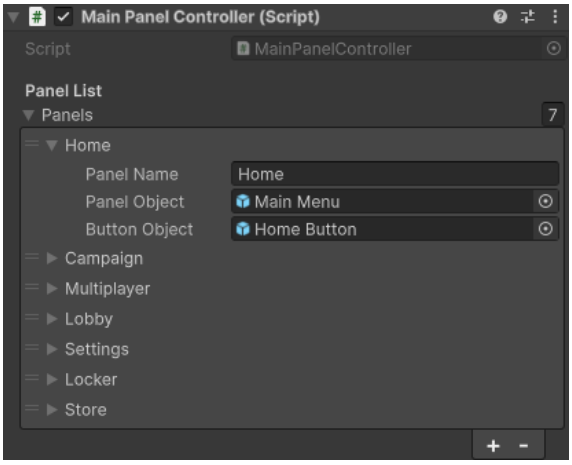
• **How to Add Content:**
Simply drag your content into the **[Menu Name]** > Content Section object hierarchy. As long as your content is a child of the Content object, it will automatically receive the panel's animation.



• **Additional Elements:**
You can also modify the Hotkeys. This object manages the hotkeys for shortcuts.

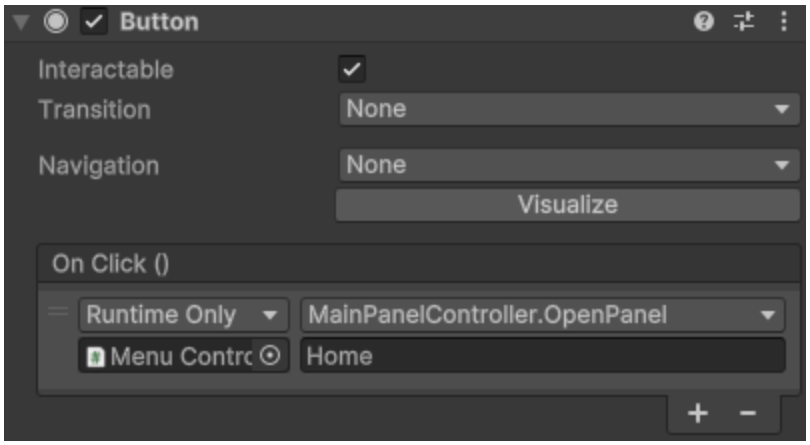
4.2. Adding Menus

Creating a new panel is as easy as duplicating an existing one:
• **Step 1:** Duplicate an existing panel to serve as your new panel.
• **Step 2:** Duplicate one of the existing buttons that open panels, then assign the correct panel index.
If you don't plan to use a button, you can assign a blank object instead.
• **Step 3:** Once you've duplicated these objects, add your new panel to the **Main Panel Controller**. Create a new entry in the 'Panels' list and drag your panel and corresponding button into their designated fields.



After these steps, you can open your panel either through a button's OnClick event or by calling it via your script. For example:

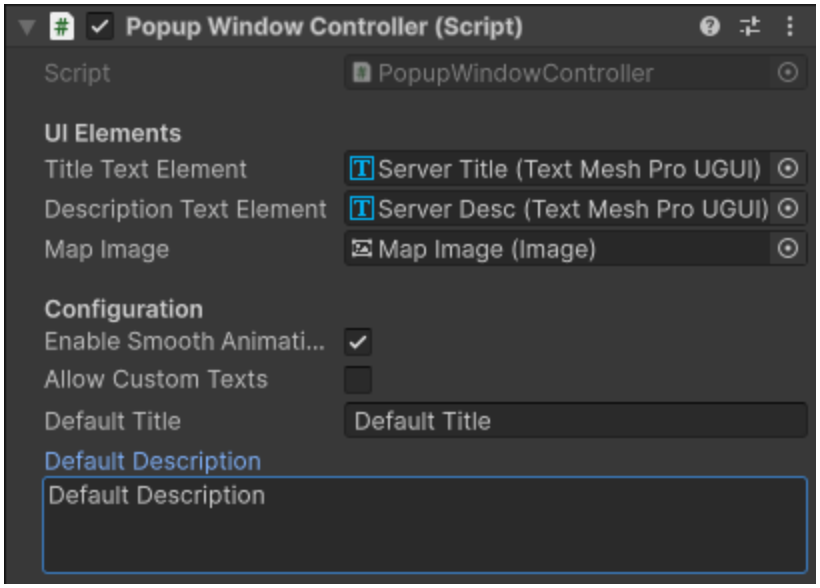
OnClick Example:
Menu Controller > MainPanelController.OpenPanel() > "Your Panel String"



Script example:
PanelControllerVariable.OpenPanel("Your Panel String");

4.3. Popup Dialogs & Modals

To modify the content of a Modal Window, adjust the settings in the **Popup Window Controller** component attached to the modal object.



During runtime, the window will update automatically based on these settings. If you prefer to change these values directly on the main object, simply enable the "**Allow Custom Texts**" option.

To display or hide the modal window, you can trigger it either via an OnClick event or from your script by calling one of these methods:

- **PopupWindowController.ShowWindow();** — to open the window.
- **PopupWindowController.HideWindow();** — to close the window.

5. Operation of specific panels and windows

5.1 Settings & Graphics Quality Controller

The Settings Menu is a crucial part of any game interface, allowing players to tailor the game to their preferences and hardware capabilities. In Frontier UI, we've aimed to make this window flexible, expandable, and easy to use – both for the player and for you, the developer.

Overall Structure

The Settings Menu functions similarly to other main menus in our asset (like the Main Menu, Lobby, or Store). It's built using prefabs and utilizes the Main Panel Controller to manage its content.

The primary components of the Settings Menu are:

- Main Panel Controller: Manages switching between the main settings tabs.
- Tab Panels (Graphics, Control, Audio): These are the individual "pages" containing the relevant settings.
- Tab Buttons: The buttons used to switch between these panels.
- Description Panel: Dynamically displays information about the currently selected setting option.
- Settings Elements (Prefabs): The interactive elements (sliders, selectors, buttons) that make up the tab panels.
- Detailed Component Breakdown

1. Main Panel Controller and Tabs
   - How it Works: Just like how you add new main menus, the Main Panel Controller within the Settings Menu governs which panel (Graphics, Control, or Audio) is currently visible. When a user clicks the "Graphics" button, the controller activates the graphics panel and deactivates the others.

   - How to Find and Configure:
     - Locate the Settings Menu object in your scene's hierarchy.
     - Find the child object that contains the Main Panel Controller.
     - In the Inspector, you'll see the Panels list. Here, you can see which panels (Graphics, Control, Audio) and their corresponding buttons are assigned.
     - You can add new tabs here if you need to expand the settings.

2. Description Panel & Config Button Controller
   - Purpose: To help players understand what each setting does, we've included a description panel. When a player hovers over (or selects with a gamepad) an option like "Vertical Sync," a short text explaining its purpose appears in this panel.
   - How it Works:

    - Each interactive settings element (slider, selector) likely has a script, perhaps named Config Button Controller (or similar), attached to it or its parent object.
    - Within this script, you can specify the description text for that particular setting.
    - When the player interacts with the element (e.g., hovers or selects), this script sends the relevant text to the Description Panel, which then displays it.
- Customization: You can easily change the description text for any setting by finding its object and editing the text within its Config Button Controller or associated script.

3. Settings Elements (Prefabs)
Each individual setting in the window is typically a prefab. This makes the system highly flexible: you can easily add new settings by duplicating existing prefabs or create your own.

The main types of prefabs used are:

- Horizontal Option Selector: Ideal for settings with several fixed values (e.g., Texture Quality: Low, Medium, High). You can add any number of options and assign functions to each.
- Key Content (Key Binding Element): A specialized prefab for control settings.
- Custom Slider: Used for settings requiring smooth adjustment (e.g., Music Volume, Mouse Sensitivity). It's based on Unity's built-in slider but includes enhancements like value saving.

- How to Add/Modify:
    1. To change a setting's text, find its prefab.
    2. Locate the text element and change it, or if the prefab has a dedicated controller, edit the text directly in the Inspector.

3. To add a new setting, duplicate an existing prefab, rename it, change its parameters, and position it on the panel. Don't forget to link it to the relevant systems (like the Graphics Quality Controller or Audio Mixer).

4. Graphics Quality Controller & Audio Mixer
   - Graphics Quality Controller: This script acts as the "brain" for graphics settings.
     - How it Works: It holds references to the UI elements (sliders, selectors). When a player changes a value on a UI element, that element calls a corresponding function in the Graphics Quality Controller (e.g., SetResolution, VsyncSet). The controller then applies these changes to Unity's quality settings.
     - Configuration: You generally just need to ensure that each slider and selector is correctly linked to its respective field in the Graphics Quality Controller within the Inspector.

   - Audio Mixer: Audio settings work via the standard Unity Audio Mixer.
     - How it Works: We have a mixer with groups (Master, SFX, Music). The volume sliders in the settings window adjust the volume levels of these groups within the mixer.
     - Configuration: Ensure your Audio Sources have their Output field assigned to one of these mixer groups (SFX or Music) for the volume sliders to affect them.

How to Add a New Setting (Example: Field of View)

1. Create UI Element: Duplicate a setting prefab. Name it FOV_Setting. Change its label text to "Field of View".
2. Add Logic:
   a. If it's a graphics setting, add a new function to the Graphics Quality Controller, like SetFOV(float value). This function should change the FOV of your game camera.

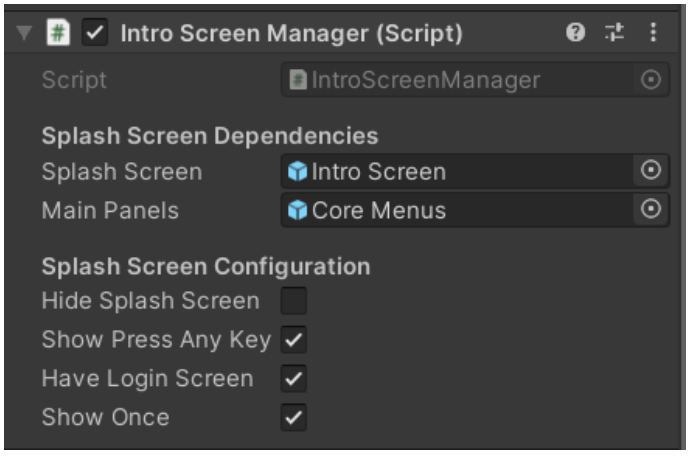
   b. Add a slider change to the Graphics Quality Controller script, connect the necessary method to it on OnValueChanged. Connect the desired slider to the variable
   c. If it's another type, add the logic to the appropriate manager.
3. Add Description: Configure the Config Button Controller for FOV_Setting, adding text like "Changes the camera's field of view”.

Also, you can find some **useful information** in **10. Quality Settings**

5.2. Menu Controller

The Menu Controller contains the **Intro Screen Manager** and **Main Panel Controller scripts**.

Intro Screen Manager - manages the initial screen, which displays registration/login and download. Using checkboxes, you can enable/disable certain windows in the Intro Screen



Manages the screens shown at game launch (splash, login, loading) and the transition to the main menu.

Key Features:

- Configures the screen sequence via Inspector checkboxes.
- Optionally shows the intro only on the first launch (showOnce).

- Uses animations for screen transitions.
- Supports an optional login screen (haveLoginScreen).

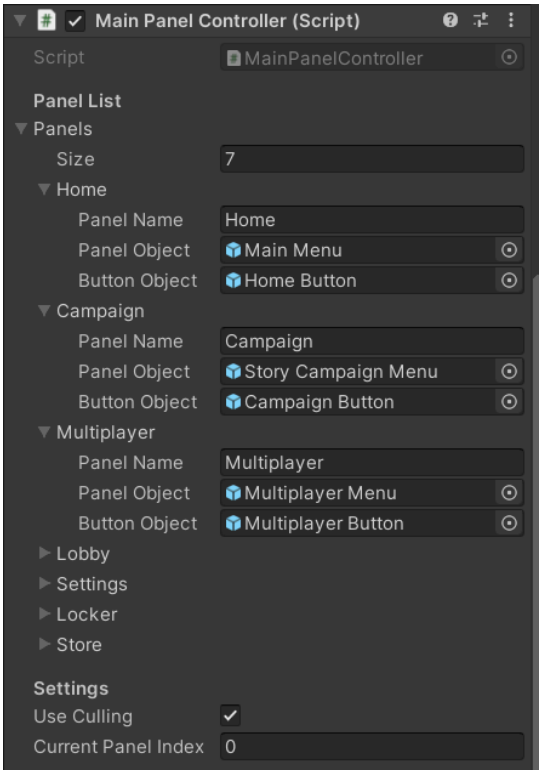
Key Settings:

- splashScreen, mainPanels: References to the intro screens and main menu GameObjects.
- hideSplashScreen, showPressAnyKeyScreen, haveLoginScreen, showOnce: Checkboxes to configure the flow logic.

How to Use:

1. Attach the script to your UI manager.
2. Assign splashScreen, mainPanels, and their Animators.
3. Set the checkboxes in the Inspector.

Main Panel Controller - all the panels and their buttons that we have in the game are declared there. They are also switched through this script. Therefore, if you create a new panel, you should definitely add it here for further interaction



This is the central script for navigating between your main UI panels (menus) using animations.

Key Features:

- Manages a list of panels and their associated buttons.
- Switches panels by name, index, or to next/previous.
- Animates transitions for both panels and buttons.
- Optionally disables inactive panels for optimization (useCulling).

Key Settings:

- panels: A list where each panelName is linked to a panelObject and a buttonObject.
- useCulling: A checkbox to enable/disable optimization.
- currentPanelIndex: The starting panel.

Requirements:

- panelObject and buttonObject must have Animator components.
- Animators need states: Panel In, Panel Out (for panels) and Normal to Pressed, Pressed to Dissolve (for buttons).

How to Use:

1. Attach the script to your UI manager.
2. Fill the panels list, assigning your panels and buttons.
3. Set up button OnClick events to call OpenPanel("PanelName").

Scene Loader - responsible for switching between scenes.

5.3 Intro Screen
Prefab of the initial screen. It has an initial window with an action (click the button), a login, registration and download window, and a Delay Action script

Delay Action is responsible for what will happen after the download and how long it will last.

Intro Screen Background - the BackgroundRandomer script hangs on it, where sprites and a picture are set to be randomly assigned. You can also bind the onClick method from this script to a button and the sprites will change cyclically.

Login Screen - the game logo, a field for entering an email and password, an input button, a “remember me” checkbox, and switching to the registration screen

Registration Screen - the game logo, a field for entering email, nickname and password, an enter button, an “agree to the terms of service” checkbox and switching to the login screen.

5.4 Locker Menu

The locker window, in which we have the available items that are available to us.

Locker Manager - a script for managing the locker. It has root data (a scriptable object in which all our items are stored).

Key Features:

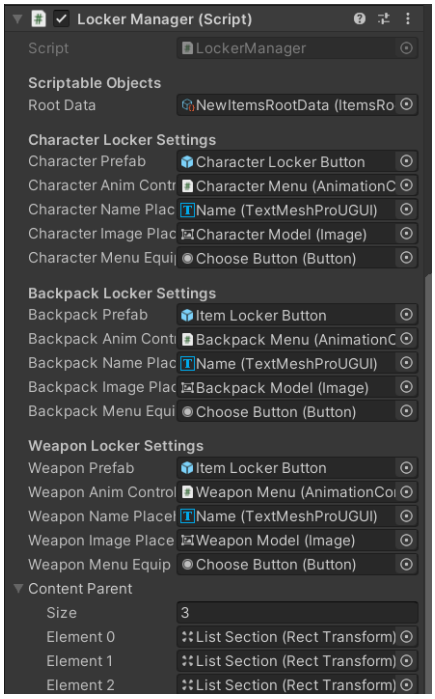
- **Singleton:** Provides a single access point.
- **Display:** Creates UI elements for unlocked items from ItemsRootData.
- **Equipping:** Manages selection and the visual state of equipped items via AnimationController.
- **Updates:** Shows details of the selected item and handles "new" item tags.

Settings for different locker panels:

You need to pass the prefab from which the object will be created, the AnimationController of the panel, a placeholder for the text with the name of the object, a placeholder for the image with the object sprite, and a button that is responsible for selecting the object. Also, a list of transform objects in which the locker objects will be created is passed to the Content Parent (in order according to the order of the panels in the script)

How to Use:

1. Attach the script to your UI manager.
2. Configure all Inspector fields, **especially the contentParent order**.
3. Ensure your prefabs have the LockerElementController.
4. Call **CreateOne...** when the player acquires a new item.



This script creates objects in the locker, changes the status to “active”, checks if the object is new (just purchased), and changes the placeholders to the data of the selected object.

5.5 Store Menu

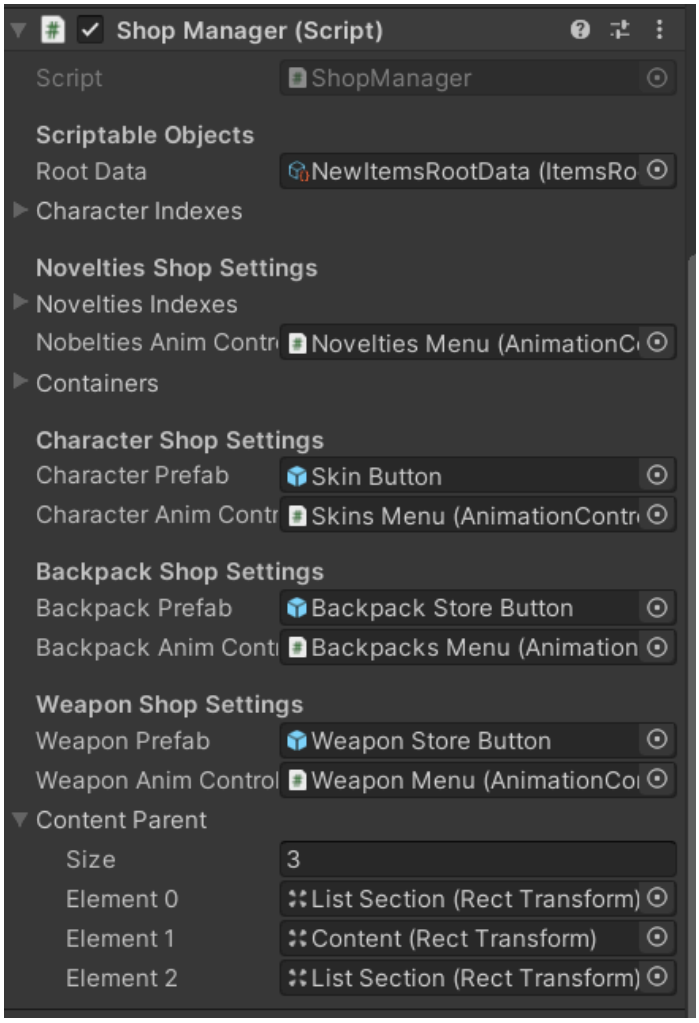
Shop Manager is a script for managing the store. It has root data (a script object where all our stuff is stored).
Settings for different store panels:
Character Indexes - character indexes with root date that will be created in the character panel.

Novelties settings:
Novelties Indexes - items with these indexes will be taken from the root date, from the corresponding list specified in the code. Anim Controller of the Novelties panel and the list of containers in which objects will be created.

The settings for the character, backpack, and weapon panels are similar:
Prefab from which the object will be created and Anim Controller of the panel.
Also, a list of containers in which objects will be created.

How to Use:

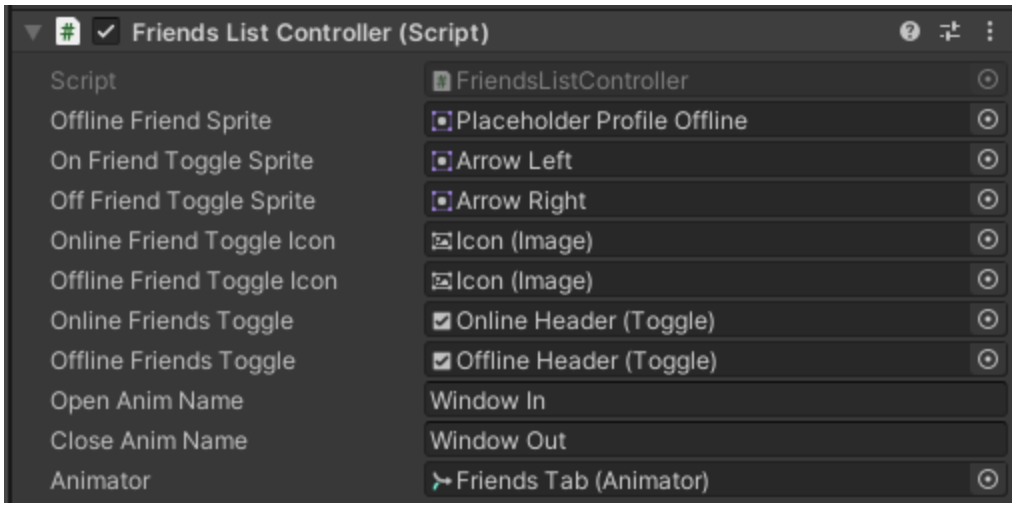
1. Attach the script to your UI manager.
2. Configure all Inspector fields.
3. Ensure your prefabs have a ShopElementController.
4. The shop populates on start; update the item's locked status after purchase.



This script creates items in the store from root data and checks whether they are already purchased or not.

5.6 Friends Tab

FriendsListController is a component designed to manage the display and functionality of the "Friends List" window in your UI. It provides animation, filtering, and basic interaction capabilities.



Key Features:

- **Window Animation:** Handles the smooth opening (WindowIn) and closing (WindowOut) of the window using an Animator. The ToggleWindow method switches between these states.
- **Online/Offline Filtering:** Utilizes two Toggle components to show or hide friends based on their status (online or offline).
- **Dynamic Icons:** Automatically changes the icons of the Toggle components to visually represent their active or inactive states. It can also set a specific icon for offline friends.
- **External Visibility Control:** Provides UnshownFriends and ShowFriends methods, allowing other scripts (like a search system) to temporarily hide or show individual friends, independent of the main filters.

Key Settings (in Unity Inspector):

For the script to work correctly, you need to configure the following fields in the Inspector:

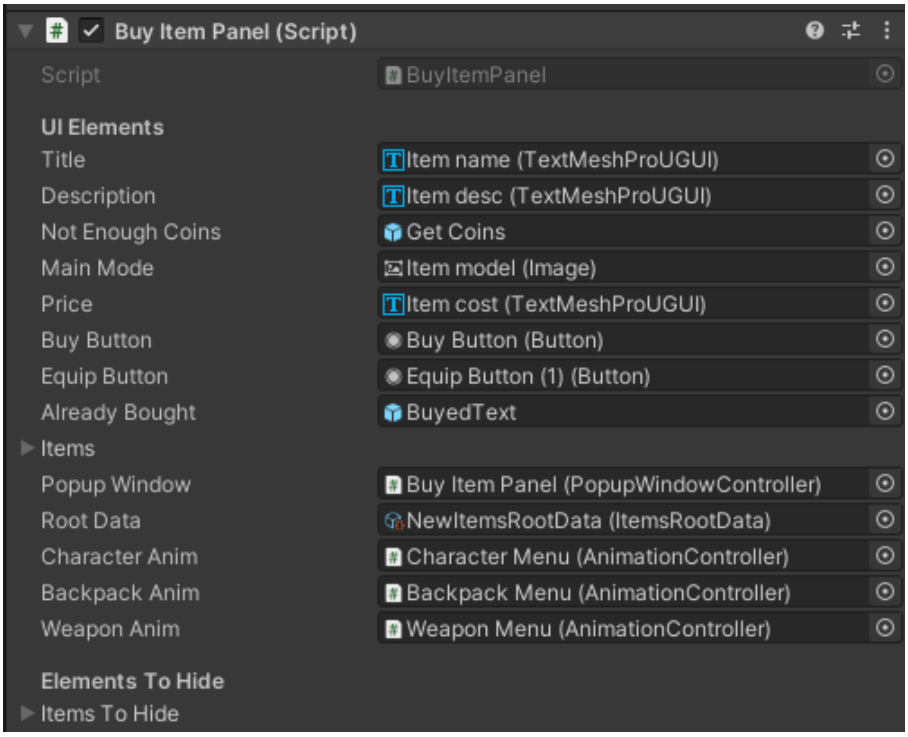
- **Sprites:** Assign icons for the toggle states (on/off) and for offline friends.
- **UI Elements:** Provide references to the Image (icons) and Toggle components for both online and offline filters.
- **Animator:** Assign the window's Animator and specify the names for the open (openAnimName) and close (closeAnimName) animations.
- **Hierarchy: It's crucial** to arrange the child objects so that the Online toggle comes first, followed by the list of online friends, then the Offline toggle, and finally, the list of offline friends.

Usage:

Attach this script to the main object of your friends list window, configure its fields in the Inspector, and call ToggleWindow() to control the window's visibility. Use UnshownFriends/ShowFriends to integrate with additional filtering logic.

5.7 Buy Item Panel

BuyItemPanel is a crucial component responsible for the functionality of the popup panel that appears when a player wants to purchase or equip an item from the shop. It manages the display of item information, the purchase process, and the equipping logic.



Purpose and Key Features:

- **Singleton:** The script is implemented as a Singleton, ensuring that only one instance of this panel exists in the game. This provides easy access to it from other scripts via BuyItemPanel.Instance.
- **Information Display:** Dynamically updates UI elements (title, description, image, price) according to the selected item.
- **Purchase Process:** Handles the "Buy" button click, checks for sufficient currency (via GameEvents), performs the purchase, updates the item's status, and adds it to the LockerManager.
- **Status Check:** Determines if an item is already bought and accordingly shows either the "Buy" or "Equip" button, along with an "Already Bought" message.
- **Purchase Availability:** Checks if the player has enough funds and enables/disables the "Buy" button accordingly.
- **Equipping:** Handles the "Equip" button click, interacting with LockerManager and GameEvents to update the player model and the state of equipped items.

- **Display Management:** Controls the showing and hiding of the panel itself (using PopupWindowController) and can hide other UI elements while it's active.

Settings in Unity Inspector:

To function correctly, the BuyItemPanel script requires the following fields to be set up in the Inspector:

- **Header("UI Elements")**:
  - title, description, price: Text fields (TextMeshPro) for displaying the item's name, description, and price.
  - notEnoughCoins: The GameObject shown when funds are insufficient.
  - mainMode: The main image display for the item.
  - buyButton, equipButton: Buttons for buying and equipping.
  - alreadyBought: The GameObject shown if the item is already owned.
  - items: A list of GameObjects (up to 3) representing the images of the item or set.
  - popupWindow: A reference to the PopupWindowController of this panel.
  - rootData: A reference to the ScriptableObject (ItemsRootData) containing data for all items.
  - characterAnim, backpackAnim, weaponAnim: References to the AnimationControllers for the locker, used to manage equipping animations.
- **Header("Elements To Hide")**:
  - itemsToHide: A list of UI elements to hide when this panel is active.

Core Methods:

- **UpdateItems(...)**: The main method for updating the panel. It takes item data (LockerElement), title, description, images, and a GameObject. It populates all UI elements and checks the item's status (bought/not bought, can afford).

- **ShowPanel()**: Displays the purchase panel and hides elements from the itemsToHide list.
- **BuyItem()**: Called when the "Buy" button is clicked. Executes the purchase logic.
- **CheckBuyOpportunity()**: Checks if there are enough funds and updates the "Buy" button's state.
- **CheckIfBuy()**: Checks if the item is owned and toggles the visibility of the "Buy" / "Equip" buttons.
- **SetEquip(Image image)**: Called when the "Equip" button is clicked. Executes the equipping logic.

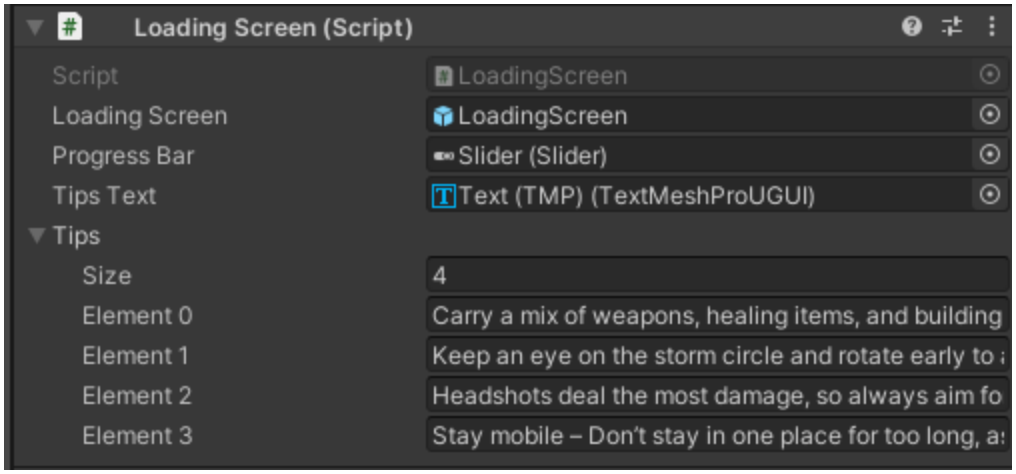
How to Use:

1. Attach the BuyItemPanel script to the main GameObject of your purchase panel.
2. Ensure only one instance of this panel exists in the scene.
3. Carefully configure all fields in the Inspector by dragging the respective UI elements and ScriptableObjects.
4. From other scripts (like ShopManager or LockerManager), call BuyItemPanel.Instance.UpdateItems(...) with the selected item's data, followed by BuyItemPanel.Instance.ShowPanel() to display the panel.
5. The "Buy" and "Equip" buttons are set up via the Start() method, but ensure they have the Button component attached.

5.8 Loading Screen

LoadingScreen is a simple yet effective script for displaying a loading screen during transitions between scenes. It simulates the loading process using a progress bar and shows helpful tips to players while they wait.

Important: This script implements a **simulated** loading process. The progress bar fills up over a fixed time (fakeLoadingTime) and is **not** based on the actual scene loading progress. This is useful for quick transitions or when tracking real progress is complex, but it's something to keep in mind.



Key Features:

- **Screen Display:** Manages the visibility of the loadingScreen object.
- **Progress Simulation:** Smoothly fills a Slider (progress bar) over a predefined time.
- **Tips Display:** Cyclically shows random tips from the tips list.
- **Scene Loading:** Loads the specified scene after the "simulation" is complete.

Settings in Unity Inspector:

- [SerializeField] private GameObject loadingScreen;
  - **What it is:** The main parent object for the entire loading screen. The script will activate/deactivate it.
  - **How to set up:** Drag your loading screen GameObject here.
- [SerializeField] private Slider progressBar;
  - **What it is:** The UI Slider element that will be used as the progress indicator.
  - **How to set up:** Drag your Slider here.
- [SerializeField] private TextMeshProUGUI tipsText;
  - **What it is:** The TextMeshPro text field where tips will be displayed.
  - **How to set up:** Drag your TextMeshProUGUI element here.
- [SerializeField] private List<string> tips;
  - **What it is:** A list of text strings that will be shown as tips.

    - **How to set up:** In the Inspector, set the list size and enter each tip into its respective field.

Internal Parameters:

- private float tipsChangeInterval = 3f;
    - **What it is:** The interval in seconds at which the tip on the screen will change.
- private float fakeLoadingTime = 10f;
    - **What it is:** The total time in seconds that the "loading" will last and the progress bar will fill.

Core Methods:

- **StartLoad(int sceneIndex)**: This is the **public** method you should call from elsewhere (e.g., a "Start Game" button or a level manager) to begin the loading process. It takes the index of the scene to load.
- **FillProgressBar(int sceneIndex)**: This is a **Coroutine** that performs the main logic:
    1. Starts another coroutine, ChangeTipRandomly, to display tips.
    2. Gradually increases progressBar.value from 0 to 1 over fakeLoadingTime.
    3. Calls LoadScene upon completion.
- **LoadScene(int sceneIndex)**: Simply loads the scene by its index using SceneManager.LoadScene.
- **ChangeTipRandomly()**: Another Coroutine that runs while the loading screen is active. Every tipsChangeInterval seconds, it selects a random tip from the tips list and displays it in tipsText.

How to Use:

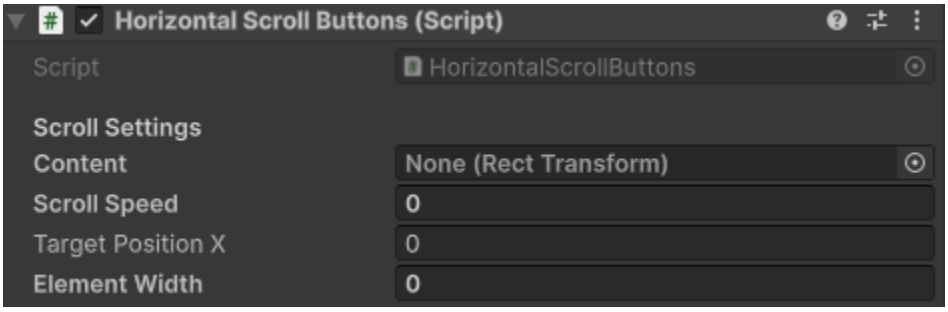
1. Create the UI for your loading screen, including a Slider and a TextMeshProUGUI.
2. Create an empty GameObject (or use the main screen object) and attach the LoadingScreen script to it.

3. Configure all fields in the Inspector: drag the UI elements and fill in the tips list.
4. Ensure your loadingScreen becomes active (SetActive(true)) before calling StartLoad.
5. From where you want to initiate the scene load, get a reference to this script and call StartLoad(your_scene_index).

6. Custom Scripts and Functionality

6.1 Horizontal Scroll Buttons

This is a script for horizontal Scroll View. You need to connect the content of your Scroll View, set the speed at which the animation will move, set the initial position and width of the element that will be switched to when you click on the

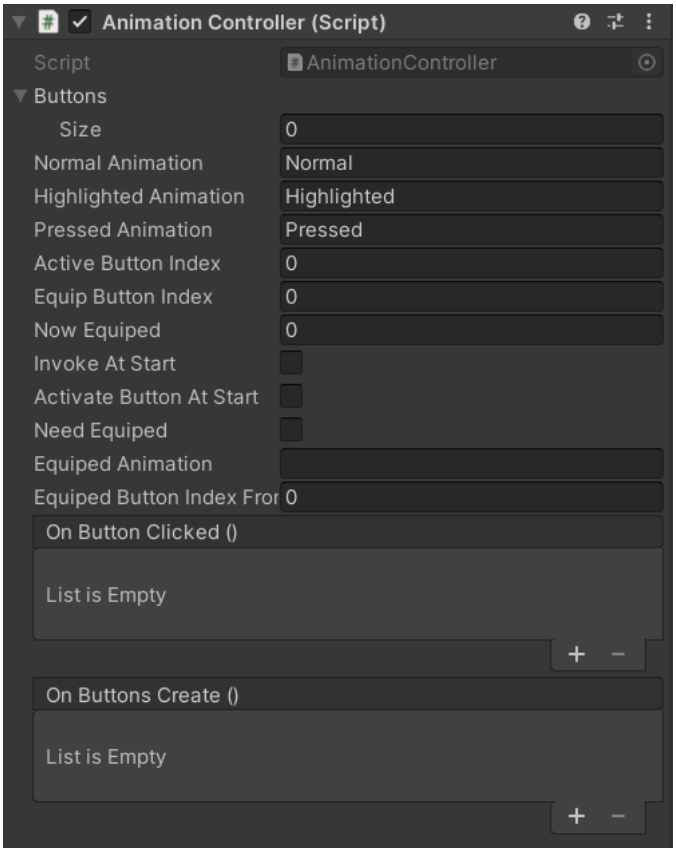


To set up left/right switching, you can call it either through the OnClick event or from your script by calling one of these methods:
- HorizontalScrollButtons.ScrollLeft(); - Scroll elements to left.
- HorizontalScrollButtons.ScrollRight(); - Scroll elements to right.

6.2 Animation Controller

This script controls the animation of the specified buttons. The selected button will remain with its animation until another button is clicked or disabled.

In the variable string, you must specify the names of the animations from the animator. If there is no animation, it will simply not be played. Indexes are shown for better interaction with the script.



If **Invoke At Start** is enabled, the buttons will be created immediately at startup. I recommend using it if you set them manually.
Active Button At Start - allows you to have a button immediately selected when displaying
Need Equiped - implements the mechanics of equipping an item if necessary. You can start the fun of work in the Locker Menu. Also, you need to specify what kind of animation will be played if the item is equipped.
Equiped Button Index From List - the index of the object in the list of objects (see section) - allows you to set which item will be equipped immediately.
OnButtonClicked() & **OnButtonsCreate()** - here you can call certain methods after pressing the buttons of a certain Animation Controller and/or after creating these buttons.

6.3 Game Events

The main script with events used in the project. Since this is not MonoBehavior, it is not initialized anywhere. You can create events and methods there, and then set the buttons that will call them.

6.4 Element Scripts & Element Controllers & Element Lists & Managers

The asset has an implementation of creating objects through scriptable objects (chapter 6)

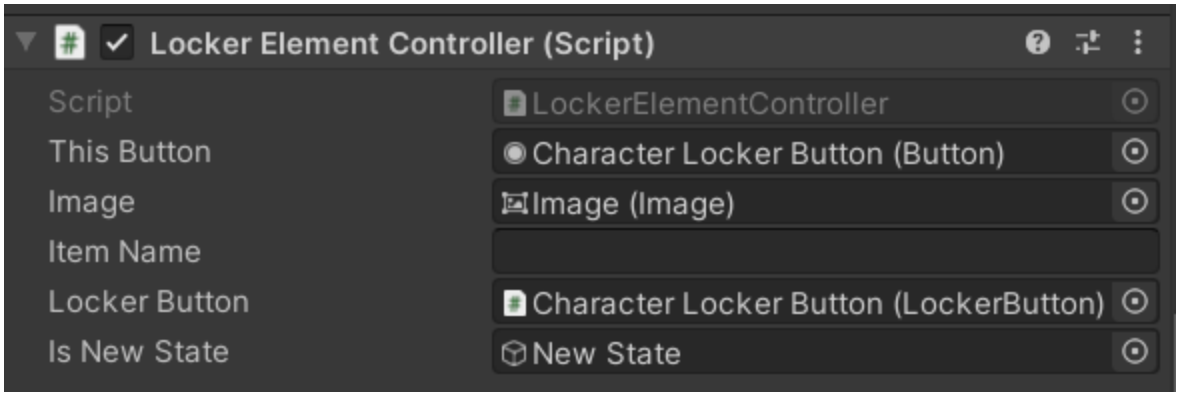
For this implementation, the following are created:

Element Script:

A list of parameters that will be set and configured in the list in the scriptable object (chapter 6)

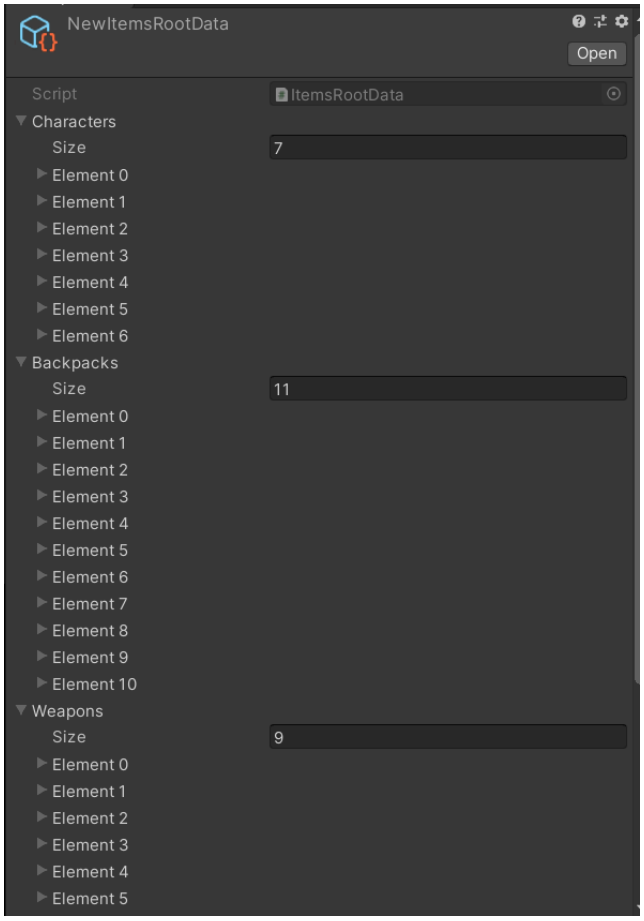
Element Controller:

The script that should be on the element prefab. Objects are passed to it to which values from the object in the list will be assigned to the scriptable object.



Element List:

A script that can be used to create a scriptable object list with elements and their parameters.



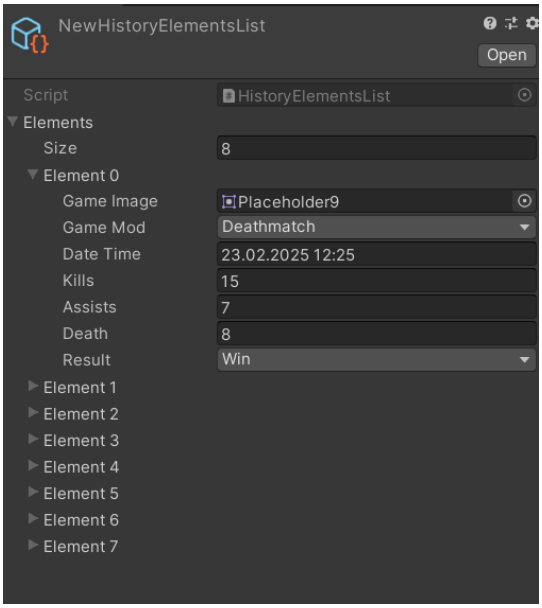
Manager:
Responsible for creating and working with objects from the list in the scriptable object.

In general, there are many other scripts used in the assortment. But in this section, we've described the ones you'll need to work with if you want to add to them or create something new.

7. Scriptable Objects

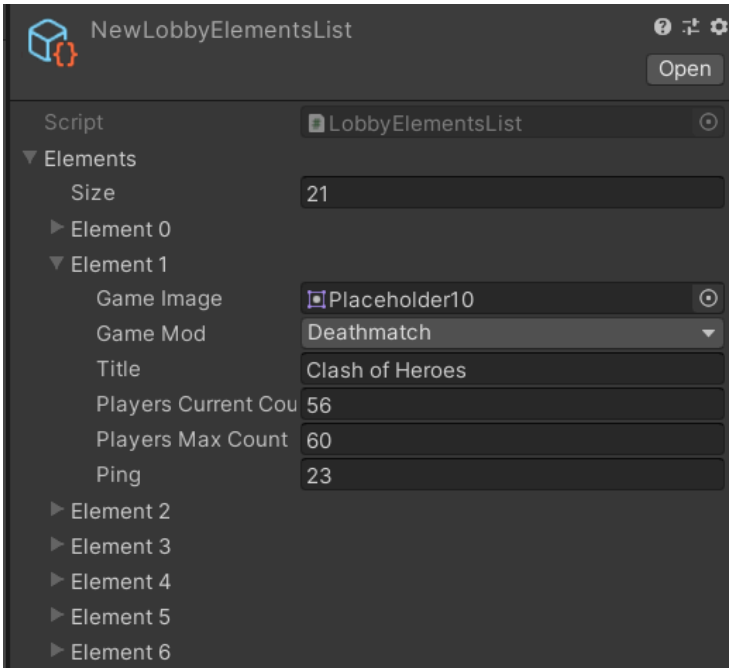
7.1 NewHistoryElementsList

This is a scriptable object list where game history elements are created and customized. This is where you set the picture, mod, date and time, your game statistics (kills, assists, and deaths), and the outcome of the game (win/lose).



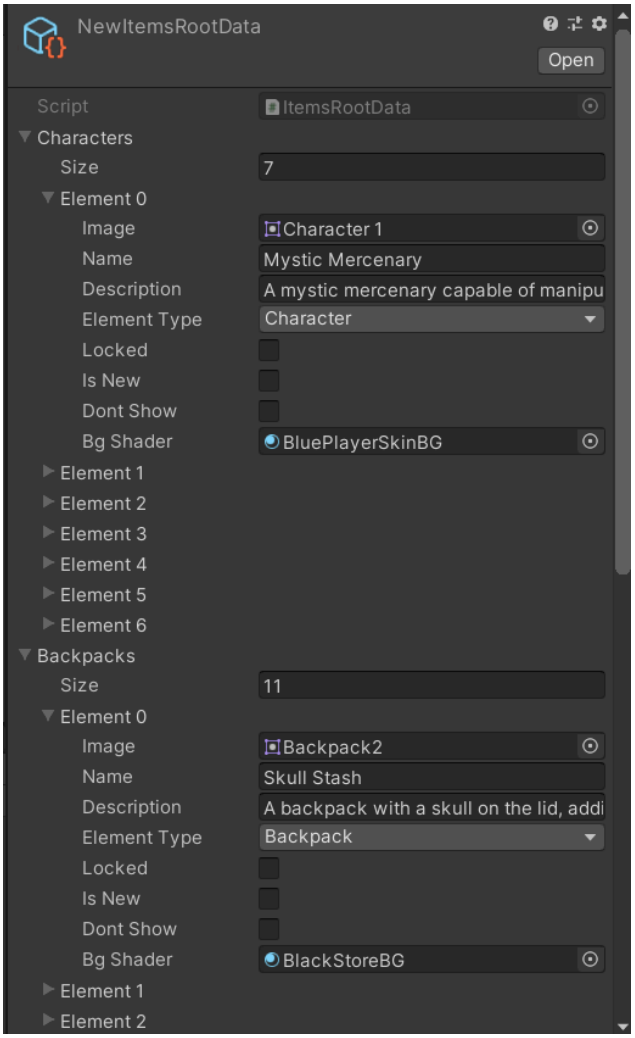
7.2 NewLobbyElementsList

This is a scriptable object list where game lobby elements are created and customized. Here you can set the picture, mod, name, number of players (current and maximum) and ping.



7.3 NewItemsRootData

This is a scriptable object list where game items (characters, backpacks, and weapons) are created and customized. Here you can set the picture, name, description, type of item, locker (available/unavailable), new (just bought), don't show (won't be displayed in the store) and shader for the background of the button in the store.



8. Animations & Effects

Frontier UI leverages Unity's Mecanim system, which offers extensive customization options. This makes it simple to adjust and fine-tune animations to suit your needs.

8.1. Customizing Animation Effects

To modify animations, open the **Animator** tab via **Window > Animation > Animator**. When you select an object that includes animations, you'll see its animation states displayed. From there, you can choose any animation and edit it as needed. For more detailed guidance, please refer to the official Unity documentation on the Animator system.

9. Input System

This package accommodates input events from touchscreens, keyboard/mouse setups, and gamepads.

A component called Gamepad Input Handler is attached to the Menu Manager object in the Cross Platform scene. Depending on the connected controller, the corresponding input objects in the list (Keyboard & Gamepad) will be enabled or disabled. For instance, if a gamepad is connected, the keyboard-related objects will automatically be turned off.

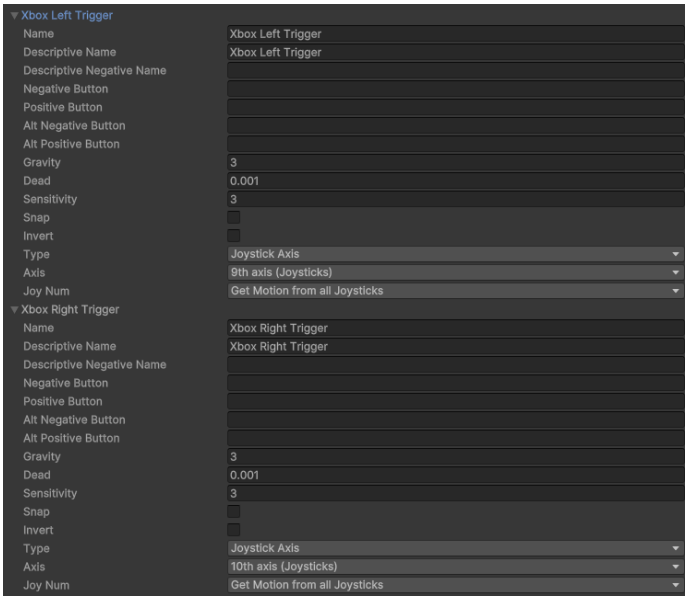
9.1. Shortcut Configuration

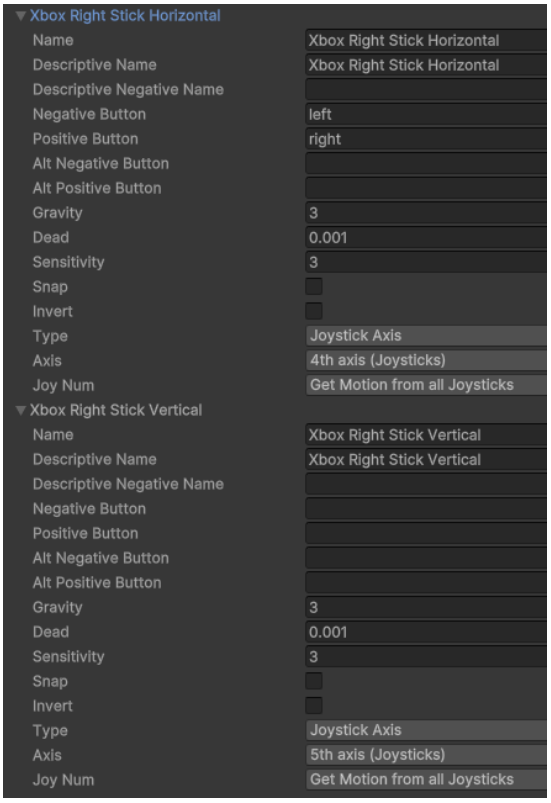
You can trigger specific events when a particular key is pressed. These events are organized under the **[Modal Name]** > Hotkeys.

9.2. Gamepad Integration

Gamepad functionality is integrated through the key shortcut system along with a virtual cursor. If you prefer traditional navigation, you can switch the **Navigation** setting on UI objects to **Automatic**.

To fully utilize all gamepad features, you need to incorporate four new inputs. You also have the option to download the **InputManager.asset** file and replace your existing one with it.



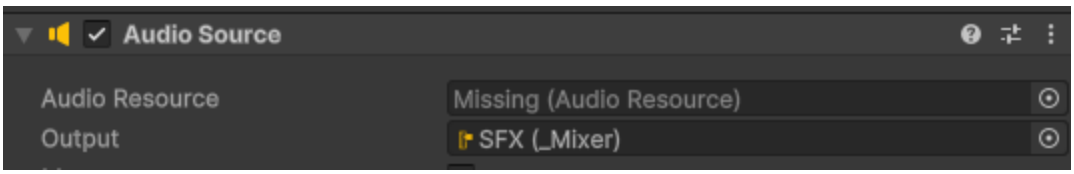


10. Quality Settings

Frontier UI comes with a built-in graphics, quality, and audio management system, saving you the hassle of building these features from scratch.

10.1. Audio Controls

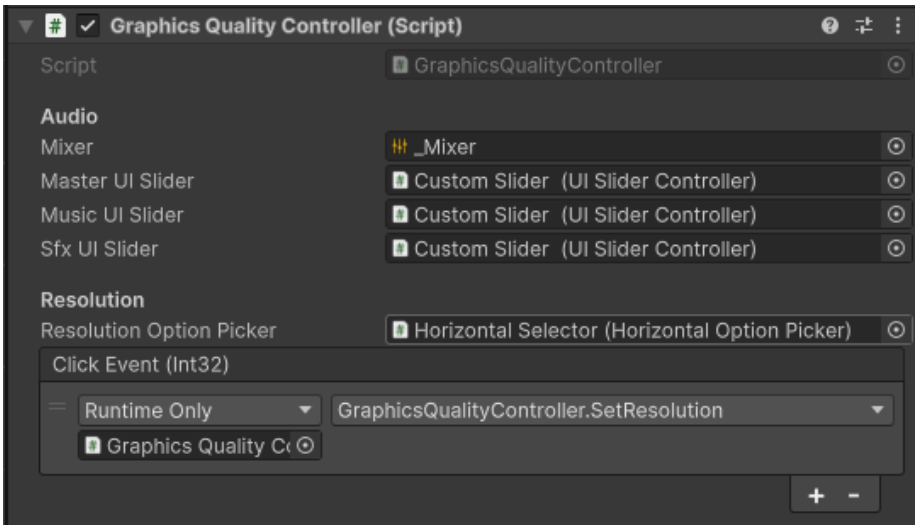
The audio settings are controlled via an Audio Mixer, which includes three distinct channels: Master, SFX, and Music. To integrate an audio source with the mixer, simply assign one of these states to the source's **Output** field. The audio source will then adopt the corresponding mixer settings.



10.2. Graphics Optimization

Graphics settings are primarily adjusted through events. For example, calling **GraphicsQualityController.VsyncSet(0);** will disable vertical synchronization (Vsync). If you're working with one of the demo scenes, there's no need for additional configuration—the settings are already optimized and ready to use.

Resolution options are generated dynamically by the Graphics Quality Controller. To set this up, assign your resolution selector to the **Resolution Option Picker** field within the Graphics Quality Controller. Then, add a new **Click Event** with the function **SetResolution**, and the system will handle the rest.



11. Contact & License Info

You can contact me or get the latest updates via:

Website E-mail Discord

Also, if you are interested in us, you can learn more about us through our social networks

Instagram ArtStation Facebook

If you encounter any issues, have questions, or simply want to share your suggestions or feedback, please don’t hesitate to get in touch.

Licence

This package is governed by the default Asset Store license and terms of use. For more details, please visit: https://unity3d.com/legal/as_terms