VR UI Package

by Realary

List of content:

Introduction:

The VR UI Package is a Unity package designed to accelerate the development of VR applications and games. It offers a collection of prebuilt UI tools that can be easily integrated into your projects. By leveraging these tools, you can eliminate the need to build basic UI functionalities from scratch, allowing you to focus on more creative and specialized aspects specific to your project. We believe that this package will enhance your development experience and empower you to create compelling VR experiences. Enjoy developing with the VR UI Package!

Key Features:

- Prebuilt UI tools for VR applications and games.
- Accelerates development by providing essential UI functionalities.
- Reduces the time and effort required to implement common UI components.
- Enables developers to concentrate on more creative and specialized aspects of their projects.

Please refer to the package documentation and included examples for detailed instructions on how to integrate and utilize the VR UI Package in your Unity projects. Should you encounter any issues or have any questions, please consult the provided support resources. Happy development!

Additional Information

- Feedback or Feature Requests: We value your feedback and welcome any suggestions or feature requests you may have. Please feel free to reach out to us via email or leave your comments, and we will promptly address them. Your input helps us improve VRUIP and deliver a better experience to our users. Thank you for choosing VRUIP!
- Free VR Development Tutorials: Enhance your VR development skills by exploring our YouTube channel, RealaryVR. We offer a variety of free tutorials that cover various aspects of VR development, providing you with valuable insights and guidance. Visit our channel to access a wealth of resources that can help you make the most out of VRUIP and create immersive VR experiences.

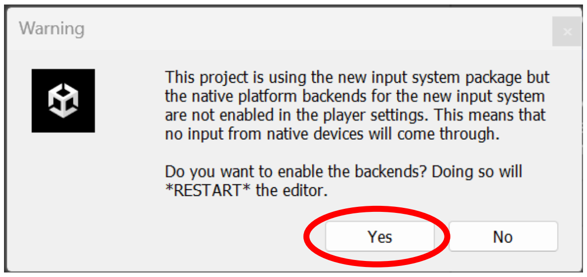
We appreciate your support and look forward to assisting you in your VR development journey. If you have any further questions or require assistance, please don't hesitate to reach out to us.

Setup & Implementation:

Setup for Editor:

1. Import package.

2. If you don't have the new input system in your project, go to the package manager. Select Unity Registry, and then search for Input System. Install the "Input System" package. If it asks you to "enable the backend" accept (Press **Yes**). That will restart the project.



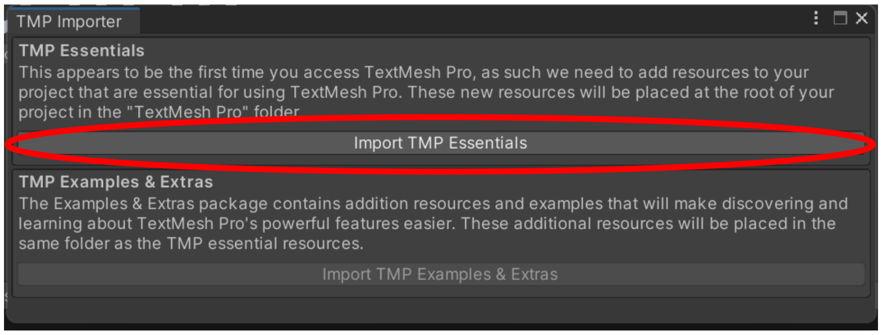
Warning

This project is using the new input system package but the native platform backends for the new input system are not enabled in the player settings. This means that no input from native devices will come through.

Do you want to enable the backends? Doing so will *RESTART* the editor.

3. Go to VRUIP/Demo Scenes/ and open the DemoSceneEditor scene.

4. It will prompt you to import TMP Essentials (TextMeshPro) if you don't already have them. Import them to your project (No need for the TMP examples and extras).



TMP Importer

TMP Essentials

This appears to be the first time you access TextMesh Pro, as such we need to add resources to your project that are essential for using TextMesh Pro. These new resources will be placed at the root of your project in the "TextMesh Pro" folder.

TMP Examples & Extras

The Examples & Extras package contains addition resources and examples that will make discovering and learning about TextMesh Pro's powerful features easier. These additional resources will be placed in the same folder as the TMP essential resources.

5. You are now ready to play and test the demo scene in Editor ✅

Setup for Oculus Integration:

1. Same as editor.
2. Same as editor.
3. Go to package manager and import Oculus Integration package from Unity Asset Store.
4. Once you import the Oculus Integration, go to the top bar in Unity and you should see VRUIP. Choose Oculus Integration for Target Framework and wait for scripts to reload.

5. Open **OVRInputModule** script by searching in Unity project search bar and add this function to it (check setup video on package page if not sure how to do it):

```
protected override void ProcessMove(PointerEventData pointerEvent)
{
    var targetGO = pointerEvent.pointerCurrentRaycast.gameObject;
    HandlePointerExitAndEnter(pointerEvent, targetGO);
}
```

6. Go to VRUIP/Demo Scenes/ and open the DemoSceneOculus scene.

7. You are now ready to play and test the demo scene in your headset (If your project is setup for VR correctly) ✅

Setup for XR Interaction Toolkit:

1. Same as editor.
2. Same as editor.
3. Go to package manager, Unity Registry, and import the "XR Interaction Toolkit" (Version 3.0.1)

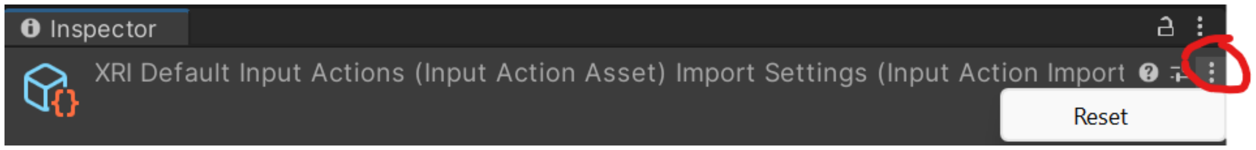
4. Once that is done importing, you will see a "Samples" option in the XR Interaction Toolkit window in the package manager. Open it and import "Starter Assets".

5. Go to the top bar in Unity and you should see VRUIP. Choose XR Interaction Toolkit for Target Framework and wait for scripts to reload.

6. Go to VRUIP/Demo Scenes/ and open the DemoSceneXRITK scene.

7. You are now ready to play and test the demo scene in your headset (If your project is setup for VR correctly) ✅

8. Bonus: if you find that your hands are not tracking, this is due to a Unity bug. To fix go to **"Assets/Samples/XR Interaction Toolkit/3.0.1/Starter Assets"** and select the file with the name **"XRI Default Input Actions".** In the inspector tab, click on the three dots and click **Reset**.


Inspector
XRI Default Input Actions (Input Action Asset) Import Settings (Input Action Import
Reset

Setup for Meta SDK (All-in-One):

1. Same as editor.
2. Same as editor.
3. Install “Meta XR All-in-One SDK” package from Unity Asset Store (Free).
4. Go to VRUIP/Demo Scenes/ and open the DemoSceneMetaSDK scene.
5. You are now ready to play and test the demo scene in your headset (If your project is setup for VR correctly) ✅

Note: Please watch the setup video on the package page for additional help. These steps will **NOT** teach you how to setup your Unity project for VR. If you don’t know how to do that there are plenty of videos on YouTube and plenty of documentation online. Checkout our YouTube channel RealaryVR for video tutorials on how to setup your Unity project for VR.

FAQ:

- ## There is a jitter/lag in my VR application while moving things around. How can I fix that?

  You can fix that by going to the ***Player Settings*** in Unity, going to the ***Time*** tab, and setting the ***Fixed Timestep*** setting to **0.0138** (Usually it's 0.02).

- ## The Camera tool pictures are not complete/look like they were cropped/have black portions. How can I fix that?

  This is a known issue in Vulkan (Graphics API) in some versions of Unity. If this issue is happening for you, go to the ***Player Settings***, then ***Other Settings***, Uncheck ***Auto Graphics API***, and remove Vulkan from the list (Keep OpenGLES3).

- ## Does VR UI Package support VRIF or HurricaneVR packages?

  This question gets asked a lot. While it is possible to make VRUIP work with these packages, we do not provide out of the box support. This means that some work will need to be done to get it to work. We don't have access to these packages and hence can't guarantee support as their UI systems are not public.

- ## What frameworks does VRUIP support?

  VRUIP supports the following:

  1. Editor / Mouse.
  2. XR Interaction Toolkit (3.0.1)
  3. Meta XR SDK (All-in-One SDK).
  4. Oculus Integration (Meta's deprecated/old XR package).

Components in Package:

There are 3 sections of prefabs in the package: UI, 3D UI, and Tools. More details in Documentation.

UI:

This section contains some basic and intermediate UI components that can be used in many cases. All these elements will follow the theme specified in the VRUIPManager, unless specifically told not to.

1) Button
2) Dropdown
3) Gallery
4) Icon
5) Image
6) Input Field
7) Loading Circle
8) Option
9) Slider
10) Switch
11) Text
12) Toggle
13) VR Canvas
14) Keyboard
15) Mini Keyboard
16) Color Picker
17) Tab Elements
18) Login Template
19) Main Menu Template
20) Universal Menu Template
21) Video Player
22) UI Handle for moving things.
23) Hand HUD

Tools:

This section provides a couple of fun tools that can be used in your VR application.

1) Drawing with Pen, Eraser, and Color Changer.
2) VR Camera

3D UI:

This section provides a 3D UI interface. While it’s true that the collection and elements are still 2D elements, they are in the 3D section because each one of them is its own canvas and moves in 3D dimensions for the effects, giving them a more 3D UI feel.

1) 3D Physical Button
2) Collection of Elements (For a selection display).
3) Highlight Effect Element
4) Rotating Highlight Effect Element
5) Pop Effect Element
6) Rotating Effect Element

Script Documentation:

List of scripts (Click to navigate):

VRUIPManager:

The **VRUIPManager** class is responsible for managing the package elements and is considered the main controller. It provides functionality for setting color themes, managing VR-specific components, handling input, and controlling the keyboard.

IMPORTANT: This component is a must in any scene that implements this packages' elements. There is a prefab provided that has the VRUIPManager script attached to it. Import this prefab to your scene and assign the correct variables to make sure your scene works with VRUIP components!

Prefab path: VRUIP/Prefabs/UI/Other/VRUIP_Manager.prefab

Properties

- **colorMode**: The current color theme mode.
- **customColorTheme**: The custom color theme.
- **mainCamera**: The main camera used in the VR scene.
- **lineRenderer**: The line renderer component.
- **scaleButton**: The scale button component.
- **keyboardPrefab**: The prefab for the keyboard UI.

Methods

- **GoToScene(string scene)**: Loads the specified scene.
- **ShowKeyboard()**: Displays the keyboard UI.
- **HideKeyboard()**: Hides the keyboard UI.
- **SetKeyboardInput(TMP_InputField inputField)**: Sets the input field for the keyboard.
- **SetTheme()**: Sets the UI theme based on the current color mode.
- **RegisterElement(A_ColorController element)**: Registers a color controller element.

Public Variables

- **colorMode**: The current color theme mode.
- **customColorTheme**: The custom color theme.
- **mainCamera**: The main camera used in the VR scene.
- **lineRenderer**: The line renderer component.

- **scaleButton**: The scale button component.
- **keyboardPrefab**: The prefab for the keyboard UI.

Public Methods

- **GoToScene(string scene)**: Loads the specified scene.
- **ShowKeyboard()**: Displays the keyboard UI.
- **HideKeyboard()**: Hides the keyboard UI.
- **SetKeyboardInput(TMP_InputField inputField)**: Sets the input field for the keyboard.

Private Variables

- **_colorControllers**: A list of color controller elements.
- **_colorThemes**: A dictionary mapping color theme modes to color themes.
- **_audioSource**: The audio source component.
- **_keyboard**: The instance of the keyboard UI.

Private Methods

- **Awake()**: Initializes the instance of **VRUIPManager**.
- **Start()**: Runs at the start of the game and sets up additional components.
- **SetupInstance()**: Sets up the singleton instance of **VRUIPManager**.
- **CalculateSpawnPoint()**: Calculates the spawn position, rotation, and content rotation for the keyboard UI.
- **SetupThemes()**: Sets up the color themes for the UI elements.

Enums

- **TargetFramework**: Represents the target framework of the Unity project.
- **ColorThemeMode**: Represents the color theme mode for the UI.

ButtonController:

Summary

The **ButtonController** class is responsible for managing the behavior and appearance of a button in a VR user interface. It provides various customization options for colors, animations, sounds, and properties of the button.

Usage

1. Attach the **ButtonController** script to a UI button object in your VR scene.
2. Customize the properties of the **ButtonController** to modify the button's appearance and behavior.
3. Optionally, register event handlers or functions to be called when the button is clicked or interacted with.

Properties

- **buttonNormalColor**: The normal color of the button.
- **borderNormalColor**: The normal color of the button's border.
- **textNormalColor**: The normal color of the button's text.
- **buttonHoverColor**: The color of the button when hovered over.
- **borderHoverColor**: The color of the button's border when hovered over.
- **textHoverColor**: The color of the button's text when hovered over.
- **buttonClickColor**: The color of the button when clicked.
- **borderClickColor**: The color of the button's border when clicked.
- **textClickColor**: The color of the button's text when clicked.
- **text**: The text displayed on the button.
- **borderWidth**: The width of the button's border.
- **roundness**: The roundness of the button's corners.
- **hasBorder**: Determines if the button has a border.
- **clickSoundEnabled**: Determines if a sound is played when the button is clicked.
- **clickSound**: The sound played when the button is clicked.
- **hoverSound**: The sound played when the button is hovered over.
- **animationType**: The type of animation applied to the button.

Methods

- **SetupButton()**: Sets up the button's properties and functionalities.
- **RegisterOnClick(action)**: Registers a function to be called when the button is clicked.

Events

- **OnEnter()**: Event triggered when the pointer enters the button.

- **OnExit()**: Event triggered when the pointer exits the button.
- **OnDown()**: Event triggered when the button is pressed.
- **OnUp()**: Event triggered when the button is released.

ColorPicker:

ColorPickerController Documentation

The **ColorPickerController** is a script that allows users to pick colors using a color picker interface. It provides functionality to change the color using a gradient picker and an input field for entering a color value in hexadecimal format.

Features

- Gradient Picker: Users can select a color by interacting with a gradient picker, which allows them to choose the hue, saturation, and value of the color.
- Hexadecimal Input: Users can also enter a color value directly using the provided input field, allowing them to input colors in hexadecimal format.
- Color Events: The color picker raises an event whenever the selected color is changed, providing the updated color to any subscribed listeners.

Usage

To use the **ColorPickerController** in your project, follow these steps:

1. Attach the **ColorPickerController** script to a canvas object in your scene.
2. Create a UI that includes the following components:
    - An **Image** component for displaying the background of the color picker.
    - An **Image** component for displaying the gradient picker.
    - An **Image** component for displaying the currently selected color.
    - An **Image** component for displaying an outline around the currently selected color.
    - A **Slider** component for selecting the hue of the color.
    - An **Image** component for displaying the handle of the hue slider.
    - An **Image** component for displaying an outline around the handle.
    - An **Image** component for displaying the background of the hue slider.
    - An **Image** component for displaying a circular indicator in the gradient picker.

    - A **TMP_InputField** component for entering color values in hexadecimal format.

3. Assign the necessary components to the corresponding fields in the **ColorPickerController** script:
    - Assign the background **Image** component to the **background** field.
    - Assign the gradient picker **Image** component to the **gradientImage** field.
    - Assign the currently selected color **Image** component to the **currentColorImage** field.
    - Assign the outline **Image** component for the currently selected color to the **currentColorOutline** field.
    - Assign the hue slider **Slider** component to the **hueSlider** field.
    - Assign the handle **Image** component for the hue slider to the **handleImage** field.
    - Assign the outline **Image** component for the handle to the **handleOutline** field.
    - Assign the background **Image** component for the hue slider to the **sliderBackground** field.
    - Assign the circular indicator **Image** component to the **colorPickerCircle** field.
    - Assign the **TMP_InputField** component for entering color values to the **currentColorText** field.
4. (Optional) Add an event listener to the **onColorChanged** event if you need to be notified when the selected color changes.
5. (Optional) Customize the appearance of the color picker by modifying the assigned UI components or updating the **SetColors** method to change the colors dynamically based on your theme.

Public Properties

- **CurrentColor**: Gets the currently selected color.

Public Methods

- **OnPointerDown(PointerEventData eventData)**: Handles the pointer down event on the gradient picker, allowing users to select a color by clicking on the gradient.
- **OnDrag(PointerEventData eventData)**: Handles the drag event on the gradient picker, enabling users to interactively select a color by dragging the circular indicator.
- **OnEndDrag(PointerEventData eventData)**: Handles the end of the drag event on the gradient picker.

Events

- **onColorChanged**: An event that is raised whenever the selected color changes. Subscribers can listen to this event to receive updates about the new color. The updated color is passed as a parameter

DropdownController:

DropdownController Documentation

The **DropdownController** is a script that customizes the appearance of a dropdown menu UI component. It allows you to define colors for different states of the dropdown and customize the text color.

Features

- Color Customization: You can specify different colors for the normal, hover, and pressed states of the dropdown, as well as the text color.
- Compatibility with VRUIP: The script includes a context menu option to set up the dropdown with colors from the VRUIP color theme.

Usage

To use the **DropdownController** in your project, follow these steps:

1. Attach the **DropdownController** script to a GameObject that contains a dropdown menu UI component.
2. Assign the necessary components to the corresponding fields in the **DropdownController** script:
    - Assign the **TMP_Dropdown** component to the **dropdown** field.
    - Assign the **Toggle** component to the **option** field.
    - Assign the **TextMeshProUGUI** component for displaying the selected option text to the **selectedText** field.
    - Assign the **TextMeshProUGUI** component for displaying the option text to the **optionText** field.
3. Customize the colors:
    - Set the **normalColor** field to the desired color for the normal state of the dropdown.
    - Set the **hoverColor** field to the desired color for the hover state of the dropdown.
    - Set the **pressedColor** field to the desired color for the pressed state of the dropdown.
    - Set the **textColor** field to the desired color for the text in the dropdown.
4. (Optional) Customize the appearance of the dropdown further by modifying the assigned UI components.
5. (Optional) Use the **SetColors** method to dynamically update the colors based on your theme or requirements. This method is called automatically when the color theme is changed.

6. (Optional) Use the **SetupDropdown** method to manually set up the dropdown with colors from the VRUIP color theme. This can be done through the context menu option "Setup Dropdown (VRUIP)" available in the Unity Editor.

Public Fields

- **normalColor**: The color used for the normal state of the dropdown.
- **hoverColor**: The color used for the hover state of the dropdown.
- **pressedColor**: The color used for the pressed state of the dropdown.
- **textColor**: The color used for the text in the dropdown.

Public Methods

- **SetColors(ColorTheme theme)**: Sets the colors of the dropdown based on the provided **ColorTheme**. This method is called automatically when the color theme is changed.
- **SetupDropdown()**: Sets up the dropdown with colors from the VRUIP color theme. This method can be used manually through the context menu option "Setup Dropdown (VRUIP)" available in the Unity Editor.

Context Menu Option

- **Setup Dropdown (VRUIP)**: A context menu option available in the Unity Editor to automatically set up the dropdown with colors from the VRUIP color theme. This option calls the **SetupDropdown** method.

GalleryController:

GalleryController Documentation

The **GalleryController** is a script that manages a gallery of images and provides navigation functionality. It allows you to scroll through the images in the gallery with optional animation and customize the colors of the gallery buttons and title text.

Features

- Gallery Navigation: You can navigate through the images in the gallery by clicking the left and right buttons.
- Scroll Animation: You can enable a smooth scrolling animation when transitioning between images.
- Color Customization: You can specify different colors for the gallery buttons and the title text.
- VR Support: The script adjusts the scroll speed based on whether the application is running in VR or not.

Usage

To use the **GalleryController** in your project, follow these steps:

1. Attach the **GalleryController** script to a GameObject that contains the gallery UI components.
2. Assign the necessary components to the corresponding fields in the **GalleryController** script:
    - Assign the current image **Image** component to the **currentImage** field.
    - Assign the next image **Image** component to the **nextImage** field.
    - Assign the current title **TextMeshProUGUI** component to the **currentTitle** field.
    - Assign the next title **TextMeshProUGUI** component to the **nextTitle** field.
    - Assign the left button **IconController** component to the **leftButton** field.
    - Assign the right button **IconController** component to the **rightButton** field.
3. Create an array of **GalleryItem** instances in the **images** field. Each **GalleryItem** should contain a title (**string**) and an image (**Sprite**).
4. Customize the properties:
    - Set the **scrollAnimation** field to **true** if you want to enable the scrolling animation when transitioning between images. Set it to **false** to disable the animation.
    - Set the **buttonNormalColor** field to the desired color for the normal state of the gallery buttons.
    - Set the **buttonHoverColor** field to the desired color for the hover state of the gallery buttons.
    - Set the **buttonClickColor** field to the desired color for the click state of the gallery buttons.
    - Set the **titleTextColor** field to the desired color for the gallery title text.
5. (Optional) Customize the appearance of the gallery further by modifying the assigned UI components.
6. (Optional) Use the **SetColors** method to dynamically update the colors based on your theme or requirements. This method is called automatically when the color theme is changed.
7. (Optional) Use the **SetupGallery** method to manually set up the gallery. This can be done through the context menu option "Setup Gallery (VRUIP)" available in the Unity Editor.

Public Fields

- **images**: An array of **GalleryItem** instances representing the images and titles in the gallery.
- **scrollAnimation**: A boolean indicating whether the scrolling animation is enabled or disabled.
- **buttonNormalColor**: The color used for the normal state of the gallery buttons.

- **buttonHoverColor**: The color used for the hover state of the gallery buttons.
- **buttonClickColor**: The color used for the click state of the gallery buttons.
- **titleTextColor**: The color used for the gallery title text.

Public Methods

- **SetColors(ColorTheme theme)**: Sets the colors of the gallery buttons and title text based on the provided **ColorTheme**. This method is called automatically when the color theme is changed.
- **SetupGallery()**: Sets up the gallery by populating the components and setting the initial image and title. This method can be used manually through the context menu option "Setup Gallery (VRUIP)" available in the Unity Editor

HUD:

This class represents a HUD (Heads-Up Display) class in Unity, used for displaying various visual elements in a virtual reality user interface. Here's a concise explanation of each part of the code:

- The class contains several serialized fields representing different UI elements, such as circles, bars, images, and text.
- Private integer variables **_health**, **_maxHealth**, **_stamina**, **_maxStamina**, **_xp**, **_maxXp**, and **_level** store the current and maximum values for health, stamina, experience points, and level.
- Private variables **_healthVisual**, **_staminaVisual**, **_xpVisual**, **_healthText**, **_staminaText**, and **_xpText** store references to the appropriate visual and text UI elements based on the selected HUD type.
- The **Start** method initializes the HUD by setting the appropriate UI section, visuals, and text based on the selected HUD type. It also sets the parent and rotation if the VRUIPManager indicates that the environment is in VR mode. Additionally, it updates the UI elements with example values for health, stamina, and experience points.
- The **SetupHUD** method allows for dynamically changing the HUD type based on the given **hudType** parameter. It updates the UI elements accordingly.
- The **Update** method is called on every frame update and calls the **KeepStable** method to keep the HUD stable in VR environments.
- Public methods **UpdateHealth**, **UpdateStamina**, **UpdateXp**, **UpdateMaxHealth**, **UpdateMaxStamina**, **UpdateMaxXp**, and **UpdateLevel** are provided to update the respective HUD values and trigger UI updates.
- The **UpdateUI** method is a private utility method that calculates the fill amount and updates the visual and text elements accordingly based on the current and maximum values.
- The **HUDOptions** enum defines two options for the HUD type: **Circle** and **Bar**.

- The **KeepStable** method updates the HUD position to stay stable relative to the hand's position.

This code serves as a foundation for managing and updating a HUD in a VR environment, allowing for different visual styles and dynamic changes in HUD type and values.

IconController:

IconController Documentation

The **IconController** class is part of the VRUIP (Virtual Reality User Interface Package) and is responsible for managing an icon button in a VR user interface.

Overview

The **IconController** script provides functionalities for setting up and controlling an icon button in a VR user interface. It allows customization of button colors, icon sprite, and associated sounds. It also provides events for handling button interactions.

Public Properties

Color Properties

- **iconNormalColor** (Color): The normal color of the icon button.
- **iconHoverColor** (Color): The color of the icon button when hovering over it.
- **iconClickColor** (Color): The color of the icon button when clicked.

Icon Properties

- **iconSprite** (Sprite): The sprite to be displayed as the icon on the button.

Public Methods

RegisterOnClick(UnityAction action)

- Description: Register a function to be called when this button is clicked.
- Parameters:
    - **action** (UnityAction): The function to be called.

ClearListeners()

- Description: Clear all listeners from the button.

Events

onClick (UnityEvent)

- Description: Unity event invoked when the button is clicked.

InputController:

InputController Documentation

The **InputController** class is part of the VRUIP (Virtual Reality User Interface Package) and is responsible for managing an input field in a VR user interface.

Overview

The **InputController** script provides functionalities for setting up and controlling an input field in a VR user interface. It allows customization of the input field's appearance, text visibility, and color. It also handles interactions with the keyboard in VR.

Public Properties

Components

- **inputField** (TMP_InputField): The TMP_InputField component associated with the input field.
- **background** (Image): The Image component representing the background of the input field.

Text

- **Text** (string): The current text value of the input field.

Text Visibility

- **TextIsHidden** (bool): Indicates whether the text in the input field is hidden (e.g., for passwords).

Public Methods

ToggleTextHidden(bool hide)

- Description: Toggles the visibility of the text in the input field.
- Parameters:
    - **hide** (bool): Indicates whether the text should be hidden.

Events

None.

Inherited Properties

The **InputController** class inherits properties from the **A_ColorController** class.

Please refer to the documentation for the **A_ColorController** class for more information about inherited properties and methods.

Keyboard:

Keyboard Documentation

The **Keyboard** class is part of the VRUIP (Virtual Reality User Interface Package) and represents a virtual keyboard in a VR user interface.

Overview

The **Keyboard** script provides functionalities for creating and controlling a virtual keyboard in a VR user interface. It allows inputting text into associated input fields or text components. The keyboard supports various button types, including characters, special keys, and control keys like Tab and Enter.

Public Properties

Keyboard Properties

- **fieldType** (KeyboardFieldType): The type of the associated input field or text component.
- **tmpInput** (TMP_InputField): The TMP_InputField component associated with the keyboard (if **fieldType** is set to **TMPInputField**).
- **tmpText** (TextMeshProUGUI): The TextMeshProUGUI component associated with the keyboard (if **fieldType** is set to **TMPText**).
- **tabSpaces** (int): The number of spaces added when the Tab key is clicked.

Keyboard Components

- **background** (Image): The Image component representing the background of the keyboard.
- **normalSection** (GameObject): The GameObject representing the section of the keyboard with normal characters.
- **shiftSection** (GameObject): The GameObject representing the section of the keyboard with shifted characters.
- **contentTransform** (Transform): The Transform component representing the content (buttons) of the keyboard.
- **handles** (GameObject): The GameObject representing the movement handles for the keyboard.

Keyboard Audio

- **audioSource** (AudioSource): The AudioSource component used for playing audio clips.
- **buttonClickSound** (AudioClip): The audio clip played when a keyboard button is clicked.
- **shiftCapsClickSound** (AudioClip): The audio clip played when the Shift or Caps Lock button is clicked.
- **spaceClickSound** (AudioClip): The audio clip played when the Space button is clicked.

- **backspaceClickSound** (AudioClip): The audio clip played when the Backspace or Clear button is clicked.
- **enterClickSound** (AudioClip): The audio clip played when the Enter button is clicked.

Public Methods

Create(Vector3 position, Quaternion rotation, Vector3 contentRotation, bool fadeIn = false)

- Description: Creates an instance of the keyboard at the specified position and rotation.
- Parameters:
    - **position** (Vector3): The position of the keyboard.
    - **rotation** (Quaternion): The rotation of the keyboard.
    - **contentRotation** (Vector3): The rotation of the keyboard's content (buttons).
    - **fadeIn** (bool, optional): Indicates whether the keyboard should fade in upon creation. Default is **false**.
- Returns: The created instance of the keyboard.

ButtonPressed(KeyboardButton button)

- Description: Handles the button press event of the keyboard.
- Parameters:
    - **button** (KeyboardButton): The pressed button.

Events

None.

Inherited Properties

The **Keyboard** class inherits properties from the **A_Canvas** class.

Please refer to the documentation for the **A_Canvas** class for more information about inherited properties and methods.

LoginController:

LoginController Documentation

The **LoginController** class is a script that controls the functionality of a login UI canvas. It handles user interactions such as logging in, signing up, and toggling the visibility of the password field.

Overview

The **LoginController** script provides functionalities for managing the login UI, including handling input fields, buttons, and toggles. It allows users to log in, sign up, and navigate between the login and sign-up pages.

Public Properties

- **background** (Image): The Image component representing the background of the login UI.
- **usernameInputField** (InputController): The InputController component representing the username input field.
- **passwordInputField** (InputController): The InputController component representing the password input field.
- **showPasswordToggle** (ToggleController): The ToggleController component representing the toggle for showing/hiding the password.
- **loginButton** (ButtonController): The ButtonController component representing the login button.
- **signUpButton** (ButtonController): The ButtonController component representing the sign-up button.
- **signUpNewUserButton** (ButtonController): The ButtonController component representing the button for signing up a new user.
- **backIcon** (IconController): The IconController component representing the back icon for navigating back to the login page.
- **loginPage** (GameObject): The GameObject representing the login page.
- **signUpPage** (GameObject): The GameObject representing the sign-up page.

Public Methods

Login()

- Description: Function for logging in. Implement your login logic here.

SignUp()

- Description: Function for signing up. Implement your sign-up logic here.

Private Methods

Awake()

- Description: Called when the script instance is being loaded. Sets up the login UI.

SetupLogin()

- Description: Sets up the login UI by registering event handlers for buttons and toggles.

ToggleHidePassword(bool hide)

- Description: Toggles the visibility of the password field based on the value of the toggle.

BackToLoginPage()

- Description: Navigates back to the login page and hides the sign-up page.

SetColors(ColorTheme theme)

- Description: Overrides the base class method to set the colors of the login UI elements based on the provided theme.

Events

None.

Inherited Properties

The **LoginController** class inherits properties from the **A_Canvas** class.

Please refer to the documentation for the **A_Canvas** class for more information about inherited properties and methods.

Main Menu:

MainMenu Documentation

The **MainMenu** class is a script that controls the functionality of a main menu UI canvas. It handles common buttons such as play and quit, navigation between pages, and event callbacks for play and quit actions.

Overview

The **MainMenu** script provides functionalities for managing the main menu UI, including handling buttons, pages, and event callbacks. It allows users to navigate between different pages within the main menu, trigger play and quit actions, and customize the appearance of the UI.

Public Properties

- **playButton** (Button): The Button component representing the play button.
- **quitButton** (Button): The Button component representing the quit button.
- **mainPage** (GameObject): The GameObject representing the main page of the menu.
- **pageButtons** (Button[]): An array of Button components representing the buttons that navigate to different pages in the menu.
- **pages** (GameObject[]): An array of GameObjects representing the different pages in the menu.
- **onPlay** (UnityEvent): An event that is invoked when the play button is clicked.
- **onQuit** (UnityEvent): An event that is invoked when the quit button is clicked.

- **background** (Image): The Image component representing the background of the main menu UI.

Public Methods

SwitchToMainPage()

- Description: Switches the UI to the main page, hiding all other pages.

Private Methods

Start()

- Description: Called when the script instance is being loaded. Initializes the main menu buttons.

SetColors(ColorTheme theme)

- Description: Overrides the base class method to set the colors of the main menu UI elements based on the provided theme.

InitializeButtons()

- Description: Initializes the play button, quit button, and page buttons by assigning event handlers.

SwitchToPage(GameObject page)

- Description: Switches the UI to the specified page, hiding other pages.

OnPlayButton()

- Description: Event handler for the play button. Invokes the **onPlay** event.

OnQuitButton()

- Description: Event handler for the quit button. Invokes the **onQuit** event and quits the application.

Events

- **onPlay** (UnityEvent): An event that is invoked when the play button is clicked.
- **onQuit** (UnityEvent): An event that is invoked when the quit button is clicked.

Inherited Properties

The **MainMenu** class inherits properties from the **A_Canvas** class.

Please refer to the documentation for the **A_Canvas** class for more information about inherited properties and methods.

OptionController:

OptionController Documentation

The **OptionController** class is a script that controls the functionality of an option UI element. It allows users to navigate through a list of options and displays the currently selected option.

Overview

The **OptionController** script provides functionalities for managing an option UI element. It includes methods for navigating through a list of options, updating the displayed option text, and customizing the appearance of the UI.

Public Properties

- **optionText** (TextMeshProUGUI): The TextMeshProUGUI component representing the text of the currently selected option.
- **optionLabel** (TextMeshProUGUI): The TextMeshProUGUI component representing the label of the option.
- **leftArrowButton** (IconController): The IconController component representing the left arrow button for navigating to the previous option.
- **rightArrowButton** (IconController): The IconController component representing the right arrow button for navigating to the next option.
- **options** (string[]): An array of strings representing the available options.
- **optionLabelText** (string): The label text displayed above the option.

Public Methods

None.

Private Methods

Awake()

- Description: Called when the script instance is being loaded. Sets up the option UI element.

SetupOption()

- Description: Sets up the option UI element by assigning event handlers and initializing the option text.

NavigateLeft()

- Description: Navigates to the previous option in the options array and updates the displayed option text.

NavigateRight()

- Description: Navigates to the next option in the options array and updates the displayed option text.

SetColors(ColorTheme theme)

- Description: Overrides the base class method to set the colors of the option UI elements based on the provided theme.

Public Properties

- **CurrentOption** (string): The currently selected option.

Inherited Properties

The **OptionController** class inherits properties from the **A_ColorController** class.

Please refer to the documentation for the **A_ColorController** class for more information about inherited properties and methods.

SliderController

SliderController Documentation

The **SliderController** class is a script that controls the functionality of a slider UI element. It allows users to interact with the slider and provides options for customizing its appearance.

Overview

The **SliderController** script provides functionalities for managing a slider UI element. It includes methods for setting the slider value, displaying the value as a percentage, customizing the appearance of the slider, and handling events when the value of the slider changes.

Public Properties

- **showPercentage** (bool): Determines whether to show the percentage text representation of the slider value.
- **showText** (bool): Determines whether to show the slider text value.
- **showIcon** (bool): Determines whether to show the slider icon.
- **iconClickable** (bool): Determines whether the slider icon is clickable and can set the slider to 0 or 100%.
- **showDecimals** (bool): Determines whether to display decimals in the percentage text.
- **sliderTextValue** (string): The text value displayed on the slider.
- **emptyColor** (Color): The color of the empty portion of the slider.
- **fillColor** (Color): The color of the filled portion of the slider.
- **pressedColor** (Color): The color of the slider when it is pressed.

- **textColor** (Color): The color of the slider text and percentage text.
- **slider** (Slider): The Slider component representing the slider UI element.
- **sliderIcon** (IconController): The IconController component representing the slider icon.
- **percentageText** (TextMeshProUGUI): The TextMeshProUGUI component representing the percentage text.
- **sliderText** (TextMeshProUGUI): The TextMeshProUGUI component representing the slider text.
- **emptyImage** (Image): The Image component representing the empty portion of the slider.
- **fillImage** (Image): The Image component representing the filled portion of the slider.

Public Methods

- **SetValue(float value, bool notify)**: Sets the value of the slider.
- **RegisterOnChanged(UnityAction<float> action)**: Registers a callback to be invoked when the value of the slider changes.

Private Methods

Awake()

- Description: Called when the script instance is being loaded. Sets up the slider UI element.

SetupSlider()

- Description: Sets up the slider UI element by assigning event handlers and initializing the appearance and values.

SetColors(ColorTheme theme)

- Description: Overrides the base class method to set the colors of the slider UI elements based on the provided theme.

SetPercentageText(float percentage)

- Description: Updates the percentage text to display the current value of the slider as a percentage.

LogoClicked()

- Description: Handles the click event on the slider icon. Sets the slider value to 0 or 100% depending on its current value.

Inherited Properties

The **SliderController** class inherits properties from the **A_ColorController** class.

Please refer to the documentation for the **A_ColorController** class for more information about inherited properties and methods.

SwitchController:

SwitchController Documentation

The **SwitchController** class is a script that controls the functionality of a switch UI element. It allows users to toggle between two states (on/off) and provides options for customizing its appearance and behavior.

Overview

The **SwitchController** script provides functionalities for managing a switch UI element. It includes methods for setting the switch state, handling events when the state changes, and customizing the appearance of the switch.

Public Properties

- **backgroundOnColor** (Color): The color of the switch background when it is in the "on" state.
- **circleOnColor** (Color): The color of the switch circle when it is in the "on" state.
- **backgroundOffColor** (Color): The color of the switch background when it is in the "off" state.
- **circleOffColor** (Color): The color of the switch circle when it is in the "off" state.
- **enableStateText** (bool): Determines whether to enable the text that displays the current state of the switch.
- **customStateText** (bool): Determines whether to use custom text for the switch state.
- **onStateText** (string): The custom text to display when the switch is in the "on" state.
- **offStateText** (string): The custom text to display when the switch is in the "off" state.
- **switchSound** (AudioClip): The sound clip to play when the switch state changes.
- **onTurnOn** (UnityEvent): An event invoked when the switch is turned on.
- **onTurnOff** (UnityEvent): An event invoked when the switch is turned off.
- **toggle** (Toggle): The Toggle component representing the switch UI element.
- **background** (Image): The Image component representing the background of the switch.
- **circleImage** (Image): The Image component representing the circle of the switch.
- **stateText** (TextMeshProUGUI): The TextMeshProUGUI component representing the state text.
- **audioSource** (AudioSource): The AudioSource component used to play the switch sound.

Public Methods

- **IsOn**: Returns a boolean indicating whether the switch is currently in the "on" state.
- **SetupSwitch()**: Sets up the switch UI element by assigning event handlers and initializing the appearance and values.
- **TurnOn()**: Turns on the switch.
- **TurnOff()**: Turns off the switch.
- **RegisterOnValueChanged(UnityAction<bool> action)**: Registers a callback to be invoked when the value of the switch changes.

Private Methods

Awake()

- Description: Called when the script instance is being loaded. Sets up the switch UI element.

HandleToggleChanged(bool on)

- Description: Handles the toggle value changed event. Plays the switch sound and calls the appropriate methods based on the new state.

Inherited Properties

The **SwitchController** class inherits properties from the **A_UIIntercations** class.

Please refer to the documentation for the **A_UIIntercations** class for more information about inherited properties and methods.

TabController:

TabController Documentation

The **TabController** class is a script that manages the functionality of a tabbed UI interface. It allows users to switch between different tabs and control the visibility of tab panels.

Overview

The **TabController** script provides functionalities for managing a tabbed UI interface. It includes methods for selecting tabs, initializing tab buttons, and controlling the visibility of tab panels.

Public Properties

- **tabs** (Tab[]): An array of **Tab** objects representing the tabs in the tabbed interface.
- **tabsBackground** (Image): The **Image** component representing the background of the tabbed interface.

Public Methods

- **SelectTab(int index)**: Selects the tab at the given index in the **tabs** array.
- **SelectTab(TabButtonController tabButton)**: Selects the specified tab button and deselects all other tabs.
- **SetColors(ColorTheme theme)**: Overrides the base method to set the colors of the tab background.

Private Methods

Awake()

- Description: Called when the script instance is being loaded. Initializes the tab buttons.

Start()

- Description: Overrides the base method to select the first tab when the scene starts.

InitializeTabButtons()

- Description: Initializes the tab buttons by calling the **Initialize()** method on each tab button.

SelectTab(int index)

- Description: Selects the tab at the specified index. Sets the corresponding tab button as selected and shows its associated panel. Deselects all other tab buttons and hides their panels.

SelectTab(TabButtonController tabButton)

- Description: Selects the specified tab button. Sets the button as selected and shows its associated panel. Deselects all other tab buttons and hides their panels.

Inherited Properties

The **TabController** class inherits properties from the **A_ColorController** class.

Please refer to the documentation for the **A_ColorController** class for more information about inherited properties and methods.

Nested Class

Tab

- Description: Represents a tab in the tabbed interface, consisting of a tab button and its associated panel.
- Properties:
    - **button** (TabButtonController): The **TabButtonController** component representing the tab button.
    - **panel** (TabPanelController): The **TabPanelController** component representing the tab panel.

TextController:

TextController Documentation

The **TextController** class is a script that provides typing animation functionality for text elements. It allows the text to be gradually displayed as if it is being typed out.

Public Methods

- **StartTyping()**: Starts the typing animation.

Public Properties

- **enableTyping** (bool): If enabled, the text will be empty on start and will wait for a function call to start typing.
- **typingOnAwake** (bool): If enabled, the typing animation will start automatically on Awake.
- **pauseLongerOnPeriod** (bool): If enabled, the typing animation will pause for a longer duration after each period ('.') character.
- **delayBetweenLetters** (float): The delay (in seconds) between each typed character.

Events

- **onTypingFinished** (UnityEvent): An event invoked when the typing animation finishes.

Components

- **text** (TextMeshProUGUI): The **TextMeshProUGUI** component that displays the text.

Inherited Properties

The **TextController** class inherits properties from the **A_ColorController** class.

Please refer to the documentation for the **A_ColorController** class for more information about inherited properties and methods.

ToggleController:

ToggleController Documentation

The **ToggleController** class is a script that manages the appearance and behavior of a toggle control. It provides functionality to change the color and style of the toggle based on its state.

Public Methods

- **RegisterOnToggleChanged(UnityAction<bool> action)**: Registers a listener for when the toggle value changes.

Public Properties

- **IsOn** (bool): Indicates whether the toggle is currently in the "on" state.
- **textNormalColor** (Color): The color of the toggle text in the normal state.
- **boxNormalColor** (Color): The color of the toggle box in the normal state.
- **textSelectedColor** (Color): The color of the toggle text in the selected state.
- **boxSelectedColor** (Color): The color of the toggle box in the selected state.
- **checkmarkColor** (Color): The color of the toggle checkmark.
- **crossOutSelected** (bool): If enabled, the selected toggle text will be displayed with a strikethrough style.
- **toggleSound** (AudioClip): The audio clip to play when the toggle value changes.

Components

- **toggle** (Toggle): The Unity UI Toggle component.
- **boxImage** (Image): The image component representing the toggle box.
- **checkmarkImage** (Image): The image component representing the toggle checkmark.
- **toggleText** (TextMeshProUGUI): The TextMeshProUGUI component representing the toggle text.
- **audioSource** (AudioSource): The AudioSource component used to play the toggle sound.

Inherited Properties

The **ToggleController** class inherits properties from the **A_ColorController** class.

Please refer to the documentation for the **A_ColorController** class for more information about inherited properties and methods.

UniversalMenu:

UniversalMenu Documentation

The **UniversalMenu** class is a script that represents a universal menu canvas in a VR user interface. It provides functionality to open and close windows within the menu and manage the appearance of the menu components.

Public Methods

- **OpenWindow(UniversalMenuWindow window)**: Opens a specific window in the universal menu.

- **EnableOrDisableObject(GameObject obj)**: Enables or disables a GameObject based on its current state.

Components

- **time** (TextMeshProUGUI): The TextMeshProUGUI component used to display the current time.
- **background** (Image): The Image component representing the background of the menu.
- **volumeBackground** (Image): The Image component representing the background of the volume control.

Private Fields

- **_currentTime** (DateTime): The current time used for updating the time display.
- **_windowIsTransitioning** (bool): Indicates whether a window transition is currently in progress.

Windows

The **UniversalMenu** class contains an array of **UniversalMenuWindow** objects representing the windows within the menu. Each **UniversalMenuWindow** object has its own properties and functionality for managing a specific window in the menu.

Inherited Properties

The **UniversalMenu** class inherits properties from the **A_Canvas** class.

Please refer to the documentation for the **A_Canvas** class for more information about inherited properties and methods.

VideoPlayerController:

VideoPlayerController Documentation

The **VideoPlayerController** class is a script that controls the functionality of a video player in a VR user interface. It provides controls for playing and pausing the video, adjusting the volume, and displaying the video progress.

Public Methods

- **OnVideoClicked()**: Called when the video is clicked. Toggles the visibility of the overlay.
- **BeginDragSlider()**: Called when the slider for adjusting the video progress starts being dragged.
- **EndDragSlider()**: Called when the slider for adjusting the video progress finishes being dragged.
- **SetVideo(VideoClip videoClip, string title = "", float startingTime = 0)**: Sets the video to be played, optionally specifying the title and starting time.

Components

- **videoPlayer** (VideoPlayer): The VideoPlayer component responsible for playing the video.
- **overlay** (CanvasGroup): The CanvasGroup component representing the overlay that appears when the video is clicked.
- **playButton** (IconController): The IconController component for the play/pause button.
- **volumeButton** (IconController): The IconController component for the volume button.
- **videoSlider** (Slider): The Slider component used for adjusting the video progress.
- **timeText** (TextMeshProUGUI): The TextMeshProUGUI component displaying the current video time.
- **titleText** (TextMeshProUGUI): The TextMeshProUGUI component displaying the video title.
- **volumeSlider** (SliderController): The SliderController component used for adjusting the volume.
- **playSprite** (Sprite): The sprite for the play icon.
- **pauseSprite** (Sprite): The sprite for the pause icon.
- **playButtonBackground** (Image): The Image component representing the background of the play button.
- **volumeButtonBackground** (Image): The Image component representing the background of the volume button.
- **handleImage** (Image): The Image component representing the handle of the video progress slider.

Private Fields

- **_overlayActive** (bool): Indicates whether the overlay is currently active (visible).
- **_inTransition** (bool): Indicates whether a transition (fade-in/fade-out) animation is currently in progress.
- **_playing** (bool): Indicates whether the video is currently playing.
- **_blockHideOverlayCounter** (bool): Indicates whether the hide overlay counter is blocked (prevents overlay from hiding).
- **_preSliderVideoIsPlaying** (bool): Indicates whether the video was playing before dragging the video progress slider.
- **_currentVideoLengthString** (string): The formatted string representing the length of the current video.
- **_previousTime** (double): The previously recorded time of the video player.
- **_overlayHideCounter** (float): Counter for hiding the overlay after a certain time of inactivity.

Inherited Properties

The **VideoPlayerController** class inherits properties from the **A_Canvas** class.

Please refer to the documentation for the **A_Canvas** class for more information about inherited properties and methods.

VRCanvas:

VRCanvas Documentation

The **VRCanvas** class represents a canvas in a VR user interface. It extends the **A_Canvas** class and provides additional functionality for scaling the canvas by dragging its corners.

Public Fields

- **backgroundColor** (Color): The background color of the canvas.

Private Fields

- **isScalable** (bool): Indicates whether the canvas is scalable by dragging its corners.
- **minSize** (Vector2): The minimum size the canvas can be scaled to.
- **maxSize** (Vector2): The maximum size the canvas can be scaled to.
- **background** (Image): The Image component representing the background of the canvas.
- **_scaleButtonPrefab** (ScaleUIButton): The prefab used for creating scale buttons.

Methods

- **SetupCanvas()**: Sets up the canvas by applying the background color.
- **SetColors(ColorTheme theme)**: Overrides the base class method to set the background color based on the specified color theme.
- **InitializeScaling()**: Initializes the canvas scaling functionality by creating scale buttons for each corner of the canvas.

Inherited Properties

The **VRCanvas** class inherits properties from the **A_Canvas** class.

Please refer to the documentation for the **A_Canvas** class for more information about inherited properties and methods.

ColorTheme:

ColorTheme Documentation

The **ColorTheme** class represents a color theme used in a VR user interface. It contains four color properties: **primaryColor**, **secondaryColor**, **thirdColor**, and **fourthColor**, which define the colors used for various UI elements.

Fields

- **primaryColor** (Color): The most intense color in the theme, typically used for backgrounds.
- **secondaryColor** (Color): The opposite color of the primary color, typically used for text.
- **thirdColor** (Color): A less intense version of the primary color, often used for normal state UI elements like buttons.
- **fourthColor** (Color): A different color used for special UI elements or pressed state of buttons.

Constructor

- **ColorTheme(Color primary, Color secondary, Color third, Color fourth)**: Creates a new color theme with the specified colors for **primaryColor**, **secondaryColor**, **thirdColor**, and **fourthColor**.

Usage

You can create a new **ColorTheme** instance by providing the desired colors:

Code:

```
Color primary = new Color(0.5f, 0.7f, 1f);
Color secondary = new Color(1f, 1f, 1f);
Color third = new Color(0.7f, 0.9f, 1f);
Color fourth = new Color(0.2f, 0.5f, 1f);
ColorTheme theme = new ColorTheme(primary, secondary, third, fourth);
```

Once you have a **ColorTheme** instance, you can use it to set the colors of UI elements in your VR user interface.

Note: The specific usage of the color properties may vary depending on the implementation of the UI elements in your project.

ButtonVR:

ButtonVR Script Documentation

Public Properties

button

- Type: GameObject
- Description: The GameObject representing the button.

onPress

- Type: UnityEvent
- Description: UnityEvent invoked when the button is pressed.

onRelease

- Type: UnityEvent
- Description: UnityEvent invoked when the button is released.

Public Methods

SpawnSphere()

- Description: Creates a sphere GameObject.
- Functionality:
    - Creates a primitive sphere GameObject with a scale of 0.5 in all dimensions.
    - Sets the position of the sphere GameObject.
    - Adds a Rigidbody component to enable physics simulation.

Abstract Classes:

A_Canvas:

Any class that is UI related and has a main canvas can inherit from the 'A_Canvas' class. By doing that a canvas component will appear in the inspector of that script. After you assign the target canvas to the canvas slot, the 'A_Canvas' class will do all the setup for your canvas to be VR Ready depending on the target framework that you selected in the VRUIPManager.

A_ColorController:

Any class that inherits from this class will be required to implement a function SetColors that will tell the VRUIPManager what elements of this class will take what color from the current ColorTheme. For example, this is the SetColors function the the KeyboardButton class, which inherits from A_ColorController:

```
protected override void SetColors(ColorTheme theme)
{
    buttonBackground.color = theme.thirdColor;
    buttonText.color = theme.secondaryColor;
}
```

We're telling the VRUIPManager that we want to set the button background color to the third color of the theme, and the text color of the button to the secondary color of the theme. Depending on your current theme selection, these colors will differ. This class can be used to further introduce elements into the unified theme UI system of the VRUIPManager.

A_Grabbable:

This abstract class should be used as the parent class for a class that is on a grabbable object. For example, the Eraser script for the drawing tool inherits from this class.

A_UIInteractions:

Any class that inherits from this class can use functions such as RegisterOnEnter, RegisterOnExit, RegisterOnDown, RegisterOnUp, RegisterOnOver, and RegisterOnOff to call certain functions or add certain behavior to a UI element based on pointer/mouse events.