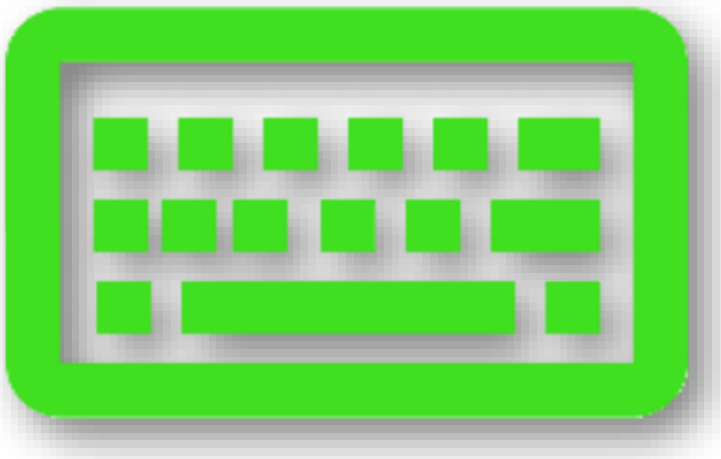

Custom Keyboard Settings

Documentation

_____

Creator: Lucas Gomes Cecchini

Pseudonym: AGAMENOM

Overview

This document will help you use **Assets Custom Keyboard Settings** for **Unity**.

With it, you can easily configure and change your game's **Keyboards** from a **menu**.

It will create two types of **Scriptable Objects**, one for saving data to **PlayerPrefs** and another for accessing the Keyboard via script.

It also has **auto-save** capabilities as soon as settings are changed.

Instructions

You can get more information from the Playlist on YouTube:

https://www.youtube.com/playlist?list=PL5hnfx09yM4Kqkhx0KHyUW0kWviPMTPCs

Script Explanations

KeyboardControlData

The script is split into two main parts: one for defining the data structure **`KeyboardControlData`** and another for handling the editor interface **`KeyboardControlDataEditor`**.

1. `KeyboardControlData` Class

```
public class KeyboardControlData : ScriptableObject
{
    public List<InputData> inputDataList;
}
```

Description

- **`KeyboardControlData`**: This class inherits from **`ScriptableObject`**, which is a Unity-specific class used to store data as assets.
- **`inputDataList`**: This public list holds instances of **`InputData`**, which are used to configure keyboard inputs. Each **`InputData`** object represents a specific keyboard input configuration.

Usage

- **ScriptableObject**: This Unity class is used to create data containers that can be edited in the Unity Editor. **`KeyboardControlData`** is intended to store and manage configurations related to keyboard controls.

2. `KeyboardControlDataEditor` Class

```
public class KeyboardControlDataEditor
{
    [MenuItem("Window/Custom Keyboard Settings/Keyboard Control Data")]
    public static void OpenKeyboardSettingsData()
    {

        ScriptableObject settingsData = KeyboardTagHelper.GetKeyboardControlData();


        if (settingsData != null)
        {

            EditorWindow existingWindow = Resources.FindObjectsOfTypeAll<EditorWindow>().FirstOrDefault(window => window.titleContent.text == settingsData.name);

            if (existingWindow != null)
            {
                existingWindow.Focus();
            }
            else
            {
                EditorUtility.OpenPropertyEditor(settingsData);
            }
        }
        else
        {
            Debug.LogError("Failed to find or load KeyboardControlData. Ensure that the ScriptableObject exists and is properly referenced.");
        }
    }
}
```

Description

- **`#if UNITY_EDITOR`**: This preprocessor directive ensures that the code inside this block is only compiled and executed within the Unity Editor, not in the built game.
- **`KeyboardControlDataEditor`**: This class is used to provide custom functionality in the Unity Editor.
- **`[MenuItem("Window/Custom Keyboard Settings/Keyboard Control Data")]`**: This attribute adds a menu item to the Unity Editor. Selecting this menu item will call the **`OpenKeyboardSettingsData`** method.

- **`OpenKeyboardSettingsData`**: This method handles opening the **`KeyboardControlData`** asset in the Unity Property Editor.

Key Operations

1. **Retrieve `KeyboardControlData`**: The method uses **`KeyboardTagHelper.GetKeyboardControlData()`** to obtain an instance of **`KeyboardControlData`**.
2. **Find or Open Property Editor**:
   - **Find Existing Window**: The script attempts to locate an already open Property Editor window for the **`KeyboardControlData`** instance.
   - **Focus or Open New Window**: If the window is found, it brings it into focus. If not, it opens a new Property Editor window using **`EditorUtility.OpenPropertyEditor()`**.
3. **Error Handling**: If **`KeyboardControlData`** cannot be found, an error message is logged in the Unity Console.

Summary

- **`KeyboardControlData`** is a data container for keyboard configurations.
- **`KeyboardControlDataEditor`** provides a custom editor window for managing the **`KeyboardControlData`** asset, accessible from the Unity Editor's menu.

This setup allows developers to easily manage keyboard control configurations through a dedicated interface in the Unity Editor, facilitating better organization and customization of input settings.

InputData

The **`InputData`** script is a Unity script that defines a class to represent individual keyboard input configurations. Here's a detailed explanation of the script and its components:

Script Purpose

The purpose of the **`InputData`** script is to encapsulate data for a single keyboard input configuration. This configuration includes a tag to identify the input and a key code representing the specific keyboard key. The script is implemented as a **`ScriptableObject`**, allowing the creation of assets that store this data persistently within the Unity Editor.

Script Breakdown

1. Namespace and Imports

```
using UnityEngine;
```

The script starts by importing the **`UnityEngine`** namespace, which is essential for accessing Unity's core engine features, including **`ScriptableObject`** and **`KeyCode`**.

2. Class Declaration

```
public class InputData : ScriptableObject
{
    public string keyboardTag;
    public KeyCode keyboard;
}
```

- The class is declared as **`public`**, making it accessible from other scripts.

- The class inherits from **`ScriptableObject`**, a special Unity class that allows for the creation of custom asset files.

3. Public Fields

```
public string keyboardTag;
public KeyCode keyboard;
```

- **`keyboardTag`** is a public field of type **`string`**. This field is used to store a tag or identifier for the input data. It helps in distinguishing between different input configurations.
- **`keyboard`** is a public field of type **`KeyCode`**. **`KeyCode`** is an enumeration in Unity that represents all possible keys on a keyboard. This field stores the specific key associated with the input configuration.

ScriptableObject Explanation

`ScriptableObject` is a special type of object in Unity designed for holding data. Unlike **`MonoBehaviour`**, **`ScriptableObject`** instances do not need to be attached to GameObjects. They are useful for creating assets that store data which can be easily shared and reused across different parts of your game or application.

Conclusion

The **`InputData`** script provides a simple yet effective way to manage individual keyboard input configurations using Unity's **`ScriptableObject`** system. By storing input data in assets, you can easily configure and reuse keyboard settings across different parts of your game or application. This approach also makes it easier to manage and update input configurations without hardcoding them into scripts.

KeyboardControlDataInspector

This script is a custom Unity editor for managing **`KeyboardControlData`**, a ScriptableObject that holds a list of **`InputData`** objects. The custom editor enhances the Unity Inspector interface for **`KeyboardControlData`** to make it easier to interact with and manage the input data. Here's a detailed explanation of what the script does:

General Overview

The custom editor script provides a user interface within the Unity Editor to:
- Display and edit **`InputData`** entries.
- Detect and assign key codes.
- Create new **`InputData`** assets.
- Save and rename **`InputData`** assets.

Key Components

1. Editor Class:
- The script defines a class **`KeyboardControlDataInspector`** that inherits from Unity's **`Editor`** class.
- This class is annotated with **`[CustomEditor(typeof(KeyboardControlData))]`**, indicating that it customizes the inspector for **`KeyboardControlData`** ScriptableObject.

2. Fields:
- **`isDetectingKey`**: A boolean that indicates if the editor is currently in key detection mode.
- **`detectingIndex`**: Tracks the index of the **`InputData`** entry being edited for key detection.

3. OnInspectorGUI Method:
- This method is overridden to define how the custom inspector should be drawn.
- It initializes the **`inputDataList`** if it's null.

- Displays a read-only field showing the script name.
- Renders a list of **`InputData`** entries with options to edit the **`keyboardTag`** and **`KeyCode`**.
- Provides buttons for deleting entries, detecting key presses, saving all entries, and creating new entries.
- Handles key detection by updating the **`InputData`** with the pressed key code.

4. DeleteInputData Method:
- Deletes an **`InputData`** asset from the project and removes it from the list.
- Uses Unity's **`AssetDatabase`** to delete the asset file and update the list.

5. CreateInputData Method:
- Creates a new **`InputData`** asset and adds it to the list.
- Ensures that the asset is placed in a proper folder and generates a unique file name to avoid conflicts.
- Uses **`AssetDatabase`** to create the new asset and save it.

6. SaveAllInputData Method:
- Saves all **`InputData`** assets and renames them based on their **`keyboardTag`**.
- Counts occurrences of each **`keyboardTag`** to handle naming conflicts.
- Renames files to ensure each asset has a unique name if the **`keyboardTag`** is duplicated.
- Updates and saves assets using **`AssetDatabase`**.

User Interface

- **Read-Only Script Field**: Displays the script name without allowing edits.
- **Input Data List**: Shows a list of **`InputData`** entries with editable fields for **`keyboardTag`** and **`KeyCode`**, and buttons for actions.
- **Action Buttons**:
  - **Delete**: Removes the selected **`InputData`** after confirmation.
  - **Detect**: Enables key detection mode for the selected entry.
  - **Save All**: Saves and renames all **`InputData`** assets.
  - **Create Input Data**: Creates a new **`InputData`** asset and adds it to the list.

Key Detection

- When in key detection mode, the editor waits for a key press and updates the **`InputData`** with the pressed key code.
- The detection mode is toggled with a button, and once a key is pressed, the detection mode ends and the editor updates accordingly.

Overall, this script provides a powerful and user-friendly way to manage **`InputData`** assets within the Unity Editor, facilitating tasks such as editing, creating, and saving input configurations.

KeyboardControlDataCreator

This script is designed to facilitate the creation and management of custom **`KeyboardControlData`** assets within the Unity Editor. Here's a detailed explanation of how it works:

Overview

The script automates the process of creating a **`KeyboardControlData`** asset, which is a type of custom data used to store keyboard control configurations. It also ensures that this asset is created automatically if it doesn't already exist in the project.

Components

1. **KeyboardControlDataCreator Class**:
   - **Purpose**: Provides functionality to create a **`KeyboardControlData`** asset through a custom menu option in Unity.

- **CreateCustomObjectData Method**:
  - **Menu Item**: Adds a new option to the Unity Editor's Asset menu under **"Create/Custom Keyboard Settings/Keyboard Control Data"**.
  - **Asset Path**: Defines the path where the asset will be saved (`**Assets/Resources/Keyboard Control Data.asset**`).
  - **Folder Check**: Verifies if the `**Resources**` folder exists in the project. If not, it creates the folder.
  - **Duplicate Check**: Checks if an asset with the same name already exists at the specified path. If it does, it prompts the user to confirm if they want to replace the existing asset.
  - **Asset Creation**: Creates a new instance of `**KeyboardControlData**`, saves it as an asset at the specified path, and marks it as dirty to ensure it is saved correctly.
  - **Focus Editor**: Automatically focuses on the Project window and selects the newly created asset.

2. **KeyboardControlDataCreatorStartup Class**:
  - **Purpose**: Ensures that the `**KeyboardControlData**` asset is created automatically when the Unity Editor starts, if it doesn't already exist.

  - **Static Constructor**:
    - **Delayed Execution**: Uses a delayed call to check for the asset after the Unity Editor has started up.
    - **Asset Existence Check**: Checks if the asset already exists at the specified path. If it doesn't, it triggers the `**CreateCustomObjectData**` method to create it.

Functionality

1. **Creating the Asset**:
  - The script provides a menu option in the Unity Editor to create a `**KeyboardControlData**` asset.
  - When this menu option is selected, it ensures that the necessary folder structure is in place and handles the creation and saving of the asset.
  - If the asset already exists, the user is prompted to confirm if they want to replace it, avoiding accidental overwrites.

2. **Automatic Asset Creation**:
  - On editor startup, the script checks for the presence of the `**KeyboardControlData**` asset.
  - If the asset does not exist, it automatically creates it, ensuring that the project always has this essential asset available without requiring manual intervention.

Summary

This script streamlines the process of managing `**KeyboardControlData**` assets in Unity by providing a convenient way to create the asset through the editor menu and ensuring its presence in the project through automatic creation on editor startup. This automation simplifies asset management, especially in larger projects where such assets are crucial for keyboard control settings.

KeyboardTagDropdownAttribute

This script introduces a custom attribute and property drawer for Unity's Inspector. The attribute is designed to be used with string properties to provide a dropdown list of keyboard tags. The property drawer manages the rendering of this dropdown and handles validation and warnings.

1. `KeyboardTagDropdownAttribute` Class

```
#if UNITY_EDITOR
using System.Collections.Generic;
using CustomKeyboard;
using UnityEditor;
using UnityEngine;

public class KeyboardTagDropdownAttribute : PropertyAttribute
{
    // This attribute is just a marker, it does not need additional implementation.
}
#endif
```

Description

- **`KeyboardTagDropdownAttribute`**: This class is a custom attribute derived from **`PropertyAttribute`**. It doesn't need any additional implementation beyond marking fields in Unity's Inspector. Its presence indicates that the property should use a custom property drawer.

2. `KeyboardTagDropdownDrawer` Class

```
#if UNITY_EDITOR
[CustomPropertyDrawer(typeof(KeyboardTagDropdownAttribute))]
public class KeyboardTagDropdownDrawer : PropertyDrawer
{
    ...
```

Description

- **`[CustomPropertyDrawer(typeof(KeyboardTagDropdownAttribute))]`**: This attribute specifies that **`KeyboardTagDropdownDrawer`** is the custom drawer for properties marked with `KeyboardTagDropdownAttribute`.

Methods

- **`OnGUI(Rect position, SerializedProperty property, GUIContent label)`**: This method handles the drawing of the property in the Inspector.
  - **Property Type Check**: Ensures the property is a string. If not, it shows an error message.
  - **Retrieve Keyboard Tags**: Uses **`KeyboardTagHelper.GetKeyboardControlData()`** to fetch the list of keyboard tags from **`KeyboardControlData`**.
  - **Dropdown Options**: Constructs a list of tags including a **"Missing Tag"** option to handle cases where the current string doesn't match any tag.
  - **Dropdown Rendering**: Uses **`EditorGUI.Popup`** to display the dropdown and handle user selection.
  - **Warning Message**: Shows a warning if the selected value is **"Missing Tag"**.

- **`GetPropertyHeight(SerializedProperty property, GUIContent label)`**: Determines the height of the property field in the Inspector.
  - **Additional Height**: Adds extra height if the current string value does not match any of the keyboard tags to accommodate the warning message.

Summary

This script enhances Unity's Inspector by providing a custom dropdown for string properties marked with **`KeyboardTagDropdownAttribute`**. It ensures that only valid keyboard tags are selectable and provides feedback if the current value does not match any existing tags. This custom editor functionality improves the usability and reliability of keyboard tag assignments within Unity's development environment.

KeyboardTagHelper

This script provides functionality for managing and persisting keyboard control data within a Unity project. It handles tasks such as retrieving, updating, and saving **`InputData`** associated with specific keyboard tags using Unity's PlayerPrefs system. Here's a detailed breakdown of its functionality:

Overview

The script is designed to work with **`KeyboardControlData`**, a data structure that holds multiple **`InputData`** instances. Each **`InputData`** instance contains a keyboard tag and a corresponding key code. This script facilitates interactions with this data and ensures that changes are saved and loaded correctly.

Components

1. **Retrieving Data**

  - **GetKeyboardControlData**:
    - **Purpose**: Loads the **`KeyboardControlData`** asset from the Resources folder.
    - **Error Handling**: Logs an error if the asset cannot be found or loaded, ensuring that the rest of the operations have a valid data source.

  - **GetInputFromTag**:
    - **Purpose**: Finds and returns the **`InputData`** associated with a specific keyboard tag.
    - **Error Handling**: Logs an error if the **`KeyboardControlData`** asset is null or if no matching **`InputData`** is found.

2. **Updating Data**

  - **SetKey**:
    - **Purpose**: Updates the key code for a given **`InputData`** instance.
    - **Error Handling**: Logs an error if the **`InputData`** instance is null, ensuring that invalid operations are not performed.

  - **SetKeyFromTag**:
    - **Purpose**: Updates the key code for the **`InputData`** associated with a specific tag.
    - **Error Handling**: Logs an error if no matching **`InputData`** is found for the given tag.

3. **Saving and Loading Data**

  - **SaveKeyboardControlData**:
    - **Purpose**: Saves the current state of **`KeyboardControlData`** to PlayerPrefs.
    - **Process**:
      1. Retrieves the **`KeyboardControlData`** asset.
      2. Converts each **`InputData`** in the list to a serializable format (**`InputDataListSave`**).
      3. Serializes the data to JSON and saves it to PlayerPrefs under a specific key ("Keyboard Control Data").

- **Error Handling**: Logs an error if the **`KeyboardControlData`** asset cannot be found.

- **LoadKeyboardControlData**:
  - **Purpose**: Loads and applies saved keyboard control data from PlayerPrefs.
  - **Process**:
    1. Checks if PlayerPrefs contains saved data.
    2. Retrieves and deserializes the JSON data.
    3. Updates the **`KeyboardControlData`** asset with the loaded data, ensuring that existing **`InputData`** instances are updated or logs errors if they cannot be found.
  - **Error Handling**: Logs errors if the **`KeyboardControlData`** asset is not found or if individual **`InputData`** entries cannot be matched with saved data.

Serializable Classes

- **KeyboardControlDataSave**:
  - **Purpose**: A container for storing the serialized **`InputData`** list.
  - **Structure**: Holds a list of **`InputDataListSave`** objects, which represent the saved **`InputData`** instances.

- **InputDataListSave**:
  - **Purpose**: Represents individual **`InputData`** entries for saving and loading.
  - **Structure**: Contains a keyboard tag and a key code, which are serialized into JSON format.

Summary

This script provides a robust utility for managing keyboard control configurations in Unity. It ensures that changes to the control data are saved persistently and can be loaded correctly at runtime. By using PlayerPrefs for storage and providing methods to handle data updates, retrieval, and persistence, the script integrates seamlessly with Unity's editor and runtime systems.

KeyboardSettingsManager and TMP_KeyboardSettingsManager

This script is a Unity component designed to manage keyboard settings through a user interface. It allows users to select, reset, and save key configurations, ensuring that key codes are unique across different managers. Here's a detailed breakdown of its functionality:

Overview

The script handles keyboard settings within Unity by providing a UI for users to select key codes, reset them to defaults, and save these settings. It also manages conflicts between key codes used by different instances of the settings manager.

Key Components

1. **UI Elements**

   - **Buttons and Text**:
     - **Select Button**: Initiates the process of selecting a new key code.
     - **Selected Button Text**: Displays the currently selected key code or indicates the selection process.
     - **Reset Button**: Resets the key code to its default value.

2. **Default Settings**

   - **Default Key Code**: Defines the default key code used when resetting settings.

- **Keyboard Tag**: A unique identifier used to associate the settings with specific **`InputData`** in a data management system.

3. **Current Settings**

- **Current Key Code**: Stores the currently selected key code.

- **Previous Key Code**: Keeps track of the key code before the current selection, used for comparison and UI updates.

- **Listening State**: Indicates whether the script is currently listening for a new key code input.

- **Delay Timer**: Manages a delay before starting to listen for new inputs.

- **Other Managers**: Keeps track of other instances of the **`KeyboardSettingsManager`** in the scene to avoid key code conflicts.

Functionality

1. **Button Click Handlers**

- **OnSelectButtonClick**:
  - Starts a delay before listening for a new key code input.
  - Sets the UI to indicate the key selection process.

- **OnResetButtonClick**:
  - Resets the key code to its default value and saves the updated settings.

2. **Initialization**

- **Start Method**:
  - Initializes a list of all **`KeyboardSettingsManager`** instances in the scene.
  - Sets up button click listeners to ensure they respond to user interactions.
  - Loads the saved settings or applies default settings if no saved data is found.

3. **Settings Management**

- **SetSettings**:
  - Applies saved key code settings and updates the UI to reflect the current selection.

- **SetDefaultSettings**:
  - Resets the key code to the default value and updates the UI accordingly.

4. **Update Method**

- **Update**:
  - Checks for new key inputs if the script is in listening mode.
  - Manages the delay timer to ensure inputs are detected correctly.
  - Enables or disables the reset button based on whether the current key code is different from the default.

5. **Input Handling**

- **ListenForNewInput**:
  - Iterates through possible key codes to detect new inputs.
  - Checks if the new key code is not already used by other managers.
  - Updates the selected key code and saves the settings if a valid new input is detected.

- **IsKeyCodeUsedByOtherManagers**:

- Checks if the current key code is already used by another instance of **`KeyboardSettingsManager`** to prevent conflicts.

6. **Saving Settings**

- **SaveSettings**:
  - Updates the **`InputData`** associated with the current tag with the new key code.
  - Saves the updated settings to persistent storage.

Summary

This script integrates with Unity's UI system to provide a customizable and user-friendly way of managing keyboard settings. It ensures that key codes are unique and avoids conflicts between different settings managers. The script also supports persistence of settings through saving and loading, making it easy to maintain consistent keyboard configurations across sessions.

MovementHandler

This script defines a class, **`MovementHandler`**, which manages directional input for player movement within a game. The class tracks the state of input keys (like forward, backward, left, and right) and calculates the resulting movement direction based on the player's key presses.

Here's a breakdown:

1. **Variables:**
- **`forwardLastPressedTime`**, **`backLastPressedTime`**, **`rightLastPressedTime`**, **`leftLastPressedTime`**:
These variables store the time when each directional key (forward, back, right, left) was last pressed. They start with a value of **`-1f`** to indicate no key has been pressed initially.

- **`forwardInputActive`**, **`backInputActive`**, **`rightInputActive`**, **`leftInputActive`**:
These boolean flags track whether a specific direction key is currently being pressed.

- **`forwardInput`**, **`backInput`**, **`rightInput`**, **`leftInput`**:
These are instances of **`InputData`**, which represent the keyboard keys used for the respective directions. They are initialized in the constructor, and their fields contain information on which key triggers each movement.

2. **Constructor:**
The constructor takes four parameters—**`forward`**, **`back`**, **`right`**, and **`left`**—which represent the input data for each direction. These are then assigned to the corresponding variables.

3. **`GetMovementInput` Method:**
This method returns the movement direction based on the active inputs. It handles the key press and release events and computes the player's movement accordingly.

- Key Down Events:
- If a key is pressed (like the forward key), the script records the current time and marks that input as active (e.g., **`forwardInputActive = true`**).

- Key Events:
- If the key is released, the active flag for that direction is set to **`false`** (e.g., **`forwardInputActive = false`**).

- Movement Calculation:
- The script checks whether the forward or backward key is active and prioritizes the most recently pressed key by comparing the time values (**`forwardLastPressedTime`** and **`backLastPressedTime`**).
- Similarly, it compares right and left key inputs, giving priority to the more recently pressed key.

- Based on which key was pressed last, the method adjusts the movement direction by subtracting the corresponding vector (e.g., **`Vector3.forward`** for forward, **`Vector3.back`** for backward, etc.).

- Normalization:

- After determining the direction based on active inputs, the resulting vector is normalized. This ensures consistent speed regardless of the direction the player is moving in.

Summary

In essence, the script allows players to move in four directions (forward, back, left, right), and it handles cases where multiple keys are pressed by prioritizing the key pressed most recently. The movement is then output as a normalized vector, which can be used to move the player in a consistent direction and speed.