Simple Pie Menu

Table of contents

Introduction....1
1. Prerequisites....2
1.1. Text Mesh Pro....2
1.2. Input System....2
2. Creating the Pie Menu....3
2.1. Getting started....3
2.2. Customization basics....4
2.3. Input settings....5
2.4. Background settings....5
2.5. Shape settings....6
2.6. General settings....6
2.7. Position settings....7
2.8. Menu Item color settings....7
2.9. Info Panel settings....8
2.10. Icons settings....8
2.11. Animations settings....9
2.12. Audio settings....9
2.13. Selection settings....10
2.14. Close functionality settings....10
2.15. Preview Mode settings....11
3. Project structure....12
3.1. Scripts with 'Settings' suffix....12
3.2. Scripts with 'SettingsHandler' suffix....12
3.3. Singleton pattern....12
3.4. PieMenu script....13
4. Instructions....14
4.1. Selecting specific Menu Item in hierarchy....14
4.2. Modifying specific Menu Items....15
4.3. Showing and hiding the Pie Menu....17
4.4. How to change header and details text....18
4.4.1. In editor....18
4.4.2. At runtime....19
4.5. How to disable Menu Items....20
4.6. How to hide Menu Items....22
4.7. How to restore hidden Menu Items....24
4.8. How to add Menu Item at runtime....25
4.9. How to handle clicks on specific Menu Item....27
4.10. How to change icons....28
4.10.1. In editor....28
4.10.2. At runtime....28
4.11. How to add a new Input Device....29
4.11.1. Prerequisite....29
4.11.2. Old Input System....29
4.12.3. New Input System....32
4.12. How to add more shapes....33
4.13. How to add more sounds....35
4.14. How to add more animations....36
4.15. How to add a header to your Pie Menu....37
4.16. How to create a menu that contains only icons....38
4.17. How to change rotation at runtime....39

4.18. Anti-aliasing....40
4.18.1. Canvas render mode....40
4.18.2. Mipmaps....42
4.19. Key Mapping Configuration....44
4.19.1. Old Input System....44
4.19.2. New Input System....44
4.20. Precision of Selection....45
5. Summary....46
5.1. Best practices....46
5.1.1. Performance Optimization....46
5.1.2. Prefab Search Optimization for 'PieMenuContextMenuExtension'....46
5.2. Discord server....47
5.3. Resources....47
5.4. Acknowledgments....47

Legend:

- You need to read it to learn the basics.
- These chapters contain useful information, but you can do without them.
- Read if you intend to modify your menu at runtime.
- Here you will learn how to add custom options tailored to your specific needs.

Introduction

This asset empowers you to effortlessly create a customized Pie Menu in your Unity projects.

The documentation is crafted to offer you a comprehensive grasp of how to efficiently utilize and integrate this tool. No matter if you're an experienced developer or just starting out, you'll find valuable insights and instructions here to help you unlock the full potential of this asset.

1. Prerequisites

1.1. Text Mesh Pro

Before you begin, you need to import TMP Essential Resources. To do this follow these steps:

1. Open Unity.
2. Navigate to the top bar and select 'Window'.
3. From the drop-down menu, choose 'TextMeshPro'.
4. Within the TextMeshPro menu, select 'Import TMP Essential Resources'.

[More details]

1.2. Input System

You can choose between using the Old Input System or the New Input System for interacting with your Pie Menu. However, to avoid errors, please install the Input System, even if you do not plan to use it, in order to correctly import this asset.

Automatic installation:

- When importing the asset into your project, you will be asked if you want to install the required dependencies. Choose 'Yes' to install this package automatically.

Manual installation:

1. Navigate to the Package Manager window by selecting 'Window' in the top bar and then choosing 'Package Manager'.
2. In the Package Manager window, select 'Packages: Unity Registry' tab.
3. Search for 'Input System' in the search field.
4. Once found, click on the package and select 'Install'.

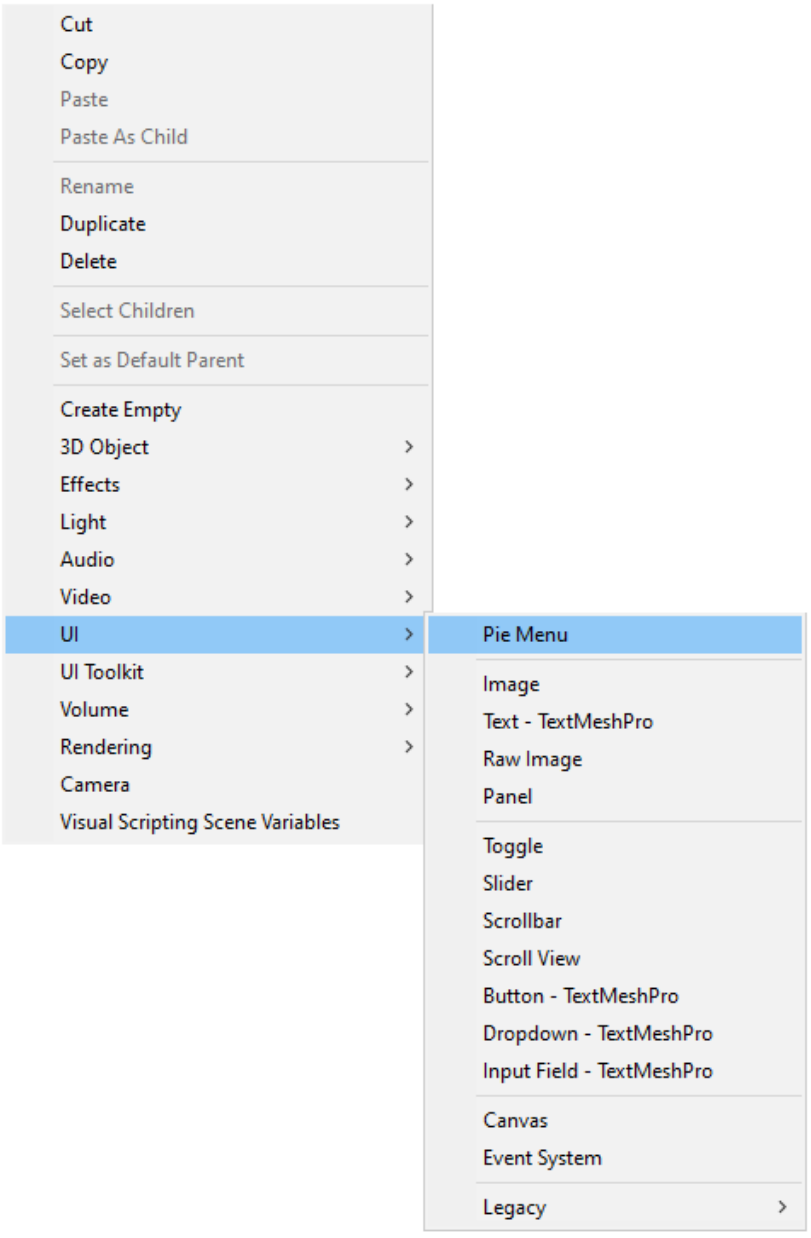
[More details]

2. Creating the Pie Menu

2.1. Getting started

For safety reasons, it's highly recommended to create a Pie Menu in a separate scene as the scripts provided by this asset operate in edit mode and can remove objects from the Hierarchy. While theoretically, nothing bad should happen, it's better to err on the side of caution.

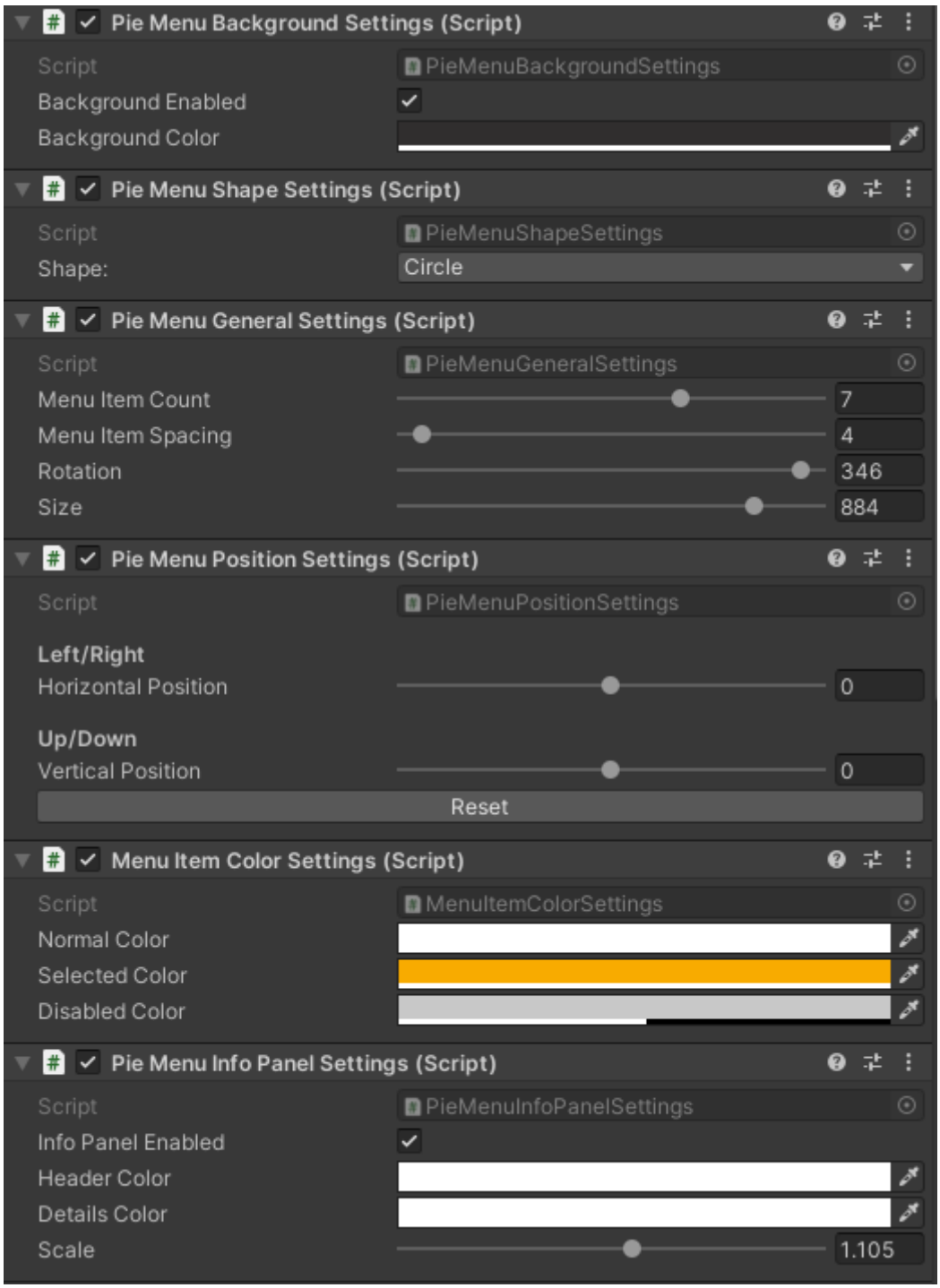
To create a Pie Menu, right-click in the Unity project Hierarchy panel. Then, select 'UI' and choose 'Pie Menu'.



Creating a new Pie Menu

2.2. Customization basics

To start, find the Pie Menu object in the Hierarchy. It is located under the path 'Pie Menu - Canvas/Pie Menu'. Within it, you will find an Inspector that will allow you to customize it according to your requirements.

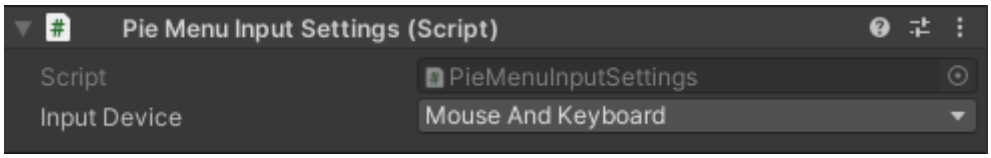


Part of the Pie Menu Inspector

2.3. Input settings

By default, the Pie Menu is controlled using the mouse and keyboard. Users can simply hover the mouse pointer over a Menu Item, and click left mouse button to activate the corresponding action.

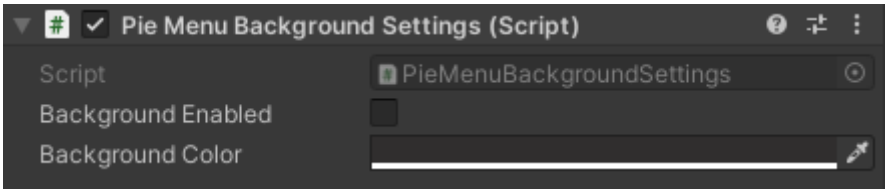
Expanding input device support, such as controllers or VR devices, is simple and will be detailed in a dedicated section of this documentation called 4.11.'How to add a new Input Device'.



Input settings

2.4. Background settings

For your Pie Menu, you can choose to enable or disable the background. Additionally, you can customize its color.

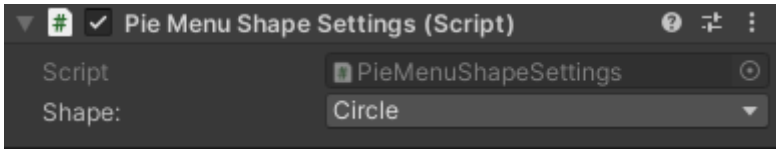


Background settings

If you want more control over the background, such as changing it to an image, locate the object 'Background - Image' in the Hierarchy and make the necessary changes there. The options in this Inspector are just a shortcut to that specific object.

2.5. Shape settings

You have the option to choose from several default shapes for your Pie Menu. Moreover, if you wish to use a custom shape, you can easily create and add your own. This feature will be discussed in more detail in a later section of the documentation titled 4.12.'How to add more shapes'.

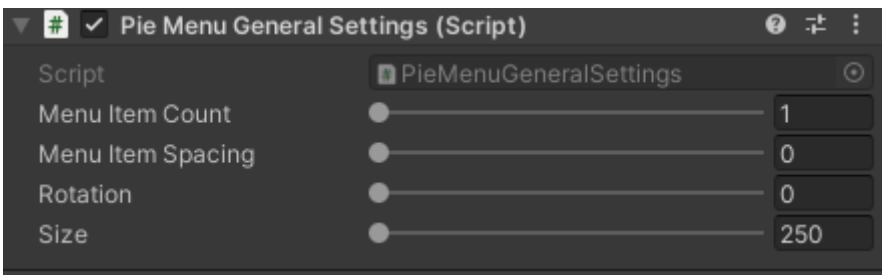


Shape settings

2.6. General settings

General settings consist of:

- **Menu Item Count**: Adjust the number of Menu Items. Please note that this setting is intended for use during the initial menu setup. For managing Menu Items during runtime, please refer to the functions described in 'Adding, Hiding, Disabling, and Restoring Menu Items' sections.
- **Menu Item Spacing**: Define the spacing between individual Menu Items, allowing you to control how closely or distantly they are positioned.
- **Rotation**: Set the rotation of the Pie Menu. When modifying the Menu Item Count or Spacing, the rotation is automatically calculated to ensure symmetry in your menu.
- **Size**: Customize the overall size of the Pie Menu, making it larger or smaller as needed. Set this parameter before modifying the scale of the Info Panel and icons because it resets them to their default sizes.

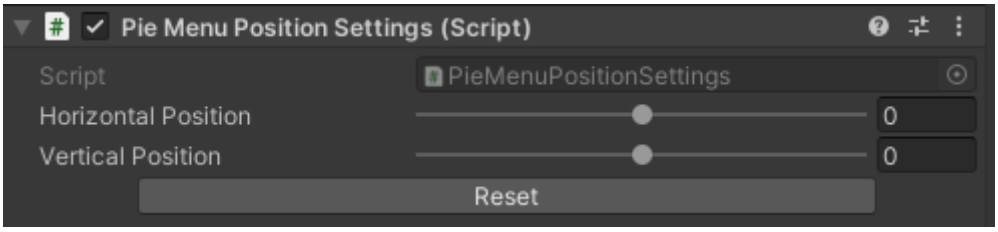


General settings

2.7. Position settings

Using these settings, you can define the position of your menu on the screen.

- **Horizontal Position** - sets the menu from the left to the right side of the screen.
- **Vertical Position** - adjusts the menu from the top to the bottom of the screen.
- **Reset button** - Restores the original position of the menu at the center.



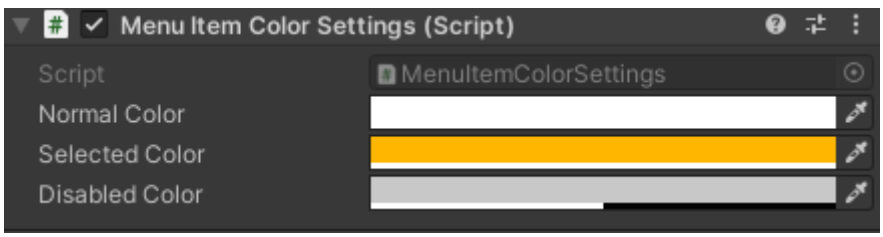
Position settings

Note that some animations, such as Horizontal Swipe and Vertical Swipe, override the menu's position and always set it in the center of the screen.

2.8. Menu Item color settings

Customize color scheme using the following settings:

- **Normal Color**: Define the default color that will be applied to the Menu Items.
- **Selected Color**: Specify the color that will be used to highlight selected Menu Items.
- **Disabled Color**: Choose the color for Menu Items when they are in a disabled state.

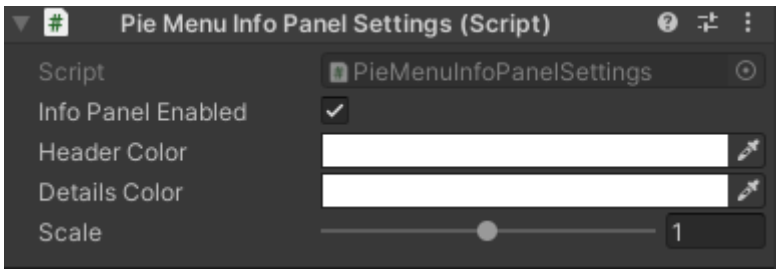


Menu Item Color settings

2.9. Info Panel settings

Info panel consists of the following options:

- **Info Panel Enabled**: Decide whether the Info Panel is enabled or disabled.
- **Header Color**: Specify the color applied to the Info Panel's header.
- **Details Color**: Define the color applied to Info Panel's details section.
- **Scale**: Modify the scale of the Info Panel.

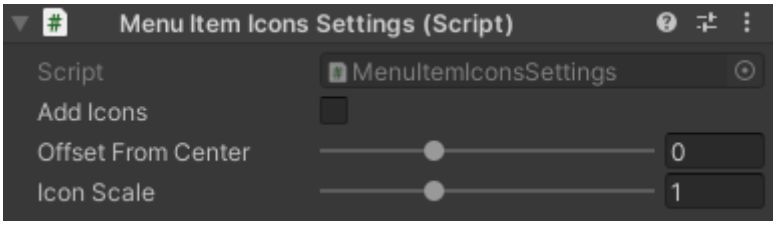


Info Panel settings

2.10. Icons settings

Next, you have the icon settings:

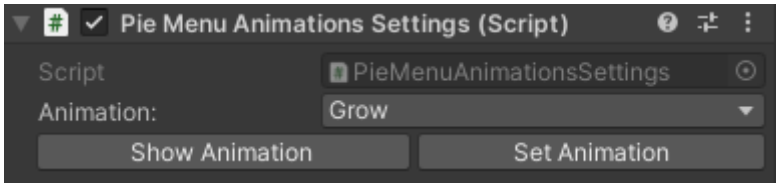
- **Add Icons**: This option allows you to add or remove icons.
- **Offset From Center**: It determines the distance of icons from the center of the Pie Menu.
- **Icon Scale**: This setting allows you to adjust the size of the icons.



Icons settings

2.11. Animations settings

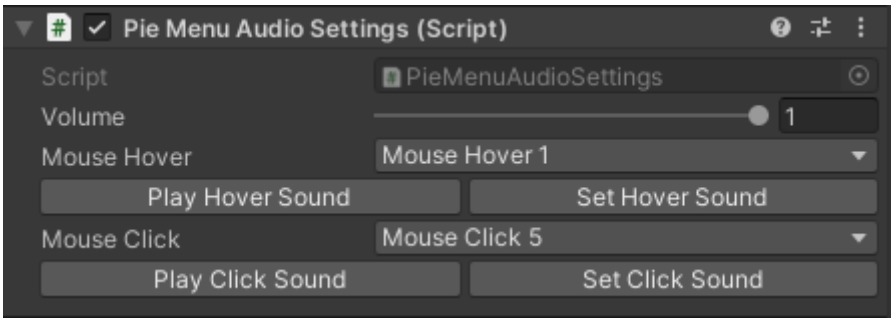
With these settings, you can configure the animation of the Pie Menu as it appears and disappears from the screen. After selecting one of the available animations, you can click '**Show Animation**' to preview how it looks. If you are satisfied, click '**Set Animation**'.



Animations settings

2.12. Audio settings

These settings will allow you to configure sounds for hover and click events. Adjust the volume parameter in case you find a specific sound too loud or quiet. It will be applied within your menu and can have different values for hover and click sounds. You can test the sound by selecting one from the available options in the list and clicking the '**Play Sound**' button. If you like the selected sound, choose '**Set Sound**'.



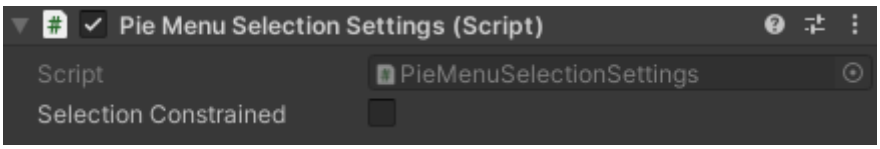
Audio settings

If you can't hear anything, make sure you have sound enabled in the editor scene (it is one of the buttons at the top of the screen)

Mute/Unmute button

2.13. Selection settings

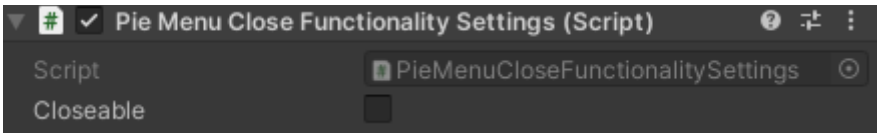
This option disables selection in the center of your Pie Menu. This can be especially useful when you have a large number of options, where even slight mouse movements can easily change the selection.



Selection settings

2.14. Close functionality settings

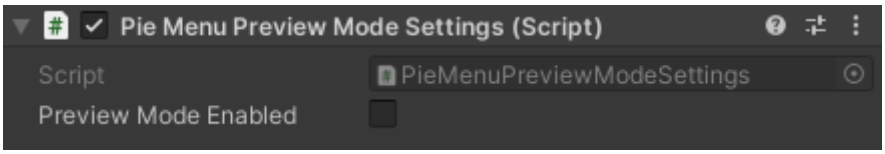
This setting allows players to close the Pie Menu without selecting any of the available options. By default, it is configured to the 'Escape' button for the Mouse and Keyboard input device. Further details on changing button mappings will be provided in the upcoming section of the documentation 4.19. 'Key Mapping Configuration'.



Close funtionality settings

2.15. Preview Mode settings

The purpose of this script is to assist in testing animations and sounds during the configuration of your menu. When enabled, your menu will automatically reopen after it has been closed.



Preview Mode settings

3. Project structure

3.1. Scripts with 'Settings' suffix

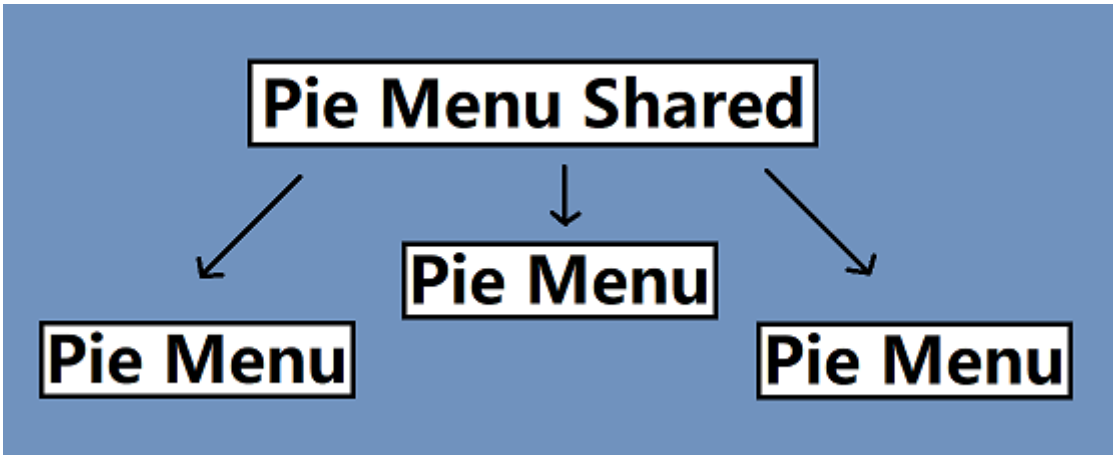
Each script with 'Settings' in its name is responsible for handling serialize fields in the Inspector. These scripts are present on every 'Pie Menu' object, just as shown in the 'Creating the Pie Menu' section.

3.2. Scripts with 'SettingsHandler' suffix

Every script with the name 'Settings' has its corresponding 'SettingsHandler' script responsible for managing changes to those settings. For instance, 'PieMenuShapeSettings' corresponds to 'PieMenuShapeSettingsHandler'. Therefore, if you want to perform actions on your menu through code that can be configured via Inspector settings, you can achieve this by exploring the relevant methods in the corresponding 'SettingsHandler' scripts.

3.3. Singleton pattern

When the first Pie Menu appears in the scene, a singleton 'PieMenuShared' is created, which contains mostly 'SettingsHandlers' but also some other shared components. Its lifecycle is automatically managed, so you don't need to worry about it. It will remain in the scene as long as there is at least one Pie Menu object. Do not delete this object manually, as doing so will result in losing the ability to edit your Pie Menu until the scene is refreshed.



PieMenuShared diagram

3.4. PieMenu script

The PieMenu script, located on the Pie Menu object, is the most important component responsible for managing your menu. It includes the following components:

- **PieMenuInfo**, which contains properties that provide information about the current state of the Pie Menu.
- **PieMenuElements**, which holds references to various elements within the Pie Menu, such as the Info Panel, Animator, and more.
- **MenuItemsTracker**, which allows you to obtain references to each Menu Item belonging to the Pie Menu.
- **MenuItemSelectionHandler**, which is responsible for selecting the Menu Item nearest to the input device's on-screen pointer."

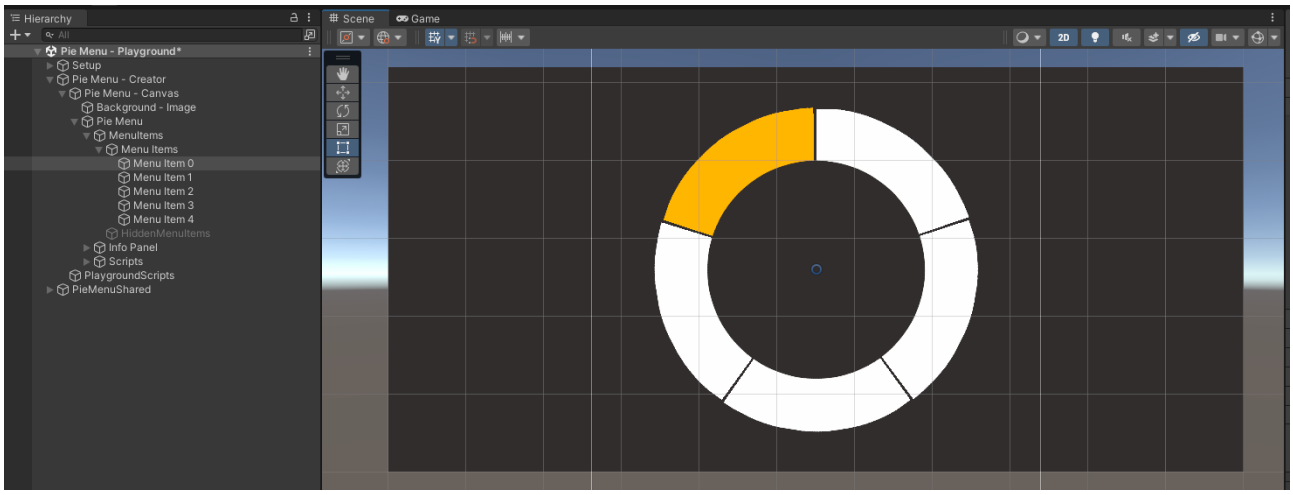
Additionally, the script includes two essential events:

- **OnComponentsInitialized** - This event is triggered after initializing all components in the script. In this project, it is primarily used in scripts with the 'Settings' suffix in their name, which requires references to these components.
- **OnPieMenuFullyInitialized** - This event is particularly significant and is triggered as the earliest point at which you can safely make changes to your Pie Menu after it and all the other components have been fully initialized.

4. Instructions

4.1. Selecting specific Menu Item in hierarchy

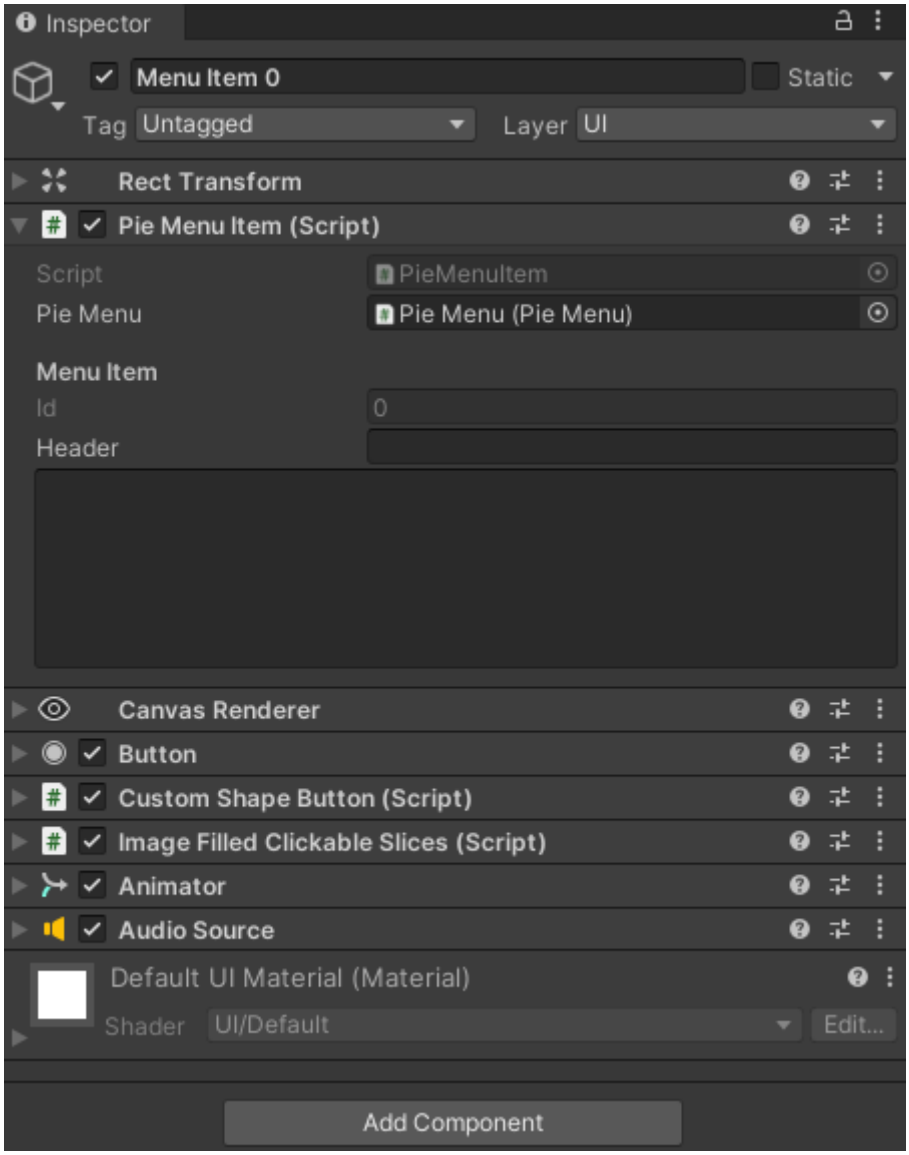
After selecting a Menu Item within your Pie Menu hierarchy, it will be visually highlighted in the scene. This highlighting serves to indicate which Menu Item you are currently modifying. The highlighting color corresponds to the 'Selected Color' which you have previously set in 'Menu Item Color settings'.



Selected Menu Item

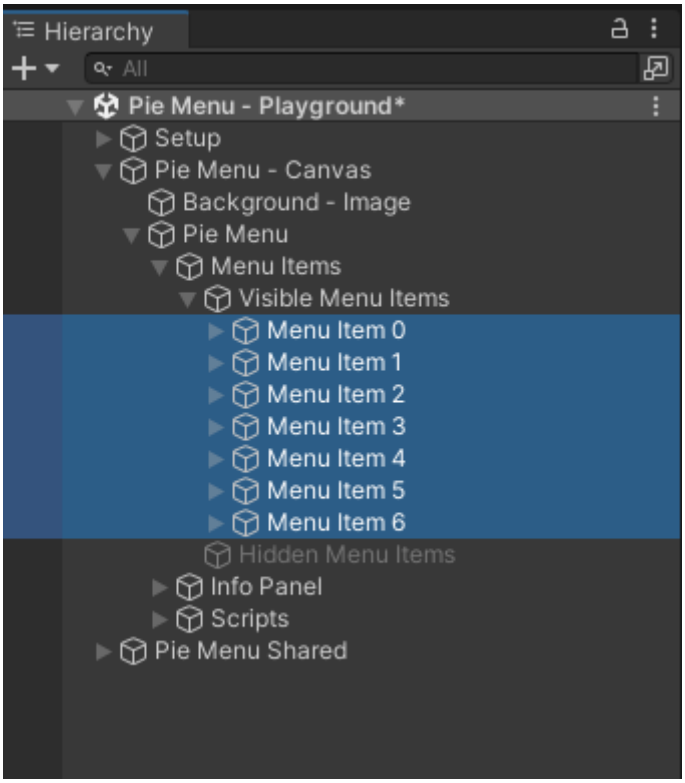
4.2. Modifying specific Menu Items

Every Menu Item of the Pie Menu is stored in the MenuItemsTracker component within the PieMenuItems dictionary. To access a specific Menu Item trough code, you will need its ID. An ID for a Menu Item is assigned after its initialization, and it corresponds to its position in the Hierarchy. To view these IDs, simply inspect the PieMenuItem script located on each of the Menu Items in the Inspector.



Menu Item 0

Typically, specific IDs are also included in the names of Menu Items. However, it's important to note that these values may differ after using the 'hide Menu Item' functionality, as the names are refreshed while the IDs remain unchanged.



Menu Items in the Hierarchy

This is an example of accessing the Menu Item through code.

```
using SimplePieMenu;
using UnityEngine;

public class Example : MonoBehaviour
{
    [SerializeField] PieMenu pieMenu;

    private void OnSomeAction()
    {
        int menuItemId = 3;
        PieMenuItem menuItem =
             pieMenu.MenuItemsTracker.GetMenuItem(menuItemId);

        /// ...
    }
}
```

4.3. Showing and hiding the Pie Menu

To show the menu in your scene:

- Set the Pie Menu Canvas of your created menu to inactive in the Unity Hierarchy.
- Attach the 'PieMenuDisplayer' script to your scene.
- In the Inspector, assign your created Pie Menu to the 'pieMenu' field in the 'PieMenuDisplayer' component.
- Get references to the 'PieMenu' component and 'PieMenuDisplayer' in your code.
- Use the 'ShowPieMenu()' method from the 'PieMenuDisplayer' script as shown in the example below.

```
using UnityEngine;
using SimplePieMenu;

public class Example : MonoBehaviour
{
    [SerializeField] PieMenu pieMenu;

    private PieMenuDisplayer displayer;

    private void Awake()
    {
        displayer = GetComponent<PieMenuDisplayer>();
    }

    private void Update()
    {
        Display();
    }

    private void Display()
    {
        if (Input.GetKeyDown(KeyCode.F))
        {
            displayer.ShowPieMenu(pieMenu);
        }
    }
}
```

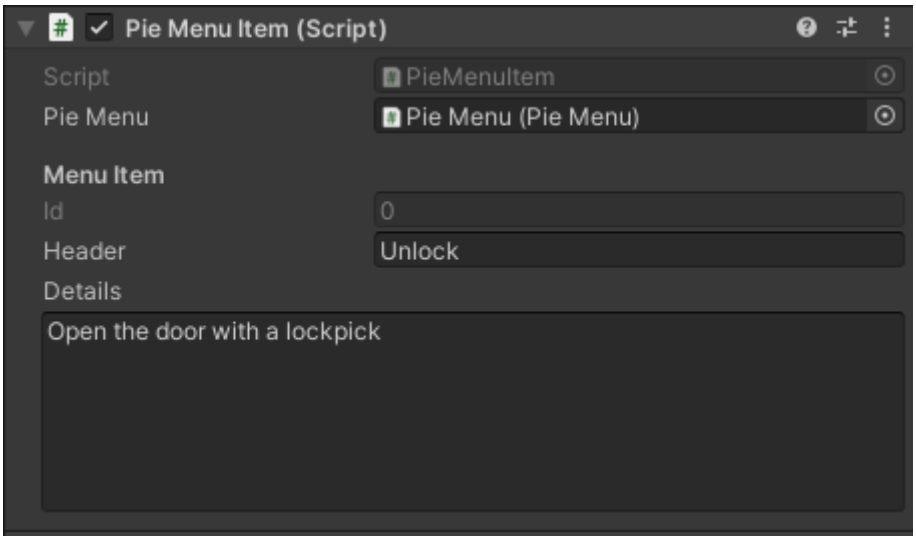
Note: Your menu will automatically hide after selecting one of the available Menu Items or closing it by the user (check section 2.14. Close functionality settings). Also, if you leave your Pie Menu active in the scene, upon entering play mode, its animation will play automatically, and you will be able to interact with it.

I placed two example scenes in the 'demos' folder that demonstrate showing the Pie Menu using both the Old Input System and the New Input System.

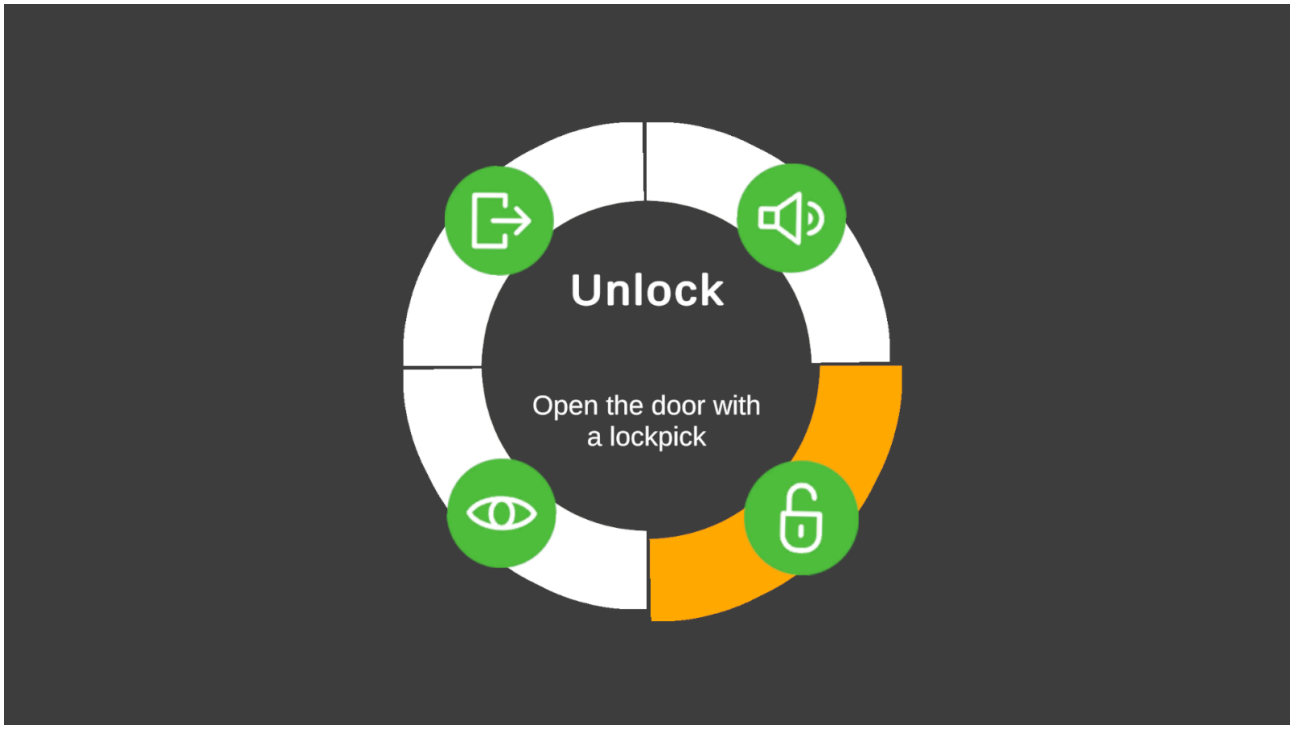
4.4. How to change header and details text

4.4.1. In editor

To set the text in the Header and Details sections, click on the specific Menu Item in the Hierarchy and find the Pie Menu Item script. You will see two [SerializeField] fields there. Fill them in accordingly.



PieMenuItem script



Header and Details on Pie Menu

4.4.2. At runtime

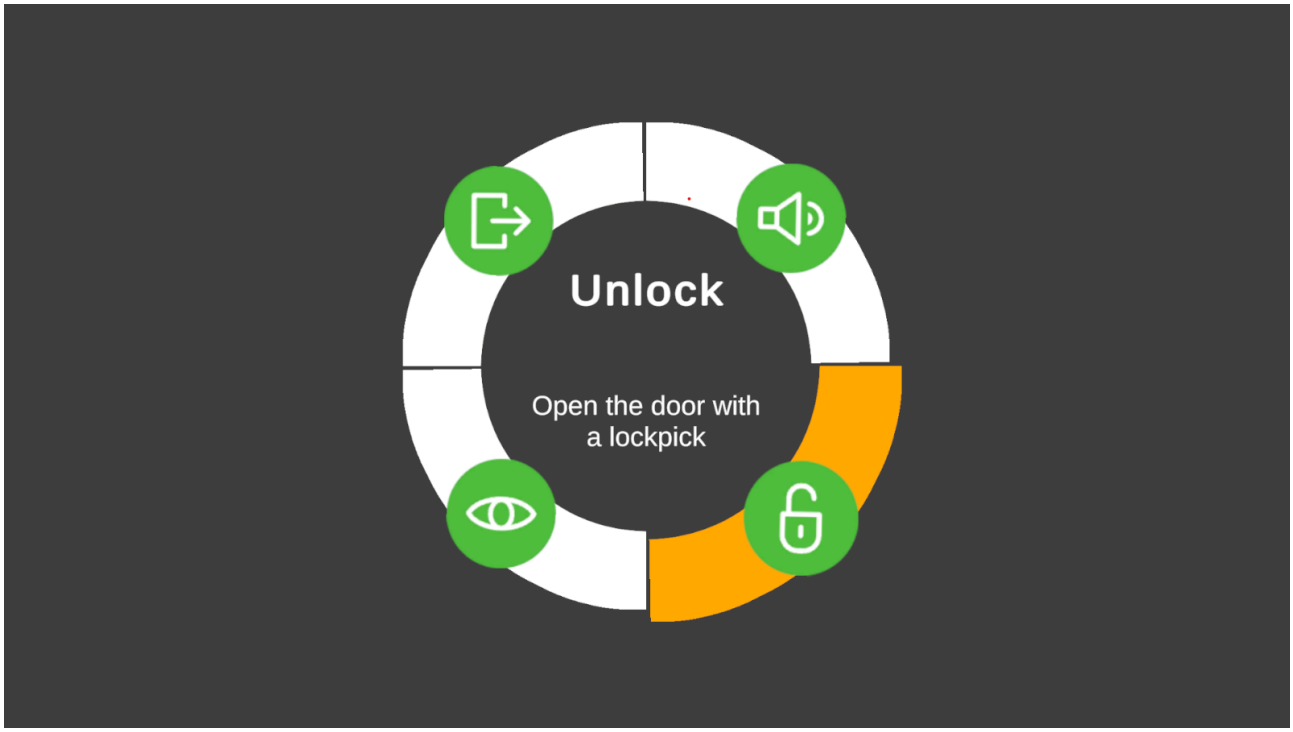
Here's an example of changing the Header and Details text at runtime:

```
using SimplePieMenu;
using UnityEngine;

public class Example : MonoBehaviour
{
    [SerializeField] PieMenu pieMenu;

    private void OnSomeAction()
    {
        int menuItemId = 2;
        PieMenuItem menuItem =
             pieMenu.MenuItemsTracker.GetMenuItem(menuItemId);

        if(menuItem != null)
        {
            menuItem.SetHeader("Unlock");
            menuItem.SetDetails("Open the door with a lockpick");
        }

    }
}
```

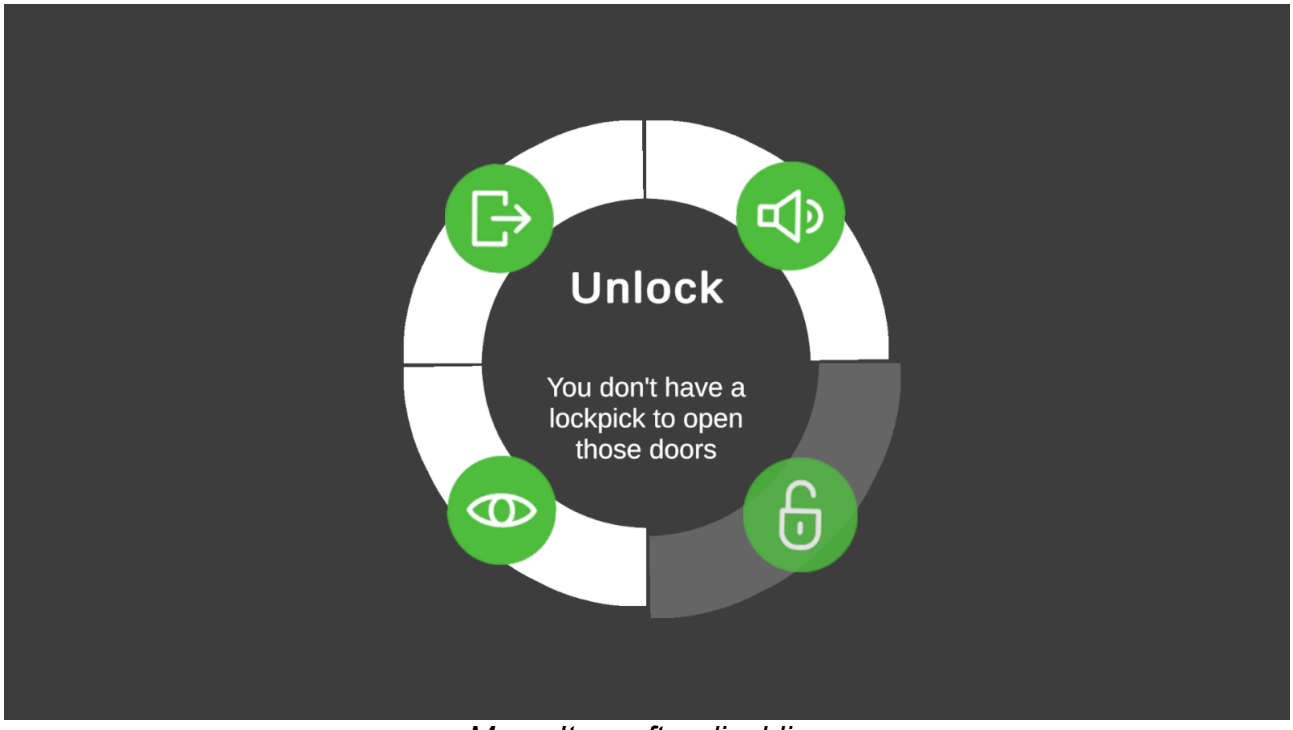
4.5. How to disable Menu Items

This function disables the interaction with a specific Menu Item. You can use it as soon as the 'OnPieMenuFullyInitialized' event is triggered. Here's an example of how to use it:

```
using SimplePieMenu;
using UnityEngine;

public class Example : MonoBehaviour
{
    [SerializeField] PieMenu pieMenu;

    private void Awake()
    {
        pieMenu.OnPieMenuFullyInitialized += DisableMenuItem;
    }

    private void OnDestroy()
    {
        pieMenu.OnPieMenuFullyInitialized -= DisableMenuItem;
    }

    private void DisableMenuItem()
    {
        int menuItemId = 2;
        bool disabled = true;

        MenuItemDisabler disabler =
             PieMenuShared.References.MenuItemsManager.MenuItemDisabler;

        disabler.ToggleDisable(pieMenu, menuItemId, disabled);
    }
}
```



Menu Item before disabling



Menu Item after disabling

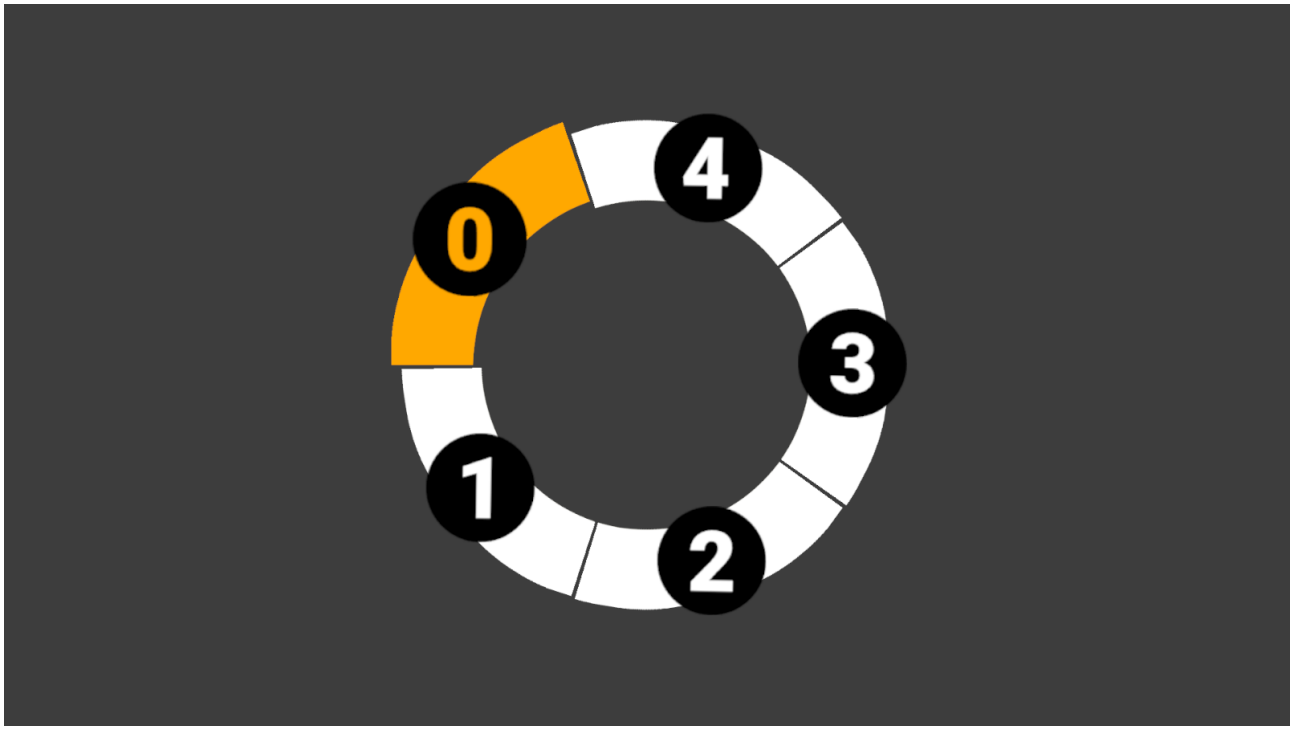
4.6. How to hide Menu Items

This option temporarily removes unnecessary Menu Items from your Pie Menu.

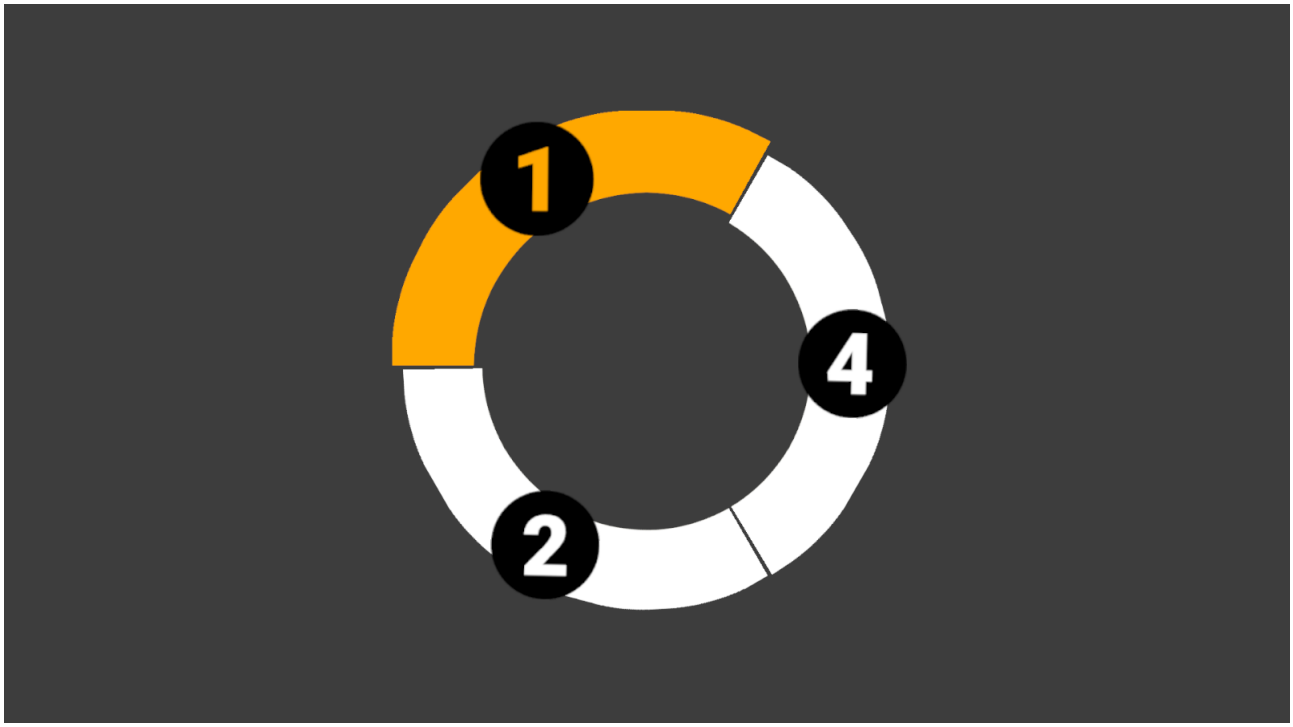
Note: You can use it as soon as the 'OnPieMenuFullyInitialized' event is triggered. Here is an example of how to hide specific Menu Items:

```
using SimplePieMenu;
using System.Collections.Generic;
using UnityEngine;

public class Example : MonoBehaviour
{
    [SerializeField] PieMenu pieMenu;

    private void Awake()
    {
        pieMenu.OnPieMenuFullyInitialized += HideMenuItem;
    }

    private void OnDestroy()
    {
        pieMenu.OnPieMenuFullyInitialized -= HideMenuItem;
    }

    private void HideMenuItem()
    {
        List<int> menuItemsIds = new();
        menuItemsIds.Add(0);
        menuItemsIds.Add(3);

        PieMenuShared.References.MenuItemsManager.MenuItemHider.Hide(pieMenu,
             menuItemsIds);

    }
}
```

It's worth noting that the rotation of your menu will be preserved after this action. If you want to alter the rotation, refer to section 4.17, 'How to Change Rotation at Runtime.'



Pie Menu before hiding Menu Items

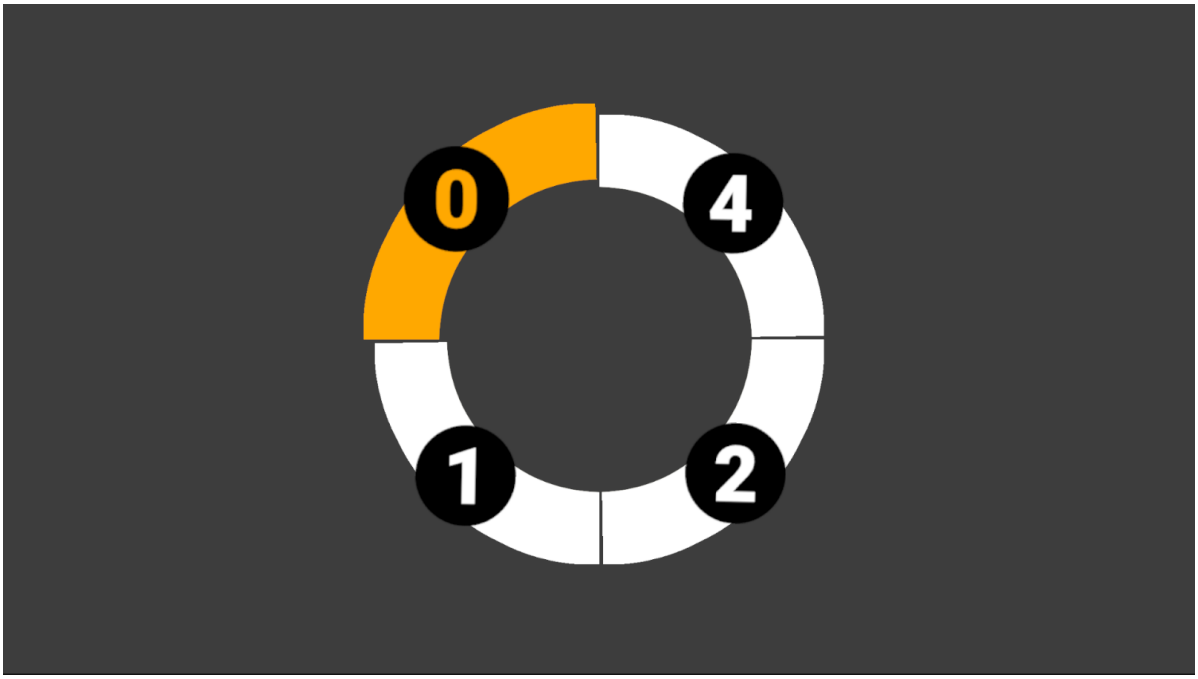


Pie Menu after hiding Menu Items

4.7. How to restore hidden Menu Items

To restore previously hidden Menu Items, use the Restore method, which you can access through the PieMenuShared singleton as shown:

```
using SimplePieMenu;
using System.Collections.Generic;
using UnityEngine;

public class Example : MonoBehaviour
{
    [SerializeField] PieMenu pieMenu;

    public void OnSomeAction()
    {
      // If you wish to restore specific Menu Items, create a list containing their
      // IDs and pass it to the Restore() method.

      // Alternatively, you can call Restore() without providing a list of Ids.
      // In this case, all hidden menu items will be restored.

        List<int> menuItemIds = new();
        menuItemsIds.Add(0);

        PieMenuShared.References.MenuItemsManager.MenuItemHider.Restore(
             pieMenu, menuItemIds);
    }
}
```



Pie Menu after restoring one of Menu Items

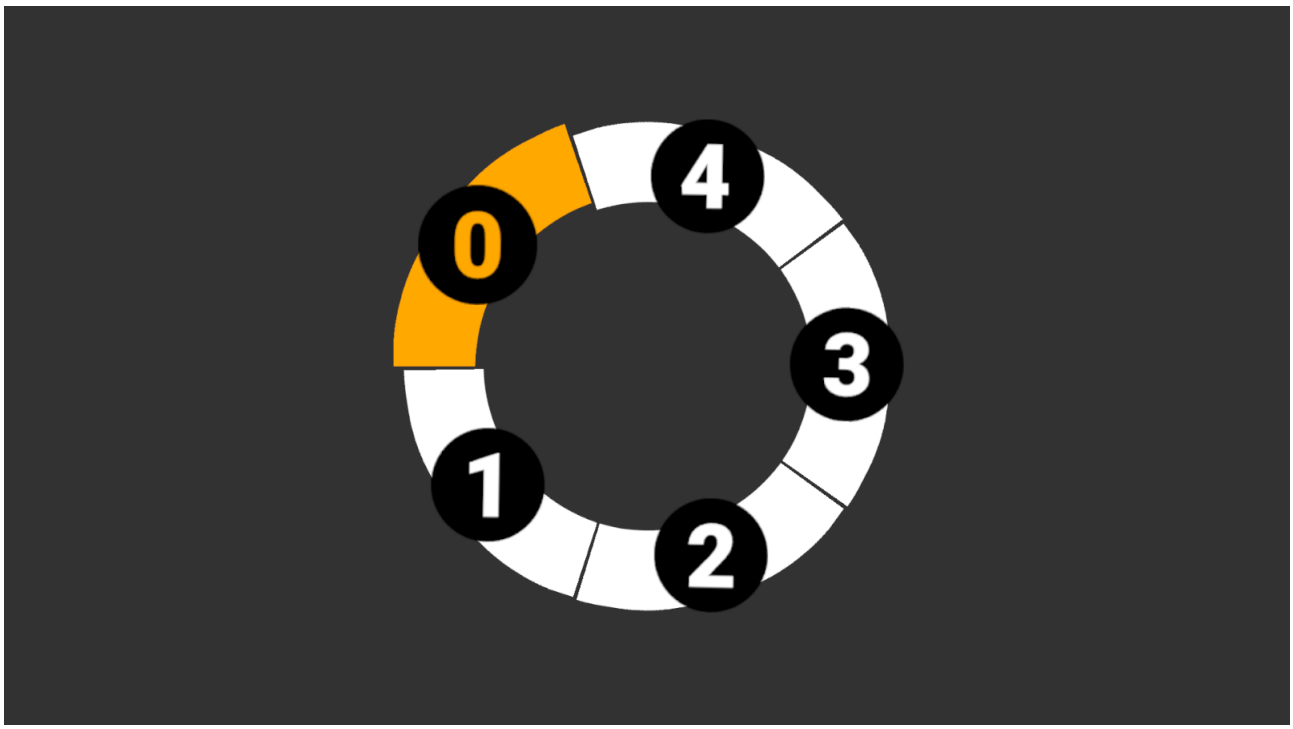
4.8. How to add Menu Item at runtime

To add a new Menu Item, follow these steps:

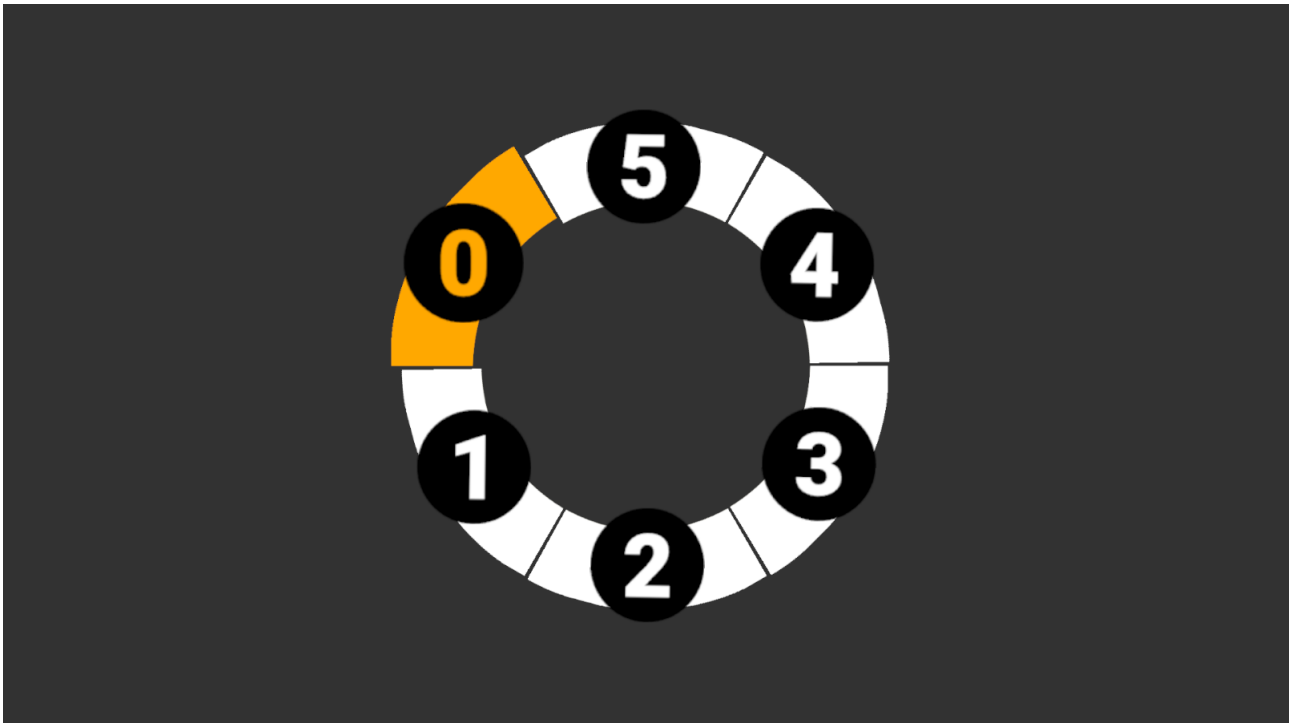
1. Start by creating a prefab based on an existing Menu Item. Ensure that you have already configured the shape, size, and icon settings (if enabled) to your liking. These settings will not be automatically adjusted to match the rest of the menu.
2. Due to the script that highlights the currently selected Menu Item in the hierarchy, your new Menu Item may be saved with incorrect normal color. If that happens, you can manually set the appropriate color or save prefab as follows: Deselect your Menu Item in the hierarchy and then click and drag it into the project window without releasing the left mouse button.
3. Remember to unpack the Menu Item from which you created the prefab, as you'll want to have a new, standalone Menu Item.
4. Now, configure your Menu Item, including adding click handling and setting the icon (if enabled). The rest will be automatically adjusted at runtime.

Here is an example of the code responsible for adding a Menu Item at runtime.

```
using SimplePieMenu;
using System.Collections.Generic;
using UnityEngine;

public class Example : MonoBehaviour
{
    [SerializeField] PieMenu pieMenu;

    [SerializeField] List<GameObject> menuItems;

    private void AddMenuItem()
    {
        MenuItemAdder adder = PieMenuShared.References.MenuItemsManager.MenuItemAdder;
        adder.Add(pieMenu, menuItems);
    }
}
```



Pie Menu before adding new Menu Item

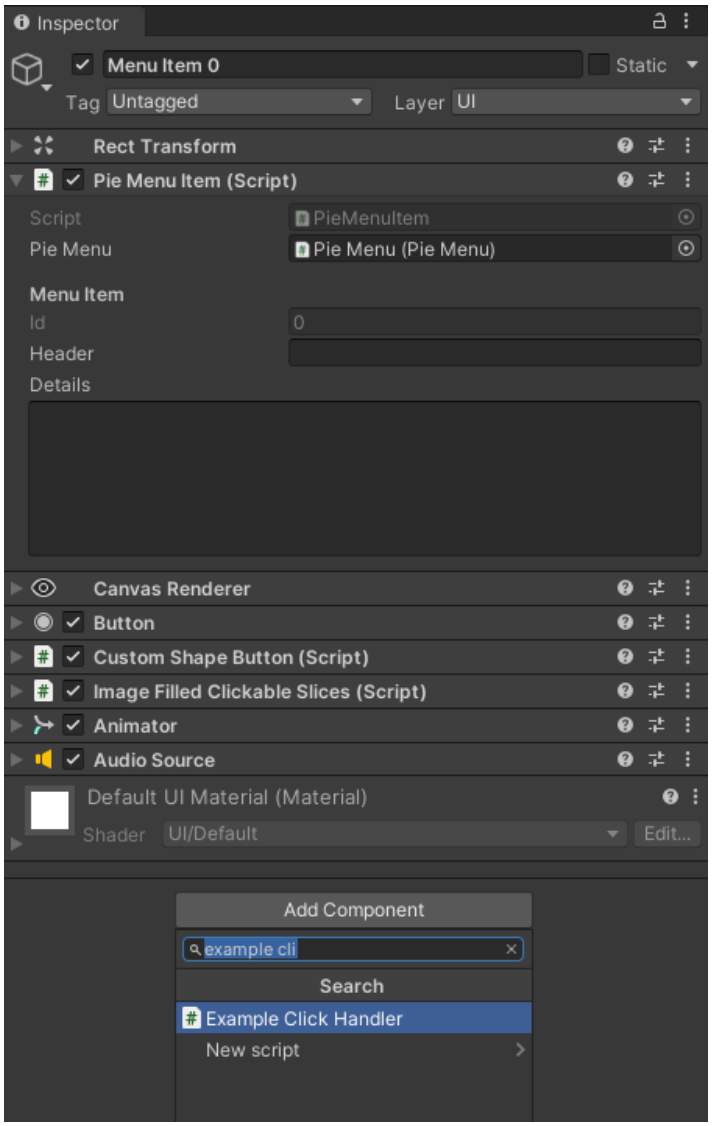


Pie Menu after adding new Menu Item

4.9. How to handle clicks on specific Menu Item

To handle clicks, create a new script deriving from MonoBehaviour and implementing the IMenuItemClickHandler interface. Then, attach the script to the relevant Menu Item in the Hierarchy under Pie Menu - Canvas/Pie Menu/Menu Items/Visible Menu Items. Example script:

```
using SimplePieMenu;
using UnityEngine;

public class ExampleClickHandler : MonoBehaviour, IMenuItemClickHandler
{
    public void Handle()
    {
        Debug.Log($"You have clicked {transform.name}");
    }
}
```

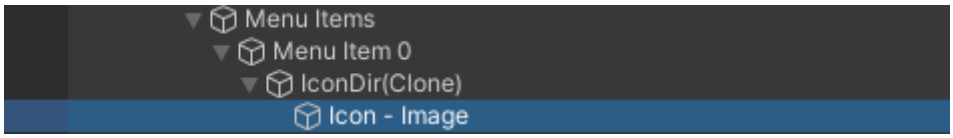


Adding Click Handler to Menu Item

4.10. How to change icons

4.10.1. In editor

If icons are enabled, locate the specific Menu Item and find the 'Icon - Image' object in its hierarchy. Then, simply replace the source image with the icon you want to display.



Icon - Image object in Hierarchy

4.10.2. At runtime

To do this, you need access to the IconGetter script. Then, call the ChangeIcon() method. Here's an example.

```
using SimplePieMenu;
using UnityEngine;

public class Example : MonoBehaviour
{
    [SerializeField] PieMenu pieMenu;
    [SerializeField] Sprite icon;

    private void OnSomeAcition()
    {
        int menuItemId = 3;
        Transform menuItem =
             pieMenu.MenuItemsTracker.GetMenuItem(menuItemId).transform;

        IconGetter iconGetter =
             PieMenuShared.References.IconsSettingsHandler.IconGetter;

        iconGetter.ChangeIcon(menuItem, icon);

    }
}
```

4.11. How to add a new Input Device

4.11.1. Prerequisite

To add a new Input Device, open the PieMenuInputSettings script. You will find an enum called 'AvailableInputDevices' there. Simply add the name of your device to it. In this example, it will be VRController.

```
public enum AvailableInputDevices
{
    MouseAndKeyboard_OLD_INPUT_SYSTEM = 0,
    MouseAndKeyboard_NEW_INPUT_SYSTEM = 1,
    VRController = 2
}
```

4.11.2. Old Input System

Create a new script in which you will derive from MonoBehaviour and implement the IInputDevice interface.

```
using SimplePieMenu;
using UnityEngine;

public class VRController_OLD_INPUT_SYSTEM : MonoBehaviour, IInputDevice
{
    public Vector2 GetPosition(Vector2 anchoredPosition)
    {
        throw new System.NotImplementedException();
    }

    public bool IsSelectionButtonPressed()
    {
        throw new System.NotImplementedException();
    }

    public bool IsCloseButtonPressed()
    {
        throw new System.NotImplementedException();
    }
}
```

As you can see, you need to provide the implementation for three methods:

- **GetPosition()**: This method returns the cursor's position on the screen, taking into account the Pie Menu anchored position. Please use the AnchoredPosition property stored in PieMenuInfo, as it is properly adjusted for screen size, and it will help you avoid errors in the Unity editor (access it with pieMenu.PieMenuInfo.AnchoredPosition).

- **IsSelectionButtonPressed()**: It checks if the user pressed the button responsible for selecting a Menu Item in your Pie Menu.

- **IsCloseButtonPressed()**: This method checks if the button responsible for closing the Pie Menu was pressed. You don't have to implement this method if you won't allow users to close the Pie Menu without selecting any options.

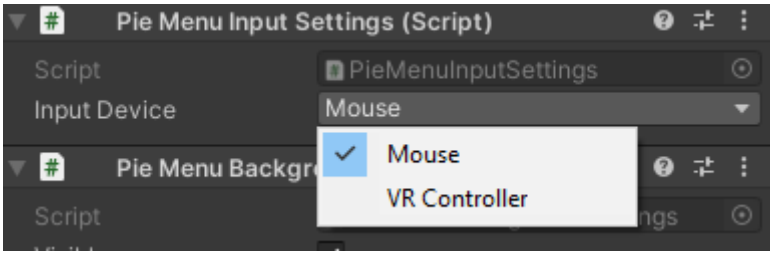
To make it easier, take a look at the class responsible for mouse handling.

```
public class MouseAndKeyboard_OLD_INPUT_SYSTEM : MonoBehaviour, IInputDevice
{
    public Vector2 GetPosition(Vector2 anchoredPosition)
    {
        Vector2 mouseInput;

        mouseInput.x = Input.mousePosition.x - (Screen.width / 2f) -
            anchoredPosition.x;
        mouseInput.y = Input.mousePosition.y - (Screen.height / 2f) -
            anchoredPosition.y;

        return mouseInput;
    }

    public bool IsButtonClicked()
    {
        return Input.GetKeyDown(KeyCode.Mouse0);
    }

    public bool IsReturnButtonPressed()
    {
        return Input.GetKeyDown(KeyCode.Escape);
    }
}
```

The final step is to open the InputDeviceGetter script and inside HandleInputDevicePreferences() add a new option in the 'if()' statement. In this example, it will be:

```
if (isOldInputSystemEnabled)
{
    if (inputDeviceId == (int)AvailableInputDevices.MouseAndKeyboard_OLD_INPUT_SYSTEM)
        SetInputDevice<MouseAndKeyboard_OLD_INPUT_SYSTEM>();
    else if (inputDeviceId == (int)AvailableInputDevices.VRController)
        SetInputDevice<VRController_OLD_INPUT_SYSTEM>();
    else SetDefault();
}
```

Thats all. After refreshing the scene, you will see the new Input Device available for selection in the list located in the PieMenuInputSettings script.



Pie Menu Input Settings script

4.12.3. New Input System

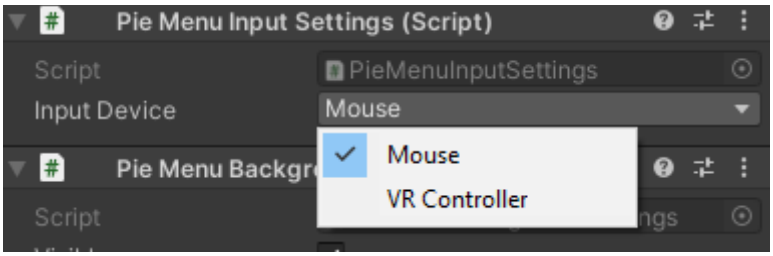
You need to open the Input Action Importer named 'PieMenuControls' and add new bindings to respective actions. If you are unsure about the process, refer to the official documentation.

Next, you need to create a new script in which you will derive from MonoBehaviour and implement the IInputDevice interface. Practically the entire 4.12.2. Old Input System section focuses on adding this script, so if you encounter any issues, simply refer to that part of the documentation. However, it seems that you could potentially use the script for mouse control, 'MouseAndKeyboard_NEW_INPUT_DEVICE,' with slight modifications in the 'GetPosition()' method. This is because the New Input System has built-in functionality for reading mouse screen positions. For other devices, it appears that you might need to add a virtual cursor for screen navigation. If you're interested in trying it yourself, I recommend checking out this tutorial.

The last step is to open the 'InputDeviceGetter' script and add a new option inside the 'HandleInputSystemPreferences()' method within the 'if()' statement.

```
else if (isNewInputSystemEnabled)
{
    if (inputDeviceId == (int)AvailableInputDevices.MouseAndKeyboard_NEW_INPUT_SYSTEM)
        SetInputDevice<MouseAndKeyboard_NEW_INPUT_SYSTEM>();
    else if (inputDeviceId == (int)AvailableInputDevices.VRController)
        SetInputDevice<VRController_NEW_INPUT_SYSTEM>();
    else SetDefault();
}
```

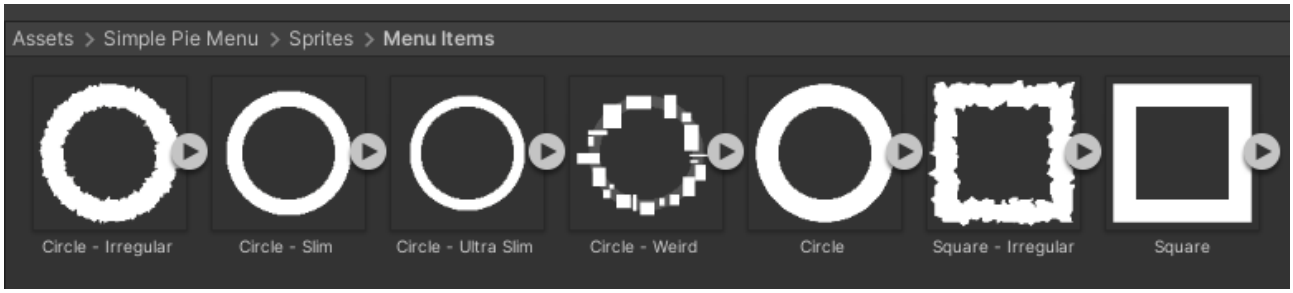
Thats all. After refreshing the scene, you will see the new Input Device available for selection in the list located in the PieMenuInputSettings script.



Pie Menu Input Settings script

4.12. How to add more shapes

First and foremost, you need to create an sprite that represents a single Menu Item as if it were the entire Pie Menu itself. Examples of such images can be found in the '.../Simple Pie Menu/Sprites/Menu Items' directory. The remaining adjustments will be handled by the appropriate scripts, which will automatically generate Menu Item sizes based on an image you provided.



Menu Item shapes

After creating your sprite and importing it into Unity, you need to:

1. Change its **Texture Type** to **Sprite (2D and UI)**.
2. Set the **Mesh Type** to **Full Rect**.
3. **Check** the **Read/Write** option.
4. **Check** the **Generate Mip Maps** option (this is optional, it enables subtle anti-aliasing).

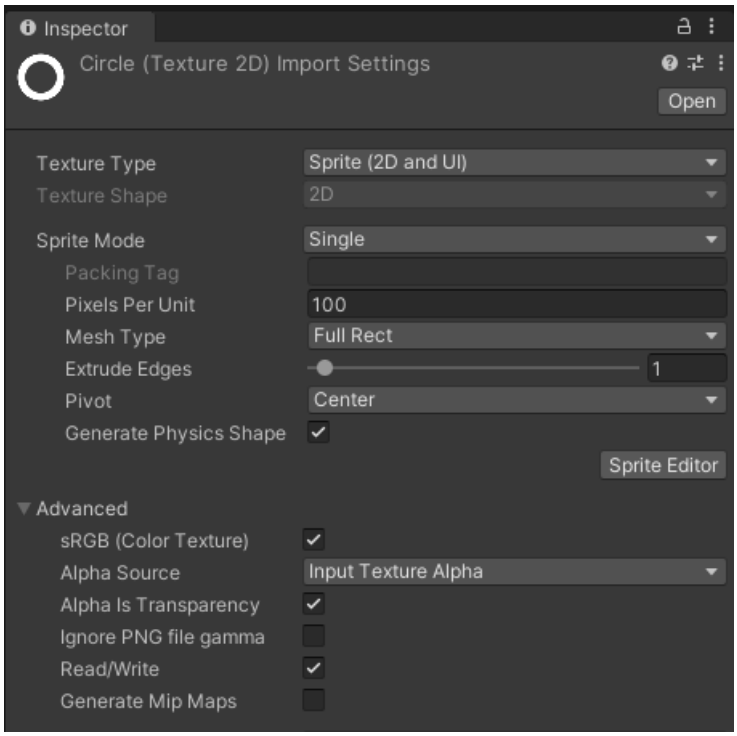
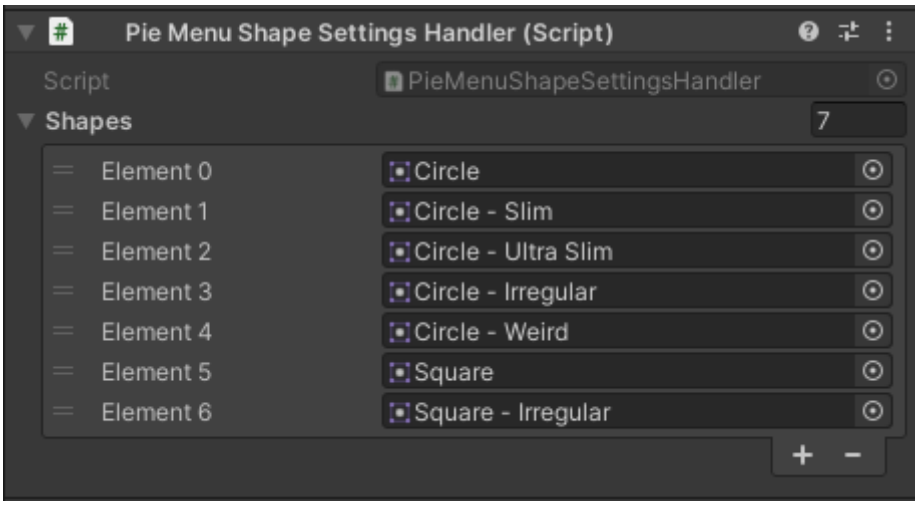


Image settings

To add a shape to the list that is available when creating a new menu:

1. Find the PieMenuShared prefab located at '.../Simple Pie Menu/Prefabs/PieMenuShared.prefab.'
2. Locate the script 'PieMenuShapeSettingsHandler' within it.
3. Add a new image to the 'Shapes' list.



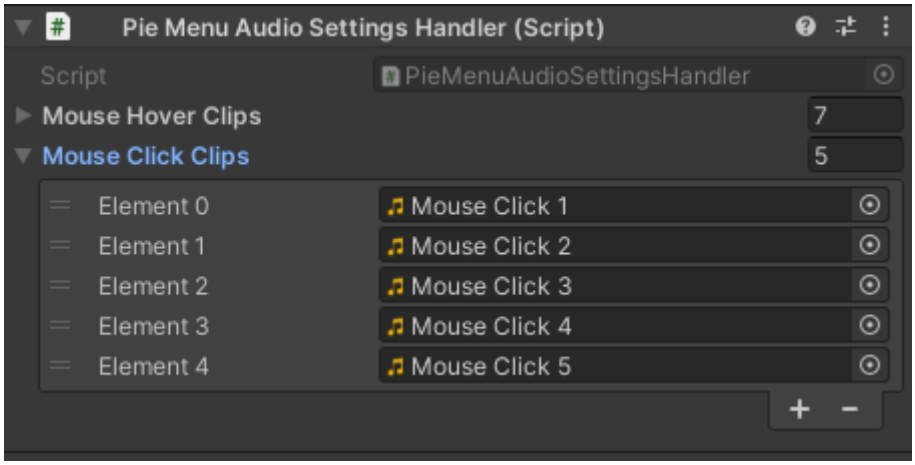
PieMenuShapeSettingsHandler script

That's it! After refreshing the scene, you should see the new shape as one of the available options.

4.13. How to add more sounds

To add new sounds:

1. Find the PieMenuShared prefab located at '.../Simple Pie Menu/Prefabs/PieMenuShared.prefab.'
2. Locate the script 'PieMenuAudioSettingsHandler' within it.
3. Add a new audio clip to the appropriate list.



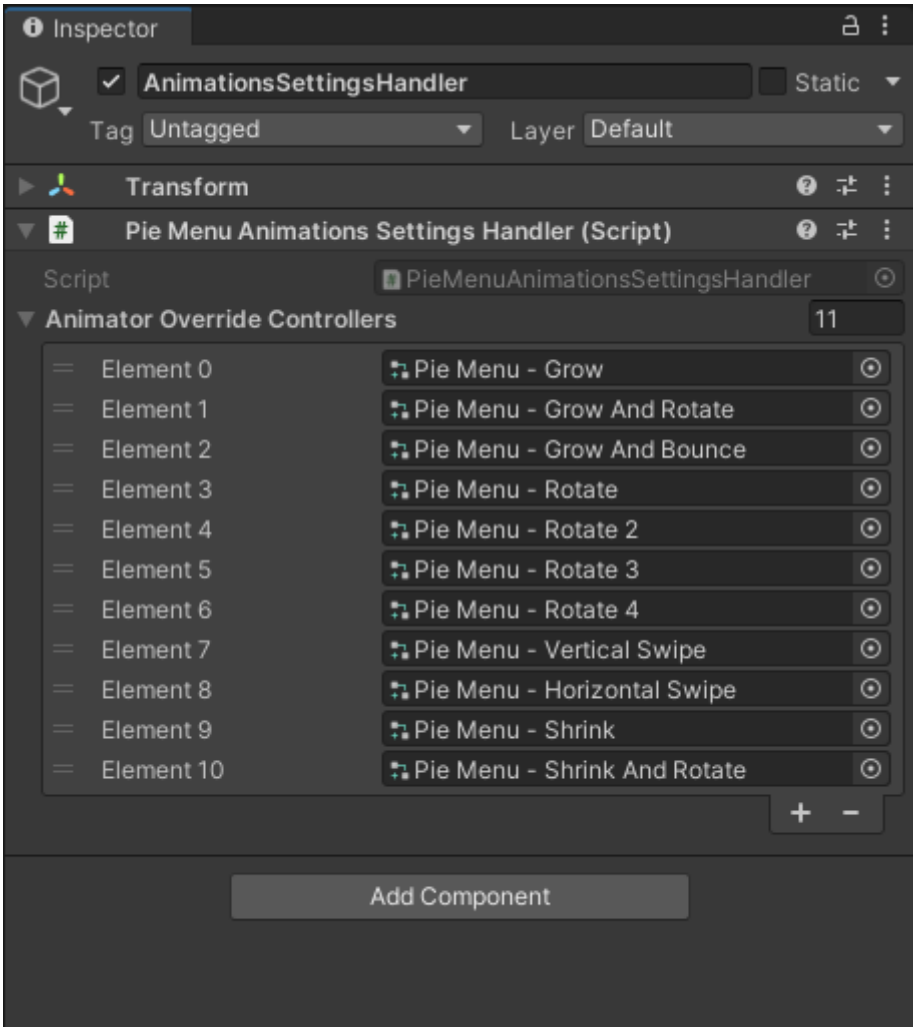
PieMenuAudioSettingsHandler

After refreshing the scene, you should see a new audio clip as one of the available options inside PieMenuAudioSettings Inspector.

4.14. How to add more animations

To add new animation:

1. Create an Animation Override Controller for the existing animator component. The easiest way to do this is to navigate to the '.../Simple Pie Menu/Animations/Pie Menu', duplicate one of the existing controllers, rename it and replace the animation with your own.
2. Find the PieMenuShared prefab located at '.../Simple Pie Menu/Prefabs/PieMenuShared.prefab'.
3. Locate the script 'PieMenuAnimationsSettingsHandler' within it.
4. Add an Animation Override Controller that you created to the list.



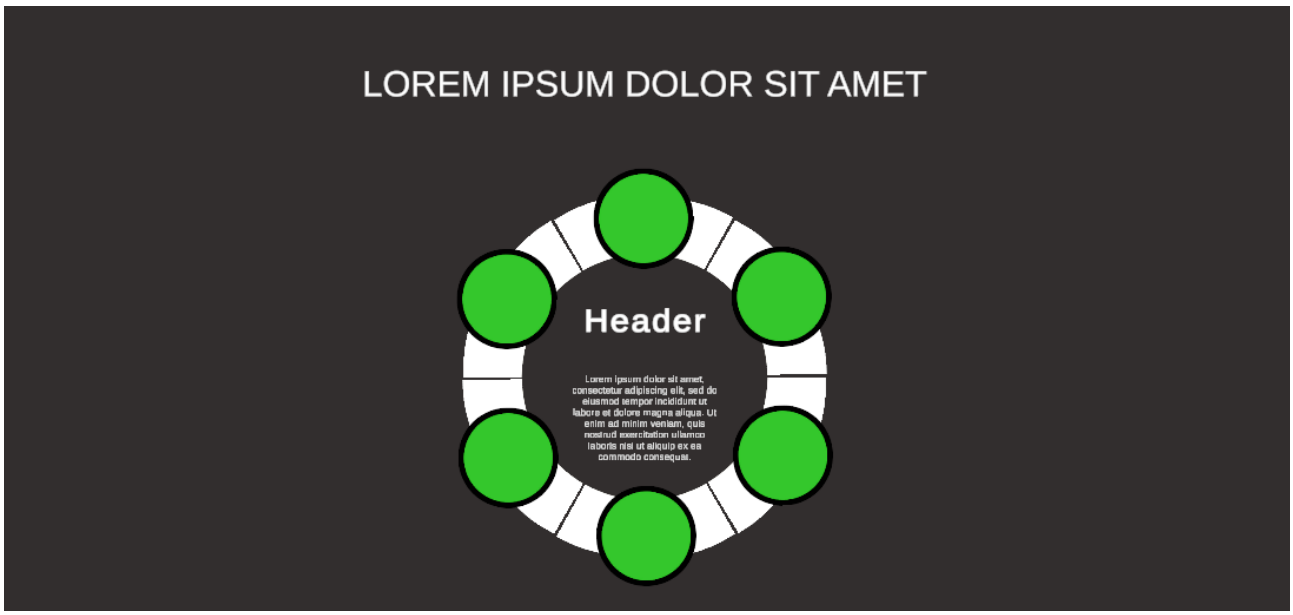
PieMenuAnimationsSettingsHandler script

After completing these steps refresh the scene, and new animation will be available in the lists inside Pie Menu Animations Settings Inspector.

4.15. How to add a header to your Pie Menu

If you want to add a header to your menu that will inform users about the interaction they are selecting and that appears and disappears at the right moment:

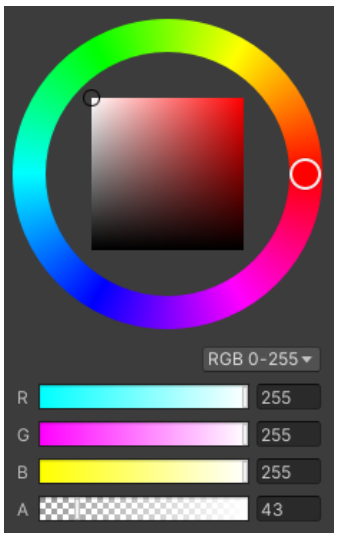
1. Make sure that the Info Panel is enabled in the PieMenuInfoPanelSettings Inspector.
2. Locate the Info Panel in your menu hierarchy and add a new component to it, 'Text - TextMeshPro'.
3. Configure this component by setting its content, display options, and position.



Pie Menu with header

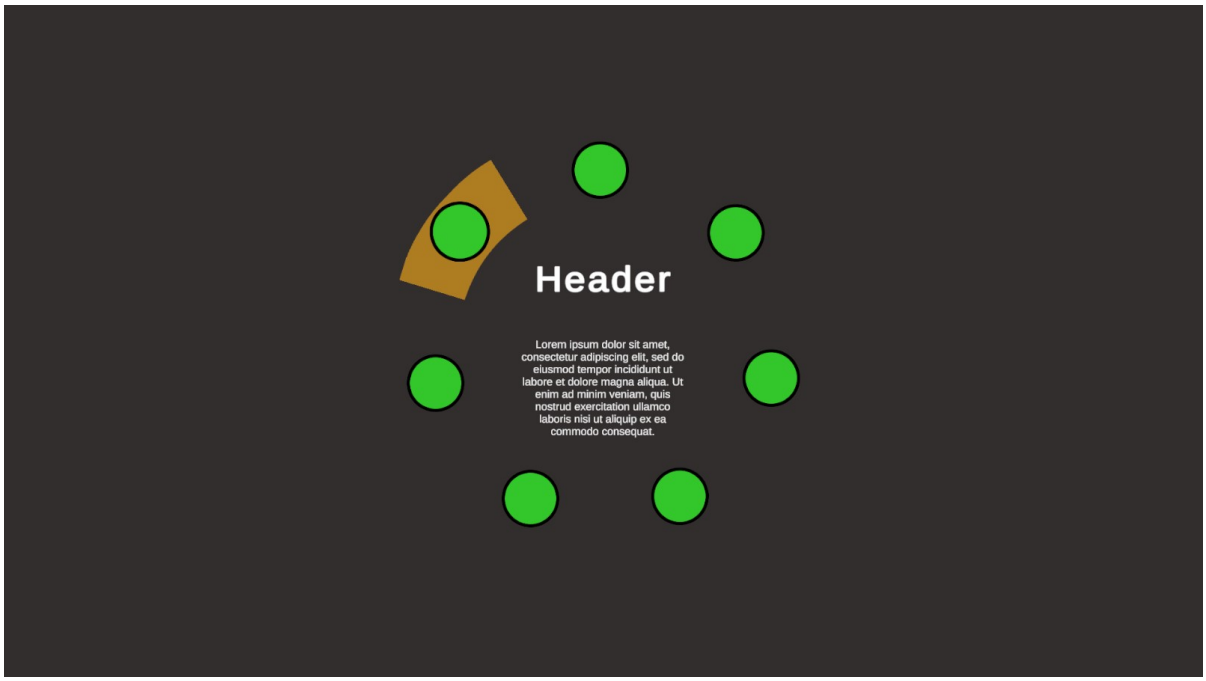
4.16. How to create a menu that contains only icons

To create a menu that includes only icons, adjust the color transparency (indicated by the letter "A" in the RGBA) in the MenuItemColorSettings Inspector to a value of 0.



Color picker

You can experiment with these settings and, for example, set the transparency of the selected color to 100 to achieve the effect shown in the illustration below.



Pie Menu with icons only

4.17. How to change rotation at runtime

After using the Hide, Add, Restore Menu Items functionality, you may want to adjust the rotation of your menu, as the rotation remains unchanged after these operations.

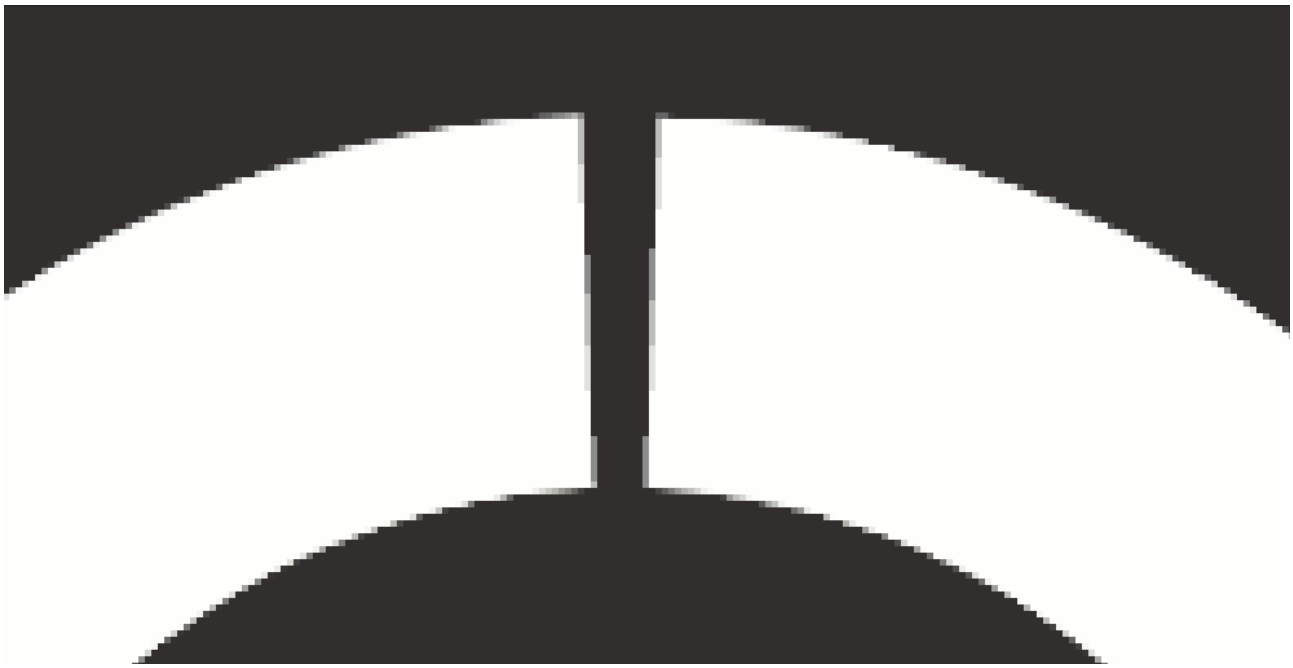
```
using SimplePieMenu;
using UnityEngine;

public class Example : MonoBehaviour
{
    private PieMenu pieMenu;

    private void ChangeRotation()
    {
        var generalSettingsHandler = PieMenuShared.References.GeneralSettingsHandler;
        int newRotation = 25;

        // for symmetrical positioning of Menu Items, use the following line:
        //newRotation =
RotationCalculator.CalculateNewRotation(pieMenu.MenuItemsTracker.PieMenuItems.Count,
pieMenu.PieMenuInfo.MenuItemSpacing);

        // reset the rotation
        generalSettingsHandler.HandleRotationChange(pieMenu, 0);
        // set the new rotation
        generalSettingsHandler.HandleRotationChange(pieMenu, newRotation);
    }
}
```

4.18. Anti-aliasing

4.18.1. Canvas render mode

In your Pie Menu, you may sometimes notice jagged edges on the Menu Items. By default, the render mode of the canvas object containing the Pie Menu is set to 'Screen Space - Overlay.' This rendering mode positions UI elements on the screen, overlaying the scene, but it has the MSAA (Multisample Anti-Aliasing) option disabled [learn more]. You can switch this to 'Screen Space - Camera,' which should enhance the appearance of your menu. However, keep in mind that this rendering mode does not display UI elements on top, so check if it's suitable for your project. For more information read the documentation.

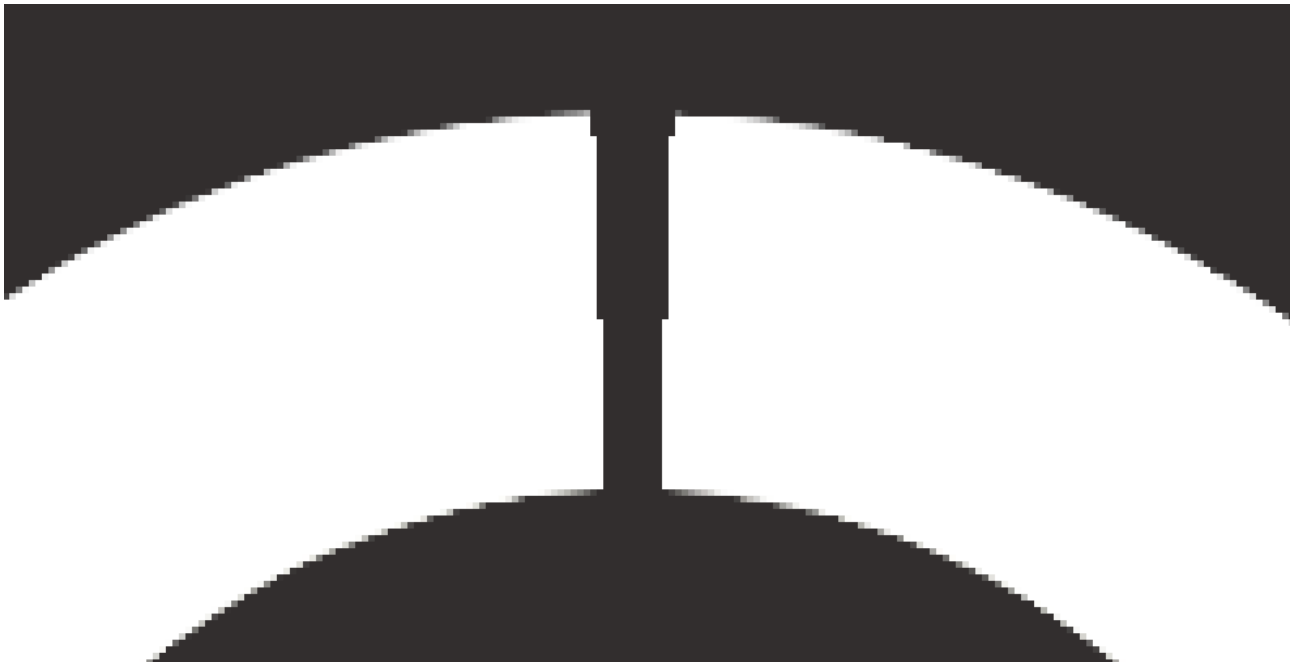
Render Mode: Screen Space - Overlay



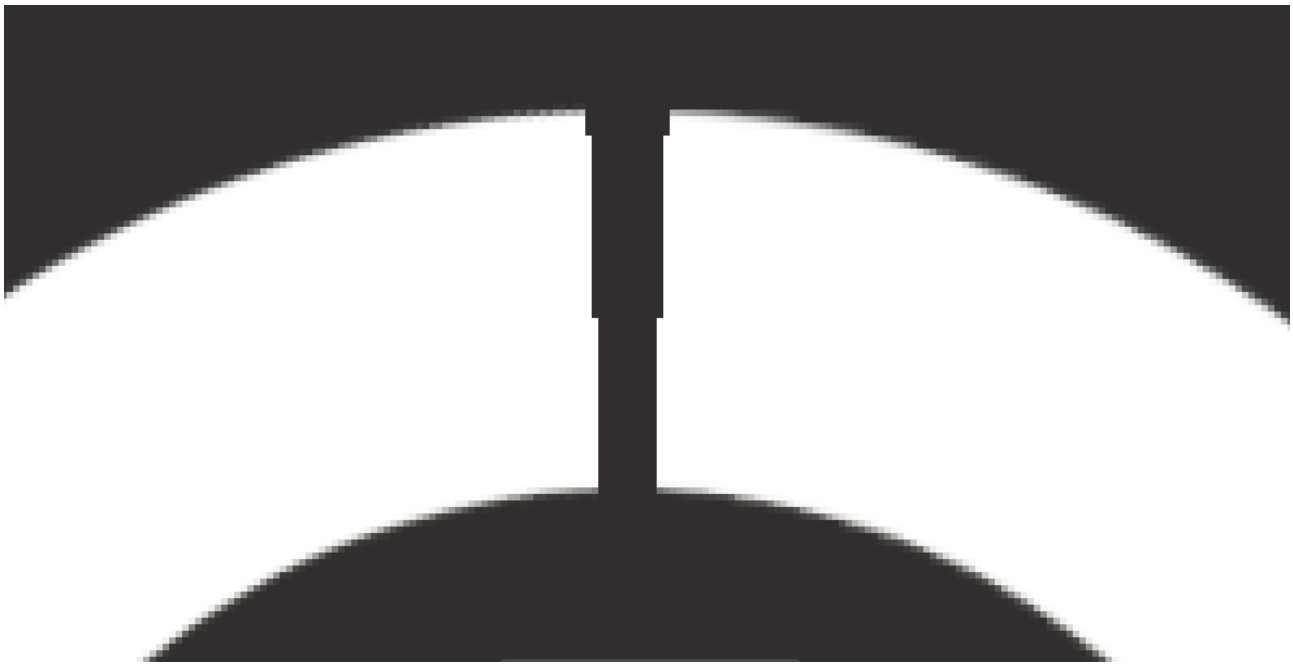
Render Mode: Screen Space - Camera

4.18.2. Mipmaps

Mipmapping is a technique in computer graphics where precomputed, lower-resolution versions of textures (called mipmaps) are used to improve rendering efficiency and reduce aliasing artifacts. Enabling mipmapping helps smooth out edges in graphics.

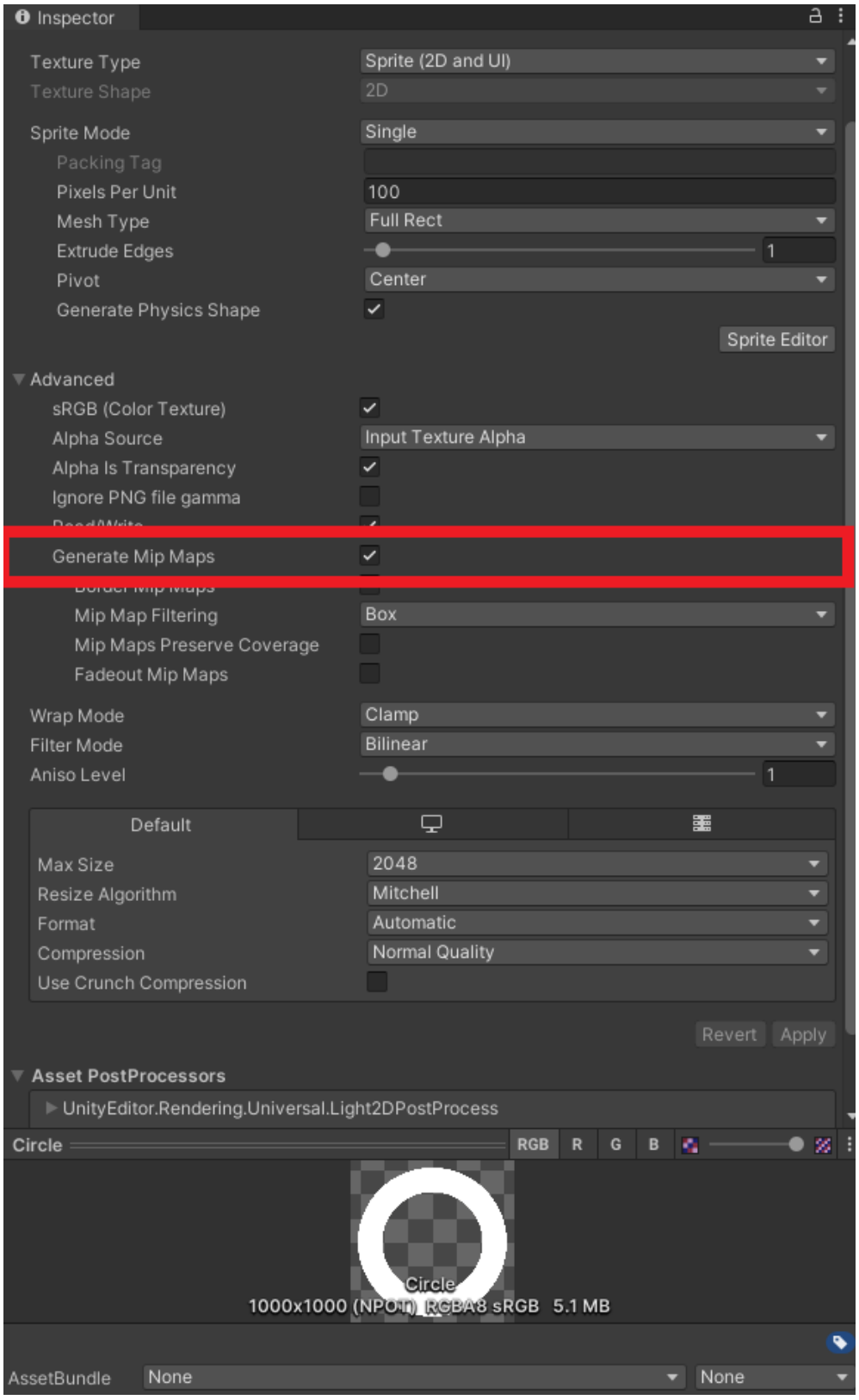


Mipmaps disabled



Mipmaps enabled

To enable mipmapping for the Pie Menu sprite, go to the directory '.../Simple Pie Menu/Sprites/Menu Items', select the specific sprite you're working with, and check the 'Generate Mipmaps' option.



Generate Mipmaps option

4.19. Key Mapping Configuration

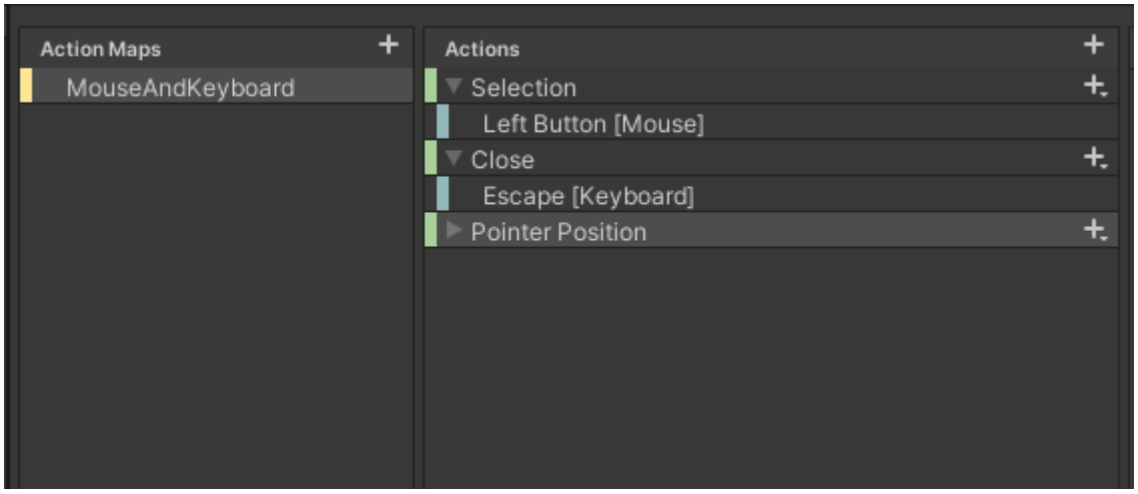
4.19.1. Old Input System

To change the key assignments for the Old Input System, open the relevant input device script. By default, this is MouseAndKeyboard_OLD_INPUT_SYSTEM. Modify the keycode values in the two methods – IsSelectionButtonPressed and IsCloseButtonPressed to the keys you prefer:

```
public bool IsSelectionButtonPressed()
{
    return Input.GetKeyDown(KeyCode.Mouse0);
}

public bool IsCloseButtonPressed()
{
    return Input.GetKeyDown(KeyCode.Escape);
}
```

4.19.2. New Input System

To change the key assignments for the New Input System, open the 'Pie Menu Controls', which serves as the Input Action Importer. Add new bindings or modify the bindings of individual actions to the keys that suit your preferences.



PieMenuControls

4.20. Precision of Selection

If you swiftly move the cursor around the Menu Items, you might observe that their selection is not entirely accurate. This is because, for optimization of resource usage, I have limited the calculation speed of the current selection in the MenuItemSelectionHandler script using the HandleSelection coroutine, which computes the correct selection 10 times per second. If you wish to increase the calculation speed, decrease the value of the coroutineDelay field.

5. Summary

5.1. Best practices

5.1.1. Performance Optimization

It is recommended not to leave the menu as active in your scene while actively working on your projects. This is because its scripts are executed every time an 'AssemblyReload' occurs, which is a Unity process that reloads all scripts, typically when you recompile your code or make changes to your project. Leaving the menu permanently in your scene can result in a minor performance overhead.

5.1.2. Prefab Search Optimization for 'PieMenuContextMenuExtension'

The 'PieMenuContextMenuExtension' script, responsible for creating the new Pie Menu as shown in the 'Getting Started' section, searches for the Pie Menu - Canvas prefab in the project files. Therefore, in the case of larger projects, it's a good practice to provide the correct path to the prefab within the project directory directly within this script to optimize the search process and avoid unnecessary delays.

5.2. Discord server

To enhance this asset, I have created a Discord server where every user can report issues or suggest new features.

I've set up a special channel dedicated to answering your questions, so be sure to check it out; you might find what you're looking for right there.

This server is also a place where you can share materials from projects that utilize the Simple Pie Menu asset. I can't wait to see them!

Join Discord!

5.3. Resources

The icons used in the Menu Items within this documentation are sourced from the website https://www.iconpacks.net/.

5.4. Acknowledgments

Thank you for using the Simple Pie Menu asset! I hope you find it valuable for your Unity projects. Your feedback is essential, so please take a moment to leave a review.