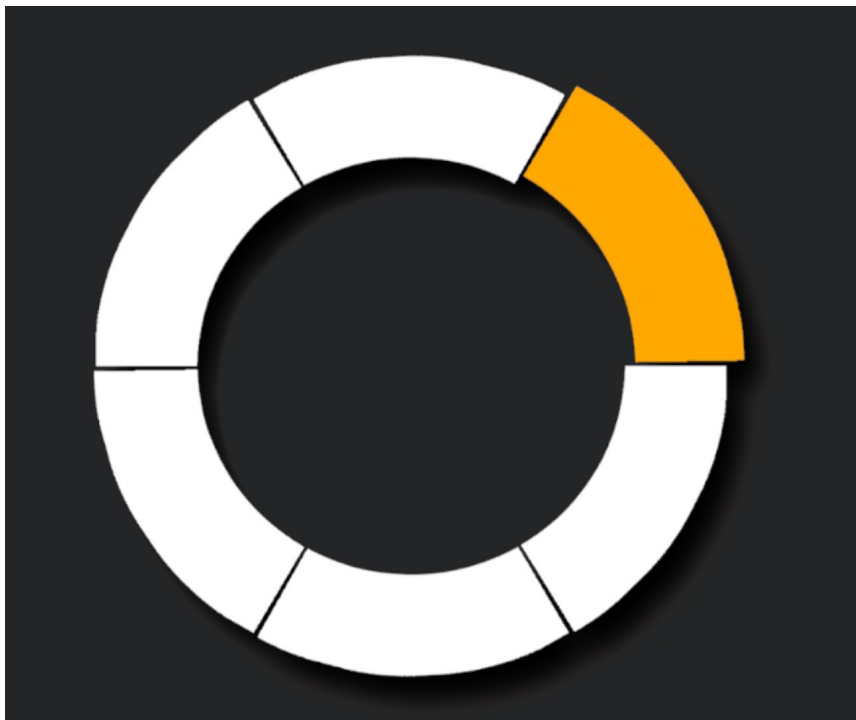


# Simple Pie Menu

Documentation







# Table of contents

Introduction.....	1
1. Prerequisites.....	2
1.1. Text Mesh Pro.....	2
1.2. Input System.....	2
1.2.1. Automatic installation.....	2
1.2.2. Manual installation.....	2
2. Creating the Pie Menu.....	3
2.1. Getting started.....	3
2.2. Customization basics.....	3
3. Initial Menu Setup.....	4
3.1. Selecting a Menu Item in the Hierarchy.....	4
3.2. Changing Header and Details Text.....	4
3.3. Changing Icons.....	4
3.4. Handling Clicks on a Menu Item.....	4
3.5. Showing and Hiding the Pie Menu.....	5
3.6. Anti-Aliasing.....	6
3.6.1. Canvas Render Mode.....	6
3.6.2. Mipmaps.....	8
4. Runtime Configuration.....	10
4.1. Modifying Specific Menu Items.....	10
4.2. Changing Header and Details at Runtime.....	11
4.3. Changing Icons at Runtime.....	11
4.4. Changing Rotation at Runtime.....	12
5. Managing Menu Items Count at Runtime.....	13
5.1. Important Information.....	13
5.1. Disabling Menu Items.....	14
5.2. Hidding Menu Items.....	15
5.3. Restoring Hidden Menu Items.....	17
5.4. Adding a Menu Item.....	18
6. Advanced Pie Menu Customization.....	20
6.1. How to Add More Shapes.....	20
6.2. How to Add More Sounds.....	22
6.3. How to Add More Animations.....	23
6.4. How to Add a Header to Your Pie Menu.....	24
6.5. How to Create a Menu with Icons Only.....	24
6.6. Key Mapping Configuration.....	26
6.6.1. Old Input System.....	26
6.6.2. New Input System.....	26
6.7. How to add a new Input Device.....	27
7. Project Structure.....	30
7.1. Scripts with the Settings suffix.....	30
7.2. Scripts with the SettingsHandler suffix.....	30
7.3. Singleton Pattern.....	30
7.4. PieMenu Script.....	31
8. Detailed Explanation of Available Configuration Options.....	32
8.1. Input settings.....	32
8.2. Background Settings.....	32
8.3. Shape Settings.....	33
8.4. General Settings.....	33
8.5. Position Settings.....	34

8.6. Menu Item Color Settings.....	34
8.7. Info Panel Settings.....	35
8.8. Icons Settings.....	35
8.9. Animations Settings.....	36
8.10. Audio Settings.....	36
8.11. Selection Settings.....	37
8.12. Close Functionality Settings.....	37
8.13. Preview Mode Settings.....	37
9. Best Practices & Optimization.....	38
9.1. Performance Optimization.....	38
9.2. Prefab Search Optimization.....	38
9.3. Precision of Selection.....	38
10. Resources and Feedback.....	39
10.1. Discord Server.....	39
10.2. Resources.....	39
10.3. Acknowledgments.....	39

#### Legend:

-  Basics.
-  Read if you intend to modify your menu at runtime.
-  Here you will learn how to add custom options tailored to your specific needs.
-  Boring.

# Introduction

This asset allows you to quickly create and customize Pie Menus in your Unity projects.

The documentation provides a clear overview of how to set up, use, and integrate the tool. Whether you are an experienced developer or a beginner, you will find step-by-step instructions and practical examples to help you make the most of this asset.

# 1. Prerequisites

## 1.1. Text Mesh Pro

Before using the asset, you need to import **TMP Essential Resources**.

Follow these steps:

1. Open Unity.
2. In the top menu bar, go to **Window**.
3. Select **TextMeshPro** from the drop-down menu.
4. Click **Import TMP Essential Resources**.

[\[More details\]](#)

## 1.2. Input System

This asset supports both the **Old Input System** and the **New Input System**. To ensure proper import and avoid errors, it is recommended to install the **Input System** package, even if you plan to use only the old system.

### 1.2.1. Automatic installation

When importing the asset, Unity will prompt you to install the required dependencies. Select **Yes** to install the Input System automatically.

### 1.2.2. Manual installation

To install the Input System manually:

1. Open the **Package Manager** by selecting **Window > Package Manager** from the top menu.
2. In the Package Manager, switch the view to **Packages: Unity Registry**.
3. Search for **Input System** using the search bar.
4. Select the package and click **Install**.

[\[More details\]](#)

## 2. Creating the Pie Menu

### 2.1. Getting started

For safety, it is recommended to create the Pie Menu in a **separate scene**. The scripts included in this asset run in **Edit Mode** and may remove objects from the Hierarchy. Although this should not cause issues, working in a separate scene minimizes potential risks.

To create a Pie Menu:

1. In the **Hierarchy** panel, right-click.
2. Select **UI > Simple Pie Menu**.

### 2.2. Customization basics

Locate the **Pie Menu** object in the **Hierarchy**.  
It can be found at: **Pie Menu - Canvas > Pie Menu**.

With the object selected, the **Inspector** window will display all customization options. Most of these settings are intuitive, so you can start experimenting right away. A detailed description of every available option is provided in section **8. Detailed Explanation of Available Configuration Options**.

## 3. Initial Menu Setup

### 3.1. Selecting a Menu Item in the Hierarchy

When you select a **Menu Item** in the **Hierarchy**, it will be visually highlighted in the **Scene** view. The highlight indicates which Menu Item is currently being modified. The color of the highlight corresponds to the **Selected Color** defined in the **Menu Item Color Settings**.

### 3.2. Changing Header and Details Text

To edit the **Header** and **Details** text of a Menu Item:

1. In the **Hierarchy**, select the specific **Menu Item**.
2. In the **Inspector**, locate the **PieMenuItem** script.
3. Fill in the two **[SerializeField]** fields:
  - **Header Text**
  - **Details Text**

### 3.3. Changing Icons

If icons are enabled:

1. Select the specific **Menu Item** in the **Hierarchy**.
2. Find the **Icon - Image** object in its hierarchy.
3. Replace the source image with your desired icon.

### 3.4. Handling Clicks on a Menu Item

To handle clicks on a Menu Item:

1. Create a new script that derives from **MonoBehaviour** and implements the **IMenuItemClickHandler** interface.
2. Attach the script to the relevant **Menu Item**.

Example:

```
using SimplePieMenu;
using UnityEngine;

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

## 3.5. Showing and Hiding the Pie Menu

To dynamically display the Pie Menu in your scene:

1. In the **Hierarchy**, set the **Pie Menu Canvas** of your menu to **inactive**.
2. Attach the **PieMenuDisplayer** script to a GameObject in your scene.
3. Create a new script (you can copy the example script below) and attach it to the same GameObject as the **PieMenuDisplayer**.
4. In the **Inspector**, assign your Pie Menu to the `pieMenu` field of the new script.

Note: The example below uses the **Old Input System**. If you would like to achieve the same result with the **New Input System**, the setup steps remain almost identical — the only difference is the way you handle input (checking button presses).

Example:

```
using UnityEngine;
using SimplePieMenu;

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

    private PieMenuDisplayer displayer;

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

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

What this script does:

- Gets a reference to the **PieMenu** component.
- Gets a reference to the **PieMenuDisplayer**.
- Calls the **ShowPieMenu()** method of the **PieMenuDisplayer**, passing your Pie Menu as a parameter.



Additional information:

- Two example scenes are included in the **Demos** folder, showing how to display the Pie Menu using both the Old Input System and the New Input System.
- The menu automatically hides after a Menu Item is selected or when the menu is closed (see **8.12 Close Functionality Settings**).
- If the Pie Menu remains active in the scene, it will automatically play its animation in **Play Mode** and become immediately interactive.

## 3.6. Anti-Aliasing

### 3.6.1. Canvas Render Mode

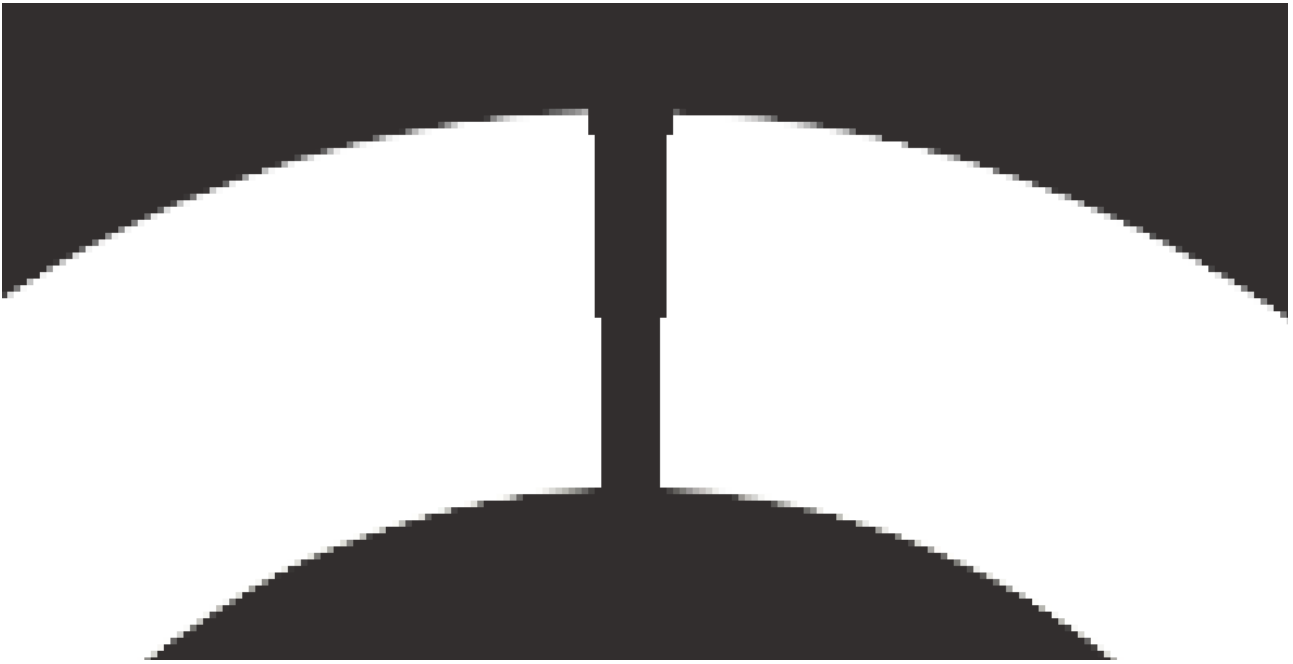
The **render mode of the Canvas** has a significant impact on the appearance of the Pie Menu. It is highly recommended to check whether your project allows you to use **Screen Space - Camera**, as this can greatly improve the visual quality.

By default, the Canvas containing the Pie Menu is set to **Screen Space – Overlay**. This mode positions UI elements on top of the scene but has **MSAA (Multisample Anti-Aliasing)** disabled.

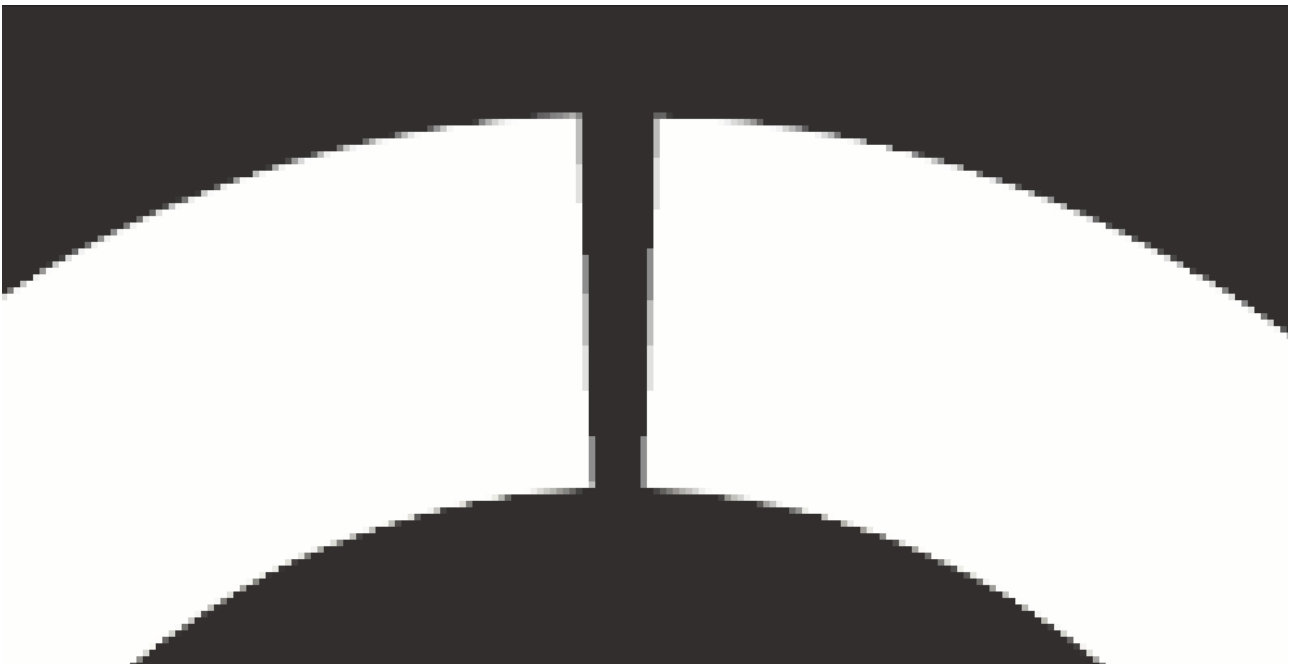
Switching to **Screen Space - Camera** can enhance the look of the Pie Menu by reducing jagged edges. However, keep in mind that this mode does not render UI elements on top of everything, so make sure it fits your project requirements. For more details, see the official Unity [documentation](#).

To switch the Canvas to **Screen Space - Camera**:

1. In the **Hierarchy**, select the **Canvas** that contains the Pie Menu.
2. In the **Inspector**, under the **Canvas** component, set **Render Mode** to **Screen Space - Camera**.
3. Assign a **Camera** to the **Render Camera** field.



*Render Mode: Screen Space - Overlay*



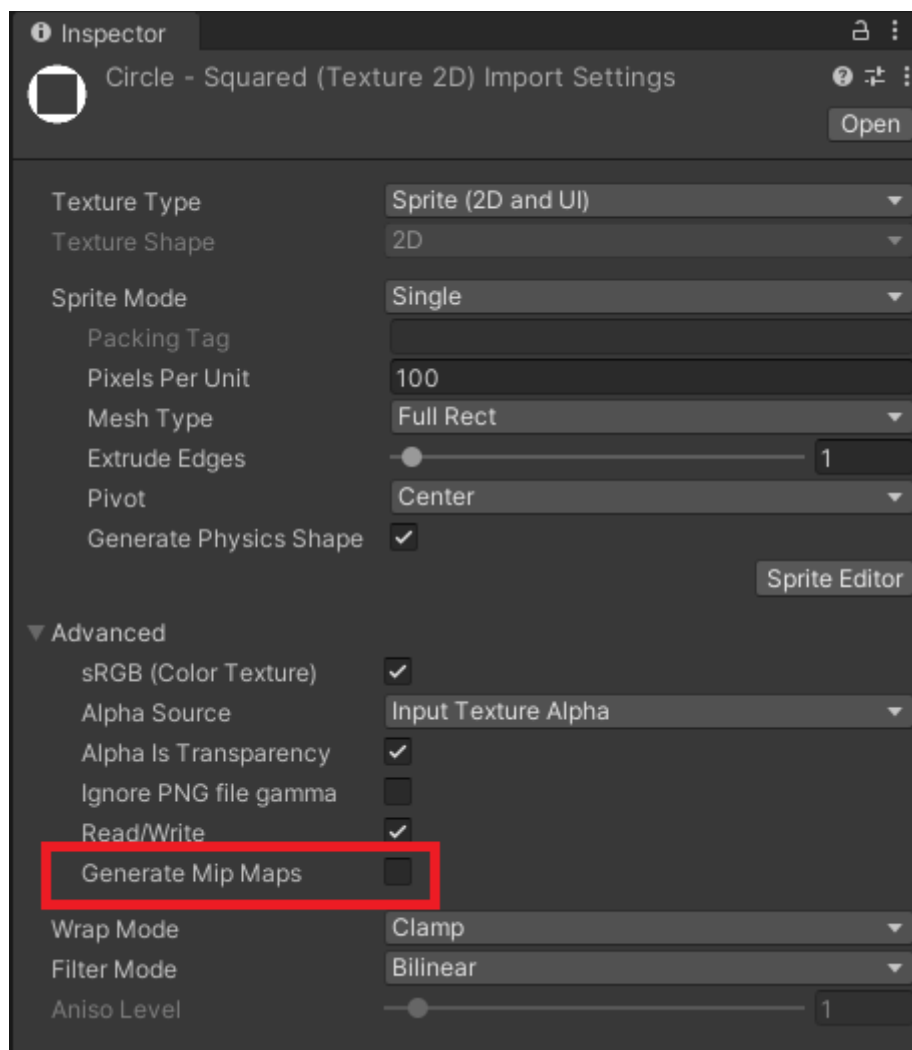
*Render Mode: Screen Space - Camera*

## 3.6.2. Mipmaps

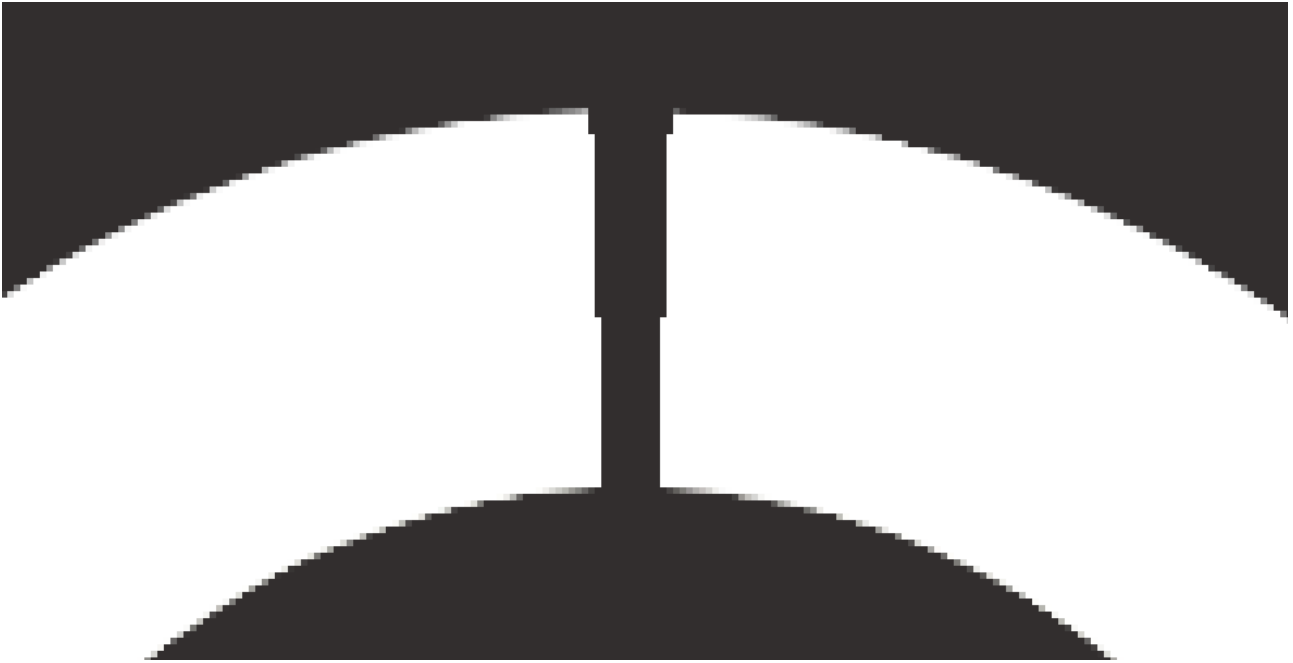
**Mipmapping** is a graphics technique where precomputed, lower-resolution versions of a texture (called **mipmaps**) are used to improve rendering efficiency and reduce aliasing artifacts. Enabling mipmaps helps smooth out edges and improves overall quality.

To enable mipmapping for the Pie Menu sprite:

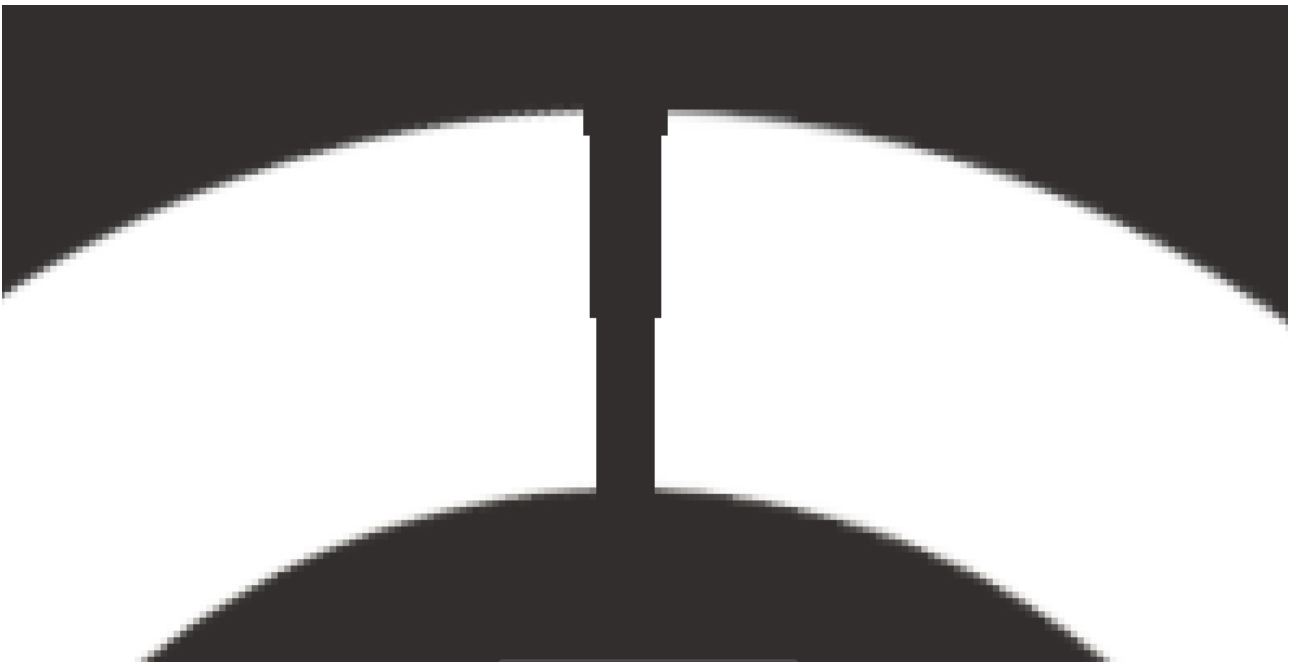
1. Select the sprite you are working with.
2. In the **Inspector**, check the **Generate Mip Maps** option.



*Generate Mip Maps Option*



*Mipmaps Disabled*



*Mipmaps Enabled*

## 4. Runtime Configuration

Not all runtime customization options are described in this section. If you are looking for additional features, please refer to **7.2 Scripts with SettingsHandler Suffix** for more details.

### 4.1. Modifying Specific Menu Items

Each **Menu Item** in the Pie Menu is stored by the **MenuItemsTracker** component within the **PieMenuItems** dictionary. To access a specific Menu Item through code, you will need its **ID**.

- An ID is assigned to a Menu Item after initialization and corresponds to its position in the **Hierarchy**.
- You can view the ID in the **PieMenuItem** script attached to each Menu Item in the **Inspector**.

By default, IDs are also included in the names of Menu Items. However, note that these names may change after using the **Hide Menu Item** functionality:

- **Names** are refreshed.
- **IDs** remain unchanged.

Example of retrieving a Menu Item by **ID**:

```
using SimplePieMenu;
using UnityEngine;

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

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

        // Here you can modify the menuItem (e.g. change icon, text, etc.)
    }
}
```

## 4.2. Changing Header and Details 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 ChangeMenuItemHeaderAndDetails()
    {
        int menuItemId = 2;
        PieMenuItem menuItem =
            pieMenu.MenuItemsTracker.GetMenuItem(menuItemId);

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

## 4.3. Changing Icons at Runtime

To change the **icon** of a **Menu Item**, you need access to the **IconGetter** script. Call the **ChangeIcon()** method and provide the target **Menu Item** and the new **Sprite**.

Example:

```
using SimplePieMenu;
using UnityEngine;

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

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

        IconGetter iconGetter =
            PieMenuShared.References.IconsSettingsHandler.IconGetter;

        iconGetter.ChangeIcon(menuItem, icon);
    }
}
```

## 4.4. Changing Rotation at Runtime

Example:

```
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, you can use:
        // newRotation = RotationCalculator.CalculateNewRotation(
        // pieMenu.MenuItemsTracker.PieMenuItems.Count,
        // pieMenu.PieMenuInfo.MenuItemSpacing);

        // Reset the current rotation
        generalSettingsHandler.HandleRotationChange(pieMenu, 0);

        // Apply the new rotation
        generalSettingsHandler.HandleRotationChange(pieMenu, newRotation);
    }
}
```

The rotation value should be between **0** and **360 degrees**.

## 5. Managing Menu Items Count at Runtime

In this chapter, you will learn how to dynamically **Disable, Hide, Add, and Restore Menu Items** in your **Pie Menu**.

### 5.1. Important Information

Please note that editing Menu Items can only be done **after the OnPieMenuFullyInitialized event** has been triggered.

This means that if you want to modify the menu immediately at the start of the game (for example, after checking certain conditions right when the scene loads), you must do it **only after this event is fired**. Please refer to **7.4 PieMenu Script** for more details.

Example:

```
using SimplePieMenu;
using UnityEngine;

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

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

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

    private void OnInitialized()
    {
        Debug.Log("Pie Menu is ready. You can now safely modify Menu Items.");
    }
}
```

Also keep in mind that the **rotation** of the menu remains unchanged after using the **Hide, Add, or Restore** functionalities. This may cause the Pie Menu to lose its **symmetrical appearance**, as the remaining Menu Items will stay in approximately the same positions. If you want to realign the menu, please refer to **4.4. Changing Rotation at Runtime**.



## 5.1. Disabling Menu Items

This will make the Menu Item visible but not selectable.

Example:

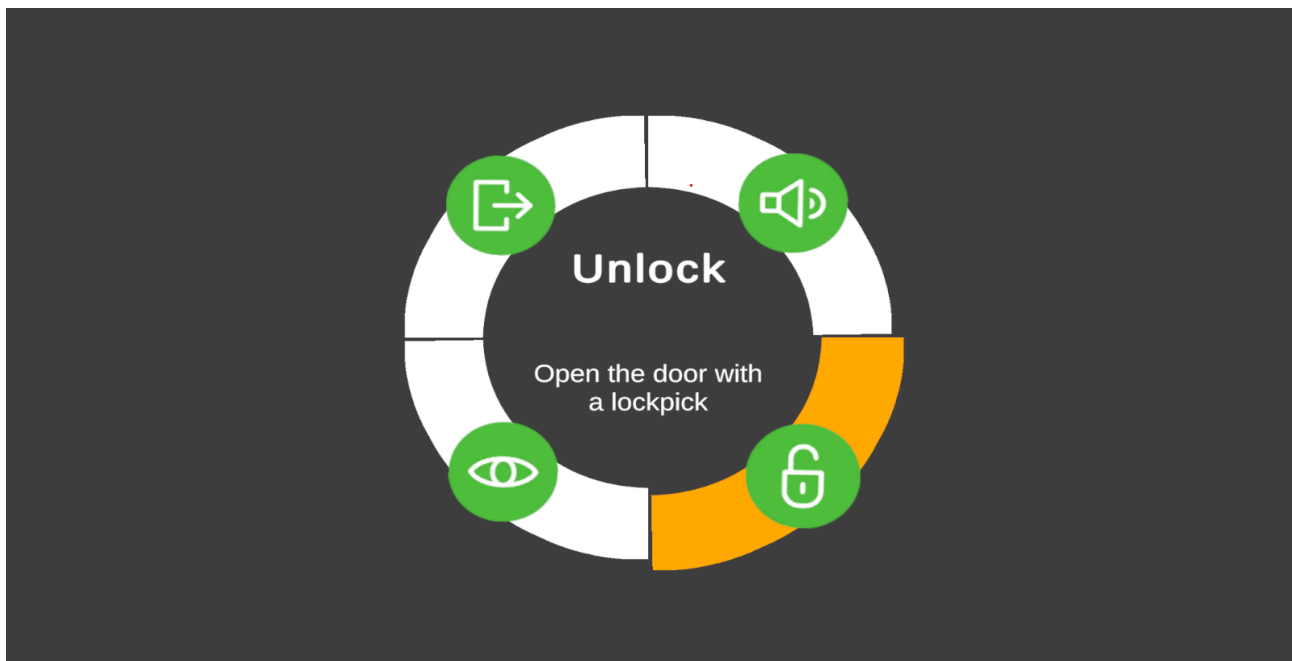
```
using SimplePieMenu;
using UnityEngine;

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

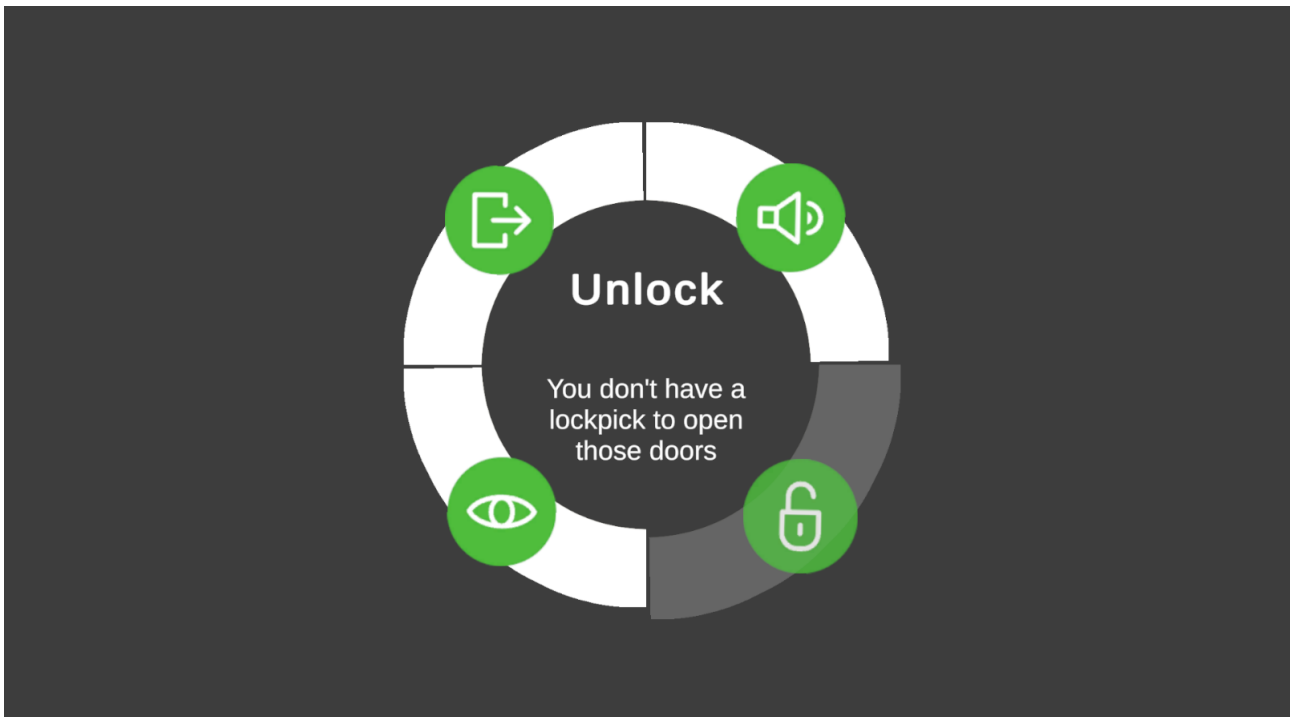
    private void DisableMenuItem()
    {
        int menuItemId = 2;
        bool disabled = true;

        MenuItemDisabler disabler =
            PieMenuShared.References.MenuItemsManager.MenuItemDisabler;

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



*Before Disabling a Menu Item*



*After Disabling a Menu Item*

## 5.2. Hidding Menu Items

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

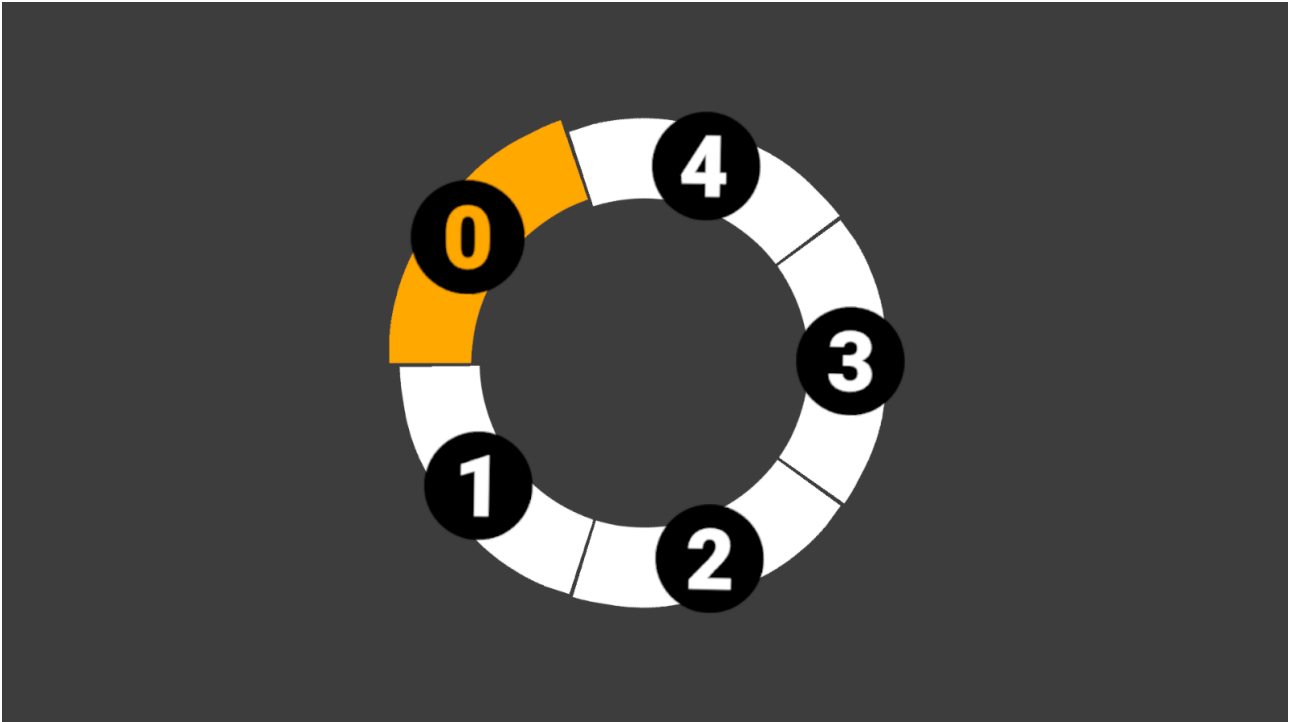
Example:

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

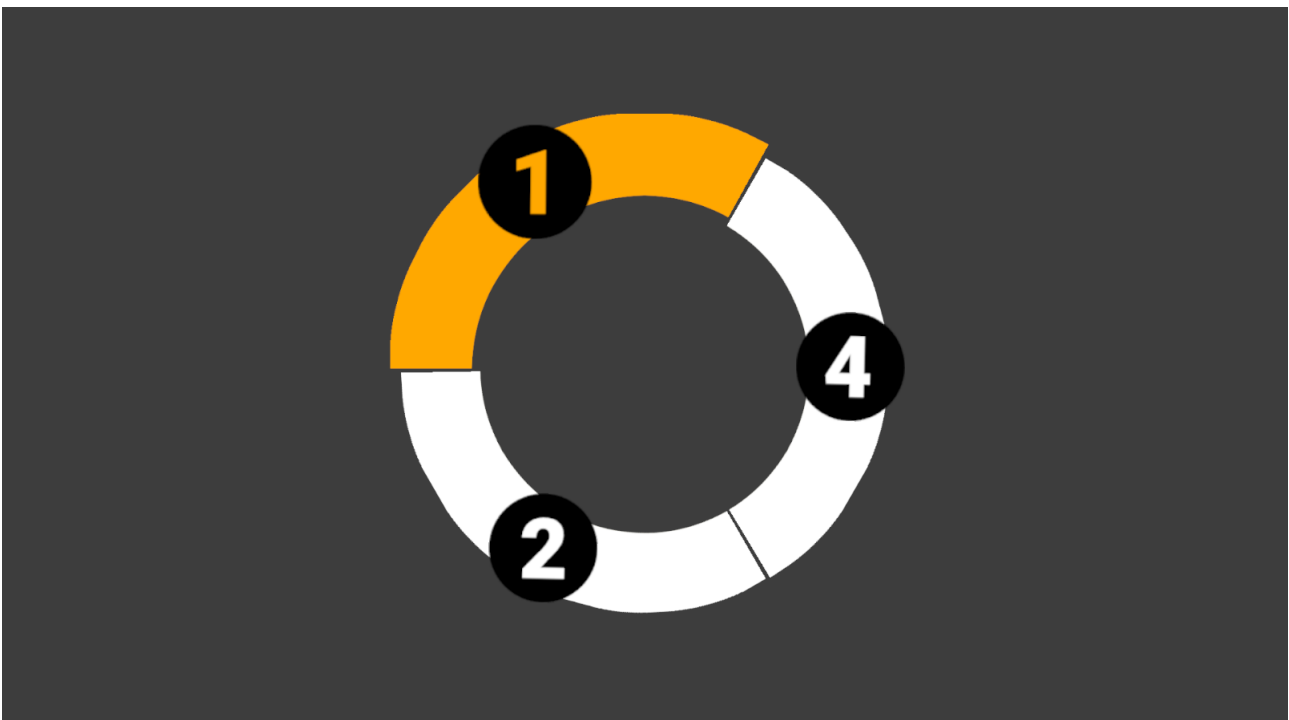
public class Example : MonoBehaviour
{
    [SerializeField] PieMenu pieMenu;

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

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



*Before Hding Menu Items*



*After Hiding Menu Items*

## 5.3. Restoring Hidden Menu Items

This functionality allows you to **restore previously hidden Menu Items** back into the Pie Menu.

Example:

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

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

    public void RestoreMenuItems()
    {
        // Restore specific Menu Items by their IDs

        List<int> menuItemIds = new() { 0 };

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

        // Alternatively, call Restore() without providing a list of IDs.
        // This will restore all hidden menu items.
    }
}
```

## 5.4. Adding a Menu Item

To add a new **Menu Item** at **runtime**:

1. **Create 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 properties will not adjust automatically.
2. Prevent saving the wrong color:
  - Due to the script that highlights the currently selected Menu Item in the **Hierarchy** window, the object may remain in its highlighted (selected) state when you save it as a prefab. This can cause the prefab to be saved with the incorrect "**Normal**" color (the highlighted color instead of the default one). If this happens, you have two options:
    - a) **Fix manually:** After saving, select the prefab and set the correct "Normal" color in the **Inspector** window.
    - b) **Save un-highlighted:** Before creating the prefab, deselect the Menu Item in the Hierarchy. Then, click and drag it into the Project window without releasing the left mouse button. Release the mouse button once the Project window is highlighted.
3. **Unpack** the **original Menu Item** to keep your new prefab as a standalone object.
4. **Configure** your Menu Item (click handling, icon, etc.).
5. Use the **MenuItemAdder** component to add the new Menu Item to the Pie Menu at the appropriate time.

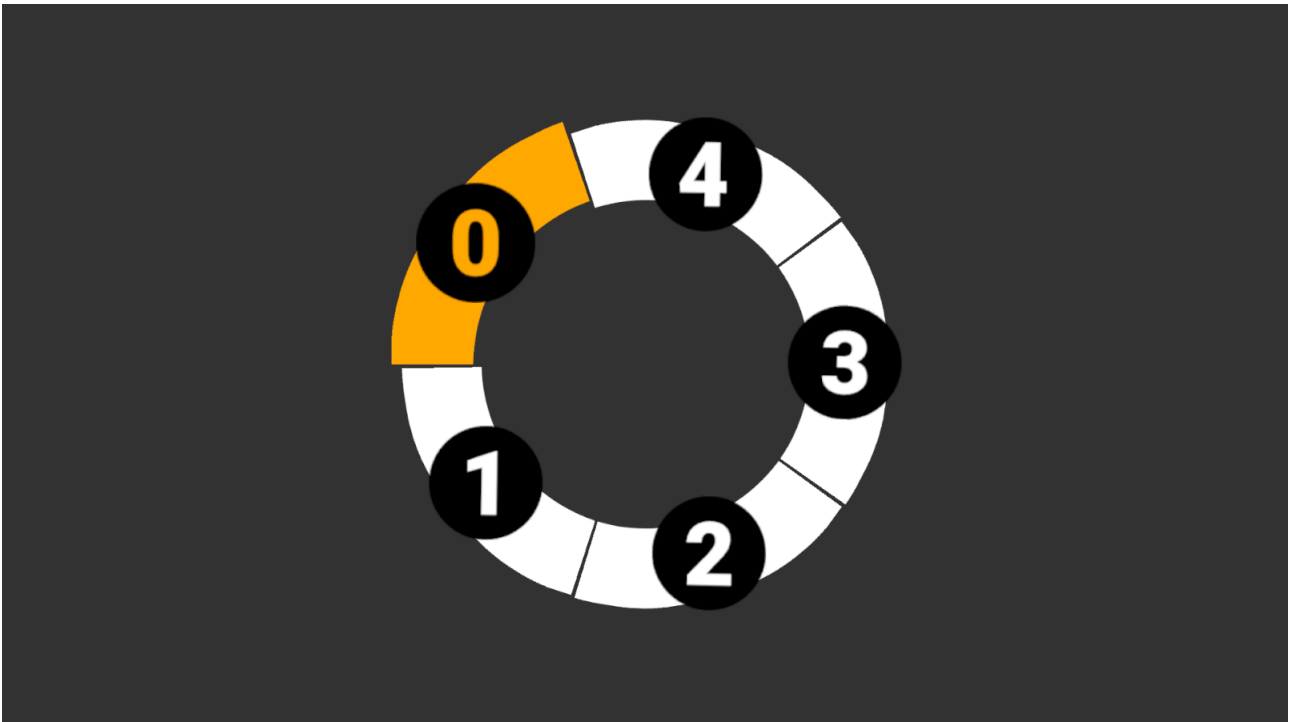
Example:

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

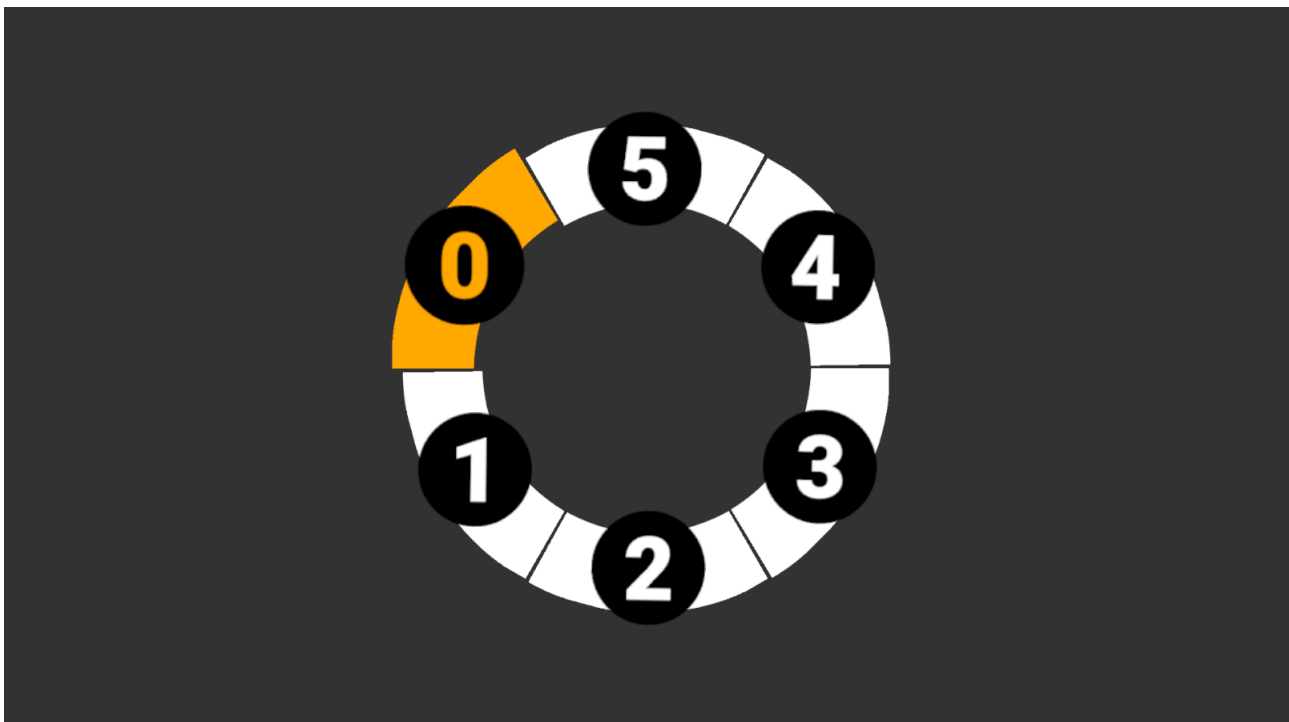
public class Example : MonoBehaviour
{
    [SerializeField] PieMenu pieMenu;

    [SerializeField] List<GameObject> menuItems;

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



*Before Adding New Menu Item*



*After Adding New Menu Item*

## 6. Advanced Pie Menu Customization

### 6.1. How to Add More Shapes

To add a new shape, you first need to create a **sprite that represents the entire Pie Menu**. The system will then automatically generate individual Menu Items by slicing this sprite using the **Image Fill** option, adjusting their sizes as needed.

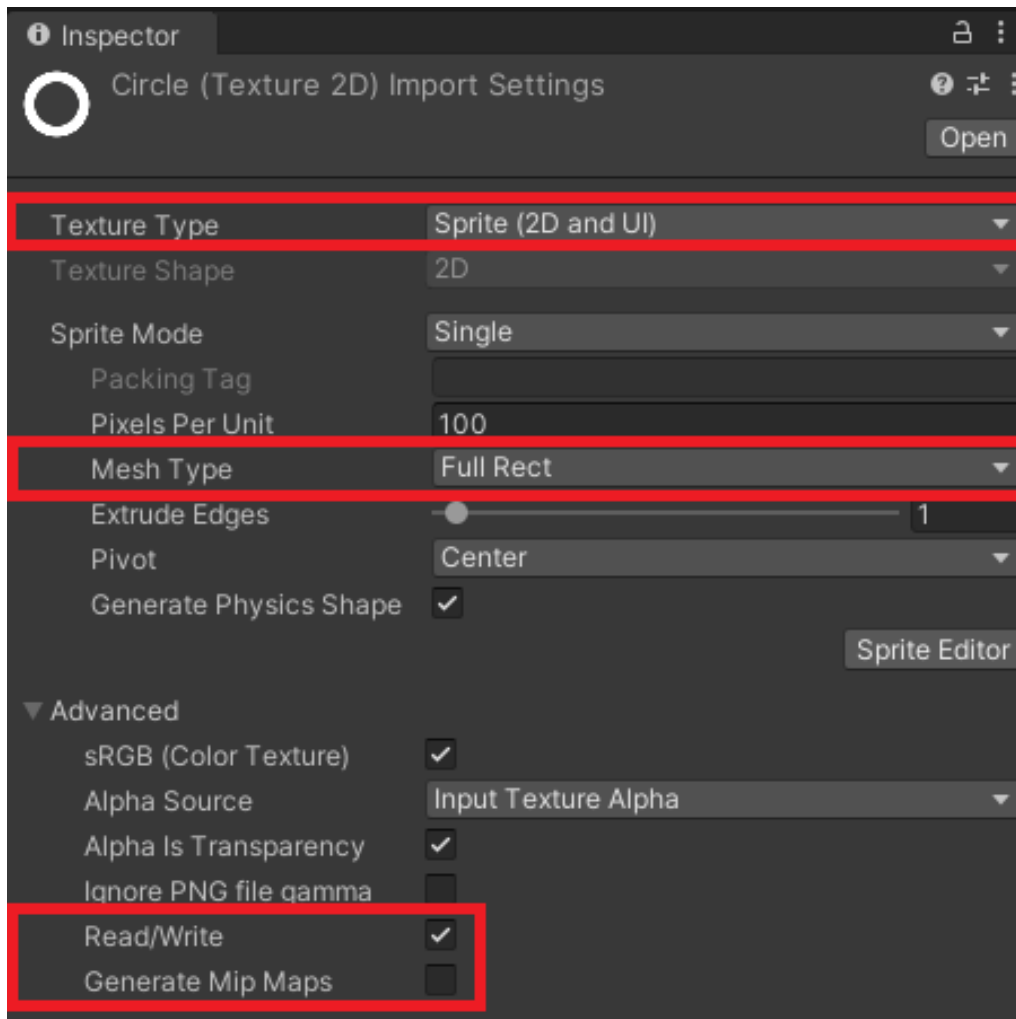
Examples can be found in `.../Simple Pie Menu/Sprites/Menu Items`.



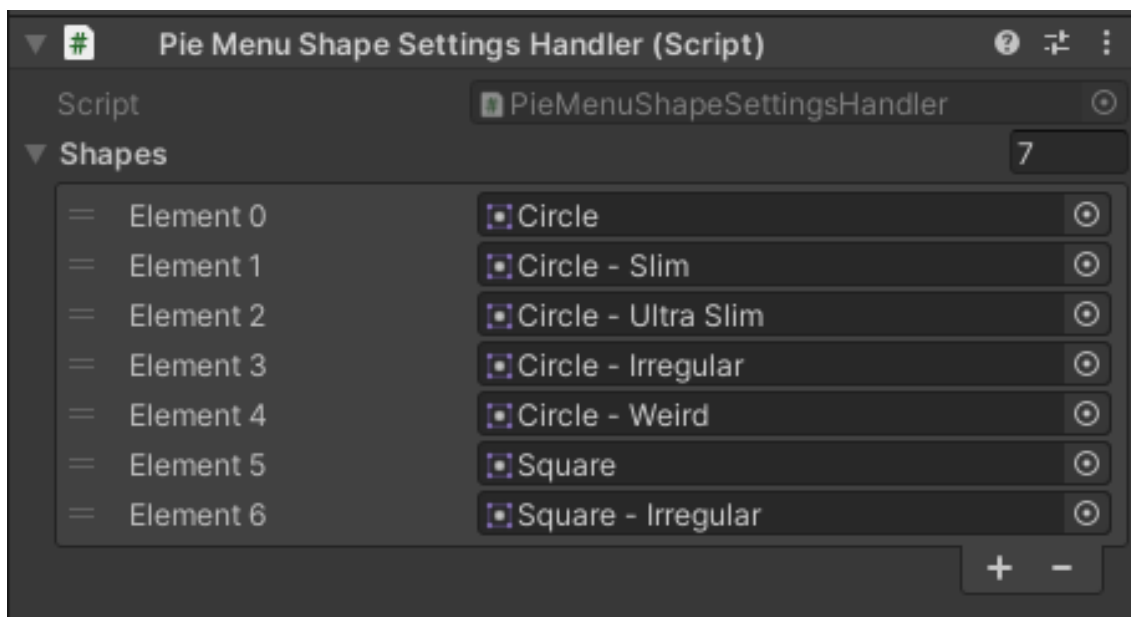
*Pie Menu Shapes*

Next:

1. Import your sprite into Unity.
2. In the Inspector:
  - Set **Texture Type** to *Sprite (2D and UI)*.
  - Set **Mesh Type** to *Full Rect*.
  - Enable **Read/Write**.
  - (Optional) Enable **Generate Mip Maps** for subtle anti-aliasing.
3. Locate the **PieMenuShared** prefab in `.../Simple Pie Menu/Prefabs/PieMenuShared.prefab`.
4. Find the `PieMenuShapeSettingsHandler` script inside it.
5. Add your sprite to the **Shapes** list.
6. After refreshing the scene, the new shape will be available for selection in the Pie Menu Inspector.



*Sprite Settings*



*PieMenuShared - Shapes List*

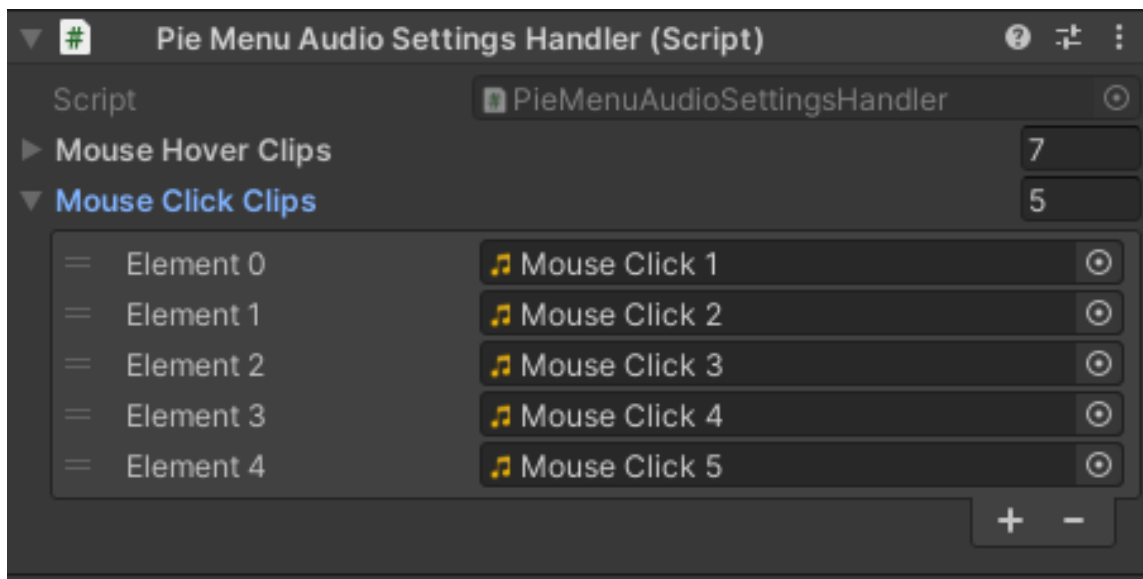


## 6.2. How to Add More Sounds

Steps:

1. Locate the **PieMenuShared prefab** at `.../Simple Pie Menu/Prefabs/PieMenuShared.prefab`.
2. Find the **PieMenuAudioSettingsHandler** script.
3. Add your audio clip to the corresponding list.

After refreshing the scene, the sound will appear in the Pie Menu Inspector.



*PieMenuShared - Audio Lists*

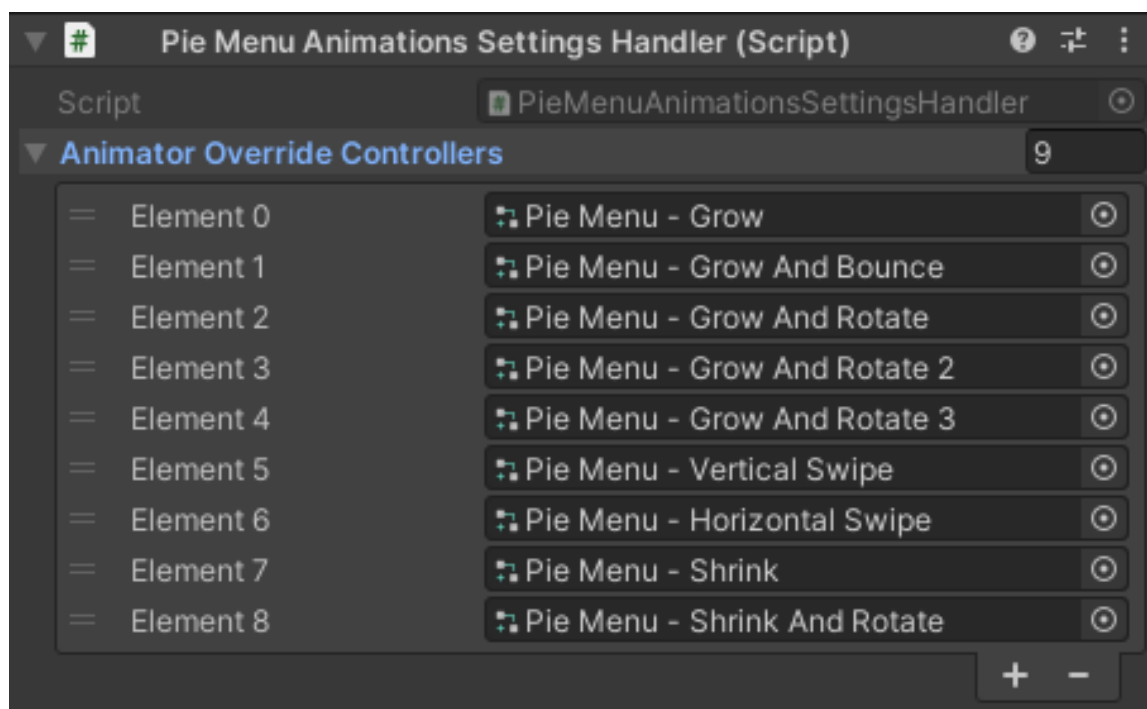
## 6.3. How to Add More Animations

The Pie Menu uses an Animator Controller to handle its animations. To add your own animation without modifying the original controller, you should create an **Animator Override Controller (AOC)**. This allows you to replace specific animations while keeping the existing animator logic intact.

### Steps:

1. Create an **Animation Override Controller** for the existing animator.
  - Go to .../Simple Pie Menu/Animations/Pie Menu.
  - Duplicate one of the controllers.
  - Rename it and replace the animations with your own.
2. Open the **PieMenuShared prefab** at .../Simple Pie Menu/Prefabs/PieMenuShared.prefab.
3. Find the **PieMenuAnimationsSettingsHandler** script.
4. Add your new Animation Override Controller to the list.

After refreshing the scene, the new animation will appear in the Pie Menu Inspector.

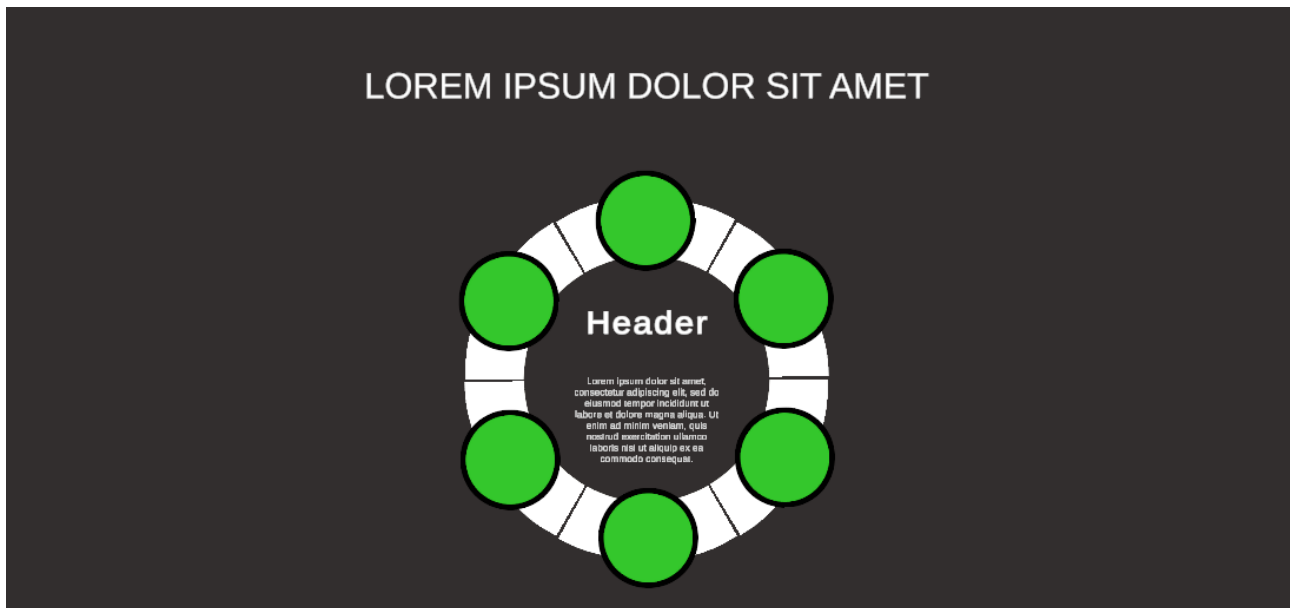


*PieMenuShared - AOC List*

## 6.4. How to Add a Header to Your Pie Menu

If you want to display a header, do the following:

1. Ensure the **Info Panel** is enabled in the **PieMenuInfoPanelSettings Inspector**.
2. In the menu hierarchy, locate the **Info Panel**.
3. Add a new **Text – TextMeshPro** component to it.
4. Configure its text, appearance, and position.

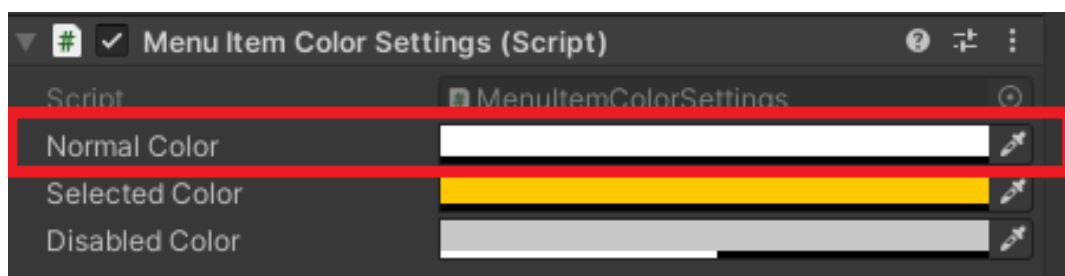


*Pie Menu with Header*

## 6.5. How to Create a Menu with Icons Only

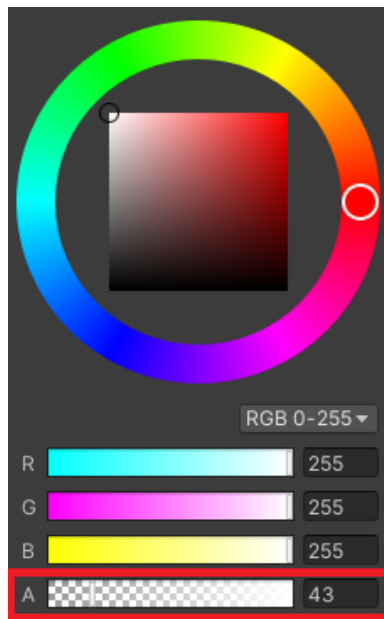
To display only icons:

1. Open the **MenuItemColorSettings Inspector**.

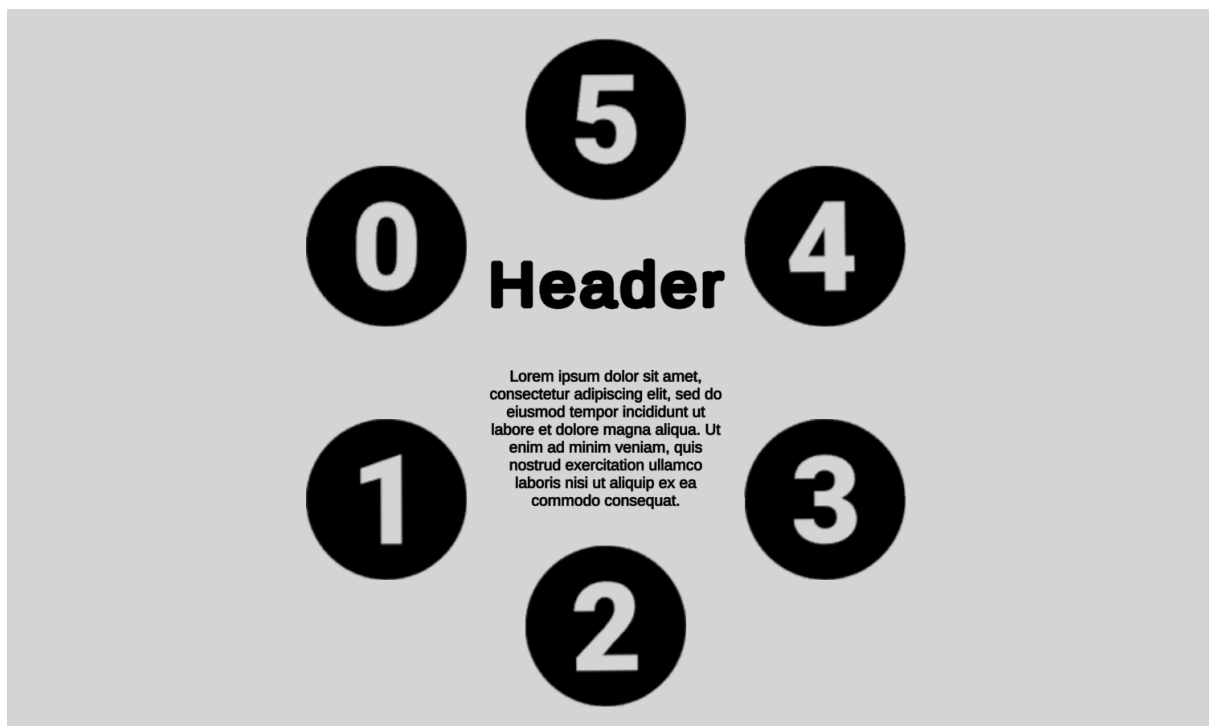


*Pie Menu - Color Settings Inspector*

2. Set the **Normal Color** transparency (Alpha value “A” in RGBA) to 0.



*Color Picker*



*Pie Menu with Only Icons*

## 6.6. Key Mapping Configuration

### 6.6.1. Old Input System

To change key bindings:

1. Open the input device script  
(by default **MouseAndKeyboard\_OLD\_INPUT\_SYSTEM**).
2. Modify the key codes in these methods:

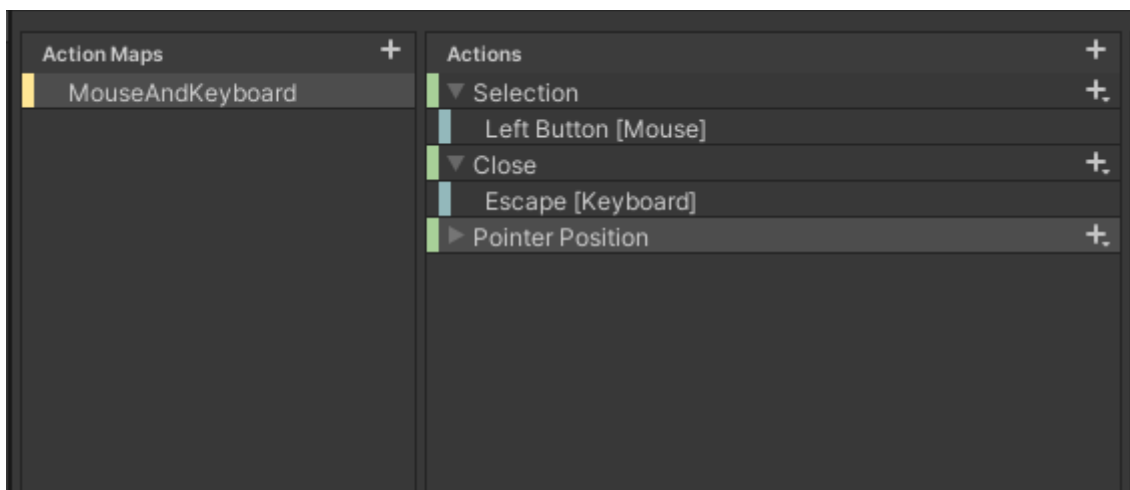
```
public bool IsSelectionButtonPressed()
{
    return Input.GetKeyDown(KeyCode.Mouse0);
}

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

### 6.6.2. New Input System

To modify bindings in the New Input System:

1. Open **Pie Menu Controls** (Input Action Importer).
2. Add new bindings or adjust existing ones for each action.
3. Assign keys that best fit your project.



*PieMenuControls*

## 6.7. How to add a new Input Device

To add a new Input Device:

1. Update the enum
  - Open the **PieMenuInputSettings** script. Inside you will find an enum called **AvailableInputDevices**. Add your new device to it.

Example:

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

2. Implement a new class
  - Create a script for your device. It must inherit from **MonoBehaviour** and implement the **IInputDevice** interface.

Example:

```
using SimplePieMenu;
using UnityEngine;

public class CustomInputDevice : 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();
    }
}
```

### 3. Implement the IInputDevice methods

- **GetPosition()** – returns the cursor's position relative to the Pie Menu.
  - Use the sample method below, but replace `mouseInput` with input from your device.
  - Always use **`pieMenu.PieMenuInfo.AnchoredPosition`**, because it accounts for screen scaling and prevents errors in the Unity editor.
  - For non-pointer devices (controllers, VR, etc.), you may need to implement a **virtual cursor** using input actions to move a 2D position across the screen. A good introduction can be found in [this tutorial](#).
- **IsSelectionButtonPressed()** – checks whether the selection button was pressed.
- **IsCloseButtonPressed()** – *(optional)* checks whether the close button was pressed. You can skip this method if you don't allow closing the Pie Menu without a selection.

#### Reference: Mouse & Keyboard implementation

```
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);
    }
}
```

#### 4. Register the new device

- Open the **InputDeviceGetter** script and add your device to the **HandleInputDevicePreferences()** method.

Example:

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

After refreshing the scene, your new Input Device will appear in the Pie Menu Inspector and will be available for selection.



## 7. Project Structure

### 7.1. Scripts with the **Settings** suffix

Scripts ending with **Settings** are responsible for handling **serialized fields** in the Inspector. These scripts are attached to every **Pie Menu** object. Please refer to **8. Detailed Explanation of Available Configuration Options** for more informations.

### 7.2. Scripts with the **SettingsHandler** suffix

Each **Settings** script has a corresponding **SettingsHandler** script, which manages changes to those settings.

For example:

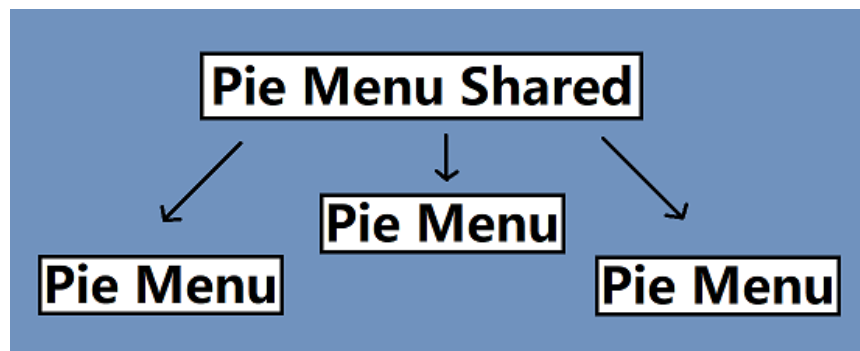
**PieMenuShapeSettings** → **PieMenuShapeSettingsHandler**

If you want to manipulate your menu via code (using values that can also be configured in the Inspector), you should explore the methods in the relevant **SettingsHandler** scripts.

### 7.3. Singleton Pattern

When the first Pie Menu appears in the scene, a singleton called **PieMenuShared** is created. It contains mostly **SettingsHandlers**, along with other shared components.

- Its lifecycle is automatically managed.
- It stays in the scene as long as there is at least one Pie Menu object.
- **Do not delete this object manually** — doing so will break editing functionality until the scene is refreshed.



*PieMenuShared Diagram*

## 7.4. PieMenu Script

The **PieMenu** script, attached to the Pie Menu object, is the **core component** responsible for managing your menu.

It contains the following elements:

- **PieMenuInfo** – which contains properties that provide information about the current state of the Pie Menu.
- **PieMenuElements** – stores references to various elements such as the Info Panel, Animator, and more.
- **MenuItemsTracker** – allows you to access references to all Menu Items in the Pie Menu.
- **MenuItemSelectionHandler** – determines which Menu Item is closest to the input device's on-screen pointer.

Additionally, the script defines two key events:

- **OnComponentsInitialized** – is triggered once all the **core components and references of the Pie Menu** are set up.
  - At this point, the script has already created and initialized objects such as **PieMenuInfo**, **PieMenuElements**, **MenuItemsTracker**, and **MenuItemSelectionHandler**.
  - Internal fields (e.g., **fill amount**, **item size**, **scale**, **rotation**, **anchored position**) are calculated and stored in **PieMenuInfo**.

In practice, this means the Pie Menu has been **pre-initialized** – all essential data is available and ready to be read.

- **OnPieMenuFullyInitialized** – is triggered once the **Pie Menu and all its components are fully initialized and ready to use**.
  - At this stage, all **Settings** have been applied, references are resolved, and the menu structure is complete.
  - This is the **earliest safe point** where you can modify the Pie Menu through code (e.g., adding, disabling, hiding, or restoring Menu Items).

This event is especially useful for scripts with the **Settings** suffix, as they often rely on the initialized values of **PieMenuInfo** and other components before applying their own settings.

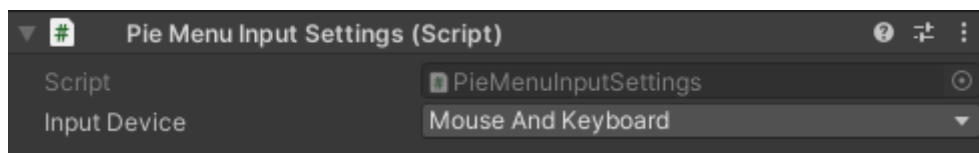
## 8. Detailed Explanation of Available Configuration Options

### 8.1. Input settings

By default, the Pie Menu is controlled with **mouse and keyboard**:

- Hover the pointer over a Menu Item.
- Click the **left mouse button** to activate the action.

Expanding support for **controllers or VR devices** is described in **6.7. How to add a new Input Device**.

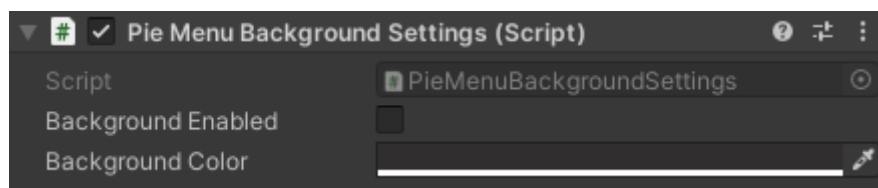


*Input Settings*

### 8.2. Background Settings

You can enable or disable the **background** of your Pie Menu and adjust its **color**.

For more advanced customization (e.g., replacing the background with an image), locate **Background – Image** in the Hierarchy and make the changes directly there. The Inspector options here act as a shortcut to that object.

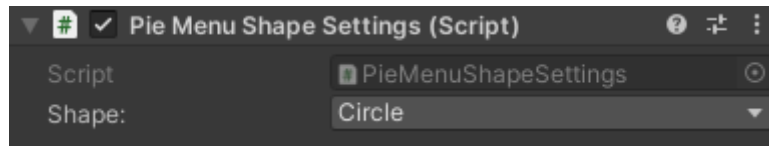


*Background Settings*

## 8.3. Shape Settings

Choose from several **default shapes** for your Pie Menu.

If you need a **custom shape**, you can create and add your own. This process is explained in detail in **6.1. How to Add More Shapes**.

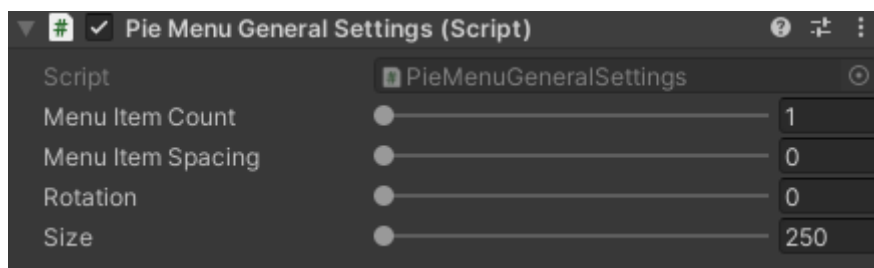


*Shape Settings*

## 8.4. General Settings

These options control the **overall structure** of the Pie Menu:

- **Menu Item Count** – sets the number of Menu Items.
  - Use this only during **initial setup**. For runtime changes, see Adding, Hiding, Disabling, and Restoring Menu Items sections.
- **Menu Item Spacing** – adjusts the spacing between items.
- **Rotation** – sets the Pie Menu's rotation. When modifying item count or spacing, rotation is recalculated automatically to keep symmetry.
- **Size** – changes the overall menu size.
  - Apply this before scaling the Info Panel or icons, since resizing resets them to defaults.



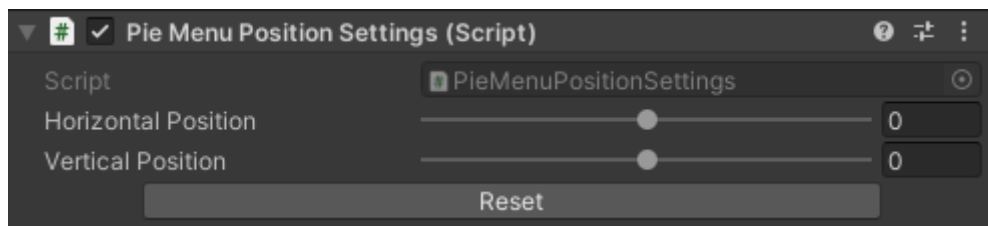
*General Settings*

## 8.5. Position Settings

Defines where the menu appears on the screen:

- **Horizontal Position** – moves the menu left/right.
- **Vertical Position** – moves the menu up/down.
- **Reset** – restores the menu to the center.

⚠ Note: Some animations (e.g., **Horizontal Swipe**, **Vertical Swipe**) ignore these settings and always position the menu at the screen center.

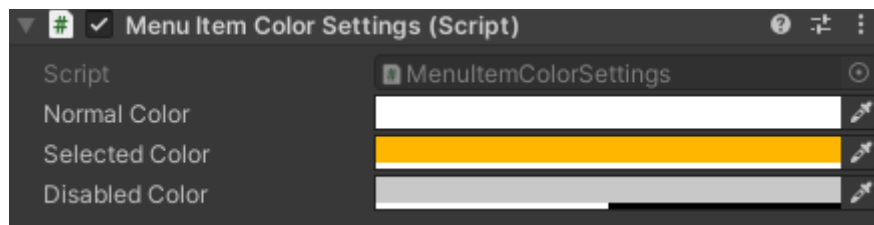


*Position Settings*

## 8.6. Menu Item Color Settings

Set the color scheme for Menu Items:

- **Normal Color** – default state.
- **Selected Color** – when highlighted.
- **Disabled Color** – when visible but not interactable.

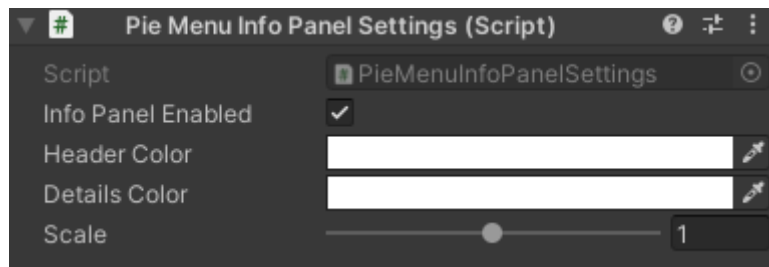


*Menu Item Color Settings*

## 8.7. Info Panel Settings

Configure the **Info Panel**:

- **Info Panel Enabled** – toggle the panel on or off.
- **Header Color** – color of the header section.
- **Details Color** – color of the details section.
- **Scale** – adjust panel size.

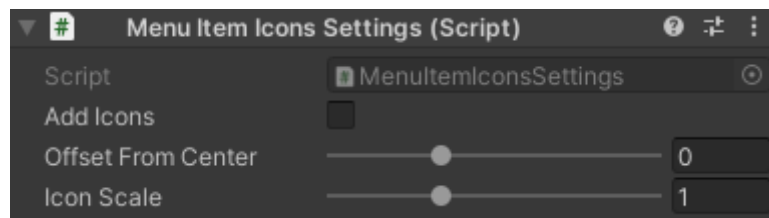


*Info Panel Settings*

## 8.8. Icons Settings

Controls how icons are displayed:

- **Add Icons** – toggle icons on/off.
- **Offset From Center** – distance of icons from the menu center.
- **Icon Scale** – size of the icons.

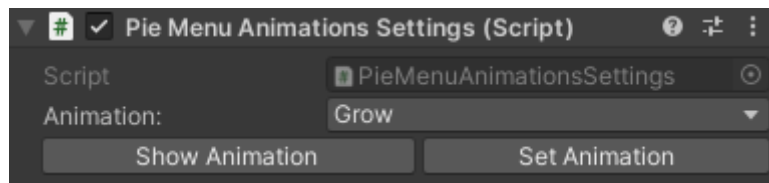


*Icons Settings*

## 8.9. Animations Settings

Configure animations for showing and hiding the menu.

- **Choose Animation** – select one of the available animations.
- **Show Animation** – preview how the selected animation looks.
- **Set Animation** – apply the chosen animation to your menu.

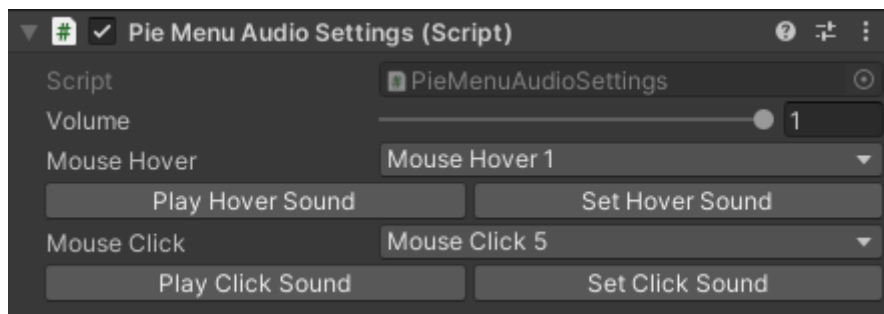


*Animations Settings*

## 8.10. Audio Settings

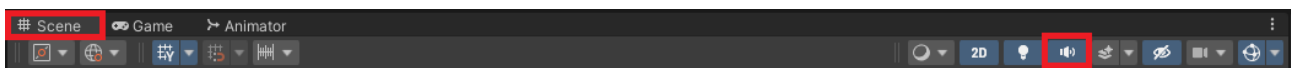
Configure **sounds** for hover and click events:

- **Choose Sound** – select one of the available sounds from the list.
- **Adjust Volume** – set separate volume levels for hover and click sounds.
- **Play Sound** – preview the selected sound.
- **Set Sound** – apply the chosen sound to your menu.



*Audio Settings*

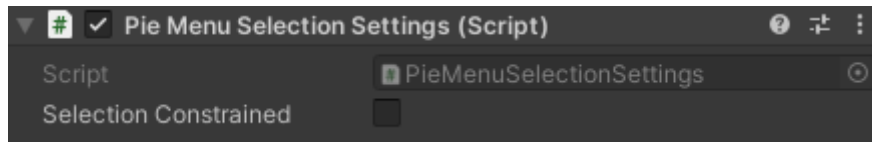
⚠ If you can't hear anything, check that **sound is enabled** in the editor (top of the Unity window).



*Mute/Unmute Button*

## 8.11. Selection Settings

This option disables selection in the center of the Pie Menu. Useful when there are many items, where small pointer movements can unintentionally change the selection.

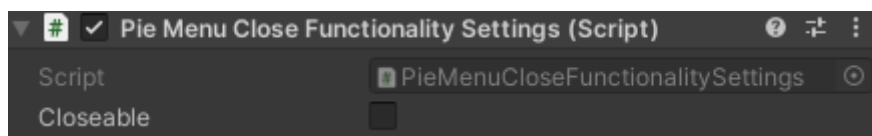


*Selection Settings*

## 8.12. Close Functionality Settings

Allows players to **close the menu without making a selection**.

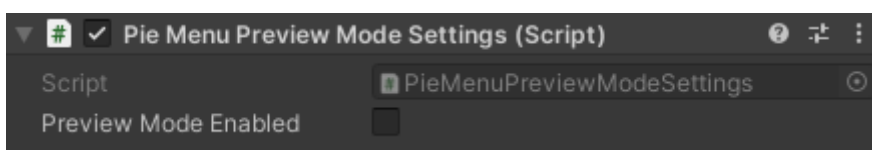
- By default, this is bound to the **Escape key** for mouse & keyboard input.
- Button remapping is covered in **6.6. Key Mapping Configuration**.



*Close functionality Settings*

## 8.13. Preview Mode Settings

When enabled, the menu **reopens automatically** after being closed. This is mainly for testing **animations and sounds** during configuration.



*Preview Mode Settings*



## 9. Best Practices & Optimization

### 9.1. Performance Optimization

It is recommended not to leave the Pie Menu permanently active in your scene while working on your project.

The reason is that its scripts are executed every time an **AssemblyReload** occurs - a Unity process that reloads all scripts when you recompile code or make changes to the project. Keeping the menu always active can cause minor performance overhead.

### 9.2. Prefab Search Optimization

The **PieMenuContextMenuExtension** script, used to create a new Pie Menu as shown in the **2.1. Getting started** section, searches the project for the **Pie Menu - Canvas** prefab. In large projects this search can be slow, so it is recommended to directly provide the correct path to the prefab inside the script.

### 9.3. Precision of Selection

When moving the cursor quickly across Menu Items, you may notice that selection is not always perfectly precise. This is intentional - for performance reasons, the selection check in the **MenuItemSelectionHandler** script is limited by the **HandleSelection** coroutine, which updates the current selection **10 times per second**.

If you need higher accuracy, you can increase the update rate by decreasing the value of the **coroutineDelay** field. Keep in mind that lowering this delay improves precision but increases resource usage.

# 10. Resources and Feedback

## 10.1. Discord Server

To support and expand this asset, I've created a dedicated [Discord](#) server.

Here you can:

- Report issues or bugs.
- Suggest new features.
- Get answers to your questions in a dedicated help channel (check it first - you may find your solution already posted).
- Share materials or projects that use the **Simple Pie Menu** asset.

## 10.2. Resources

The icons used in this documentation for Menu Items come from: <https://www.iconpacks.net/>.

## 10.3. 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**.