Input Icons for TextMeshPro

Requirements (from package manager. Window → Package Manager):

- **TextMeshPro 2.1.6 or higher**
- **Input System 1.2.0 or higher** (**1.3.0**, **1.4.4** or higher recommended)
  (avoid 1.4.1, 1.4.2 and 1.4.3 as they produce errors on exiting play mode)

With Input Icons we can easily display current input bindings in TextMeshPro texts. Once set up, we can use the style tag of TMP (**<style=NameOfActionMap/NameOfAction>)** to display keyboard keys and gamepad controls inline with other texts. Since we use the style tag and we can update the default style sheet at runtime, we are able to update all TMP texts whenever we change the input device (e.g. if the player was using a keyboard and decides to switch to a gamepad). Additionally we can update the displayed sprites whenever we make changes to the input action bindings.

Included in this package are sprites for keyboard and mouse and sprites for XBox, PlayStation and Nintendo controllers. For any other controller, this system will display the sprites of the fallback Scriptable Object (this is set to XBox by default since most players on PC -according to Steam statistics - use XBox controllers).

Video guide: https://youtu.be/pbm0UvPGCag

Overview

Updating From An Earlier Version

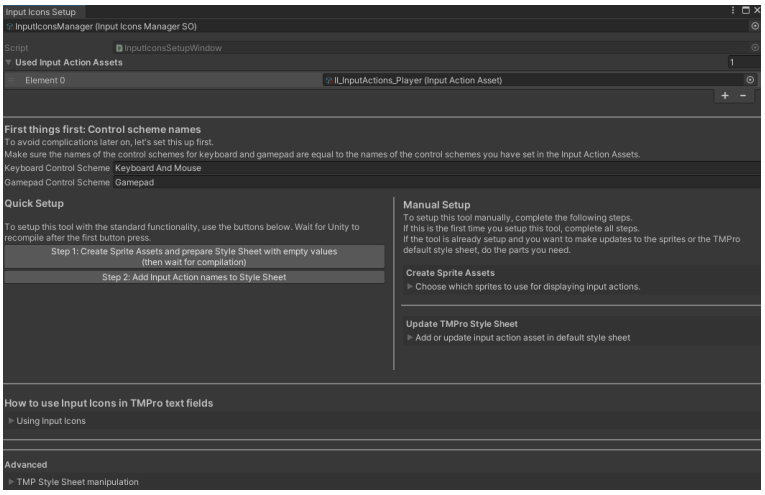
When you update to version 1.1.0 or higher, you might experience missing input icons in your scenes. **Do the Quick Setup** in "**Tools→Input Icons Setup"** again and everything should be working as earlier. **Check the** "**Used Action Assets" list** at the top of the setup window or in the Input Icons Manager, as the list might have changed.

The reason why you might be experiencing this issue is because of performance improvements. Previously, the needed styles for the default style sheet were calculated every time the user switched devices. Now, to save calculation time, the Input Icons Manager calculates all needed style data once and updates the default style sheet with the respective data when needed. Doing the Quick Setup again will make sure that all needed actions from our Input Action Assets will be stored in the manager to be available when needed.

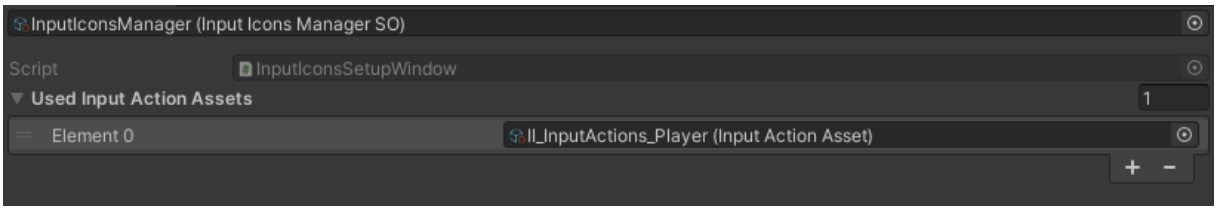
Getting Started

To get started we can choose between using the sprites that come with this package or to use different sprites. Sprites for keyboard/mouse and controller sprites for Play Station, Steam, Nintendo Switch and Xbox are included and already set up in the package. Scriptable objects which store the data for all our sprites are located in the Assets/InputIcons folder.

From the toolbar, open "**Tools/Input Icons Setup"**.



First we need an Input Action Asset containing our input actions. If we haven't already, let's head to the Package Manager (Window/Package Manager) and import the Input System package (this tool requires **Input System 1.2.0** or higher to save and load changes to the input bindings). Then create a new Input Action Asset and set up the ActionMaps and Actions we need, or copy the one which comes with this package and adjust as needed. Add the Input Action Asset to the "Used Input Action Assets" list.

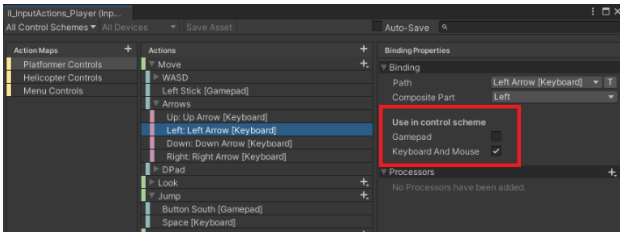


Setting Up The Input Action Asset

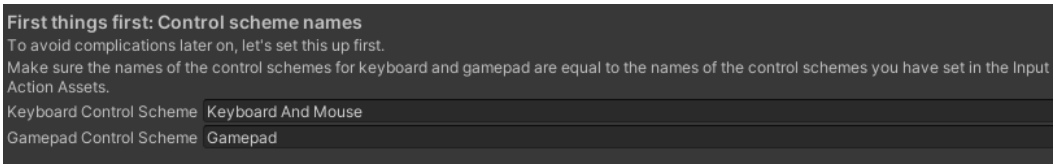
Our Input Action Assets need control schemes for keyboard and gamepads. We have to add devices to the control schemes so the PlayerInput component can know which control schemes to use once we switch to gamepad or keyboard.

Verifying Control Scheme Names

Let's be sure that the Input Action Asset has **control schemes** set up **for keyboard and gamepad** control. Or only one of them, depending which device(s) we want to support.

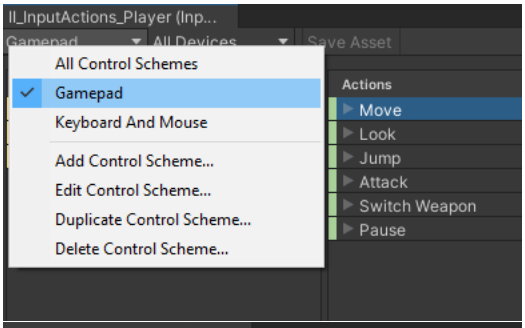


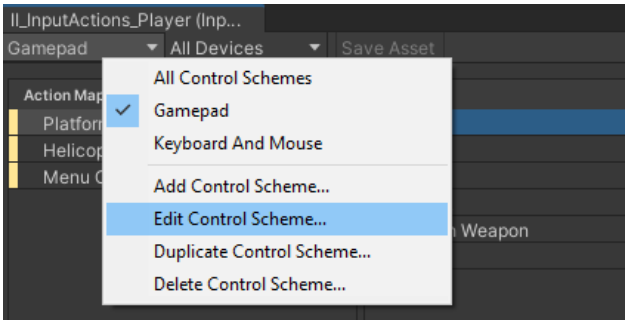
The InputIconsManager needs to know about these names. Fill in the names in the required fields in the setup window or in the InputIconsManager scriptable object.

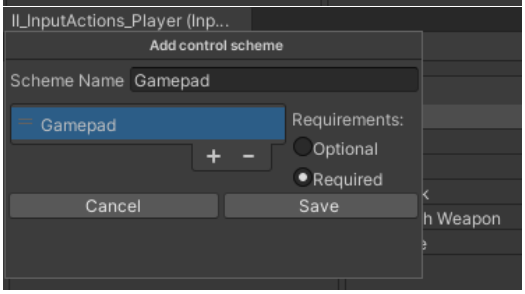


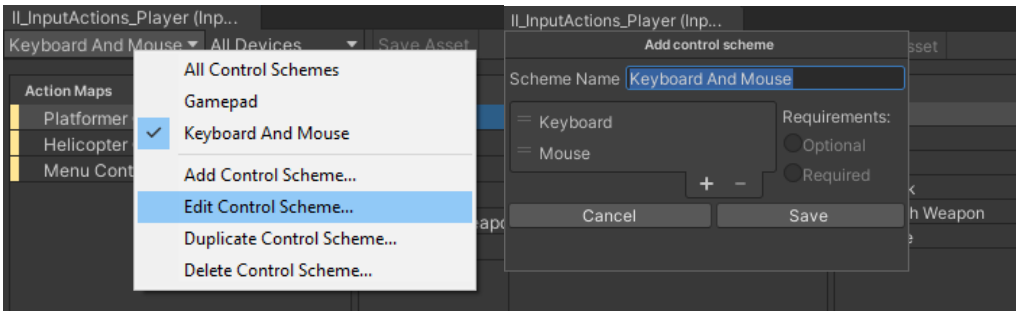
Adding Devices To Control Schemes

We should now have two control schemes. One for gamepads, one for keyboard and mouse. We need to add devices to these control schemes in order for the PlayerInput component to know which control scheme to use when the user switches to a different device.







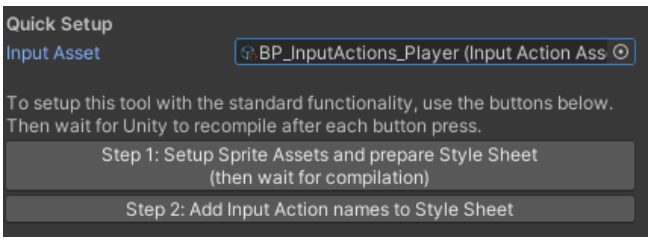


Once we have set up the Input Action Asset and the control scheme names + devices, we can continue with either the Quick Setup (recommended) or the Manual Setup which breaks up the setup process into smaller steps.

Quick Setup

With the Input Icons Setup window opened (“**Tools/Input Icons Setup”** in the toolbar) we can use the buttons to setup Input Icons. These steps are here described in more detail.

Step 1: Once we have our Input Action Asset, we can start by creating Sprite Assets, filled with our keyboard and gamepad sprites. They will be placed in the default folder for Sprite Assets (usually "Assets/TextMesh Pro/Resources/Sprite Assets"). In the same step we can prepare the TMP Style Sheet by filling it with empty values which we can fill in the next step with the input action names of our Input Action Asset. (Unfortunately we can not insert the action names right away since we have limited access to the TMP Style Sheet – maybe this will change later but for now we have to do it this way.)



Step 2: After we have prepared the style sheet with empty entries, we can now insert the input action names of our Input Action Asset by pressing the second button.

Done! We can now display our binding icons by typing **<style=NameOfActionMap/NameOfAction>** (for example if we want to display the move controls for the platformer control scheme we would write **<style=Platformer Controls/Move>**). Once we start the game, the InputIconManager will update the default style sheet to display the sprites.

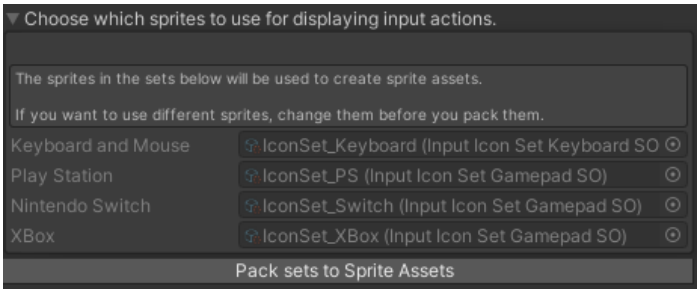
Custom Setup

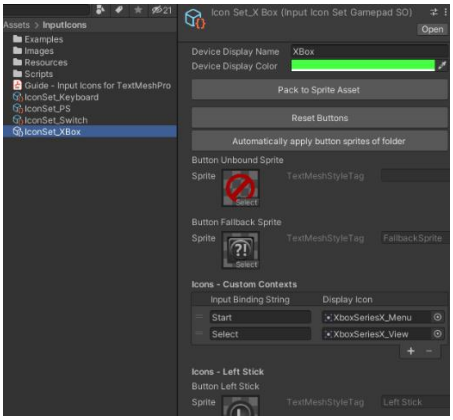
Open "**Tools/Input Icons Setup"** from the toolbar to find the custom setup section. This section comes in handy if at some point we

- decide to use a different set of sprites
- want to add more Input Action Assets to display more actions (be aware that action names must be different for each action we want to display as it only makes sense to save one instance of an action name into the default style sheet)
- make changes to existing Input Action Assets

Using Different Sprites

To use a different set of sprites, other than the one which comes with this package, we need to update the InputIconSetSOs and then create the necessary sprite assets, containing all our sprites.



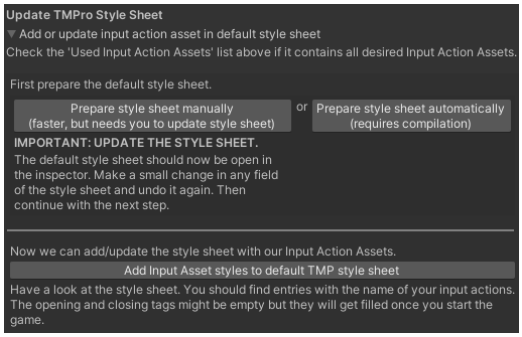


We can make use of the "Automatically apply button sprties of folder"-button to search for sprites in the current folder (and subfolder) to quickly assign new sprites to the scriptable object. Keep in mind that the naming of the sprites is very important if we use this button. To automatically assign the

sprites to the XBox icon set for example, we would move the icon set asset to the folder containing the XBox sprites and then use the button. Then we should check if every sprite was applied correctly.

Important: Once we have all the sprites setup, press the button ("Pack sets to sprite assets") to create/override the necessary Sprite Assets. They will be placed in the default folder for Sprite Assets (usually "Assets/TextMesh Pro/Resources/Sprite Assets")

Adding Or Updating Input Action Assets



We can use the Custom Setup section of the Input Icons Setup window to add and update entries in the TMP default style sheet. (It is recommended to use only one Input Action Asset and add control schemes for various controls – e.g. control schemes for player controls and menu controls)

We have to prepare the style sheet, by adding empty values which can later be filled with the names of the action names in the Input Action Asset. In this section we have 2 buttons which basically fulfill the same purpose.

1. We can choose to use the first button ("Prepare Style Sheet manually") and the empty values will be added. However, the Style Sheet Asset needs to updated/saved and the only way to do this for Style Sheets (at least the only way I found other than recompiling the project …) is to make a simple change in the inspector. So just adding a character to any field in the Style Sheet and removing it again is enough.
2. Using the second button ("Prepare Style Sheet automatically) will also add empty values and we don't have to make manual changes to the style sheet, but Unity will have to compile and thereby save the changes made to the Style Sheet.

Now with the default style sheet prepared for the selected Input Action Asset, we can use the last button to write the action names into the style sheet.

IMPORTANT: Add Input Icons Activator Prefab to Scene

It is important to know that ScriptableObjects only call their OnEnable methods when either being selected in the inspector or when being referenced by a MonoBehaviour or when being referenced by another ScriptableObject which is referenced.

Therefore the best way to ensure that the InputIconManagerSO (which is a ScriptableObject) works properly, is to reference it in the scene. We can easily do this by just dragging the **II_InputIconsActivator prefab** into the first scene.

How To Use Input Icons

Lets make sure we have completed the setup:

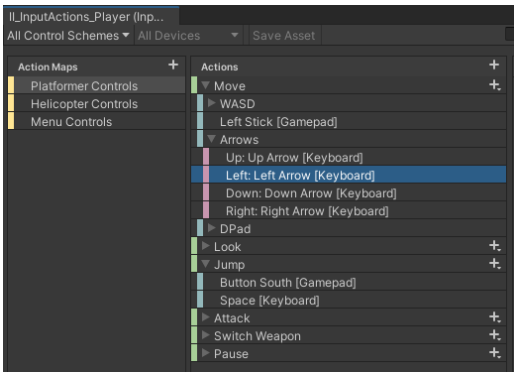
1. The control scheme names of our Input Action Asset(s) match the names in the Input Icons Manager (found in **Assets/InputIcons/Resources/InputIcons**)
2. We have added the Input Action Assets we want to use to the list of used Input Action Assets in the setup window or in the Input Icons Manager scriptable object.
3. We created the Sprite Assets out of the Input Icon Sets and they are stored in "**Assets/TextMesh Pro/Resources/Sprite Assets**"
4. We added the actions of our Input Action Asset(s) to the TMPro Style Sheet ("**Assets/TextMesh Pro/Resources/Style Sheets**")

Displaying Input Bindings in TMPro

Once we have completed the steps to setup Input Icons, we can display sprites related to our input actions by using the **TMPro-style tag (**http://digitalnativestudios.com/textmeshpro/docs/rich-text/#style**)**.

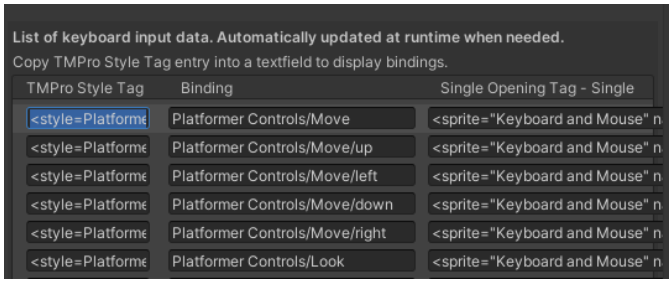
Once we enter play mode, the opening tags of the styles in the style sheet should be filled with the corresponding tags to display the icons of the currently used device. This happens automatically through the Input Icons Manager.

We can use **<style=NameOfActionMap/NameOfAction>** inside a TMPro text field to show the bindings of an action.



To show the current binding for a jump action in the "Platform Controls" action map, we can write **<style=Platform Controls/Jump>** into a TMPro text field. Similar, for a move action we would write **<style=Platform Controls/Move>** and it would likely display "W A S D" or the arrow key sprites for keyboard input - or the left stick sprite if the user is using a gamepad.

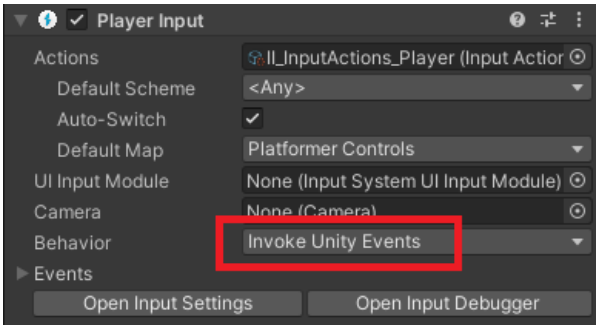
If we don't want to memorize all our action maps and bindings whenever we want to add a binding into a text field, we can open the Input Icons Manager and scroll down. There is a list available which stores all kinds of data used to display bindings. We can just copy and paste any entry in the first column ("TMPro Style Tag") into a text field to display the correlated binding.



Keyboard And Gamepad Support

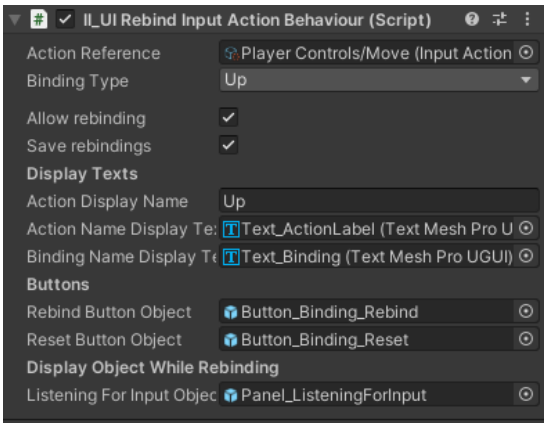
To support keyboards and gamepads, our Input Action Asset(s) need to be correctly setup (see section "**Setting Up The Input Action Asset**")

The InputIconsManager will search for a **Player Input component** in the scene every time a new scene is loaded.
The manager subscribes to the “Controls Changed Event” and will update the displayed icons every time the event fires. Therefore it is necessary that the behaviour of the PlayerInput component is set to “**Invoke Unity Events**”. If the behaviour is set to “**Send Messages**” or “**Broadcast Messages**” a ”**InputIconsDeviceChangeDetection**”-script will be added to the gameobject which has the Player Input component. This script will listen for device changes and update will call **InputIconsManagerSO.HandleControlsChanged(PlayerInput input)** when it detects a change. In case we use the “**Invoke C Sharp Events**” behaviour, we will have to call the above method manually to reflect changes to the current input device.



Rebind Buttons

In the examples folder we can find the II_UI_RebindActionObject which is a prefab we can use to rebind our input. We might want to create our own prefab though with custom visuals but this prefab is a good starting point.



The script takes an Input Action as a reference which we can then rebind by activating the button and then pressing the desired new key.

The toggle “Allow Rebinding” can be turned off for input actions we want to display to the player but should not be rebound.

The toggle “Save rebindings” is turned on by default on all rebind buttons and will cause the overridden input mappings to be stored in the PlayerPrefs. Disabling this toggle will disable it on all rebind buttons.

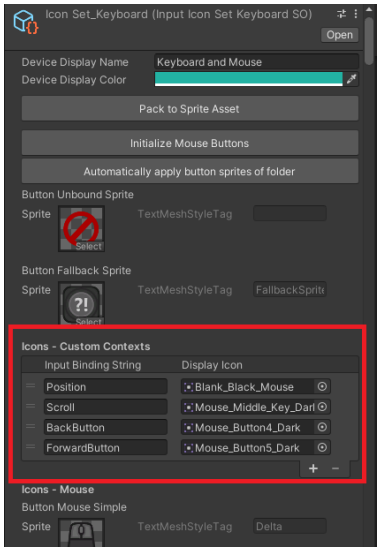
The text field “Action Display Name” provides a quick way to change the text of the action name label without having to select the text object in the hierarchy.

Limitations

- Supported keyboard layouts (with the exception of special characters) are **QWERTY**, **QWERTZ** and **AZERTY.** For all other keyboard layouts, the QWERTY layout (which is the most commonly used layout) will be used to display action bindings.
- If we use more than one Input Action Asset, we should not use the same names for the action maps across these Input Action Assets. Doing so might cause the manager to override style strings in the TMPro Default Style Sheet.

Customization

Adding Custom Context Sprites



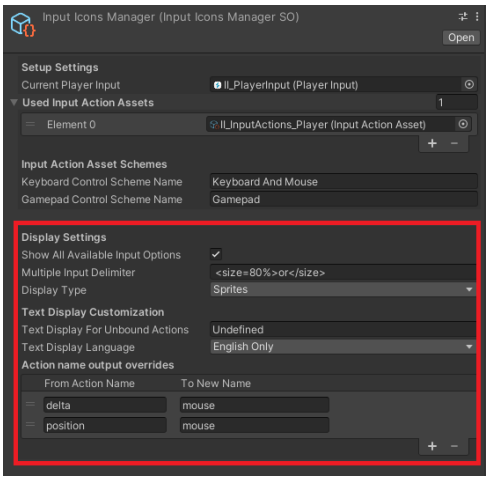
The preset Input Icon Sets that come with this package (in the folder **Assets/InputIcons/**) already have sprites defined for all common input keys, gamepad buttons and joysticks. In case we need more input controls, we can add them in the Custom Contexts list in the Input Icon Sets. We need the input binding string and a display icon, to add a new binding.

An easy way to find the input binding string for an input is to use the script “II_UITextDisplayAllActions” which is in the example folder of this package. Add it to a TextMeshProUGUI object to display current input bindings. Then we can use a rebind button to override the current binding with the new binding we want to add. The input binding string for that binding will be displayed in the brackets. For the mousewheel, the input binding string would be “Scroll” for example.

Important: Whenever we make changes to an Input Icon Set, we will have to create a new Sprite Asset out of the Input Icon Set. Otherwise, our changes only exists in the Input Icon Set, but the desired sprite will not be available in the Sprite Asset used to display sprites within TMPro texts.

Display Settings

The InputIconsManager provides options for how we want to display the input action bindings.



Show All Available Input Option: We can switch between showing only the first available binding, e.g. “WASD” or all available bindings, e.g. “WASD or Up Left Down Right” for a move action.

Multiple Input Delimiter: If we show all available options, we can define a delimiter between these actions. This delimiter can have TMPro tags for customization. The standard delimiter is “ <size=80%>or</size> “ and will therefore display a slightly smaller “or” between different bindings.

Display Type: input actions as sprites, text, or text in brackets.

Text Display For Unbound Actions: If we display bindings as Text or TextInBrackets, we can choose which text should be displayed for an unbound action.

Text Display Language: If we display bindings as Text or TextInBrackets, we can decide if we want to display this text in English or in System Language (the language currently used by the device, might produce very different results in text length and is generally not recommended).

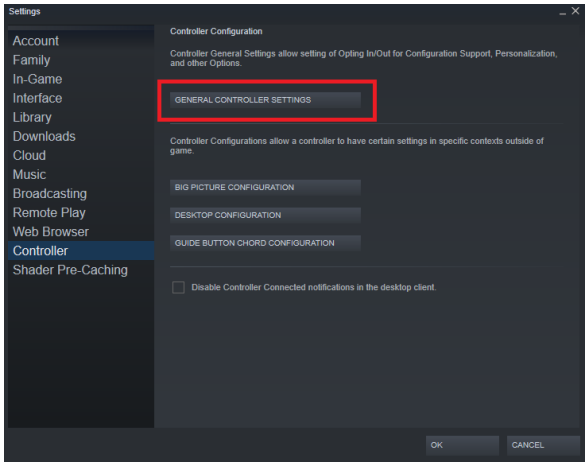
Action name output overrides: If we use Text or TextInBrackets as the option to display input actions, we can override the text which would be displayed. For example it makes more sense to display [MOUSE] instead of [POSITION] for pointer controls.

Steam Integration

Integrating the Steamworks.NET (https://steamworks.github.io/) into the project can cause some problems for the new Input System as Steam seems to literally hijack XInput. Therefore whenever we use steamworks we have to use some helper methods provided by steamworks in order to detect the currently used device. This is done in the script “InputIconsSteamworksExtensionSO” which will be active when we are using steamworks.

Fortunately, we don’t have to do a lot to support Input Icons in Steam, as the above mentioned script already handles the most important things for us. However, there are some limitations in when playing via Steam. Since we use the Steam helper methods, it is more difficult to detect the currently used gamepad. For this reason Input Icons will display the used gamepad on index 0, which means that if we have a connected PS controller and a connected XBox controller, Input Icons will always display the first connected controller and the only way to display the other one is to disconnect one controller and restart the game.

Another issue that I noticed when developing for Steam is that the gamepads might sometimes not respond altogether. This can happen in the Unity editor or in a build when the editor or the build is run via Steam. A possible solution is to go into the Steam launcher and go to Steam -> Settings -> Controller -> General Controller Settings and disabling the Configuration Support for PlayStation, Xbox and Switch Pro. Then restart Unity or the game. By doing this, Unity’s Input System will regain control and the checks for the current device will work the as if we were not using Steam. Of course we can not expect players to go into these settings and disable them, but even if they are enabled, our games using Input Icons will still be able to display the correct keyboard and gamepad icons as long as users do not switch between different gamepad types (PS, XBox, Switch).

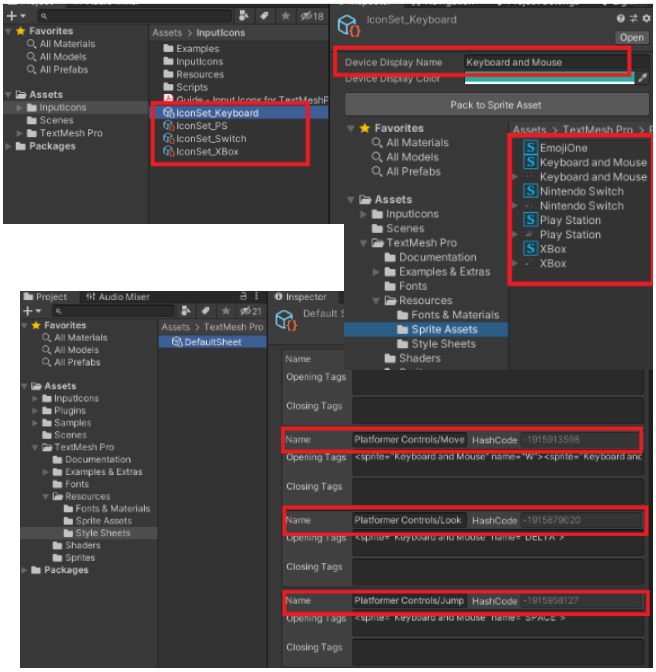


Troubleshooting

Most problems appear from an incorrect setup. Remember, whenever you make changes to an Input Action Asset, to also update the style sheet by using the setup tool (Tools → Input Icons Setup).

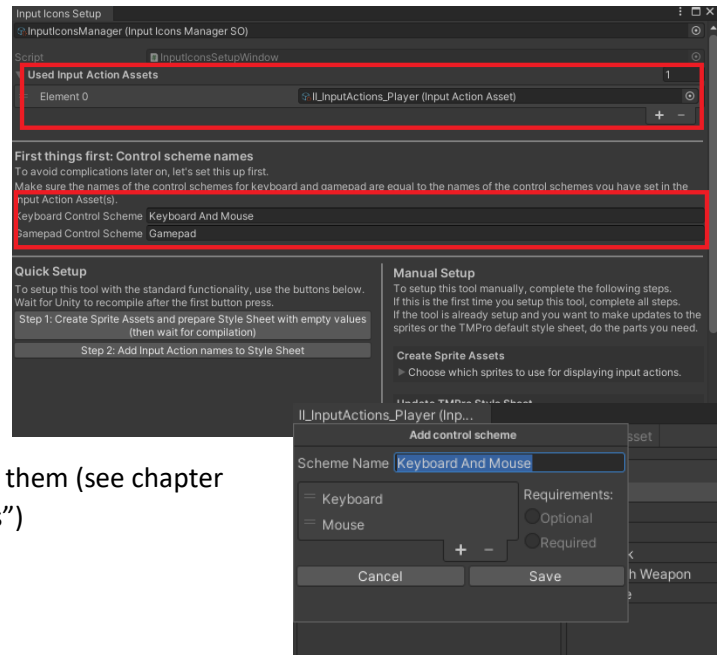
If you have updated to a new version of Input Icons, the values in the InputIconsManager might very likely have changed. To solve this, do the Quick Setup again and everything should work again.

Have our Sprite Assets been created correctly and does the TMPro Default Style Sheet contain our actions?



- **Sprite Assets**: In "Assets/TextMesh Pro/Resources/Sprite Assets/" we should find assets containing the sprites of the keys we want to display. It is important that these assets have the same names as the Icon Sets in "Assets/InputIcons/".
- **TMPro Default Style Sheet**: The Default Style Sheet should somewhere contain the actions defined in our Input Action Assets. Check the style sheet in "Assets/TextMesh Pro/Resources/Style Sheets/"
  Special case: The styles were added correctly, but **the styles disappear once we restart Unity** … the Default Style Sheet sometimes won't save the changes. A workaround is to make a small change manually in any of the fields of the style sheet and then undo the change. This seems to work and the changes will be saved.

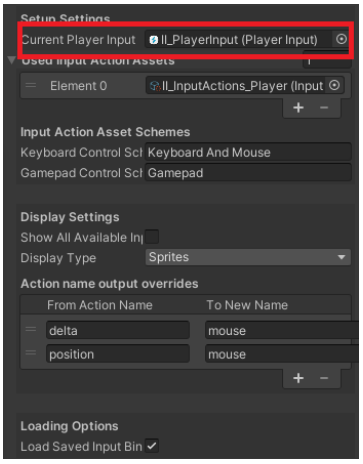
If we don't see any input icons



- The "**Used Input Action Assets**" list contains all Input Actions we use and we have updated the Default Style Sheet with all changes we made to the Input Action Asset(s).
- All our Input Action Assets have **control schemes for keyboard/gamepad** and their names are **equal to** the ones set in the Input Icons Setup or in the **Input Icons Manager** respectively.
- The **control schemes** of our Input Action Assets have **devices** added to them (see chapter "Adding Devices To Control Schemes")

Icons are shown, but don't get updated when a different device is used:

- Check section "**Keyboard And Gamepad Support**"
- Check the Input Icons Manager (in the folder "Assets/InputIcons/Resources/InputIcons/". The "**Current Player Input**" field has a Player Input which is active in the scene.
  Be aware that the Input Icons Manager will try to find an active Player Input component in the scene, whenever a scene is loaded and use it to handle changes of the active device. If you need to change to another Player Input within a scene, update the "Current Player Input" of the manager by calling InputIconsManagerSO.SetCurrentPlayerInput(PlayerInput input)



If you have gone through this guide and still experience problems, contact me at support@octacube-studios.com