Hyena Input Mapping

Remapping made easy

Quick Overview | Full Demo & ALS Integration | Docs | UE marketplace

Description

'Hyena Input Mapping' plugin for Unreal Engine 5 is a versatile and developer-friendly module designed to streamline input mapping and remapping in any project.

Can be set up for keyboard + mouse only, gamepad only, or **support both seamlessly**.

Developed in C++, this plugin is **designed to be used in Blueprints**, ensuring accessibility for all developers, regardless of their programming expertise.

Designed to use **fully customizable widgets** and icons, the plugin allows developers to create tailored input interfaces that fit any unique aesthetic.

Core concepts

(UE5) Enhanced Input Subsystem
- Input Mapping Context
- Input Action

(Plugin) Hyena Input Mapping
- HyenaIM_Box
- HyenaIM_PDA_Keys
- HyenaInputKeySelector

If you are not familiar with the first, please refer to the official documentation.

This document will try to cover the second.

Note: After uploading to the marketplace, the "Plugin Content" folder has been moved to:

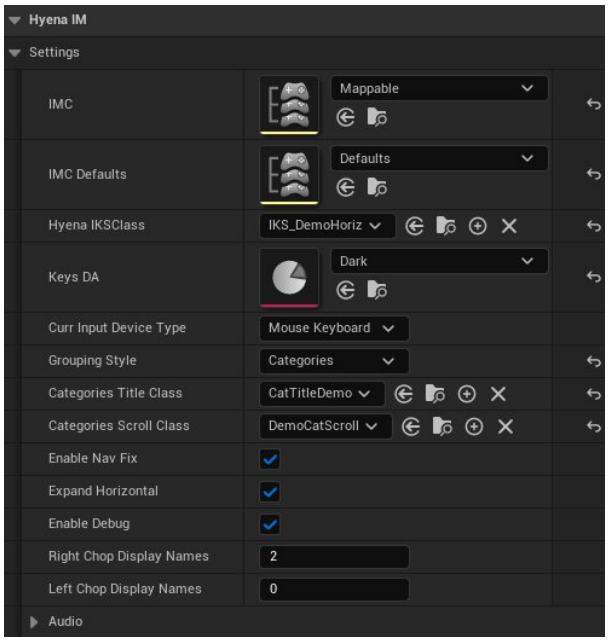
Engine / Plugins / HyenaInputMapping

Developer note: this is a code plugin, not a UI design course.
You can expect this **to work smoothly**, not necessarily to look polished. The demo widgets and menus are provided as extras, to serve as guide and as functionality showcase.

Let's start with an overview of the 3 core classes.

HyenaIM_Box

Re-Mapping box. Inherits from ScrollBox.



Hyena IM

Settings

IMC	Mappable
IMC Defaults	Defaults
Hyena IKSClass	IKS_DemoHoriz
Keys DA	Dark
Curr Input Device Type	Mouse Keyboard
Grouping Style	Categories
Categories Title Class	CatTitleDemo
Categories Scroll Class	DemoCatScroll
Enable Nav Fix	✓
Expand Horizontal	✓
Enable Debug	✓
Right Chop Display Names	2
Left Chop Display Names	0

Audio

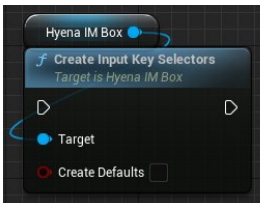
Just add it to any settings menu in the correct widget, page, tab, or whatever you're using.

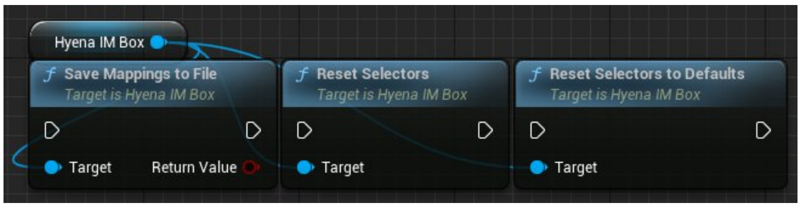
It's as drag-and-drop as possible. There are some variables that you need to set though.
The setup video, may cover this better.

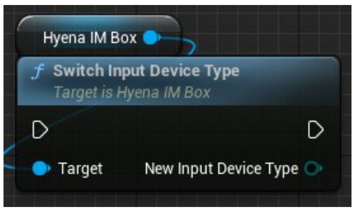
"**GroupingStyle**" can be:

- "None" - Simple setup, just a list of HyenaIKS's
- "Categories" - Little more complex, adds 2 extra widgets, one for category titles, and the other for category scrolls

BP exposed functions - hovering the nodes will give you an explanation of each one



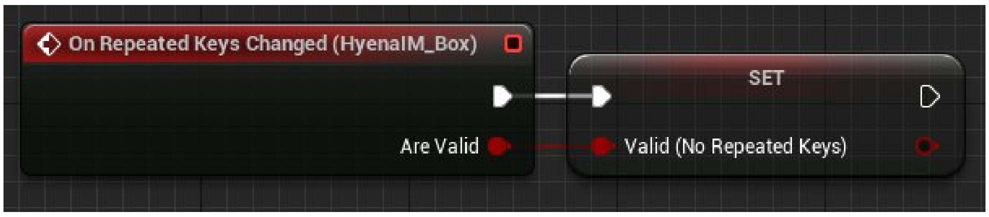




*First one expected to run most likely on Construct   *last one only if supporting both keyboard and gamepad

Repeated keys event - Can be used for feedback, logic, or simply ignored. Only fires on state change. You could block the player from saving if there are repeated keys.

Only fires at BeginPlay if there are repeated keys, which is unlikely.



*Note on design choice: I decided to go with allowing repeated keys rather than "removing" the other action that used the same key as it's safer and cleaner. I wouldn't like to bloat the plugin with options. Could be added by v2.0 if I have a lot of requests. If you want to do it yourself, should be easy to implement with C++ with a couple of extra functions, another struc like the QuestionMark in the PDA_Keys and a fallback key for the "empty key", though.

HyenaIM_PDA_Keys

Parent class mostly for “key + texture” TMaps. Inherits from PrimaryDataAsset.

It will decide what keys are allowed. If p.e. [Space Bar] key is allowed, we will, of course, need a texture for it.

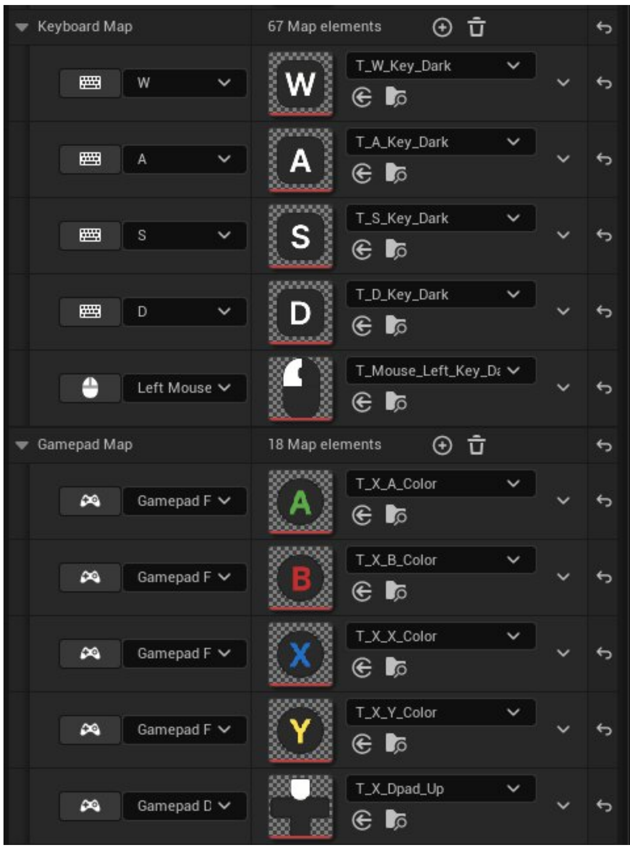
- “Question Mark” it’s the icon that will be used while “Selecting a new key”.
  Can be whatever, not necessarily a “?”. *Used “Global Menu” key as fallback

Using 2 TMaps to force non-repeated keys here as it can (and will) get messy with arrays.

*You’ll see the keyboard defaults bring ”WASD” keys first (intentionally). If you plan to move with those, you probably want to remove them straight away. QoL thing, is faster to remove than to add.

Dark DA demo

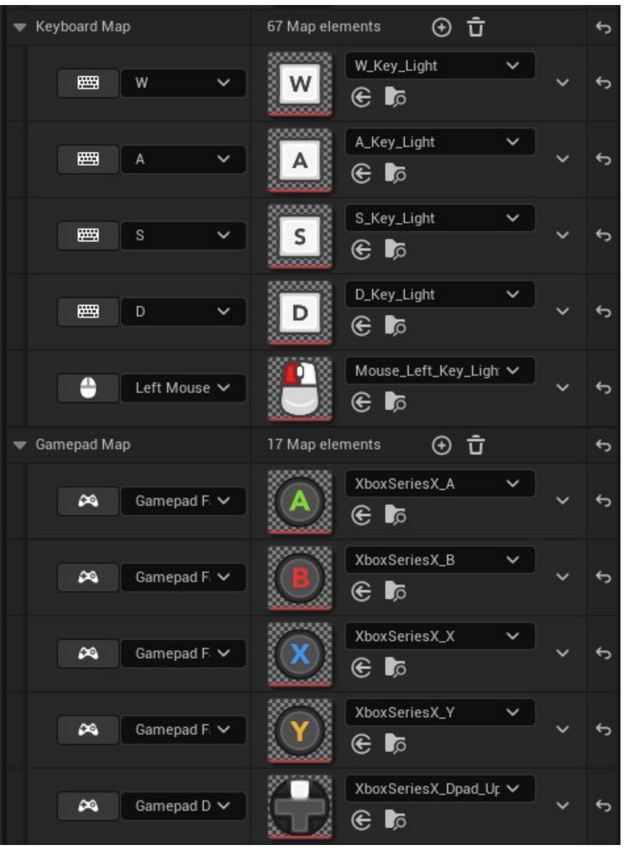
Icons: https://juliocacko.itch.io/free-input-prompts



Keyboard Map	67 Map elements
W	T_W_Key_Dark
A	T_A_Key_Dark
S	T_S_Key_Dark
D	T_D_Key_Dark
Left Mouse	T_Mouse_Left_Key_D
Gamepad Map	**18 Map elements**
Gamepad F	T_X_A_Color
Gamepad F	T_X_B_Color
Gamepad F	T_X_X_Color
Gamepad F	T_X_Y_Color
Gamepad D	T_X_Dpad_Up

Light DA demo

Icons: https://thoseawesomeguys.com/prompts/



Keyboard Map	67 Map elements
W	W_Key_Light
A	A_Key_Light
S	S_Key_Light
D	D_Key_Light
Left Mouse	Mouse_Left_Key_Ligh
Gamepad Map	**17 Map elements**
Gamepad F	XboxSeriesX_A
Gamepad F	XboxSeriesX_B
Gamepad F	XboxSeriesX_X
Gamepad F	XboxSeriesX_Y
Gamepad D	XboxSeriesX_Dpad_Up

*Yes, Light DA gamepad icons are not really “light”, those are just examples for the demo.

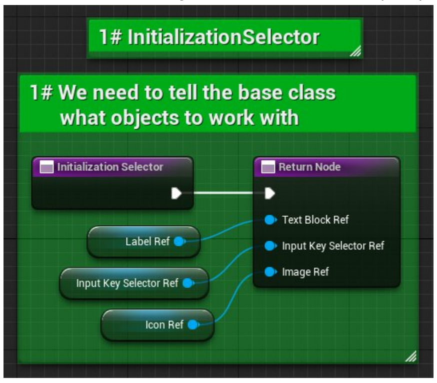
HyenaInputKeySelector

Extension of the UE “Input Key Selector”. Inherits from UserWidget though.

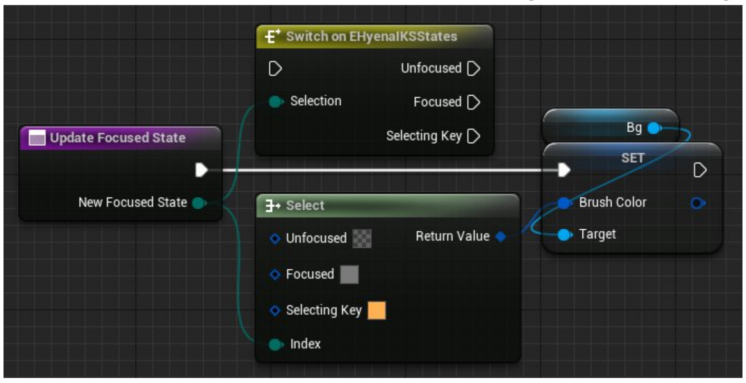
A lot of stuff is happening under the hood, but we don’t really care in BP. (/o_o)/#

Just 2 functions to consider, please refer to the demos to see examples.

We’ll need 3 “objects” for it to run properly



And we have a function for feedback and logic on focus change



You can customize the widget however you wish. Just update the InitializationSelector.
If you’re wondering why, try disconnecting, and you’ll quickly realize why **all 3** are necessary.

So now we have covered the 3 main classes.

Extra classes

Not as important as the others, but still add certain functionality or QoL.

HyenaIM_CatTitle

Category Title. Inherits from UserWidget.

Should contain at least one TextBlock to display, well, the title.

HyenaIM_CatScroll

Category Scroll. Inherits from ScrollBox.

No logic here. Created to allow A/B-ing different styles, borders, paddings, transforms, etc.

Extra notes:

Why 3 IMC?

That's a good question. You can skip this. But if you're curious, long explanation incoming:

Why 2? I decided to separate the 'Mappable' and 'UnMappable', pretty straightforward:

- No need to save UnMappable keys as those won't be ever remapped.
- Re-mapping actions with modifiers, well, no documentation, good luck trying...
- Easier to work with, I would personally do it anyway.

So why the 3rd 'Defaults'? It's supposed to be a duplicate of the 'Mappable' anyway.

Idea 1: Create a 2nd object like 'Mappable_Defaults.save' (first iteration was like this)

- Biggest issue: in editor only, IMC gets remapped to a somewhat half-persistent state. Elaborating a little: after restarting the editor remapping goes back to before remapping. Yet, while the project is open, the IMC gets "rewritten" with the new mappings. That can be inconvenient while setting them up as there's no way to know what the defaults were without restarting the editor.
- (unlikely) "Bad" problem: if the 'Mappable_Defaults.save' gets removed at runtime both in editor and in the packaged game, I don't think there's any way to get the pre-mappings anymore without a full editor/game restart.

Idea 2: Save a copy in the game instance

- We're coupling other classes (bad) when it can be avoided.
- An interface could be ok, but then again, won't simplify the process for newbies.

So, after a lot of thought and end up going for a level 1 solution.

It's not like we're changing the 'Mappable' IMC every minute, so after modification, just duplicate this 'Mappable' to keep an always-fresh IMC that (1) will never be remapped and (2) is always accessible, both in editor and in packaged game. Can't go wrong with this.

Takes a few seconds to update the 'Defaults' after changing the 'Mappable', example steps:

1. Duplicate the 'Mappable' IMC and name it 'NewDefaults'
2. Delete (old) 'Defaults'. When prompted, pick 'NewDefaults' at "Replace references"

*remember to update redirectors if you have naming issues

Done!