TNT Input Engine

A Godot 3.x Unified Input System

Contact: gregbug@gmail.com

What is it for? Why was it created?

TNT input engine is an addon made for the Godot 3.x game engine.

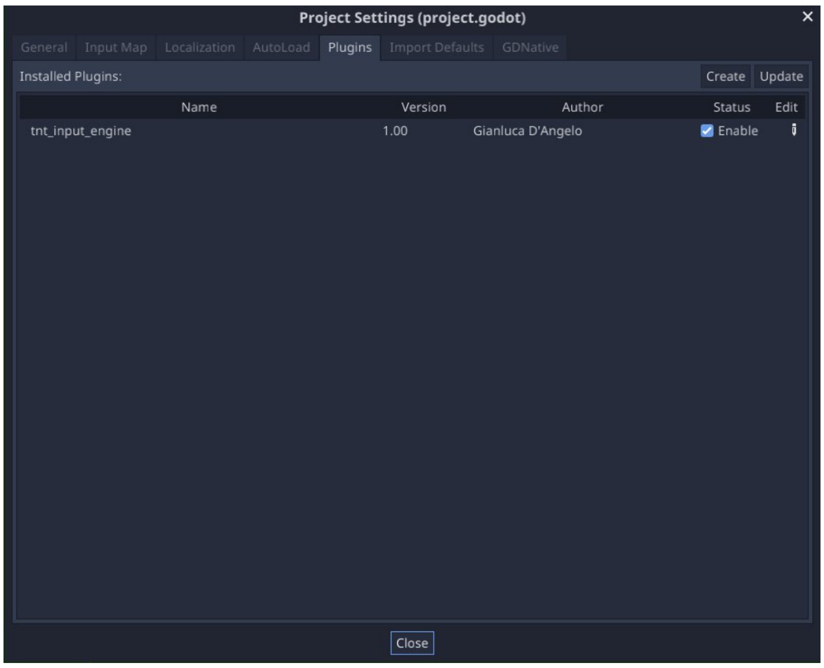
The main reason why it was created is to simplify and unify the writing of input control code allowing you to write code only once and simultaneously manage keyboard input from virtual touchscreen joystick and real joystick using the standard API of Godot.

It was designed from the outset to not impact performance and in fact the system uses the Godot input engine without adding any overload; the only additional code is for the management of the touchscreen for the virtual joypad and the code is really light and fast, the input management is instead managed by the engine itself.

To make the programmer's life easier, a simplified and automated mapping system has also been added for all those input devices that are not natively recognized by Godot.

Installation

The installation of the plugin follows the rules of Godot; copy or import the plugin under the addons folder of your project and activate it from the Godot Project-> Project setting-> Plugins menu and enable “tnt_inpuit_engine”.

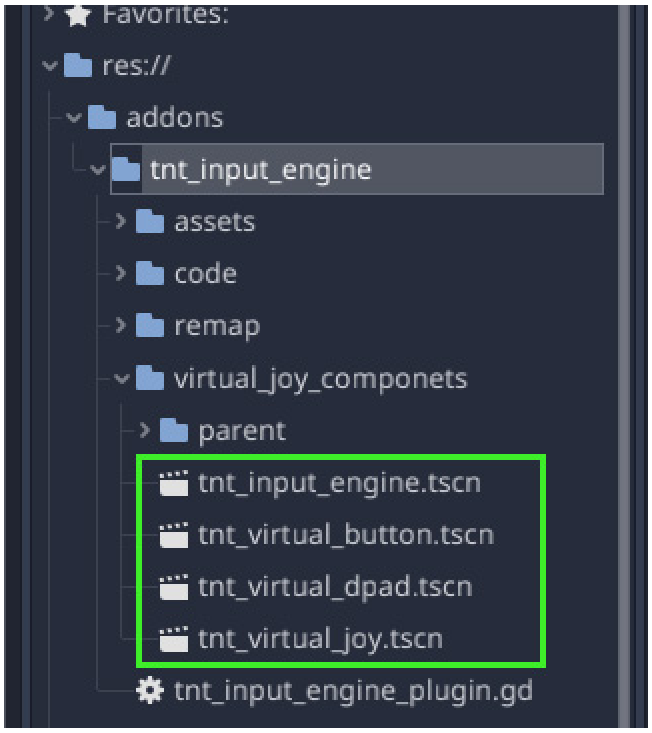


Name	Version	Author	Status	Edit
tnt_input_engine	1.00	Gianluca D'Angelo	✓ Enable	

Ok, now you can use the plugin!

PS: When developing on desktop PC remember to enable mouse touchscreen emulation. (Godot Project-> Project setting-> Input devices-> Pointing and enable “emulate touch from mouse”).

Now if you look under the addons folder, check the contents of "virtual_joy_components" folder, here you will find four items that will be need to manage the input.



Favorites:
res://
addons
tnt_input_engine
assets
code
remap
virtual_joy_componets
parent
tnt_input_engine.tscn
tnt_virtual_button.tscn
tnt_virtual_dpad.tscn
tnt_virtual_joy.tscn
tnt_input_engine_plugin.gd

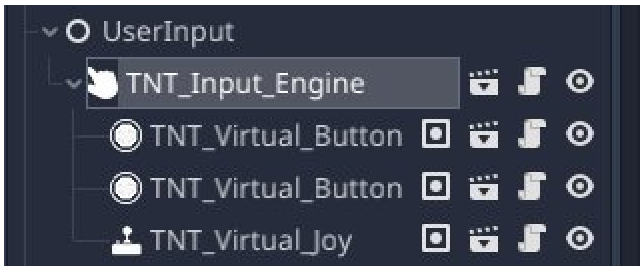
tnt_input_engine.tscn it’s the parent node that will be used to manage the global properties of the system.

tnt_virtual_button.tscn it’s the node that manages the input buttons.

tnt_virtual_dpad.tscn it’s the node that manages the digital “directional” pad. (virtual and real)

tnt_virtual_joy.tscn it’s the node that manages the analog joystick (virtual and real)

The composition of these nodes will therefore allow you to compose your touchscreen joypad which will behave exactly like the real joypad when it is not present sharing the same control code (standard Godot code, therefore you will not have to learn new APIs)



UserInput
TNT_Input_Engine
TNT_Virtual_Button
TNT_Virtual_Button
TNT_Virtual_Joy

It is important to note that:

tnt_virtual_button, tnt_virtual_dpad, tnt_virtual_joy

THEY MUST NECESSARILY BE "CHILDREN" of tnt_input_engine.

Now to speed up this kind of quick course, open the Demo_1 scene under the demo folder and start it.

Hopefully you can move the character with the keyboard, with the virtual touchscreen dpad or with the joypad! And the great part is that the control code is always this:

```
func _physics_process(_delta: float) -> void:
	if  Input.is_action_pressed(TNTInputEngine.ACTION_JOY_LEFT):
			player_anim.flip_h = true
			player_anim.play("right")
			player_xy.x = -1
	elif Input.is_action_pressed(TNTInputEngine.ACTION_JOY_RIGHT):
			player_anim.flip_h = false
			player_anim.play("right")
			player_xy.x = 1
	elif Input.is_action_just_released(TNTInputEngine.ACTION_JOY_LEFT) ||
Input.is_action_just_released(TNTInputEngine.ACTION_JOY_RIGHT):
			player_xy.x = 0
			player_anim.stop()
			player_anim.frame = 1
	if Input.is_action_pressed(TNTInputEngine.ACTION_JOY_UP):
			player_xy.y = -1
			if player_xy.x == 0:
				player_anim.play("up")
	elif Input.is_action_pressed(TNTInputEngine.ACTION_JOY_DOWN):
			player_xy.y = 1
			if player_xy.x == 0:
				player_anim.play("down")
```

What I recommend is to try and modify demo_1 and demo_2.

The description of the TNTInputEngine global class and the nodes to be used to manage the input will now follow.

TNTInputEngine Class

TNTInputEngine is the global class introduced by the plugin and which therefore lives throughout the execution of your project.

This class defines constants and utility functions for use with the plugin.

The source code of the "TNTInputEngine" class is contained in the file res: //addons/tnt_input_engine/code/global_class/tnt_input_engine_global.gd

As in all the addon source code, all variables, constants, functions that begin with "_" are to be considered private and therefore not to be used outside the addon classes.

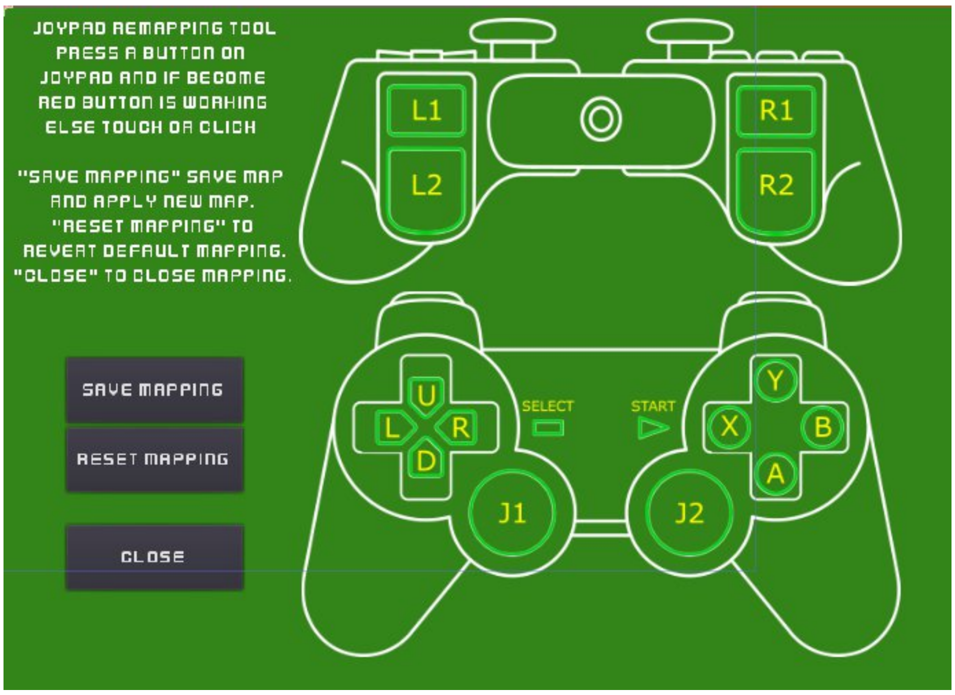
The basic principle for the correct use of this addon is that if you use it it would be better not to use the godot inputmap but to use only the addon for this purpose.

After this little preamble, let's analyze the features of the TNTInputEngine class:

func open_hw_joy_remap(device_id: int) -> void:

device_id = joypad id to remap.

Opens the scene (automatically adapts to landscape or portrait format) which allows you to visually remap the joypad specified by device_id:


JOYPAD REMAPPING TOOL
PRESS A BUTTON ON
JOYPAD AND IF BECOME
RED BUTTON IS WORKING
ELSE TOUCH OR CLICK

"SAVE MAPPING" SAVE MAP
AND APPLY NEW MAP.
"RESET MAPPING" TO
REVERT DEFAULT MAPPING.
"CLOSE" TO CLOSE MAPPING.

Once the mapping has been completed, the user can choose whether to save it and then be reused at any time. (for more info see Demo_1 scene HW_JoyInfo code res: //demo/shared/HW_JoyInfo.gd function update_joypad_info ())

func add_key_binding(key_scancode: int, action_name: String) -> void:

It allows you to connect the keyboard input to the virtual joypad and to the real joypad.

key_scancode: it's the Godot scancode which represents the key you want to use for example KEY_UP, KEY_1, KEY_A

action_name: it's the action you want to associate with the button. It is a user-defined string or a predefined string from TNTInputEgine (see Demo_1 code res: //demo/scenes/code/demo_1.gd):

```
const ACTION_JOY_A: String = "joy_a"
const ACTION_JOY_B: String = "joy_b"
const ACTION_JOY_Y: String = "joy_y"
const ACTION_JOY_X: String = "joy_x"
const ACTION_JOY_START: String = "joy_start"
const ACTION_JOY_SELECT: String = "joy_select"
const ACTION_JOY_L2: String = "joy_l2"
const ACTION_JOY_R2: String = "joy_r2"
const ACTION_JOY_L1: String = "joy_l1"
const ACTION_JOY_R1: String = "joy_r1"
const ACTION_JOY_UP: String = "joy_up"
const ACTION_JOY_LEFT: String = "joy_left"
const ACTION_JOY_DOWN: String = "joy_down"
const ACTION_JOY_RIGHT: String = "joy_right"
const ACTION_LEFTAXIS_UP: String = "axis_lefty-"
const ACTION_LEFTAXIS_RIGHT: String = "axis_leftx+"
const ACTION_LEFTAXIS_DOWN: String = "axis_lefty+"
const ACTION_LEFTAXIS_LEFT: String = "axis_leftx-"
const ACTION_RIGHTAXIS_UP: String = "axis_righty-"
const ACTION_RIGHTAXIS_RIGHT: String = "axis_rightx+"
const ACTION_RIGHTAXIS_DOWN: String = "axis_righty+"
const ACTION_RIGHTAXIS_LEFT: String = "axis_rightx-"
```

Tabella 1

func add_joy_binding(button: int, action: Array = ["", "", "", ""], dead_zone: float = 0.5) -> void:

It allows you to link components of the virtual joypad and the real joypad via code.

button: joy.DPAD, joy.JOY_LEFT, joy.JOY_RIGHT, joy.BTN_A, joy.BTN_B, joy.BTN_X, joy.BTN_Y, joy.SELECT, joy.START, joy.L1, joy.L2, joy.R1, joy.R2

action: action (defined by the user or table 1) to be connected to the component (for dpad, joy_left and joy_right there are 4 actions) if not defined, the default value is used.

joy.DPAD: ACTION_JOY_UP, ACTION_JOY_RIGHT, ACTION_JOY_DOWN, ACTION_JOY_LEFT

joy.JOY_LEFT: ACTION_LEFTAXIS_UP, ACTION_LEFTAXIS_RIGHT, ACTION_LEFTAXIS_DOWN, ACTION_LEFTAXIS_LEFT

joy.JOY_RIGHT: ACTION_RIGHTAXIS_UP, ACTION_RIGHTAXIS_RIGHT, ACTION_RIGHTAXIS_DOWN, ACTION_RIGHTAXIS_LEFT

And so on...

dead_zone: value below which the joypad does not move. (0.5 if not defined)

func delete_dbfile(joy_guid: String) -> int:

Delete the joypad mapping file with GUID if it exists **joy_guid**

Release OK if cancellation was successful.

func user_dbfile_joy_exist(joy_guid: String) -> bool:

release true if the joypad **joy_guid** mapping file exists otherwise release false

func load_and_apply_joy_db(device_id: int) -> int:

Read and apply joypad mapping with id **device_id** if it exists.

Release OK if everything is OK otherwise release ERR_FILE_NOT_FOUND.

func set_active_joy_id(id: int) -> void:

Sets the active joypad with index **id**

func get_active_joy_id() -> int:

Release the id of the currently active joypad

func get_connected_devices_count() -> int:

Releases the number of joypads currently connected

func get_device_id(device_index: int) -> int:

Release the joypad id with **device_index** index

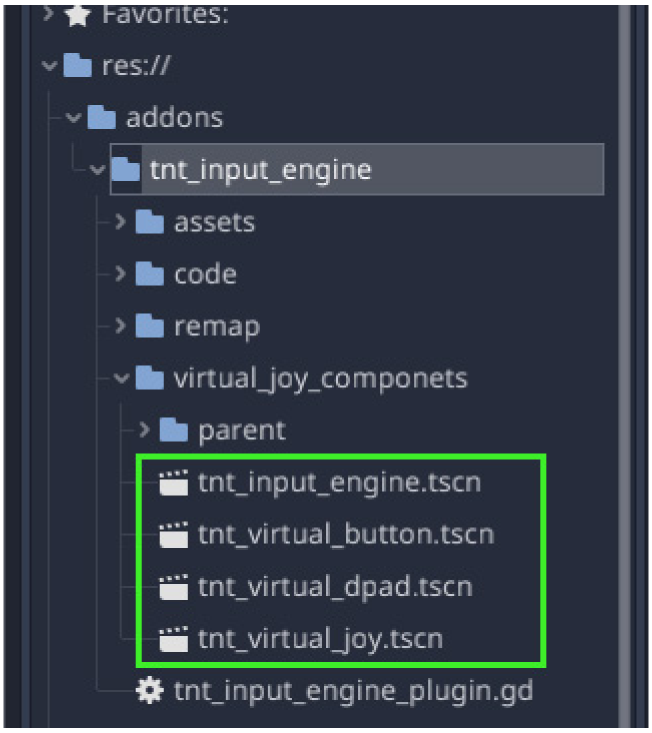
func get_device_name(device_id: int) -> String:

Release the name of the joypad with id **device_id**

func get_device_GUID(device_id: int) -> String:

Release the joypad GUID with id **device_id**

Nodi TNTInputEngine


Favorites:
res://
addons
tnt_input_engine
assets
code
remap
virtual_joy_componets
parent
tnt_input_engine.tscn
tnt_virtual_button.tscn
tnt_virtual_dpad.tscn
tnt_virtual_joy.tscn
tnt_input_engine_plugin.gd

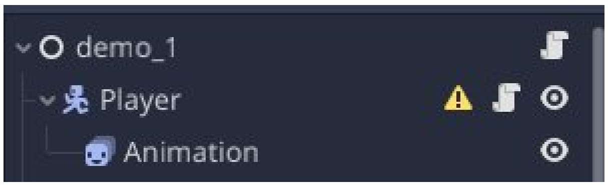
As already mentioned above the nodes that make up the whole system are these four and they are visual nodes they allow you to build a dynamic and efficient input system obviously you will find these nodes under the folder Addons-> tnt_input_engine-> virtual_joy_components, the most important is the first tnt_input_engine which is the parent node and which will check all sub-nodes.

In the next paragraph I will explain how to initialize and build the first virtual joystick obviously I remind you that it is strongly recommended to study and check the demos attached to the plugin specifically demo_1 and demo_2.

To insert nodes in your project, just drag them from the folder Addons within your scene.

How to get started

To start we open the demo_1 now if we look at the left side on the hierarchy of the scene we will find the following situation:



We click on the code symbol of the demo_1 node and observe the following code portion:

```
func _ready() -> void:
	################################################
	# adding keyboard actions binding via code.... #
	################################################
	# FOR SCENE 1
	TNTInputEngine.add_key_binding(KEY_UP, TNTInputEngine.ACTION_JOY_UP)
	TNTInputEngine.add_key_binding(KEY_DOWN, TNTInputEngine.ACTION_JOY_DOWN)
	TNTInputEngine.add_key_binding(KEY_LEFT, TNTInputEngine.ACTION_JOY_LEFT)
	TNTInputEngine.add_key_binding(KEY_RIGHT, TNTInputEngine.ACTION_JOY_RIGHT)
	# FOR SCENE 2
	TNTInputEngine.add_key_binding(KEY_1, TNTInputEngine.ACTION_JOY_A)
	TNTInputEngine.add_key_binding(KEY_2, TNTInputEngine.ACTION_JOY_B)
	TNTInputEngine.add_key_binding(KEY_UP, TNTInputEngine.ACTION_LEFTAXIS_UP)
	TNTInputEngine.add_key_binding(KEY_DOWN, TNTInputEngine.ACTION_LEFTAXIS_DOWN)
	TNTInputEngine.add_key_binding(KEY_LEFT, TNTInputEngine.ACTION_LEFTAXIS_LEFT)
	TNTInputEngine.add_key_binding(KEY_RIGHT, TNTInputEngine.ACTION_LEFTAXIS_RIGHT)

	#######################################################
	# adding VPAD/JOYPAD for scene 2 binding via code.... #
	#######################################################
	TNTInputEngine.add_joy_binding(TNTInputEngine.joy.JOY_LEFT)
	TNTInputEngine.add_joy_binding(TNTInputEngine.joy.BTN_A)
	TNTInputEngine.add_joy_binding(TNTInputEngine.joy.BTN_B)
```

In this first part of the code we are going to link the keyboard input with the input of the virtual joystick and of the real joystick as already mentioned it is preferable to insert all this through code but in reality it is also possible to do it through the Godot user interface.

The second part of the code we are going to connect for scene 2 a virtual left analog joypad and two buttons (button a and button B) since no other parameters are specified, the actions will be the standard ones defined by the TNT library as specified in table 1.

TNTInputEngine.add_key_binding (KEY_UP, TNTInputEngine.ACTION_JOY_UP)

With this line of code, we then link the action TNTInputEngine.ACTION_JOY_UP to the "up" key and which will be issued each time the "up" key is pressed.

First Virtual Joypad

We open Demo_2, leave out the other nodes and carefully observe those under the user input node.

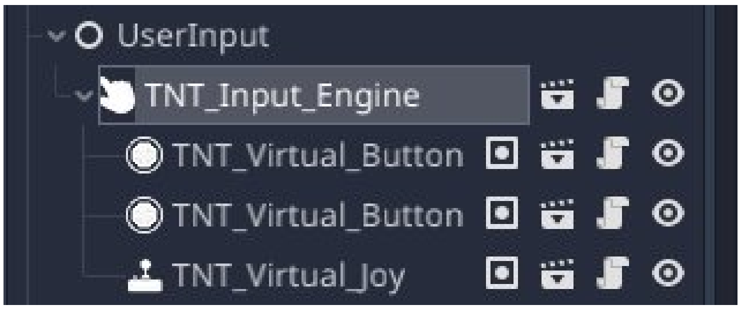
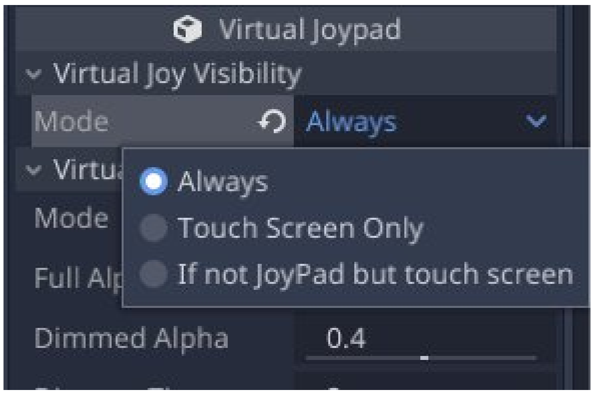

UserInput
TNT_Input_Engine
TNT_Virtual_Button
TNT_Virtual_Button
TNT_Virtual_Joy

Figura 1.

As a matter of order, I recommend always inserting a primary node called "userinput" under which we will build our joypad. To insert the TNT_input_engine node just drag it from the addon folder under the user input node. This node is the main node of our joypad and must necessarily be the parent of all the other sub-nodes and will allow us to control the behavior of the whole Joypad; below we analyze the main properties:


Virtual Joypad
Virtual Joy Visibility
Mode: Always
Always
Touch Screen Only
If not JoyPad but touch screen
Dimmed Alpha: 0.4

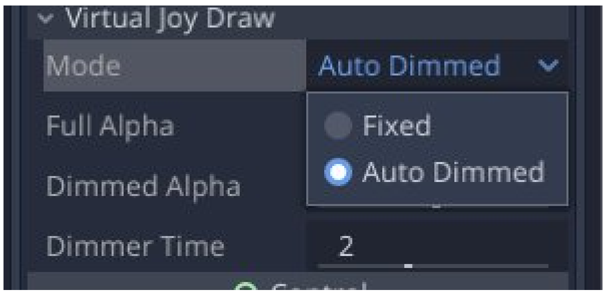
Virtual Joy Visibility:

Always: the virtual joypad is ALWAYS visible.

Touch Screen Only: the virtual joypad is visible ONLY if the device has a touch screen.

If Not joyPad but touch screen: if a real joypad is present the virtual one will not be visible, if the real joypad is not present the virtual joypad will be visible only if the touch screen is present.

PS: Please note all the input methods will always be functional if present (except the virtual joypad which is functional only if visible).

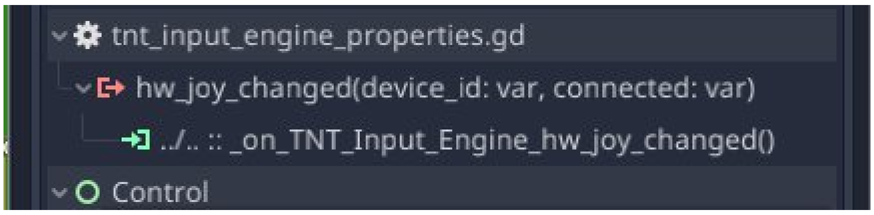

Virtual Joy Draw
Mode: Auto Dimmed
Full Alpha
Dimmed Alpha
Dimmer Time: 2
Fixed
Auto Dimmed

Virtual Joy Draw:

Fixed: The components of the virtual joypad will always be drawn with an Alpha equal to 1 therefore as originally drawn.

Auto Dimmed: the components of the virtual joypad will be drawn with an Alpha transparency equal to Full Alpha and after the time (seconds) defined in dimmer time with an Alpha equal to the value of Dimmed Alpha; thus simulating a timed transparency.

TNT_input_engine node event:



```
tnt_input_engine_properties.gd
  hw_joy_changed(device_id: var, connected: var)
    ../.. :: _on_TNT_Input_Engine_hw_joy_changed()
Control
```

In the TNT_input_engine node a very important event is defined which is executed every time a hardware joypad is inserted or disconnected in the PC or in the device.

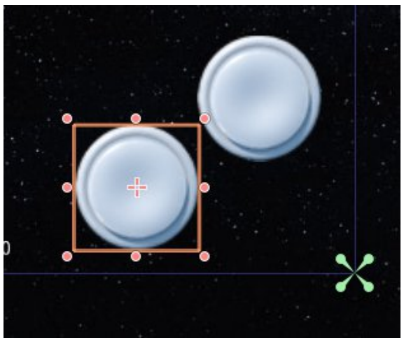
This allows us to obviously understand when a joypad is inserted or connected and eventually make the appropriate decisions.

The event brings with it two very important parameters: the device ID which contains the joypad ID and the connected information to understand if the joypad has been inserted (true) or disconnected (false).

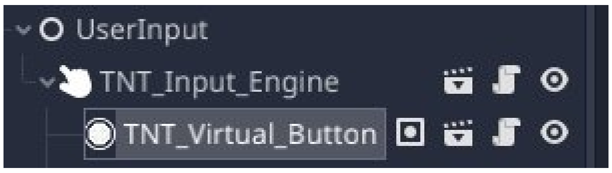
See Demo_1 for a concrete example of use.

Now that we have defined the general behavior of our virtual joypad, let's add buttons and an analog joypad for the movement of the spaceship as illustrated in demo_2.

TNT_Virtual_Button

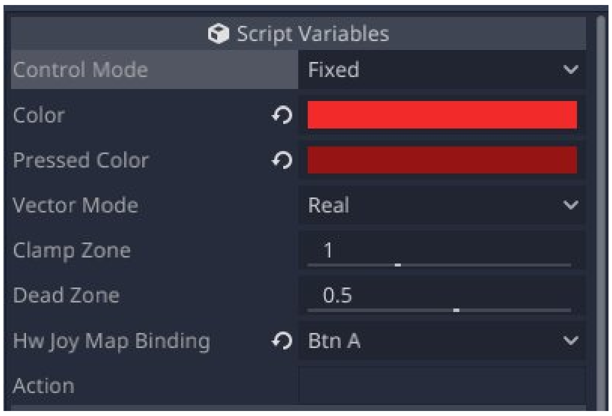


The tnt_virtual_button node was created to manage the "buttons" or "buttons" of either the keyboard, virtual pad or real joypad. If we can create as many as we want obviously if mapped to real joypads we are limited by the real number of buttons on the joypad. Both the keyboard and the joypad can be "tied" to each of them. Each button can have its own behavior and color. To detect the pressure, the standard Godot input will be followed.



```
UserInput
  TNT_Input_Engine
    TNT_Virtual_Button
```

As already said several times the tnt_virtual_button node must be inserted as a child below the TNT_input_engine node otherwise it will not work.



Script Variables

Control Mode	Fixed
Color	🟥
Pressed Color	🟥
Vector Mode	Real
Clamp Zone	1
Dead Zone	0.5
Hw Joy Map Binding	Btn A
Action	

The specific properties of the tnt_virtual_button are:

Control Mode:

Fixed: The button or component will always be fixed where it was placed in the design phase.

Dynamic:The button or component will move to the touch pressure point.

Follow: The Button or component follows the pressure movement of the touch.

Color: Specifies the predominant color of the Button or component when not pressed.

Pressed Color: Specifies the predominant color of the Button or component when pressed.

Vector Mode, Clamp zone, Dead zone: They are not relevant to the tnt_virtual_button component.

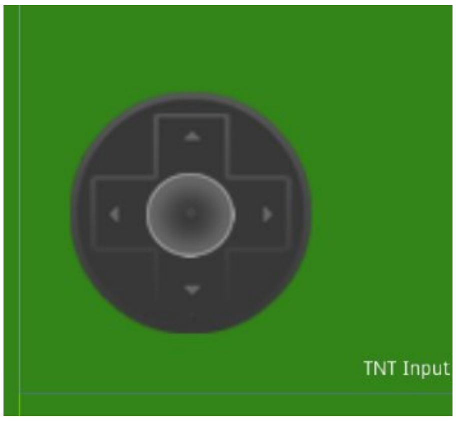
Hw Joy Map Binding: If different from "none" it is used to "map" the button corresponding to the real joypad. Possible values are: NONE, BTN_A, BTN_B, BTN_X, BTN_Y, SELECT, START, L1, L2, R1, R2.

Action: It is a string that represents the action that will be issued when the button or component is pressed and that must be intercepted by the code to perform the action. If the string is left empty, the standard TNTInputEngine action will be used as in table 1. For example, if **Hw Joy Map Binding** is assigned the value BTN_A and **Action** is empty, the value will be assigned: TNTInputEngine.ACTION_JOY_A

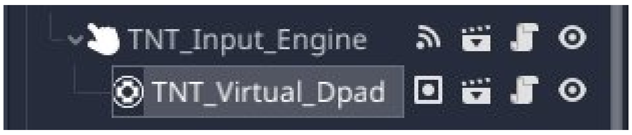
So to then detect the pressure it will base: if Input.is_action_pressed (TNTInputEngine.ACTION_JOY_A)

It does not matter if it comes from a keyboard, virtual joystick or real joystick, the event will be detected. (For keyboard entry, remember that it must necessarily be enabled via code as already seen above TNTInputEngine.add_key_binding (KEY_A, TNTInputEngine.ACTION_JOY_A))

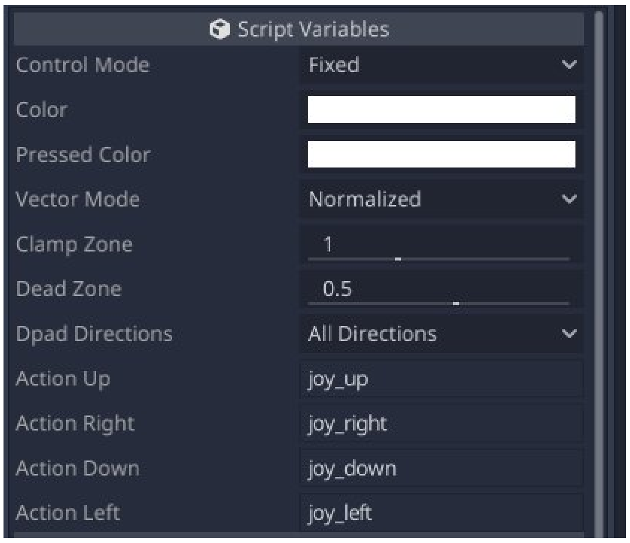
TNT_VIRTUAL_DPAD



The TNT_virtual_dpad node was created to simulate the digital directional pad of the joypad; it can be set to manage movements only horizontal, only vertical or both at the same time. Here too we can create as many as we want obviously if mapped to real joypads we are limited by the real number of DPADs of the joypad. Both the keyboard and the joypad can be "tied" to each of them. Each button can have its own behavior and color. To detect the pressure, the standard Godot input will be followed.



As already said several times the TNT_virtual_dpad node must be inserted as a child below the TNT_input_engine node otherwise it will not work.



Script Variables

Control Mode	Fixed
Color	⬜
Pressed Color	⬜
Vector Mode	Normalized
Clamp Zone	1
Dead Zone	0.5
Dpad Directions	All Directions
Action Up	joy_up
Action Right	joy_right
Action Down	joy_down
Action Left	joy_left

The specific properties of the tnt_virtual_dpad (in addition to those common to the tnt_virtual_button) are::

Dpad Directions:

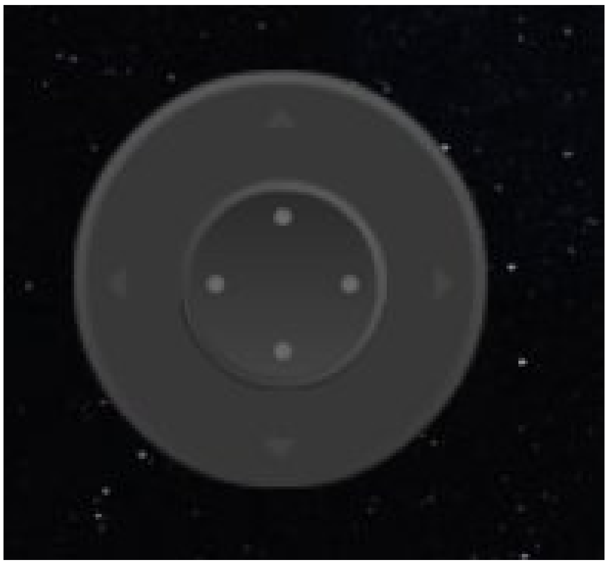
All Direcrtions: the dpad can move in all directions.

Left Right: the dpad can only move horizontally.

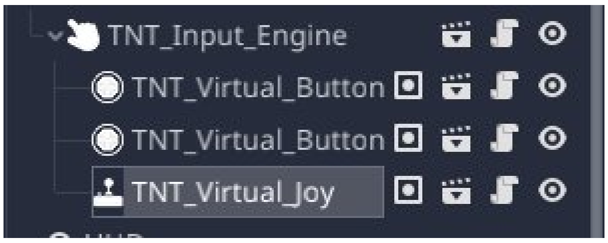
Up Down: the dpad will only be able to move vertically.

Action up, Action Right, Action down, Action Up: It is a string that represents the action that will be used when the button or component is pressed and that must be intercepted by the code to perform the action. If the string is left empty, the standard TNTInputEngine action will be used as per table 1. The standard action strings are set by default.

TNT_Virtual_Joy

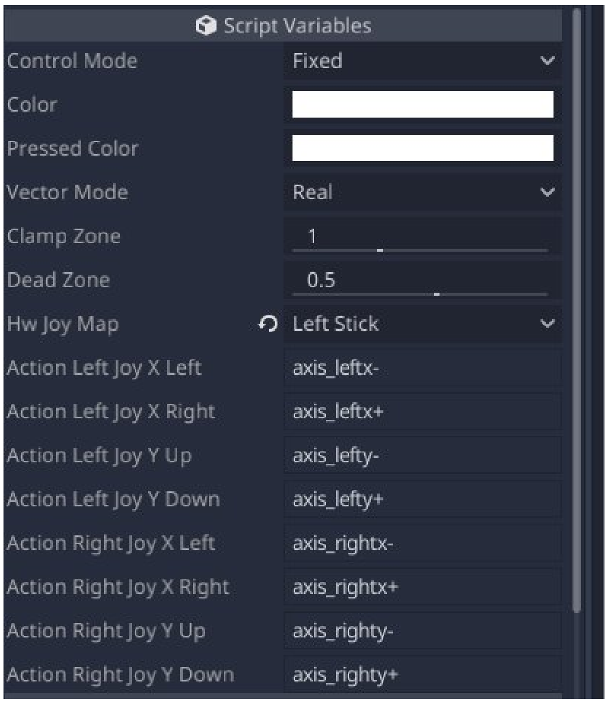


the TNT_virtual_joy node was created to simulate the analog joystick of the joypad; Here too we can create as many as we want obviously if mapped to real joypads we are limited by the real number of joypad joysticks (typically 2). Both the keyboard and the joypad can be "tied" to each of them. Each button can have its own behavior and color. To detect its movement, the standard Godot input will be followed.



```
TNT_Input_Engine
    TNT_Virtual_Button
    TNT_Virtual_Button
    TNT_Virtual_Joy
```

As already said several times the TNT_virtual_joy node must be inserted as a child below the TNT_input_engine node otherwise it will not work.



Script Variables

Control Mode	Fixed
Color	⬜
Pressed Color	⬜
Vector Mode	Real
Clamp Zone	1
Dead Zone	0.5
Hw Joy Map	Left Stick
Action Left Joy X Left	axis_leftx-
Action Left Joy X Right	axis_leftx+
Action Left Joy Y Up	axis_lefty-
Action Left Joy Y Down	axis_lefty+
Action Right Joy X Left	axis_rightx-
Action Right Joy X Right	axis_rightx+
Action Right Joy Y Up	axis_righty-
Action Right Joy Y Down	axis_righty+

The specific properties of tnt_virtual_joy (in addition to those common to tnt_virtual_button and tnt_virtual_dpad) are:

Vector Mode:

Real: real value vector from 0 to 1

Normalized: normalized vector

Clamp Zone:0..2 maximum possible distance from the center of the joystick.

Dead Zone: value below which no movements are detected.

Hw joy Map: Joystick with which the mapping takes place (none, left or right).

Action left/right: It is a string that represents the action that will be issued when the joystick moves and that must be intercepted by the code to perform the action. If the string is left empty, the standard TNTInputEngine action will be used as per table 1. The standard action strings are set by default.

PS: When the components of the virtual joypad are not visible, no code is executed so as not to impact in any way on the performances.

After this brief introduction, what I would like to recommend is to check and deepen demo 1 and 2 trying to experiment by changing the parameters of the nodes.

I hope this addon can be useful to you as it was for me if you like it, use it and want to help me you can support me in the development in the correction and optimization by contacting me directly to my e-mail or if you like you can offer me a beer through a small PayPal donation directly on my email. (Gianluca D'Angelo gregbug@gmail.com)

You are free to use for any of your projects, whether commercial or free.

Thank you, Gianluca D'Angelo (GregBUG)

L'Aquila, Italy.