



hininput

v1.6.0

Learn

[download](#) | [install](#) | [documentation](#)

hininput is a simple gamepad manager for Unity - a [hiloqo](#) project from [henri](#)

Hi, my name is Henri and I made hininput.

I wrote this short guide to teach you a few features of the plugin.

Part 1 : The basics

How to get the state of a button

One of the first things you're going to do with hininput is probably determining whether a given button is pressed or not. You can do so by using any of the following lines :

```
hininput.gamepad[0].A.pressed  
hininput.gamepad[0].leftTrigger.pressed  
hininput.gamepad[0].rightStickClick.pressed
```

As you can see, you start by calling hininput, then you select the gamepad you would like to get, then a button. You finally call the *pressed* property of that button, which returns a boolean value (true if the button is pressed, false otherwise).

A good way to attach an action to a button is simply to check if it is pressed in the Update method of one of your scripts. Something like this :

```
void Update () {  
    if (hininput.gamepad[0].A.pressed) {  
        // Do something  
    }  
}
```

justPressed

Let's say you are making a platformer game, and you want your character to jump with A. Maybe you're going to write something like this :

```
void Update () {  
    if (hinput.gamepad[0].A.pressed) {  
        // Jump  
        // That doesn't work as expected...  
    }  
}
```

If you implement it that way, pressing A will make your character jump *every single frame*.

Instead of the *pressed* property of the A button, you might want to use *justPressed*. It is similar to *pressed*, but it returns true only if A has started being pressed this exact frame. Here is what it would look like :

```
void Update () {  
    if (hinput.gamepad[0].A.justPressed) {  
        // Jump  
        // This is much better !  
    }  
}
```

Stick directions

The term “button” as I’ve used it earlier can refer to any gamepad button in `hinput`, ranging from actual buttons to triggers, bumpers and stick clicks (They’re all members of the **hPressable** class in `hinput`).

But that’s not all. I mapped 8 directions on the left stick, the right stick, and the D-pad of each gamepad, to act as virtual buttons. This means that they are also instances **hPressable**, and you can use them exactly the way you did the rest of the buttons :

```
hinput.gamepad[0].leftStick.left.pressed  
hinput.gamepad[6].rightStick.down.pressed  
hinput.anyGamepad.dPad.upRight.pressed
```

For instance, a simple character controller could look like this :

```
void Update () {  
    if (hinput.gamepad[0].leftStick.left.justPressed) {  
        transform.position += Vector3.left;  
    }  
  
    if (hinput.gamepad[0].leftStick.right.justPressed) {  
        transform.position += Vector3.right;  
    }  
  
    if (hinput.gamepad[0].leftStick.up.justPressed) {  
        transform.position += Vector3.forward;  
    }  
  
    if (hinput.gamepad[0].leftStick.down.justPressed) {  
        transform.position += Vector3.back;  
    }  
}
```

More on simple controllers a few pages down...

Part 2 : Advanced features

Other button properties

There are many other button properties that you can use. I won't cover them all in detail here, but here are a few examples :

```
if (hinput.anyGamepad.A.released) {  
    // Fall  
}  
  
if (hinput.anyGamepad.rightStickClick.justReleased) {  
    // Crouch  
}  
  
if (hinput.gamepad[7].X.doublePress) {  
    // Tumble  
}  
  
if (hinput.gamepad[4].Y.longPress) {  
    // Heal  
}
```

Feel free to explore the different possibilities, and have a look at the full documentation for more !



Sticks and D-Pads

If using stick directions as buttons isn't enough for your project, you can have a look at the sticks and D-pads themselves (they're represented by the class **hStick**).

The basic way of getting a stick's state is called *position* :

```
hinput.gamepad[0].leftStick.position  
hinput.gamepad[6].rightStick.position  
hinput.anyGamepad.dPad.position
```

This command will return the X and Y coordinates of the stick, as a Vector2. Something like (-0.78, 0.42). Stick positions are always within the unit circle, i.e. within 1 unit of the position (0, 0).

Here are examples of a few other features of this class :

```
if (hinput.gamepad[0].dPad.vertical < 0) {  
    // Look down  
}  
  
if (hinput.gamepad[6].leftStick.distance > 0.8f) {  
    // Run  
}  
  
if (hinput.anyGamepad.rightStick.angle < 45) {  
    // Look right  
}  
  
if (hinput.gamepad[2].rightStick.inPressedZone) {  
    // Walk  
}
```

hStick also has properties that allow it to convert stick positions directly to a character's movement. Check out the next page to find out how!



Simple character controllers

Let's say you decided to make a game where something (like a player character) is being moved by the stick of a gamepad. To do so, you should probably start by attaching a script to the gameobject you want to move.

In this script, you're going to use one of the features of the **hStick** class that translates a stick position to a world movement. There are two such features :

- *worldPositionFlat* : Translates a stick's position into a flat horizontal movement. Use this if you're making a 3D game or a top-down 2D game.
- *worldPositionCamera* : Translates a stick's position into a movement that faces the game's camera. Use this if you're making a side-scrolling 2D game, or to move the cursor of a RTS game, for instance.

Here is how to use these features (don't use both in the same script !) :

```
public class MyCharacter : MonoBehaviour {
    // This is a "speed" variable that you can change in the inspector
    public float speed = 5;

    void Update () {
        // To make a simple character controller, you can do this
        // (for a top-down 2D game, or a 3D game with horizontal movement)
        transform.position +=
            hinput.gamepad[0].leftStick.worldPositionFlat * speed * Time.deltaTime;

        // OR you can do this
        // (for a side-scrolling 2D game, or to move a cursor facing the camera)
        transform.position +=
            hinput.gamepad[0].leftStick.worldPositionCamera * speed * Time.deltaTime;
    }
}
```

Vibration

hinput integrates the Microsoft library XInput, which allows it to use gamepad vibration. However it will only work on Windows computers, and just for 4 controllers. Here's how to trigger a simple vibration:

```
// Default parameters can be tweaked in settings  
hinput.gamepad[0].Vibrate();
```

Most gamepads contain two vibration motors: a left-side motor, that feels like a low rumble, and a right-side motor, which is more of a sharp, higher-frequency buzz. You can decide the exact intensity of each motor, as well as the duration of the vibration:

```
// Vibrate the left side at 20% intensity and the right side at 100%  
intensity for 0.5 seconds  
hinput.gamepad[0].Vibrate(0.2f, 1, 0.5f);
```

Vibrations can take a lot of time to tweak though, so I prepared some presets you can use to gain some time:

```
// A short and intense vibration, suitable for an impact.  
// Similar to Vibrate(0.2f, 0.8f, 0.2f).  
hinput.gamepad[0].Vibrate(hVibrationPreset.Impact);  
  
// A low and powerful vibration, suitable for an explosion.  
// Similar to Vibrate(0.8f, 0.4f, 0.5f).  
hinput.gamepad[0].Vibrate(hVibrationPreset.Explosion);
```

Finally, here are some more advanced vibration features that you might want to use:

```
// Trigger a vibration based on a curve you can edit in the inspector.  
public AnimationCurve curve;  
hinput.gamepad[0].Vibrate(curve);  
  
// Vibrate a gamepad with an intensity of 30% on the left side, and 10%  
// on the right, FOREVER. Don't forget to call StopVibration!  
hinput.gamepad[0].VibrateAdvanced(0.3f, 0.1f);  
  
// Stop all vibrations on a gamepad immediately.  
hinput.gamepad[0].StopVibration();  
  
// Stop all vibrations on a gamepad progressively over 0.5 seconds.  
hinput.gamepad[0].StopVibration(0.5f);
```

Part 3 : Tips & tricks

Gamepad reference

If your game has a lot of different controls, it will quickly become tedious to write “`hinput.gamepad[0]`” every two lines in your character controller script.

That’s why you might want to create a variable that represents your controller. The simplest way to do so is to create a private **hGamepad** variable, and assign it a value in your Start method like this :

```
public class MyCharacter : MonoBehaviour {
    private hGamepad gamepad;

    // Use this for initialization
    void Start () {
        gamepad = hinput.gamepad[0];
    }

    // Update is called once per frame
    void Update () {
        if (gamepad.A.pressed) {
            // Do something
        }
    }
}
```

Notice how you just need to write “`gamepad`” instead of “`hinput.gamepad[0]`” for the rest of the script.



Implicit casts

A **hPressable** can be implicitly cast to a boolean with the value *pressed*. This means that you don't need to type ".pressed" after the name of a button to know whether it is pressed or not.

In other words, these two lines are exactly equivalent :

```
if (hinput.gamepad[0].A) { //...Some code  
if (hinput.gamepad[0].A.pressed) { //...Some code
```

The same way, a **hStick** can be implicitly cast to a **Vector2** with the value *position*. This means that the following two lines are also equivalent :

```
if (hinput.gamepad[0].leftStick == //...A Vector2  
if (hinput.gamepad[0].leftStick.position == //...A Vector2
```

If you used the previous trick and created a *gamepad* variable to store your gamepad, a piece of code that used to look like that :

```
void Update() {  
    if (hinput.gamepad[0].A.pressed) {  
        // Do something  
    }  
}
```

Now looks like this :

```
void Update() {  
    if (gamepad.A) {  
        // Do something  
    }  
}
```

Much simpler, isn't it ?

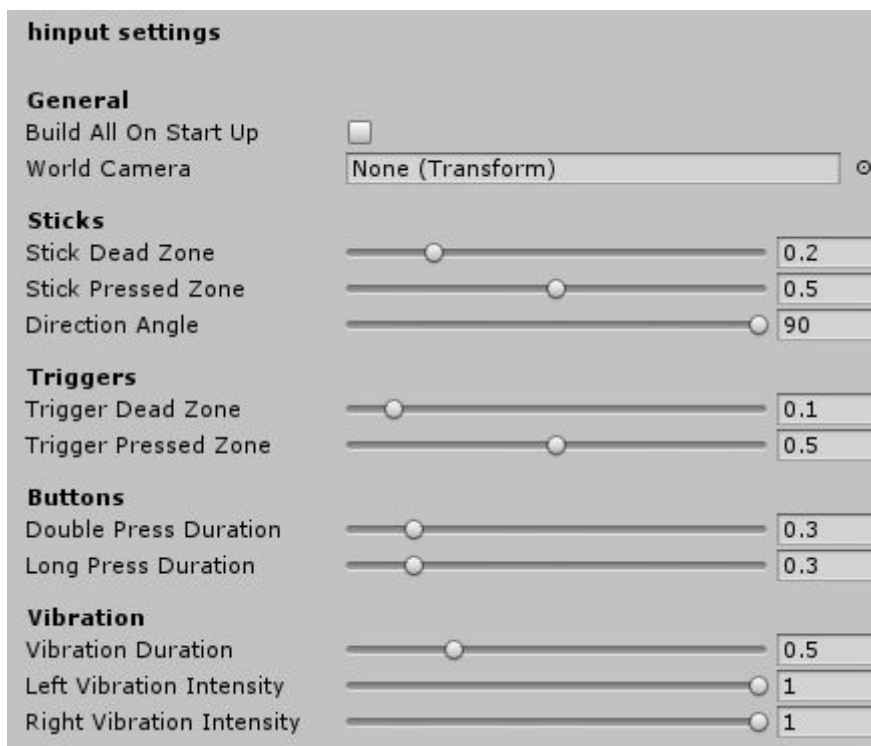
Settings

hinput has many settings that you can use to adapt gamepad controls to your own game.

If you want to use hinput's default settings, you don't have to do anything. At runtime, a gameobject will be created automatically and handle all gamepad inputs.

However, you can also instantiate the hinputSettings prefab manually (you will find it at the root of the hinput folder in your Project tab). It will expose some useful settings that you might want to tweak, such as the duration of a double press, the width of the stick's directions, or the default intensity of vibrations.

If you need help understanding a setting, hover your mouse over it in your Unity project. I wrote helpful tooltips for all of them !



Section	Setting	Value
General	Build All On Start Up	<input type="checkbox"/>
	World Camera	None (Transform)
Sticks	Stick Dead Zone	0.2
	Stick Pressed Zone	0.5
	Direction Angle	90
Triggers	Trigger Dead Zone	0.1
	Trigger Pressed Zone	0.5
Buttons	Double Press Duration	0.3
	Long Press Duration	0.3
Vibration	Vibration Duration	0.5
	Left Vibration Intensity	1
	Right Vibration Intensity	1

That's it for this (not so) short introduction to hinput. You can check out the detailed documentation for the full range of options offered by the plugin.

Feel free to contact me if you have any questions at couvreurhenri@gmail.com. I answer every email !

