Learn

download | **install** | **documentation**

hinput is a simple gamepad manager for Unity - a hilogo project from henri

The basics

Hi, I'm henri. I wrote this short guide to show a few features of hinput.

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

```
hinput.gamepad[0].A.pressed
hinput.gamepad[6].leftTrigger.pressed
hinput.anyGamepad.rightStickClick.pressed
```

As you can see, you start by calling hinput, then you select the gamepad you would like to get, then a button. You finally call the *pressed* method of the 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 :

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

public class MyCharacter : MonoBehaviour {
    // Use this for initialization
    void Start () {
    }

    // Update is called once per frame
    void Update () {
        if (hinput.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 :

```
// Update is called once per frame
void Update () {
    if (hinput.gamepad[0].A.pressed) {
        // Jump
        // This 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 :

```
// Update is called once per frame
void Update () {
    if (hinput.gamepad[0].A.justPressed) {
        // Jump
        // That's much better !
    }
}
```

Stick directions

The term “button” as I’ve used it earlier could refer to any gamepad button, 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, so that they act as virtual buttons. This means 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 :

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

public class MyCharacter : MonoBehaviour {
    // Use this for initialization
    void Start () {
    }

    // Update is called once per frame
    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...

Other button properties

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

```
if (hinput.gamepad[0].A.justPressed) {
    // Jump
}

if (hinput.anyGamepad.dPad.up.justPressed) {
    // Emote
}

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, in the shape of a Vector2. Something like (-0.78, 0.42). Stick positions are always within a circle of 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 !) :

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

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

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        // To make a simple character controller, you can do this
        // (for a 3D game or a top-down 2D game)
        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)
        transform.position +=
            hinput.gamepad[0].leftStick.worldPositionCamera * speed * Time.deltaTime;
    }
}
```

You will notice that this code only makes your character move with the left stick of the gamepad 0, but I'm sure you can guess how to change that.

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. You've been warned !

Here's how to trigger a simple vibration :

```
hinput.gamepad[2].Vibrate(0.5); // For a duration of 0.5 seconds
```

If that's enough for your game, you can skip to the next page. If you need more precise control over vibration, you first need to know a few things :

- Most gamepads contain two different vibration motors : one to the left, one to the right.
- The left motor is a low-frequency rumble motor. Its vibration feels more "deep" (think a jackhammer in the distance).
- The right motor is a high-frequency rumble motor. Its vibration feels more "intense" (think hair clippers).
- The "Vibrate" method of hinput vibrates both sides at the same time, which is what you will find in a lot of games.

Here are the other methods you can use :

```
// Vibrate only the left side, for 0.2 seconds
hinput.gamepad[0].VibrateLeft(0.2);

// Vibrate only the right side, for 0.4 seconds
hinput.gamepad[3].VibrateRight(0.4);

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

// Vibrate the left side 50% and the right side 30%, forever.
// Don't forget to call StopVibration !
hinput.gamepad[2].VibrateAdvanced(0.5, 0.3);

// Stop vibration on this controller
hinput.gamepad[2].StopVibration();
```

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 :

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

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.

This will work just fine in most situations. However, sometimes you might need your gamepad to be initialized right away, even before your Start method (even before your Awake method !).

If that’s the case, you can use this technique instead : (I won’t go into too many details because it’s a bit more complicated)

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

public class MyCharacter : MonoBehaviour {
    private hGamepad _gamepad;
    private hGamepad gamepad {
        get  {
            if (_gamepad == null) {
                _gamepad = hinput.gamepad[0];
            }

            return _gamepad;
        }
    }

    // Use this for initialization
    void Start () {

    }

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

The end result is the same : you can 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*.

It sounds a bit technical, but it just 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 player's gamepad, a piece of code that used to look like that :

```
// Update is called once per frame
void Update() {
    if (hinput.gamepad[0].A.pressed) {
        // Jump
    }
}
```

Now looks like this :

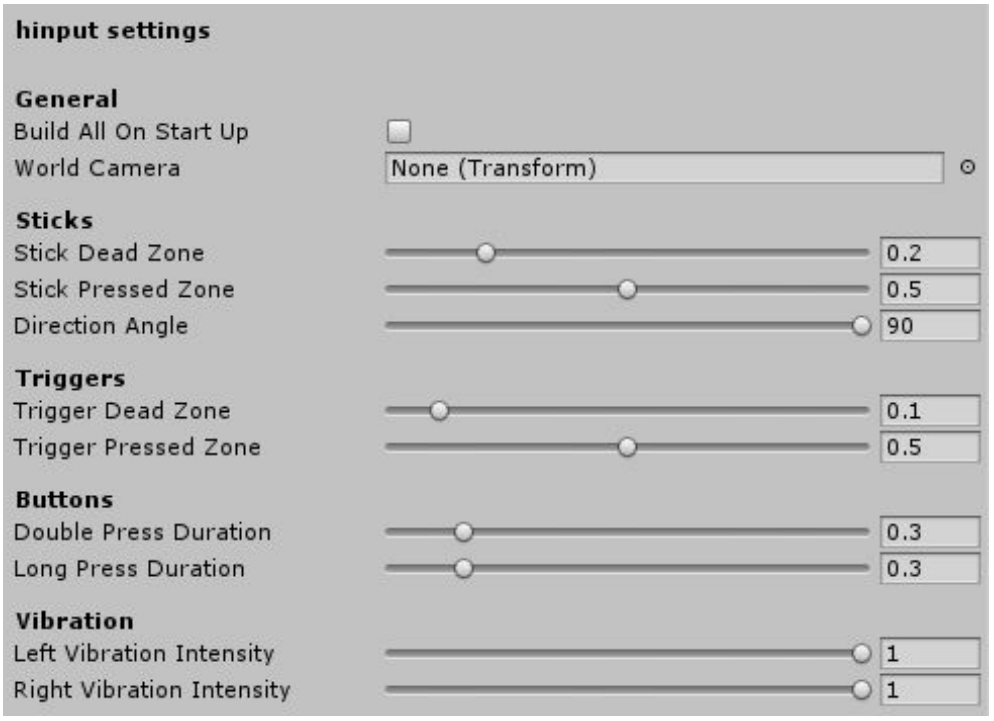
```
// Update is called once per frame
void Update() {
    if (gamepad.A) {
        // Jump
    }
}
```

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 hinput 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.



Feel free to check out the detailed documentation for the full range of options offered by hinput !