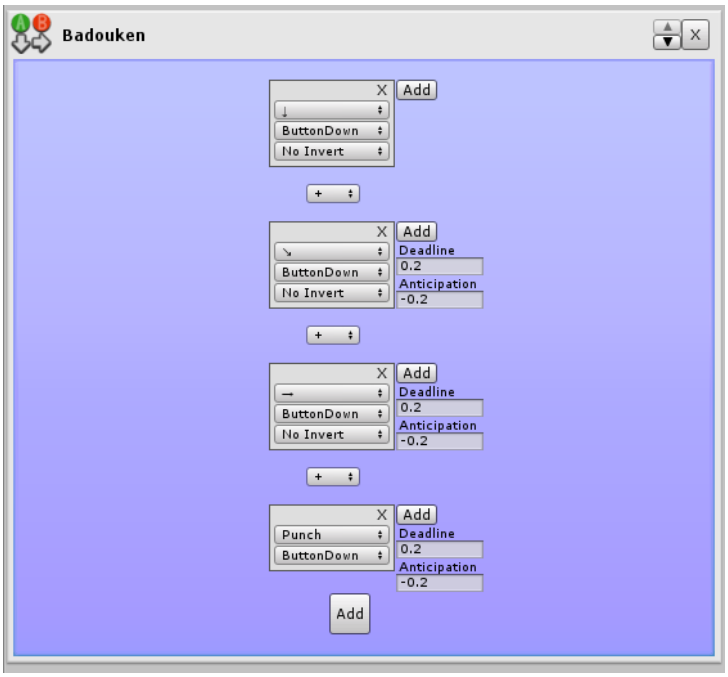
BUTTON SEQUENCE DETECTOR

Infinite Button Sequences

You can do any kind of button sequences. The development was based on the sequence needs of the Street Fighter Series. There is no limit to the number of sequences that can be made. Put the priority sequences at the top of the list.

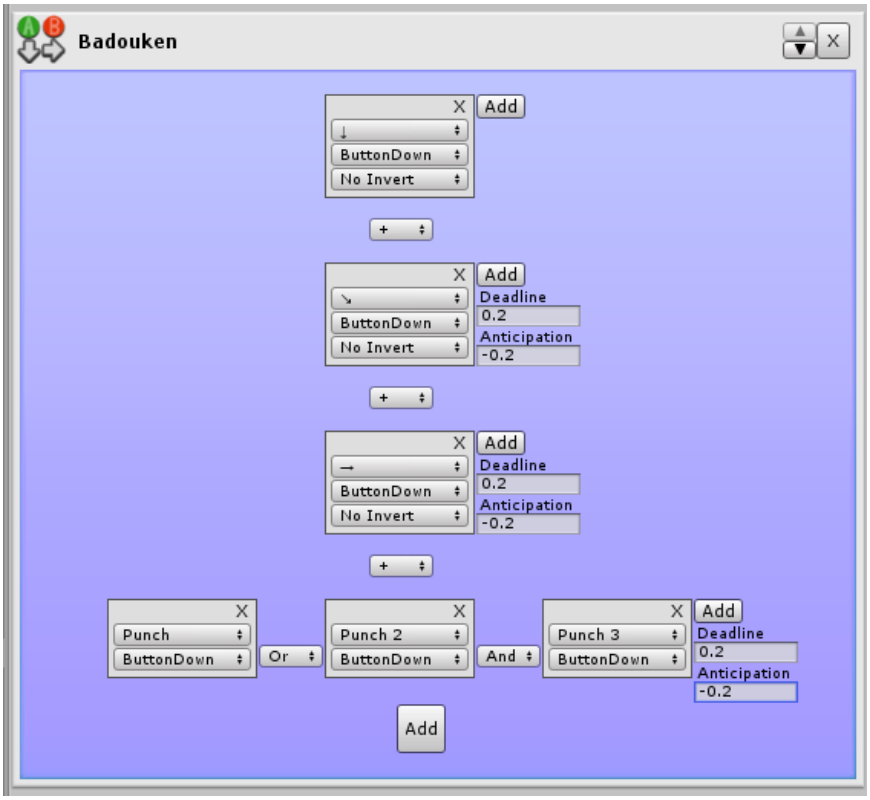
Infinite Button Steps Per Sequence

There is also no limit to the number of button steps. In addition to the subsequent steps you can also place simultaneous or arternative button steps.



Flexible Button Step Editor

It provides easy, fast and intuitive editing.



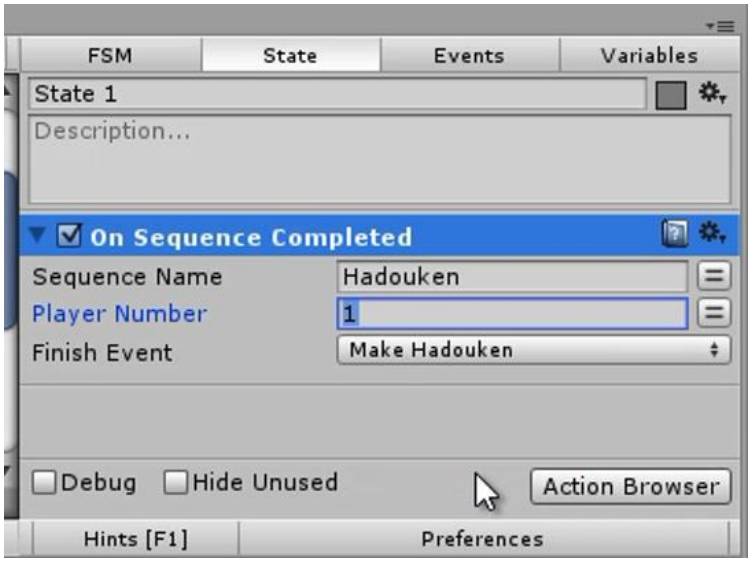
Flexible and Ultra Quicky To Implement

Programmer-friendly, no code implementations required. All you need to do is access CommandSequences.SequenceIsCompleted () to check if the sequence you want has been completed or not. Very simple.

```
void Update () {
    if (CommandSequences.SequenceIsCompleted("Badouken"))
    {
        GetComponent<Animator>().CrossFade("Badouken", 0f);
    }
}
```

Playmaker Support

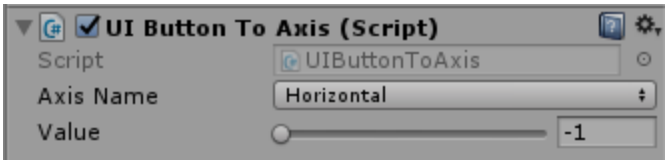
In the playmaker folder there is a package that you can install in case you have the playmaker. In this package there will be an action that will be automatically recognized by the Playmaker. This action is equivalent to the CommandSequences.SequenceIsCompleted() function.



Corgi Support

To support the virtual corgi joystick you must:

A) Add the UIButtonToAxis component for each of the arrows, and specify to which axis each one corresponds, in addition specify the value, whether it is negative or positive. For example, the left arrow is: Axis Horizontal, value -1.

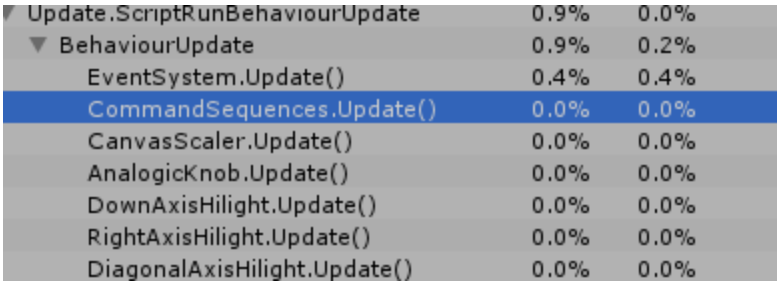


B) Add the UIButtonToButton component to each of the buttons, and specify to which axis each one corresponds.

In any character that BSD is installed, it will recognize these buttons automatically without reference to anything.

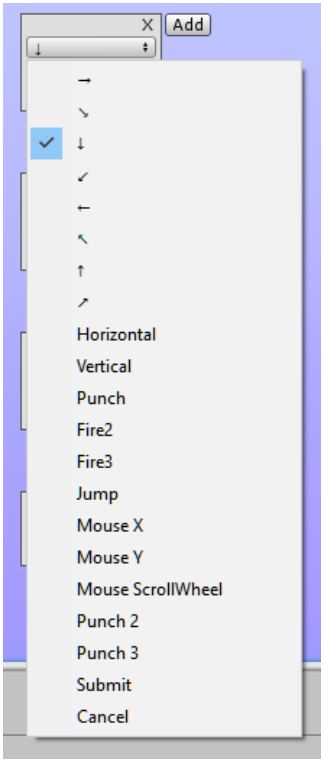
Very Light

Developed with optimization in mind.



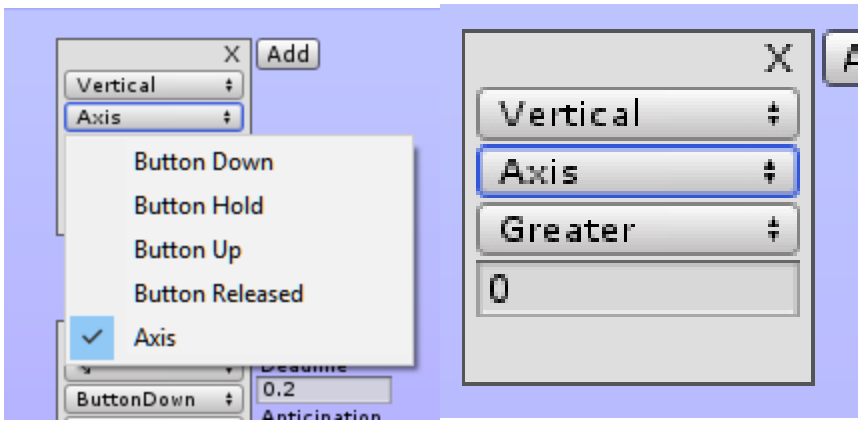
Uses Unity Default Input Axes

It does not use any crazy api to access the button sequences, so it does not cause complication to the programmer.



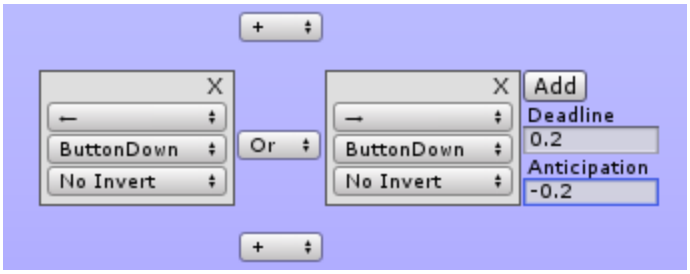
Multiple Input Events

Each button step has a specialized configuration. That checks digital directional, button down, button hold, button Up, button released and comparisons of axis values.



Time Tolerance To Delay and Anticipation

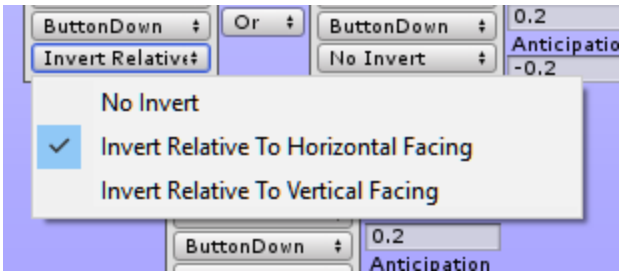
Fighting games are not as tough on their button sequences. A certain tolerance is required to make gameplay more fluid. There Deadline means the time tolerance after the point. There Anticipation means the time tolerance before the point.

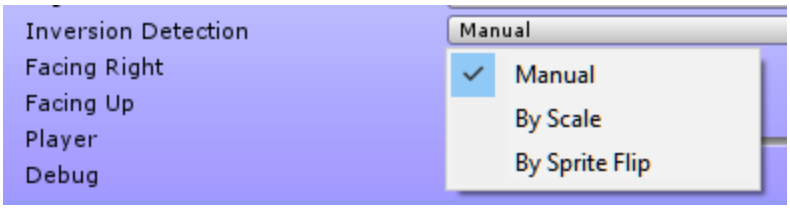


Axis Inversion By Character Facing

It has inversion detection, coordinated by the variable CommandSequences.FacingRight and CommandSequences.FacingUp or by automatic detection.

Inversion is needed to spare configuration for cases where the character's direction interferes in the direction of the axis.





Supports up to 12 players

You will need to configure the alternative axes for the axes of the other 11 players in Menu Edit>Project Settings>Input>Axes using the "p [player number]" suffix, for example "punch p2" for player 2, "punch p3" for the player in. 3 and so on. When you put the options for the second player, the plugin will warn you that this setting needs to be done, if they are not already done.

Easy Way To Access The Particular Player Sequence

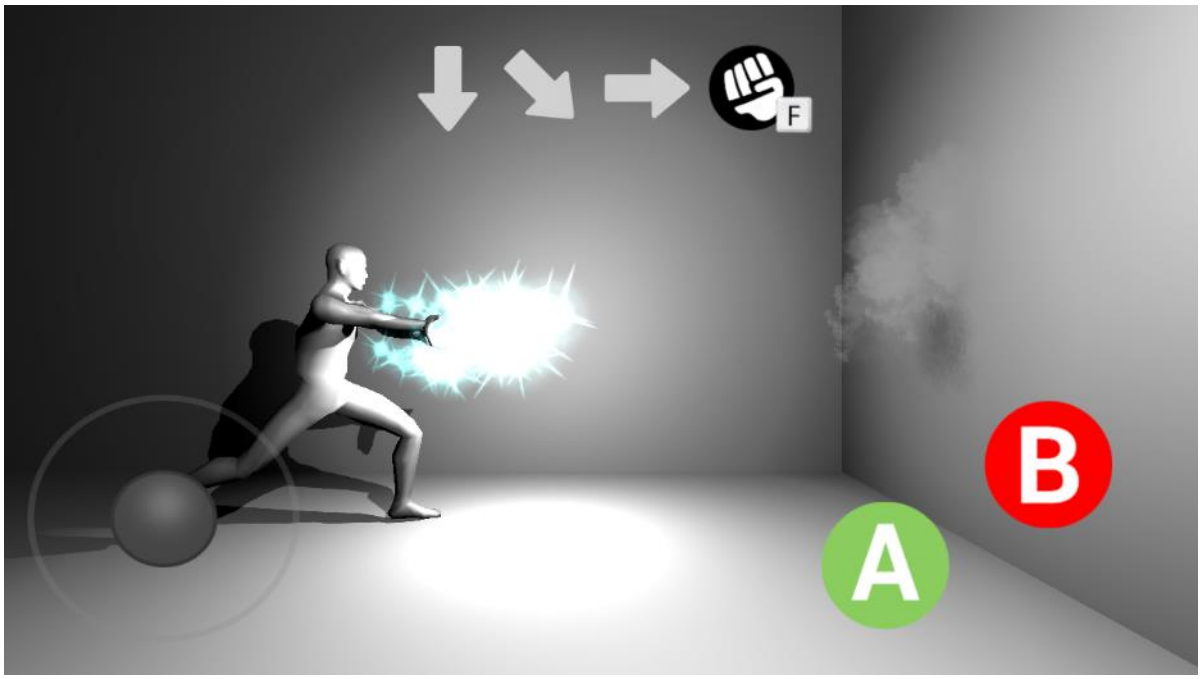
CommandSequences.SequenceIsCompleted("Sequence Name", [player number])

If you do not specify the number of the player, it will be understood as player 1.

```
void Update () {
    if (CommandSequences.SequenceIsCompleted("Badouken", 2))
    {
        GetComponent<Animator>().CrossFade("Badouken", 0f);
    }
```

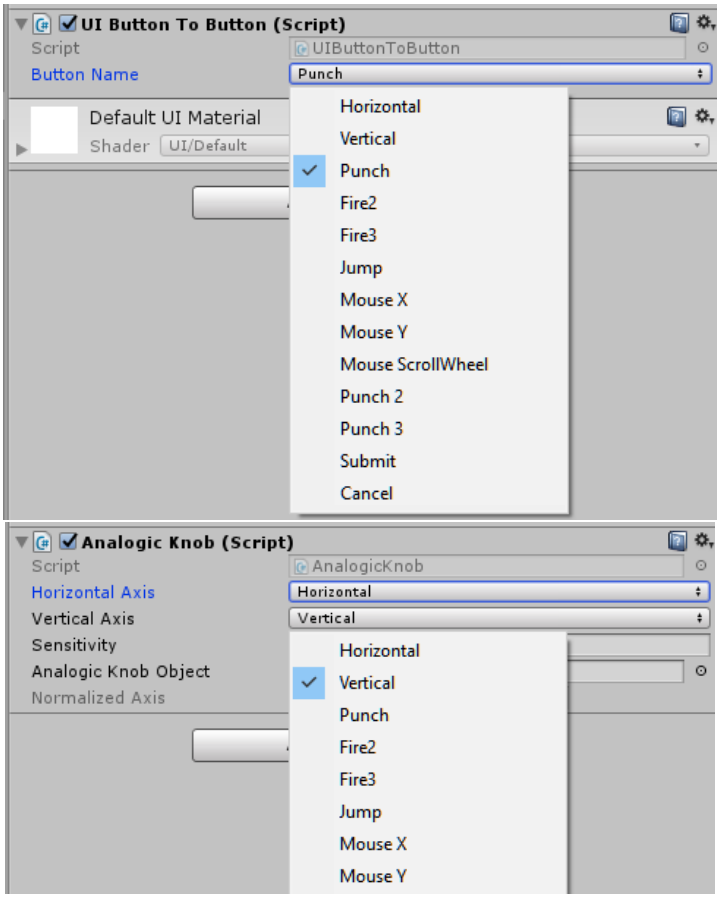
Bonus: Virtual Joystick Support

This bonus is almost a plugin apart, which also works without the BSD.

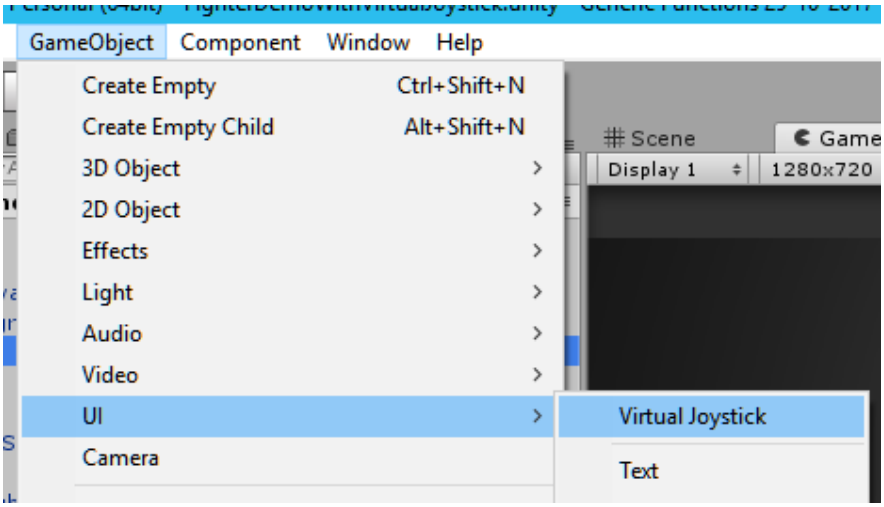


Virtual Joystick Do Not Require Implementation Code

Because it interferes with standard input of Unity. If you already have a game that uses Input.GetAxis () and Input.GetButton () then you can import this joystick and in a magic pass your game will already work on mobile without editing any code. *If you do not want to use this plugin, just delete the "Virtual Plug And Play Joystick" folder and everything will normally occur without errors.*



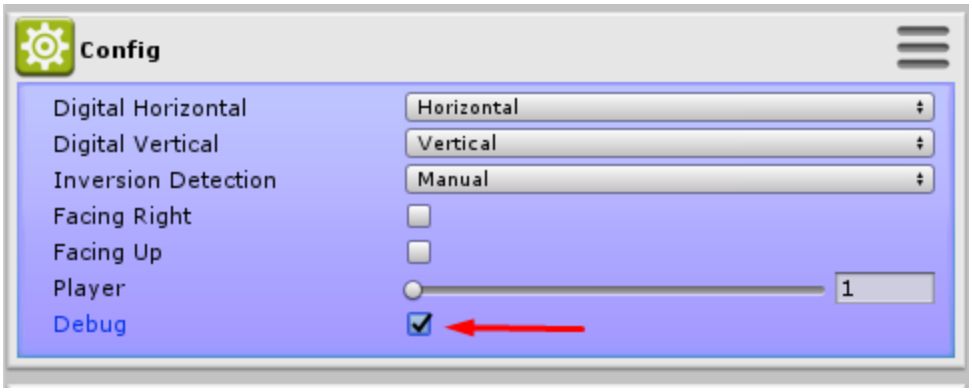
Easy Virtual joystick Implementation

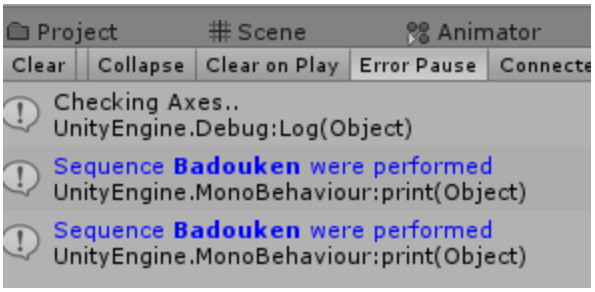


Just add the joystick in the menu, and inside it, for each button specify which axis it corresponds to.

Easily Testable

If you enable debug in the BSD you can see the execution of the configured sequences without accessing the CommandSequences.SequenceIscompleted function.





Quicking start

1 - Add component CommandSequences on your player

2 - Press + button to add button sequence

3 - Edit their name, Let's use “badouken” for this example

4 - Press Add button to add Button Step

5 - Change their axis or button on first dropdown

6 - Change their event on second dropdown

7 - Press Add button again to add new Button Step on sequence and configure it. Repeat the process for how many steps you want to require in this sequence.

8 - Play game to Test button sequence, a log will be displayed, this means your button steps are working

9 - Then go to your character's script and add the if(CommandSequences.SequenceIsCompleted ("Hadouken")) to check if this sequence is completed, now is up to you to choose which animation to play or which reaction you want with the execution of this sequence.

Support: **leoluzprog@gmail.com**