Input problems

Under construction.

ROMs take a long time to load

Some controllers (including Xbox One and 8-bitdo ones) can cause certain cores to take a long time to load a ROM, upwards of thirty seconds. In these situations, it can be worked around by setting the controller to D-input mode (may be referred to as “Android mode” by some manuals). Read more about it in this Retropie forum post.

Bluetooth connection

- Reduce the distance from the controller to the Batocera machine.
- Disable the internal Wi-Fi - this helps to extent signal strength, you can use `rfkill wifi` to disable/enable Wi-Fi.
- Force a lower baud rate in `/etc/init.d/S31emulationstation` or `/etc/init.d/S32bluetooth`. Smaller values of 115200 or 230400 have been known to alleviate input lag on a Pi 3B.
- Buy a new BT adapter and put it to your Raspberry. Add `dtoverlay=pi3-disable-bt` in `/boot/config.txt` to disable internal BT module.

No controller detected even when connected via USB

Few things to try:

- Use a different USB port
- Ensure controller itself is even functional
- If using a third-party controller, switch the input mode on it (refer to your controller's manual)
    - D-input typically works best, followed by X-input, then “Mac” and Android modes
- Disable IOMMU setting in BIOS
- Ensure there's no other obvious settings in the BIOS that could be disabling USB functionality
- Add `hid.ignore_special_drivers=1` to the APPEND line in `/boot/EFI/batocera/syslinux.cfg` (for Batocera **v39** or higher) or `/boot/EFI/BOOT/syslinux.cfg` (for Batocera **v38** or lower) in the boot partition. The line should look similar to: `APPEND label=BATOCERA console=tty3 quiet loglevel=0 vt.global_cursor_default=0 mitigations=off hid.ignore_special_drivers=1` Detailed forum post.
- Clean the contacts with rubbing alcohol (when everything is turned off and disconnected from power), let it dry, and try again
- Scream

If all else fails, please provide a detailed issue report so that the devs can look into it. Here's a real example.

Controller connected and detected, but no inputs

Try configuring the controller by manually editing the `/userdata/system/configs/emulationstation/es_input.cfg` file.

Controller connected, detected and making inputs, but they are unexpected/ghost inputs

Ensure that all of the controller's inputs are in a neutral position when plugging in/connected the controller.

If that still gives unexpected ghost inputs during mapping, try to repair the mapping process by renaming `/userdata/system/.sdl2` to `/userdata/system/.sdl2_bad` and immediately restarting ES (`[Alt] + [F4]` or restart via SSH). If this fixes the issue, it could be because of Batocera's neutral position detection method failing, please report such issues to the Github and the steps to reproduce.

Run `cat /userdata/system/.sdl2/<GUID><name of controller>.cache` to grab the neutral values that have been detected.

When using "Emulate cursor with joystick", the cursor is moving by itself

You can increase the deadzone for pad2key's mouse function by doing the following:

1. Learn how to use Batocera overlays.
2. Download the following file and save it to `/etc/evmapy.json`:

   evmapy.json

   ```
   { "mouse_config": { "every": 0.005, "deadzone": 0.15 } }
   ```

3. Launch a game using the feature and test the cursor. If it's still drifting, increase the “deadzone” value and try again. Higher values decrease the accuracy of pad2key's mouse emulation, so change this conservatively.
4. When a suitable value has been found, run `batocera-save-overlay`.

Diagnose joysticks issues

Physical joystick ←→ Linux driver ←→ Linux events stack ←**A**→ SDL2 library ←**B**→ EmulationStation

From your joystick to Batocera's menu interface (EmulationStation), there are several technical steps. There are two tools to analyze two separate parts of the sequence: `evtest` and `sdl2-jstest`.

`evtest` acts on point **A** in the above diagram, used to debug very low-level issues (like a button's state change not being registered by the controller or an incorrect amount of inputs being registered). `sdl2-jstest` acts on point **B** in the above diagram, used to debug high-level issues (like EmulationStation not reacting correctly when button X is pressed, certain emulator not “seeing” a controller's inputs which otherwise work fine in other programs, etc.).

evtests

`events` in the linux kernel are the way to handle mouses, keyboards, joysticks, etc… sdl2 is a library to help to code program compatible with linux, windows, mac os x, … it converts linux events to a common way to define events. On linux, events are define in /usr/include/linux/input-event-codes.h. For example, the button “yellow” if your joystick has one is :

```
#define KEY_YELLOW      0x190 (1*16*16 + 9*16 + 0 = 400)
```

On Windows, possibly that this button is not defined, or at least, for sure, with an other value, while on Linux, the value is arbitrary. Thus, SDL doesn't use the code. Instead, SDL uses button numbers, thus, this button may be “BUTTON 1”.

Let's see an example :

1. Connect via ssh to batocera
2. run evtest to check that linux sees your joysticks :

In my case, i've only 1 joystick plugged, it displays :

```
# evtest
No device specified, trying to scan all of /dev/input/event*
Available devices:
/dev/input/event0:  Bigben Interactive Bigben Game Pad
Select the device event number [0-0]:
```

Press 0 to see linux events on this joystick. In my case, when i press one button (button BTN_C), it displays :

```
Event: time 1492442283.255053, type 1 (EV_KEY), code 306 (BTN_C), value 1
Event: time 1492442283.255053, type 1 (EV_KEY), code 306 (BTN_C), value 0
```

value 1 is when the button is pressed. value 0 is when the buton is released.

Some general information, like the list of buttons available are provided by :

```
evtest --info /dev/input/event0
udevadm info -q all -n /dev/input/event0
```

sdl2-jstest

Run `sdl2-jstest --list` to check what EmulationStation sees when you press buttons In my case, it displays :

```
# export DISPLAY=:0.0
# sdl2-jstest --list
Found 1 joystick(s)

Joystick Name:     'Bigben Interactive Bigben Game Pad'
Joystick Path:     '/dev/input/event0'
Joystick GUID:     030000006b1400000209000011010000
Joystick Number:    0
Number of Axes:     4
Number of Buttons: 13
Number of Hats:     1
...
Button code  0:   304
Button code  1:   305
Button code  2:   306
...
```

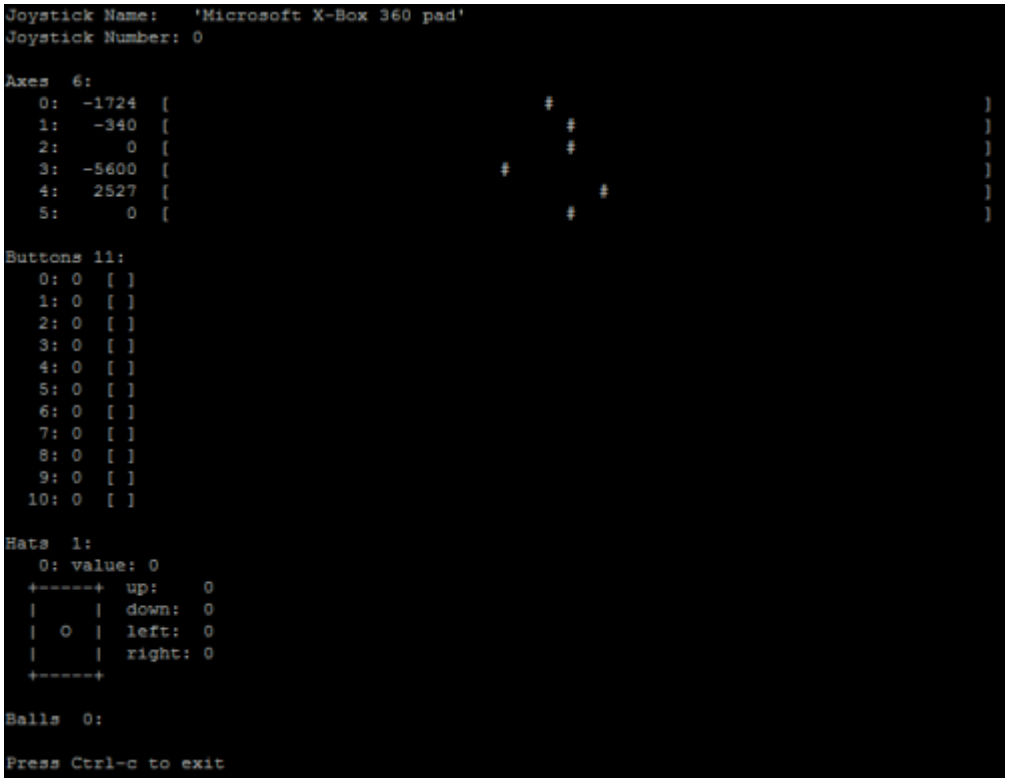
To see the events, run sdl2-jstest -e 0 (0 is the joystick number) When I press C, it displays :

```
...
SDL_JOYBUTTONDOWN: joystick: 0 button: 2 state: 1 code:306
SDL_JOYBUTTONUP: joystick: 0 button: 2 state: 0 code:306
...
```

You can see a graphical representation of your controller by running:

```
sdl2-jstest --test #
```

where # is the number of the joystick you want to test.



Configuring es_input manually

Although Batocera should be able to handle the mapping of nearly any controller, there comes a time where it is only feasibly possible to map a controller manually, such as when the controller is sending multiple signals at once for a single button press or when Batocera thinks the gyroscope is an input for all buttons. The `/userdata/system/configs/emulationstation/es_input.cfg` file can be edited manually.

We need to know the codes the controller's inputs are sending. Open up two SSH sessions to Batocera. Yes, you can do that, just open the program twice. On one session:

1. Type `sdl2-jstest -l` and press `[Enter]`. Take note of the number of the joystick you intend to remap.
2. Type `sdl2-jstest --test #` and press `[Enter]`, where # is the joystick number.

On the other session:

1. Type `evtest` and press `[Enter]`.
2. Type in the number of the input device you intend to remap (this may be different from the one in the other session).

The controller's inputs will now output values on both SSH sessions. You now have all the input information you need to make a manual remap.

es_input syntax

The syntax for each input line mapped:

```
        <input name="<name>" type="<type>" id="<#>" value="<#>" code="<#>"
/>
```

The two leading spaces are tabs, not spaces. This is important.

Explanation of the attributes:

- **name** is the virtual input on the Batocera Retropad, the input that is ultimately sent to the emulator. This is most similar to a SNES controller, however `pageup` and `pagedown` represent `[L1]` and `[R1]` respectively. Refer to the already provided configs in `es_input.cfg` for examples.
- **type** is the type of input being scanned for on the controller. This can be either `button` or `axis`. It does not have to match up with the kind of input specified in **name**.
- **id** is the number affected in the `sdl2-jstest` session when the button/axis is manipulated. This behaves differently for hats (usually D-pads), as they determine their direction pressed from value alone (when there is only one hat, it is always 0).
- **value** is the value that the control is “pressed” on in the `evtest` session. Typically boolean 0 or 1 for buttons, decimal -1 to 1 for axis.
- **code** is the code that the control is “pressed” on in the `evtest` session. This should be unique to every input.

So for instance a button input line might look like this:

```
        <input name="a" type="button" id="0" value="1" code="304" />
```

The “a” is the input that is sent to the emulator, typically bound to  on the controller itself. “button” is its type. An id of “0” is the class it belongs to; typically all the face buttons on a controller will belong to the same id class (so “b” would also have an id of “0”). The value of “1” is achieved when the button is depressed, so this is what the emulator is looking for when reading inputs, all other values will fail to register this as the input (analog controls work differently, example below). And the code “304” is the unique part that associates the configured input with that physically button; this should always be a unique value per input on the controller.

An axis might look like this:

```
        <input name="joystick1left" type="axis" id="0" value="-1" code="0"
/>
```

The “joystick1left” input represents the first analog stick (“1”, usually the left stick) being pushed to the left (usually a negative axis value). It is of the “axis” type input. Belongs to the id class of “0”. The value of “-1” indicates the direction the axis must be pushed toward to represent the input name (in this example, pushing left on the left analog stick). And the code “0” is unique to the stick.

Batocera's configgen allows for “logical deduction” in regard to the axis inputs; if “joystick1left” is achieved at axis value “-1”, then it assumes that “joystick1right” must be the opposite at axis value “1” without explicitly needing to be told. So theoretically, only two lines are needed for a typical analog stick:

```
        <input name="joystick1left" type="axis" id="0" value="-1" code="0"
/>
```

```
        <input name="joystick1up" type="axis" id="0" value="-1" code="1" />
```

And this would cover all directions the stick could be pushed in.

There is no logical deduction logic for any buttons, they must all be manually specified.