Architecture Improvement

# Proposal

### Migration from an old joystick API

The current architecture of WeJoy uses the "joystick" API. As a result, it has classes for manually wrapping the joystick API, and requires the use of the "udev" API for querying information about the connected joystick, such as a vendor or product id.

A better solution would be to use the "event" API with a wrapper like "libevdev". This allows us to leave the implementation of the event API up to libevdev, and we only need to ask for events using data structures it has already defined for us. This also allows us to remove the use of udev, as the event API exposes this information for us.

Another benefit is this removes the requirement for controllers to be connected via usb, and opens up both Bluetooth and even virtual devices.

### Testing framework

The current architecture of WeJoy has 0 tests. I would like to write a testing library that creates a virtual joystick, and then runs WeJoy with an example script configured for the joysticks. I then feed data into the virtual joystick and use libevdev to check that the virtual joysticks and keyboard created by WeJoy are outputting the correct data. I would like to implement this using Catch2 and BDD, as the Given When Then style of defining tests fits this domain very well, as we will have an input, and expect an output.

### New build system

WeJoy was using a shell script to build. This is not very flexible, so I have will replace it with a CMake script instead. This means I can now generate code from system files, which gives us the ability to remove a lot of hardcoded constants from the source code. It also gives us the ability to forward constants to Lua, so that they can be used by the user if required.

# Alterations

### Migrate from the joystick device API to the event device API

WeJoy was using on the older joystick API for querying device information. This API was not comprehensive enough, and as a result some code based on the udev API was added to query for information such as the product and vendor ids. I have migrated this code base to the event API, and as a result, the information can now come directly from the event API, and we no longer have a limitation of USB devices. This opens up the ability to pass virtual devices in for testing, or use Bluetooth devices.

### Removal of the event structure

WeJoy was previously using its own event structure in order to manually read data from the old joystick API. This is no longer needed, as the event API provides this type of data for us. As a result, the code for handling events is much simpler as it has access to the required data when called. This also allows us to no longer need to update data structures related to external APIs, which is useful to avoid features breaking if the APIs we are dependent on change.

### Added functions for receiving many events at once

While testing, I needed a way to identify what joystick button was pressed. To do this, I added some methods to the Lua script that would let you write general handlers for all axis or all button events. I have added a method to Lua called axis\_event that is called when any axis change occurs, and this event is passed in a device, the changed axis and the value the axis was changed to. This allows the user to define a handler for multiple axes to clean up their scripts. The same was done for buttons, but with a function called button\_event.

### Added an importer for the events from the Linux kernel instead of hardcoding basic events

WeJoy had a hardcoded list of events that it could listen to. I thought this was a bad idea as it means that it will need to be changed if a device is ever added to the Linux kernel.

Instead, I found and modified a script to allow me to generate a list of events, and have changed the code to reference this list.

### Added functions to get the max and min values of a specific axis

While testing, I found it difficult to map from a real device to a virtual one as they both used different max and min values. This meant that one joystick may have been at the max value, and the other joystick may not even have registered a change. I added the ability to retrieve this information so a user could write a simple script in Lua to map from one coordinate system to another. I also added constants for the coordinate system used the virtual joystick.

# Evaluation

The main problem that this improvement solves is extensibility. In order to solve this, I have migrated the code base from the old joystick API to the new event API. This allows us to handle more devices than USB, and also allows us to use newer devices that do not support the old joystick USB.

### Extensibility

Extensibility of software is a recognised aspect of quality in a product, as defined by ISO/IEC 25010. This change has made the software more compatible with other devices, allowing it to be used for more purposes. There is also a lot less hard-coded values, which means the product will stay up to date if Linux decides to add more devices.

The secondary problem that this improvement solves is lack of tests. I have implemented a full testing framework based on Catch2, which allows a developer to easily write tests using a BDD (Behaviour Driven Design) syntax. This allows someone to write a demonstration script, and then using a virtual controller, test that script and as a result test the program as long as the script is comprehensive enough.

# Stakeholder Concerns

The two main stakeholders of WeJoy are the developers and the users that are writing scripts targeting it. The testing architectural improvement primarily targets the developers, while the extensibility improvements target both the user and the developer.

The developers of a software project have an interest in making sure a product stays up to date and relevant at all times. If a product is designed with outdated components, it will eventually be replaced with a newer product if that product supports a broader range of devices. In this case, updating the API will allow the product to stay relevant for longer as it can now be used to solve problems that other products cannot.

Developers of a product will find that a product with a proper testing framework is a much more enjoyable to work with. With this testing framework in place, the developer can now work on developing features, and will not need to worry about their features causing bugs or breaking other features. Using BDD, we can make sure that tests are both easy to write, and can test all parts of the application in unison. For this reason, developers as stakeholders will find the architectural changes I have made useful.

Developers of this product will also find that a proper build system is a lot more pleasant than a single script that runs a command, as it gives the ability to run other scripts and tests from the build scripts, instead of being limited to a single command.

Users of this product will find that the project is much more pleasant to work with. While testing, I found that I needed a way to get information about a joystick from the Lua side, and so I have implemented features for getting calibration constants for a joystick. I have also added features that allow targeting multiple axes and buttons at once, to allow the user to work out key bindings, or skip repetitive scripts.

Users of this product will also find that the project is much more compatible as a result of the changes, and it in some cases no longer requires root access. It also now works with a wider range of devices, and for a project in this domain that is an important feature for a stakeholder.