DF Robot Input Shield

Now that you have the ball running across the screen, the next step is to add animation to the pong paddles. To do this, the same redrawing technique will be used, this time utilizing the input shield. However, implementation is not as straight forward as just plugging the device in and having it work off the bat.

When using multiple shields, it is important to keep track of how each device interfaces with the Arduino board. Unfortunately, sometimes multiple peripherals and shields will end up using the same interface and pins, causing the data sent from those devices to continually interfere with one another giving you erroneous results. An interface is simply a protocol used to send data and communicate between devices, and different interfaces utilize different GPIO pins. In our case, the Adafruit LCD screen uses the SPI interface (http://arduino.cc/en/Reference/SPI#.Uy8KGvldXbk_) as does the DF Robot Input Shield. While this provides some dilemma, we can get around it using a few different approaches. This project will utilize the serial interface which sends bytes of data via the TX/RX pins (usually pins 0 and 1 on the UNO) to get around this problem. This will involve using two Arduino boards communicating with each other. By using two boards, one will be dedicated to handling the Shield Input, while the other will be dedicated to the display and game code.

The first step is to connect your second Arduino board to the DF Robot input shield. The next step is to determine the values for the analog stick for which the stick is in the up, down, or middle position. Using the following pin assignment chart:

Pin	Function
Digital 8	UP
Analog 1	Down
Digital 9	Left
Digital 12	Right
Analog 3	X axis
Analog 2	Y axis

- Create a simple sketch that will read in the values when it is moved and print those values to the Serial monitor. To do this:
    - Define the pins used and assign them variable names
    - Define the pins as input or output in the setup method using the pinMode() method
    - Use the analogRead() method to read the values or assign them to a variable
    - Print those values to the serial monitor

- Use the serial monitor to determine the read in values of the analog stick for the up, down, and centered positions.

The next step is to begin serial communication. You will need two separate sketches, separate from your pong game. Using the provided sketches to help you get started, serially send the analog signals to your second Arduino using the Serial.write() method. On the receiving end, you will read in the bytes of data using the Serial.read() method. Take a look at the serial interface reference page provided by Arduino to give you more insight on the mechanics and other functions you may want to use: http://arduino.cc/en/reference/serial#.Uy8PdfldXbk. **To serially connect the two Arduino boards, connect the TX pin of the Arduino connected to the Input Shield to the RX pin, the TX pin of the other Arduino to the RX pin of the Input Shield dedicated Arduino, and then connect both Arduinos' grounds to each other.**

Some other things to keep in mind when writing your sending/receiving sketches:

- You want to read the data in a synchronized fashion; you don't want to read in the down analog value if you're expecting to read in the up value first. The serial interface itself is asynchronous. Remember, the data is read in the order that it is sent out, but if you're sending board sends data faster than the receiving board, errors will arise.

- Based on the nature of the serial interface, you need to implement a delay on the sending side so the receiving Arduino has time to accurately read in every subsequent byte.

- A delay may or may not be needed in the receiver's code depending on how you code it.

- The Serial.print() function also sends ASCII formatted data to the serial port, which could give you errors on the receiving side; comment out and Serial.prints() you may have.

- Serial.write() sends a byte at a time (0-255), but the analog values can exceed a byte (0 -1023). Use the Map() function to overcome this: http://arduino.cc/en/reference/map#.Uy8UcfldXbk

With all of this is mind, your tasks are to now:

- Create two sketches to send and receive analog input

- With both boards powered on, verify that you are reading correct values on the receiving side by printing them to the serial monitor as you move your analog stick around.

- Take your code from the receiving side and implement is as a single separate function in your actual PONG game. Then repeatedly call this function in the void loop.

- Verify that you get the same results again when you move around the analog stick when you are running the receiving code from within your game.

Some more things to keep in mind:

- As you develop your pong game, depending on how efficiently you coded your sending/receiving code, you may have to periodically test that you still receive the appropriate input from the shield and that there is no noise/invalid data being read.

- Delay may accrue as your code lengthens and there are more things to redraw and more situations that need to be handled. This delay may also affect your receiving side code depending on how you wrote it.

- If you have delay in your receiving code, because it will be called as a function in the game, that means the average game speed will slow down. Try to code the sending/receiving code as efficient as possible to avoid slow down.

Now with your serial communication fully implemented, and your Input Shield working, you can proceed with creating the player paddle movement. Because you mapped the input values, take the floor of your values divided by 4. Those will be your new base line values for reading in analog input.

Your final tasks for this session are as follows:

- Once you know the values for all three stick positions, make a variable, like you did for the ball, called something such as 'pxdir' (paddle X direction). Also make a variable that will stand for the X coordinate of the paddle ('X' will do nicely). These variables will be used to update the status of the paddle's position in order that a new paddle can be drawn.

- Depending on which way you have your interface set up (which direction with relation to the pixels of the screen is up and down) will determine how the joystick values affect the 'pxdir' value. The way we have our Pong game set up is that when the joystick is up, 'pxdir' is set to 1, when the joystick is down, 'pxdir' is set to -1, and when the joystick is in the middle, 'pxdir' is set to 0.

    - By adding 'pxdir' to X and updating the X variable:
        - X = 'pxdir' + X

    - You can update the paddle's position just by using the variable X as the x pixel in fillRect(x, y, height, width, color) instruction.