LED Brightness Shifting

Alright, buckle up, because this project is a fantastic dive into the world of interactive art and user interfaces with Arduino! We're taking a simple joystick and transforming its Y-axis movement into a dynamic light show across six LEDs, and the coolest part? We're flipping the script on that light show with a single button press! Let's break down how this magic happens.

The Core Idea: Joystick Input to LED Brightness Mapping

At its heart, this project maps the analog input from the joystick's Y-axis to the brightness of six individual LEDs. Instead of a simple linear relationship, we're using mathematical functions – specifically squared functions – to create a more visually interesting and nuanced response. Think of it like sculpting light with the joystick.

The Components:

- **Joystick (with X, Y, and Switch):** Our primary input device. The Y-axis provides an analog voltage that the Arduino reads as a value between 0 and 1023. The built-in switch gives us a digital input for changing the "program mode."
- **Six LEDs:** Our visual output. We're controlling their brightness using Pulse Width Modulation (PWM) via the analogWrite() function on digital pins 3, 5, 6, 9, 10, and 11.
- **Arduino:** The brains of the operation, reading the joystick input, performing the calculations, and controlling the LEDs.
- **Resistors (Not Explicitly in Code but Crucial!):** You'll need current-limiting resistors in series with each LED to prevent them from burning out. The value will depend on the type of LEDs you're using.

The Code Breakdown (Let's Get Excited!):

1. **Pin Definitions:**

```
int const x_axis = A0;

int const Y_axis = A1;

int const joyStick_sw = 2;


int LED1 = 3;

int LED2 = 5;

int LED3 = 6;

int LED4 = 9;

int LED5 = 10;

int LED6 = 11;
```

We're defining the analog input pins for the joystick's X and Y axes and the digital input pin for the joystick's switch. We're also defining the digital output pins connected to our six LEDs.

2. **Variables:**

```
int LED1Value;
int LED2Value;
int LED3Value;
int LED4Value;
int LED5Value;
int LED6Value;

int read_X;
int read_Y;
int read_SW;

int buttonState;
int memory = HIGH;
int programMode = 1;

int wait (10);
```

These variables will store the brightness values for each LED, the raw readings from the joystick, the state of the joystick switch, a memory variable for debouncing the switch, the current programMode (our light show style!), and a small wait delay for debouncing.

3. **setup():**

```
void setup() {
  Serial.begin(19200); // Initialize serial communication for debugging

  pinMode(x_axis, INPUT);
  pinMode(Y_axis, INPUT);
  pinMode(joyStick_sw, INPUT_PULLUP); // Enable internal pull-up for the switch

  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  pinMode(LED3, OUTPUT);
  pinMode(LED4, OUTPUT);
  pinMode(LED5, OUTPUT);
  pinMode(LED6, OUTPUT);
}
```

Here, we initialize serial communication so we can see the LED1Value for debugging. We set the joystick pins as inputs (with a PULLUP resistor for the switch, meaning it will read HIGH when not pressed and LOW when pressed). We also set all the LED pins as outputs, ready to shine!

4. **loop() - The Heart of the Action!**

- **Reading Joystick Input:**

```
read_X = analogRead(x_axis);
read_Y = analogRead(Y_axis);
read_SW = digitalRead(joyStick_sw);
buttonState = digitalRead(joyStick_sw);
```

In each loop, we read the analog values from the joystick's X and Y axes (though the X-axis isn't currently used in the LED calculations) and the digital state of the joystick's switch.

- **Program Mode Switching (The Cool Part!):**

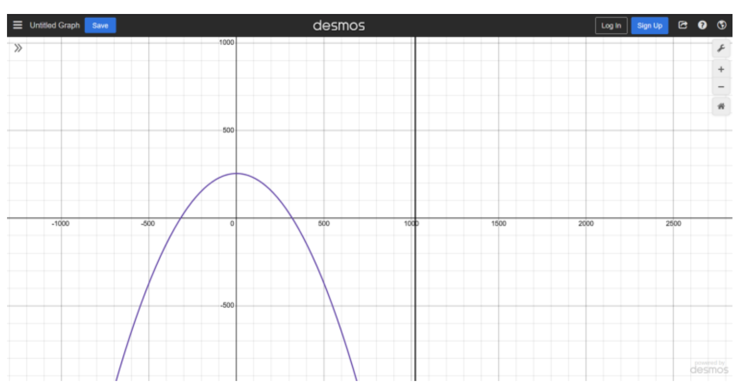
```
if (buttonState == HIGH && memory == HIGH){
 if (programMode == 2){
  programMode = 1;
 }else {
  programMode = 2;
 }
 memory = LOW;
 delay(wait);
} else if (buttonState == LOW && memory == LOW) {
 memory = HIGH;
}
```

This section is responsible for toggling between our two "program modes" (light show styles) when the joystick button is pressed and released.

- We check if the buttonState is HIGH (not pressed, due to the pull-up) AND our memory variable is HIGH. This ensures we only trigger a mode change once per press.
- If the conditions are met, we flip the programMode between 1 and 2.
- We set memory to LOW to indicate we've registered a press.
- The delay(wait) introduces a small pause to help with debouncing the physical button, preventing rapid toggling from a single press.
- The else if condition sets memory back to HIGH when the button is released, ready for the next press.

- **Light Show Logic (Program Mode 1):**

```
if (programMode == 1) {
 LED1Value = -sq(((read_Y/20.)-10.23*(0.)))+255;
 LED2Value = -sq(((read_Y/20.)-10.23*(1.)))+255;
 LED3Value = -sq(((read_Y/20.)-10.23*(2.)))+255;
 LED4Value = -sq(((read_Y/20.)-10.23*(3.)))+255;
 LED5Value = -sq(((read_Y/20.)-10.23*(4.)))+255;
 LED6Value = -sq(((read_Y/20.)-10.23*(5.)))+255;
}
```



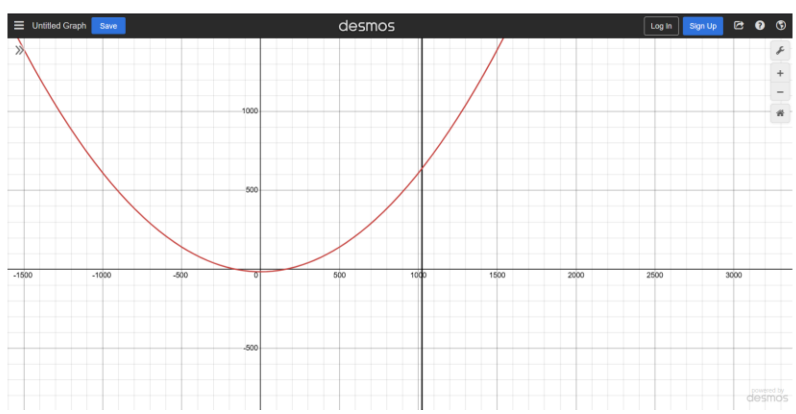
we can simply imagine that by shifting the Joystick we are shifting this function in the image above to the right and left, where the x-axis is the joystick input and the y-axis are the brightness

In programMode 1, the brightness of each LED is calculated based on the Y-axis reading (read_Y) using a squared function.

- (read_Y/20.) scales the 0-1023 Y-axis reading to a smaller range.
- -10.23*(0.), -10.23*(1.), etc., introduce offsets for each LED. This shifts the peak of the squared function along the Y-axis input range, causing each LED to brighten and dim at different joystick positions.
- sq(...) squares the result, creating a parabolic curve for the brightness response (as visualized in your Desmos graphs!).
- The negative sign inverts the parabola, and +255 shifts the entire curve upwards so the brightness values are within the 0-255 range for analogWrite(). This mode likely makes the LEDs brighter in the center of the Y-axis movement and dimmer towards the top and bottom.

- **Light Show Logic (Program Mode 2 - The Reverse!):**

```
if (programMode == 2){
 LED1Value = sq(((read_Y/40.)-5.11*(0.)))-15;
 LED2Value = sq(((read_Y/40.)-5.11*(1.)))-15;
 LED3Value = sq(((read_Y/40.)-5.11*(2.)))-15;
 LED4Value = sq(((read_Y/40.)-5.11*(3.)))-15;
 LED5Value = sq(((read_Y/40.)-5.11*(4.)))-15;
 LED6Value = sq(((read_Y/40.)-5.11*(5.)))-15;
}
```



we can simply imagine that by shifting the Joystick we are shifting this function in the image above to the right and left, where the x-axis is the joystick input and the y-axis are the brightness.

In programMode 2, the calculations are similar, but with different scaling factors (/40.), offsets (-5.11*(...)), and a different final adjustment (-15). This set of equations is designed to create the *reversed* effect you described. Instead of brightness peaking in the middle, it likely creates "darkness" (lower brightness) in the middle and brighter LEDs towards the extremes of the Y-axis movement. The Desmos graphs for this mode would show parabolas that are not inverted and potentially shifted downwards.

- **Clamping Brightness Values:**

```
if (LED1Value<1){
  analogWrite(LED1, 0);
}else if (LED1Value>254){
  analogWrite(LED1, 255);
}else {
  analogWrite(LED1, LED1Value);
}
// ... (similar blocks for LED2 to LED6)
```

  These if-else if-else blocks ensure that the calculated LEDValue for each LED stays within the valid range for analogWrite() (0-255). If the calculated value goes below 1, it's set to 0 (fully off). If it goes above 254, it's set to 255 (fully on). Otherwise, the calculated value is used.

- **Writing to LEDs and Serial Output:**

```
analogWrite(LED1, LED1Value);

analogWrite(LED2, LED2Value);

analogWrite(LED3, LED3Value);

analogWrite(LED4, LED4Value);

analogWrite(LED5, LED5Value);

analogWrite(LED6, LED6Value);


Serial.println(LED1Value); // Print for debugging
```

- Finally, we use analogWrite() to set the brightness of each LED according to its calculated LEDValue. We also print the LED1Value to the serial monitor, which is helpful for debugging and understanding the numerical output of our equations.

The "Reverse the Brightness" Magic:

The key to reversing the brightness effect lies in the different mathematical equations used in programMode 1 and programMode 2. By pressing the joystick button, we're essentially switching between these two sets of equations. One set creates a "bright in the middle" effect, while the other creates a "dark in the middle" effect (or vice-versa, depending on the exact shape of your Desmos curves).

From Arduino to IoT Potential!

This project, while seemingly simple, lays a fantastic foundation for exploring IoT concepts:

- **Sensors as Input:** The joystick acts as a sensor, providing real-time analog data. In IoT, you might use temperature sensors, light sensors, accelerometers, etc., to gather data about the environment or user interactions.
- **Actuators as Output:** The LEDs are our actuators, responding to the processed sensor data. In IoT, actuators could be motors, relays, displays, or even cloud-based commands.
- **User Interface:** The joystick and button provide a simple yet effective user interface. IoT devices often need ways for users to interact with them, whether through physical buttons, touchscreens, or even voice commands.
- **State Management (programMode, memory):** We're managing the state of our light show using variables. IoT devices often have complex states that need to be tracked and managed.
- **Real-time Control:** The LEDs respond immediately to the joystick movement and button presses, demonstrating real-time control, a crucial aspect of many IoT applications.
- **Data Visualization (Simple):** The changing brightness of the LEDs provides a basic form of data visualization, representing the joystick's position. IoT often involves visualizing complex data in understandable ways.

Imagine taking this further:

- **Controlling Smart Home Devices:** Instead of LEDs, the joystick could control the brightness of smart bulbs or the speed of a smart fan. The button could toggle devices on/off.

- **Robotics Control:** The joystick could provide analog control signals to the motors of a robot. The button could trigger different robot actions.
- **Interactive Art Installations:** This project's core concept could be expanded into larger interactive art pieces where user input dynamically manipulates light, sound, or movement.
- **Data Sonification:** Instead of brightness, the joystick position could control the pitch or volume of sound.

This project is a stepping stone, showing how physical inputs can be translated into digital outputs based on defined rules and how even a simple button can dramatically change the behavior of a system. It's all about sensing, processing, and actuating – fundamental building blocks of the exciting world of IoT! Let's get this documented on your GitHub and inspire others!